

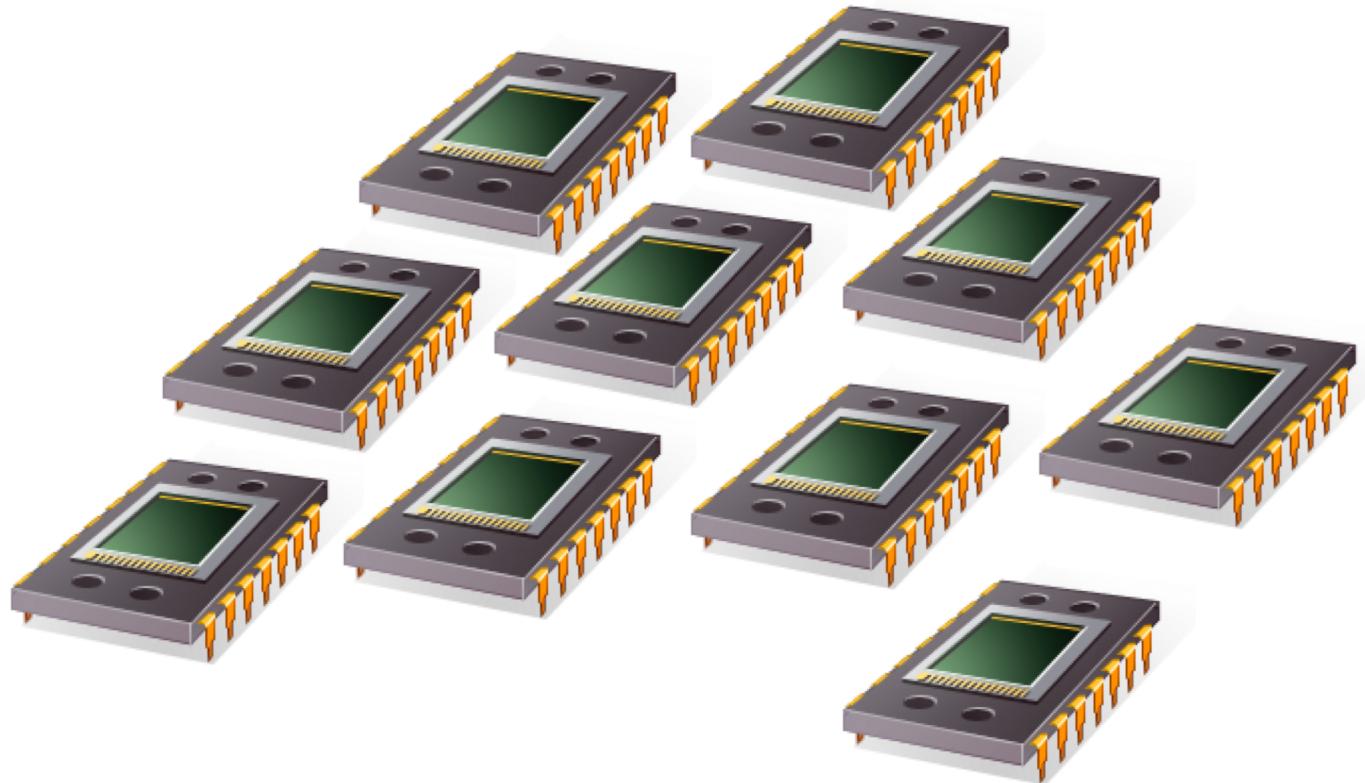
MO644/MC900

Conceitos Básicos

Hardware/Software Paralelos

Prof. Guido Araujo

www.ic.unicamp.br/~guido



Um programador pode escrever código para explorar.

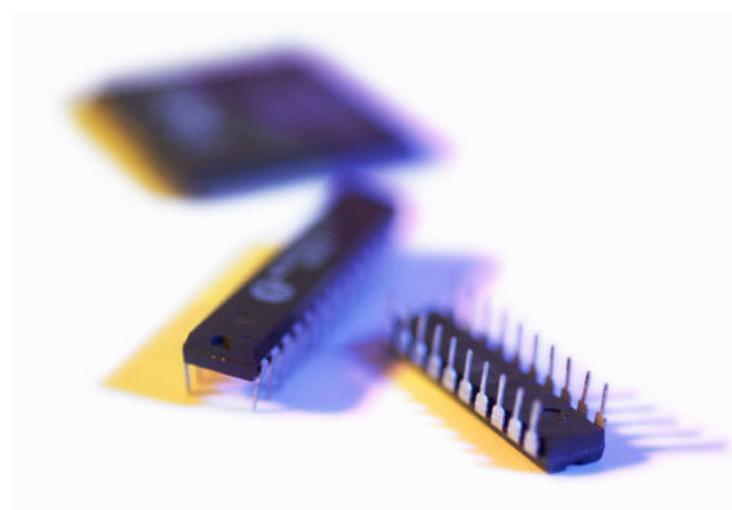
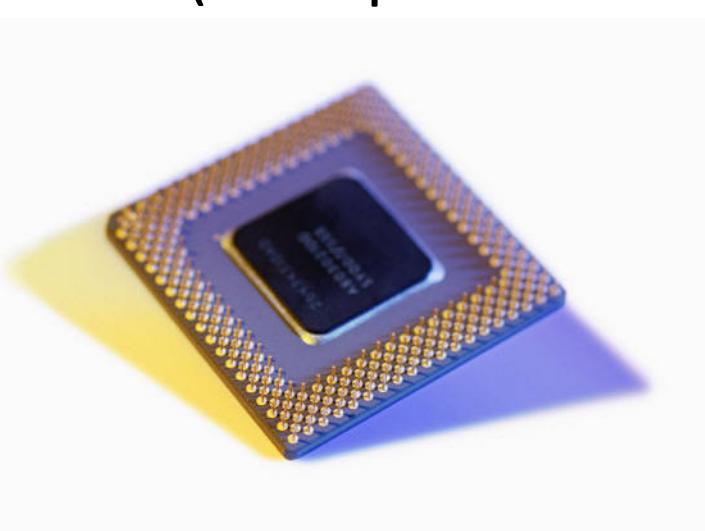
HARDWARE PARALELO

Sistema de Memória Compartilhada (1)

- Um conjunto de processadores autônomos conectados a um sistema de memória através de uma rede de interconexão.
- Cada processador pode acessar cada um dos locais da memória.
- Os processadores usualmente comunicam-se, implicitamente, através do acesso de estruturas de dados compartilhadas.

Sistema de Memória Compartilhada (2)

- Os sistemas de memória compartilhada mais conhecidos possuem um ou mais processadores multicore.
 - (múltiplos núcleos em um único chip)



Sistema de Memória Compartilhada

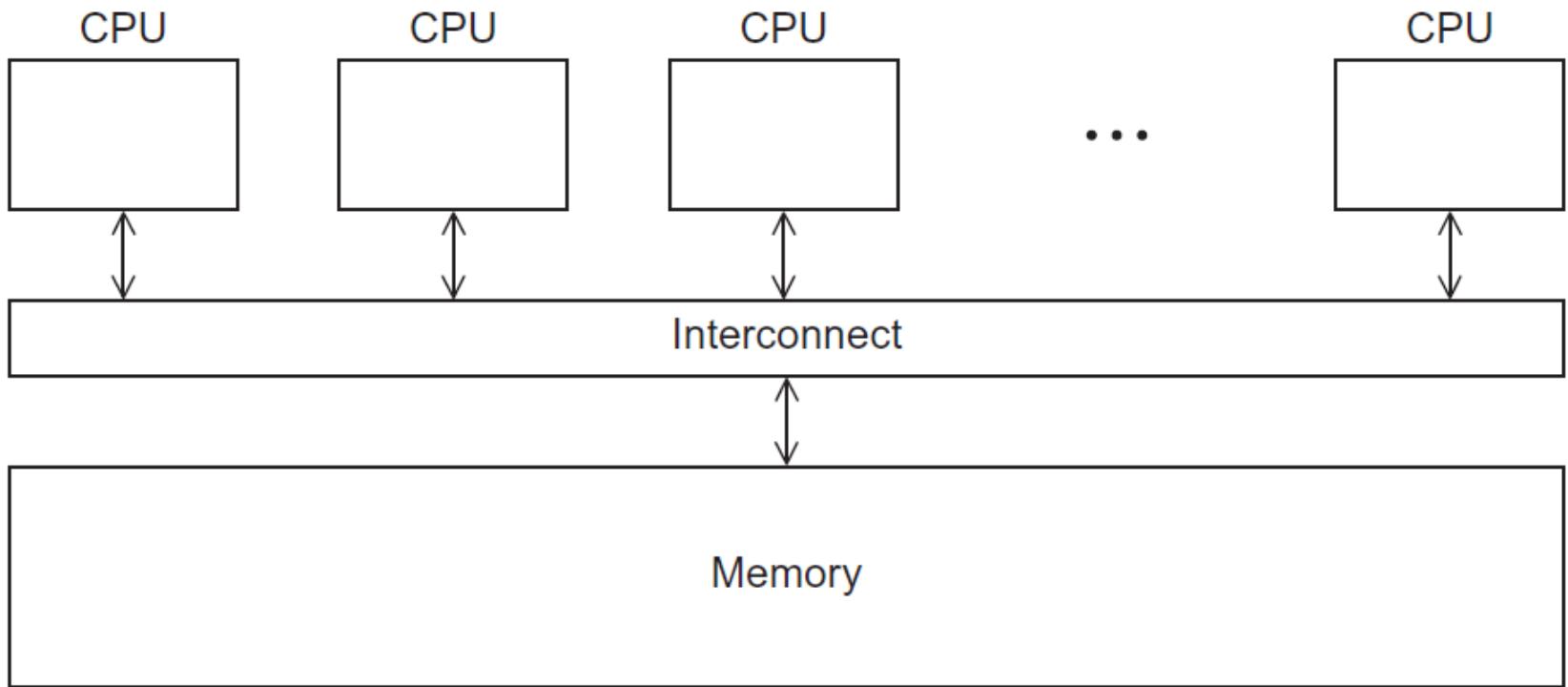
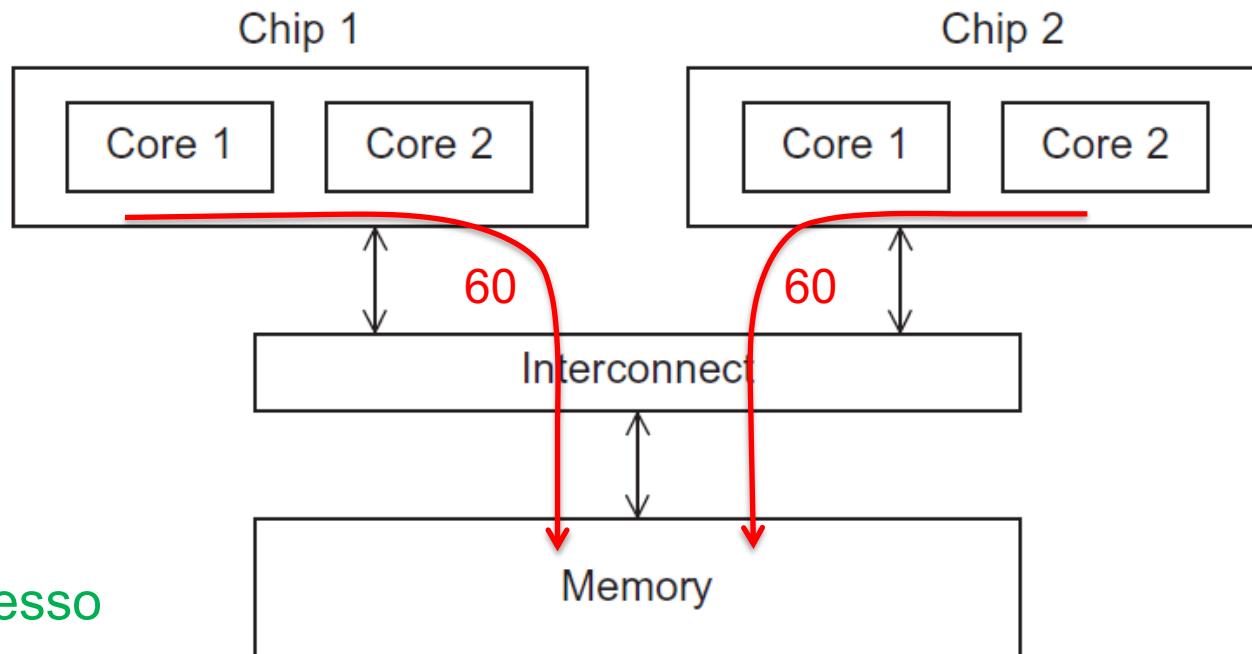


Figura 2.3

UMA multicore system



Tempo de acesso
a todos os locais da
memória é o mesmo
para todos os núcleos.

Figura 2.5

NUMA multicore system

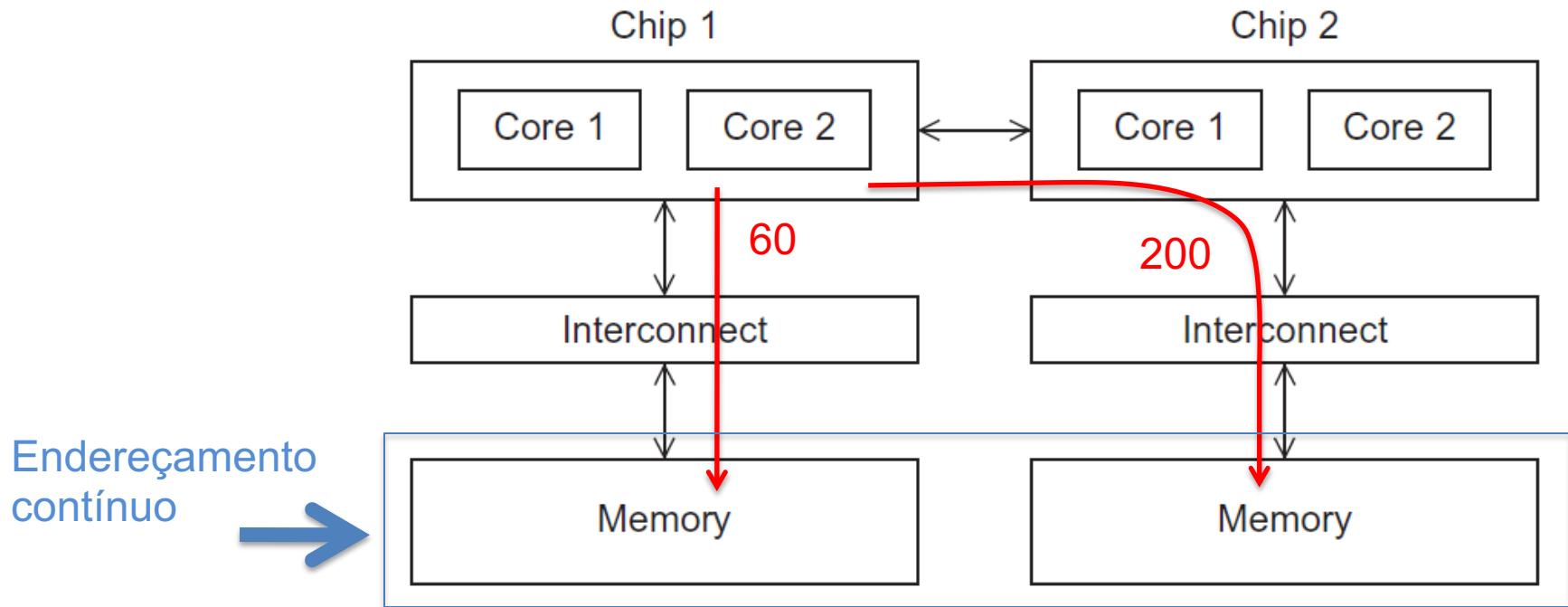


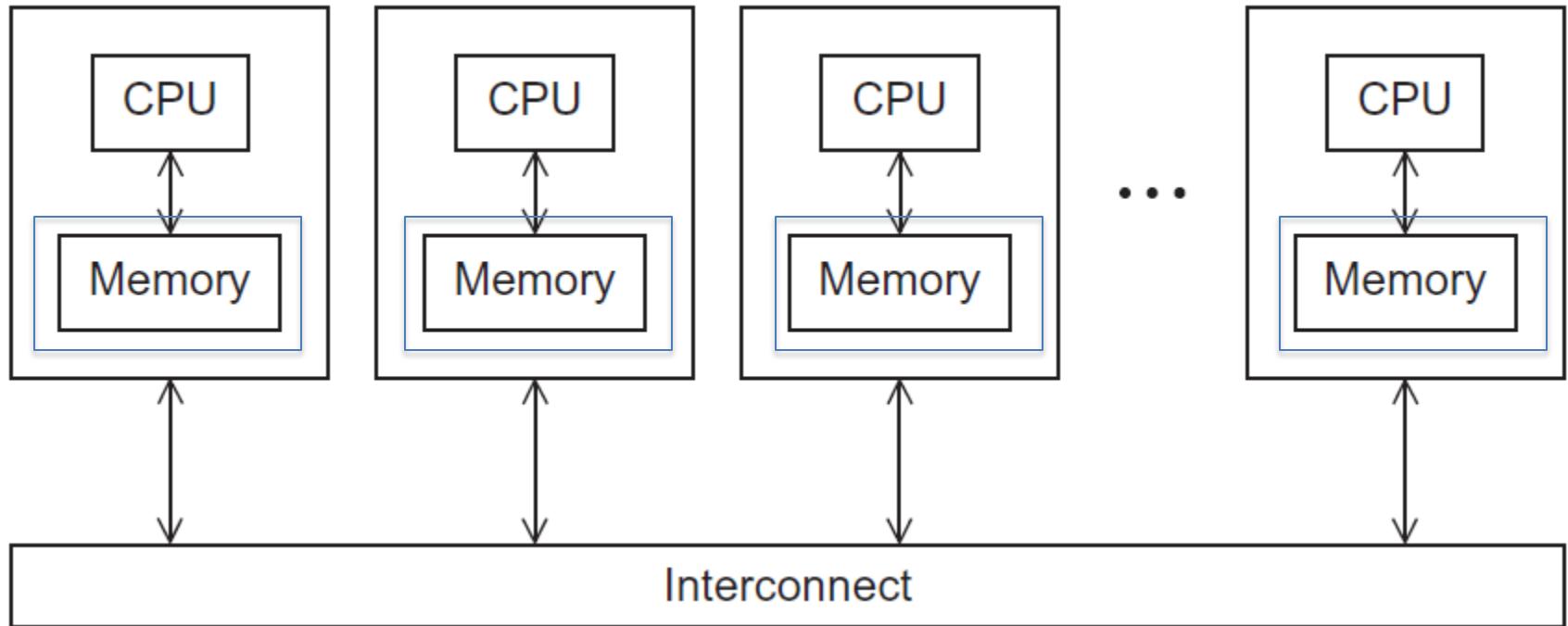
Figura 2.6

Locais da memória aos quais o núcleo está diretamente conectado podem ser acessados mais rapidamente que os locais da memória que são acessados de outro chip.

Sistema de Memória Distribuída

- **Clusters** (mais popular)
 - Uma coleção de sistemas de prateleira.
 - Conectado a uma rede de interconexão de prateleira.
- **Nodes** de um cluster são unidades de computação individuais unidas por uma rede de interconexão.

Sistema de Memória Distribuída



Endereçamento
independente

Figura 2.4

Coerência de cache

- Programadores não tem qualquer controle sobre a cache ou quando ela é atualizada.

Um sistema de memória com dois núcleos e duas caches

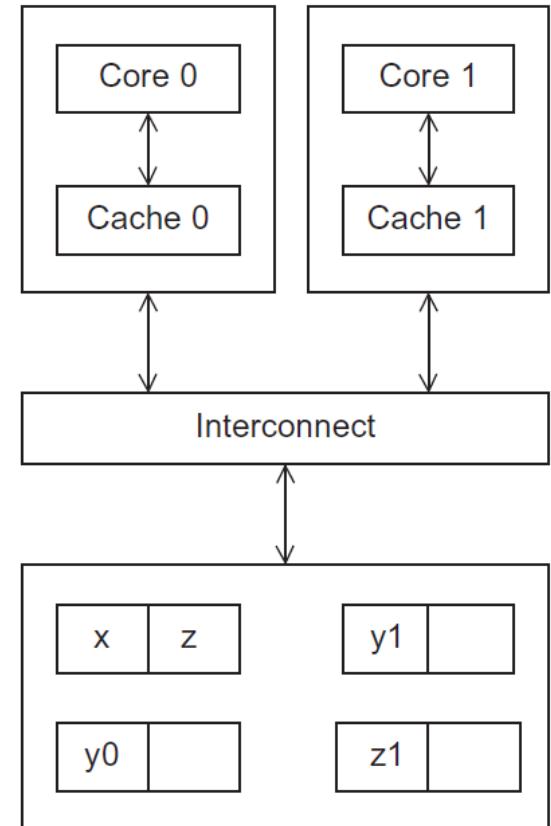
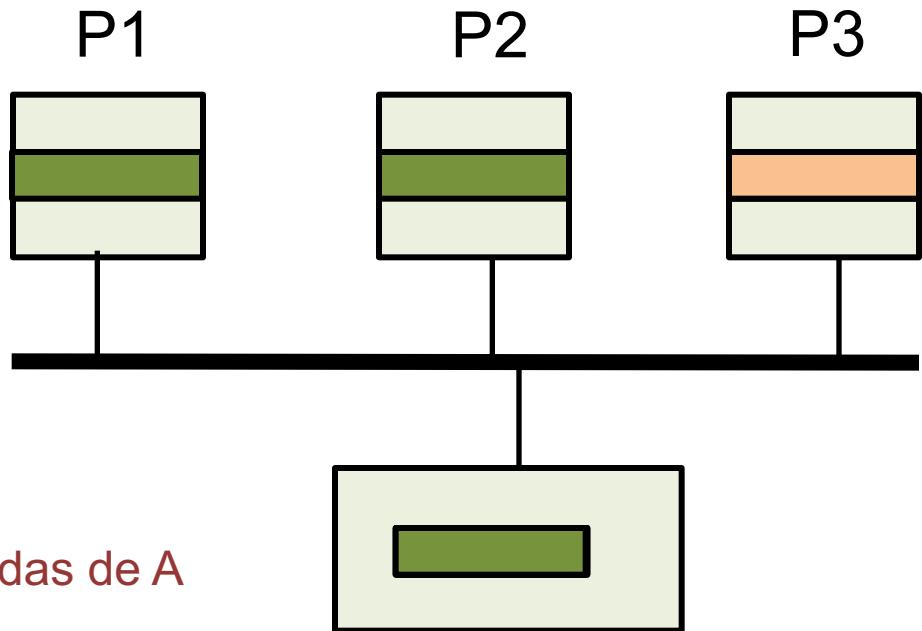


Figura 2.17

Coerência de cache usando (*snooping*)

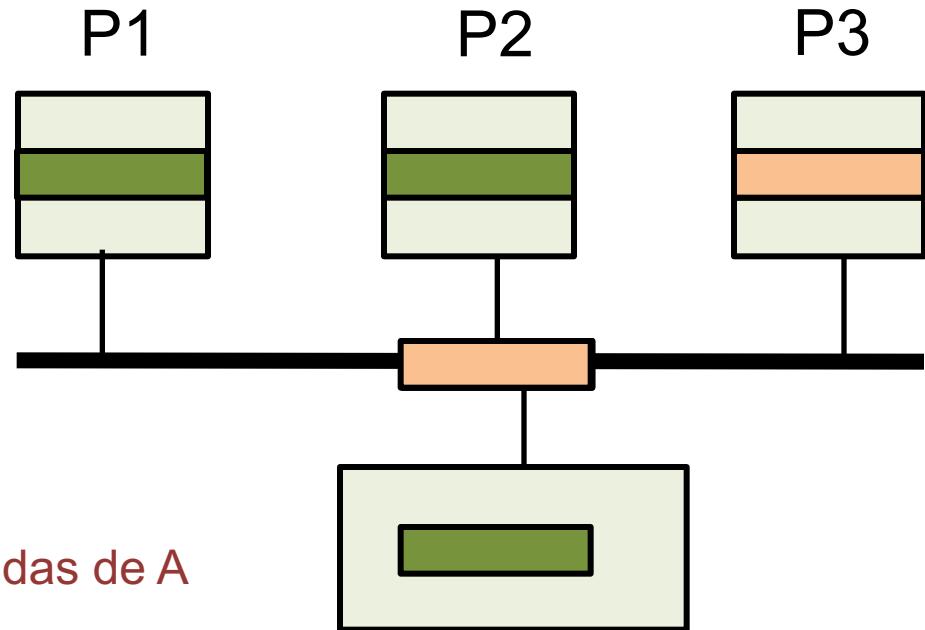
- Os núcleos partilham um barramento.
- Qualquer sinal transmitido no barramento pode ser “visto” por todos os núcleos a ele conectados.
- Quando o núcleo 0 atualiza a cópia de x que armazenada em sua cache ele transmite isto para todos os núcleos no barramento.
- Se o núcleo 1 está “snooping” o barramento, ele vai ver que x foi atualizada e pode marcar sua cópia de x como inválida.

Exemplo



1. P1 lê algum dado A
2. P2 le o mesmo dado A
// P1, P2 e memory tem cópias válidas de A
3. P3 tem um write miss em A
 - 3.1. Uma cópia de A é trazida para a cache do P3
 - 3.2. P3 modifica sua própria cópia de A
// Duas possibilidades: write-update protocol × write-invalidate protocol

Exemplo



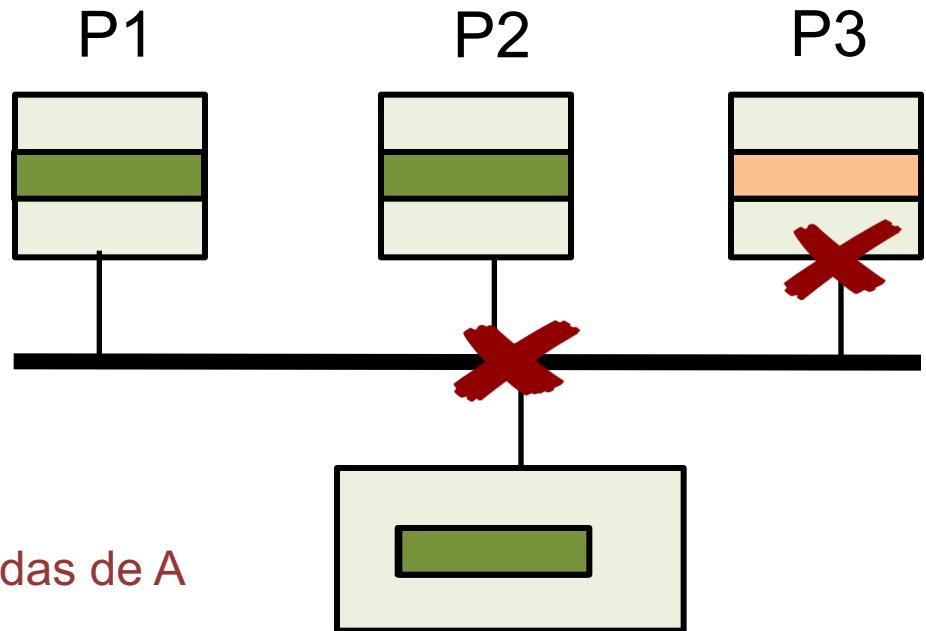
1. P1 le algum dado A
 2. P2 le o mesmo dado A
// P1, P2 e memory tem cópias válidas de A
 3. P3 tem um write miss em A
 - 3.1. Uma cópia de A é trazida para a cache do P3
 - 3.2. P3 modifica sua própria cópia de A
- // Duas possibilidades: *write-update protocol* × *write-invalidate protocol*

// write-update protocol

- // (similar to write-through policy)
- 3.3. P3 manda novo A para todas as caches e para a memória
// A fica válido em todas caches
 - // e na memória

Exemplo

1. P1 le algum dado A
2. P2 le o mesmo dado A
3. P3 tem um write miss em A



3.2. P3 modifica sua propria copia de A

// Duas possibilidades: *write-update protocol* × *write-invalidate protocol*

// write-update protocol

// (similar to write-through policy)

- 3.3. P3 manda novo A para todas as caches e para a memoria
// A fica valido em todas caches
// e na memoria

// write-invalidate protocol

// (similar to write-back policy)

- 3.3. P3 manda mensagem invalidando A para o barramento.

// P3 tem a unica copia valida de A

Exemplo

1. P1 le algum dado A
2. P2 le o mesmo dado A
3. P3 tem um write miss em A

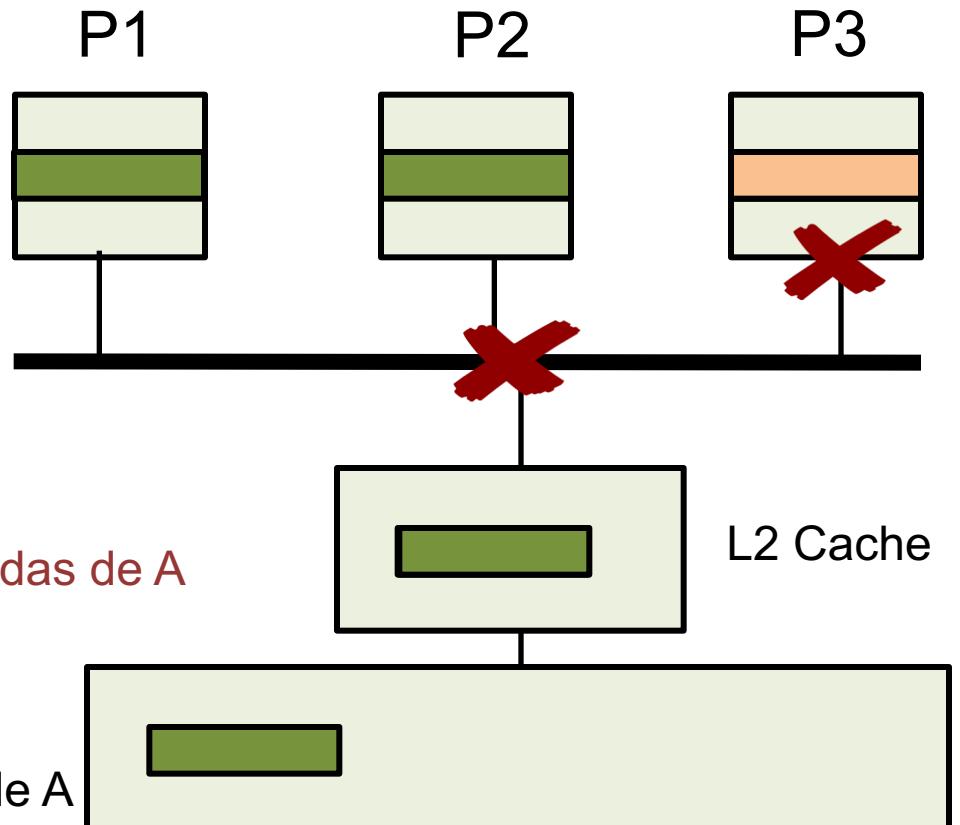
- 3.2. P3 modifica sua propria copia de A

// Duas possibilidades: *write-update protocol* × *write-invalidate protocol*

// write-update protocol

// (similar to write-through policy)

- 3.3. P3 manda novo A para todas as caches e para a memoria
// A fica valido em todas caches
// e na memoria



// write-invalidate protocol

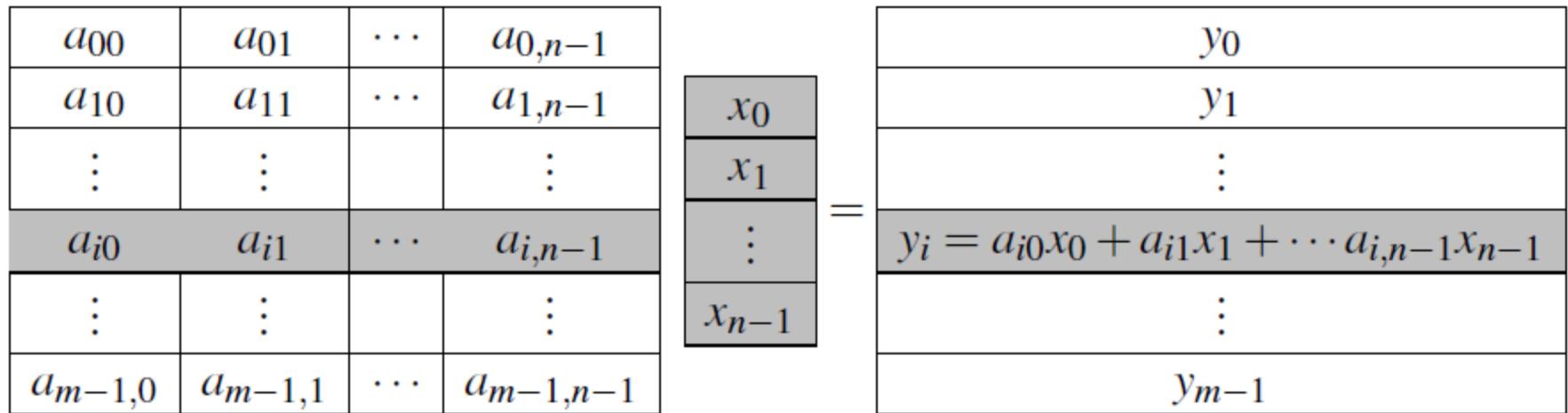
// (similar to write-back policy)

- 3.3. P3 manda mensagem invalidando A para o barramento.
// P3 tem a unica copia valida de A

Impacto da Cache

Multiplicação Matrix-vetor (1)

$$y_i = a_{i0}x_0 + a_{i1}x_1 + \cdots + a_{i,n-1}x_{n-1}$$



```
for (i = 0; i < m; i++) {  
    y[i] = 0.0;  
    for (j = 0; j < n; j++)  
        y[i] += A[i][j]*x[j];  
}
```

Pode-se usar *parallel for* no laço externo?

Impacto da Cache

Multiplicação Matrix-vetor (2)

```
# pragma omp parallel for num_threads(thread_count) \
    default(none) private(i, j) shared(A, x, y, m, n)
for (i = 0; i < m; i++) {
    y[i] = 0.0;
    for (j = 0; j < n; j++)
        y[i] += A[i][j]*x[j];
}
```

Tempos de execução e eficiência na multiplicação matriz-vetor (tempo em segundos)

Threads	Matrix Dimension					
	$8,000,000 \times 8$		8000×8000		$8 \times 8,000,000$	
	Time	Eff.	Time	Eff.	Time	Eff.
1	0.322	1.000	0.264	1.000	0.333	1.000
2	0.219	0.735	0.189	0.698	0.300	0.555
4	0.141	0.571	0.119	0.555	0.303	0.275

Impacto da Cache

Multiplicação Matrix-vetor (3)

E quem são os vilões?

```
# pragma omp parallel for num_threads(thread_count) \
    default(none) private(i, j) shared(A, x, y, m, n)
for (i = 0; i < m; i++) {
    y[i] = 0.0;
    for (j = 0; j < n; j++)
        y[i] += A[i][j]*x[j];
}
```

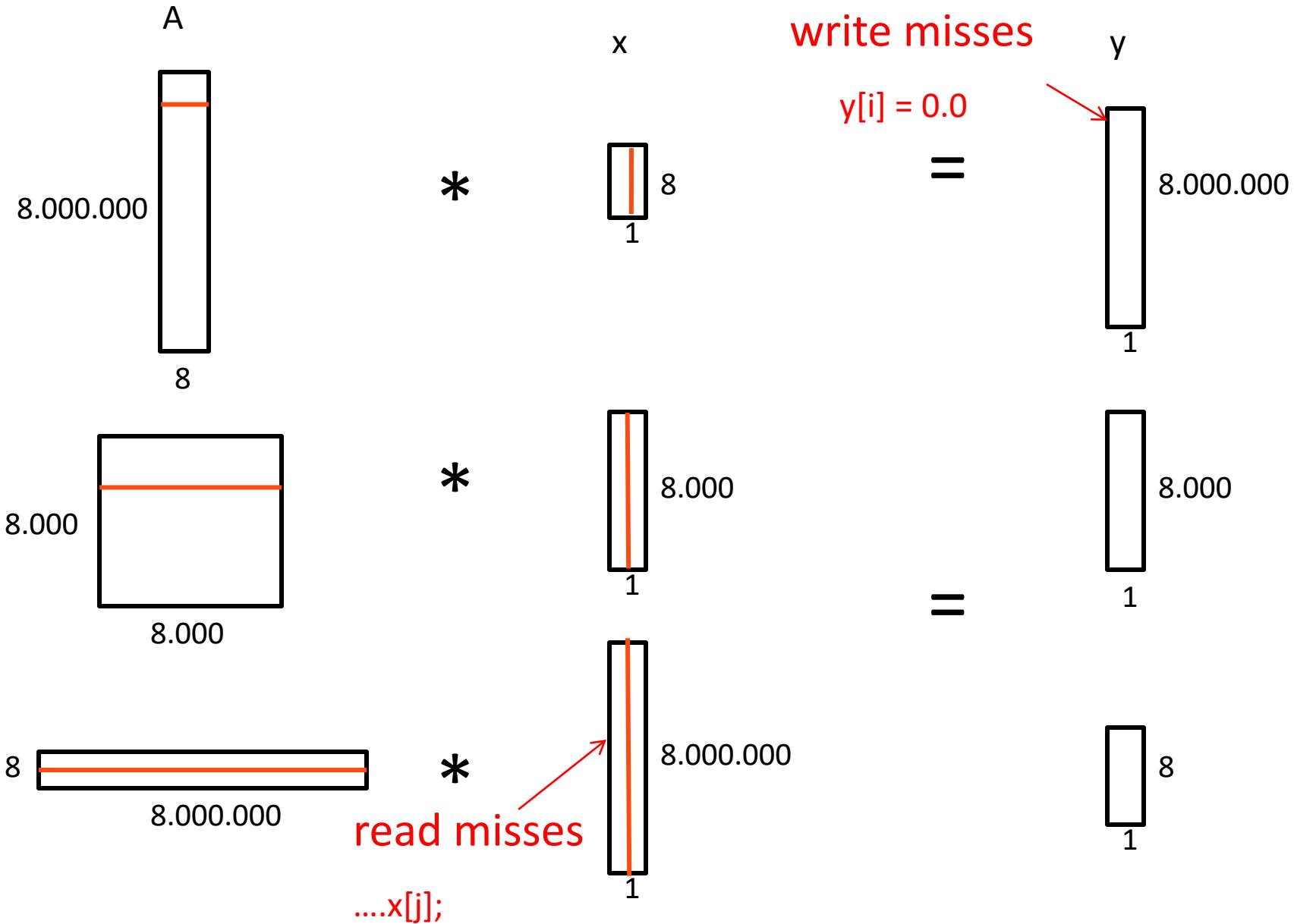
Impacto da Cache

Multiplicação Matrix-vetor (4)

E quem são os vilões?

```
# pragma omp parallel for num_threads(thread_count) \
    default(none) private(i, j) shared(A, x, y, m, n)
for (i = 0; i < m; i++) {
    y[i] = 0.0;
    for (j = 0; j < n; j++)
        y[i] += A[i][j]*x[j];
}
```

Multiplicação Matrix-vetor (5)



False sharing

- Imagine agora que x e z estão na mesma cache line.
- O que ocorre com estas threads?



T0:

```
for (i = 0; i < n; i++)  
    if (i%2 == 0)  
        t0 += 1;
```

T1:

```
for (i = 0; i < n; i++)  
    if (i%2 != 0)  
        t1 += 1;
```

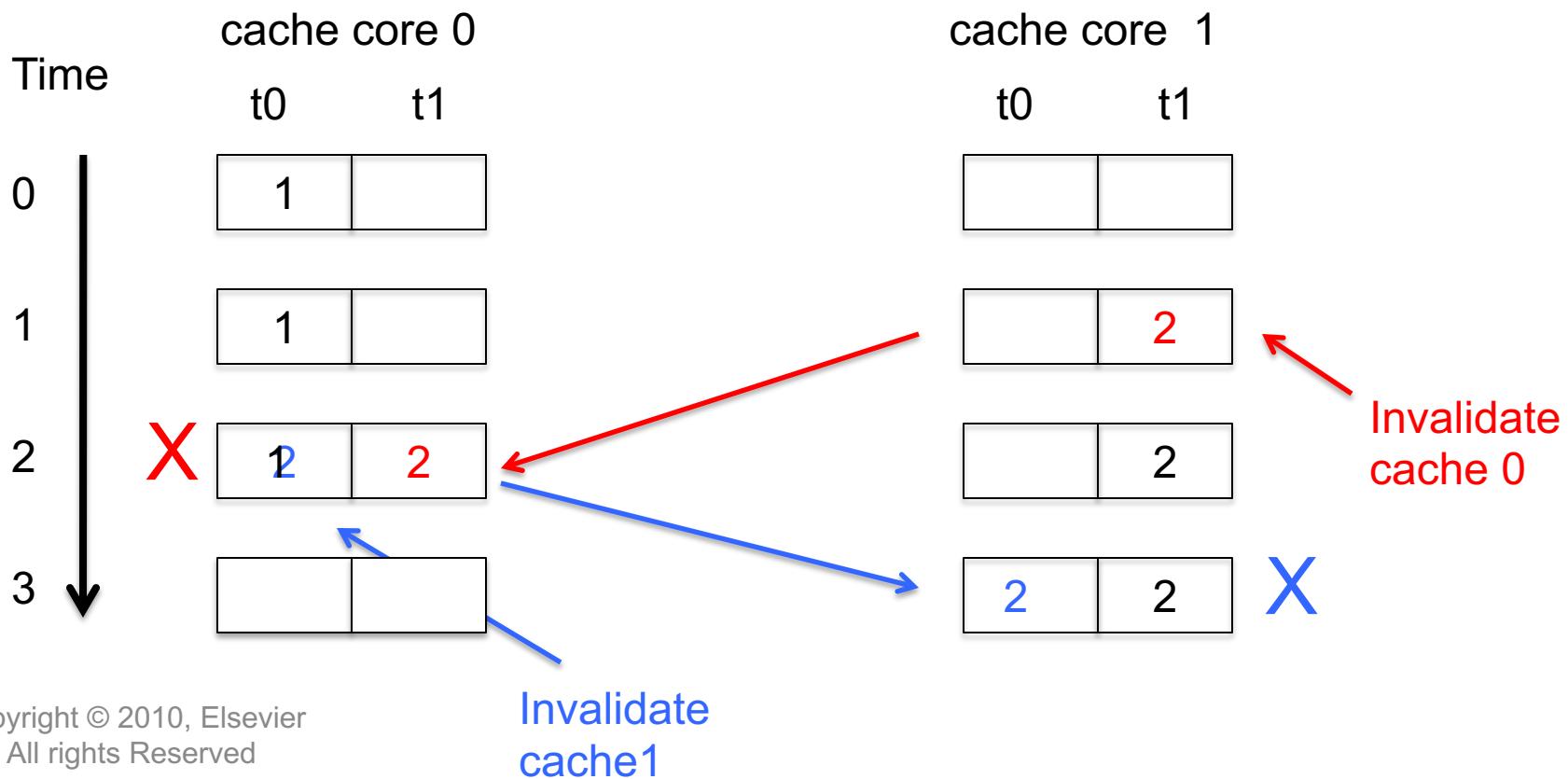
False sharing

T0:

```
for (i = 0; i < n; i++)  
    if (i%2 == 0)  
        t0 += 1;
```

T1:

```
for (i = 0; i < n; i++)  
    if (i%2 != 0)  
        t1 += 1;
```



PERFORMANCE



$$\text{Tempo de transmissão da mensagem} = l + n / b$$

latência (segundos)

comprimento da mensagem (bytes)

largura de banda (bytes por segundo)

Speedup



- Número de núcleos = p
- Tempo de execução serial = T_{serial}
- Tempo de execução paralelo = T_{parallel}

speedup linear

$$T_{\text{parallel}} = T_{\text{serial}} / p$$

$$S = \frac{T_{\text{serial}}}{T_{\text{parallel}}}$$

Eficiência de um programa paralelo

$$E = \frac{S}{p} = \frac{\frac{T_{\text{serial}}}{T_{\text{parallel}}}}{p} = \frac{T_{\text{serial}}}{p \cdot T_{\text{parallel}}}$$

O que E está medindo?

Speedups e eficiências de um programa a paralelo

p	1	2	4	8	16
S	1.0	1.9	3.6	6.5	10.8
$E = S/p$	1.0	0.95	0.90	0.81	0.68

Com quantos cores temos o melhor speed-up?

Qual a solução mais eficiente considerando-se o custo?

Variação de speedups e eficiências de um programa paralelo com o tamanho do programa

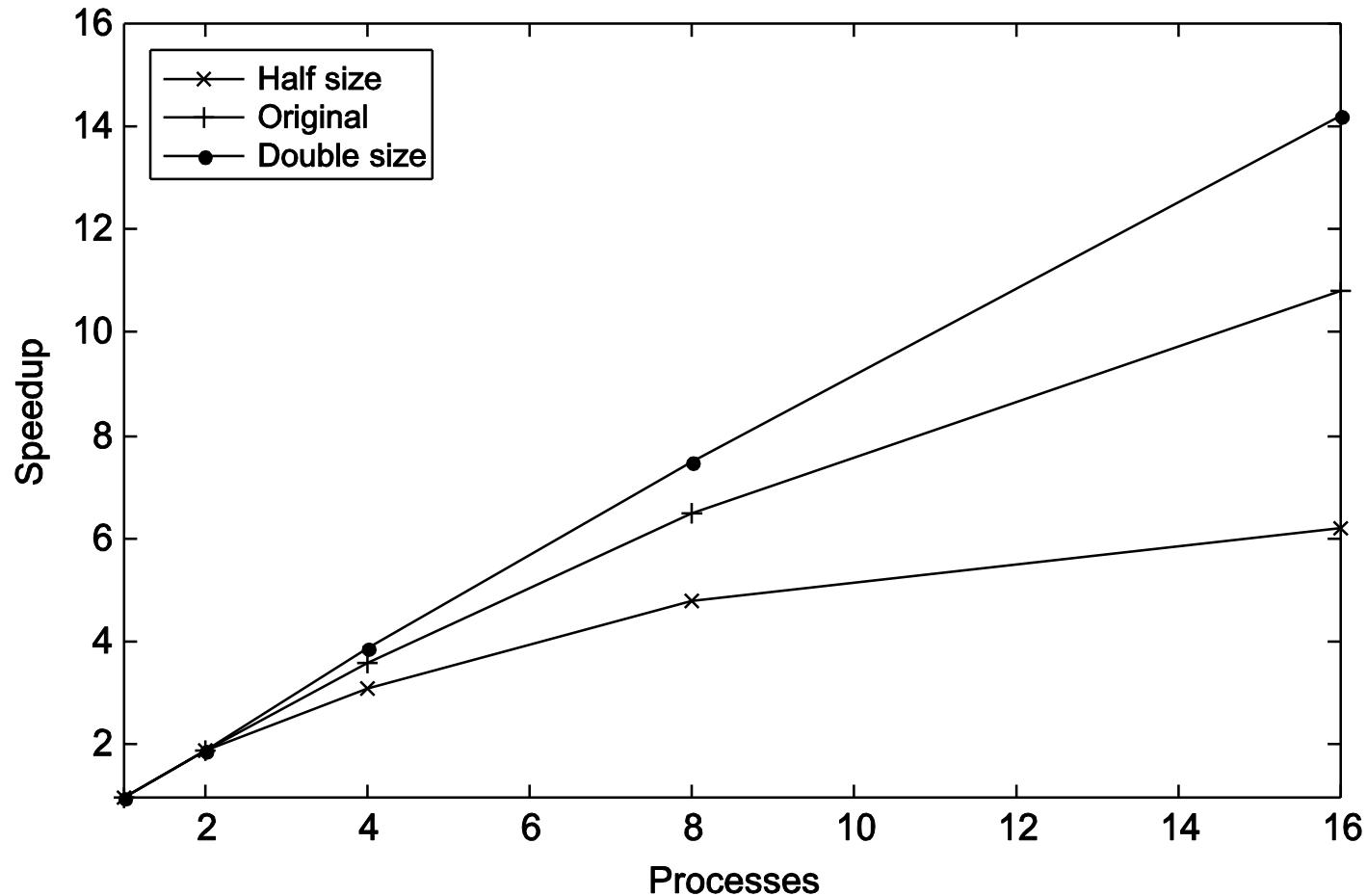
		<i>p</i>	1	2	4	8	16
Half	<i>S</i>	1.0	1.9	3.1	4.8	6.2	
	<i>E</i>	1.0	0.95	0.78	0.60	0.39	
Original	<i>S</i>	1.0	1.9	3.6	6.5	10.8	
	<i>E</i>	1.0	0.95	0.90	0.81	0.68	
Double	<i>S</i>	1.0	1.9	3.9	7.5	14.2	
	<i>E</i>	1.0	0.95	0.98	0.94	0.89	


S Aumenta

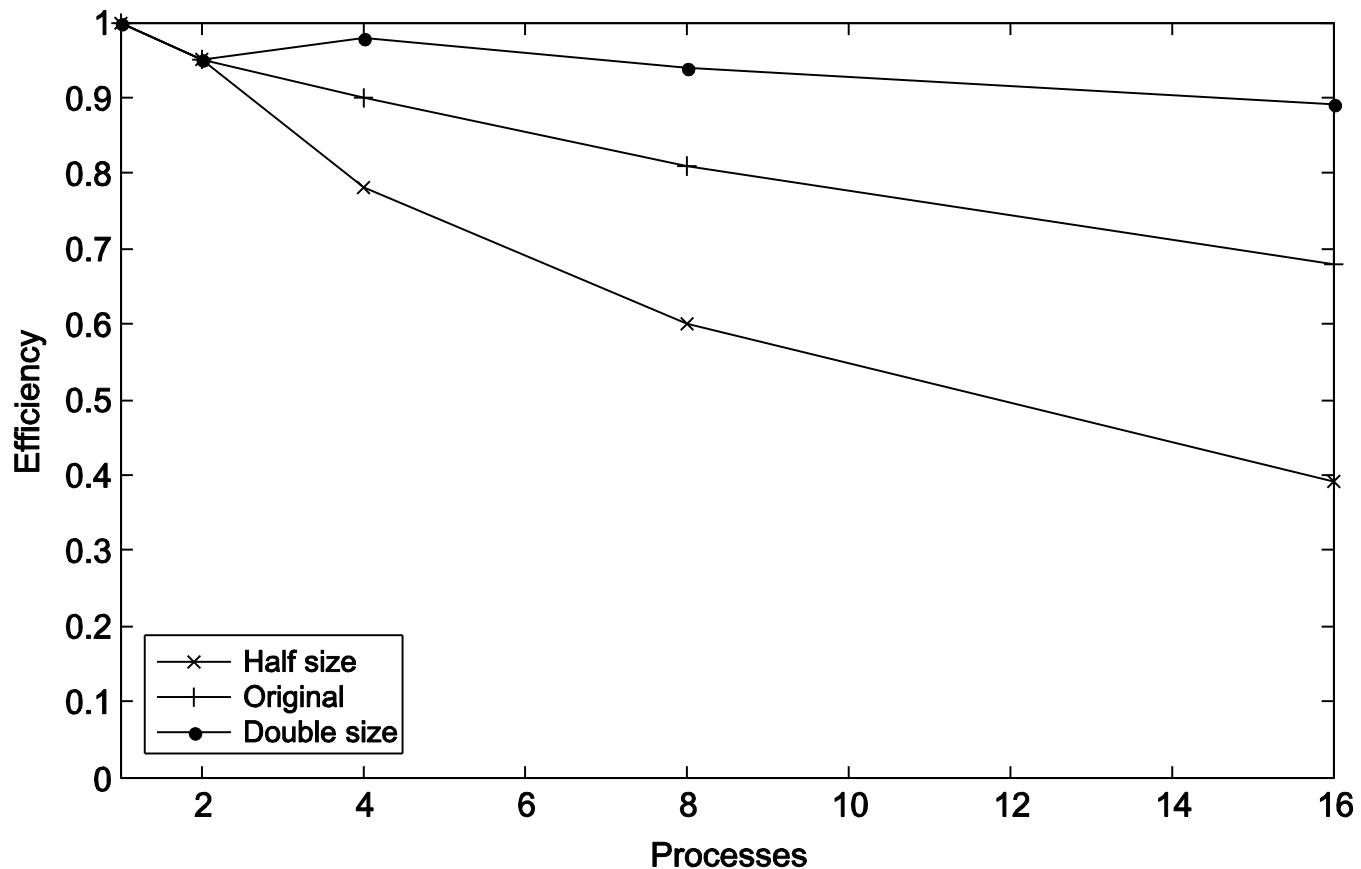

E Diminui

Como *S* e *E* variam com o tamanho do problema?

Speedup



Efficiency



O Mundo dos Porques.....

- Porque S e E aumentam com o tamanho?

Tamanho do problema esconde o overhead!

$$T_{\text{parallel}} = T_{\text{serial}} / p + T_{\text{overhead}}$$

- Por que E vai diminuindo e S saturando com o aumento de p?

Para o mesmo tamanho do problema, se p aumenta pode aumentar a comunicação e portanto o overhead

Lei de Amdahl

- Ao não ser que todo um programa serial possa ser paralelizado, o speedup possível será bem limitado – independente do número de núcleos disponíveis.



Exemplo

- Podemos paralelizar 90% de um programa serial.
- A paralelização é “perfeita” independente do número p de núcleos que usarmos.
- Assuma um programa em que $T_{\text{serial}} = 20 \text{ seconds}$

Exemplo (cont.)

- Tempo de execução da parte não “paralelizável” é

$$0.1 \times T_{\text{serial}} = 2$$

- Tempo de execução da parte paralelizável é:

$$0.9 \times T_{\text{serial}} / p = 18 / p$$

- Tempo de execução total é:

$$T_{\text{parallel}} = 0.9 \times T_{\text{serial}} / p + 0.1 \times T_{\text{serial}} = 18 / p + 2$$

Exemplo (cont.)

- Speed up

$$S = \frac{T_{\text{serial}}}{0.9 \times T_{\text{serial}} / p + 0.1 \times T_{\text{serial}}} = \frac{20}{18 / p + 2}$$

$$S = \frac{1}{f / p + (1 - f)} \quad f = \frac{T_{\text{parallelizable}}}{T_{\text{serial}}}$$

Escalabilidade

- No geral, um problema é *escalável* se ele pode lidar com tamanhos de problema sempre crescentes.
- Se o tamanho do problema é fixo
 - Aumentamos o número de processos/threads e a eficiência permanece a mesma
 - O problema é *fortemente escalável*.
- Se o tamanho do problema aumenta
 - Para manter a eficiência fixa precisamos aumentar o número de processos/threads
 - O problema é *fracamente escalável*.

Medindo tempos

- Qual é o tempo?
- Do começo até o final?
- Qual o trecho de interesse do programa?
- Usar tempo de CPU?
- Usar o tempo do relógio?



Medindo tempos

```
double start, finish;  
 . . .  
 start = Get_current_time();  
 /* Code that we want to time */  
 . . .  
 finish = Get_current_time();  
 printf("The elapsed time = %e seconds\n", finish-start);
```

função
teórica

MPI_Wtime

omp_get_wtime

Taking Timings

```
private double start, finish;  
. . .  
start = Get_current_time();  
/* Code that we want to time */  
. . .  
finish = Get_current_time();  
printf("The elapsed time = %e seconds\n", finish-start);
```

Medindo tempos

```
shared double global_elapsed;
private double my_start, my_finish, my_elapsed;
. . .
/* Synchronize all processes/threads */
Barrier();
my_start = Get_current_time();

/* Code that we want to time */
. . .

my_finish = Get_current_time();
my_elapsed = my_finish - my_start;

/* Find the max across all processes/threads */
global_elapsed = Global_max(my_elapsed);
if (my_rank == 0)
    printf("The elapsed time = %e seconds\n", global_elapsed);
```



PROJETO DE PROGRAMAS PARALELOS

Metodologia de Foster

1. Particionamento

2. Comunicação

3. Agregação

4. Mapeamento

Metodologia de Foster

1. **Particionamento:** divide a computação a ser realizada e os dados a serem operados em tarefas menores.

O foco deve ser em executar tarefas que podem ser executadas em paralelo.

Metodologia de Foster

2. **Comunicação:** determinar que comunicação precisa ser executada entre as tarefas identificadas nos passos anteriores.



Metodologia de Foster

3. **Aglomeração ou agregação:** combine tarefas e comunicações identificada no primeiro passo em tarefas maiores.

Por exemplo, se a tarefa A deve ser executada antes que a tarefa B possa ser executada, faz sentido agregá-las em uma única tarefa composta.

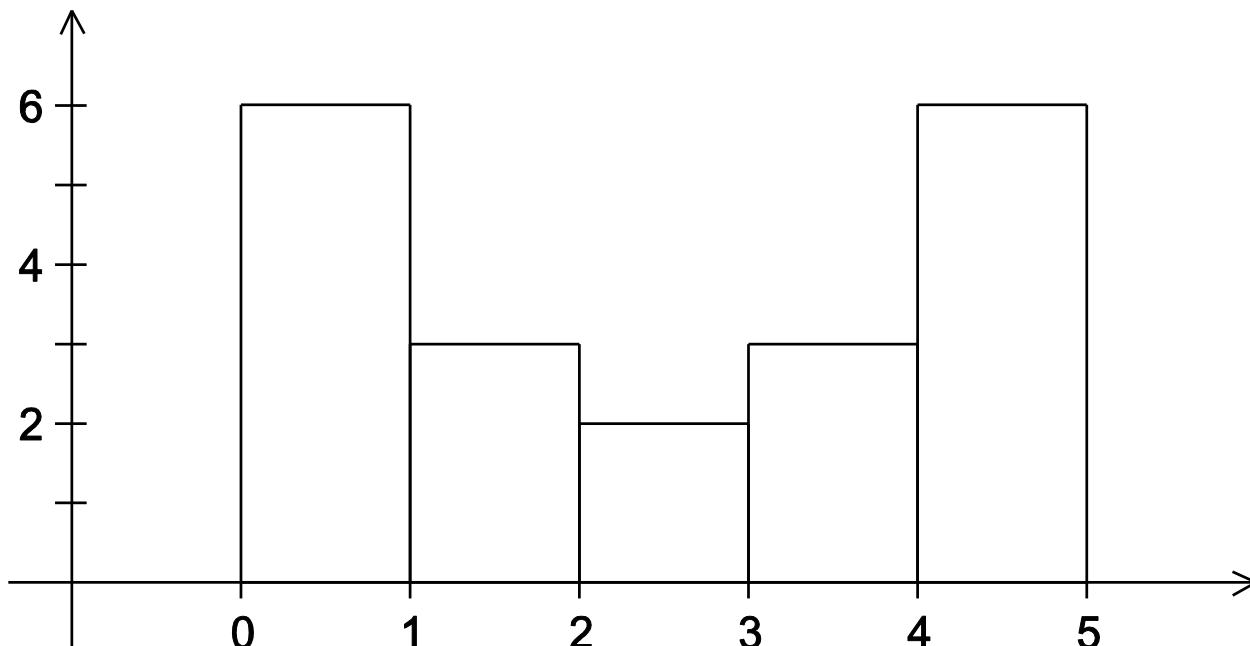
Metodologia de Foster

4. **Mapeamento:** atribua a tarefa composta identificada na etapa anterior aos processos/threads.

Isto deve ser feito de modo a minimizar a comunicação, e cada processo/thread recebe aproximadamente o mesmo volume de trabalho.

Exemplo - histograma

- 1.3, 2.9, 0.4, 0.3, 1.3, 4.4, 1.7, 0.4, 3.2, 0.3, 4.9, 2.4, 3.1, 4.4, 3.9, 0.4, 4.2, 4.5, 4.9, 0.9



Serial program

1. The number of measurements: `data_count`
2. An array of `data_count` floats: `data`
3. The number of bins: `bin_count`
4. The minimum value for the bin containing the smallest values: `min_meas`
5. The maximum value for the bin containing the largest values: `max_meas`
6. Array with `bin_count` floats: `bin_maxes`
7. Array with `bin_count` ints: `bin_counts`

O que é preciso?

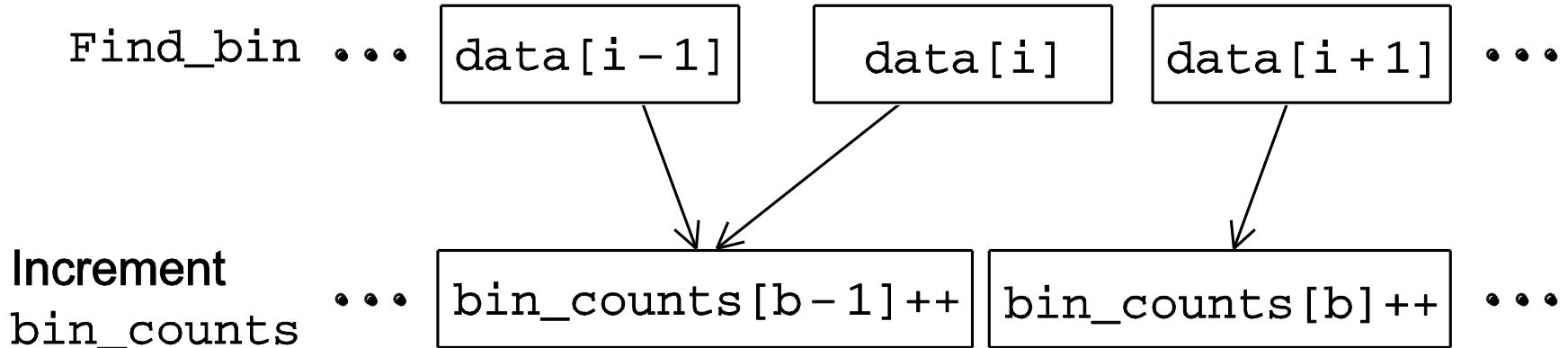
ATENÇÃO: Isto não é um programa!

```
bin_width = (max_meas - min_meas)/bin_count
for (b = 0; b < bin_count; b++)
    bin_maxes[b] = min_meas + bin_width*(b+1);
for (i = 0; i < data_count; i++) {
    bin = Find_bin(data[i], bin_maxes, bin_count, min_meas);
    bin_counts[bin]++;
}
bin_maxes[b-1] <= data[i] < bin_maxes[b]
min_meas <= measurement < bin_maxes[0]
```

Primeiras duas etapas da Metodologia de Foster

Quais seriam estas?

1. Particionamento
2. Comunicação



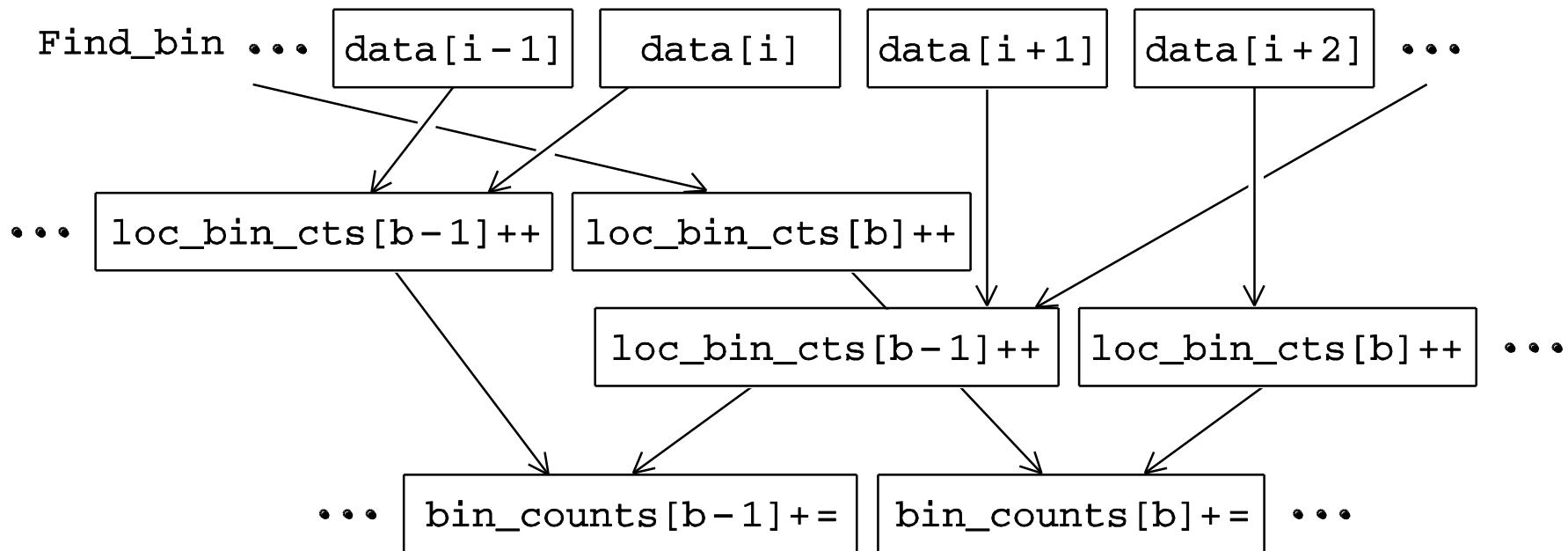
Qual o problema aqui?

Corrida daqueles que vão para o mesmo bin!

Definições alternativas de tarefas e comunicação

Qual a solução?

Criar vetores locais (privados) para os bins e somá-los no final



Somando arrays locais

