Memórias Cache Parte III

Tópicos da aula

- **Mapeamento do Bloco de cache**
- **Localidade Temporal e Espacial**
- **Conceito de Blocking Cache**
- **Exemplos**

Mapeamento do Bloco

- Para saber para qual endereço da cache será mapeado um byte específico da memória, precisamos de duas informações
  - O **endereço do bloco** que contém o byte na memória de nível **inferior**
  - A **quantidade de blocos** da cache
- **Exemplo**: considere que a cache armazena 64 blocos, com 16 bytes por bloco. Qual a posição ocupada na cache pelo byte 800?

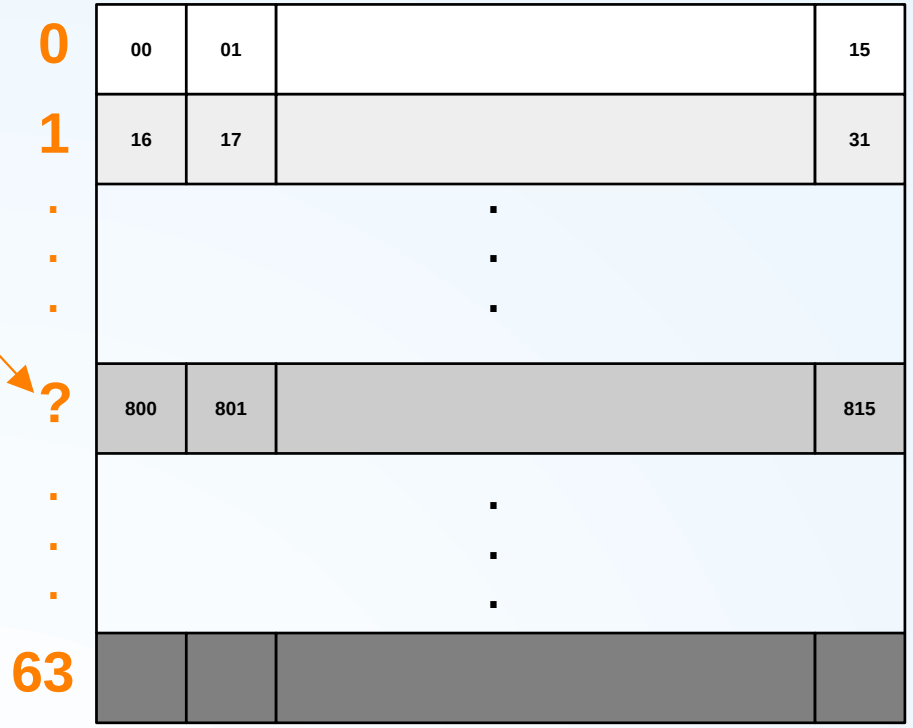
$$N^{o}\,da\,Linha = \frac{endereço\,do\,byte}{byte\,por\,bloco}$$

$$Posição\,na\,Linha = (endereço\,do\,byte)\,MOD\,(bytes\,por\,bloco)$$

Mapeamento do Bloco

- **Exemplo**: considere que a cache armazena 64 blocos, com 16 bytes por bloco. Qual a posição ocupada na cache pelo byte 800?

$$N^o\ da\ Linha = \frac{endereço\ do\ byte}{byte\ por\ bloco} = \frac{800}{16} = 50$$

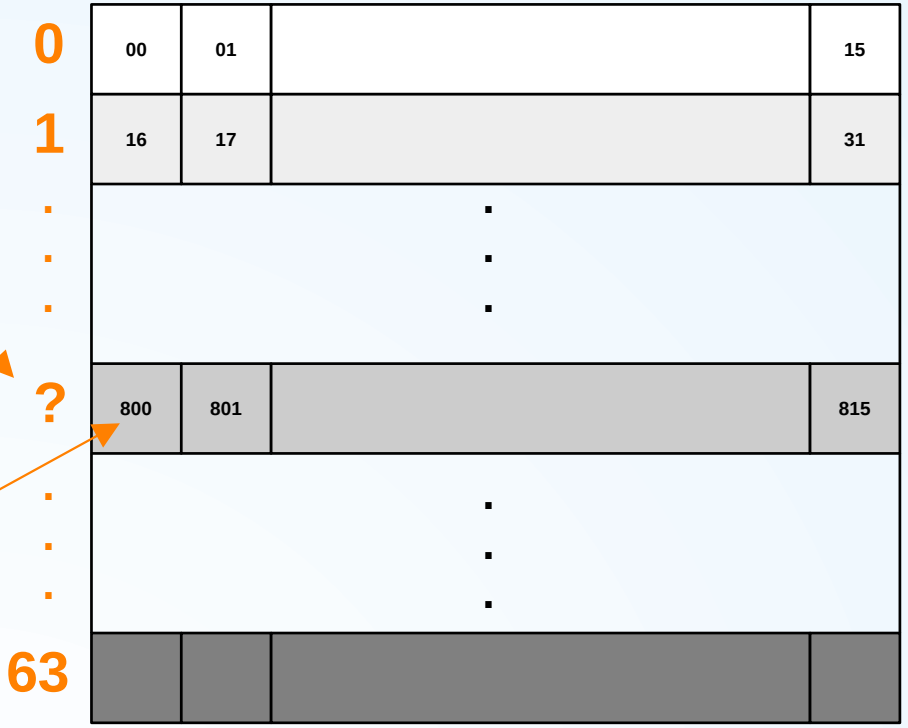


Mapeamento do Bloco

- **Exemplo**: considere que a cache armazena 64 blocos, com 16 bytes por bloco. Qual a posição ocupada na cache pelo byte 800?

$$N^o\ da\ Linha = \frac{endereço\ do\ byte}{byte\ por\ bloco} = \frac{800}{16} = 50$$

$$Posição\ na\ Linha = 800\ MOD\ 16 = 0$$



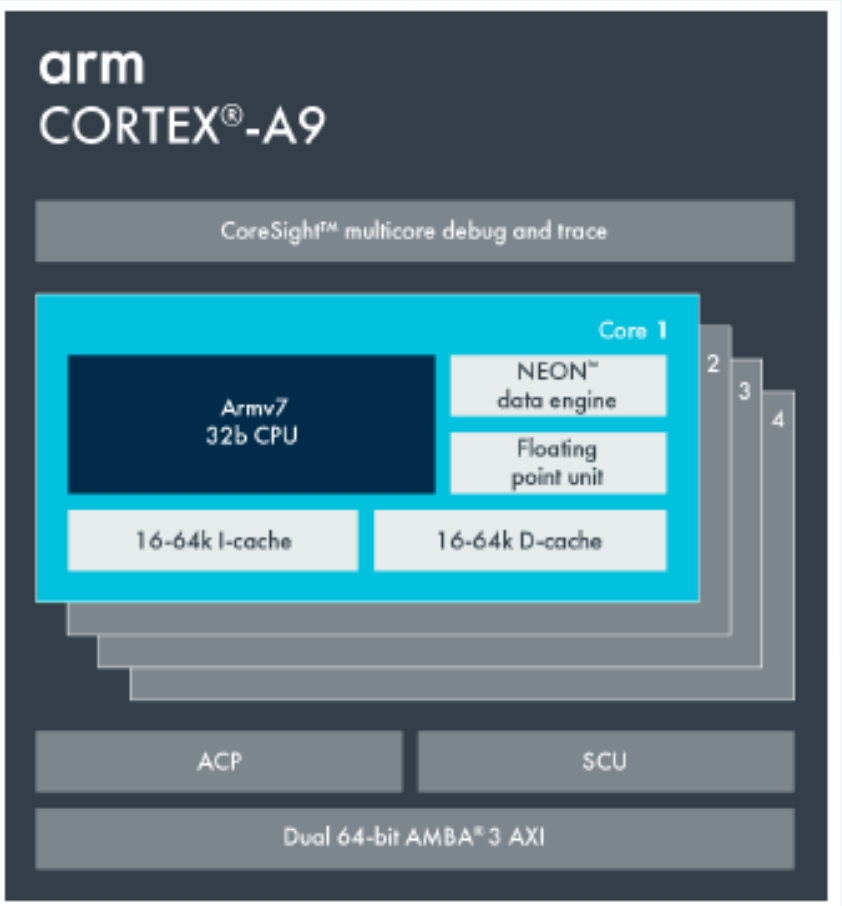
Fontes de Localidade

- **Localidade Temporal**
  - Código dentro de um *loop*
  - Mesmas instruções buscadas repetidamente
- **Localidade Espacial**
  - matrizes de dados
  - Variáveis locais na pilha
  - Dados alocados em blocos (bytes contíguos)

Fontes de Localidade

- ***Loop* em programas** e **Acesso a Dados** acabam por explorar **diferentes localidades**
  - *Loop* → temporal
  - Dados → espacial

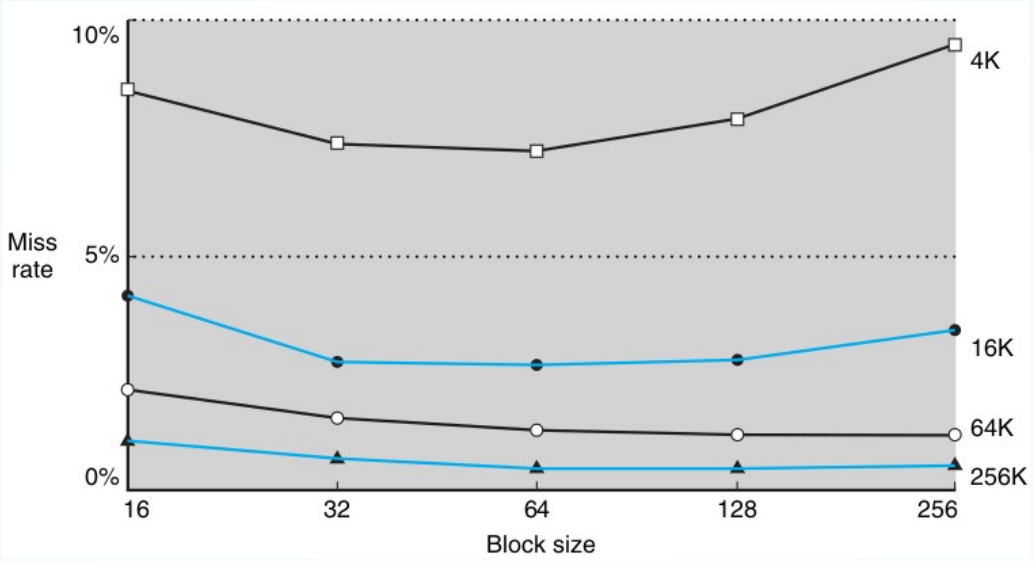
- É comum os fabricantes disponibilizarem memória cache em separado
  - Para Instruções → **I-cache**
  - Para Dados → **D-cache**



Impacto do tamanho do Bloco

- No gráfico temos o miss rate (eixo y) em função do tamanho do bloco (eixo x) para caches de 4K, 16K, 64K e 256K
- **Blocos menores** exploram melhor a **localidade temporal** e

  Blocos maiores exploram melhor a **localidade espacial**

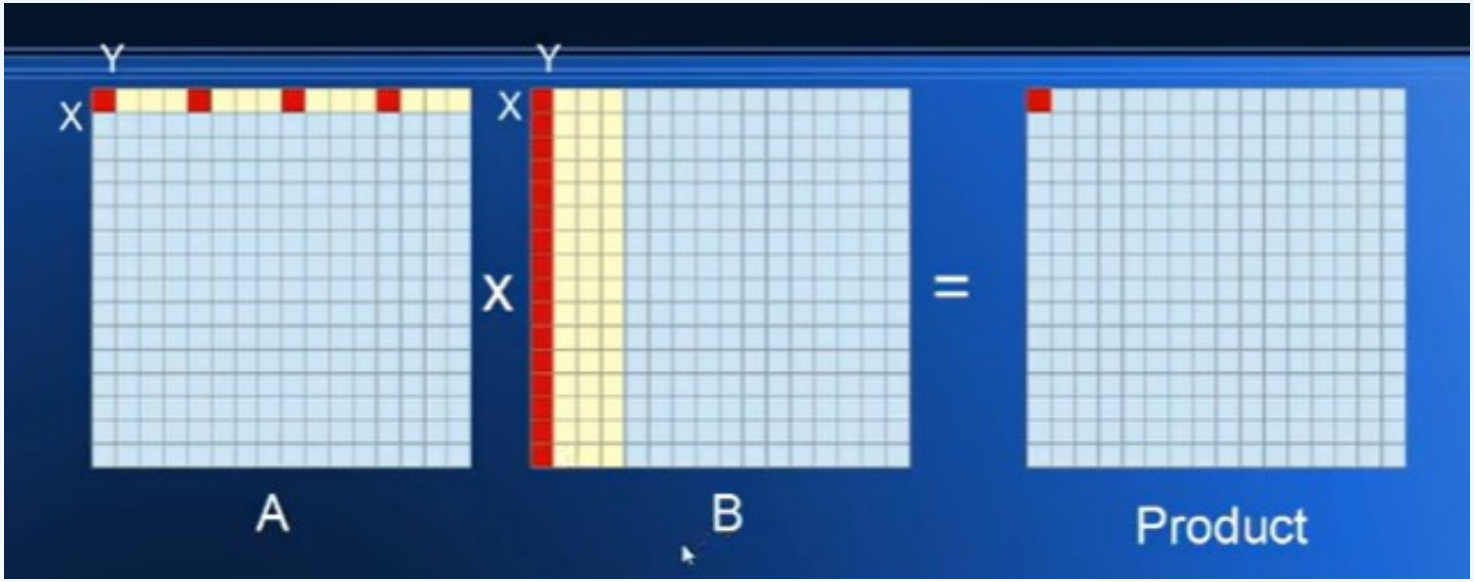


Conceito de Blocking Cache

- **Blocking** é uma técnica de otimização bem conhecida que pode ajudar a **evitar gargalos de largura de banda** de memória em **determinadas aplicações**
  - **Localidade Espacial**
- A principal ideia por trás da técnica é explorar a **reutilização de dados** inerentes da aplicação, garantindo que os **dados permaneçam** na cache em **vários reusos**
- Como isso é feito?
  - Dividindo a cache em **porções** (como blocos)
  - Fazendo acessos de modo a **maximizar o reuso** da porção já alocados

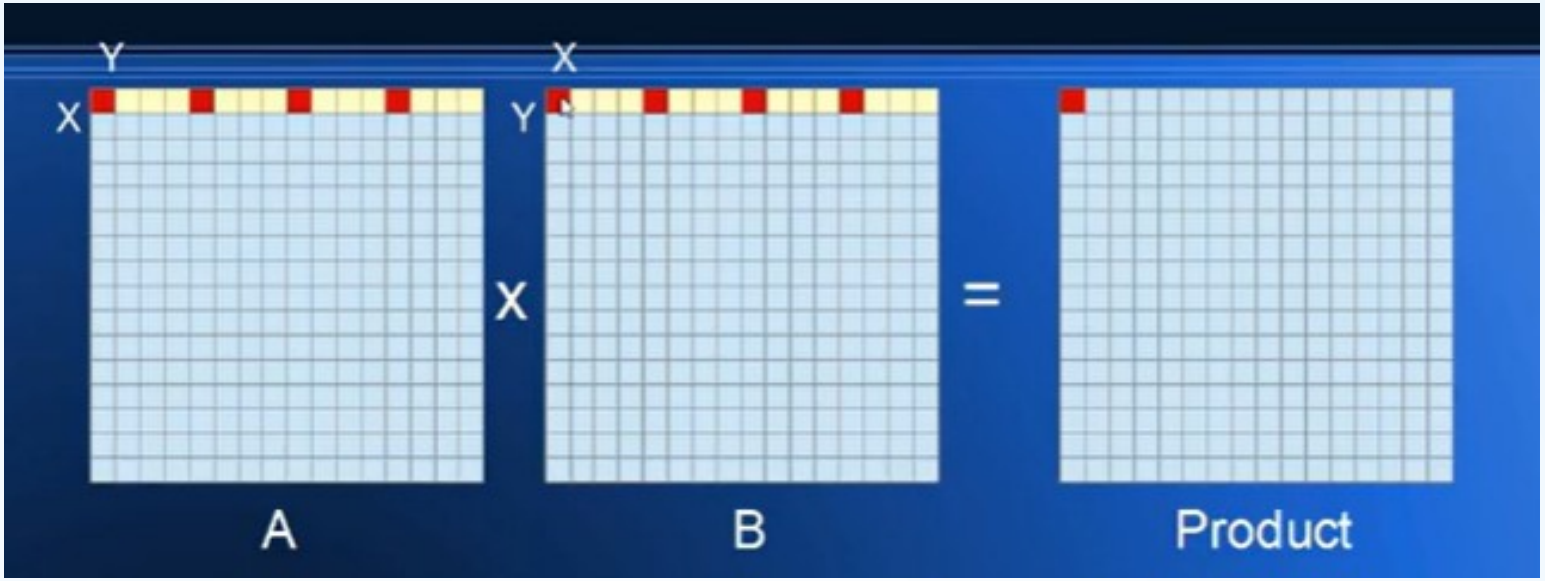
Exemplo: produto de matrizes

- No caso de produto das matrizes A e B, a linha de A sofre menos **miss** que a busca por dados na coluna de B



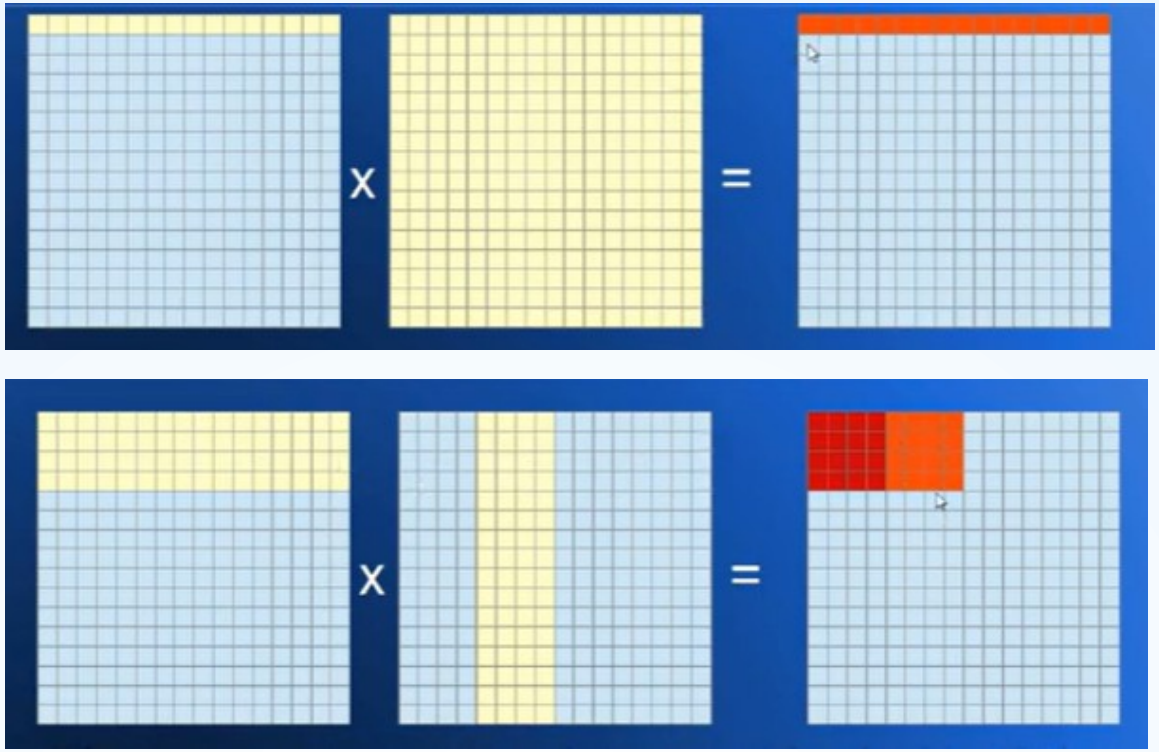
Exemplo: produto de matrizes

- O ideal seria **alterar a alocação** de B para favoreceria o acesso às linhas de cache
- Mas demanda tempo para fazer a realocação (cache menor que MP)



Solução....

- Dividir a cache em **porções menores** (***blocking***) para aproveitar o **reuso de dados** já armazenados



Implicações

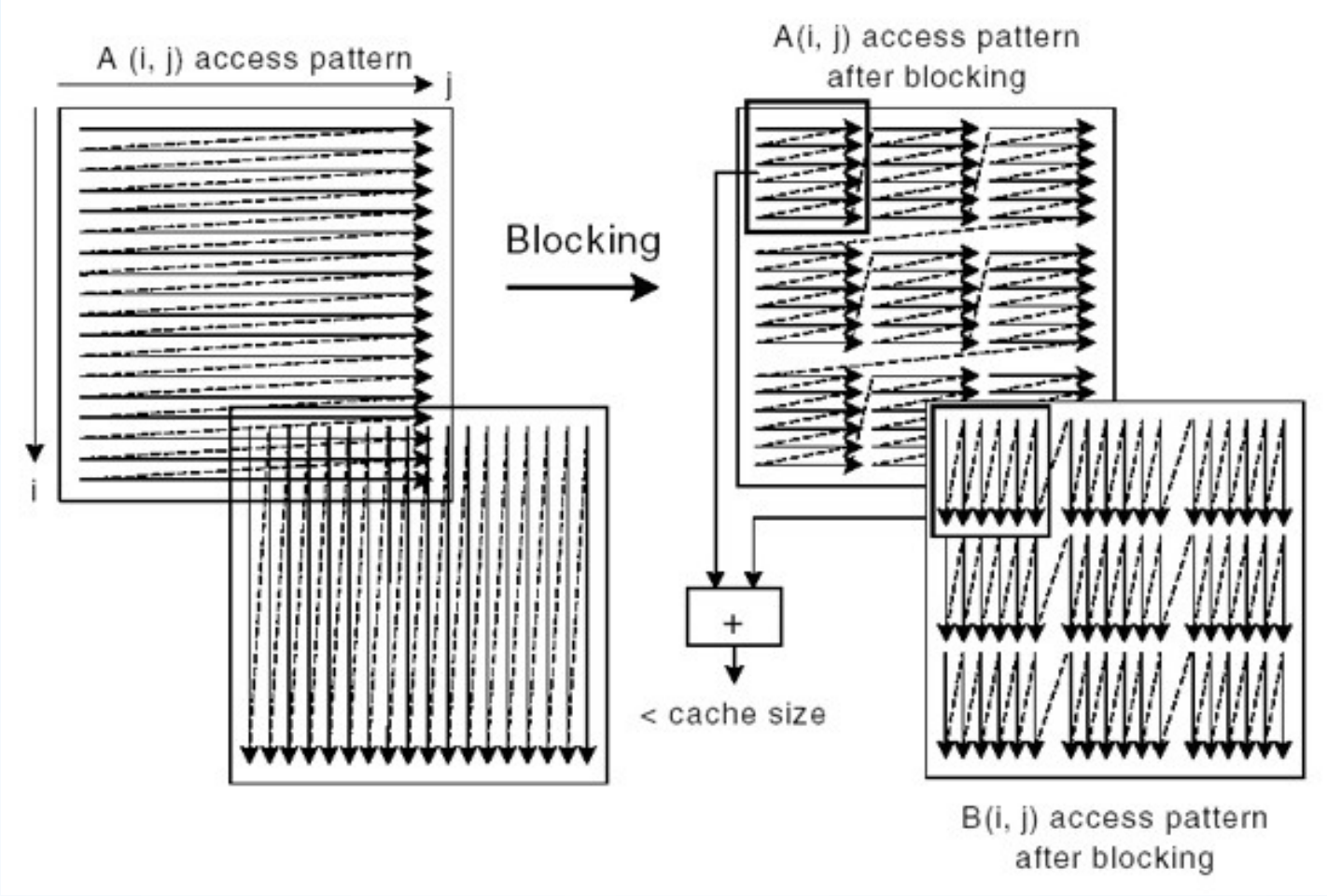
- O software precisa ser alterado
- Geralmente acrescentando **novos loops** com **novas variáveis**

Exemplo 2: soma com transposta

```
1  float A[MAX, MAX], B[MAX, MAX]
2  for (i=0; i< MAX; i++) {
3    for (j=0; j< MAX; j++) {
4      A[i,j] = A[i,j] + B[j, i];
5    }
6  }
```

```
1  float A[MAX, MAX], B[MAX, MAX];
2  for (i=0; i< MAX; i+=block_size) {
3    for (j=0; j< MAX; j+=block_size) {
4      for (ii=i; ii<i+block_size; ii++) {
5        for (jj=j; jj<j+block_size; jj++) {
6          A[ii,jj] = A[ii,jj] + B[jj, ii];
7        }
8      }
9    }
10 }
```

Conceito de Blocking Cache



Influência do tamanho do bloco

- **Falta de Capacidade** → ocorrem quando a **cache** é muito pequena para conter todos os dados usados simultaneamente

- A Falta de Capacidade é uma função de **N** e do **tamanho da cache**:
  - **$2N^3 + N^2$** → assumindo a premissa de que não há nenhum conflito!

Influência do tamanho do bloco

- **Ideia**: calcular na submatriz BxB que se encaixa
- **B** é chamado de ***Blocking Factor***
- A falta de capacidade cai de $\mathbf{2N^3 + N^2}$ para $\mathbf{2N^3/B + N^2}$

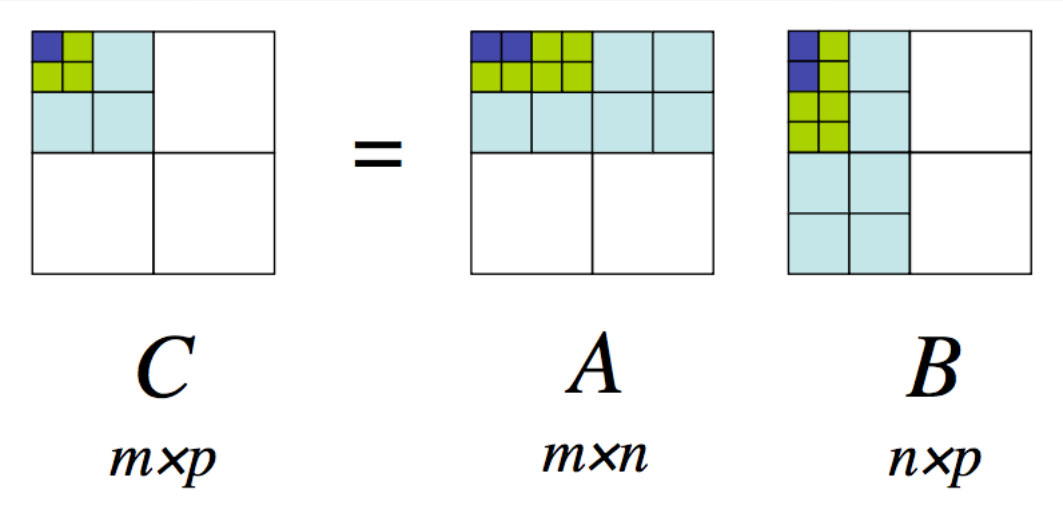
Algoritmo Cache *Oblivious*

- É um algoritmo projetado para tirar proveito da cache de processador sem conhecer o **tamanho da cache** (ou o comprimento das linhas de cache, etc.) como um parâmetro explícito

- É projetado para funcionar bem, **sem modificação**, em **várias máquinas com tamanhos de cache diferentes** ou para uma hierarquia de memória com **diferentes níveis de cache** com **tamanhos diferentes**

Algoritmo Cache *Oblivious*

- **Ideia**: dividir para conquistar!
- Blocos são calculados recursivamente



FIM MÓDULO 18