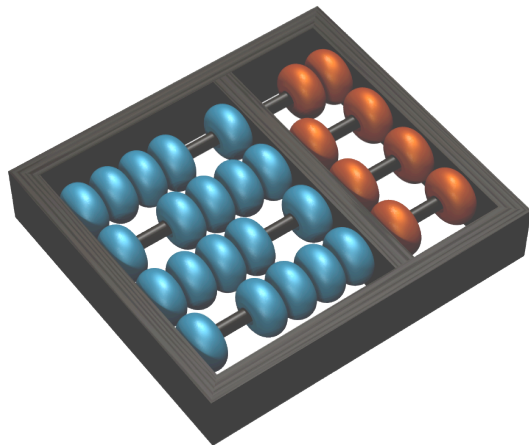


# Técnicas para desenvolvimento e aceleração de códigos científicos



**Prof. Edson Borin**  
edson@ic.unicamp.br  
Instituto de Computação  
UNICAMP

**Minicurso  
LNCC 2014**

---

# Sobre o minicurso

<b>Segunda</b>	<b>Terça</b>	<b>Quarta</b>	<b>Quinta</b>	<b>Sexta</b>
Introdução	Perfilamento - Contagem de tempo	Otimizações simples / compilação	Perfilamento - Detecção de código quente	GDB
Organização de processadores modernos	Otimização de acesso a dados	Bibliotecas otimizadas	SVN + CMake	Valgrind
Introdução ao laboratório		Vetorização de código		

# Agenda

- Perfilamento – Contagem de tempo
- Otimização de acesso a dados
- Atividade de laboratório

# Contagem de Tempo

Várias abordagens:

- `/usr/bin/time ./my_app.bin`
- `gettimeofday()`
- `rdtsc`
- bibliotecas: PAPI, ...

# Contagem de Tempo

Ferramenta time:

```
/usr/bin/time -p ./my_prog.bin  
real 0.62  
user 0.55  
sys 0.07
```

# Contagem de Tempo

`gettimeofday()` – *UNIX-like systems*

```
#include <sys/time.h>
double mysecond() {
    struct timeval tp;
    struct timezone tzp;
    gettimeofday(&tp, &tzp);
    return ((double) tp.tv_sec +
           (double) tp.tv_usec * 1.e-6 );
}
```

# Contagem de Tempo

rdtsc – Instrução em hardware

```
unsigned long long getticks(void)
{
    unsigned a, d;
    asm("cpuid");
    asm volatile("rdtsc" : "=a" (a), "=d" (d));
    return (((ticks)a) | (((ticks)d) << 32));
}
```

# Erros aleatórios

- Estado atual ou atividades paralelas no sistema podem afetar o tempo de execução.
- Solução: executar múltiplas vezes e analisar a distribuição dos resultados.



# Erros aleatórios

## Metodologia utilizada nos resultados apresentados no minicurso:

```
for(i=0; i<10; i++) {  
    clean_caches(); // limpa caches  
    t = mysecond(); // tempo em segundos  
    res = compute();  
    t = mysecond()-t;  
    min_t=MIN(t,min_t);  
}
```

# Limpendo as *caches* de dados

```
#define CACHE_SZ (8*1024*1024)

char buffer[CACHE_SZ];

int clean_caches(void) {
    int i, acc = 0;
    for (i=0; i<CACHE_SZ;i++) {
        acc += buffer[i];
    }
    return acc;
}
```

# Agenda

- Perfilamento – Contagem de tempo
- **Otimização de acesso a dados**
- Atividade de laboratório

# Otimizações de Acesso a Dados

Largura de banda da memória do meu Laptop (Banda):

Memória: 2 x 2 GB (1333 MHz DDR3 SDRAM)

Banda = 1333 MT/s x 64 bits x 2 (canais)

= 10666 MB/s x 2

= 21.3 GB/s

# Otimizações de Acesso a Dados

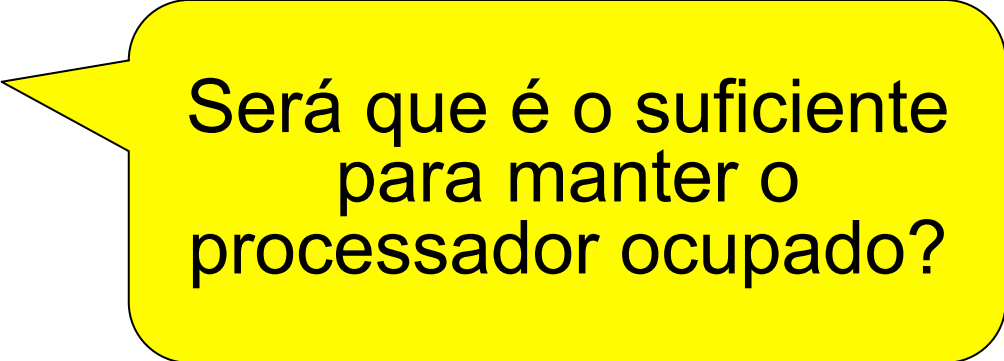
Largura de banda da memória do meu Laptop (Banda):

Memória: 2 x 2 GB (1333 MHz DDR3 SDRAM)

Banda = 1333 MT/s x 64 bits x 2 (canais)

= 10666 MB/s x 2

= 21.3 GB/s



Será que é o suficiente  
para manter o  
processador ocupado?

# Otimizações de Acesso a Dados

Flops: # de operações de ponto-flutuante por segundo

Dados do meu Laptop: (i7-2677M)

- 2 Cores
- 1.8 GHz
- ALUs: 8 operações FP (DP) por ciclo  
4 somas + 4 multiplicações
- Total =  $2 \times 1.8 \times 8 = 28.8$  GFlops

# Otimizações de Acesso a Dados

ALUs:  $14.4 \text{ GFlops} \times 2 \text{ cores} = 28.8 \text{ GFlops}$

Memória:  $21.3 \text{ GB/s}$

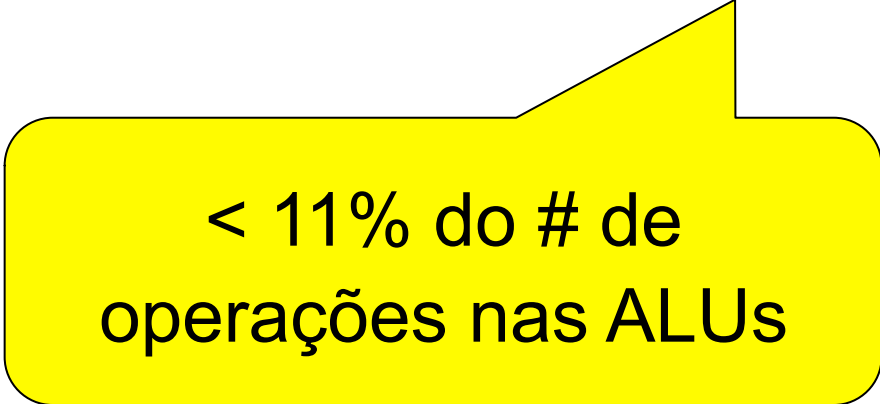
- DP FP = 8 *bytes*, logo Memória =  $2.66 \text{ G FP / s}$

# Otimizações de Acesso a Dados

ALUs: 14.4 GFlops x 2 cores = 28.8 GFlops

Memória: 21.3 GB/s

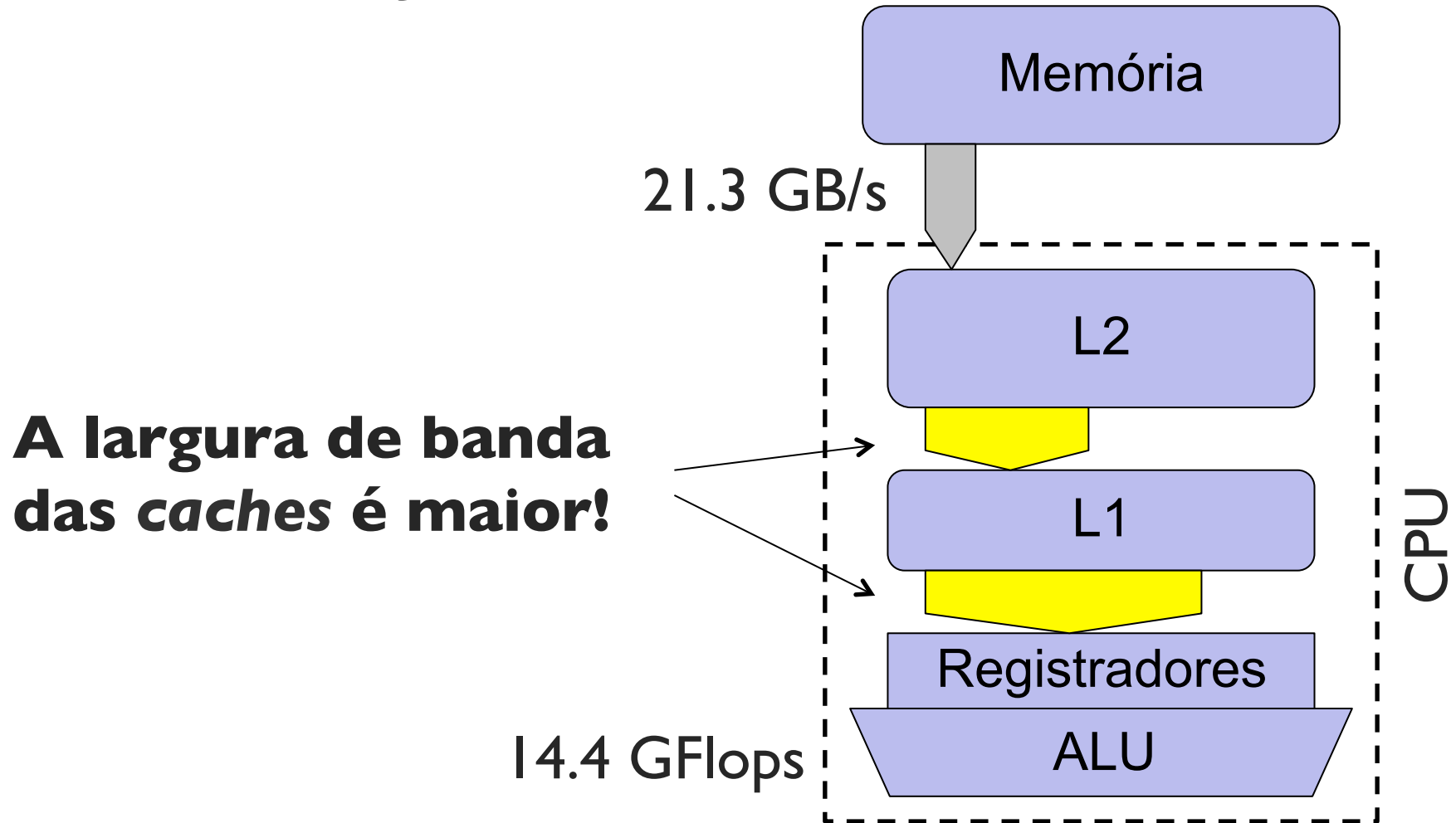
- DP FP = 8 bytes, logo Memória = 2.66 G FP / s



< 11% do # de  
operações nas ALUs



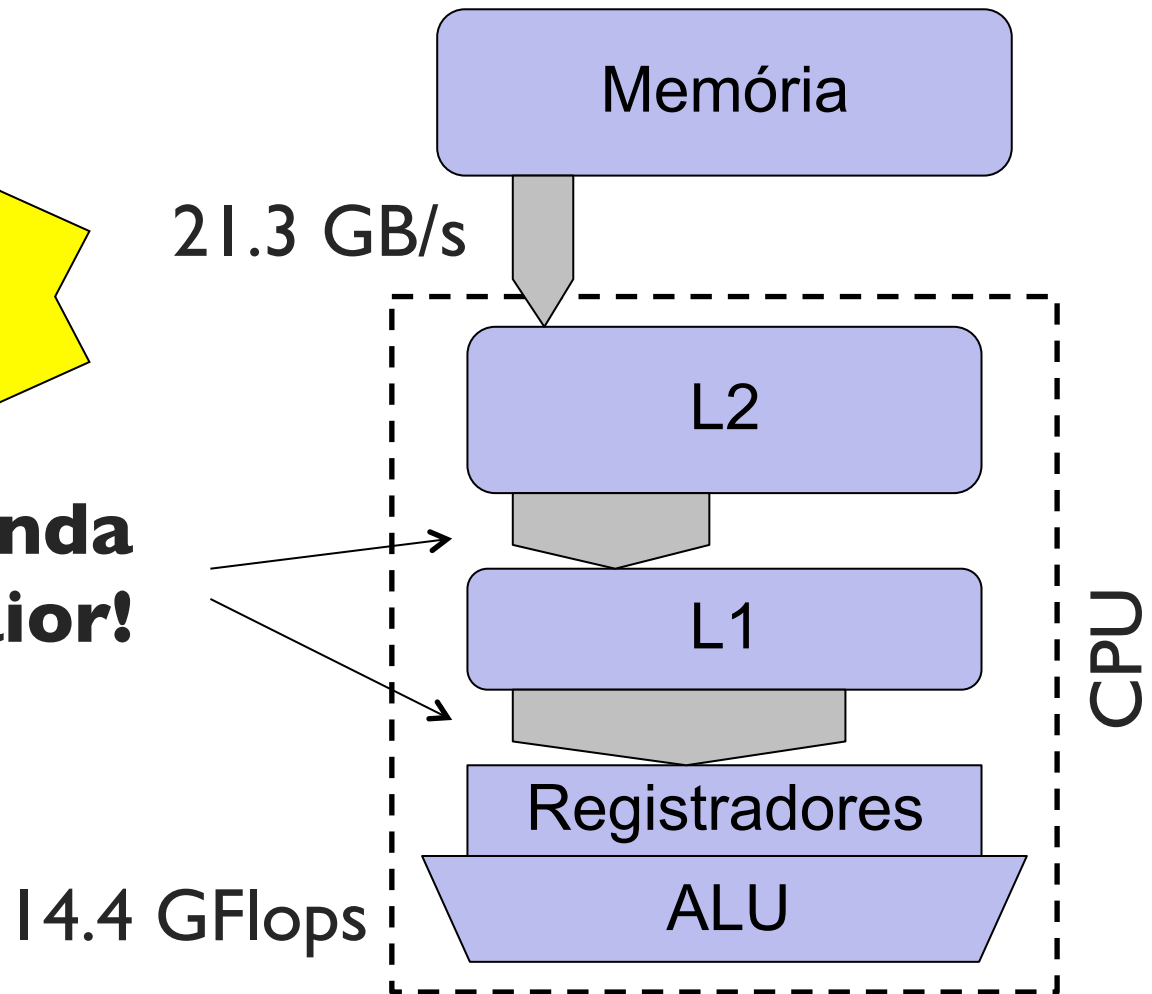
# Otimizações de Acesso a Dados



# Otimizações de Acesso a Dados

Maximizar  
o uso das  
caches!

**A largura de banda  
das *caches* é maior!**



# Otimizações de Acesso a Dados

**Exemplo:** produto interno

```
sum = 0.0;
```

```
for (i=0; i<N; i++)
```

```
    sum += A[i] x B[i];
```

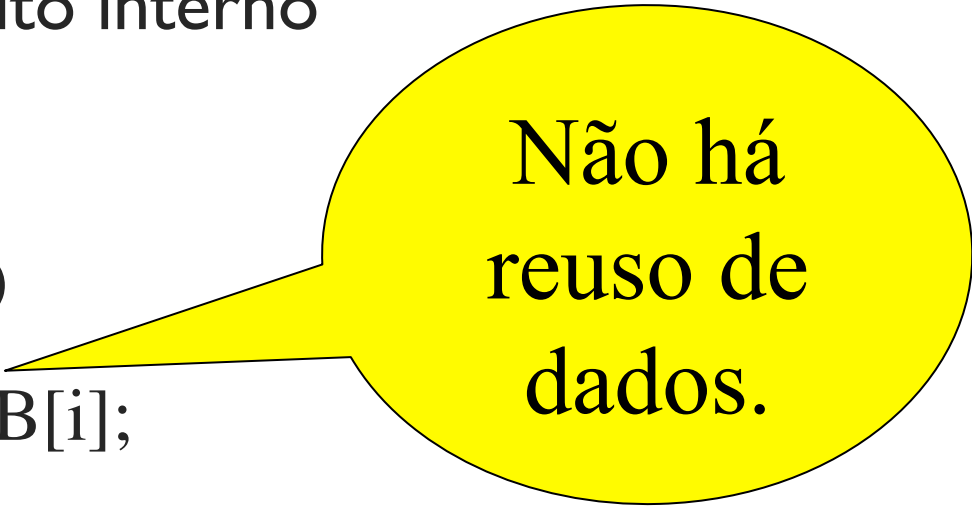
# Otimizações de Acesso a Dados

**Exemplo:** produto interno

```
sum = 0.0;
```

```
for (i=0; i<N; i++)
```

```
    sum += A[i] x B[i];
```



Não há  
reuso de  
dados.

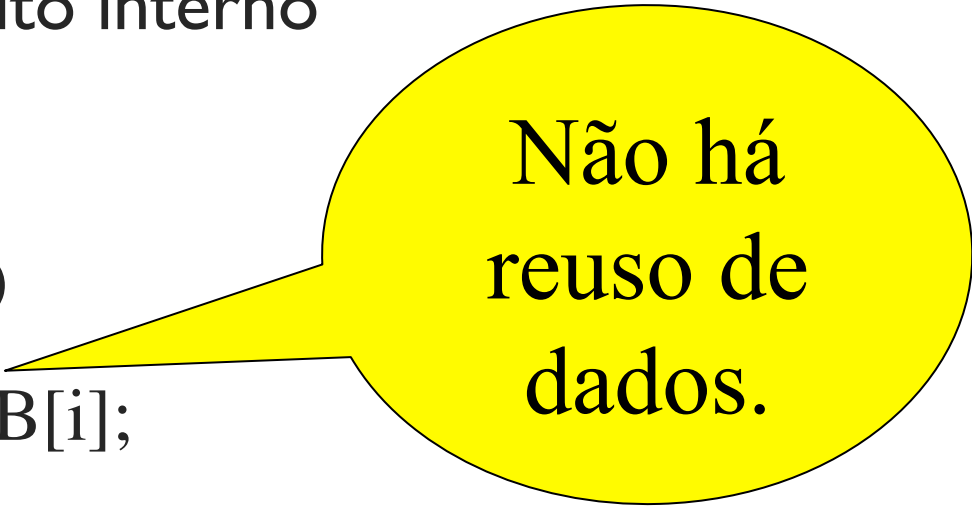
# Otimizações de Acesso a Dados

**Exemplo:** produto interno

```
sum = 0.0;
```

```
for (i=0; i<N; i++)
```

```
    sum += A[i] x B[i];
```



Não há  
reuso de  
dados.

Qual o desempenho esperado?

Lembrando que pico = 21.3 GB/s e 14.4 GFlops/s

# Otimizações de Acesso a Dados

## Experimento:

```
while(n<10) {  
    clean_caches();  
    t = mysecond(); // tempo em segundos  
    res = produto_interno(a,b);  
    t = mysecond()-t;  
    min_t=MIN(t,min);  
}
```

# Otimizações de Acesso a Dados

**Exemplo:** produto interno

```
sum = 0.0;
```

```
for (i=0; i<N; i++)
```

```
    sum += A[i] x B[i];
```

Banda =  $(2 \times \text{SIZE} \times 8) / \text{min\_t} = \sim \mathbf{6.1 \text{ GB/s}}$

- pico = 21.3 GB/s

Flops =  $(2 \times \text{SIZE}) / \text{min\_t} = \sim \mathbf{0.76 \text{ GFLOPS}}$

- pico: 14.4 GFlops

# Otimizações de Acesso a Dados

**Exemplo:** produto interno

```
sum = 0.0;
```

```
for (i=0; i<N; i++)
```

```
    sum += A[i] x B[i];
```

Limitado pela  
memória.  
*“Memory Bound”*

Banda =  $(2 \times \text{SIZE} \times 8) / \text{min\_t} = \sim \mathbf{6.1 \text{ GB/s}}$

- pico = 21.3 GB/s

Flops =  $(2 \times \text{SIZE}) / \text{min\_t} = \sim \mathbf{0.76 \text{ GFLOPS}}$

- pico: 14.4 GFlops



# Otimizações de Acesso a Dados

**Exemplo:** transposição de matrizes

```
for (i=0; i<N; i++)
```

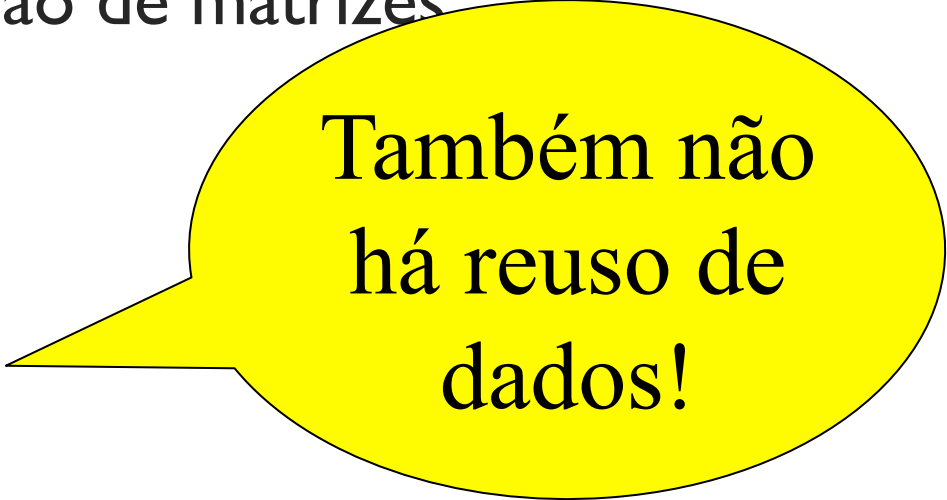
```
    for (j=0; j<N; j++)
```

```
        A[i][j] = B[j][i]
```

# Otimizações de Acesso a Dados

**Exemplo:** transposição de matrizes

```
for (i=0; i<N; i++)  
    for (j=0; j<N; j++)  
        A[i][j] = B[j][i]
```



Também não  
há reuso de  
dados!

Qual o desempenho esperado?

Lembrando que:

- Pico: 21.3 GB/s e 14.4 GFlops/s
- Prod. Interno: ~6.1 GB/s

# Otimizações de Acesso a Dados

**Exemplo:** transposição de matrizes

```
for (i=0; i<N; i++)  
    for (j=0; j<N; j++)  
        A[i][j] = B[j][i]
```



Apenas  
0.93 GB/s

Qual o desempenho esperado?

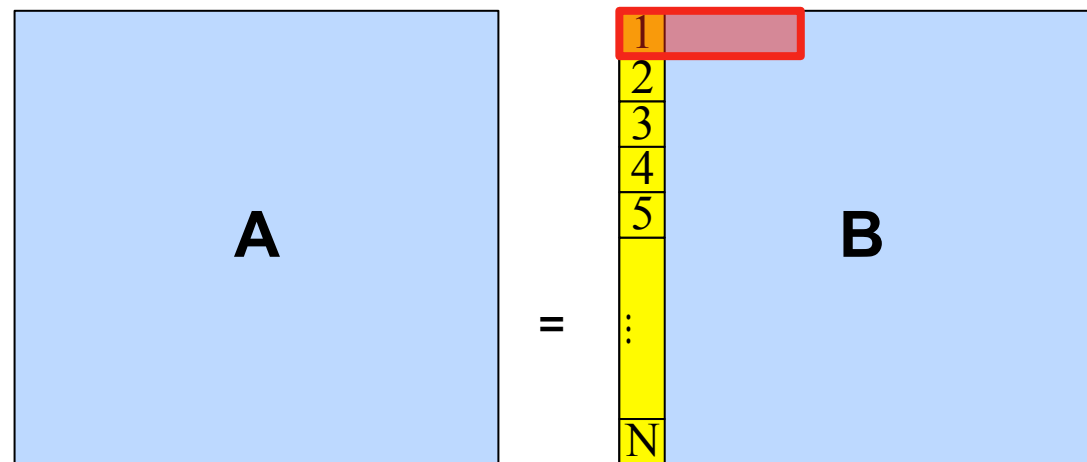
Lembrando que:

- Pico: 21.3 GB/s e 14.4 GFlops/s
- Prod. Interno: ~6.1 GB/s

# Otimizações de Acesso a Dados

**Exemplo:** transposição de matrizes (**Problema!**)

$a[0][0] = \mathbf{b[0][0]}$ ;

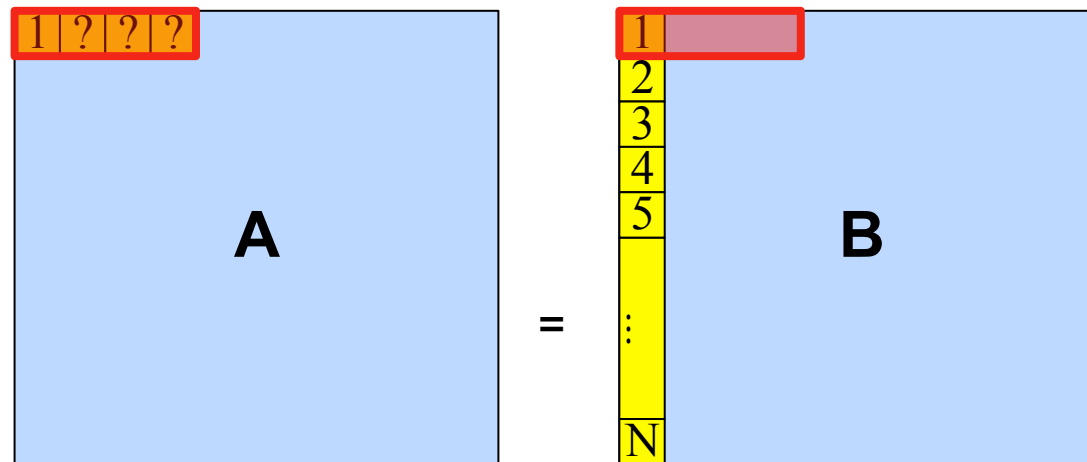


Suponha que a linha de cache comporte  $32 \text{ bytes}$   
 $\Rightarrow$  4 números de ponto flutuante (precisão dupla)

# Otimizações de Acesso a Dados

**Exemplo:** transposição de matrizes (**Problema!**)

$a[0][0] = b[0][0];$

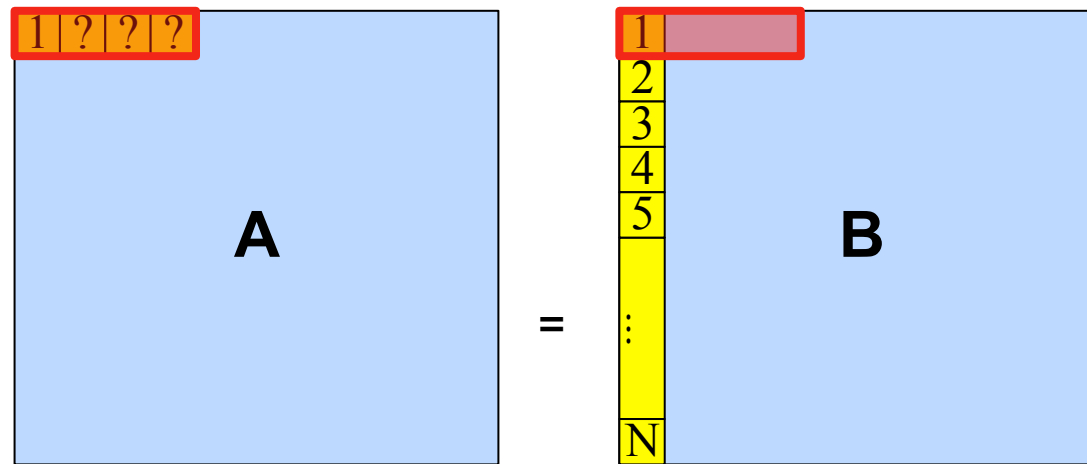


Suponha que a linha de cache comporte 32 *bytes*  
 $\Rightarrow$  4 números de ponto flutuante (precisão dupla)

# Otimizações de Acesso a Dados

**Exemplo:** transposição de matrizes (**Problema!**)

$a[0][0] = b[0][0];$

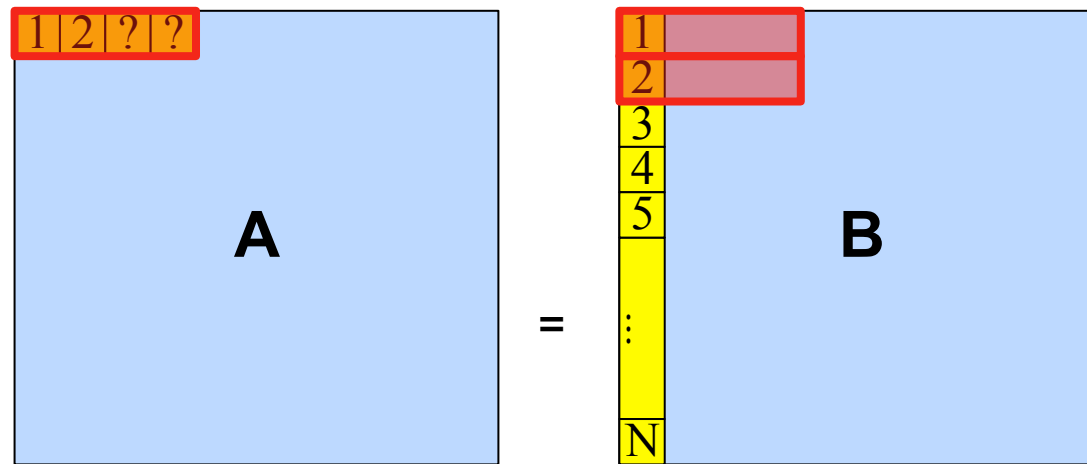


Suponha que a linha de cache comporte 32 *bytes*  
 $\Rightarrow$  4 números de ponto flutuante (precisão dupla)

# Otimizações de Acesso a Dados

**Exemplo:** transposição de matrizes (**Problema!**)

$$a[0][1] = b[1][0];$$

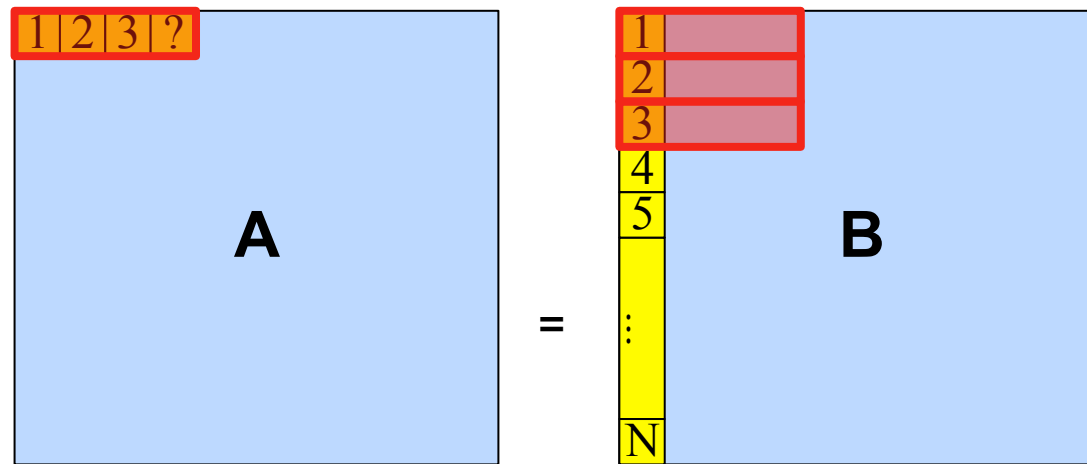


Suponha que a linha de cache comporte 32 *bytes*  
=> 4 números de ponto flutuante (precisão dupla)

# Otimizações de Acesso a Dados

**Exemplo:** transposição de matrizes (**Problema!**)

$a[0][2] = b[2][0];$



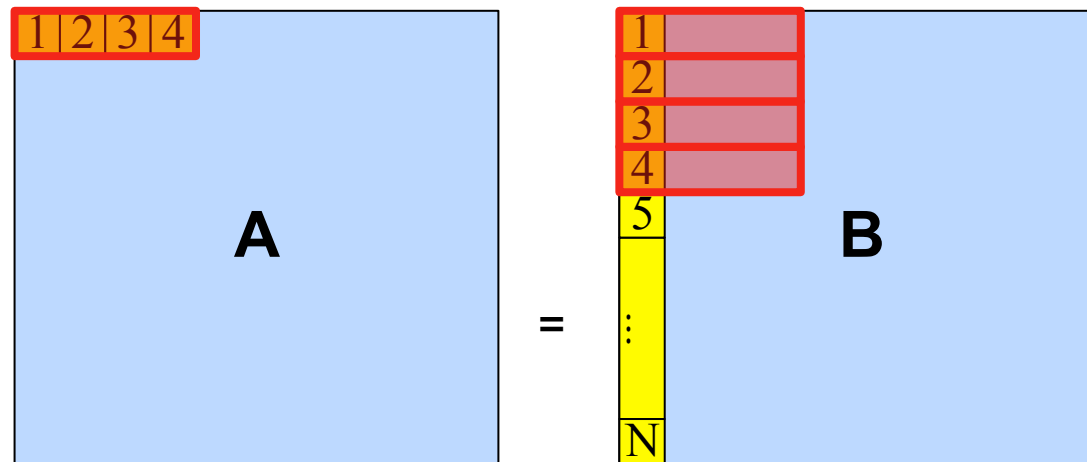
Suponha que a linha de cache comporte 32 *bytes*  
 $\Rightarrow$  4 números de ponto flutuante (precisão dupla)



# Otimizações de Acesso a Dados

**Exemplo:** transposição de matrizes (**Problema!**)

$$a[0][3] = b[3][0];$$

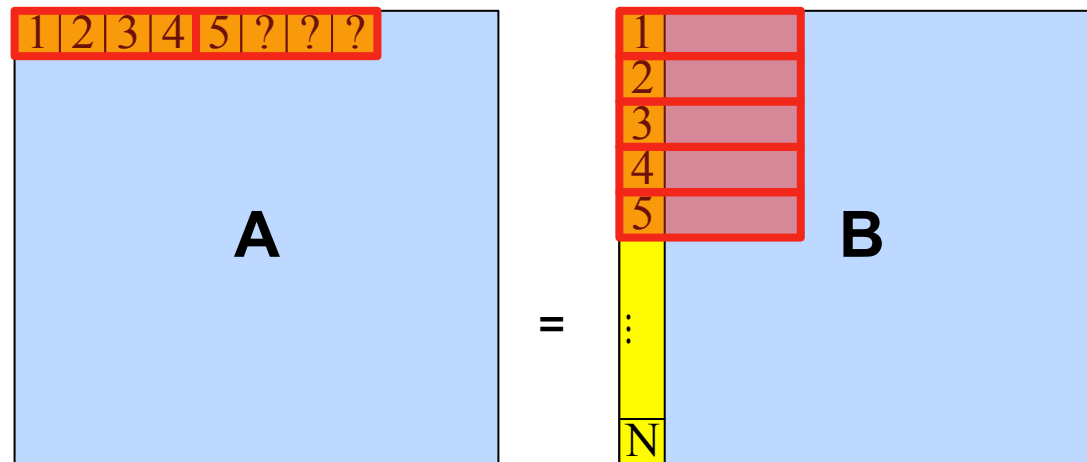


Suponha que a linha de cache comporte 32 *bytes*  
=> 4 números de ponto flutuante (precisão dupla)

# Otimizações de Acesso a Dados

**Exemplo:** transposição de matrizes (**Problema!**)

$$a[0][4] = b[4][0];$$

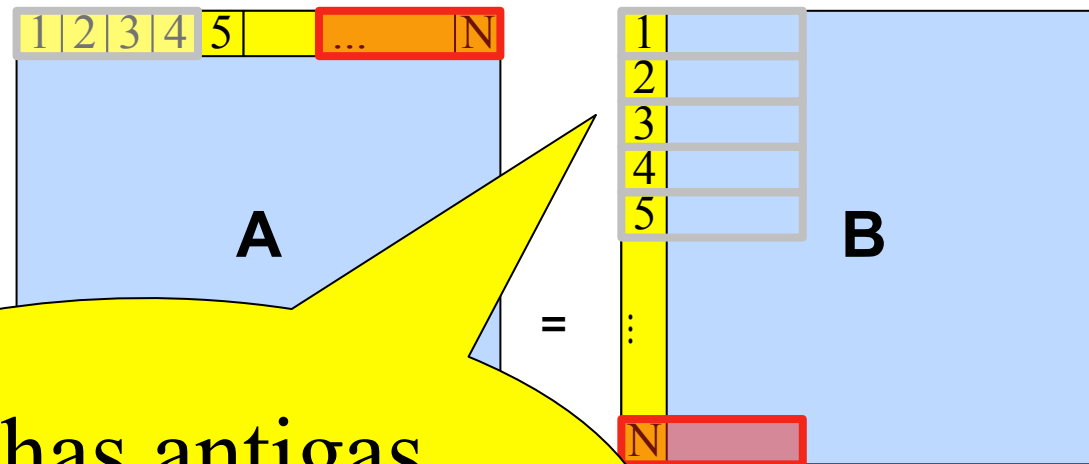


Suponha que a linha de cache comporte 32 *bytes*  
=> 4 números de ponto flutuante (precisão dupla)

# Otimizações de Acesso a Dados

**Exemplo:** transposição de matrizes (**Problema!**)

$$a[0][N-1] = b[N-1][0];$$



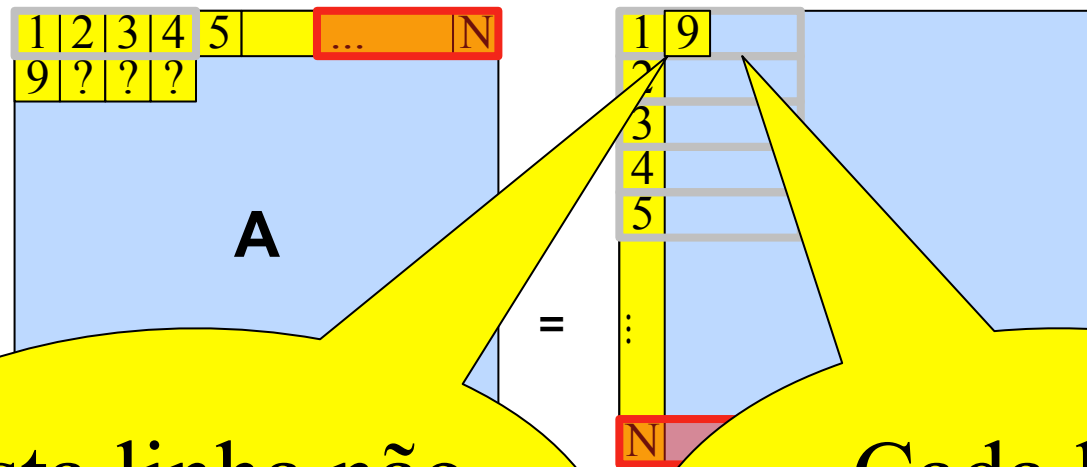
Linhas antigas  
são removidas da  
*cache!* (LRU)

comporte 32 *bytes*  
ante (precisão dupla)

# Otimizações de Acesso a Dados

**Exemplo:** transposição de matrizes (**Problema!**)

$$a[l][0] = b[0][l];$$



Esta linha não está mais na cache!

Cada linha será trazida 4 vezes!!!

# Otimizações de Acesso a Dados

**Exemplo:** transposição de matrizes (**Problema!**)

No exemplo anterior (*cache line = 32 bytes*)

# acessos esperado:  $N^2 + N^2$

# acessos realizado:  $N^2 + 4 \times N^2$

# Otimizações de Acesso a Dados

**Exemplo:** transposição de matrizes (**Problema!**)

No exemplo anterior (*cache line = 32 bytes*)

# acessos esperado:  $N^2 + N^2$

# acessos realizado:  $N^2 + 4 \times N^2$

No meu laptop (*cache line = 64 bytes*)

# acessos esperado:  $N^2 + N^2$

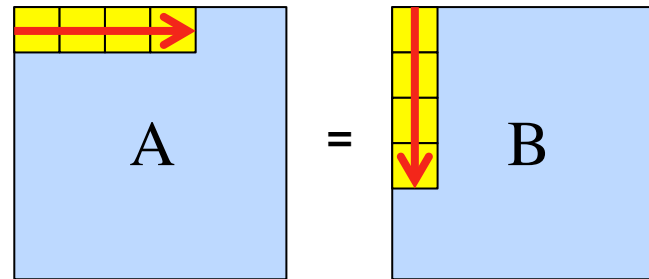
# acessos realizado:  $N^2 + 8 \times N^2$  (**4.5x** mais acessos)

# Otimizações de Acesso a Dados

**Exemplo:** transposição de matrizes (**Solução!**)

Blocagem!

- Mudar a ordem de acesso

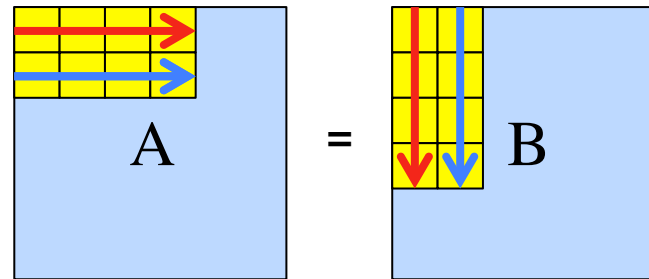


# Otimizações de Acesso a Dados

**Exemplo:** transposição de matrizes (**Solução!**)

Blocagem!

- Mudar a ordem de acesso



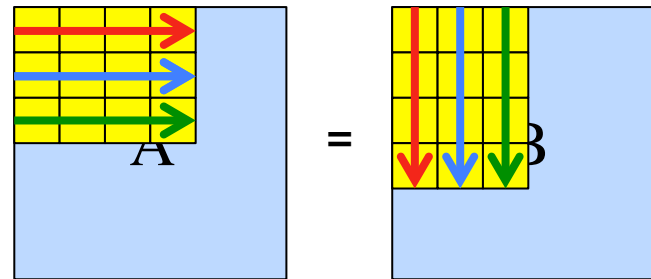


# Otimizações de Acesso a Dados

**Exemplo:** transposição de matrizes (**Solução!**)

Blocagem!

- Mudar a ordem de acesso

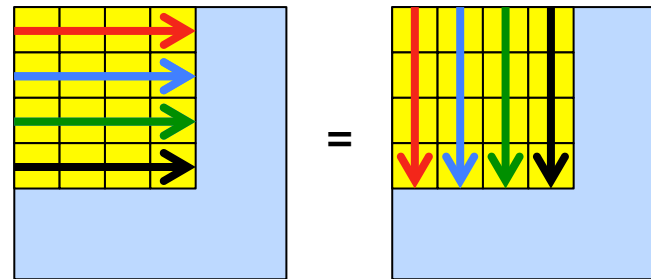


# Otimizações de Acesso a Dados

**Exemplo:** transposição de matrizes (**Solução!**)

Blocagem!

- Mudar a ordem de acesso

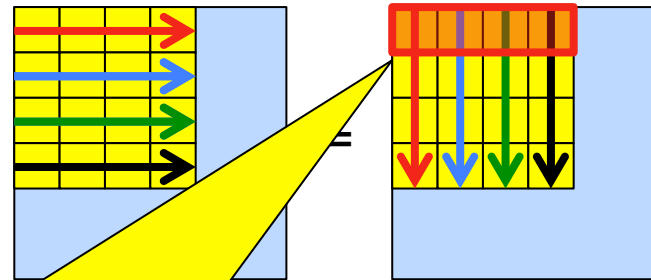


# Otimizações de Acesso a Dados

**Exemplo:** transposição de matrizes (**Solução!**)

Blocagem!

- Mudar a ordem de acesso



Preenchemos a linha de  
*cache* antes dela ser  
removida!

# Otimizações de Acesso a Dados

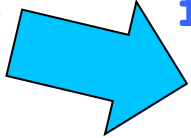
**Exemplo:** transposição de matrizes (**blocagem**)

```
for (i=0; i<N; i++)  
    for (j=0; j<N; j++)  
        A[i][j]=B[j][i]
```

# Otimizações de Acesso a Dados

**Exemplo:** transposição de matrizes (**blocagem**)

```
for(i=0;i<N;i++)      for(i=0; i<N; i++)
  for(j=0;j<N;j++)    for(jk=0; jk<N; jk+=BLK)
    A[i][j]=B[j][i]   for(j=jk; j<(jk+BLK); j++)
                       A[i][j] = B[j][i]
```



## Strip-mining

# Otimizações de Acesso a Dados

**Exemplo:** transposição de matrizes (**blocagem**)

```
for (i=0; i<N; i++)  
  for (j=0; j<N; j++)  
    A[i][j]=B[j][i]
```

```
for (jk=0; jk<N; jk+=BLK)  
  for (i=0; i<N; i++)  
    for (j=jk; j<(jk+BLK); j++)  
      A[i][j] = B[j][i]
```

## Loop interchange

# Otimizações de Acesso a Dados

**Exemplo:** transposição de matrizes

Pico: 21.3 GB/s

Cópia de matrizes: ~6.1 GBytes/s

Banda sem blocagem: ~0.93 GBytes/s

**Banda com blocagem: ~ 5.26 GBytes/s**

# Otimizações de Acesso a Dados

**Exemplo:** multiplicação de matrizes

```
for (i=0; i<N; i++)
```

```
    for (j=0; j<N; j++)
```

```
        for (k=0; k<N; k++)
```

```
            C[i][j] += A[i][k] x B[k][j]
```




# Otimizações de Acesso a Dados

**Exemplo:** multiplicação de matrizes

```
for (i=0; i<N; i++)  
  for (j=0; j<N; j++)  
    for (k=0; k<N; k++)
```

```
      C[i][j] += A[i][k] x B[k][j]
```



Há bastante  
reuso de dados.

$3 \times N^3$  acessos a  $3 \times N^2$  posições de memória

=> cada elemento é reutilizado  $N-1$  vezes!!!

# Otimizações de Acesso a Dados

**Exemplo:** multiplicação de matrizes


```
for (i=0; i<N; i++)  
    for (j=0; j<N; j++)  
        for (k=0; k<N; k++)
```

```
    C[i][j] += A[i][k] x B[k][j]
```

3 x N<sup>3</sup> acessos a 3 x N<sup>2</sup> posições de memória

=> cada elemento é reutilizado N-1 vezes!!!

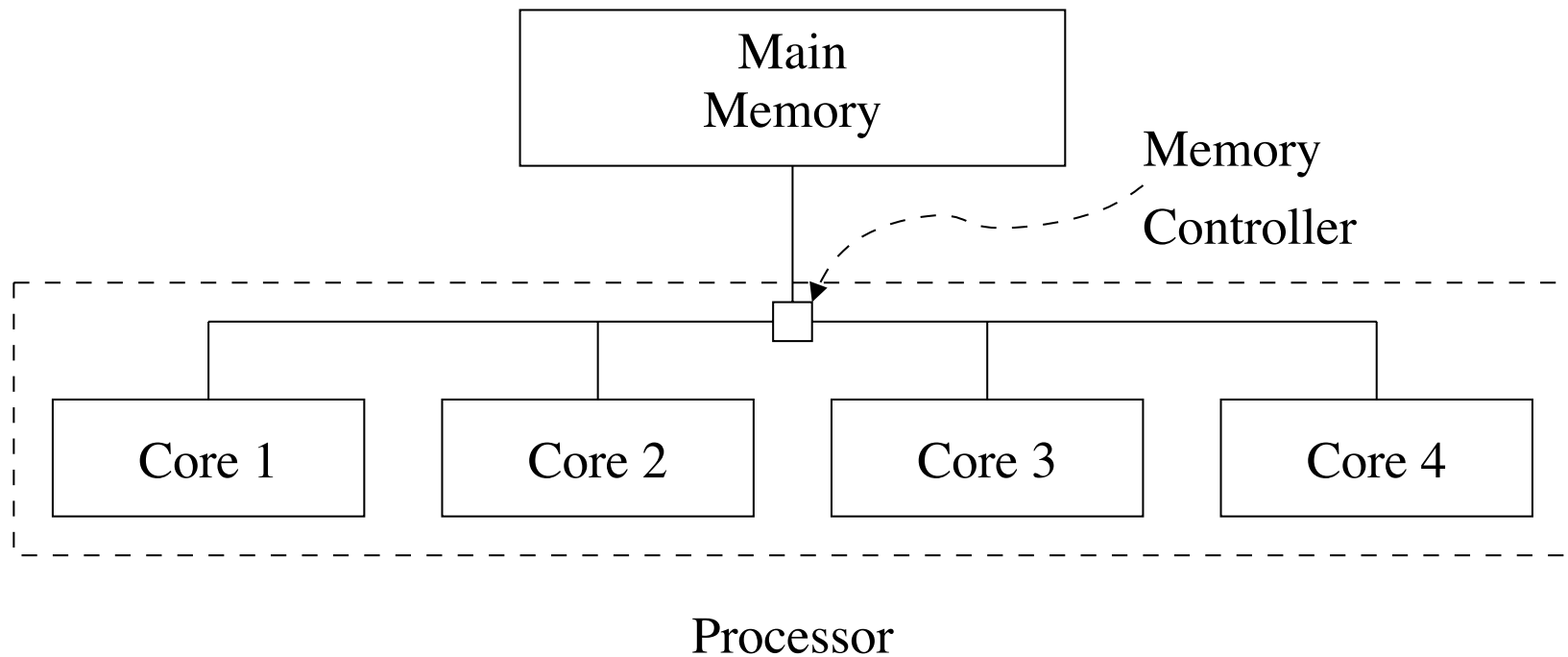
**Blocagem!**



Reutilizar o  
dado antes do  
mesmo deixar a  
*cache!!!*

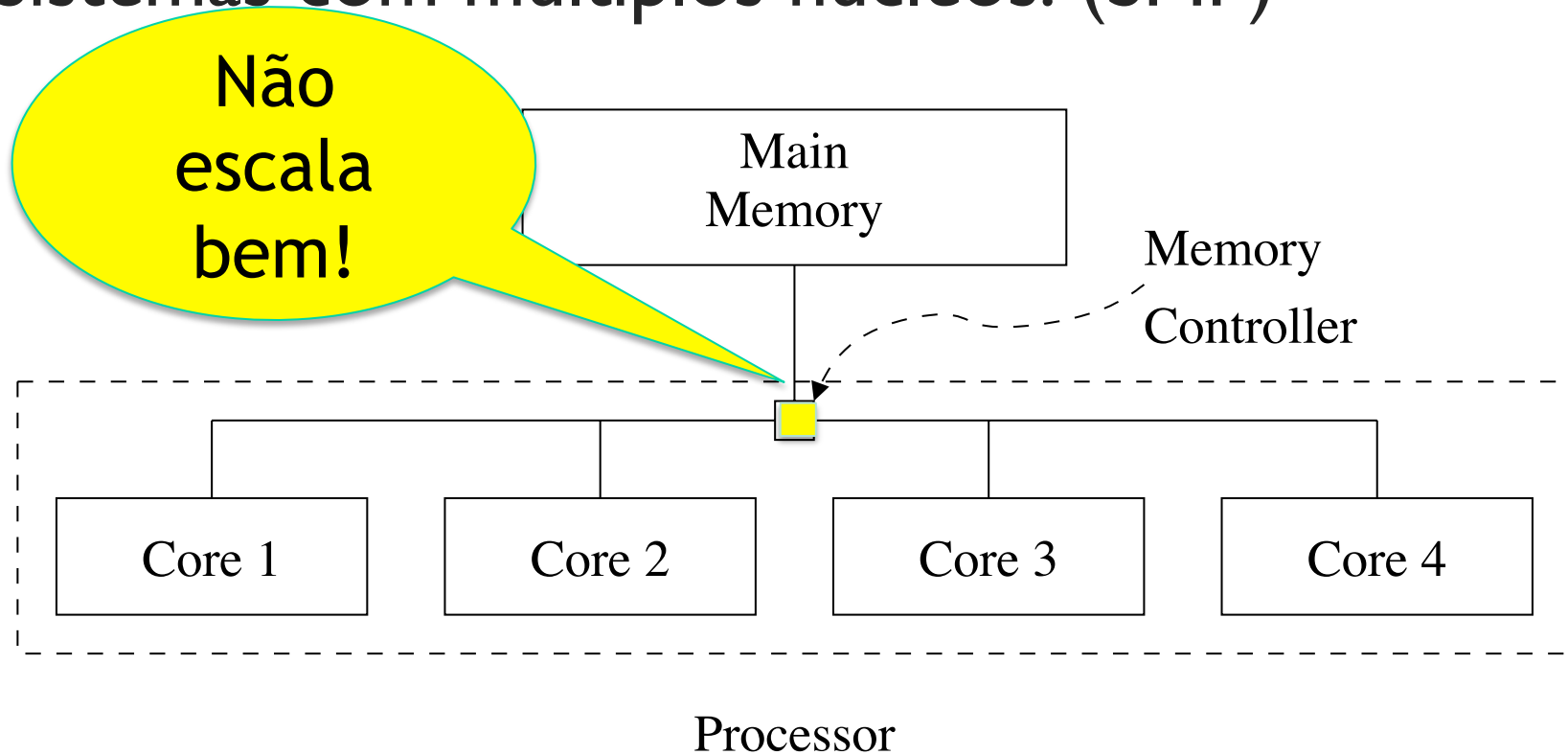
# Otimizações de Acesso a Dados

Sistemas com múltiplos núcleos. (SMP)



# Otimizações de Acesso a Dados

Sistemas com múltiplos núcleos. (SMP)



# Otimizações de Acesso a Dados

Sistemas com múltiplos núcleos. (SMP)

Primeiros *multi-cores*

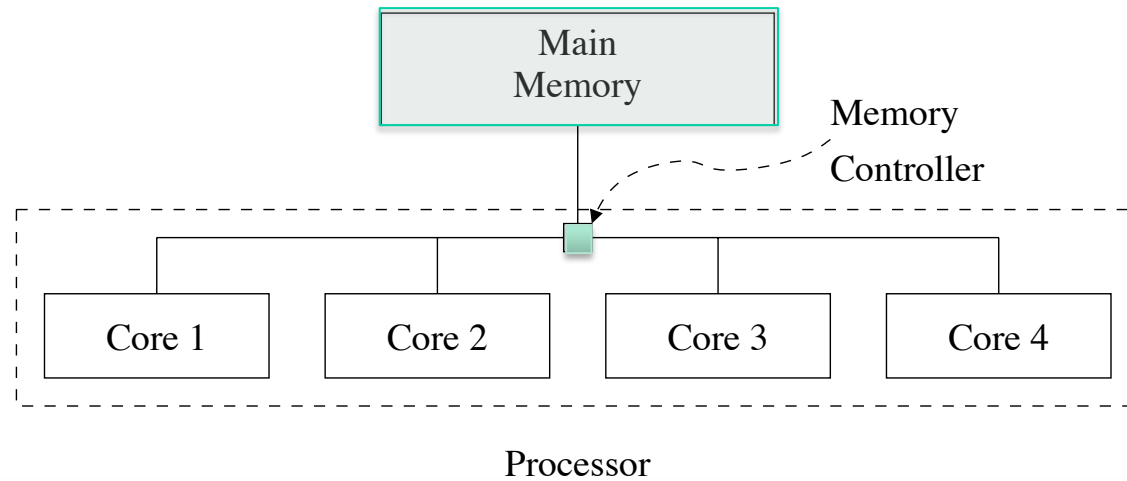
- 2 e 4 cores => SMP Ok!

Recentemente

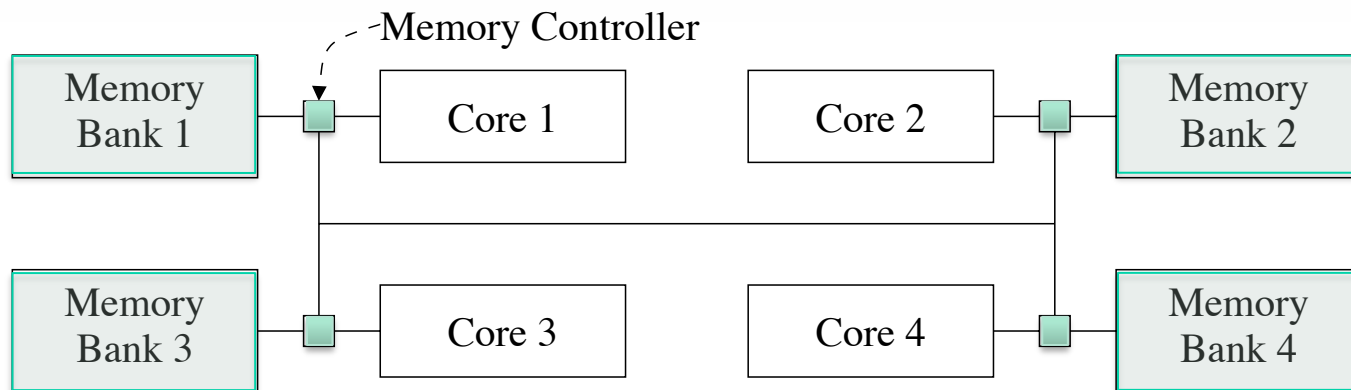
- 16, 32, 64 cores => SMP Não escala!
- Mudança para ccNUMA  
(cache-coherent Non-Uniform Memory Access)

# Otimizações de Acesso a Dados

SMP



ccNUMA



# Otimizações de Acesso a Dados

Sistemas com múltiplos núcleos. (SMP)

ccNUMA:

- Acesso à memória é transparente para o usuário
- Localização do dado no sistema pode afetar o desempenho (acesso não uniforme)

# Otimizações de Acesso a Dados

Sistemas com múltiplos núcleos. (SMP)

Exemplo:

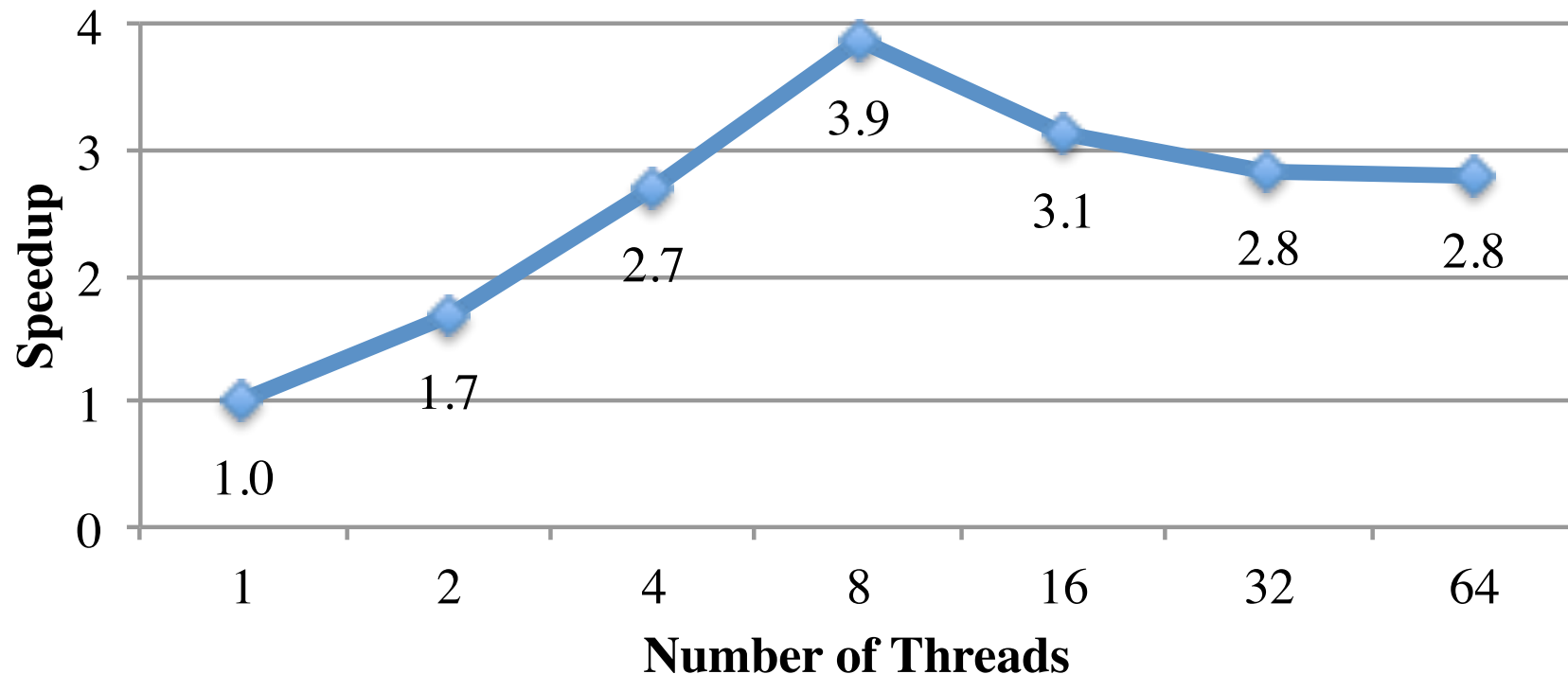
- SW: Decomp. Cholesky em 64 matrizes diferentes
- HW: 64-core Opteron 6282 SE
- ccNUMA
- 8 Bancos de Memória conectados a 8 nós
- Cada nó possui 8 núcleos



# Otimizações de Acesso a Dados

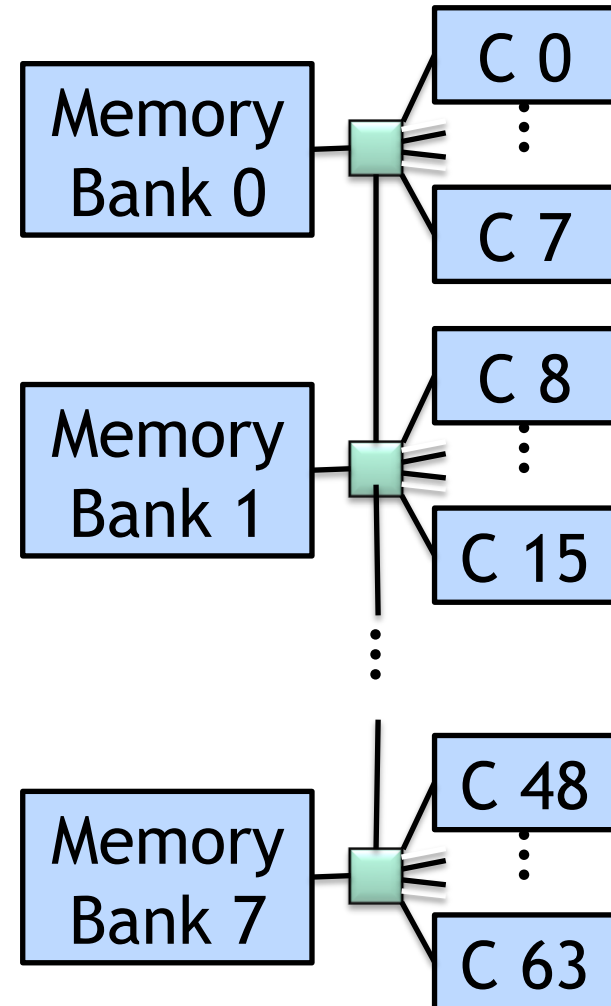
Sistemas com múltiplos núcleos. (SMP)

Resultados Iniciais:



# Otimizações de Acesso a Dados

Thread 1

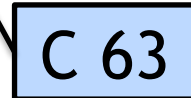
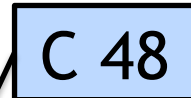
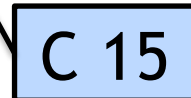
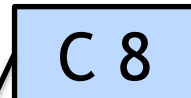
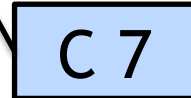
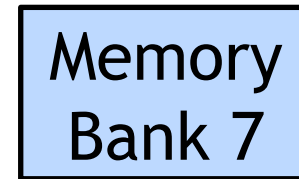
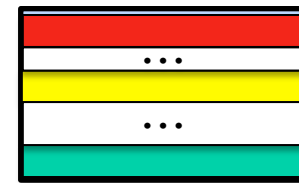
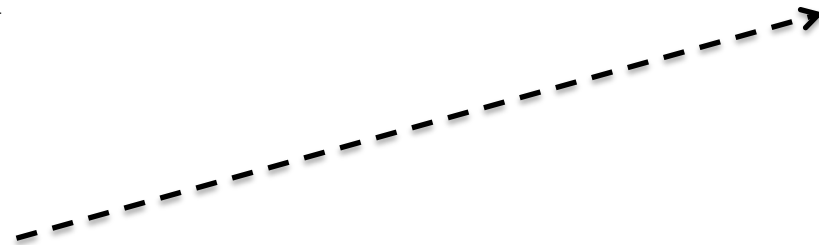


# Otimizações de Acesso a Dados

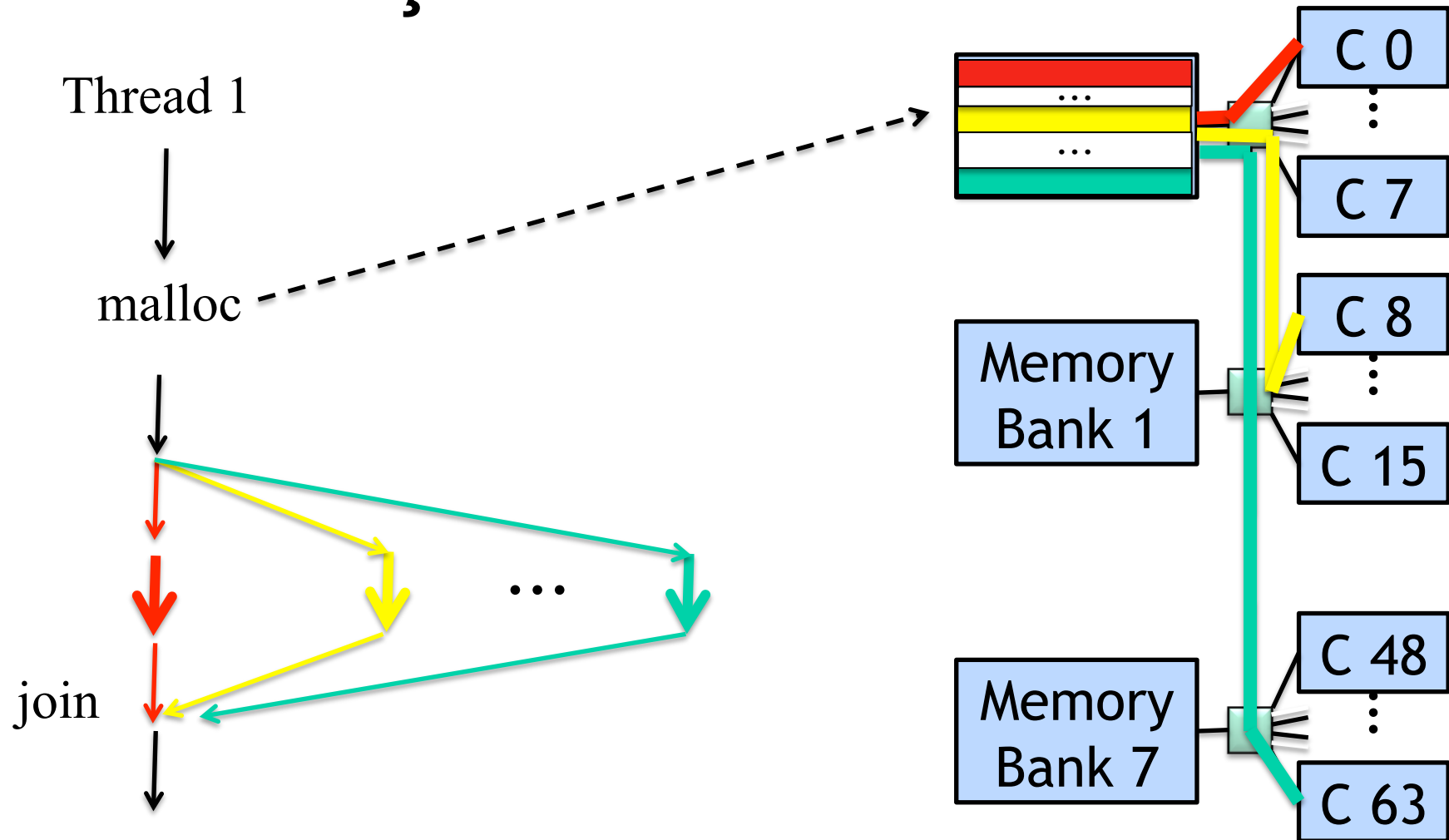
Thread 1



malloc



# Otimizações de Acesso a Dados



# Otimizações de Acesso a Dados

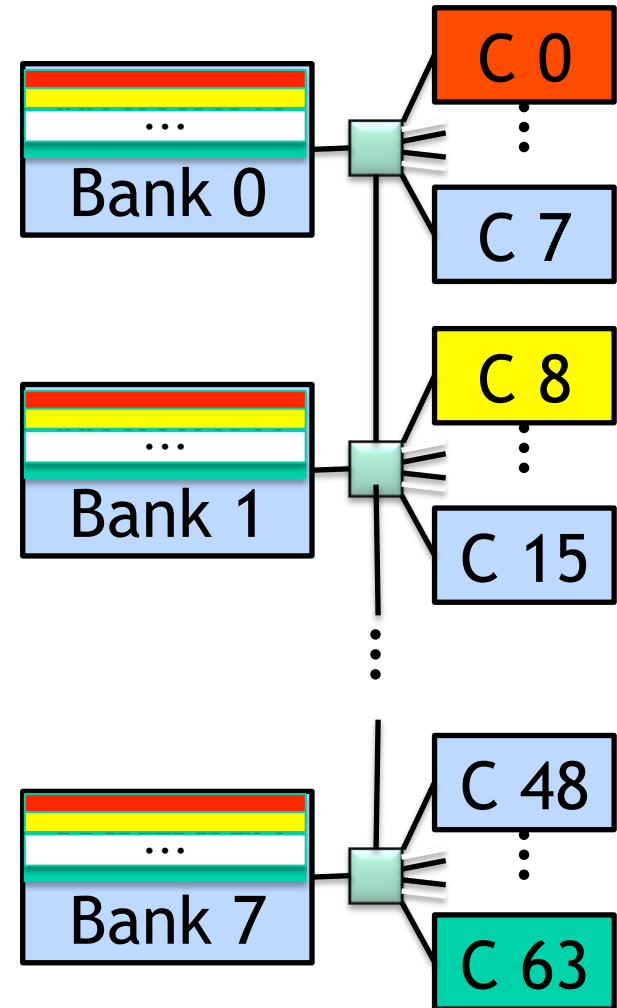
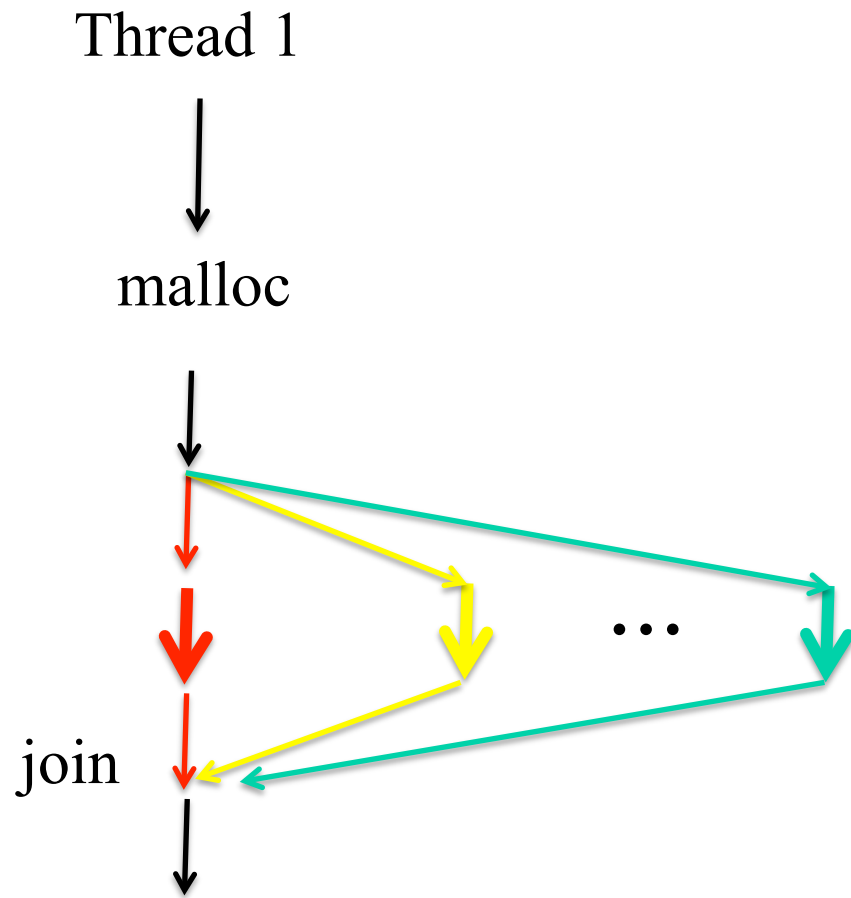
## Solução #1: Intercalar os dados

- Ferramenta: numactl

```
numactl --interleave=all ./my_app.exe
```

Informa ao OS que os dados devem ser distribuídos de forma circular entre os bancos de memória.

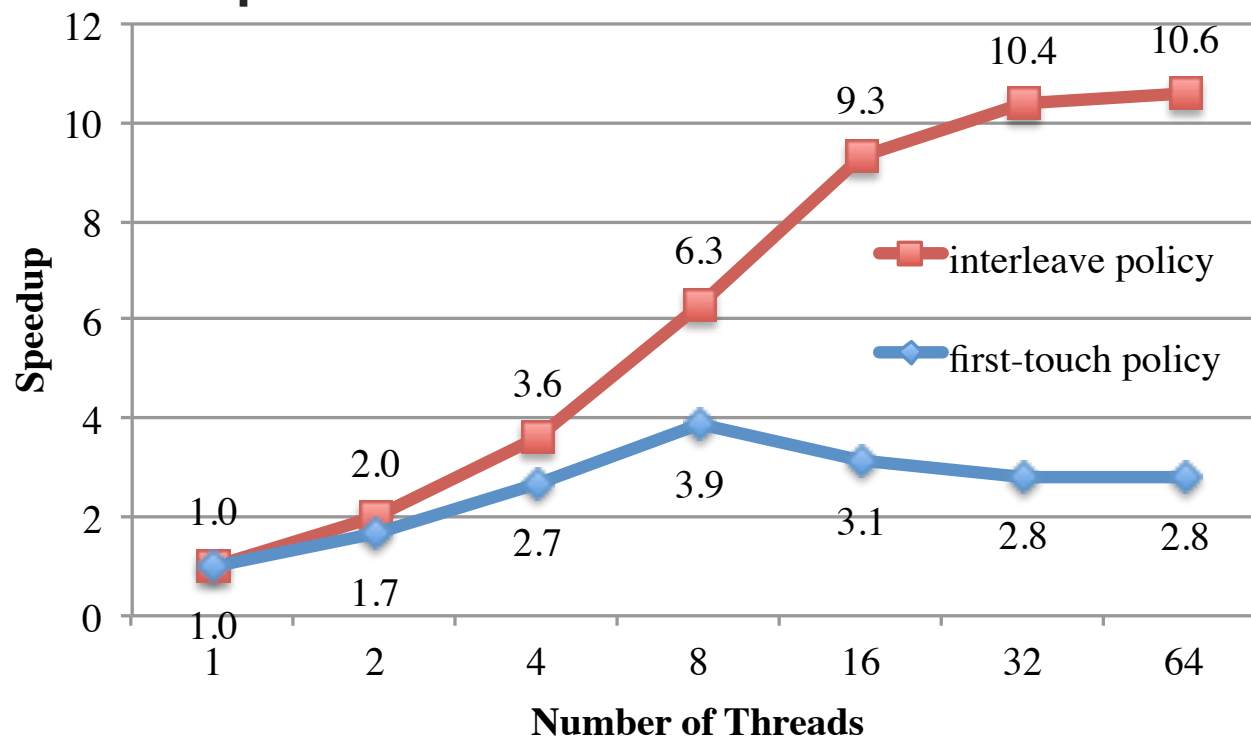
# Otimizações de Acesso a Dados



# Otimizações de Acesso a Dados

Sistemas com múltiplos núcleos. (SMP)

Resultados Após o *Interleave*:



# Otimizações de Acesso a Dados

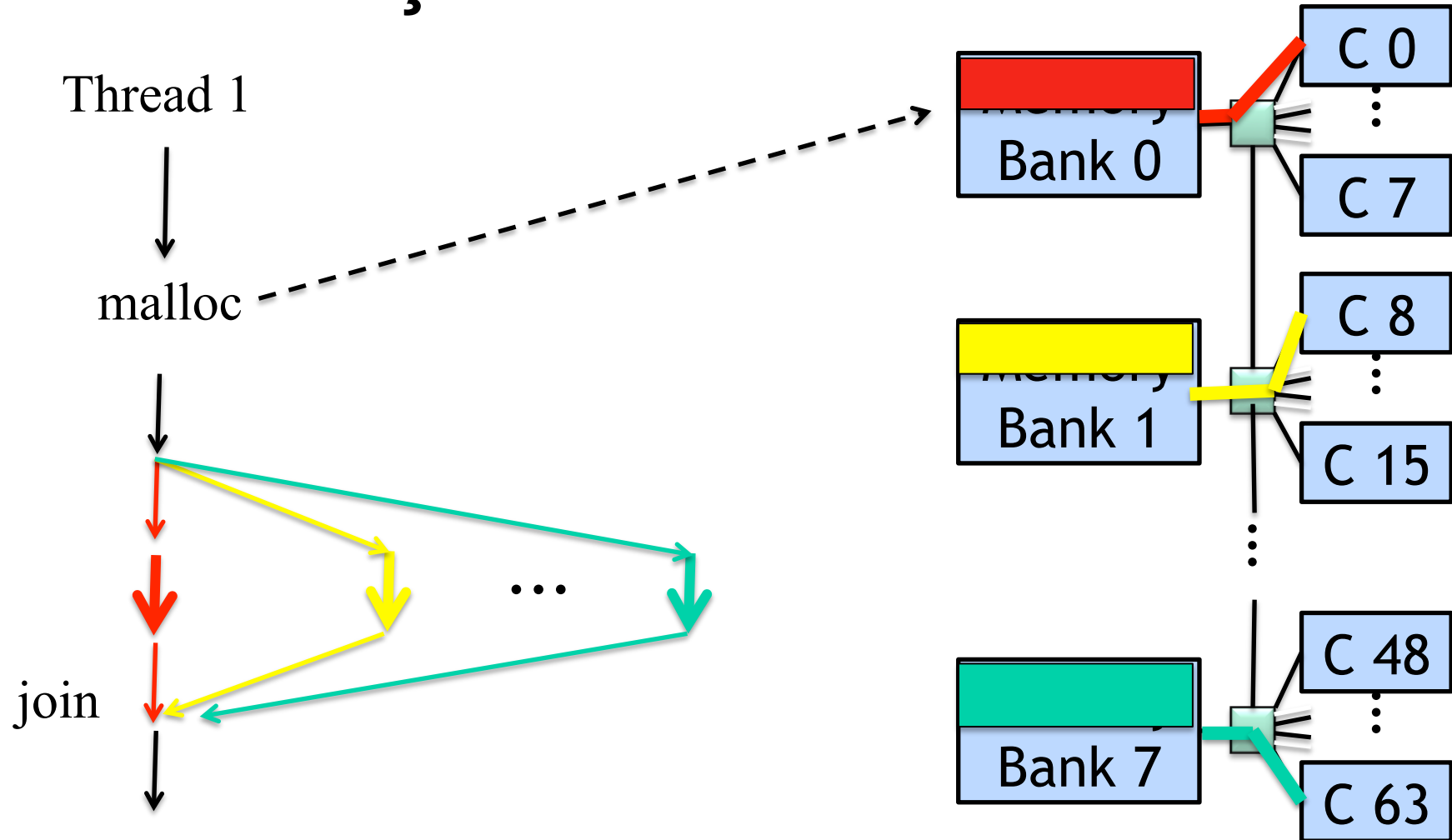
## Solução #2: Distribuir os dados manualmente

Usar a biblioteca libnuma para:

- Alocar os dados das matrizes explicitamente em cada banco de memória
- Restringir o escalonamento de *threads* para o conjunto de núcleos próximo ao banco que contém a matriz a ser decomposta pela *thread*

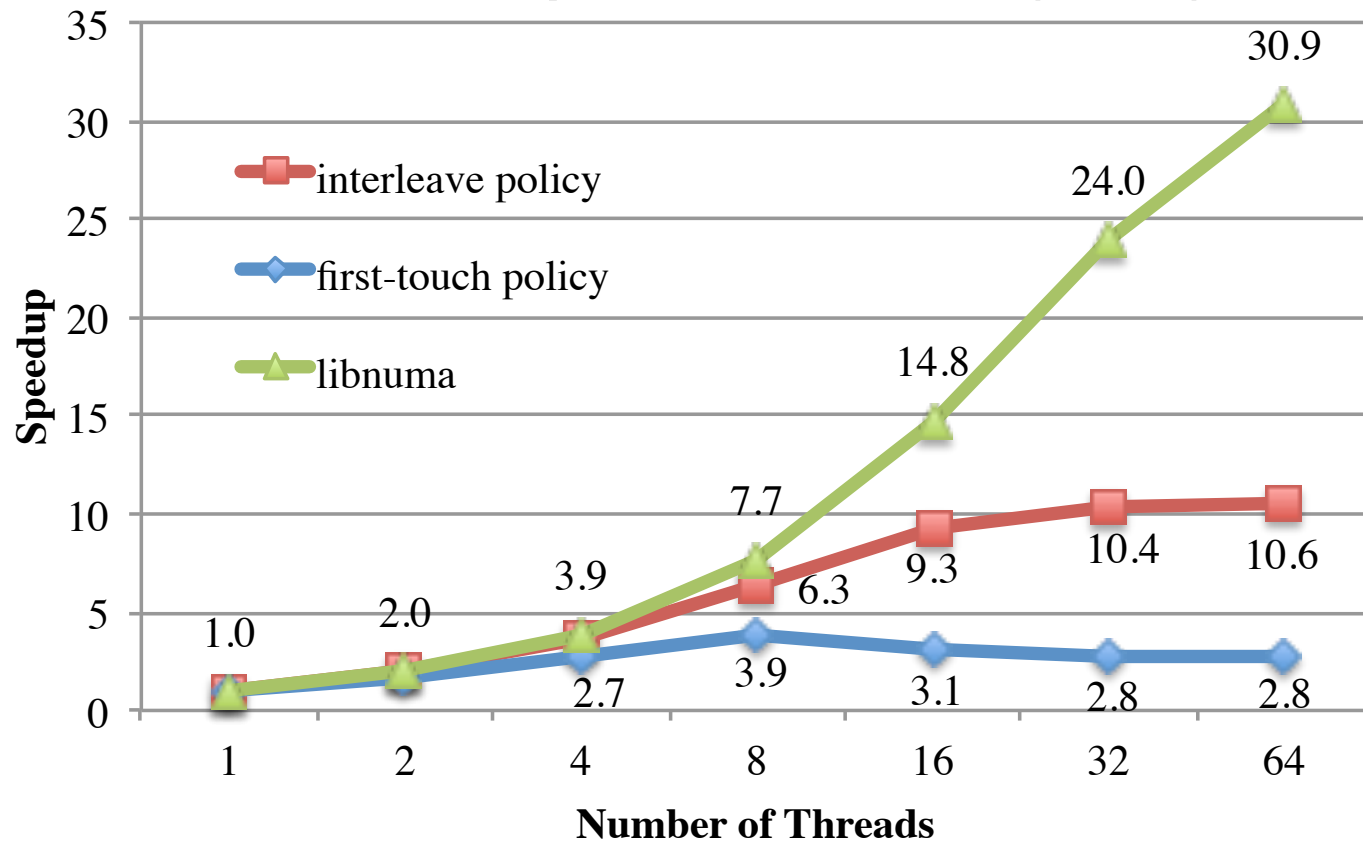


# Otimizações de Acesso a Dados



# Otimizações de Acesso a Dados

## Sistemas com múltiplos núcleos. (SMP)



# Otimizações de Acesso a Dados

Sistemas com múltiplos núcleos. (SMP)

Observações:

- Tendências indicam que os futuros multicores serão ccNUMA
- Extrair bom desempenho destas máquinas pode exigir gerenciamento de dados explícito

# Conclusões

- Tráfego de dados no sistema pode determinar o desempenho (“*memory bound*”).
- Muito comum em simulações numéricas!
- Otimizações de acesso a dados são fundamentais nestes casos.

# Agenda

- Perfilamento – Contagem de tempo
- Otimização de acesso a dados
- **Atividade de laboratório**

# Atividade de laboratório

[www.ic.unicamp.br/~edson/disciplinas/  
InccI4/index.html](http://www.ic.unicamp.br/~edson/disciplinas/InccI4/index.html)