Instituto de Informática

TRABALHO DE ARQUITETURA DE COMPUTADORES II:

Superescalaridade e Taxonomia de Flynn

Alberto Borsatto

Anderson Anan

Henrique Felizzola

Porto Alegre

2022

Pipeline superscalar é classificado como SISD, SIMD ou MIMD?

No processo de pipeline, as unidades operacionais serão encarregadas de realizar partes das instruções a serem executadas durante os ciclos. Por exemplo, quando a unidade operacional de busca de uma instrução houver realizado a operação de um dos ciclos, já poderá inicializar a busca da instrução para o próximo ciclo, assim permitindo menos ociosidade entre as unidades operacionais e otimizando o trabalho do processador.

O pipeline superescalar simboliza um passo a mais se comparado ao pipeline tradicional, pois utiliza do paralelismo espacial ao replicar as unidades funcionais, o que representa a utilização de mais de um pipeline. Essa replicação permite uma vazão ainda maior de instruções e também a capacidade de finalizar mais de uma instrução no mesmo ciclo. Porém, mesmo que a vazão de instruções seja aumentada e, com isso, a otimização do processador seja ainda mais incrementada; durante a execução, apenas um endereço de memória é buscado, o que acarreta em apenas um fluxo de instrução, ou seja, apenas um processo é executado por vez.

Na taxonomia de Flynn, um processador com apenas um núcleo poderá ser classificado como SISD ou SIMD. Entretanto, a característica que irá diferenciar as duas classificações será se dentro das instruções suportadas pelo processador estão presentes instruções vetoriais ou não, ou seja, instruções que realizam operações com mais de um dado da memória. Caso haja instruções vetoriais, a classificação do processador será SIMD, caso contrário, será SISD. Em um exemplo diferente, um processador com mais de um núcleo será classificado como MIMD. Nessa situação, os seus diferentes núcleos e suas múltiplas threads permitirão que existam fluxos de instrução paralelos, ou seja, mais de um processo ocorrendo ao mesmo tempo.

Desse modo, o conceito de superescalaridade não deve ser confundido com a classificação MIMD, pois a superescalaridade propriamente dita não implementa a execução de fluxos de instrução independentes, isso só é possível em arquiteturas que realizam computação paralela, o que é definido por outras características do processador, como por exemplo multithreading.

Portanto, é possível concluir que o fato de um processador ser superescalar não permite uma conclusão definitiva sobre a sua classificação na taxonomia de Flynn. Os dois conceitos são ortogonais, ou seja, a existência de um não torna necessária a existência do outro. Como já apontado, a superescalaridade apenas aumenta a vazão de instruções, porém não altera a quantidade de fluxos de execução ou se o processador possui disponibilidade de instruções que trabalham com múltiplos dados. Um processador pode ser superescalar e MIMD, superescalar e SIMD ou superescalar e SISD.

Processadores e Operações SIMD

A arquitetura SIMD é capaz de processar uma instrução com múltiplos dados ao mesmo tempo, com o objetivo de agilizar cálculos e operações envolvendo vetores e matrizes. A seguir estão algumas instruções classificadas como SIMD dos processadores Intel Core i3-2120, i5-7400 e i7-8550U. Todos processadores são de arquitetura 64 bits e possuem suporte para as tecnologias Streaming SIMD Extensions (SSE, SSE4.1, SSE4.2) e Advanced Vector Extensions (AVX e AVX2).

Processador Intel Core i3-2120

_mm_test_all_zeros (ptest): A instrução testa se 128 bits (representando inteiros) recebidos por parâmetro são compostos de somente 0's. Ela calcula a lógica AND bit a bit entre o conjunto de valores recebidos e um conjunto 'mask' que representa uma máscara. Caso o resultado seja 0 a função retornará 1, caso contrário retornará 0.

__mm_max_epi32 (pmaxsd): A instrução compara números inteiros de 32 bits com sinal em formato packed passados em 'a' e em 'b' e armazena os valores em 'dst'.

__mm_cmpgt_epi64 (pcmpgtq): A instrução compara números inteiros de 64 bits com sinal em formato packed passados em 'a' e 'b' testando se os valores de a são maiores que os valores de b. Os resultados das comparações são armazenados em 'dst'.

Processador Intel Core i5-7400

_mm256_abs_epi32 (vpabsd): A instrução calcula o valor absoluto de inteiros de 32 bits em formato packed passados em ‘a’ e armazena os resultados em ‘dst’.

_mm256_add_epi64 (vpaddq): A instrução soma inteiros de 64 bits em formato packed passados em ‘a’ e ‘b’ e armazena os resultados em ‘dst’.

_mm256_cmpeq_epi64 (vpcmpeqq): A instrução compara inteiros de 64 bits em formato packed passados em ‘a’ e ‘b’ e testar se são iguais. O resultado das comparações é armazenado em ‘dst’.

_mm_test_all_ones (pcmpeqd): Calcula o resultado lógico de comparações envolvendo um vetor de 128-bits que contém todos os "uns", e retornando 1 se o resultado for zero.

Processador Intel Core i7-8550U

_mm_abs_epi16 (pabsw): A instrução calcula o valor absoluto de inteiros de 16 bits em formato packed passados em ‘a’ e armazena os resultados num espaço de memória ‘dst’.

_mm_shuffle_epi8 (pshufb): A instrução tem como objetivo embaralhar pacotes de 8-bits de inteiros, em acordo com a máscara de controle de embaralhamento nos elementos de 8-bits correspondentes de b, e guardar os resultados em dst

_mm256_add_epi32 (vpaddd): A instrução soma inteiros de 32 bits em formato packed recebidos em ‘a’ e ‘b’ e armazena os resultados.

_mm256_extract_epi16: A instrução extrai inteiros de 16 bits passados em ‘a’ selecionados com um indexador e armazena os resultados em ‘dst’.

Experimento Instruções Vetoriais

Para testar as diferenças de instruções SISD e SIMD, foi utilizado um código na linguagem C no qual são criados 3 vetores de “int” (a, b e c), em que os 2 primeiros já são preenchidos. Após isso, é feito um loop para somar os vetores ‘a’ e ‘b’ e salvar o resultado no vetor ‘c’. E depois, foi utilizado mais alguns “*print*” dentro de loops para mostrar os valores dos vetores.

Ao compilar o código com os comandos: ‘*gcc -S somaVetores.c -o cod-sem-instr-vetorial.asm -O0’*, foi gerado um código .asm (*cod-sem-instr-vetorial.asm*) sem instruções vetoriais. Da linha 31 a 44 é realizado a soma de forma sequencial. Primeiro é salvo o valor do vetor ‘a’, depois o valor do vetor ‘b’ e então é realizada a soma dos 2 valores e salvo no índice correspondente no vetor ‘c’. Esse procedimento é executado 4 vezes, uma vez para cada índice do vetor.

E, ao compilar o código com os comandos: ‘*gcc -S somaVetores.c -o cod-com-instr-vetorial.asm -ftree-vectorize -ftree-vectorizer-verbose=5 -ffast-math -fomit-frame-pointer -march=native -mtune=native -finline -fopt-info-vec-all -O3 -funroll-loops -mavx -msse -msse2’*, foi gerado um código .asm (*cod-com-instr-vetorial.asm*) com instruções vetoriais. Nas linhas 34 a 38 é realizada a soma de forma vetorial, em que a instrução “*vpaddd”* realiza a soma de todo o vetor ‘a’ com o vetor ‘b’ em uma única instrução e depois salva no vetor ‘c’.

Já que os comandos de “*print*” não são vetorizáveis, a parte final de loops para imprimir os valores dos vetores ficou igual nos dois códigos. A mudança mais significativa foi na aplicação da instrução vetorial.

Desse modo, utilizar o código com instruções vetoriais faz o programa ser executado mais rapidamente, visto que possui menos instruções e loops para serem percorridos. Entretanto, também aumenta o tempo para compilar o programa, pois é necessário aplicar as otimizações na hora de gerar o código em assembly. Considerando o objetivo final do programa, é mais vantajoso aumentar o tempo de execução em detrimento do tempo de compilação, pois será compilado apenas uma vez.

Referências Bibliográficas

Parte A

https://course.ece.cmu.edu/~ece447/s13/lib/exe/fetch.php?media=01447203.pdf **(Artigo Original Taxonomia de Flyyn)**

https://cs.stackexchange.com/questions/37758/why-is-a-superscalar-processor-simd **(Discussão Superescalaridade)**

https://www.quora.com/Computer-Architecture-What-is-the-difference-between-scalar-and-superscalar **(Diferença escalar, superescalaridade e mimd)**

https://www.quora.com/Can-a-superscalar-processor-work-with-SIMD-architecture **(SIMD e Superescalaridade)**

https://en.wikipedia.org/wiki/Superscalar_processor **(Processador Superescalar)**

https://en.wikipedia.org/wiki/Multiple_instruction,_multiple_data **(MIMD)**

https://stringfixer.com/pt/Flynn%27s_taxonomy **(Classificações Taxonomia de Flynn)**

https://oque-e.com/o-que-e-superescalar **(Superescalaridade e Multicore)**

Parte B

https://www.intel.com/content/www/us/en/docs/intrinsics-guide/index.html#ig_expand=1493,7182,3146 **(Conjunto Instruções SIMD Intel)**

https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm.html **(Manuais Processadores Intel)**

https://ark.intel.com/content/www/br/pt/ark/products/53426/intel-core-i32120-processor-3m-cache-3-30-ghz.html **(Especificações Intel i3)**

https://ark.intel.com/content/www/br/pt/ark/products/97147/intel-core-i57400-processor-6m-cache-up-to-3-50-ghz.html **(Especificações Intel i5)**

https://www.intel.com.br/content/www/br/pt/products/sku/122589/intel-core-i78550u-processor-8m-cache-up-to-4-00-ghz/specifications.html **(Especificações Intel i7)**

Parte C

https://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html **(Flags de Otimizações)**

https://www.rapidtables.com/code/linux/gcc/gcc-o.html **(Flags -O)**