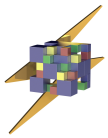
Lecture 22

Optimizations for Caches

이재진
서울대학교 데이터사이언스대학원
서울대학교 공과대학 컴퓨터공학부
http://aces.snu.ac.kr/~jlee



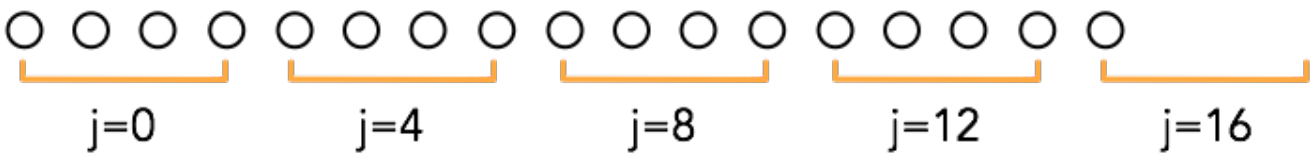
THUNDER Research Group
Seoul National University
서울대학교 천둥 연구실

Strip Mining

- Adjusts the granularity of an operation
  - Break loops into pieces
- Usually for vectorization
  - Vector registers have finite length

```
for (i = 0; i < N; i++) {
    a[i] = b[i] + 3;
}
```

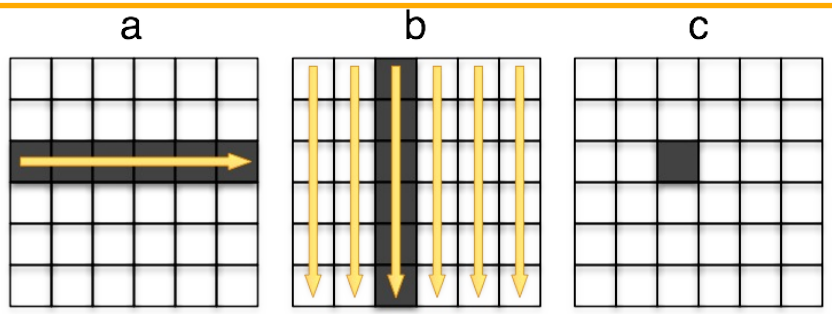
```
K = ceil(N/4)
for (j = 0; j < N; j += K) {
    for (i = j; i < MIN(j + K, N); i++) {
        a[i] = b[i] + 3;
    }
}
```



Tiling (Blocking) for Matrix Multiply

- While using one row of a, the algorithm accesses all the elements of b, column by column
  - Elements of a column are stored among N different cache lines
- The number of cache misses (when the cache cannot hold even one column): $\frac{N^2}{m} + N^3$
  - $\frac{N^2}{m}$ for a (m: number of elements in a cache line)
  - N^3 for b

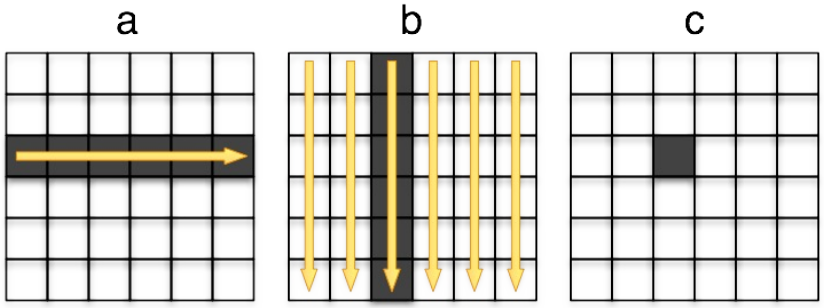
```
for (i = 0; i < N; i++)
    for (j = 0; j < N; j++)
        for (k = 0; k < N; k++)
            c[i][j] += a[i][k] * b[k][j];
```



Tiling for Matrix Multiply (cont'd)

- If the cache is big enough to hold N cache lines, and no other uses of the cache force some of these lines to be evicted, then there will not be another cache misses reading b from j = 0 until j = m

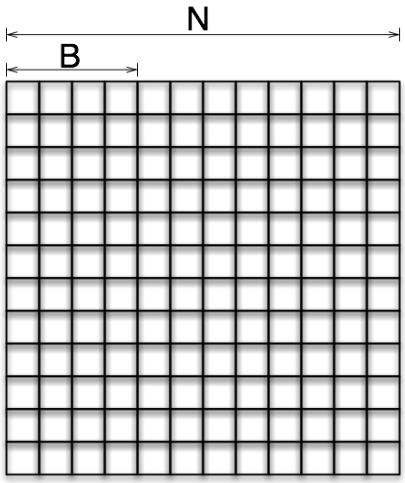
```
for (i = 0; i < N; i++)
    for (j = 0; j < N; j++)
        for (k = 0; k < N; k++)
            c[i][j] += a[i][k] * b[k][j];
```



Tiling for Matrix Multiply (cont'd)

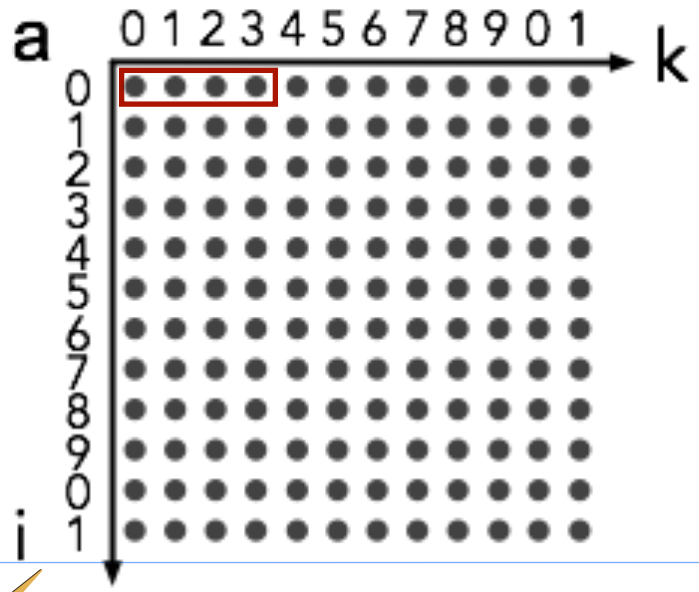
- All of a, b, or c cannot fit in the cache
  - Choose the block size B (the number of elements in a row of B) such that it is possible to fit one block from each of the matrices in the cache
- To improve spatial and temporal locality

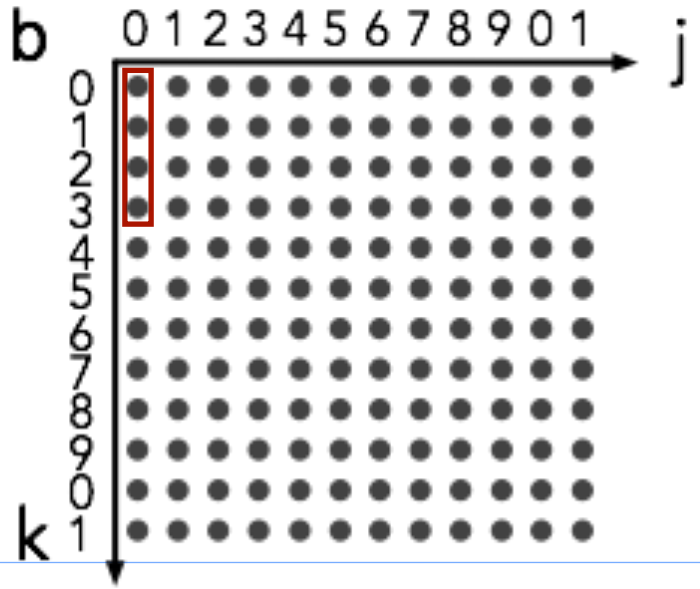
```
for (kk = 0; kk < N; kk += B)
    for (i = 0; i < N; i++)
        for (j = 0; j < N; j++)
            for (k = kk; k < MIN(kk+B, N); k++)
                c[i][j] += a[i][k] * b[k][j];
```



Tiling for Matrix Multiply (cont'd)

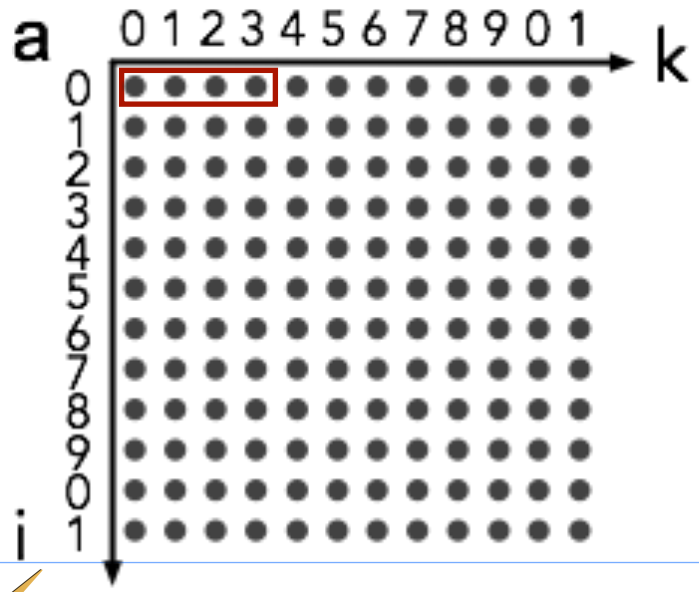
```
for (kk = 0; kk < N; kk += B)
    for (i = 0; i < N; i++)
        for (j = 0; j < N; j++)
            for (k = kk; k < MIN(kk+B, N); k++)
                c[i][j] += a[i][k] * b[k][j];
```

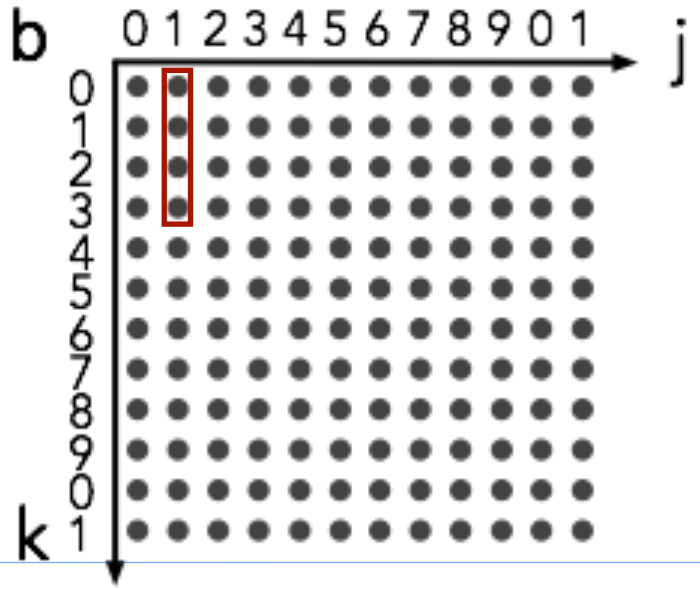




Tiling for Matrix Multiply (cont'd)

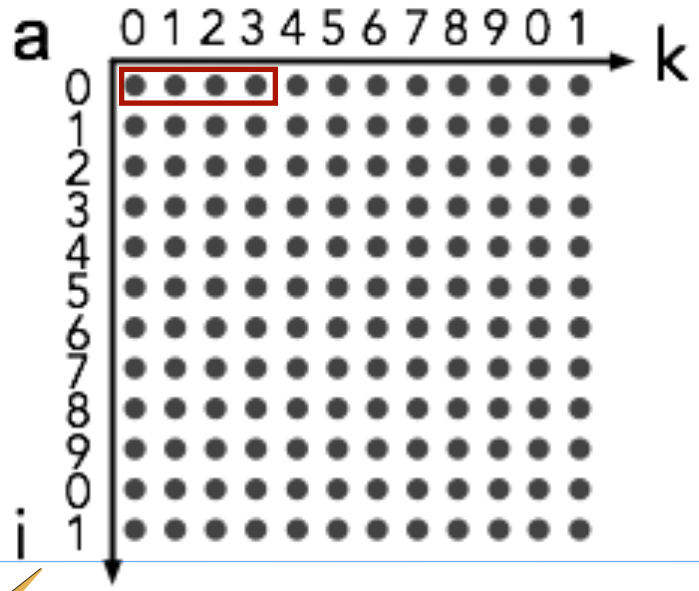
```
for (kk = 0; kk < N; kk += B)
    for (i = 0; i < N; i++)
        for (j = 0; j < N; j++)
            for (k = kk; k < MIN(kk+B, N); k++)
                c[i][j] += a[i][k] * b[k][j];
```

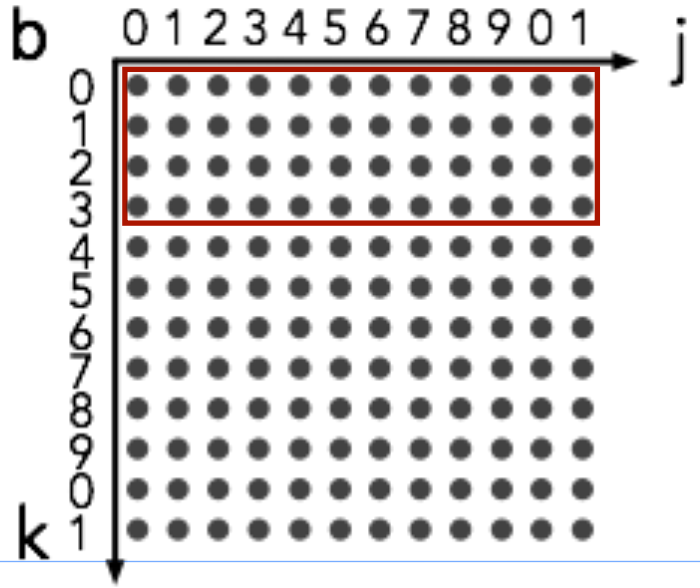




Tiling for Matrix Multiply (cont'd)

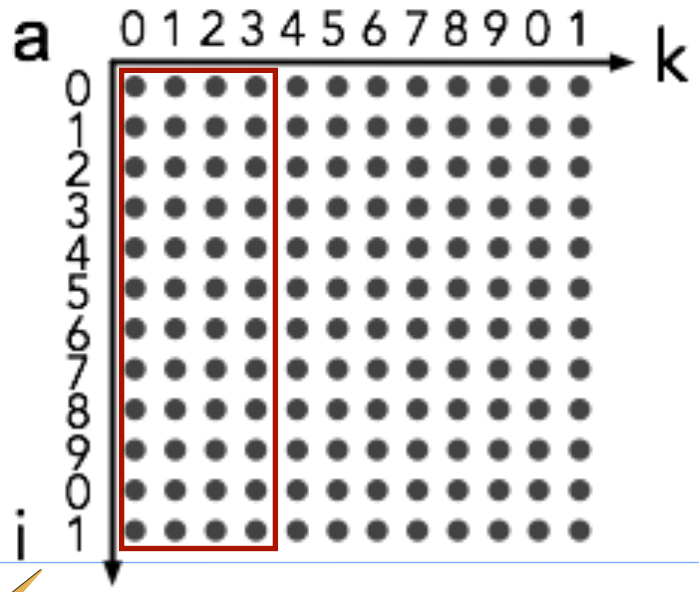
```
for (kk = 0; kk < N; kk += B)
    for (i = 0; i < N; i++)
        for (j = 0; j < N; j++)
            for (k = kk; k < MIN(kk+B, N); k++)
                c[i][j] += a[i][k] * b[k][j];
```

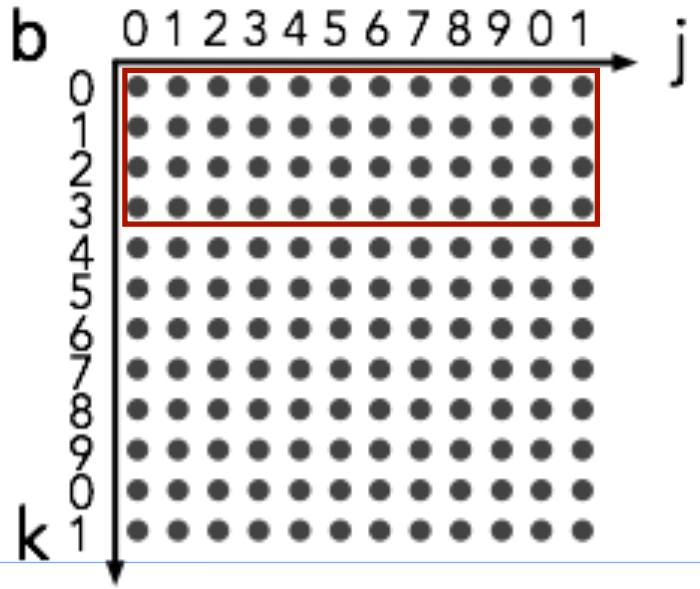




Tiling for Matrix Multiply (cont'd)

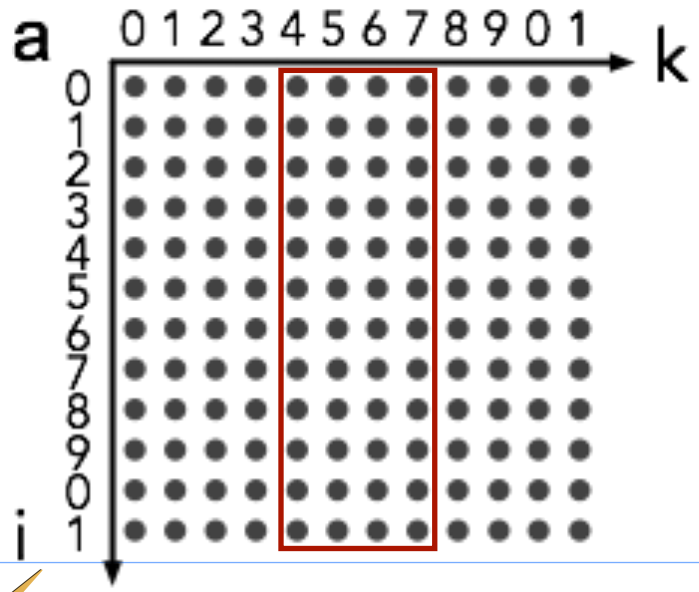
```
for (kk = 0; kk < N; kk += B)
    for (i = 0; i < N; i++)
        for (j = 0; j < N; j++)
            for (k = kk; k < MIN(kk+B, N); k++)
                c[i][j] += a[i][k] * b[k][j];
```

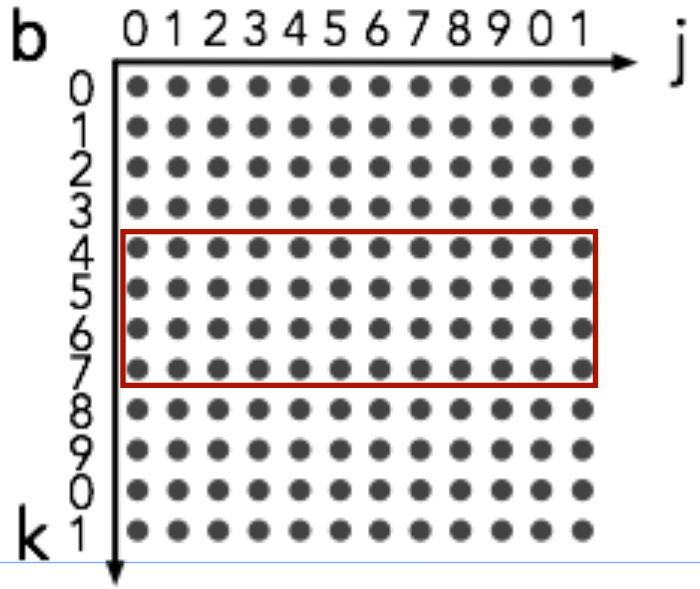




Tiling for Matrix Multiply (cont'd)

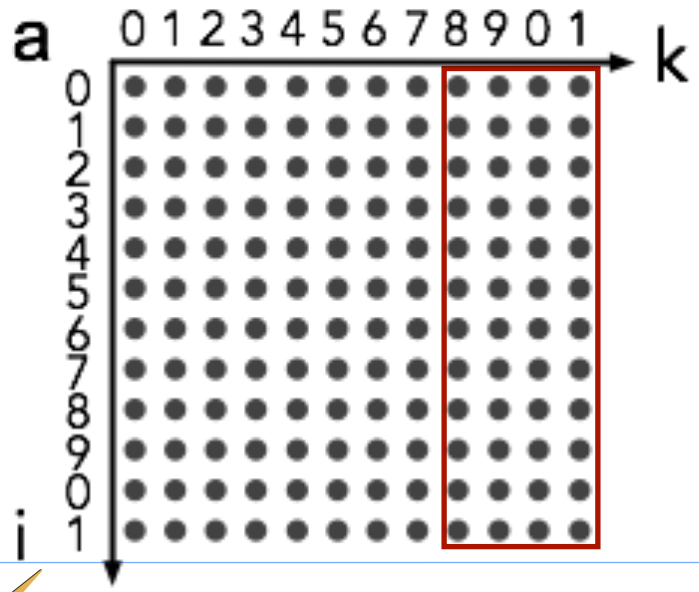
```
for (kk = 0; kk < N; kk += B)
    for (i = 0; i < N; i++)
        for (j = 0; j < N; j++)
            for (k = kk; k < MIN(kk+B, N); k++)
                c[i][j] += a[i][k] * b[k][j];
```

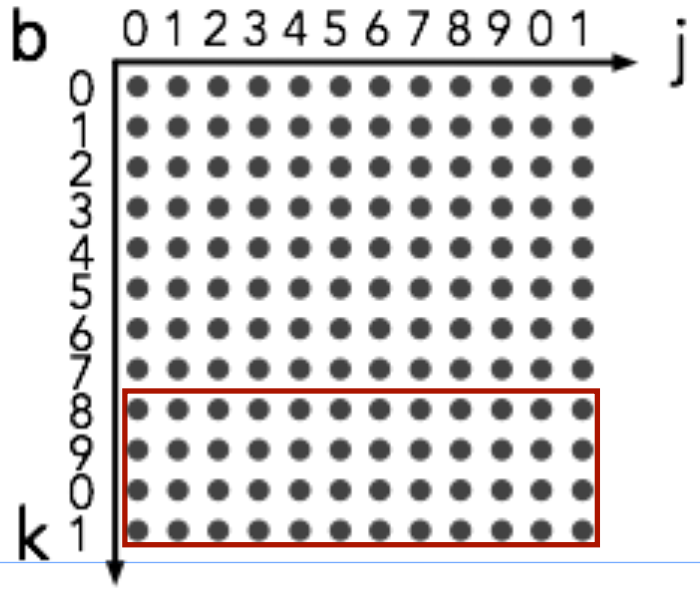




Tiling for Matrix Multiply (cont'd)

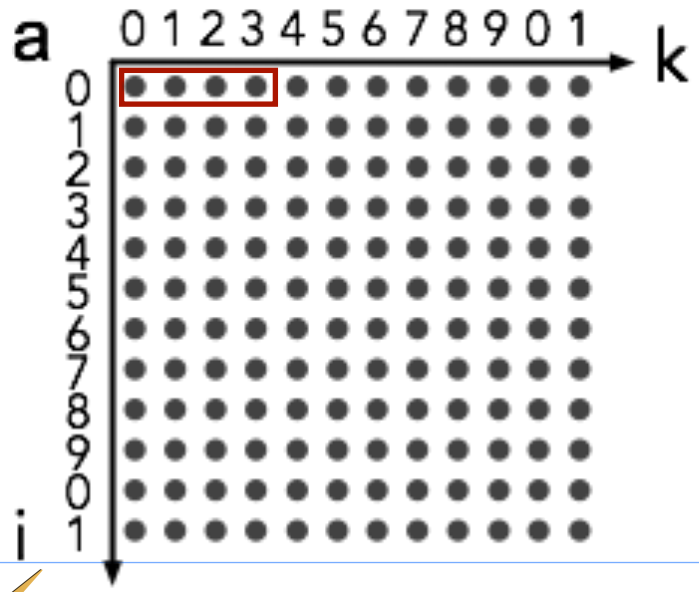
```
for (kk = 0; kk < N; kk += B)
    for (i = 0; i < N; i++)
        for (j = 0; j < N; j++)
            for (k = kk; k < MIN(kk+B, N); k++)
                c[i][j] += a[i][k] * b[k][j];
```

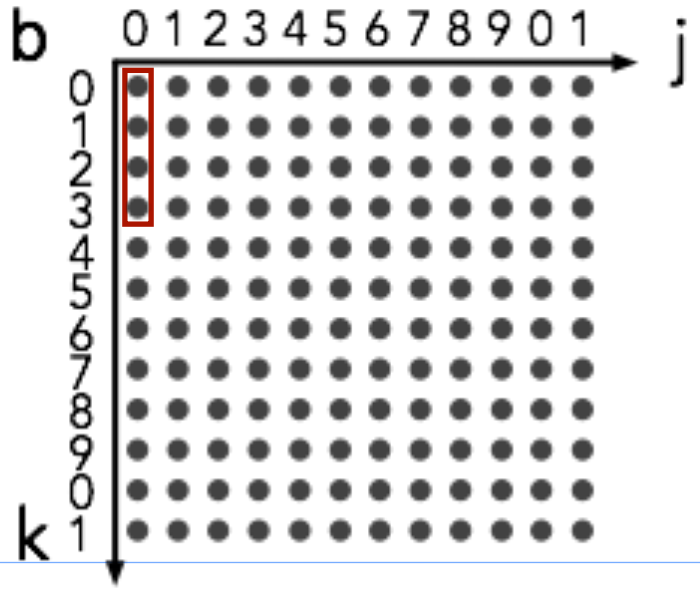




Tiling for Matrix Multiply (cont'd)

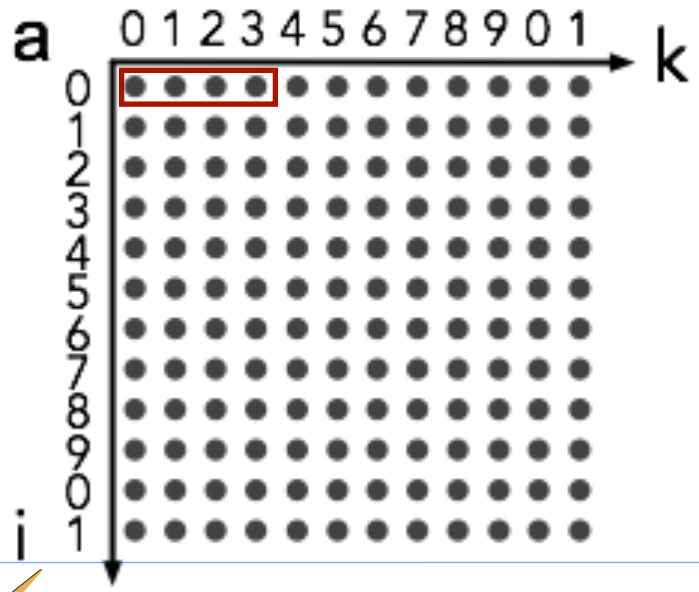
```
for (jj = 0; jj < N; jj += B)
    for (kk = 0; kk < N; kk += B)
        for (i = 0; i < N; i++)
            for (j = jj; j < MIN(jj+B, N); j++)
                for (k = kk; k < MIN(kk+B, N); k++)
                    c[i][j] += a[i][k] * b[k][j];
```

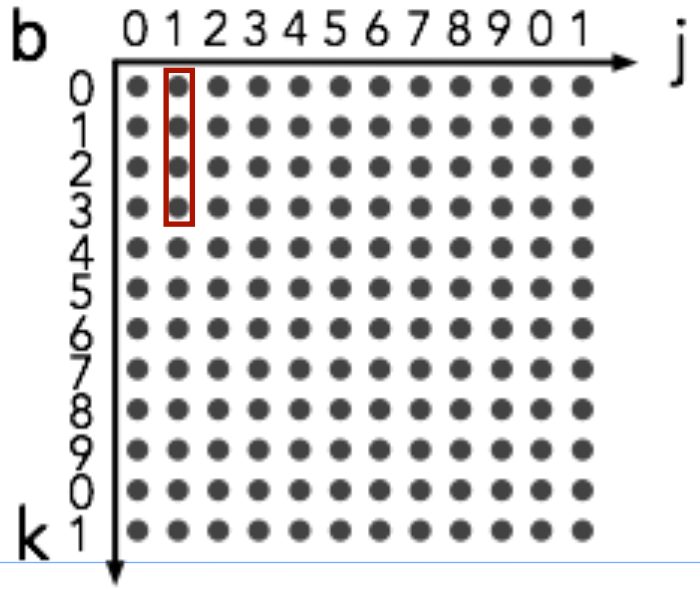




Tiling for Matrix Multiply (cont'd)

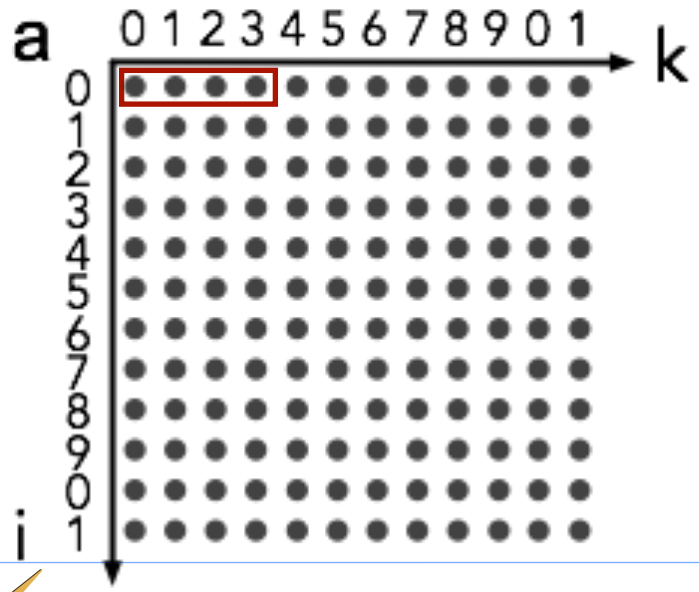
```
for (jj = 0; jj < N; jj += B)
    for (kk = 0; kk < N; kk += B)
        for (i = 0; i < N; i++)
            for (j = jj; j < MIN(jj+B, N); j++)
                for (k = kk; k < MIN(kk+B, N); k++)
                    c[i][j] += a[i][k] * b[k][j];
```

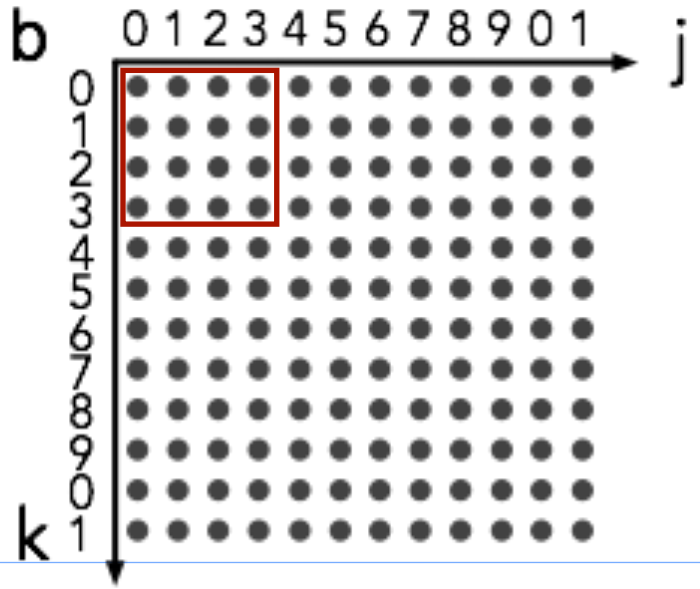




Tiling for Matrix Multiply (cont'd)

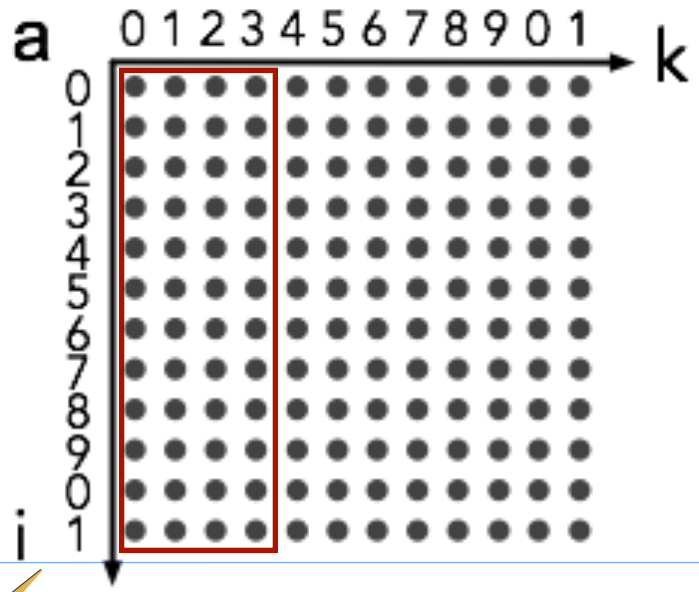
```
for (jj = 0; jj < N; jj += B)
    for (kk = 0; kk < N; kk += B)
        for (i = 0; i < N; i++)
            for (j = jj; j < MIN(jj+B, N); j++)
                for (k = kk; k < MIN(kk+B, N); k++)
                    c[i][j] += a[i][k] * b[k][j];
```

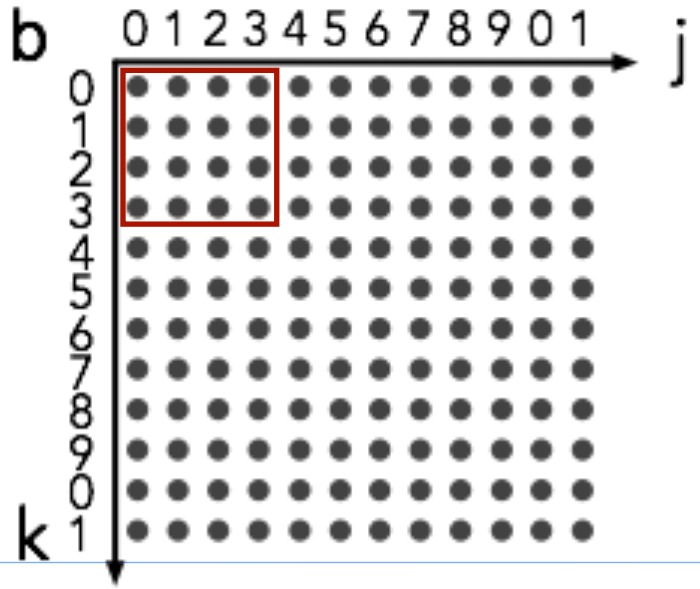




Tiling for Matrix Multiply (cont'd)

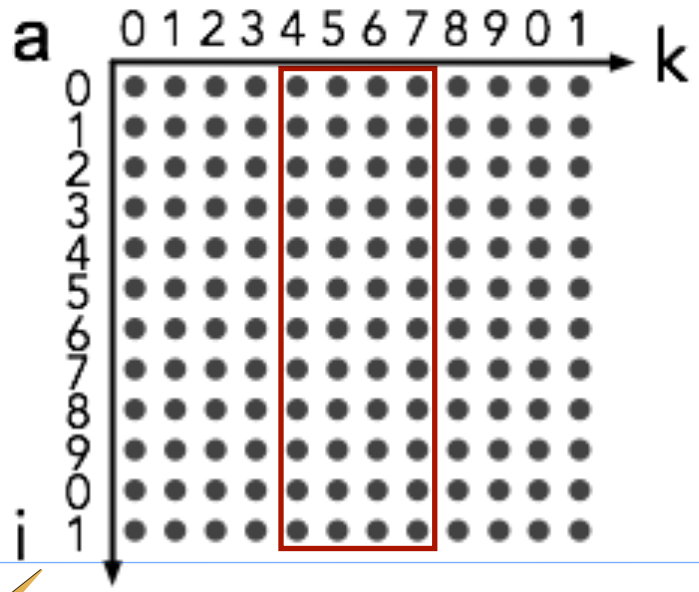
```
for (jj = 0; jj < N; jj += B)
    for (kk = 0; kk < N; kk += B)
        for (i = 0; i < N; i++)
            for (j = jj; j < MIN(jj+B, N); j++)
                for (k = kk; k < MIN(kk+B, N); k++)
                    c[i][j] += a[i][k] * b[k][j];
```

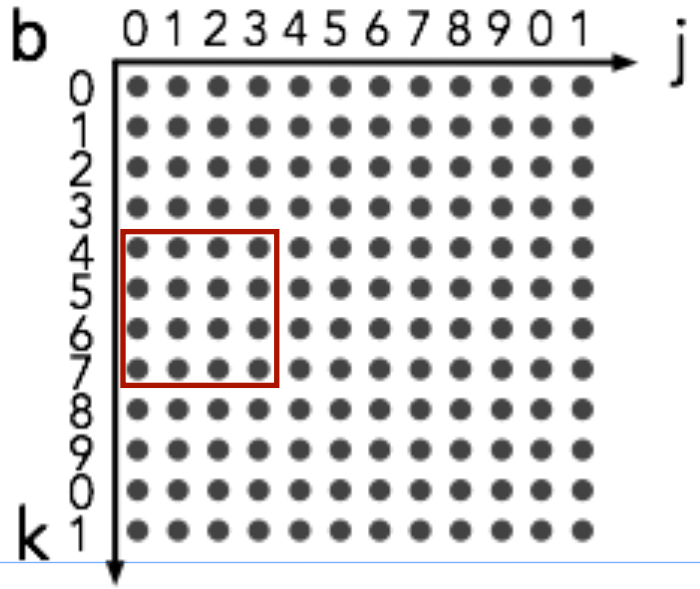




Tiling for Matrix Multiply (cont'd)

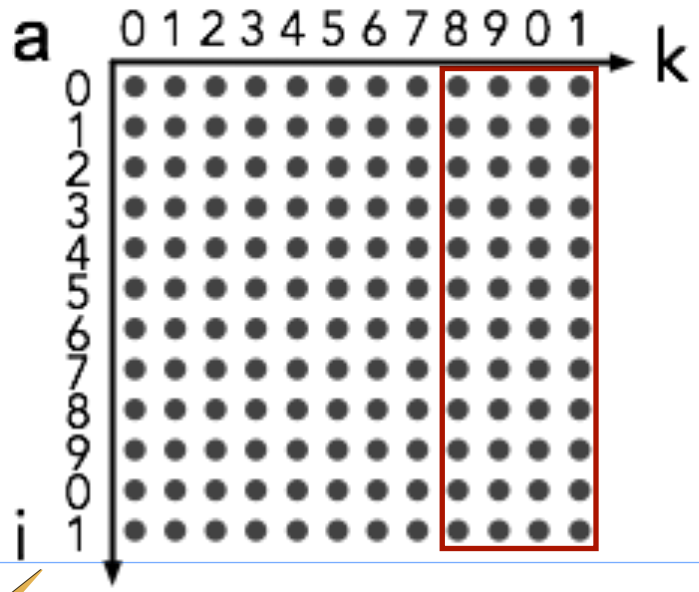
```
for (jj = 0; jj < N; jj += B)
    for (kk = 0; kk < N; kk += B)
        for (i = 0; i < N; i++)
            for (j = jj; j < MIN(jj+B, N); j++)
                for (k = kk; k < MIN(kk+B, N); k++)
                    c[i][j] += a[i][k] * b[k][j];
```

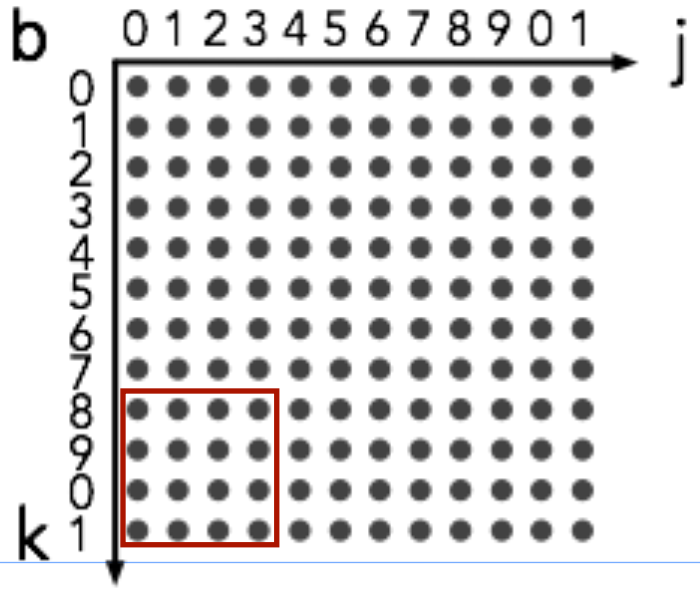




Tiling for Matrix Multiply (cont'd)

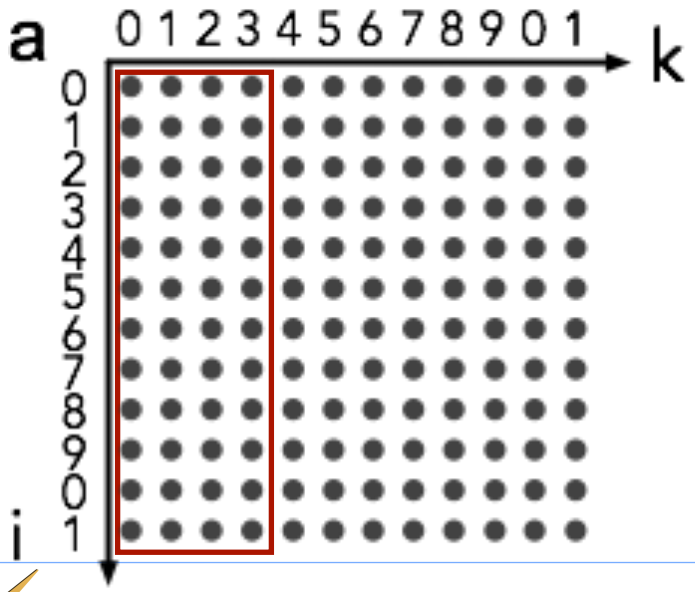
```
for (jj = 0; jj < N; jj += B)
    for (kk = 0; kk < N; kk += B)
        for (i = 0; i < N; i++)
            for (j = jj; j < MIN(jj+B, N); j++)
                for (k = kk; k < MIN(kk+B, N); k++)
                    c[i][j] += a[i][k] * b[k][j];
```

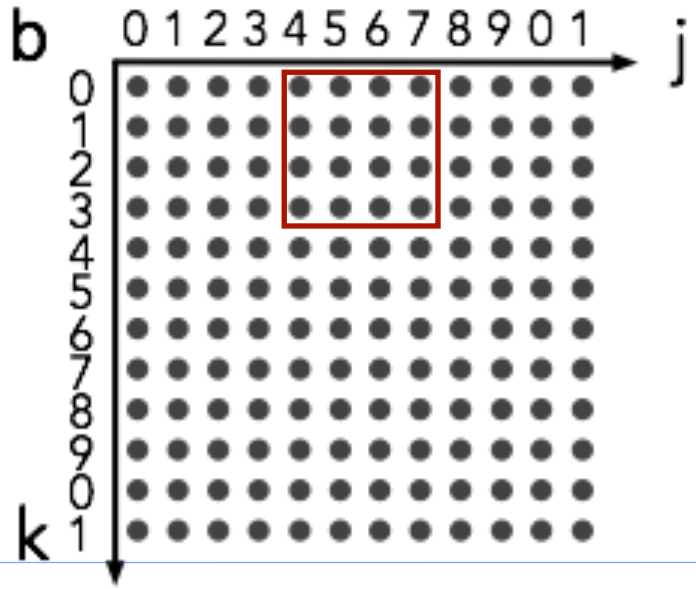




Tiling for Matrix Multiply (cont'd)

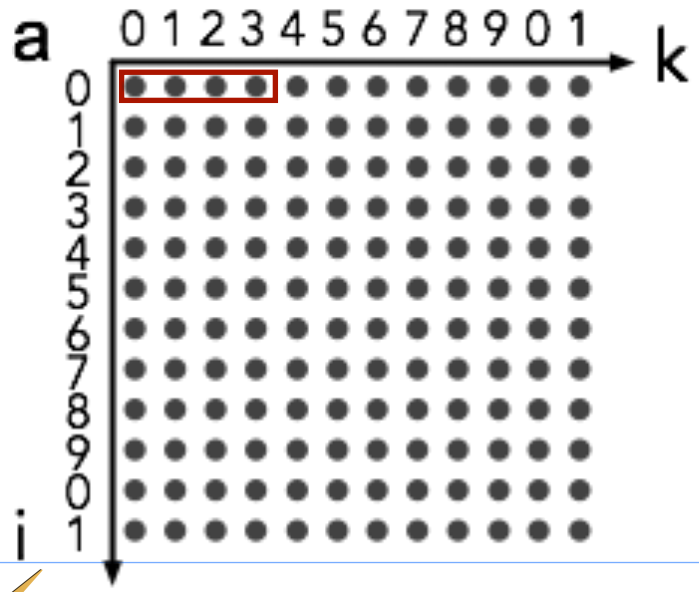
```
for (jj = 0; jj < N; jj += B)
    for (kk = 0; kk < N; kk += B)
        for (i = 0; i < N; i++)
            for (j = jj; j < MIN(jj+B, N); j++)
                for (k = kk; k < MIN(kk+B, N); k++)
                    c[i][j] += a[i][k] * b[k][j];
```

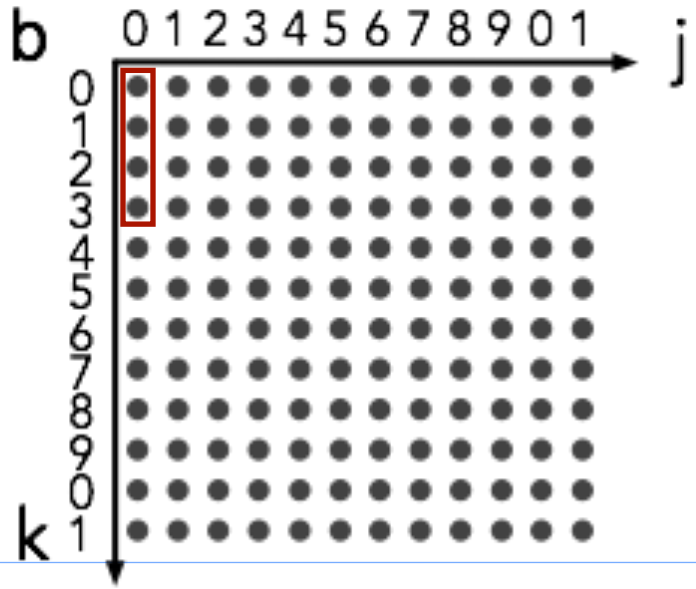




Tiling for Matrix Multiply (cont'd)

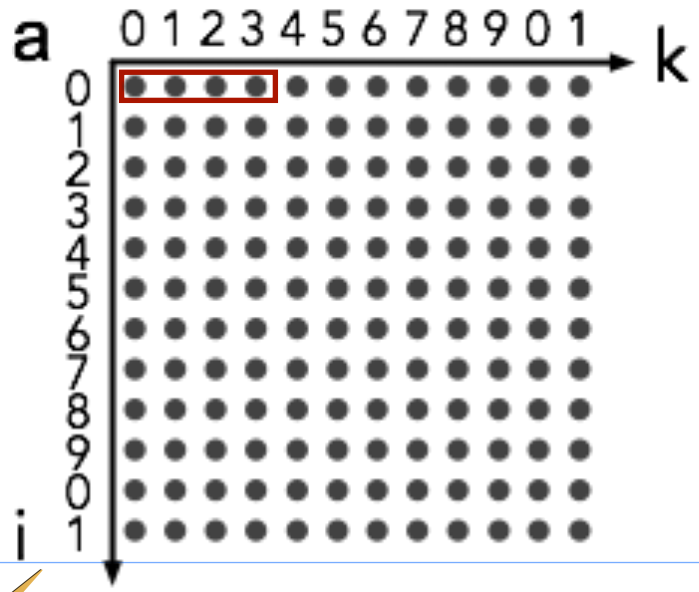
```
for (ii = 0; ii < N; ii += B)
    for (jj = 0; jj < N; jj += B)
        for (kk = 0; kk < N; kk += B)
            for (i = ii; i < MIN(ii+B, N); i++)
                for (j = jj; j < MIN(jj+B, N); j++)
                    for (k = kk; k < MIN(kk+B, N); k++)
                        c[i][j] += a[i][k] * b[k][j];
```

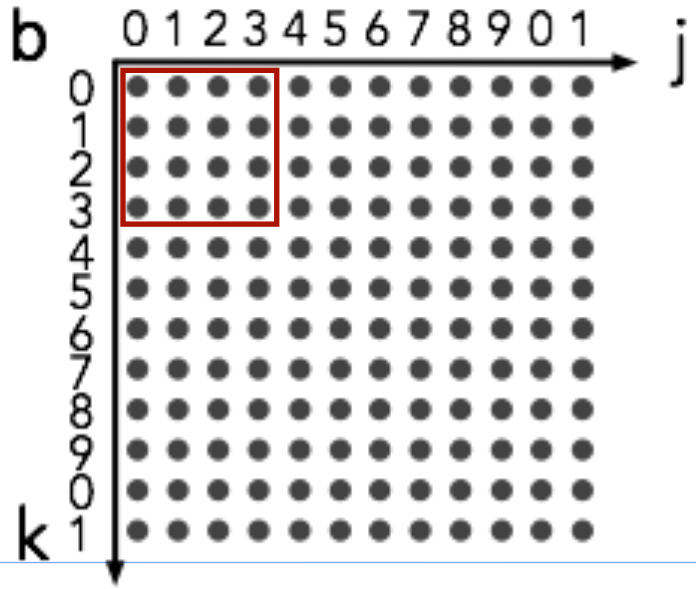




Tiling for Matrix Multiply (cont'd)

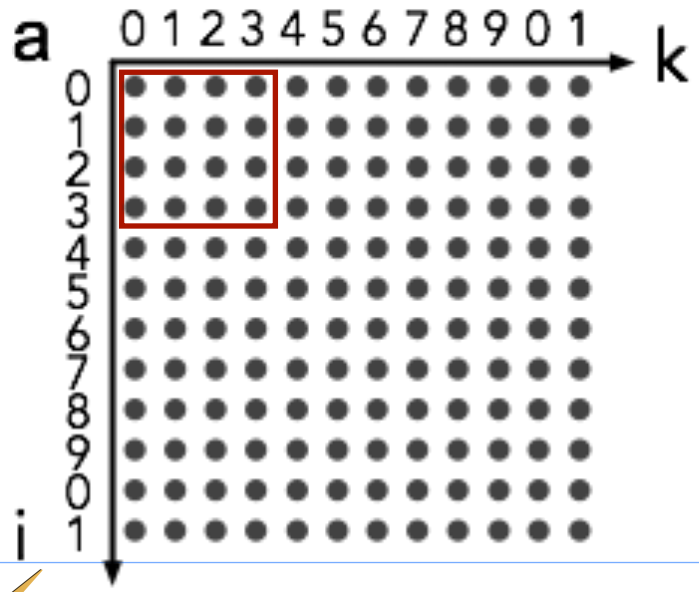
```
for (ii = 0; ii < N; ii += B)
    for (jj = 0; jj < N; jj += B)
        for (kk = 0; kk < N; kk += B)
            for (i = ii; i < MIN(ii+B, N); i++)
                for (j = jj; j < MIN(jj+B, N); j++)
                    for (k = kk; k < MIN(kk+B, N); k++)
                        c[i][j] += a[i][k] * b[k][j];
```

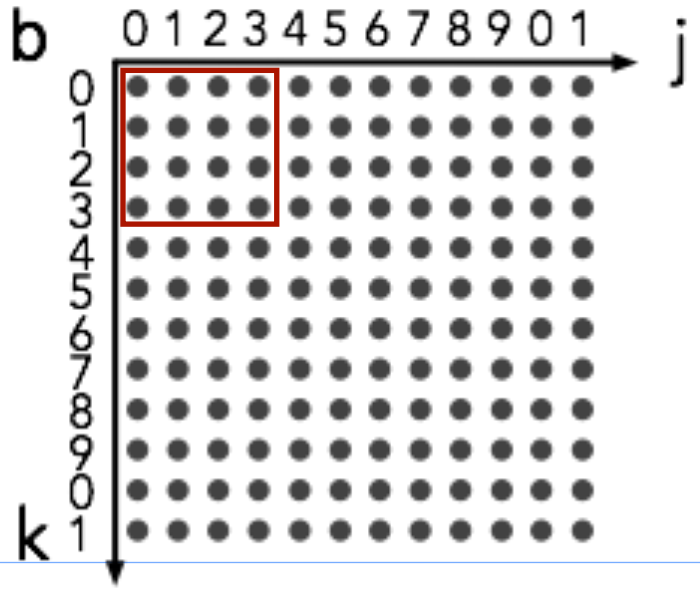




Tiling for Matrix Multiply (cont'd)

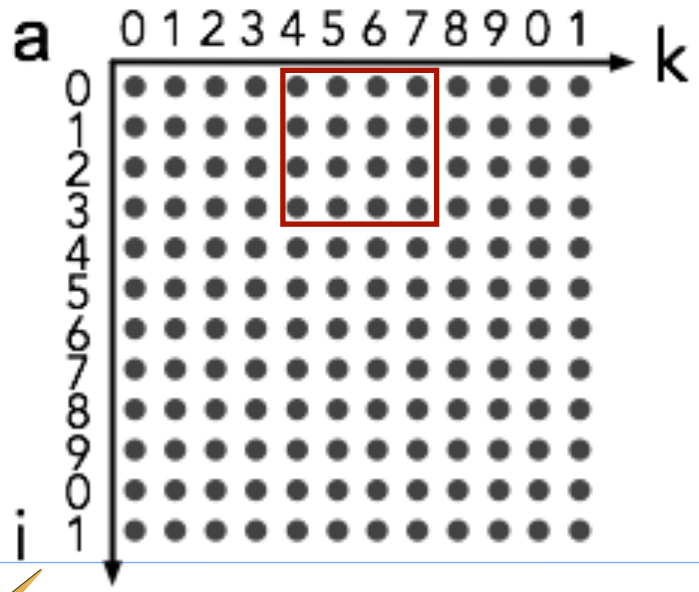
```
for (ii = 0; ii < N; ii += B)
    for (jj = 0; jj < N; jj += B)
        for (kk = 0; kk < N; kk += B)
            for (i = ii; i < MIN(ii+B, N); i++)
                for (j = jj; j < MIN(jj+B, N); j++)
                    for (k = kk; k < MIN(kk+B, N); k++)
                        c[i][j] += a[i][k] * b[k][j];
```

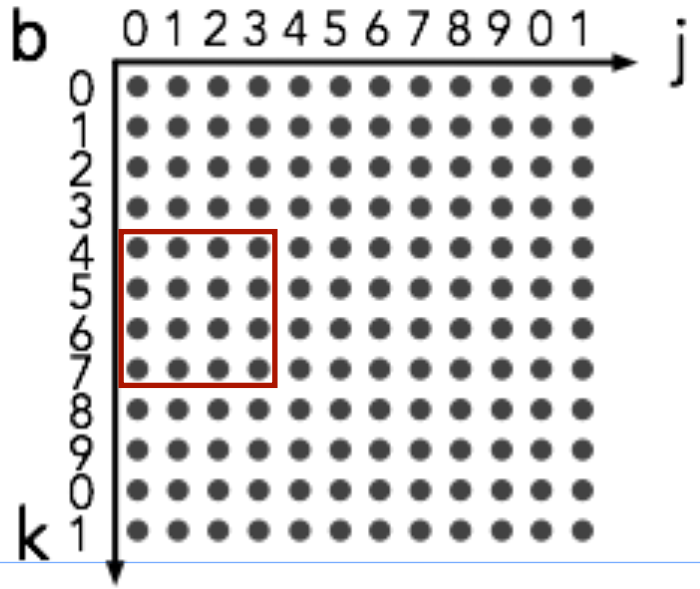




Tiling for Matrix Multiply (cont'd)

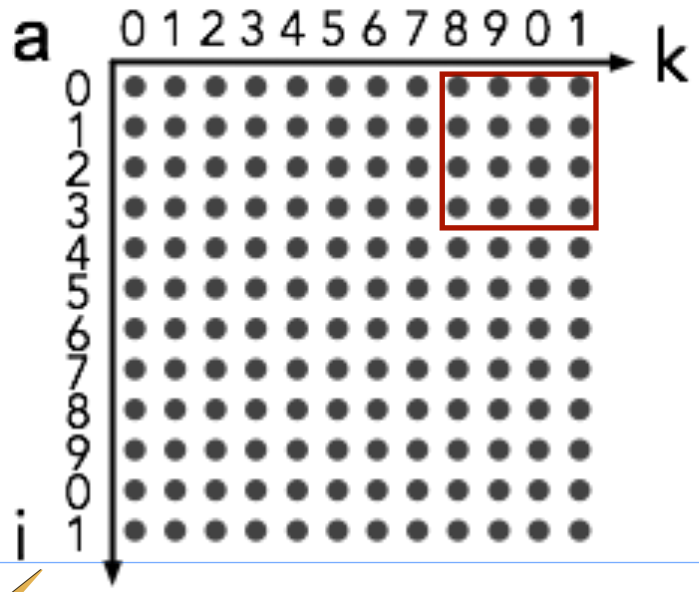
```
for (ii = 0; ii < N; ii += B)
    for (jj = 0; jj < N; jj += B)
        for (kk = 0; kk < N; kk += B)
            for (i = ii; i < MIN(ii+B, N); i++)
                for (j = jj; j < MIN(jj+B, N); j++)
                    for (k = kk; k < MIN(kk+B, N); k++)
                        c[i][j] += a[i][k] * b[k][j];
```

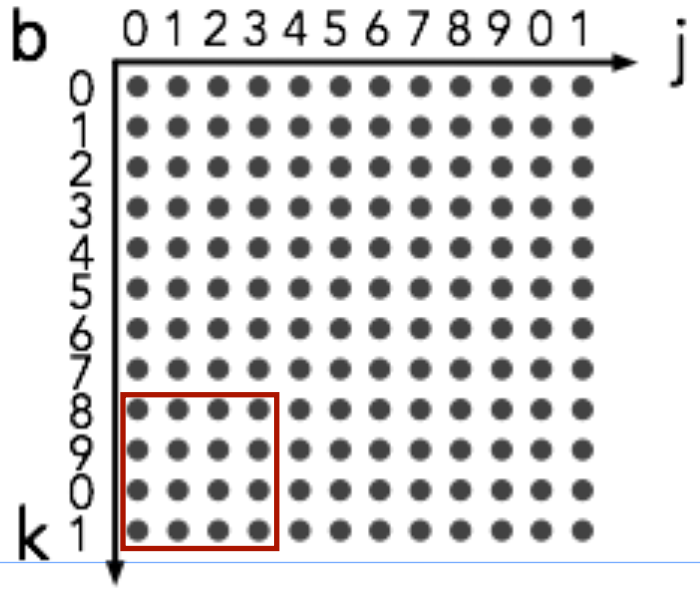




Tiling for Matrix Multiply (cont'd)

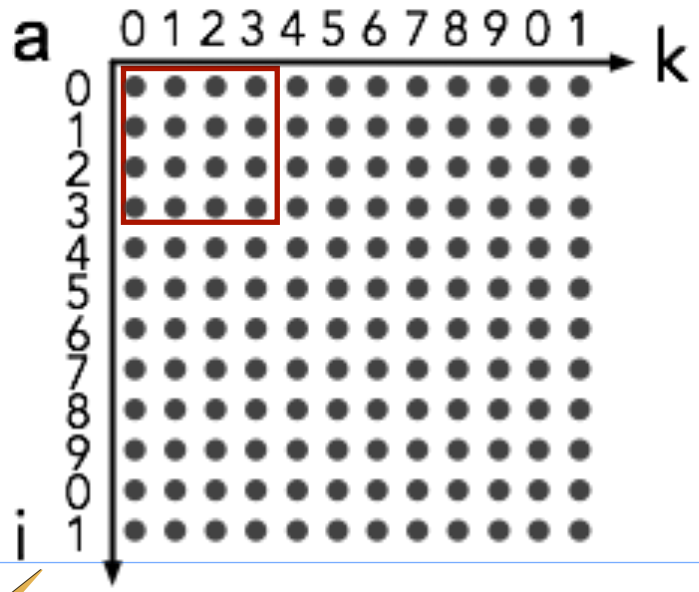
```
for (ii = 0; ii < N; ii += B)
    for (jj = 0; jj < N; jj += B)
        for (kk = 0; kk < N; kk += B)
            for (i = ii; i < MIN(ii+B, N); i++)
                for (j = jj; j < MIN(jj+B, N); j++)
                    for (k = kk; k < MIN(kk+B, N); k++)
                        c[i][j] += a[i][k] * b[k][j];
```

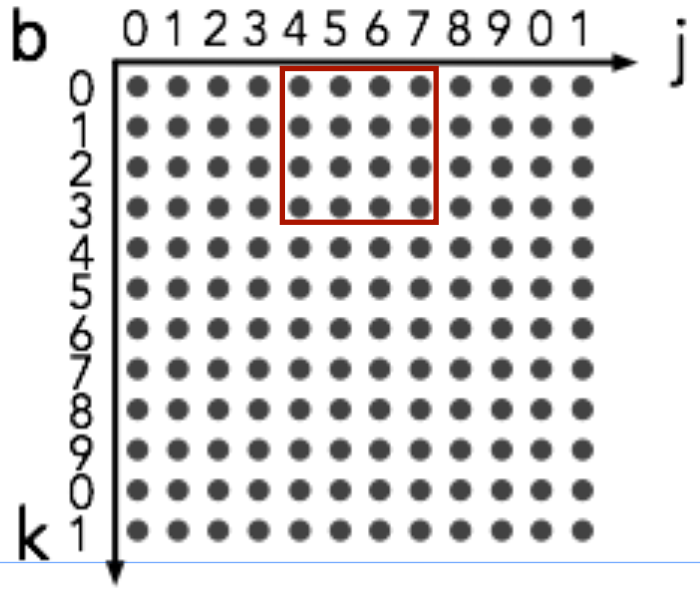




Tiling for Matrix Multiply (cont'd)

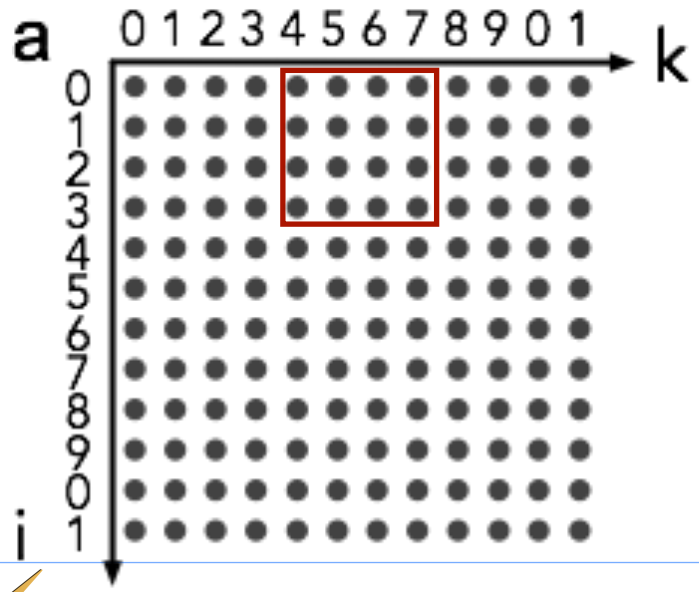
```
for (ii = 0; ii < N; ii += B)
    for (jj = 0; jj < N; jj += B)
        for (kk = 0; kk < N; kk += B)
            for (i = ii; i < MIN(ii+B, N); i++)
                for (j = jj; j < MIN(jj+B, N); j++)
                    for (k = kk; k < MIN(kk+B, N); k++)
                        c[i][j] += a[i][k] * b[k][j];
```

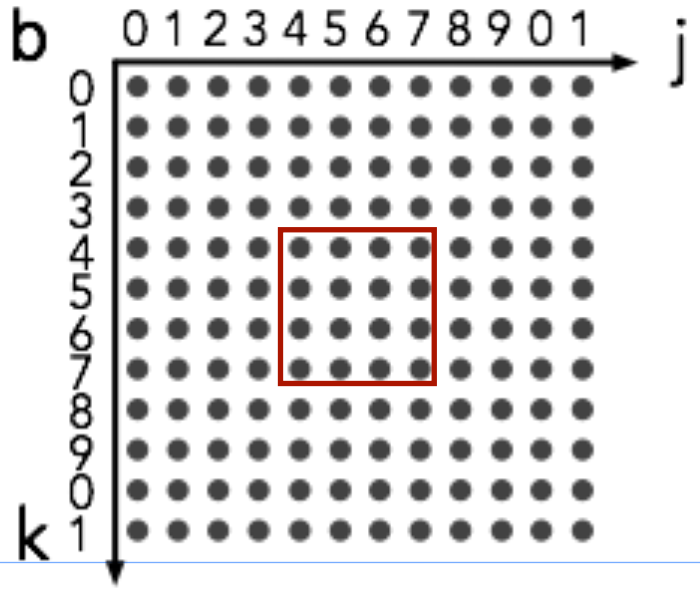




Tiling for Matrix Multiply (cont'd)

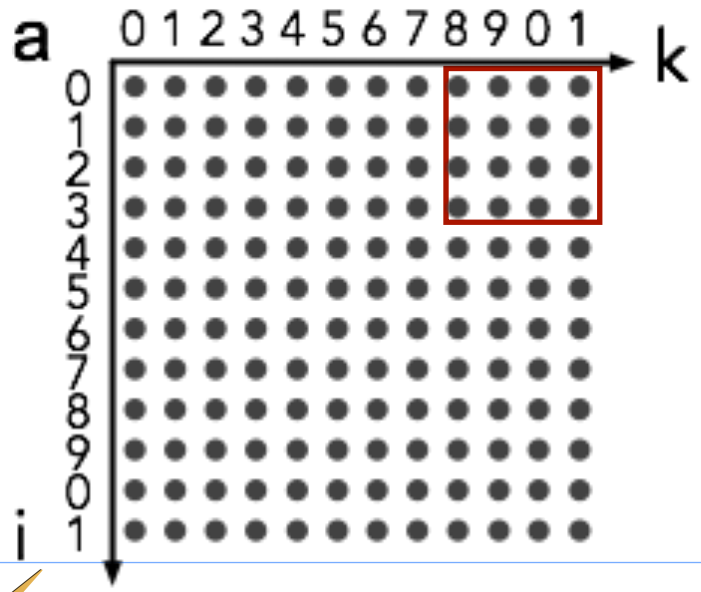
```
for (ii = 0; ii < N; ii += B)
    for (jj = 0; jj < N; jj += B)
        for (kk = 0; kk < N; kk += B)
            for (i = ii; i < MIN(ii+B, N); i++)
                for (j = jj; j < MIN(jj+B, N); j++)
                    for (k = kk; k < MIN(kk+B, N); k++)
                        c[i][j] += a[i][k] * b[k][j];
```

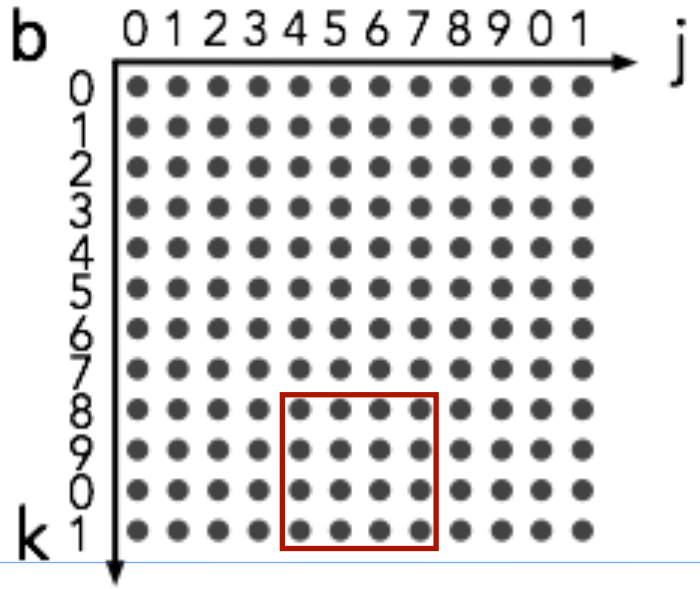




Tiling for Matrix Multiply (cont'd)

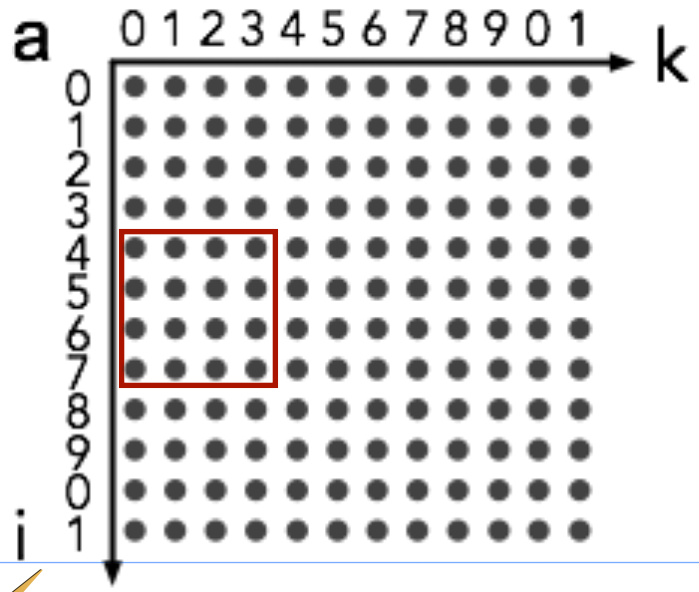
```
for (ii = 0; ii < N; ii += B)
    for (jj = 0; jj < N; jj += B)
        for (kk = 0; kk < N; kk += B)
            for (i = ii; i < MIN(ii+B, N); i++)
                for (j = jj; j < MIN(jj+B, N); j++)
                    for (k = kk; k < MIN(kk+B, N); k++)
                        c[i][j] += a[i][k] * b[k][j];
```

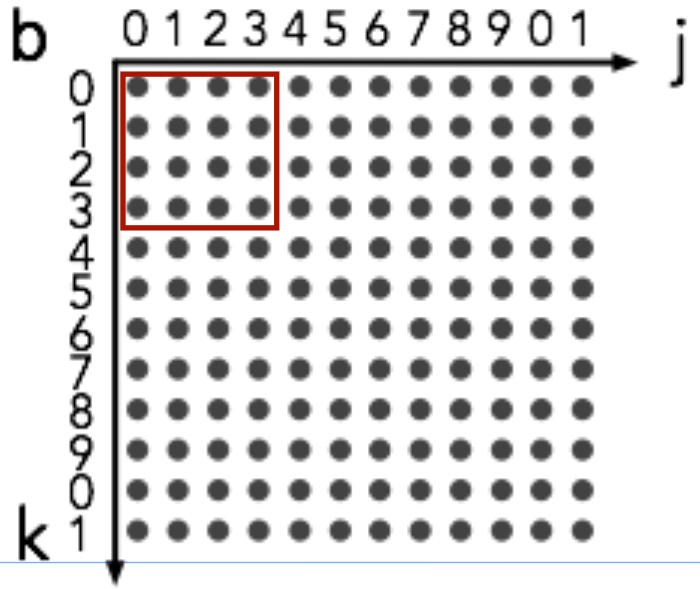




Tiling for Matrix Multiply (cont'd)

```
for (ii = 0; ii < N; ii += B)
    for (jj = 0; jj < N; jj += B)
        for (kk = 0; kk < N; kk += B)
            for (i = ii; i < MIN(ii+B, N); i++)
                for (j = jj; j < MIN(jj+B, N); j++)
                    for (k = kk; k < MIN(kk+B, N); k++)
                        c[i][j] += a[i][k] * b[k][j];
```





Tiling for Matrix Multiply (cont'd)

- Sub-blocks (Aij) can be treated just like scalars

$$\begin{matrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{matrix} \times \begin{matrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{matrix} = \begin{matrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{matrix}$$

$$C_{11} = A_{11}B_{11} + A_{12}B_{21} \qquad C_{12} = A_{11}B_{12} + A_{12}B_{22}$$
$$C_{21} = A_{21}B_{11} + A_{22}B_{21} \qquad C_{22} = A_{21}B_{12} + A_{22}B_{22}$$

```
for (ii = 0; ii < N; ii += B)
    for (jj = 0; jj < N; jj += B)
        for (kk = 0; kk < N; kk += B)
            for (i = ii; i < MIN(ii+B, N); i++)
                for (j = jj; j < MIN(jj+B, N); j++)
                    for (k = kk; k < MIN(kk+B, N); k++)
                        c[i][j] += a[i][k] * b[k][j];
```

Tiling for Matrix Multiply (cont'd)

- Total number of misses is $\frac{2N^3}{Bm}$
- To bring a block of a or b to the cache takes $\frac{N^2}{m}$ cache misses
  - The number of times we bring a pair of blocks to the cache is $\frac{N^3}{B^3}$
  - $\frac{2B^2}{m}$ cache misses per pair
- $\frac{N^2}{m} + N^3$ vs. $\frac{2N^3}{Bm}$