Insper

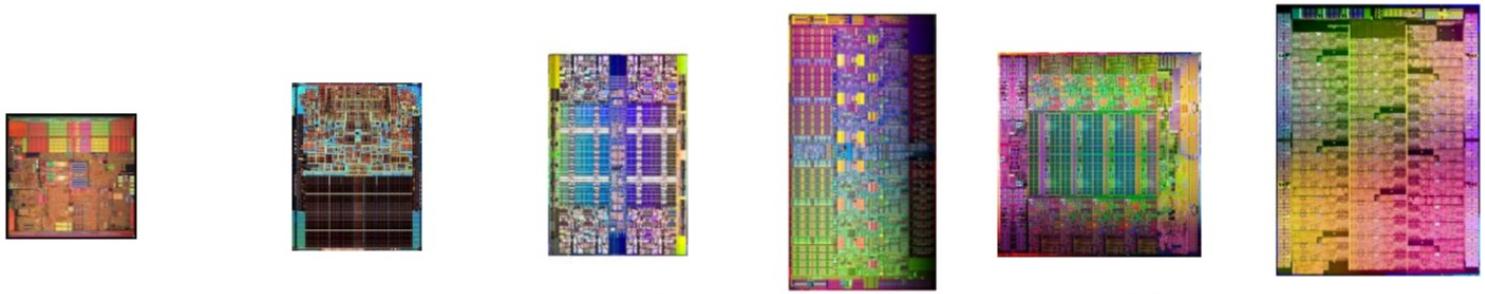
SuperComputação

Aula 5 – Introdução e SIMD

2019 – Engenharia

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

Arquiteturas modernas de CPU



Images not intended to reflect actual die sizes

	64-bit Intel® Xeon® processor	Intel® Xeon® processor 5100 series	Intel® Xeon® processor 5500 series	Intel® Xeon® processor 5600 series	Intel® Xeon® processor E5-2600 series	Intel® Xeon Phi™ Co-processor 5110P
Frequency	3.6GHz	3.0GHz	3.2GHz	3.3GHz	2.7GHz	1053MHz
Core(s)	1	2	4	6	8	60
Thread(s)	2	2	8	12	16	240
SIMD width	128 (2 clock)	128 (1 clock)	128 (1 clock)	128 (1 clock)	256 (1 clock)	512 (1 clock)

Instruction pipelining

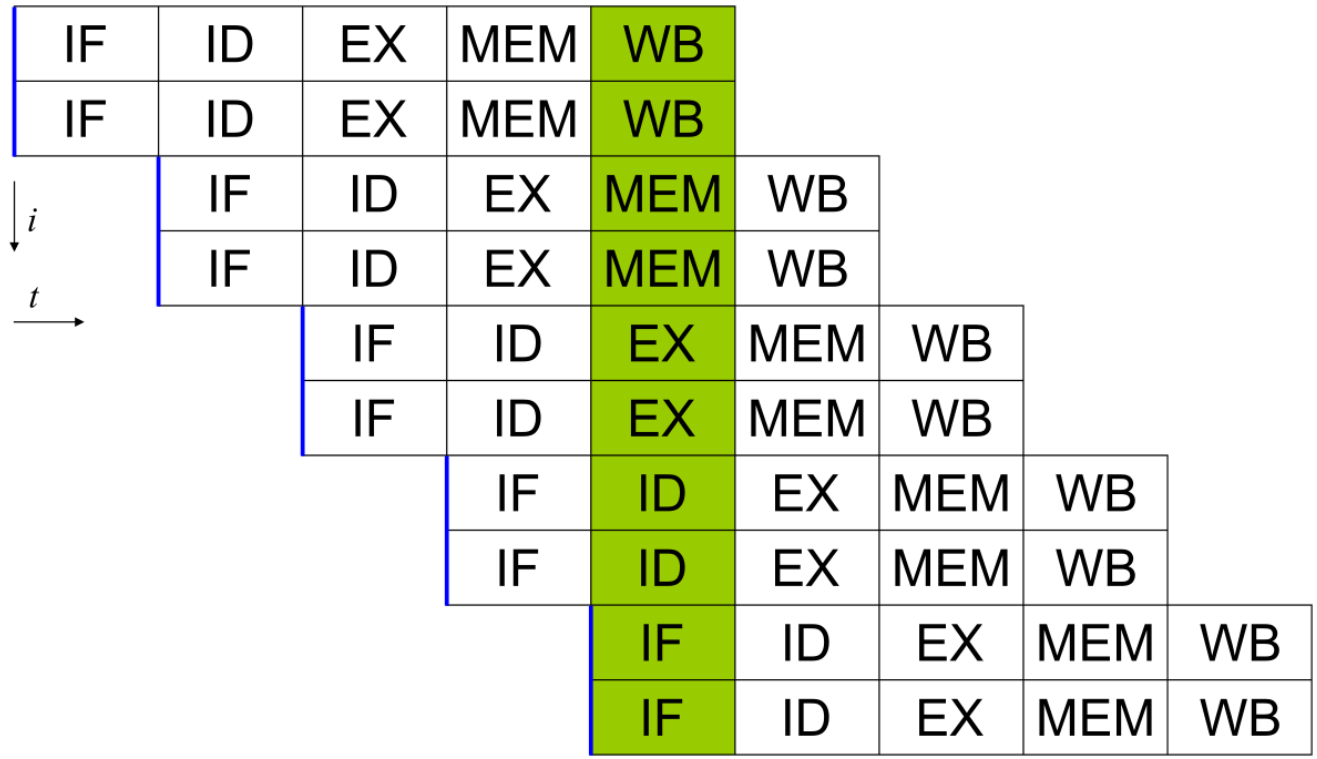
- Não reduz latência de uma tarefa
- Aumenta a taxa de execução de tarefas

Instr. No.	Pipeline Stage						
1	IF	ID	EX	MEM	WB		
2		IF	ID	EX	MEM	WB	
3			IF	ID	EX	MEM	WB
4				IF	ID	EX	MEM
5					IF	ID	EX
Clock Cycle	1	2	3	4	5	6	7

Legend:
IF: Instruction Fetch
ID: Instruction Decode
EX: Execute
MEM:Memory Access
WB: Register Write Back

Instruction pipelining

- Paralelismo em nível de instrução!

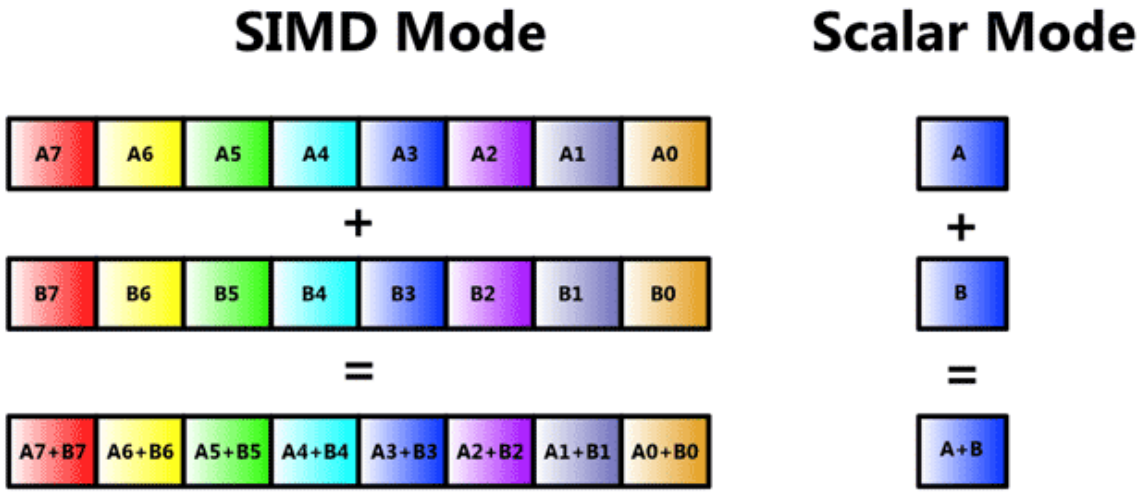


Insper

Instruction pipelining

Base para as vulnerabilidades Spectre e Meltdown

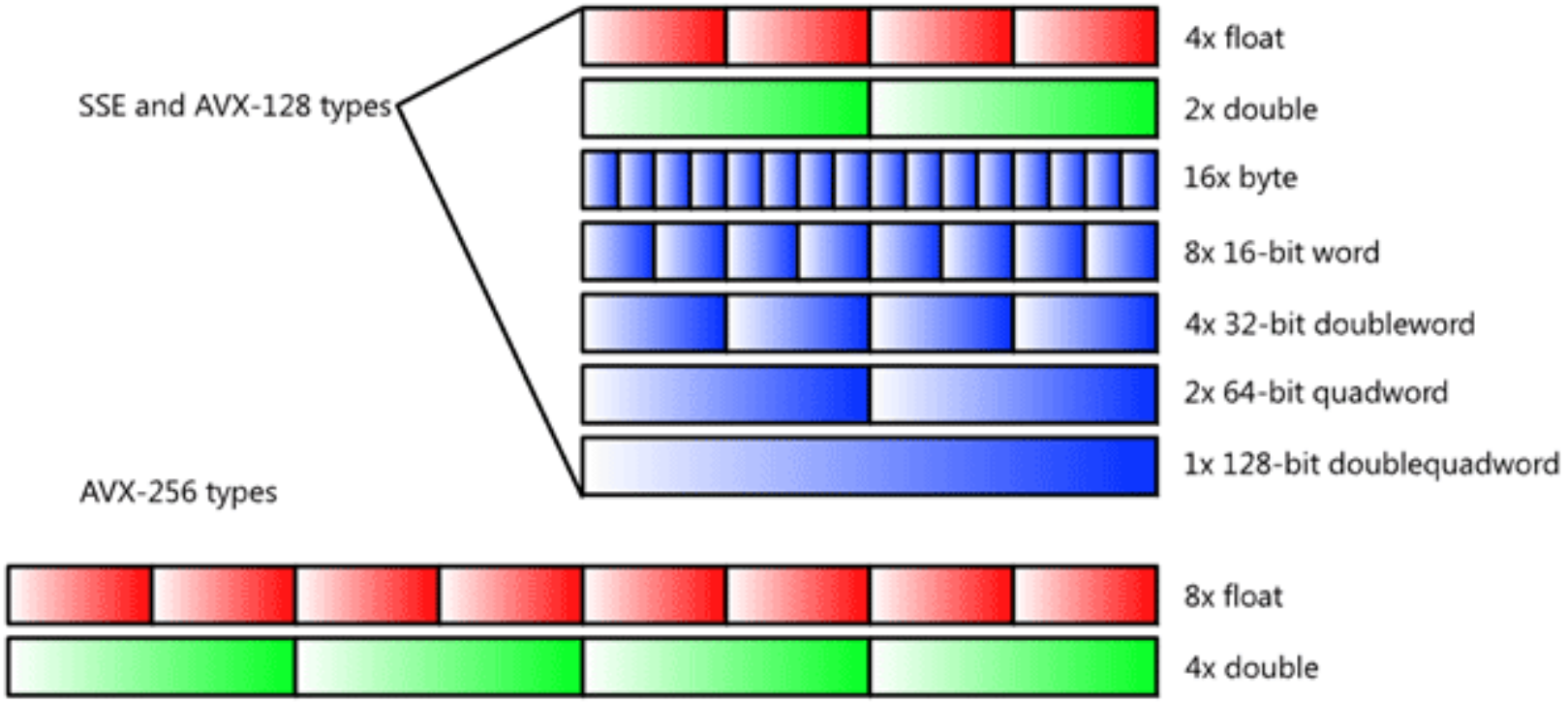
Single Instruction Multiple Data (SIMD)



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

Insper

Single Instruction Multiple Data (SIMD)



Suporte a vetores cada vez maiores

Insper

Arquiteturas modernas de CPU

- Menos velocidade de clock
- Mais núcleos
- Mais trabalho efetuado por clock (pipelining)
- Vetorização (SIMD)

Discussão I

Código

```
#define SIZE (400)
long sum(int v[SIZE]) throw() {
    int s = 0; // este exemplo eh didatico.
               // soma de ints deveria ser long ;)
    for (unsigned i=0; i<SIZE; i++) s += v[i];
    return s;
}
```

Arquivo: tarefa1.cpp

Discussão I

Sem SIMD

```
    leaq    1600(%rdi), %rdx
    xorl    %eax, %eax
.L2:
    addl    (%rdi), %eax
    addq    $4, %rdi
    cmpq    %rdx, %rdi
    jne     .L2
    cltq
    ret
```

Com -ftree-vectorize -mavx

```
   vpxor   %xmm0, %xmm0, %xmm0
   leaq    1600(%rdi), %rax

.L2:
   vpaddd  (%rdi), %xmm0, %xmm0
   addq    $16, %rdi
   cmpq    %rdi, %rax
   jne     .L2
   vpsrldq $8, %xmm0, %xmm1
   vpaddd  %xmm1, %xmm0, %xmm0
   vpsrldq $4, %xmm0, %xmm1
   vpaddd  %xmm1, %xmm0, %xmm0
   vmovd   %xmm0, %eax
   cltq
   ret
```

Discussão I

De onde vem os ganhos de desempenho na versão vetorizada automaticamente?

Discussão I

De onde vem os ganhos de desempenho na versão vetorizada automaticamente?

- Número menor de instruções executadas
- Número menor de pulos
    - Instruções *J** estragam o pipeline da CPU
- 4 somas pelo preço de uma

Tarefa 2

- Objetivos:
    - Explorar o funcionamento do autovetorizador
    - Estudar técnicas de medição de tempo
- Criem do zero um programa e verifique se a autovetorização traz ganhos de desempenho.

Tarefa 3

- Objetivo: comparar desempenho de código vetorizado com código “normal”
  - Funções usadas nas aulas anteriores
  - Pequeno relatório em Jupyter Notebook

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