

SuperComputação

Aula 1 – Introdução e SIMD

2018 – Engenharia

Igor Montagner, Luciano Soares [<igorsm1@insper.edu.br>](mailto:igorsm1@insper.edu.br)

Hoje

- Resumo geral do curso
- Burocracias
- Arquiteturas modernas de CPU
- SIMD

Objetivos de aprendizagem (I)

- Desenvolver algoritmos usando recursos de computação paralela/distribuída para ganhos de desempenho da aplicação final;
- Aplicar estrutura lógica de computação distribuída para o desenvolvimento de algoritmos multitarefas;
- Usar GPGPU para computação numérica e comparar com soluções baseadas em CPU

Objetivos de aprendizagem (II)

- Planejar e projetar sistemas de computação de alto desempenho;
- Analisar a complexidade dos algoritmos paralelos e a eficiência de uma implementação particular, identificando as medidas de desempenho mais adequadas para esta tarefa;
- Aplicar recursos específico de sistemas operacionais para melhorar o desempenho de algoritmos;
- Desenvolver aplicações que utilizam protocolos otimizados para paralelização.

Burocracias

Curso baseado em Projetos

$$MF = (2P1 + 2P2 + 2P3 + 2P4 + AT) / 9$$

- Conceito I em qualquer projeto implica em reprovação
- Só é permitido tirar D em um dos 4 projetos
 - Avaliação complementar no dia da prova final.
 - Não recupera nota, apenas evita reprovação

Burocracias

Horário de atendimento:

Quarta 18:00-19:30

Notas de projetos seguem rubricas específicas divulgadas junto com o enunciado

Esta disciplina requer testes de desempenho decentes.

Tamanhos de entradas

Configurações de paralelismo/sincronização

Profiling

Inspier

PLANEJAMENTO DO SEMESTRE 2018-2 - ENGENHARIA
DISCIPLINAS DO 7º PERÍODO - COMP - DP

DISCIPLINA	AGOSTO				SETEMBRO				OUTUBRO				NOVEMBRO				DEZEMBRO		
	06/08 ATÉ 10/08	13/08 ATÉ 17/08	20/08 ATÉ 24/08	27/08 ATÉ 31/08	03/09 ATÉ 07/09	10/09 ATÉ 14/09	17/09 ATÉ 21/09	24/09 ATÉ 28/09	01/10 ATÉ 05/10	08/10 ATÉ 12/10	15/10 ATÉ 19/10	22/10 ATÉ 26/10	29/10 ATÉ 02/11	05/11 ATÉ 09/11	12/11 ATÉ 16/11	19/11 ATÉ 23/11	26/11 ATÉ 30/11	03/12 ATÉ 07/12	10/12 ATÉ 14/12
					FERIADO - 07/09			PI - 26/09 ATÉ 28/09	PI - 01/10 E 02/10 AULA CANCELADA 03/10	FERIADO - 12/10			FERIADO - 02/11		FERIADO - 15/11 E 16/11	FERIADO - 19/11 E 20/11	PI - 28/11 ATÉ 30/11 AULA CANCELADA 11/04	PI - 03/12 E 04/12 - 05/12 E 06/12	PS
JOGOS DIGITAIS						APSI		PI				APSI					PF Projeto		
TECNOLOGIAS HACKER			AB1				ARSII		P1			ARSIII	ARSIII			ARSIV		P2	
LÓGICA DA COMPUTAÇÃO	*								P1								APS LING P2	PARTE I LING P2	
SUPERCOMPUTAÇÃO	Atividade I				P1			P	PJ2				PJ3				PJ4		P4

Existem 2 assuntos de 20 minutos cada

* Existe o projeto de compilador, são 10 entregas,
mas idealmente é feito em horário de aula.

Ferramentas

- GCC 7.3
- C++11
- Linux / Windows Subsystem for Linux
- Github da disciplina

<http://github.com/igordsm/supercomp>

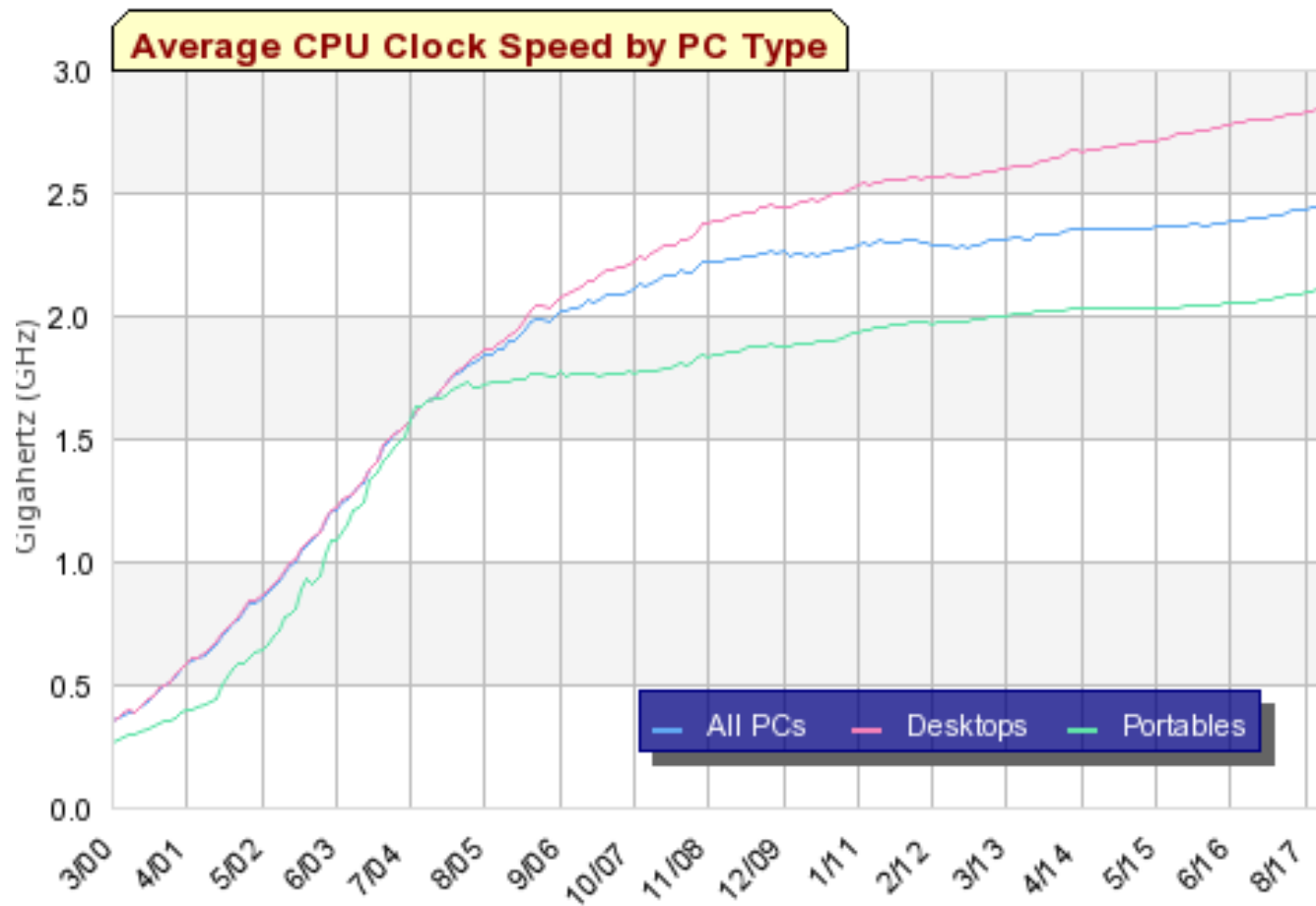
- Blackboard usado para notas e avisos

Objetivo de SuperComputação

Aumentar velocidade de processamento! (Hardware)

- Mais clock
- Memória mais rápida
- Mais núcleos
- Melhor resfriamento

Objetivo de SuperComputação



Objetivo de SuperComputação

Event	Latency	Scaled
1 CPU cycle	0.3 ns	1 s
Level 1 cache access	0.9 ns	3 s
Level 2 cache access	2.8 ns	9 s
Level 3 cache access	12.9 ns	43 s
Main memory access (DRAM, from CPU)	120 ns	6 min
Solid-state disk I/O (flash memory)	50–150 μ s	2–6 days
Rotational disk I/O	1–10 ms	1–12 months
Internet: San Francisco to New York	40 ms	4 years
Internet: San Francisco to United Kingdom	81 ms	8 years
Internet: San Francisco to Australia	183 ms	19 years
TCP packet retransmit	1–3 s	105–317 years
OS virtualization system reboot	4 s	423 years
SCSI command time-out	30 s	3 millennia
Hardware (HW) virtualization system reboot	40 s	4 millennia
Physical system reboot	5 m	32 millennia

Objetivo de SuperComputação

Aumentar velocidade de processamento! (Software)

Objetivo de SuperComputação

Aumentar velocidade de processamento! (Software)

- Interpretador/JIT/compilador mais rápido
- Algoritmos melhores
- Melhorar organização dos dados

Objetivo de SuperComputação

Event	Latency	Scaled
1 CPU cycle	0.3 ns	1 s
Level 1 cache access	0.9 ns	3 s
Level 2 cache access	2.8 ns	9 s
Level 3 cache access	12.9 ns	43 s
Main memory access (DRAM, from CPU)	120 ns	6 min
Solid-state disk I/O (flash memory)	50–150 μ s	2–6 days
Rotational disk I/O	1–10 ms	1–12 months
Internet: San Francisco to New York	40 ms	4 years
Internet: San Francisco to United Kingdom	81 ms	8 years
Internet: San Francisco to Australia	183 ms	19 years
TCP packet retransmit	1–3 s	105–317 years
OS virtualization system reboot	4 s	423 years
SCSI command time-out	30 s	3 millennia
Hardware (HW) virtualization system reboot	40 s	4 millennia
Physical system reboot	5 m	32 millennia

Objetivo de SuperComputação

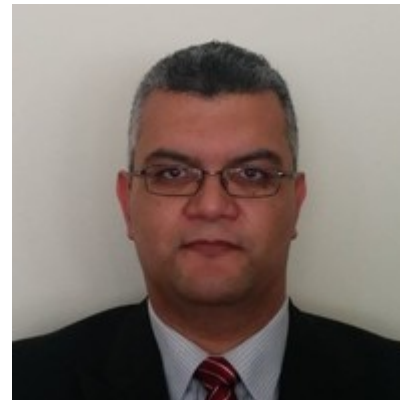
Notação	Nome	Característica	Exemplo
$O(1)$	constante	independe do tamanho n da entrada	determinar se um número é par ou ímpar; usar uma tabela de dispersão (hash) de tamanho fixo
$O(\log n)$	logarítmica	o problema é dividido em problemas menores	busca binária
$O(n)$	linear	realiza uma operação para cada elemento de entrada	busca sequencial; soma de elementos de um vetor
$O(n \log n)$	log-linear	o problema é dividido em problemas menores e depois junta as soluções	heapsort, quicksort, merge sort
$O(n^2)$	quadrática	itens processados aos pares (geralmente loop aninhado)	bubble sort (pior caso); quick sort (pior caso); selection sort; insertion sort
$O(n^3)$	cúbica		multiplicação de matrizes $n \times n$; todas as triplas de n elementos
$O(n^c), c > 1$	polinomial		caixeiro viajante por programação dinâmica
$O(c^n)$	exponencial	força bruta	todos subconjuntos de n elementos
$O(n!)$	fatorial	força bruta: testa todas as permutações possíveis	caixeiro viajante por força bruta

Objetivo de SuperComputação

Aumentar velocidade de processamento! (Software)

- Interpretador/JIT/compilador mais rápido
- Algoritmos melhores
- Melhorar organização dos dados

Sistemas Hardware-software



Objetivo de SuperComputação

Aumentar velocidade de processamento! (Software)

- Interpretador/JIT/compilador mais rápido
- Algoritmos melhores
- Melhorar organização dos dados

Desafios de Programação

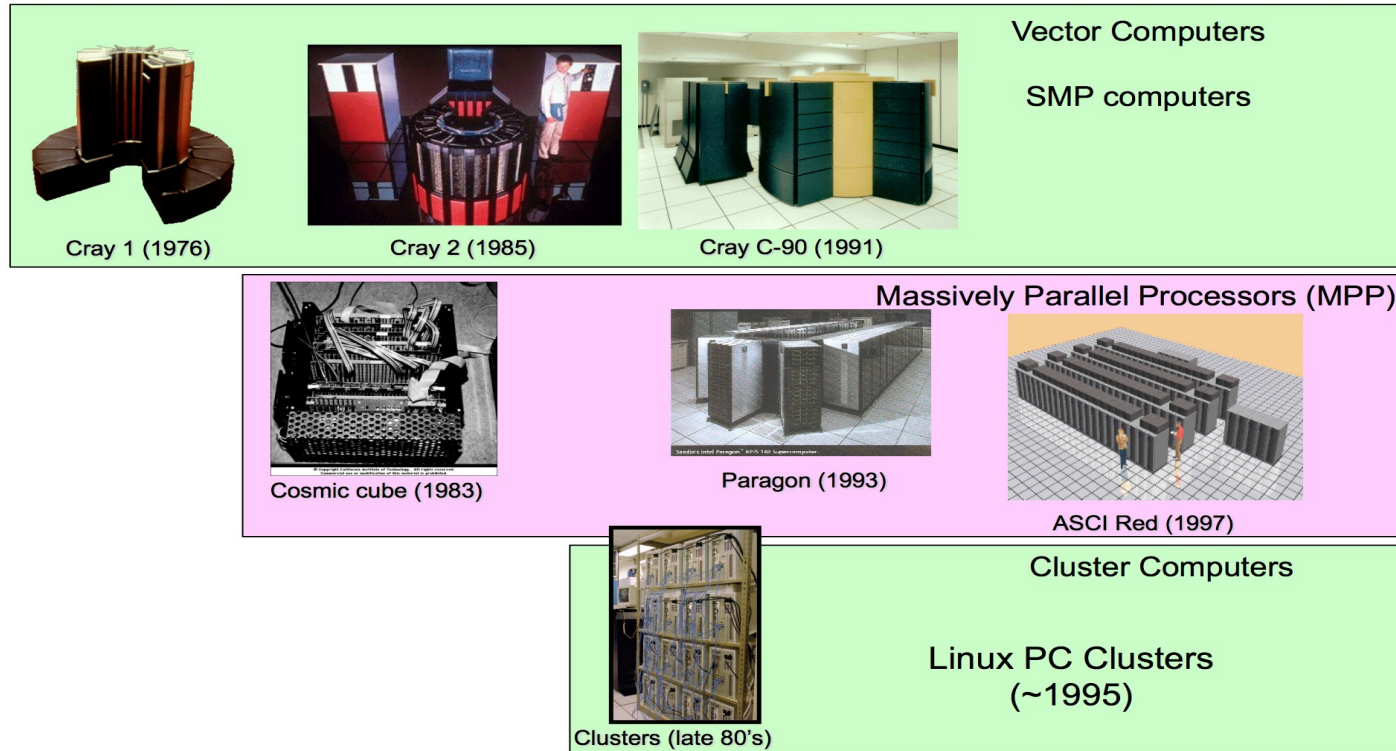


Objetivo de SuperComputação

Aumentar velocidade de processamento!

Discussão: como vocês fariam isso?

Soluções encontradas



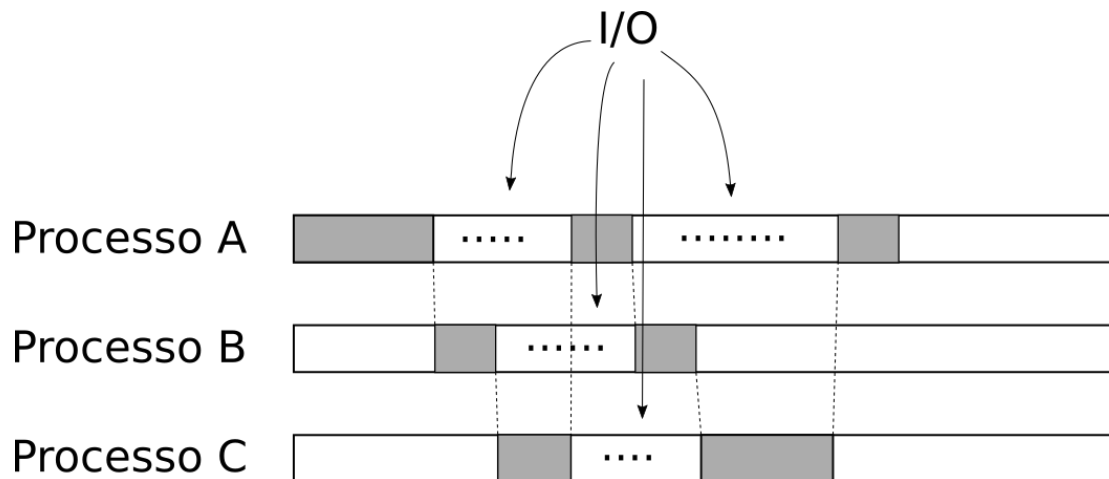
Como programar para estes computadores/arquiteturas?

Conteúdos

- Programação concorrente e sincronização
- Programação paralela em CPUs multi core
- Sistemas distribuídos
- GPGPU

Programação concorrente

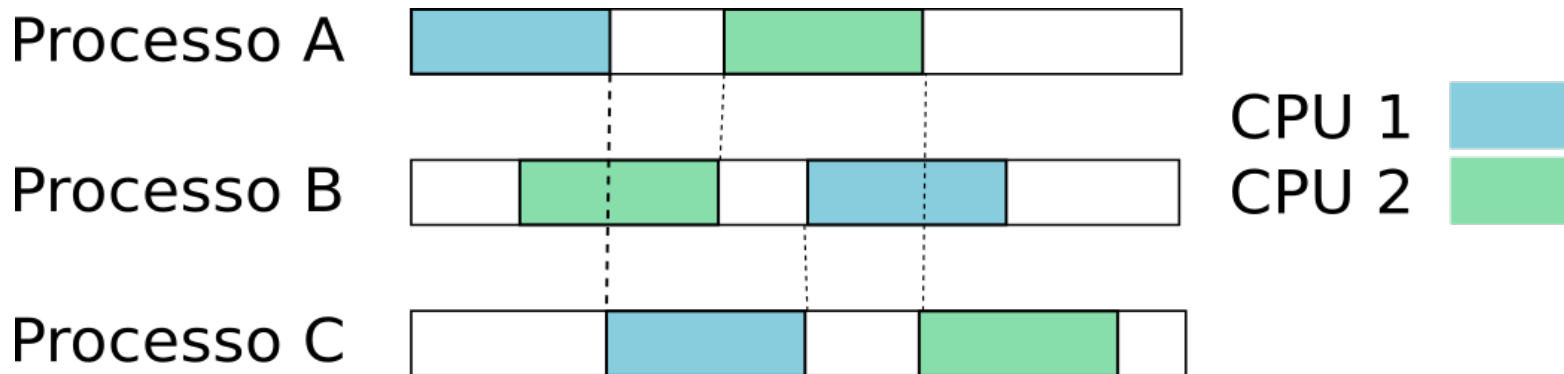
- Tarefas limitadas por entrada e saída



- Estratégia de divisão do problema em tarefas
- Primitivas para sincronizar a execução

Programação Multi core

- Tarefas limitadas por CPU



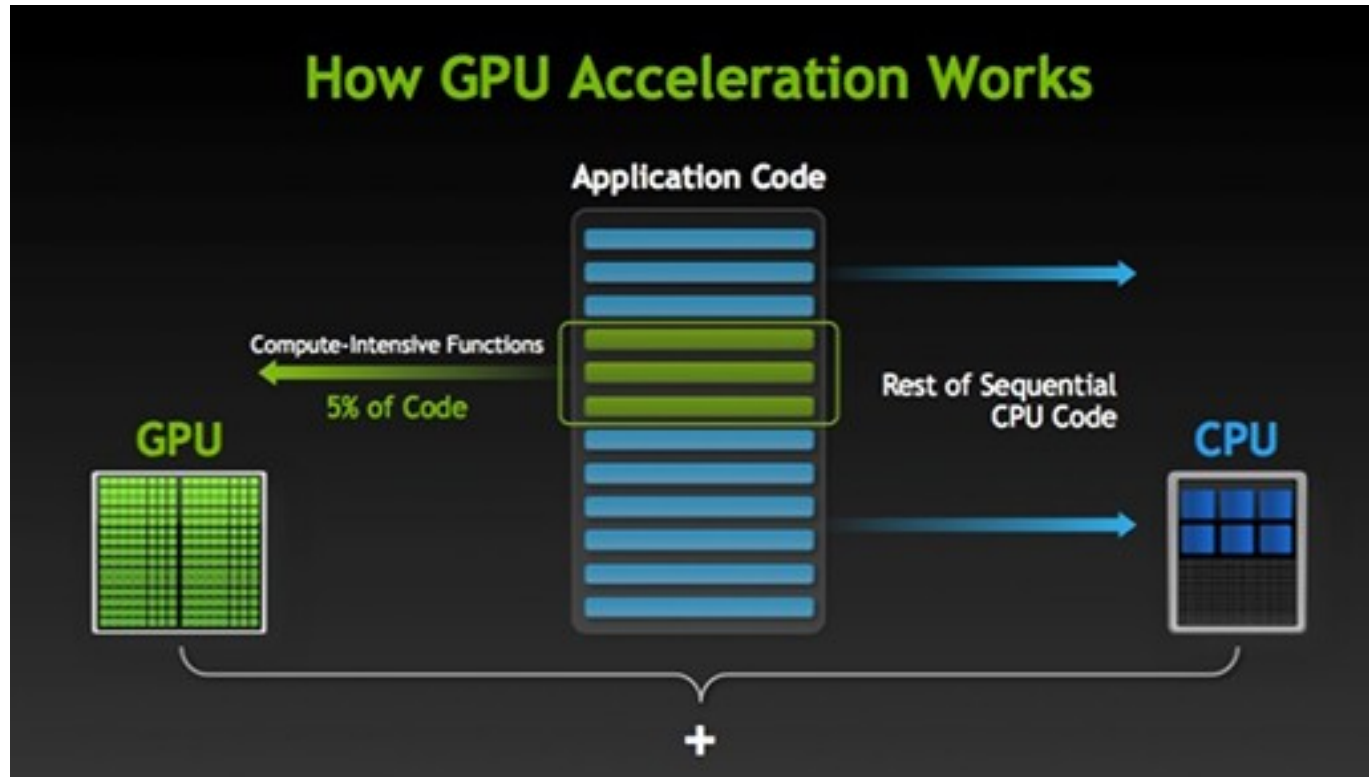
- Divisão em partes independentes
- Modelo fork-join

Sistemas distribuídos



- Divisão de tarefas em clusters
- Passagem de mensagens entre processos/máquinas

GPGPU



- Versão turbinada do modelo multi-core
- Arquitetura completamente diferente

Revolução no acesso a recursos

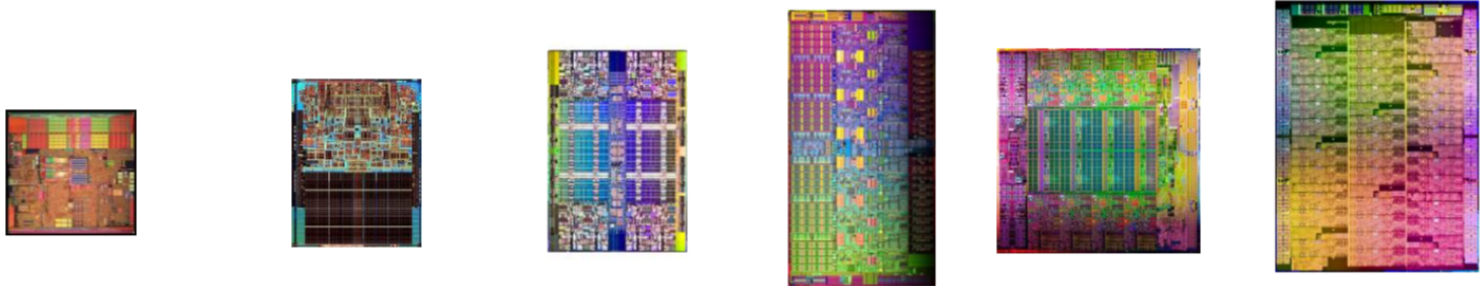


OPENSIFT

Super Computação sob demanda!

Parte 0 – Vetorização

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

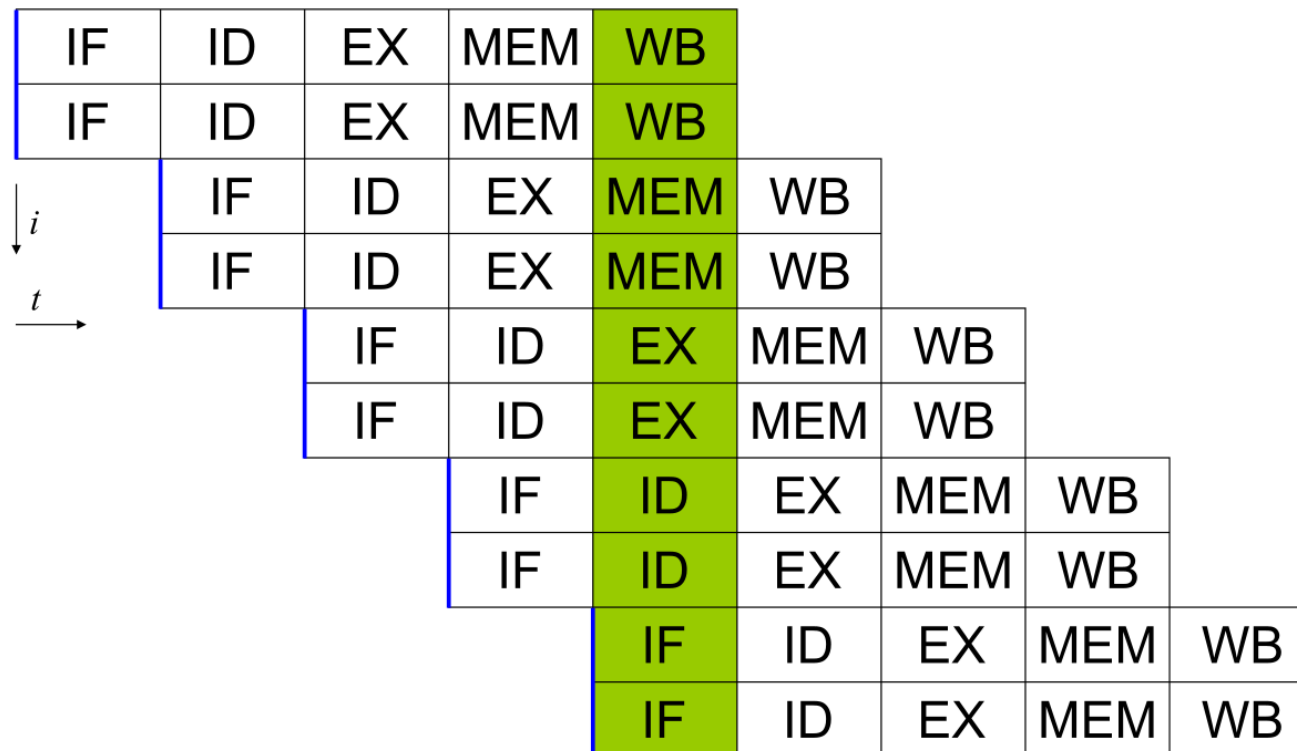
EX: Execute

MEM: Memory Access

WB: Register Write Back

Instruction pipelining

- Paralelismo em nível de instrução!

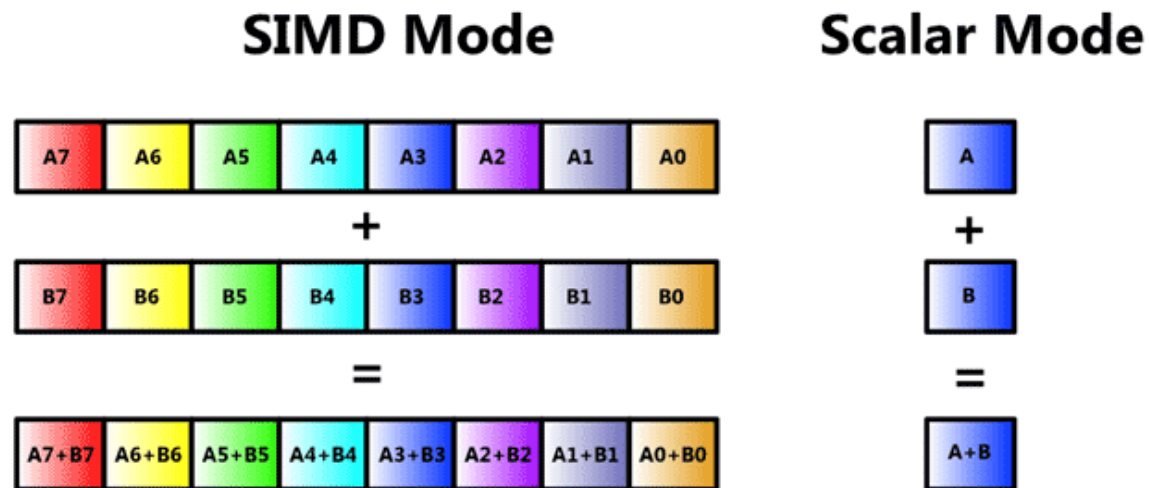


Instruction pipelining



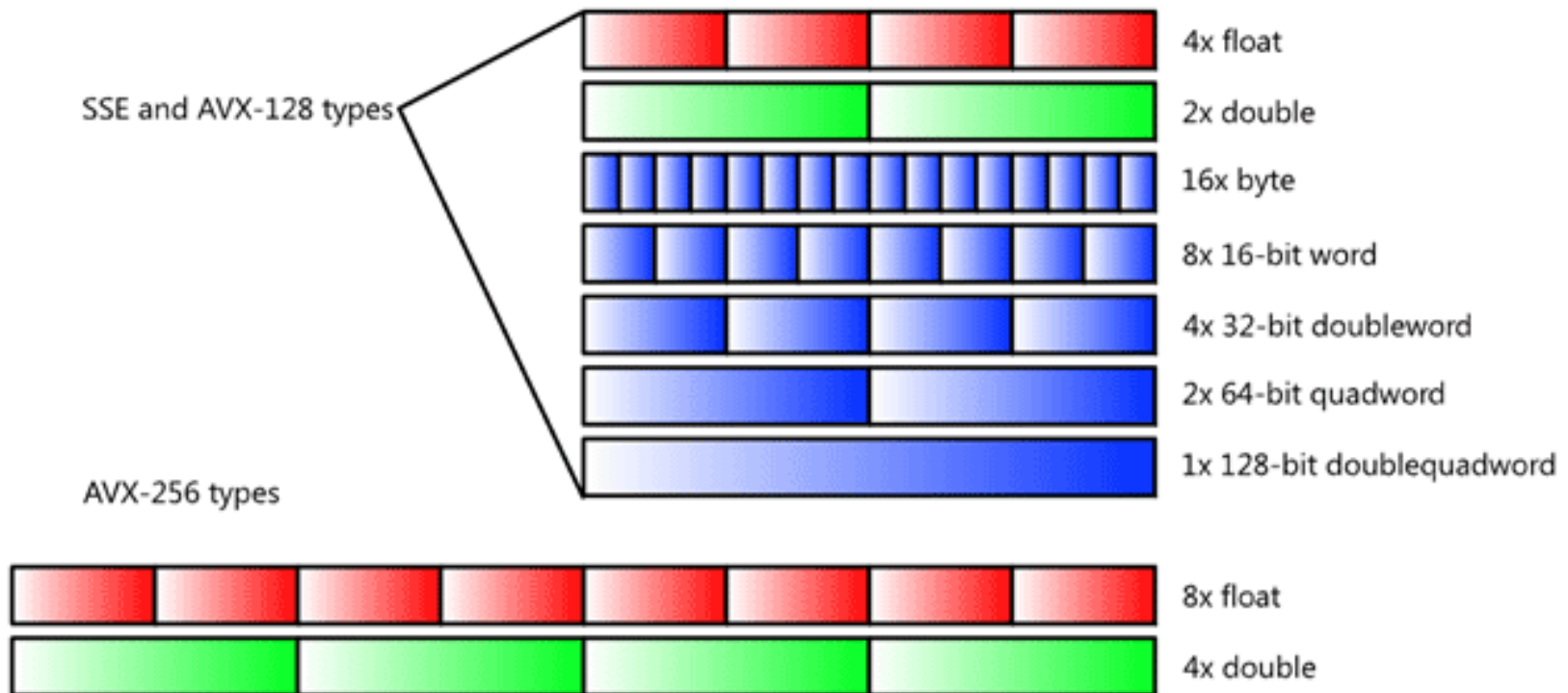
Base para as vulnerabilidades Spectre e Meltdown

Single Instruction Multiple Data (SIMD)



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

Single Instruction Multiple Data (SIMD)



Suporte a vetores cada vez maiores

Arquiteturas modernas de CPU

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

Atividade

- Baixem repo da disciplina

<http://github.com/igordsm/supercomp>

- Tarefa 1 da aula 01-introducao (10 minutos)

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

Discussão II

Qual versão da função anterior você usaria se seu código fosse executado em processadores de baixo custo (Intel Celeron) ou muito antigos (mais de 5 anos)? E se o plano for executar em processadores novos?

Discussão II

Qual versão da função anterior você usaria se seu código fosse executado em processadores de baixo custo (Intel Celeron) ou muito antigos (mais de 5 anos)? E se o plano for executar em processadores novos?

Versão if = só CPUs com SIMD

Versão ? = CPUs antigas e novas

Tarefa 3

- Objetivo: comparar desempenho de código vetorizado com código “normal”
 - Diversas funções em *tarefa3.cpp*
 - Realizem testes e escrevam um relatório mostrando seus resultados

Entrega via Blackboard para a próxima aula

Traga duas cópias impressas do seu relatório

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