Insper

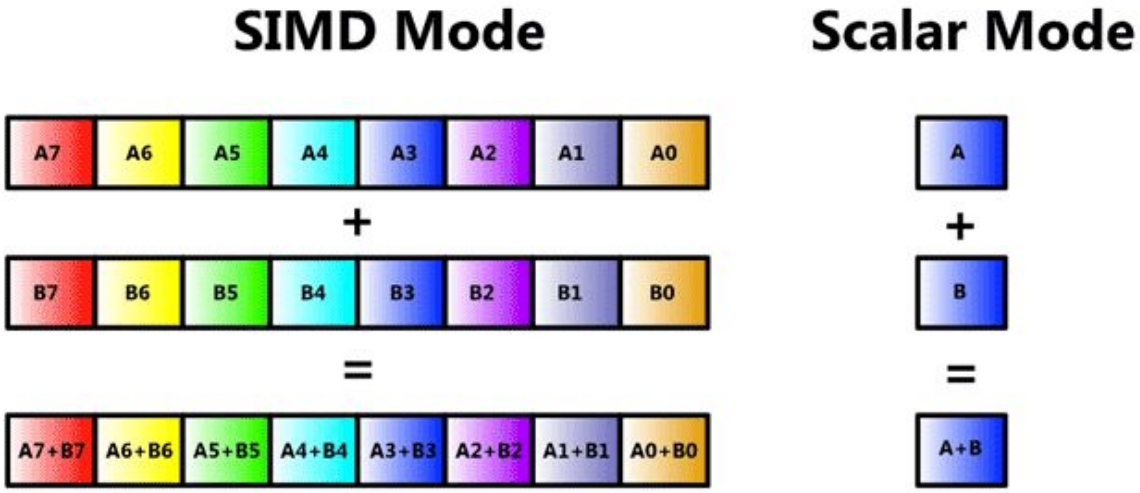
SuperComputação

Aula 6 – Funções Intrínsecas

2019 – Engenharia

Igor Montagner, Luciano Soares <igorsm1@insper.edu.br>

Single Instruction Multiple Data (SIMD)



- Processamento de itens de dados em conjunto
- Operações aritméticas básicas
- Operações comuns mais complexas

Funções intrínsecas

- Streaming SIMD Extensions (SSE) e Advanced Vector Extensions (AVX) podem ser executados através de funções intrínsecas em C.
- Exemplo:
  - Assembly ***vaddps***: soma dois operandos e coloca resultado em um terceiro.
  - Função intrínseca **_mm256_add_ps()** mapeia diretamente para o vaddps

Tipos de Dados

Tipo de Dado	Descrição
__m128	Vetor de 128-bit contendo 4 floats
__m128d	Vetor de 128-bit contendo 2 doubles
__m128i	Vetor de 128-bit contendo inteiros*
__m256	Vetor de 256-bit contendo 8 floats
__m256d	Vetor de 256-bit contendo 4 doubles
__m256i	Vetor de 256-bit contendo inteiros*

* _m256i pode conter 32 chars, 16 shorts, 8 ints, ou 4 longs, além de serem com ou sem sinal (signed ou unsigned).

SSE/AVX

Teste o seguinte código:

```
#include <x86intrin.h>
#include <iostream>

int main() {
   // Cria dois vetores
   __m256 evens = _mm256_set_ps(2.0,4.0,6.0,8.0,10.0,12.0,14.0,16.0);
   __m256 odds = _mm256_set_ps(1.0,3.0,5.0,7.0,9.0,11.0,13.0,15.0);

   // Calcula a diferença dos dois vetores
   __m256 result = _mm256_sub_ps(evens, odds);

   // Exibe o resultado da operação
   float* f = (float*)&result;
   for(int i=0; i<8; ++i) std::cout << f[i] << ' ';
   std::cout << std::endl;
}
```

Compile com: g++ -mavx avx.cpp -o avx

Saída: 1 1 1 1 1 1 1 1

Inicialização com escalares

Função	Descrição
_mm256_setzero_ps/pd	Retorna um vetor de floating-point preenchidos com zero
_mm256_setzero_si256	Retorna um vetor de integer preenchidos com zero
_mm256_set1_ps/pd	Preenche um vetor com um valor floating-point
_mm256_set1_epi8/epi16 _mm256_set1_epi32/epi64	Preenche um vetor com um valor integer
_mm256_set_ps/pd	Inicia o vetor com oito floats (ps) ou quatro doubles (pd)
_mm256_set_epi8/epi16 _mm256_set_epi32/epi64	Inicia o vetor com integers
_mm256_set_m128/m128d/ _mm256_set_m128i	Inicia um vetor de 256-bit com dois de 128-bit
_mm256_setr_ps/pd	Inicia o vetor com oito floats (ps) ou quatro doubles (pd) na ordem reversa
_mm256_setr_epi8/epi16 _mm256_setr_epi32/epi64	Inicia o vetor com integers na ordem reversa

Inicialização com valores em memória

Função	Descrição
_mm256_load_ps/pd	Carrega um vetor alinhado de floating-point de um endereço de memória
_mm256_load_si256	Carrega um vetor alinhado de integers de um endereço de memória
_mm256_loadu_ps/pd	Carrega um vetor não alinhado de floating-point de um endereço de memória
_mm256_loadu_si256	Carrega um vetor não alinhado de integers de um endereço de memória
_mm_maskload_ps/pd _mm256_maskload_ps/pd	Carrega porções de vetor floating-point de 128/256-bit de acordo com máscara

Cada função _maskload_ aceita dois argumentos: um endereço de memória e um vetor de inteiros com o mesmo número de elementos que o vetor de saída. Para cada elemento no vetor de inteiros cujo bit mais significativo é um, o elemento correspondente no vetor retornado é carregado a partir da memória. Se o bit mais significativo no vetor inteiro for zero, o elemento correspondente no vetor retornado é definido como zero.

Exemplo

```
#include <x86intrin.h>
#include <iostream>

int main() {
   float int_array[8] = {100, 200, 300, 400, 500, 600, 700, 800};

   // Criando vetor de máscara
   __m256i mask = _mm256_setr_epi32(-20, -72, -48, -9, -100, 3, 5, 8);

   // Carregando dados de forma seletiva no vetor
   __m256 result = _mm256_maskload_ps(int_array, mask);

   // Exibindo o resultado da operação
   float* res = (float*)&result;
   for(int i=0; i<8; ++i) std::cout << res[i] << ' ';
   std::cout << std::endl;
}
```

Saída: 100 200 300 400 500 0 0 0

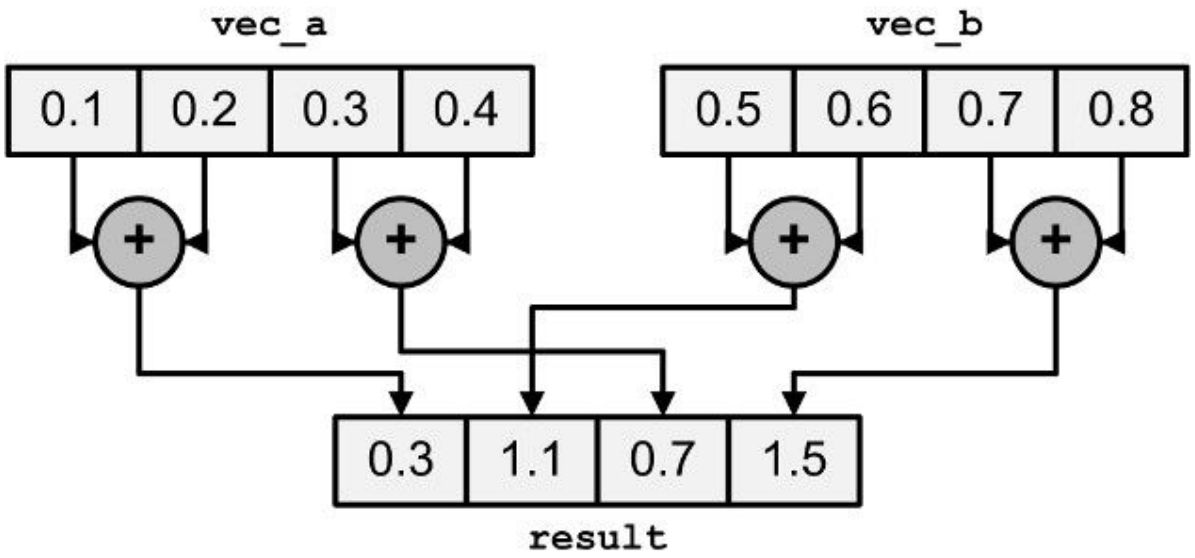
Insper

Aritmética

Função	Descrição
_mm256_add_ps/pd	Soma dois vetores floating-point
_mm256_sub_ps/pd	Subtrai dois vetores floating-point
_mm256_hadd_ps/pd	Soma dois vetores floating-point horizontalmente
_mm256_hsub_ps/pd	Subtrai dois vetores floating-point horizontalmente
_mm256_addsub_ps/pd	Some e subtrai dois vetores floating-point
_mm256_mul_ps/pd	Multiplica dois vetores floating-point
_mm256_div_ps/pd	Divide dois vetores floating-point

Soma “Horizontal”

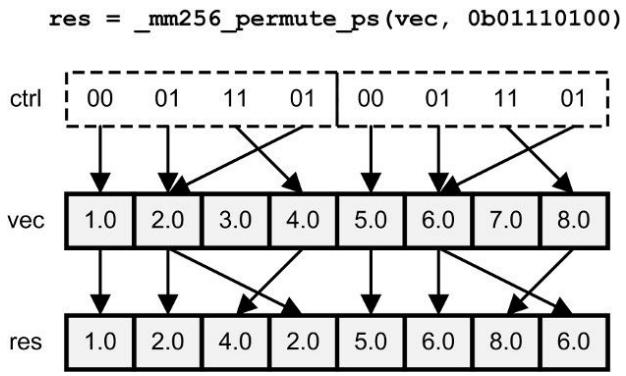
```
__m256d result = _mm256_hadd_pd(vec_a, vec_b);
```

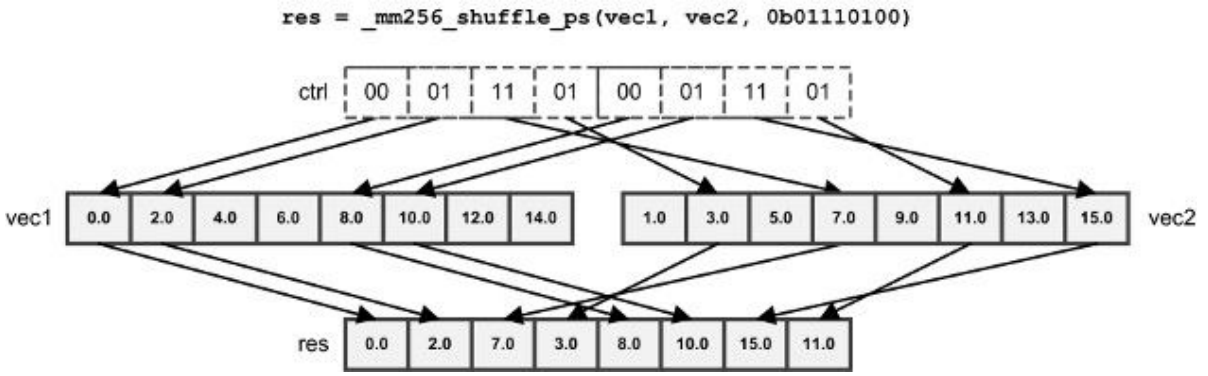


Permutações e embaralhamento

Função	Descrição
_mm_permute_ps/pd/ _mm256_permute_ps/pd	Select elements from the input vector based on an 8-bit control value
_mm256_permute2f128_ps/pd	Select 128-bit chunks from two input vectors based on an 8-bit control value
_mm256_permute2f128_si256	Select 128-bit chunks from two input vectors based on an 8-bit control value
_mm_permutevar_ps/pd _mm256_permutevar_ps/pd	Select elements from the input vector based on bits in an integer vector
_mm256_shuffle_ps/pd	Select floating-point elements according to an 8-bit value
_mm256_shuffle_epi8/ _mm256_shuffle_epi32	Select integer elements according to an 8-bit value

Permutações e embaralhamento





Insper

Auto vetorização

- A vectorização automática só ajuda em alguns casos
- A complexidade crescente das instruções dificulta para o compilador selecionar instruções apropriadas
- Código e lógica precisa ser reconhecido pelo compilador
- Os requisitos de precisão muitas vezes inibem o código SIMD

Auto vetorização

- Dependências de dados
- Outros motivos potenciais
  - Alinhamento
  - Chamadas de função no bloco de loop
  - Fluxo de controle complexo / ramos condicionais
  - Loop não "contabilizável"
    - E.g. tamanho do loop não é uma constante em tempo de execução
  - Tipos de dados misturados
  - Não possui espaçamento comum entre elementos de um vetor
  - Loop muito complexo
  - Vetorização parece ineficaz
- Muitos outros ... mas menos provável que ocorram

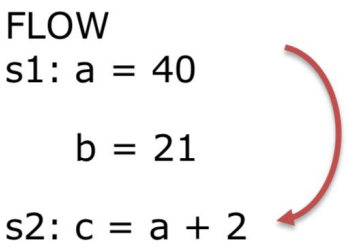
Auto vetorização

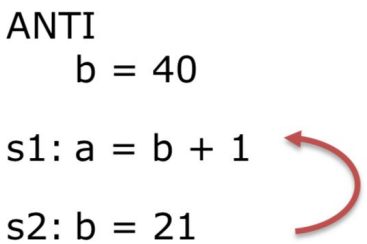
Suponha duas instruções S1 e S2

S2 depende do S1, se o S1 deve ser executado antes do S2

- Dependência do fluxo de controle
- Dependência de dados
- Dependências podem ser transmitidas entre as iterações de loop

Principais sabores das dependências de dados





Atividade final

Implementação de soma_positivos usando intrínsecas

Referências

- Livros:
  - Hager, G. ; Wellein, G. **Introduction to High Performance Computing for Scientists and Engineers**. 1ª Ed. CRC Press, 2010.
- Artigos:
  - Firasta, Nadeem, Mark Buxton, Paula Jinbo, Kaveh Nasri, and Shihjong Kuo. "Intel AVX: New frontiers in performance improvements and energy efficiency." *Intel white paper* 19 (2008): 20.
  - Jeong, Hwancheol, Sunghoon Kim, Weonjong Lee, and Seok-Ho Myung. "Performance of SSE and AVX instruction sets." *arXiv preprint arXiv:1211.0820* (2012).
- Internet:
  - https://monoinfinito.wordpress.com/series/vectorization-in-gcc/
  - https://software.intel.com/en-us/articles/introduction-to-intel-advanced-vector-extensions
  - https://software.intel.com/en-us/isa-extensions
  - https://tech.io/playgrounds/283/sse-avx-vectorization/autovectorization
  - https://software.intel.com/sites/landingpage/IntrinsicsGuide/
  - https://www.codeproject.com/Articles/874396/Crunching-Numbers-with-AVX-and-AVX

Insper

www.insper.edu.br