

“É o hardware que torna um computador rápido. É o software que transforma um computador rápido em lento” (Craig Bruce).

Paralelismo e SIMD

Paulo Ricardo Lisboa de Almeida

No processador MIPS32 estudado

Temos um paralelismo a nível de instrução.

Através do pipeline, múltiplas instruções são executadas “ao mesmo tempo na CPU”.

No entanto.

Somente uma instrução é enviada a cada ciclo de clock.

No máximo uma instrução é completada a cada ciclo.

Temos um CPI de 1, na melhor das hipóteses.

As instruções operam em apenas um dado.

Exemplo: fazem a operação e armazenam o resultado em um registrador.

SISD

A CPU MIPS32 estudada é um exemplo de uma CPU **SISD**.

Single Instruction **S**ingle **D**ata.

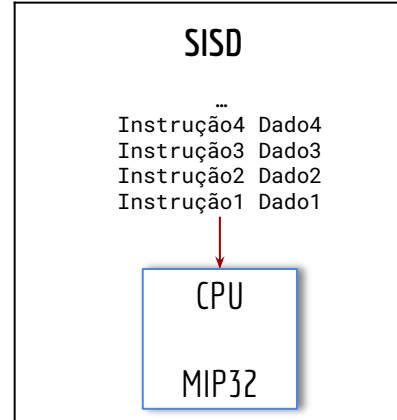
Uma instrução para um dado.

Processadores que executam uma instrução por vez, e cada instrução é capaz de operar em apenas um dado.

Taxonomia de Flynn

Única Instrução.
Single Instruction - SI.

Único Dado.
Single Data - SD.



Atenção: estamos nos referindo ao MIPS32 estudado. Existem implementações MIPS que não são SISD. Pesquise na internet.

SIMD

SIMD - Single instruction Multiple Data.

Uma instrução operando em múltiplos dados.

Exemplo:

Uma instrução que soma um valor imediato em múltiplos registradores ao mesmo tempo.

```
addi $s0, $s0, 1 #instrução MIPS32 SISD
```

```
addi $s0, $s1, $s2, $s3, 1 #Exemplo SIMD de uma instrução somando o imediato em 4 registradores ao mesmo tempo.
```

SIMD

SIMD - Single instruction Multiple Data.

Uma instrução operando em múltiplos dados.

Exemplo:

Uma instrução que soma um valor imediato em múltiplos registradores ao mesmo tempo.

```
addi $s0, $s0, 1 #instrução MIPS32 SISD
```

```
addi $s0, $s1, $s2, $s3, 1 #Exemplo SIMD de uma instrução somando o imediato em 4 registradores ao mesmo tempo.
```

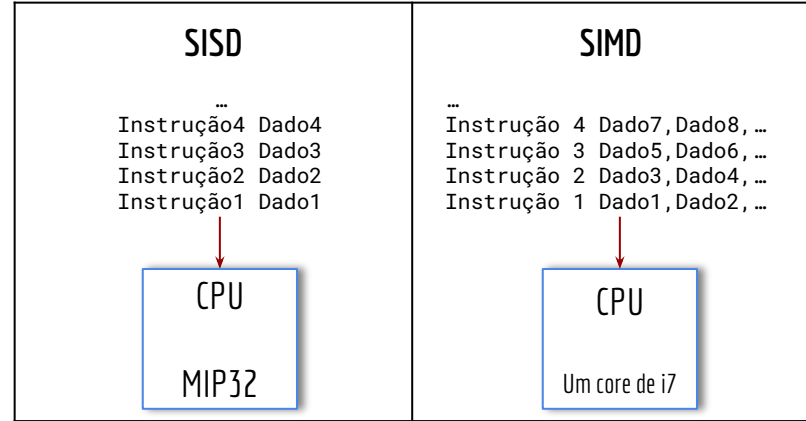
Comum em nossos processadores atuais.

Taxonomia de Flynn

Única Instrução.
Single Instruction - SI.

Único Dado.
Single Data - **SD.**

Múltiplos Dados.
Multiple Data - **MD.**

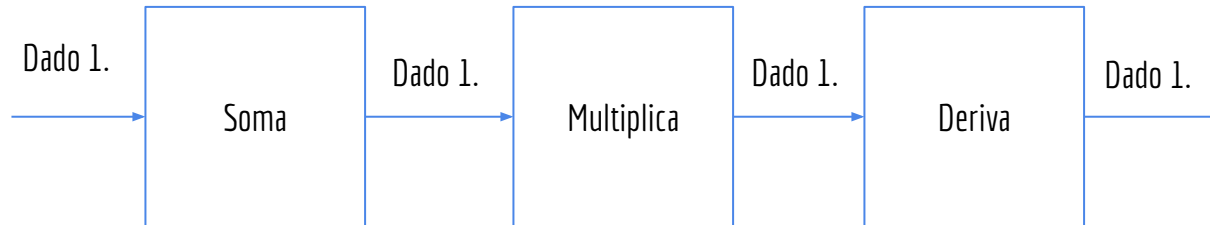


MISD

MISD - Multiple instruction single data.

Múltiplas instruções operando em um dado único.

Exemplo:

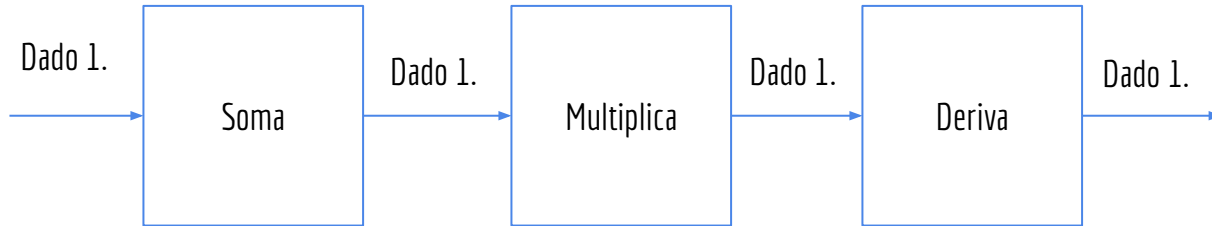


MISD

MISD - Multiple instruction single data.

Múltiplas instruções operando em um dado único.

Exemplo:



Não existem computadores puramente MISD atualmente.

Exemplo de processadores que são (discutivelmente) similares a MISD são CPUs sistólicas e GPUs.

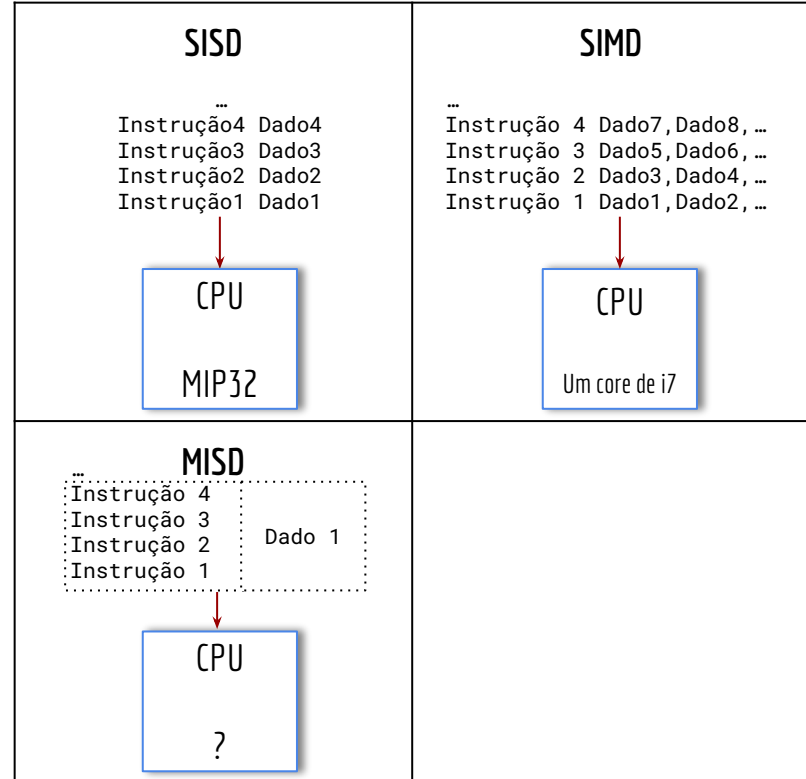
Leia sobre eles na Bibliografia.

Taxonomia de Flynn

Único Dado.
Single Data - SD.

Múltiplos Dados.
Multiple Data - MD.

Única Instrução.
Single Instruction - SI.



Múltiplas Instruções.
Multiple Instruction - MI.

Multiprocessadores

Um multiprocessador é um sistema de processamento composto de **múltiplos processadores**.

Alguns tipos de multiprocessadores:

- Multiprocessadores UMA.

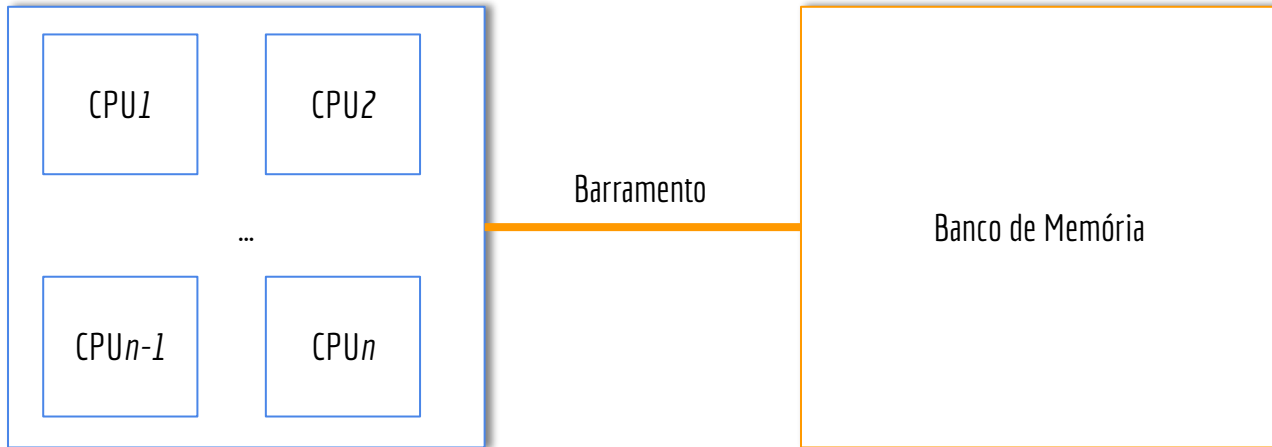
- Multiprocessadores NUMA.

- Clusters.

Multiprocessadores UMA

UMA - Uniform Memory Access.

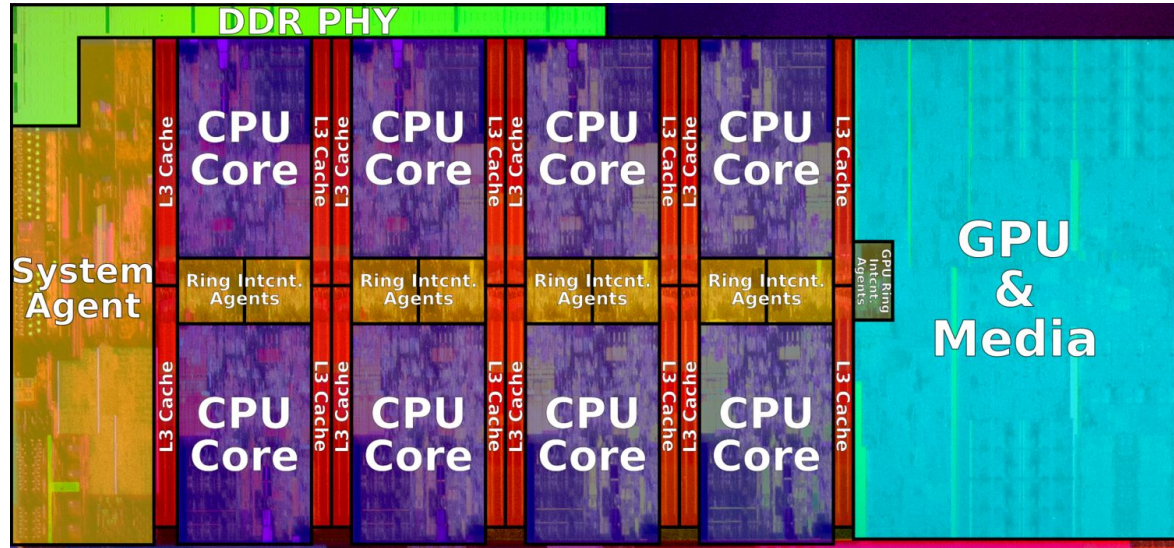
Acesso uniforme à memória - Todos os processadores têm a mesma prioridade para acessar o banco de memória principal.



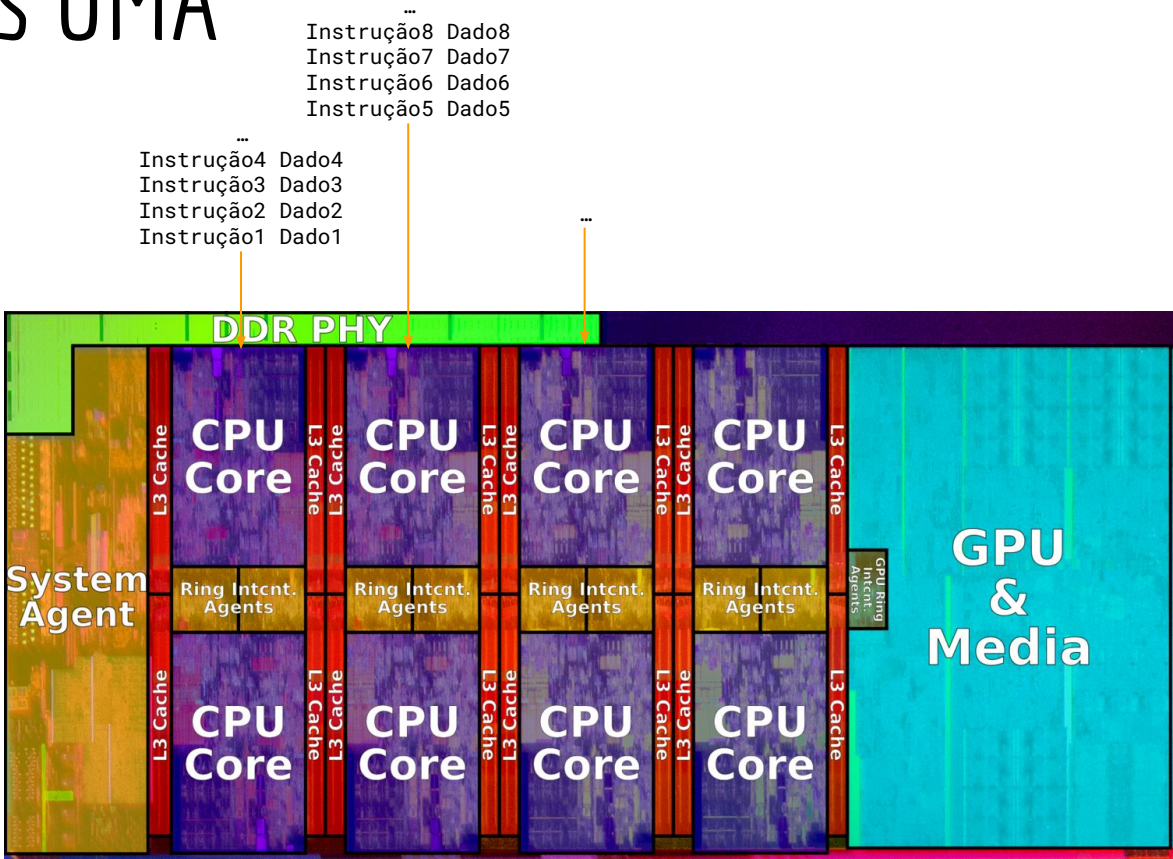
Multiprocessadores UMA

A maioria das máquinas x86-64 atuais são exemplos de multiprocessadores UMA.

A indústria os chama de Processadores Multicore.



Multiprocessadores UMA





Intel i9 9900k.



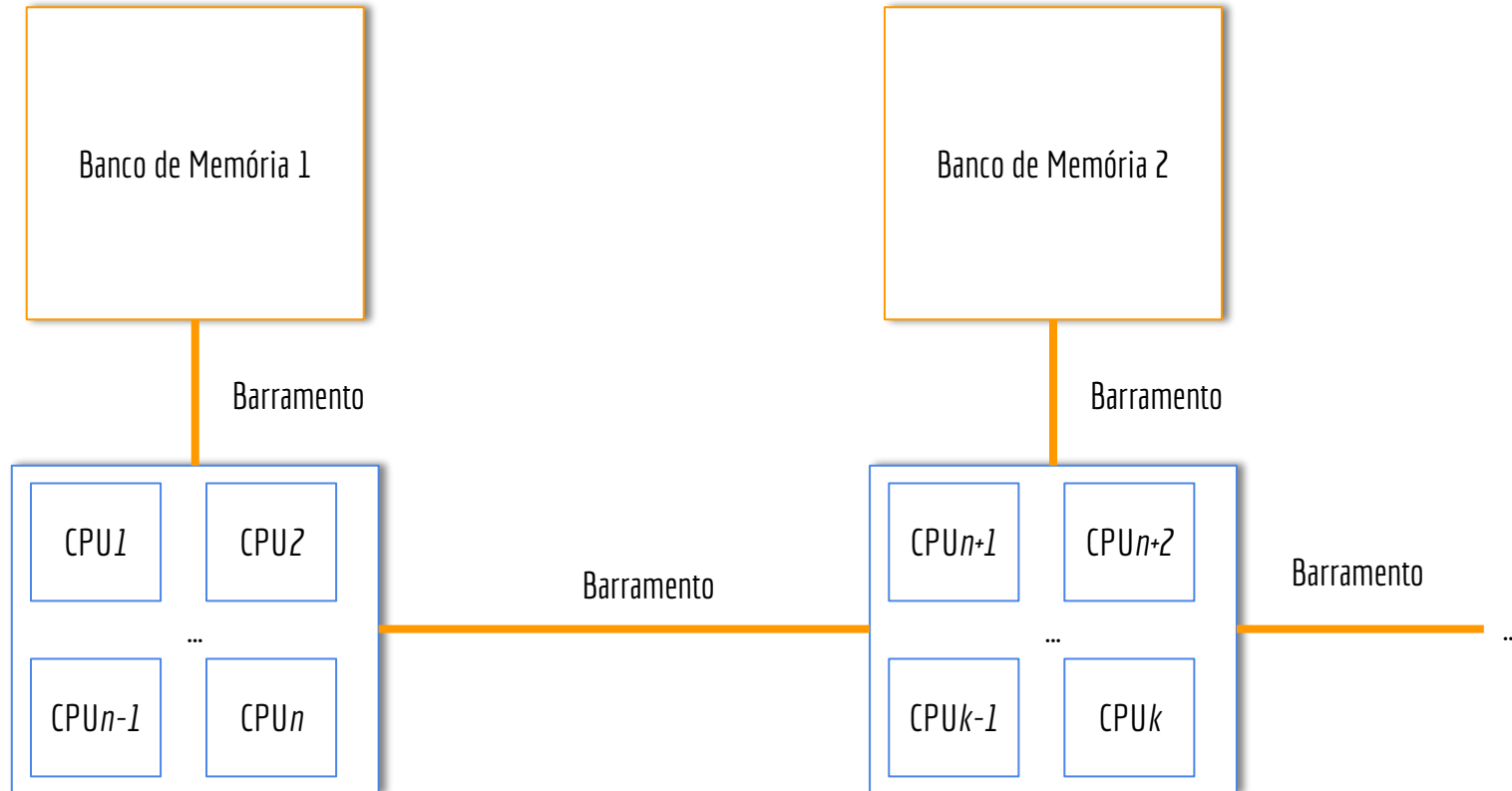
Gigabyte Z390 Aorus Ultra.

Multiprocessadores NUMA

NUMA - Non-uniform memory access.

Acesso **não** uniforme à memória - Determinados grupos de processadores têm prioridade para acessar determinados bancos de memória.

Multiprocessadores NUMA





Intel Xeon 4110.



ASUS WS C621E.

Multiprocessadores NUMA - Alguns Problemas

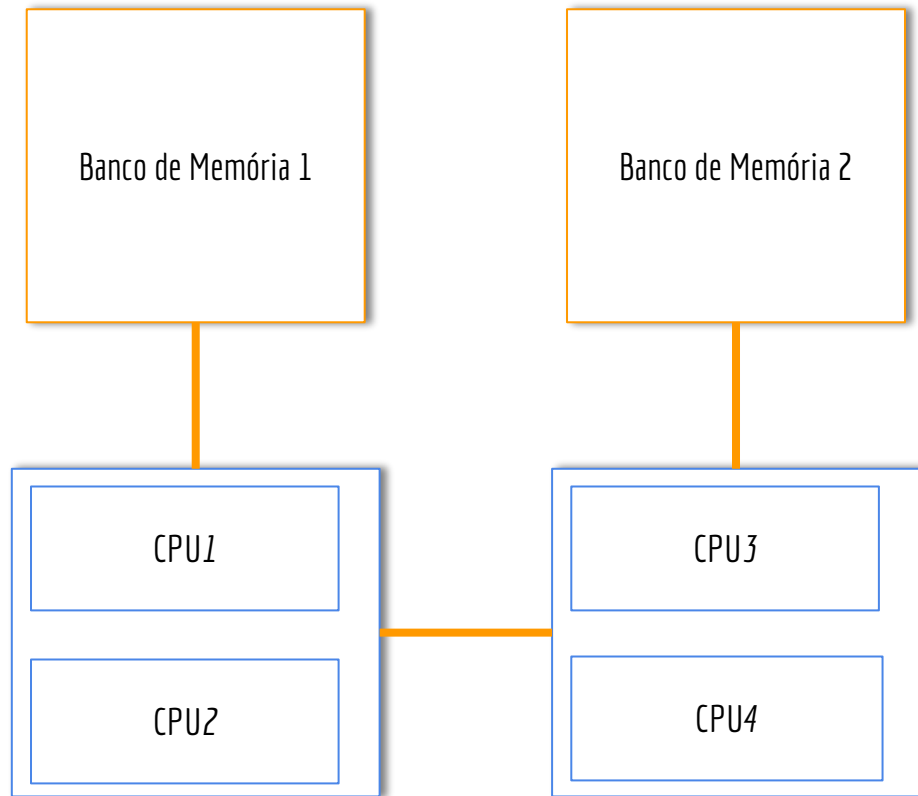
First Touch

Quando a memória é “tocada pela primeira vez”, é definido o seu banco de memória.

```
int main(){
    int vetor* = malloc(...);
    vetor[0] = ...; //first touch

    //...

    return 0;
}
```

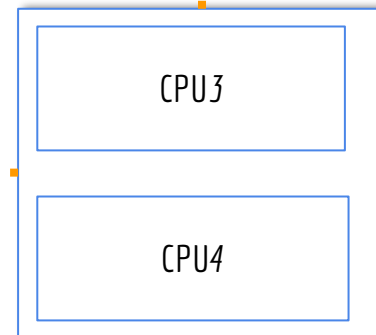
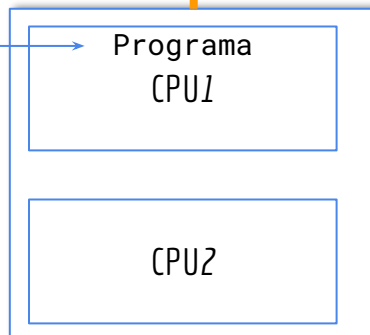
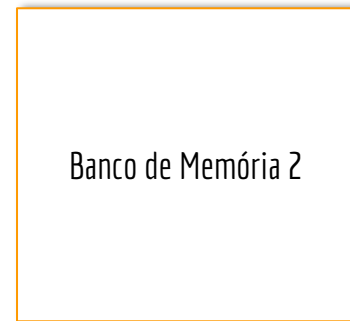


Multiprocessadores NUMA - Alguns Problemas

First Touch

Quando a memória é “tocada pela primeira vez”, é definido o seu banco de memória.

```
int main(){  
    int vetor* = malloc(...);  
    vetor[0] = ...; //first touch  
    //...  
    return 0;  
}
```

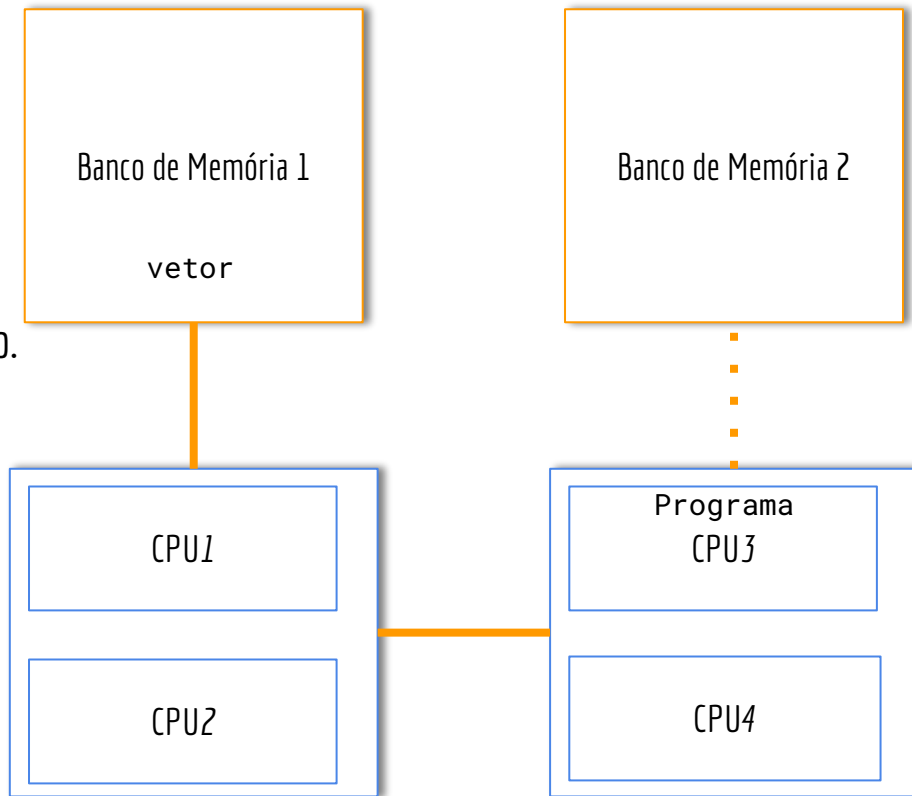


Multiprocessadores NUMA - Alguns Problemas

First Touch

Se o programador e o sistema operacional não forem espertos, o processo pode migrar para uma CPU que está em outro banco durante sua execução.

O acesso à memória pode se tornar penosamente lento.



Faça você mesmo

Execute em um terminal o comando `stress-ng --cpu 1 -t 1m`

Em outro terminal execute `htop`

Note que o processo que está estressando a CPU pode trocar de CPU no meio do caminho.

O sistema operacional pode escolher trocar de CPU para, por exemplo, reduzir o aquecimento em um núcleo.

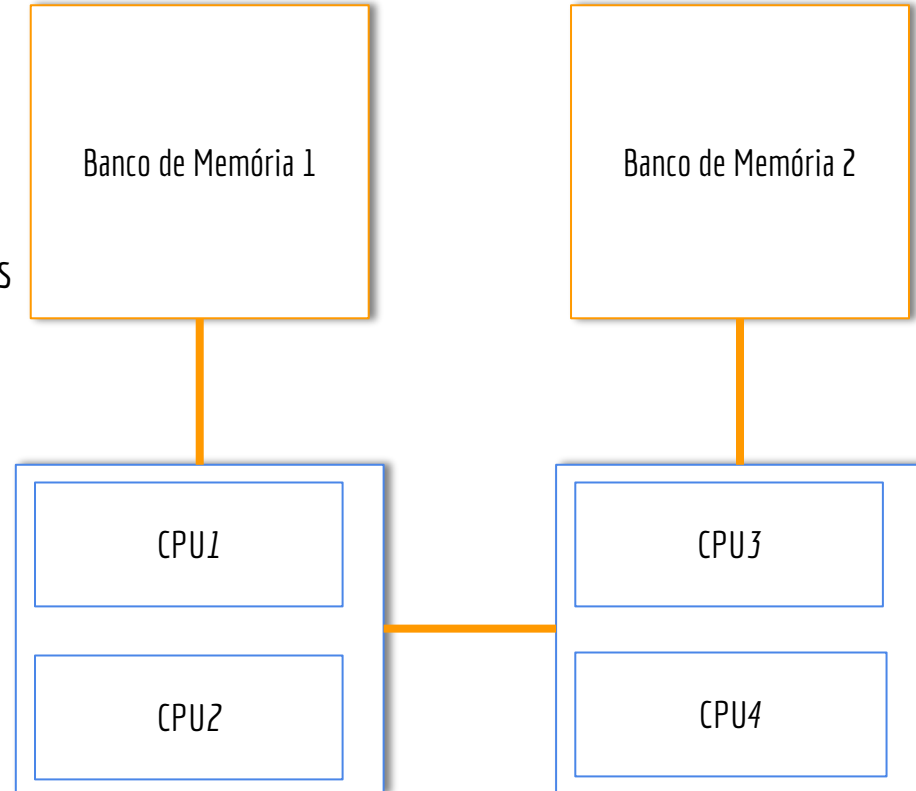
Multiprocessadores NUMA - Alguns Problemas

Os processadores geralmente possuem uma memória cache.

Uma pequena cópia da memória principal.

Estudaremos nas próximas aulas.

Em uma máquina NUMA, as CPUs podem ficar com as caches incoerentes.



Multiprocessadores NUMA - Alguns Problemas

Os processadores geralmente possuem uma memória cache.

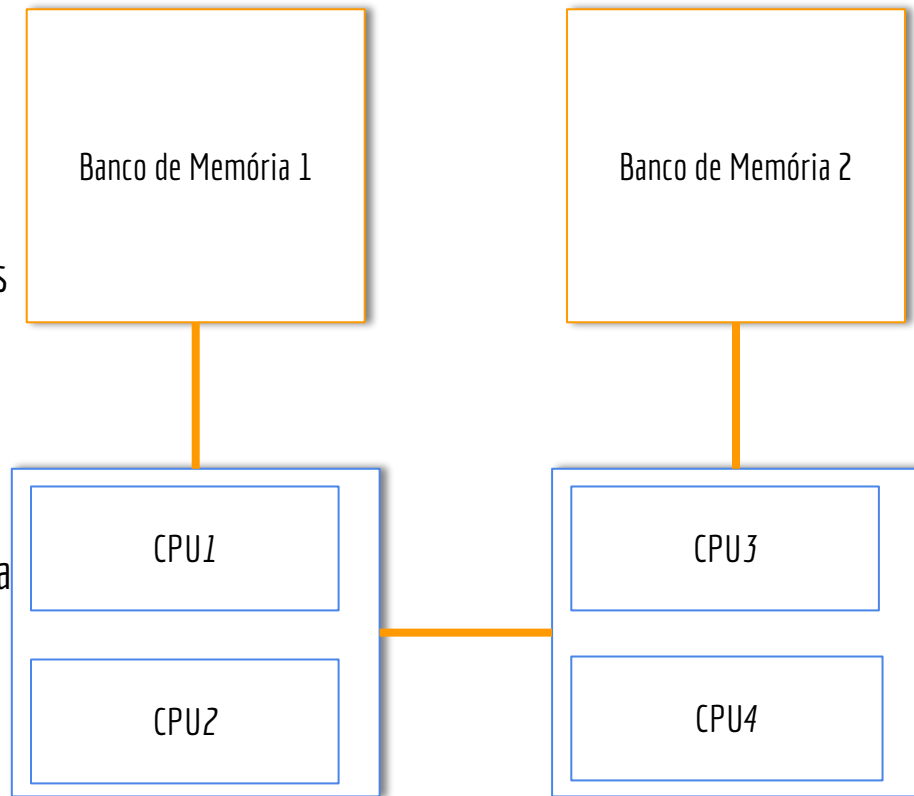
Uma pequena cópia da memória principal.

Estudaremos nas próximas aulas.

Em uma máquina NUMA, as CPUs podem ficar com as caches incoerentes.

CC-NUMA (Cache Coherent NUMA): Coerência de cache garantida pelo hardware. Custa caro em tempo e \$\$.

NCC-NUMA (Non-Cache Coherent NUMA): Não existe garantia de coerência da cache. O programador que se vire.



Cluster

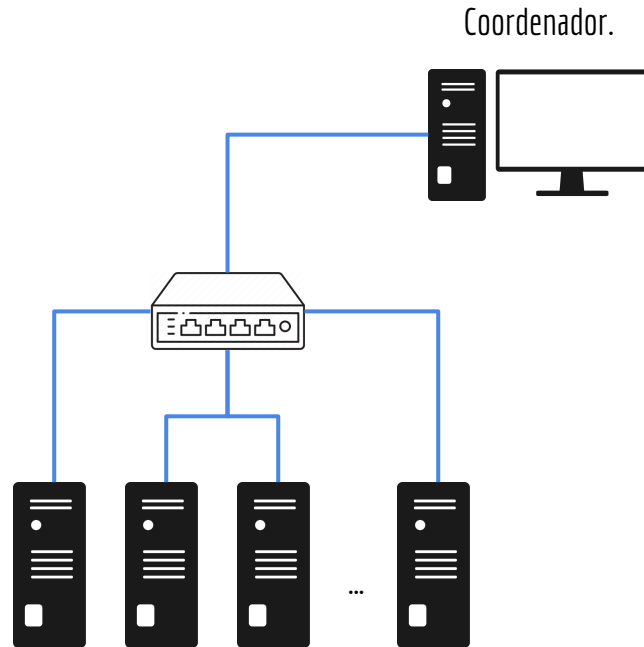
Coleção de máquinas trabalhando em conjunto.

Espaços de endereçamento separados.

Um S.O. executando cada máquina.

Máquinas interconectadas: internet, Ethernet, InfiniBand, ...

Coordenador de tarefas: Slurm, Torque, ...



Santos Dumont

558 nós com 2 CPU Intel Xeon E5-2695v2 e 64GB de DRAM;

198 nós com GPUs K40, 2 x CPU Intel Xeon E5-2695v2 e 64GB de DRAM;

1 nó com 16 x CPU Intel Xeon e 6TB de DRAM;

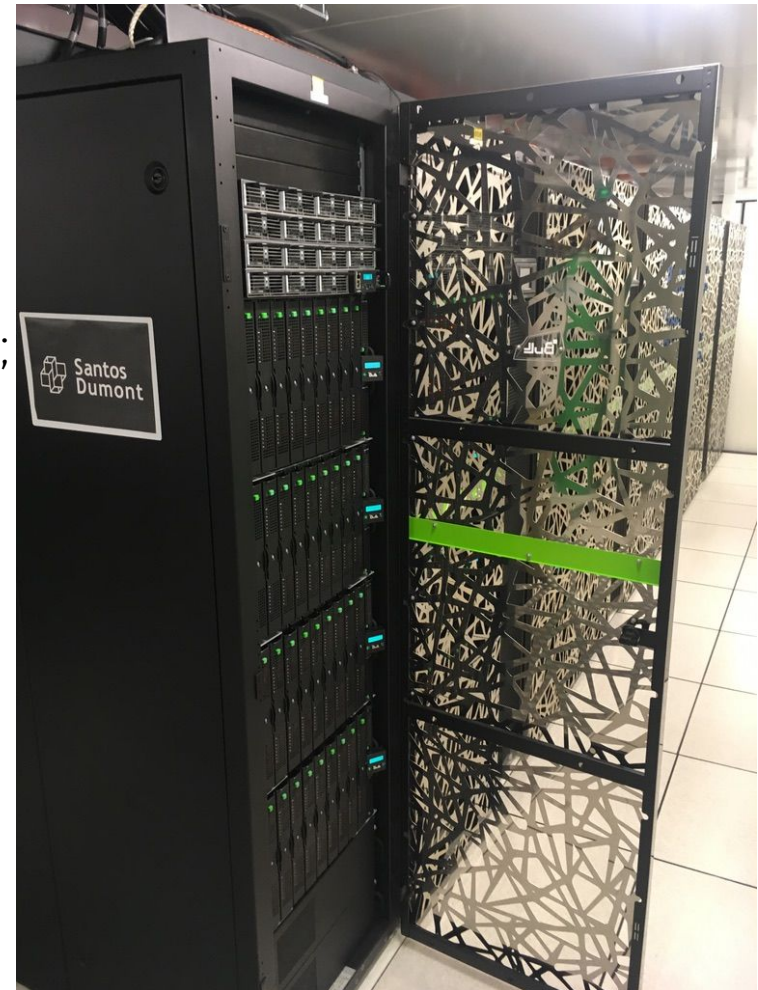
246 nós com 2x Intel Xeon 6252 e 384GB de DRAM;

36 nós com 2x Intel Xeon 6252 e 768 GB de DRAM;

94 nós com 2x Intel Xeon 6252, 4x NVIDIA V100 e 384GB de DRAM;

1 nó com 2x Intel Xeon 6148, 8x NVIDIA V100-16GB e 384GB de DRAM.

sdumont.Incc.br



Santos Dumont



Multiprocessadores

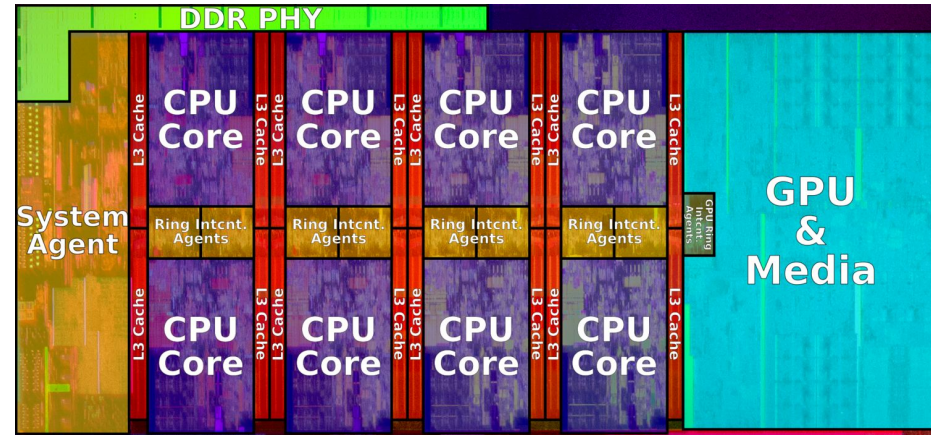
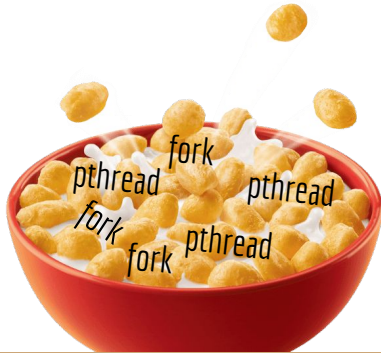
Em um sistema multiprocessado, **cada processador pode executar uma tarefa independente.**

Se você escrever um programa sequencial, ele vai usar somente 1 dos n processadores disponíveis.

Saber criar programas que executam em paralelo não é opcional.

Coma *Pthreads* e *forks* no café da manhã.

Adicione cereais e leite para uma dieta balanceada.

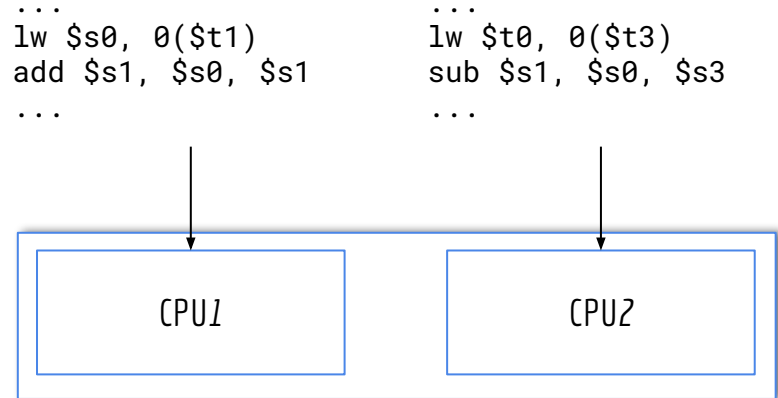


MIMD

Em um sistema multiprocessado, **cada processador pode executar uma tarefa independente.**

São processadores que operam com múltiplas instruções simultaneamente, cada instrução operando em dados diferentes.

MIMD - Multiple Instruction Multiple Data.



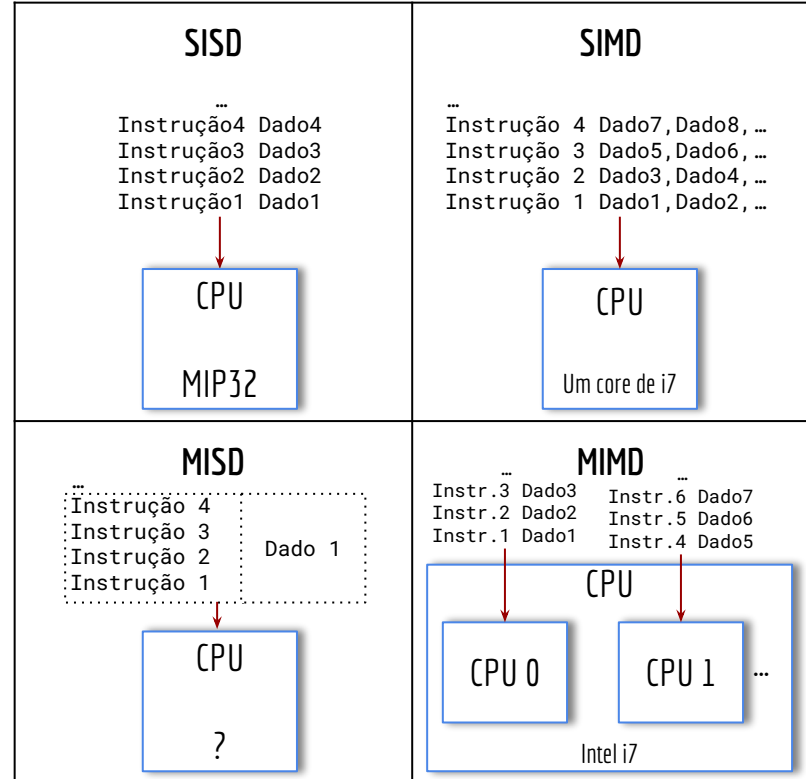
Taxonomia de Flynn

Único Dado.
Single Data - SD.

Múltiplos Dados.
Multiple Data - MD.

Única Instrução.
Single Instruction - SI.

Múltiplas Instruções.
Multiple Instruction - MI.



SIMD

Relembrando: **SIMD** - Single instruction Multiple Data.

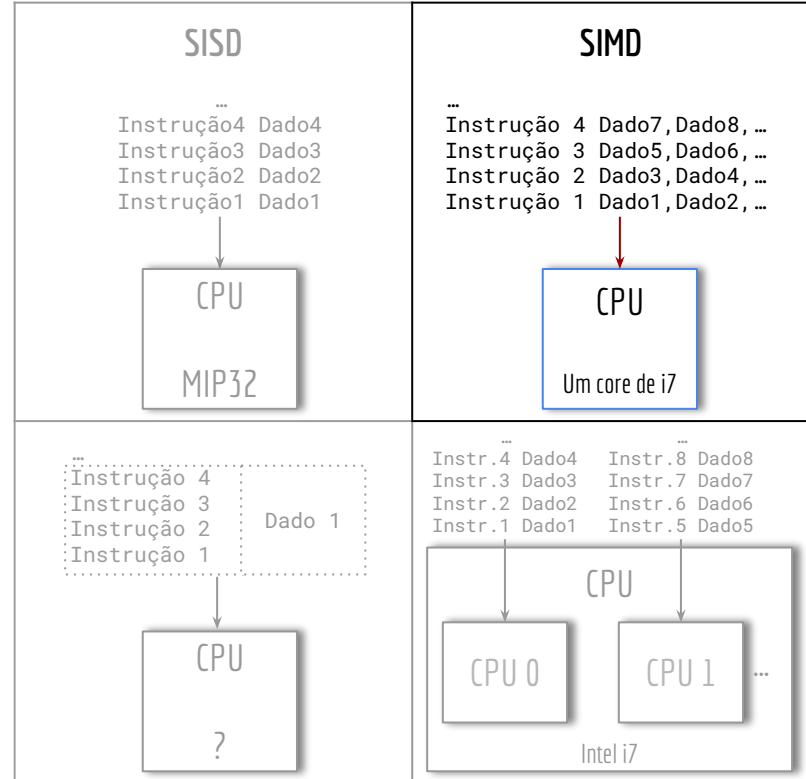
Uma instrução operando em múltiplos dados

Única Instrução.
Single Instruction - SI.

Múltiplas Instruções.
Multiple Instruction - MI.

Único Dado.
Single Data - **SD.**

Múltiplos Dados.
Multiple Data - **MD.**



SIMD

Vamos discutir algumas ideias sobre como adicionar capacidades SIMD no processador MIPS32.

Considere que é comum carregar múltiplos endereços de memória nos registradores, somar um imediato, e depois armazenar os resultados.

Isso realmente é comum quando fazemos operações em vetores ou matrizes.

SIMD

Um exemplo dessas operações em 4 endereços consecutivos de memória.

Como podemos criar instruções que operam com os dados de 4 em 4?

Instruções que:

- Carregam os dados da memória de 4 em 4;

- Adicionam um imediato em 4 registradores;

- Armazenam os dados de 4 registradores na memória.

Como pode ser o formato dessas instruções?

```
lw $s0, 0($t0)
lw $s1, 4($t0)
lw $s2, 8($t0)
lw $s3, 12($t0)
addi $s0, $s0, 1
addi $s1, $s1, 1
addi $s2, $s2, 1
addi $s3, $s3, 1
sw $s0, 0($t0)
sw $s1, 4($t0)
sw $s2, 8($t0)
sw $s3, 12($t0)
```

SIMD - Primeira Ideia

Será adicionado um **s** no final dos mnemônicos das instruções para indicar que são SIMD.

As instruções podem ficar na forma:

```
lws REG1, REG2, REG3, REG4, REG_BASE, DESLOCAMENTO  
    REG1 = MEMORIA[BASE+DESLOCAMENTO+0]  
    REG2 = MEMORIA[BASE+DESLOCAMENTO+4]  
    ...
```

```
addis REG1, REG2, REG3, REG4, REG5, REG6, REG7, REG8, IMEDIATO  
    REG1 = REG2 + IMEDIATO  
    REG3 = REG4 + IMEDIATO  
    ...
```

```
sws REG1, REG2, REG3, REG4, REG_BASE, DESLOCAMENTO  
    MEMORIA[BASE+DESLOCAMENTO+0] = REG1  
    MEMORIA[BASE+DESLOCAMENTO+4] = REG2  
    ...
```

Problemas?

`lws REG1, REG2, REG3, REG4, REG_BASE, DESLOCAMENTO`

Problemas com uma instrução desse tipo?

Problemas?

`lws REG1, REG2, REG3, REG4, REG_BASE, DESLOCAMENTO`

Problemas com uma instrução desse tipo?

O tamanho da instrução cresce proporcionalmente com o número de operandos.

OPCODE | REG_BASE | REG1 | REG2 | REG3 | REG4 | IMEDIATO

Adicionando registradores “grandes”

Outra solução é adicionar registradores “grandes” na CPU.

Registradores SIMD.

Exemplo:

Vamos chamar de registradores `xmm0`, `xmm1`, ... `xmm7`.

Cada registrador com capacidade para 128 bits.

Agora a operação `lws xmm0, 0($t0)` carrega 128 bits (16 bytes) a partir de `$t0`

OPCODE | REG_BASE | REG_SIMD | IMEDIATO

Registradores SIMD

Exemplo utilizando os registradores de 128 bits.

Carregar 128 bits para o registrador.

```
lws xmm0, 0($t0)
```

Registradores SIMD

Exemplo utilizando os registradores de 128 bits.

Carregar 128 bits para o registrador.

```
lws xmm0, 0($t0)
```

Realizar uma soma.

```
addis xmm0, xmm0, 10
```

Como o `addis` faz os cálculos para inteiros, ele pode:

Pegar os primeiros 32 bits de `xmm0`, somar com 10, e armazenar nos primeiros 32 bits de `xmm0`;

Os próximos 32 bits de `xmm0` são somados com 10, e armazenados nos próximos 32 bits de `xmm0`;

...

A instrução segmenta o registrador e guarda cada resultado “em pedaços” no registrador.

Registradores SIMD

Exemplo utilizando os registradores de 128 bits.

Carregar 128 bits para o registrador.

```
lws xmm0, 0($t0)
```

Realizar uma soma.

```
addis xmm0, xmm0, 10
```

Como o `addis` faz os cálculos para inteiros, ele pode:

Pegar os primeiros 32 bits de `xmm0`, somar com 10, e armazenar nos primeiros 32 bits de `xmm0`;
Os próximos 32 bits de `xmm0` são somados com 10, e armazenados nos próximos 32 bits de `xmm0`;

...

A instrução segmenta o registrador e guarda cada resultado “em pedaços” no registrador.

Salvar os 128 bits na memória.

```
sws xmm0, 0($t0)
```


Pergunta

Os registradores convencionais (`$zero`, `$s0`, ... `$ra`) são endereçados de 00000_2 a 11111_2 .

Precisamos de mais endereços para comportar os 8 registradores XMM?

Pergunta

Os registradores convencionais (`$zero`, `$s0`, ... `$ra`) são endereçados de 00000_2 a 11111_2 .

Precisamos de mais endereços para comportar os 8 registradores XMM?

Poderíamos colocar mais bits para endereço, mas isso não é obrigatório.

Os registradores XMM podem usar, por exemplo, os endereços 0000_2 , até 00111_2 .

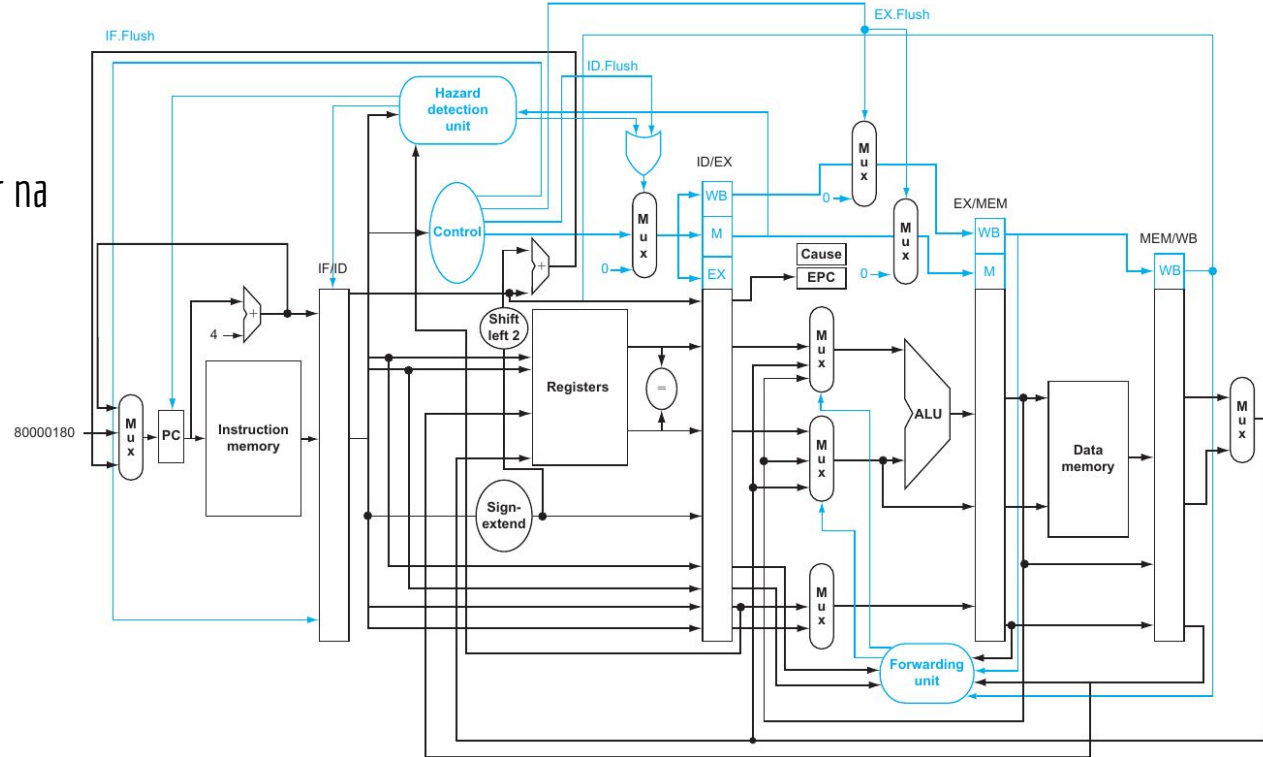
A unidade de controle deve levar em consideração o `OPCODE` para definir se o endereço especificado trata de um registrador convencional ou SIMD.

SIMD

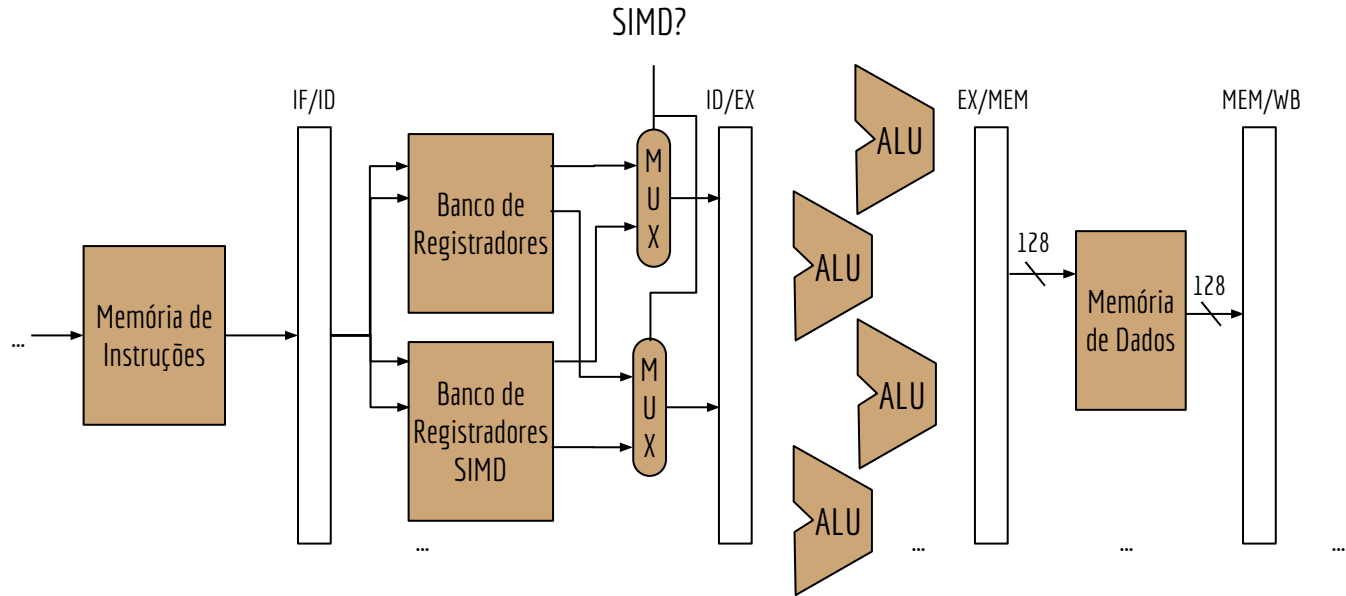
`addis xmm0, xmm0, 10`

Precisamos modificar o hardware.

O que precisamos modificar/adicionar na CPU MIPS32?



SIMD



Para fazer as operações em paralelo, são necessários **no mínimo**:

Adicionar os registradores SIMD no banco de registradores.

Quatro ALUs, para que cada ALU faça o cálculo de um dos inteiros.

Caso contrário criaríamos uma fila na ALU, executando o cálculo de um inteiro por vez

SIMD em x86-64

É essa a estratégia utilizada nas CPUs x86-64 atuais.

Inicialmente isso foi introduzido como o conjunto MMX do Pentium 1 em 1997.



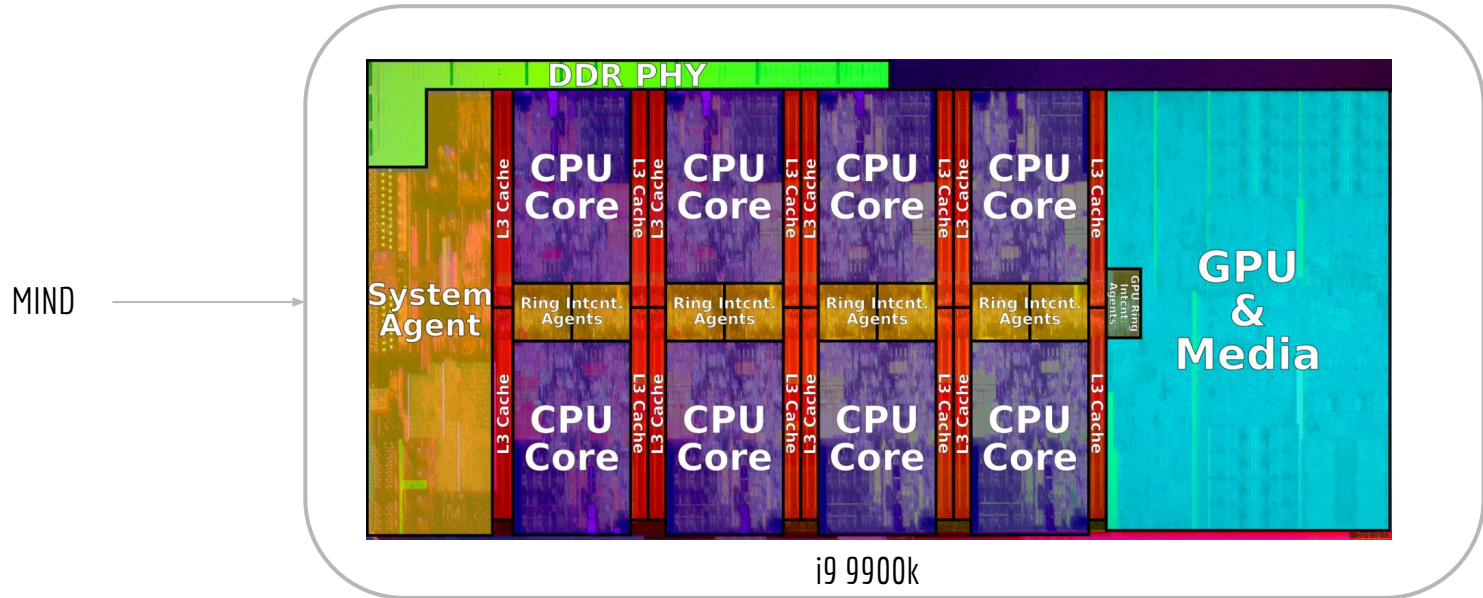
Um dos conjuntos de instruções SIMD especializadas do x86-64 atuais é o SSE.

Streaming SIMD Extensions.

Execute `1scpu` em sua máquina e verifique a versão do SSE que ela é capaz de executar.

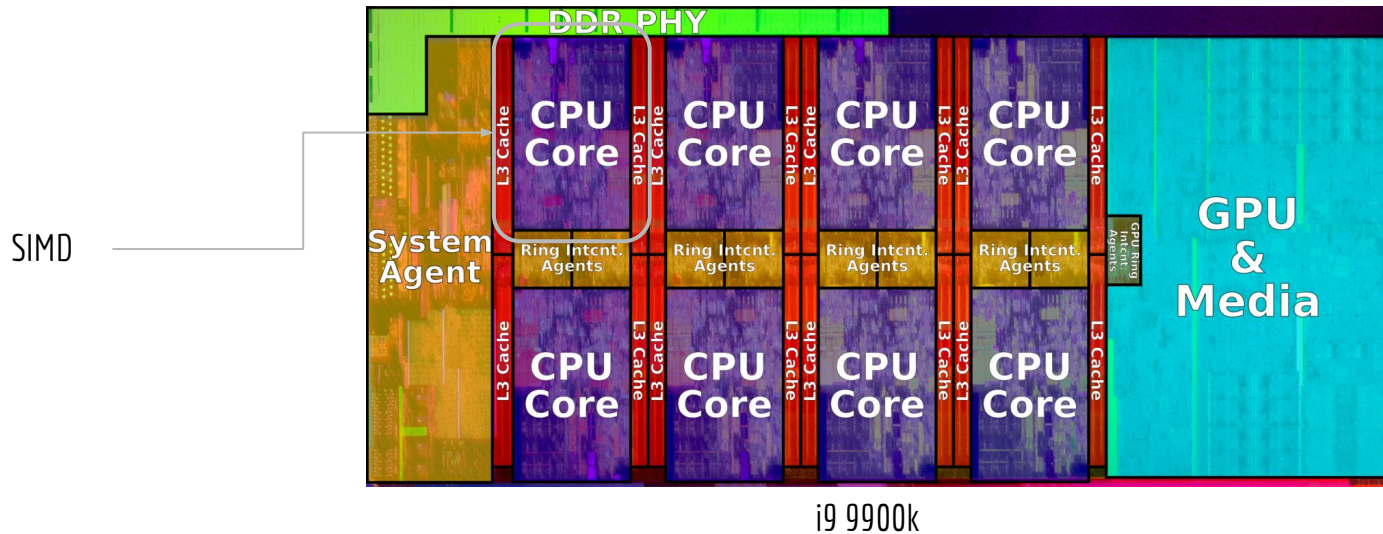
SIMD em x86-64

Então se olharmos para um processador x86-64 atual como um todo, comumente veremos um processador MIND.



SIMD em x86-64

Se olharmos para um único processador (core) de um x86-64, veremos um processador com capacidades SIMD.



SIMD em x86-64

O exercício a seguir foi baseado nas seguintes documentações e tutoriais

software.intel.com/sites/landingpage/IntrinsicsGuide

[docs.microsoft.com/en-us/previous-versions/visualstudio/visual-studio-2010/kcwz153a\(v=vs.100\)](https://docs.microsoft.com/en-us/previous-versions/visualstudio/visual-studio-2010/kcwz153a(v=vs.100))

felix.abecassis.me/2011/09/cpp-getting-started-with-sse

SIMD em x86-64

O SSE do x86-64 funciona utilizando 8 registradores de 128 bits.

xmm0, xmm1, ..., xmm7

Dependendo da operação que executamos nesses registradores, podemos operar em paralelo:

2 doubles, ou;

2 longs, ou;

4 floats, ou;

4 ints ...

Restrição de alinhamento

As operações SSE exigem que o vetor no qual estamos operando comece em um endereço de memória múltiplo de 16.

Restrição de alinhamento de memória.

Restrição de alinhamento

As operações SSE exigem que o vetor no qual estamos operando comece em um endereço de memória múltiplo de 16.

Restrição de alinhamento de memória.

Reclamação aleatória: o SSE força também que os endereços na pilha do x86-64 sejam alinhados em múltiplos de 16 bytes.

Essas e outras idiossincrasias tornam o assembly e código de máquina do x86-64 um verdadeiro inferno.



SIMD em x86-64

Exemplo: alocando um vetor de floats alinhado na memória:

```
float* vetor;  
int ret = posix_memalign((void**)&vetor, 16, TAM_VETOR * sizeof(float));
```

Onde:

`posix_memalign` aloca a memória (`malloc`) de maneira alinhada, e armazena o endereço no ponteiro `vetor`.

A função retorna 0 em caso de sucesso, ou `EINVAL/ENOMEM` em caso de erro.

Constantes definidas em `errno.h`

Em caso de dúvida, digite `man posix_memalign`.

SIMD em x86-64

Vamos criar um programa que calcula a raiz quadrada de todos os elementos de um vetor de floats de 2GiB.

Realizar a operação de raiz quadrada custa caro para a CPU (Método parecido com Newton-Raphson).

Primeiro execute uma versão sem SSE.

Baixe o programa “sse.c” no site da disciplina,

Nessa versão a memória já está sendo alocada com o alinhamento de 16.

Vai ser útil quando adicionar o SSE.

Compile o programa, execute 5x e anote o tempo que o programa levou em cada rodada.

Para compilar utilize o comando `gcc sse.c -o sse -lm -O3`

```
#include <emmintrin.h>
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <math.h>
#include <time.h>

#define TAM_VETOR 536870912//vetor ocupando 2GiB - pode depender da máquina!

void normal(float* a, int tamanho){
    for (int i = 0; i < tamanho; i++, a++){
        *a = sqrt(*a);
    }
}

int main(int argc, char** argv){
    clock_t tempo;
    float* vetor;
    int codigoRetorno = posix_memalign((void*)&vetor, 16, TAM_VETOR * sizeof(float));

    if(TAM_VETOR % (sizeof(__m128) / sizeof(float)) != 0){
        printf("Aceitos somente vetores multiplas de %lu. Para outros valores, compre a versao premium.", sizeof(__m128) / sizeof(float));
    }
    if(codigoRetorno != 0){
        if(codigoRetorno == EINVAL)
            printf("Alinhamento Invalido. O valor de alinhamento deve ser uma potencia de 2.");
        if(codigoRetorno == ENOMEM)
            printf("Impossivel alocar a quantidade de memoria requisitada.");
        return -1;
    }
    for (int i = 0; i < TAM_VETOR; ++i)
        vetor[i] = 3.1415;
    tempo = clock();

    normal(vetor, TAM_VETOR);
    tempo = clock() - tempo;

    printf("Tempo para calcular a raiz quadrada: %lf segundos\n", ((double)tempo)/CLOCKS_PER_SEC);
    //imprimindo o último valor, se não imprimir, as otimizações do compilador podem ignorar os cálculos
    printf("vetor[%d] = %f\n", TAM_VETOR-1, vetor[TAM_VETOR-1]);

    free(vetor);
}
```

Adicionando SSE

Funções e tipos prontos para lidar com SSE em C estão disponíveis na biblioteca `emmintrin.h`.

O tipo `__m128` representa uma variável de 128 bits, que o compilador vai carregar em um dos registradores `xmm`.

Para começar, imprima na tela o tamanho dos tipos das variáveis que serão envolvidas no programa.

```
printf("%lu %lu %lu\n", sizeof(__m128), sizeof(float), sizeof(__m128) / sizeof(float));
```

Execute o programa e veja que um `__m128` ocupa 16 bytes (128 bits).

Note que cabem 4 floats dentro de um `__m128`.

Adicionando SSE

Adicione a seguinte função, que faz o mesmo calculo, porém utilizando SSE.

```
void sse(float* a, int tamanho){
    int numBlocos = tamanho / 4;
    __m128* ptr = (__m128*)a;

    for (int i = 0; i < numBlocos; ++i, ++ptr)
        _mm_store_ps((float*)ptr, _mm_sqrt_ps(*ptr));
}
```

Veja a descrição das funções utilizadas em software.intel.com/sites/landingpage/IntrinsicsGuide

Substitua a chamada da função normal pela função com sse

Para compilar use `gcc sse.c -o sse -lm -O3 -msse2`

Meça o tempo 5x novamente e compare.

Não se confunda!

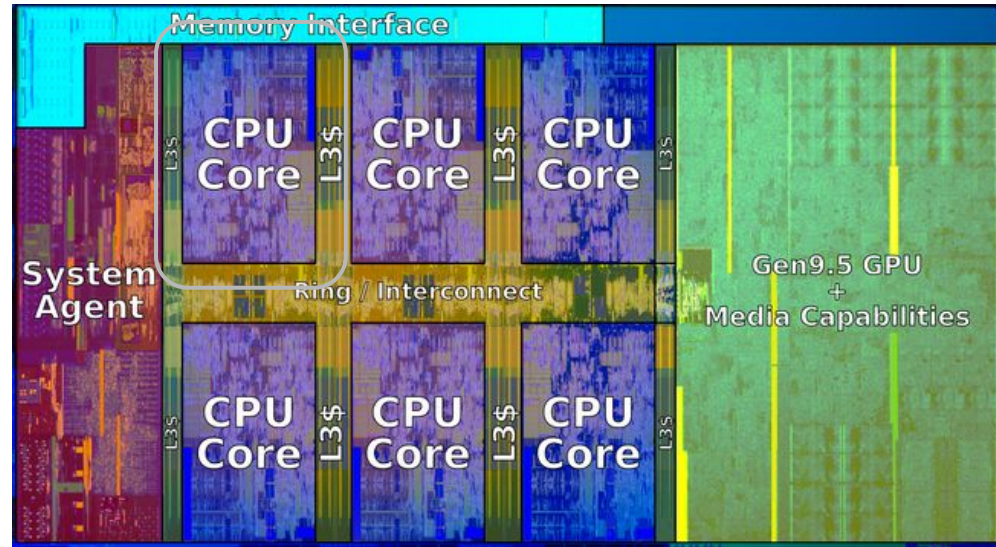
As instruções SIMD são executadas em **uma única CPU**.

Único “núcleo de processamento”.

Se você possui, por exemplo, um i7 com 6 processadores (núcleos), somente 1 deles está operando.

Os demais estão “ociosos”.

Imagine quanto poderíamos otimizar se executarmos o programa em paralelo em múltiplos processadores, levando em consideração o pipeline, instruções SIMD, memória cache, ...



Observações

O exemplo que fizemos é “simples”.

Os dados são totalmente independentes uns dos outros, e a operação é simples.

Independência de dados e um problema **trivialmente paralelizável**.

Problemas do mundo real são um tanto mais complicados.

Observações

Segundo o manual da Intel, no modo 64 bits existem 8 registradores XMM extras, XMM8 a XMM15.

10.2.1 SSE in 64-Bit Mode and Compatibility Mode

In compatibility mode, SSE extensions function like they do in protected mode. In 64-bit mode, eight additional XMM registers are accessible. Registers XMM8-XMM15 are accessed by using REX prefixes. Memory operands are specified using the ModR/M, SIB encoding described in Section 3.7.5.

Exercícios

1. Compile novamente o programa, mas agora utilizando o seguinte comando: `gcc sse.c -lm -O3 -msse2 -S`. O compilador vai gerar o assembly do seu programa. Analise o assembly e veja o uso dos registradores `xmm`.
2. Pesquise as propriedades do processador do seu celular e categorize-o. É um processador MIMD, SISD, SIMD, ...? Faça o exercício olhando o processador do celular como um todo, e depois olhando cada núcleo de processamento de forma separada.
3. Modifique o programa dado em aula para que ele opere com doubles. Compare os tempos com e sem instruções SIMD.
4. Quais os problemas de compatibilidade gerados por instruções SIMD (exemplo: SSE). Quais as dificuldades relativas à compatibilidade se modificarmos uma CPU x86-64 para operar com registradores SSE (registradores `xmm`) de 256 bits? E quais são as vantagens?

Exercícios

5. Entre no site da Steam e procure pelo jogo “Assassin’s Creed Odyssey”. Procure pela CPU nos requisitos mínimos. Baseado na aula de hoje, qual um dos possíveis motivos para o fabricante ser tão específico com os modelos de CPU presentes nos requisitos mínimos?

store.steampowered.com/app/812140/Assassins_Creed_Odyssey

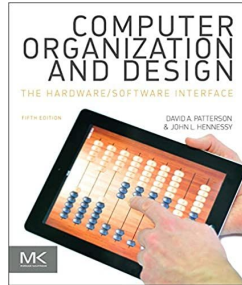


6. Existem registradores SIMD ainda maiores que os XMM. Veja AVX-256 e AVX-512, que usam os registradores YMM e ZMM. Modifique o programa feito em aula para usar AVX-256 e AVX-512 (primeiro verifique se sua máquina aceita essas instruções).

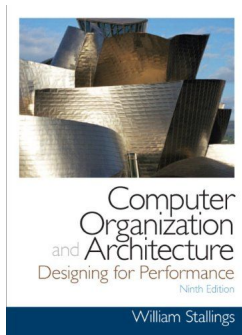
7. Desafio: Faça um programa que multiplica matrizes da maneira “normal”, e um que utiliza SSE. Compare os tempos.

Referências

Patterson, Hennessy .
Arquitetura e Organização de
Computadores: A interface
hardware/software. 2014.



Stallings, W. Organização
de Arquitetura de
Computadores. 10a Ed.
2016.



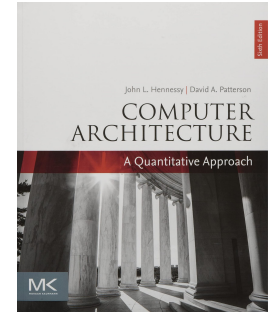
Intel Intrinsic Guide.
www.intel.com/content/www/us/en/docs/intrinsics-guide/index.html

Intel® Intrinsic Guide

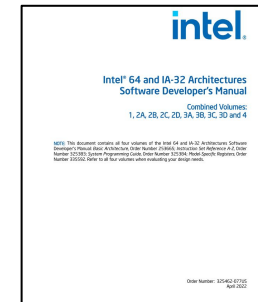
Updated
04/22/2022

Version
3.6.2

Hennessy, Patterson.
Arquitetura de Computadores:
uma abordagem quantitativa.
2019.



Manual de instruções x86-64
Intel. 2022.
www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm.html



Licença

Esta obra está licenciada com uma Licença [Creative Commons Atribuição 4.0 Internacional](https://creativecommons.org/licenses/by/4.0/).

