Lecture 15

Optimizations for Caches

이재진

서울대학교 컴퓨터공학부

http://aces.snu.ac.kr

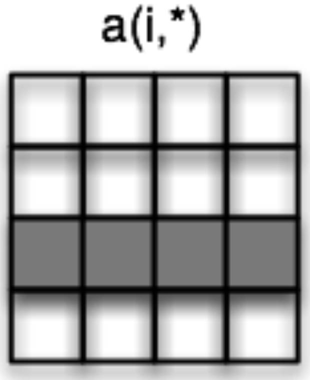
THUNDER Research Group
Seoul National University
서울대학교 천둥 연구실

Effect of Data Access Patterns

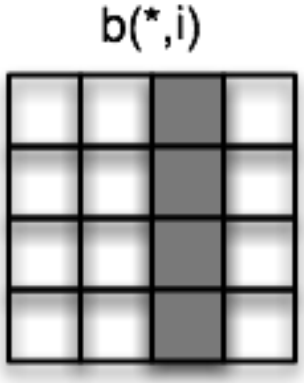
- Execution time
  - Access pattern 1: 1.15 sec
  - Access pattern 2: 9.34 sec

```
for(k = 0; k < ITER; k++)
   for(i = 0; i < SIZE; i++)
      for(j = 0; j < SIZE; j++)
         a[i][j] = a[i][j] + 1;
```

```
for(k = 0; k < ITER; k++)
   for(i = 0; i < SIZE; i++)
      for(j = 0; j < SIZE; j++)
         b[j][i] = b[j][i] + 1;
```



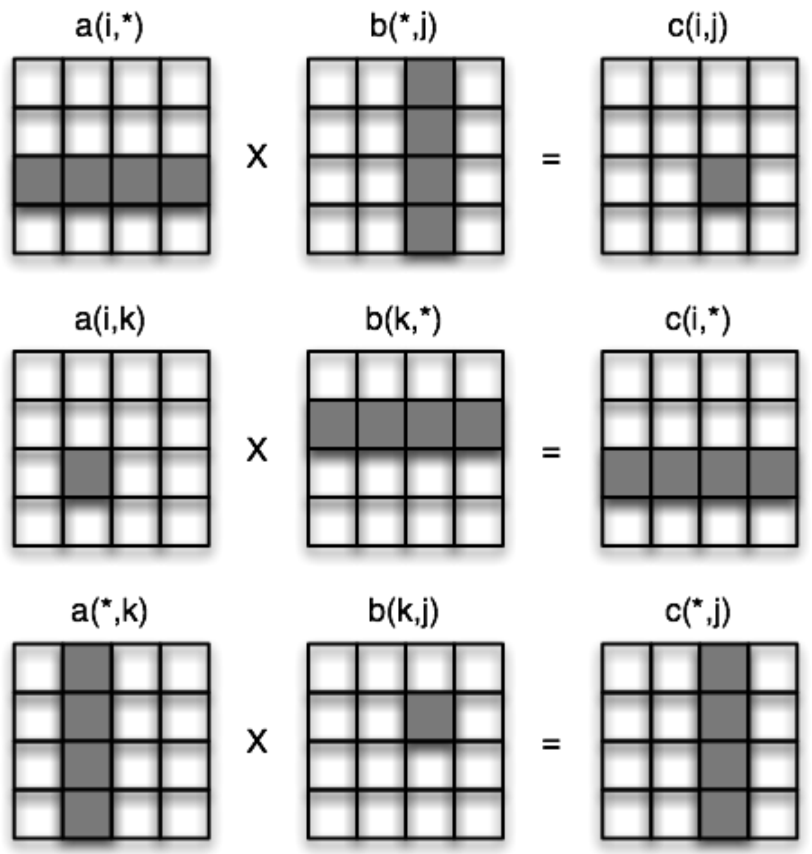
Pattern 1



Pattern 2

Access Patterns in Matrix Multiply

- Compare the execution times of the ijk, kij, and jki



Matrix Multiply

- Need to consider
  - Total cache size
    - Exploit temporal locality and keep the working set small
  - Cache block size
    - Exploit spatial locality
- Multiply N x N matrices
  - $O(N^3)$ total operations

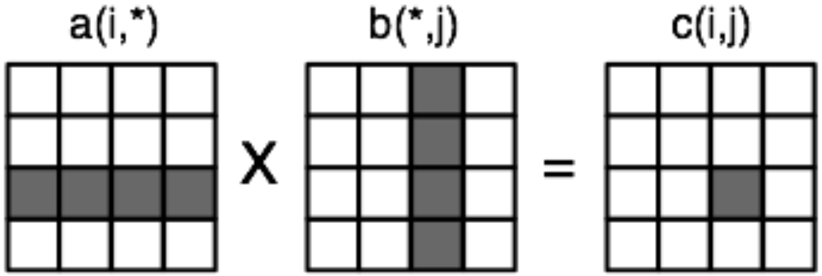
```
/* ijk */
for (i=0; i<n; i++) {
  for (j=0; j<n; j++) {
    sum = 0.0;
    for (k=0; k<n; k++)
      sum += a[i][k] * b[k][j];
    c[i][j] = sum;
  }
}
```

Reuse and Locality

- Reuse
    - Accessing a location that has been accessed in the past
- Locality
    - Accessing a location that is currently found in the cache
- Locality only occurs when there is reuse
- However, reuse does not necessarily result in locality

Miss Rate Analysis for Matrix Multiply

- Assume:
  - Line size = L bytes
  - Word size = W bytes
  - Matrix dimension (N) is very large
  - Cache is not even big enough to hold multiple rows
- Focus on the access pattern of the innermost loop



```
/* ijk */
for (i=0; i<n; i++){
  for (j=0; j<n; j++){
    sum = 0.0;
    for (k=0; k<n; k++)
      sum += a[i][k] * b[k][j];
    c[i][j] = sum;
  }
}
```

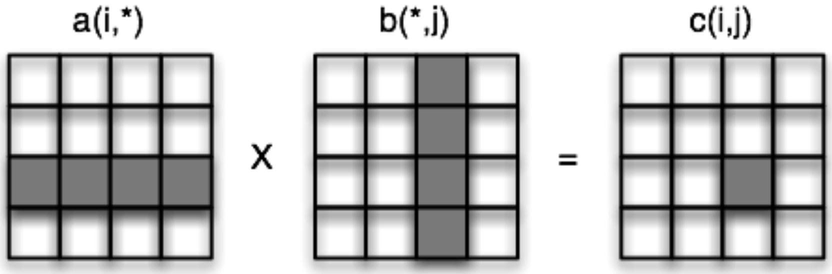
Layout of C Arrays in Memory

- C arrays allocated in row-major order
  - Each row in a matrix in contiguous memory locations
- Stepping through columns in one row
  - Accesses successive elements
  - If $L > W$, exploit spatial locality
    - Compulsory miss rate = W / L
- Stepping through rows in one column
  - No spatial locality
    - Compulsory miss rate = 1

Matrix Multiplication (ijk)

- Misses per inner loop iteration:
  - a: 0.25
  - b: 1.0
  - c: 0.0

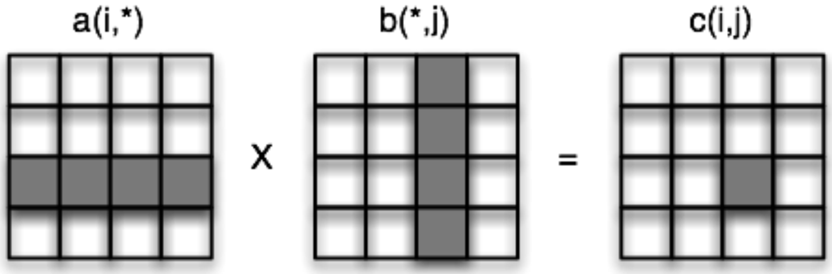
```
/* ijk */
for (i=0; i<n; i++){
  for (j=0; j<n; j++){
    sum = 0.0;
    for (k=0; k<n; k++)
      sum += a[i][k] * b[k][j];
    c[i][j] = sum;
  }
}
```



Matrix Multiplication (*jik*)

- Misses per inner loop iteration:
  - a: 0.25
  - b: 1.0
  - c: 0.0

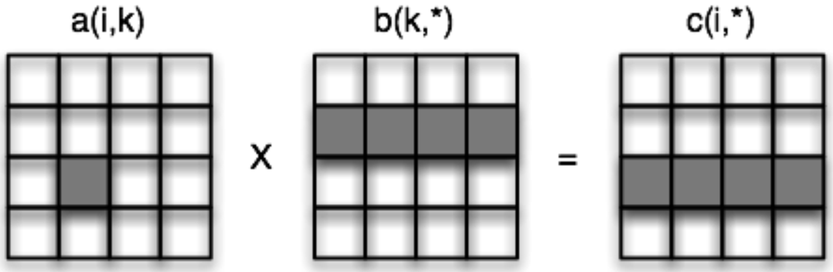
```
/* jik */
for (j=0; j<n; j++){
  for (i=0; i<n; i++){
    sum = 0.0;
    for (k=0; k<n; k++)
      sum += a[i][k] * b[k][j];
    c[i][j] = sum;
  }
}
```



Matrix Multiplication (kij)

- Misses per inner loop iteration:
  - a: 0.0
  - b: 0.25
  - c: 0.25

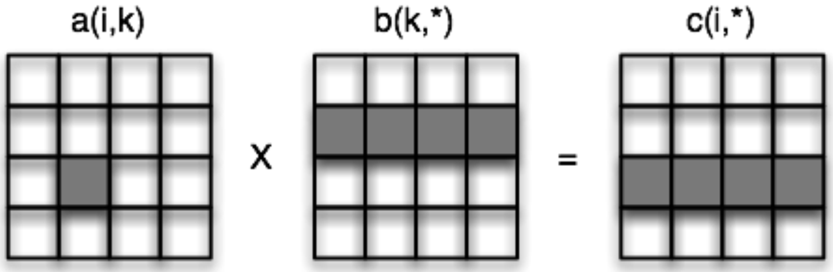
```
/* kij */
for (k=0; k<n; k++){
  for (i=0; i<n; i++){
    r = a[i][k];
    for (j=0; j<n; j++)
      c[i][j] += r * b[k][j];
  }
}
```



Matrix Multiplication (ikj)

- Misses per inner loop iteration:
  - a: 0.0
  - b: 0.25
  - c: 0.25

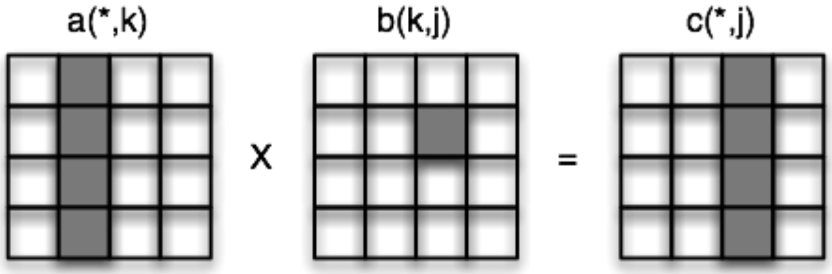
```
/* ikj */
for (i=0; i<n; i++){
  for (k=0; k<n; k++){
    r = a[i][k];
    for (j=0; j<n; j++)
      c[i][j] += r * b[k][j];
  }
}
```



Matrix Multiplication (jki)

- Misses per inner loop iteration:
  - a: 1.0
  - b: 0.0
  - c: 1.0

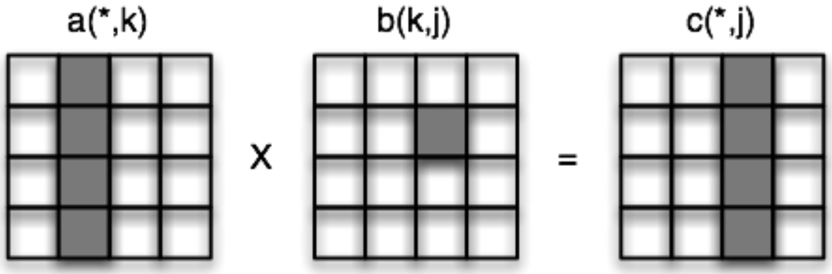
```
/* jki */
for (j=0; j<n; j++){
  for (k=0; k<n; k++){
    r = a[k][j];
    for (i=0; i<n; i++)
      c[i][j] += a[i][k] * r;
  }
}
```



Matrix Multiplication (kji)

- Misses per inner loop iteration:
  - a: 1.0
  - b: 0.0
  - c: 1.0

```
/* kji */
for (k=0; k<n; k++){
  for (j=0; j<n; j++){
    r = a[k][j];
    for (i=0; i<n; i++)
      c[i][j] += a[i][k] * r;
  }
}
```



Summary of Matrix Multiply

ijk (& jik)

2 loads, 0 stores

misses/iter = 1.25

```
for (i=0; i<n; i++)  {
  for (j=0; j<n; j++) {
    sum = 0.0;
    for (k=0; k<n; k++)
      sum += a[i][k] * b[k][j];
    c[i][j] = sum;
  }
}
```

kij (& ikj)

2 loads, 1 store

misses/iter = 0.5

```
for (k=0; k<n; k++) {
  for (i=0; i<n; i++) {
    r = a[i][k];
    for (j=0; j<n; j++)
      c[i][j] += r * b[k][j];
  }
}
```

jki (& kji)

2 loads, 1 store

misses/iter = 2.0

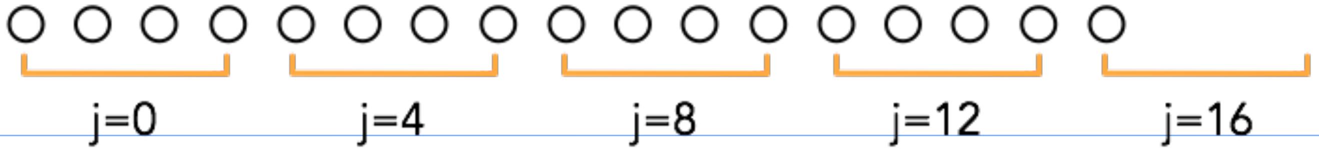
```
for (j=0; j<n; j++) {
  for (k=0; k<n; k++) {
    r = b[k][j];
    for (i=0; i<n; i++)
      c[i][j] += a[i][k] * r;
  }
}
```

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;
}
```

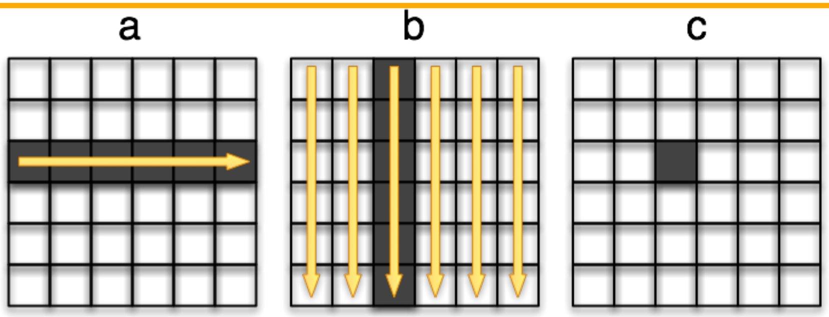
```
K = ceil(N/4)
for (j = 0; j < N; j += K) {
    for (i = j; i < MIN(j + K, N); i++) {
        a[i] = b[i] + 3;
    }
}
```



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

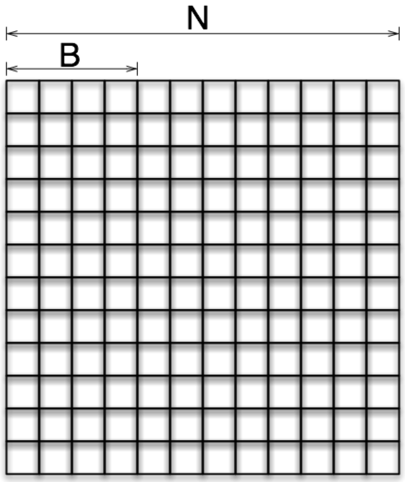
```
for (i = 0; i < N; i++)
    for (j = 0; j < N; j++)
        for (k = 0; k < N; k++)
            c[i][j] += a[i][k] * b[k][j];
```



Tiling for Matrix Multiply (cont'd)

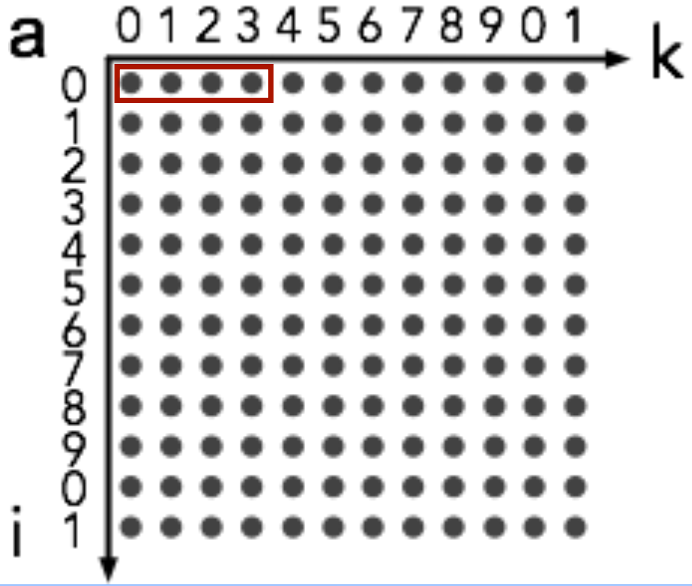
- All of a, b, or c cannot fit in the cache
  - Choose the block size B (the number of elements in a row of B) such that it is possible to fit one block from each of the matrices in the cache
- To improve spatial and temporal locality

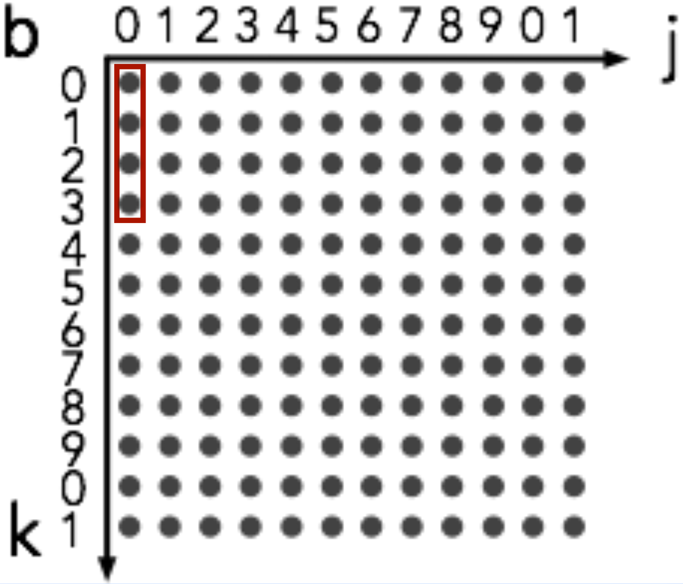
```
for (kk = 0; kk < N; kk += B)
    for (i = 0; i < N; i++)
        for (j = 0; j < N; j++)
            for (k = kk; k < MIN(kk+B, N); k++)
                c[i][j] += a[i][k] * b[k][j];
```



Tiling for Matrix Multiply (cont'd)

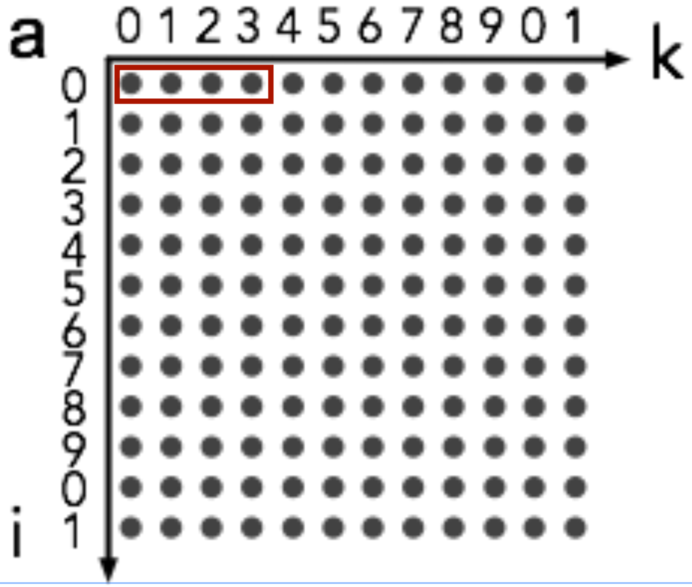
```
for (kk = 0; kk < N; kk += B)
    for (i = 0; i < N; i++)
        for (j = 0; j < N; j++)
            for (k = kk; k < MIN(kk+B, N); k++)
                c[i][j] += a[i][k] * b[k][j];
```

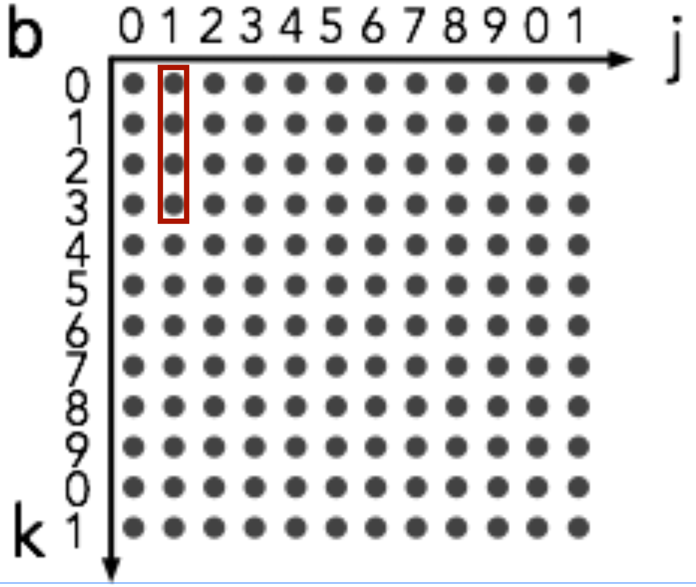




Tiling for Matrix Multiply (cont'd)

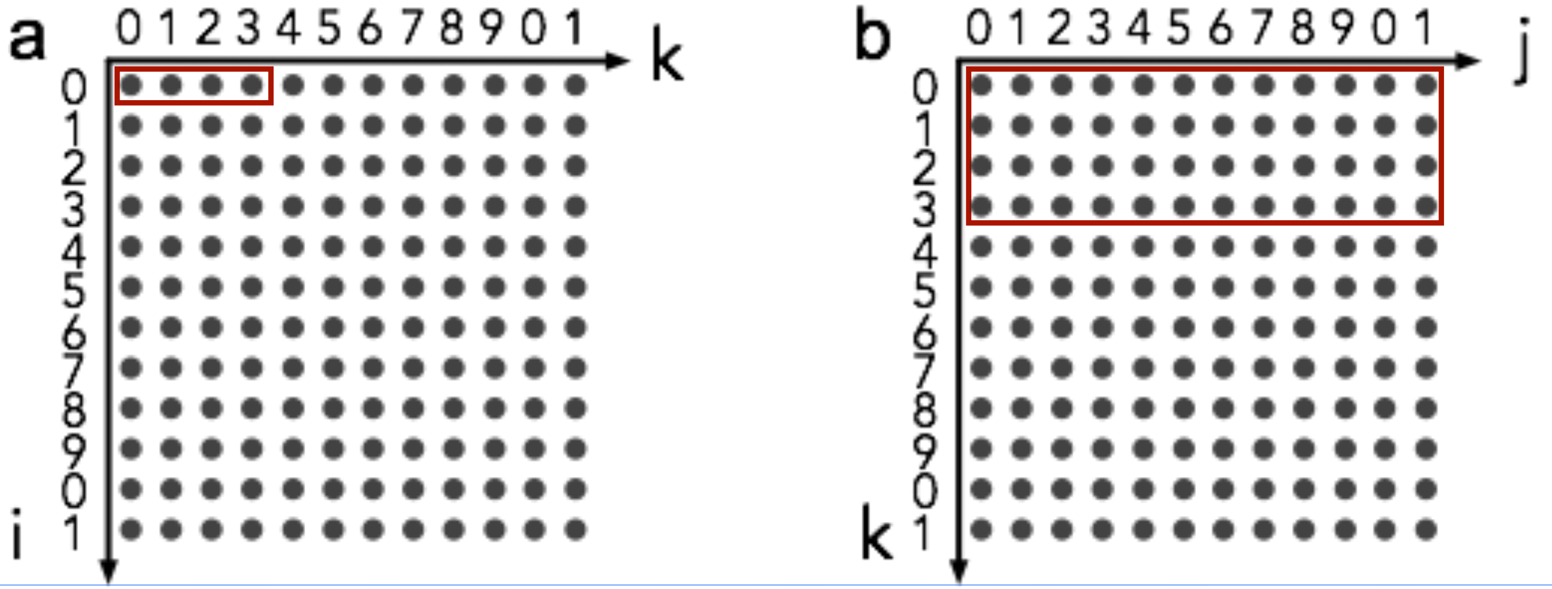
```
for (kk = 0; kk < N; kk += B)
    for (i = 0; i < N; i++)
        for (j = 0; j < N; j++)
            for (k = kk; k < MIN(kk+B, N); k++)
                c[i][j] += a[i][k] * b[k][j];
```





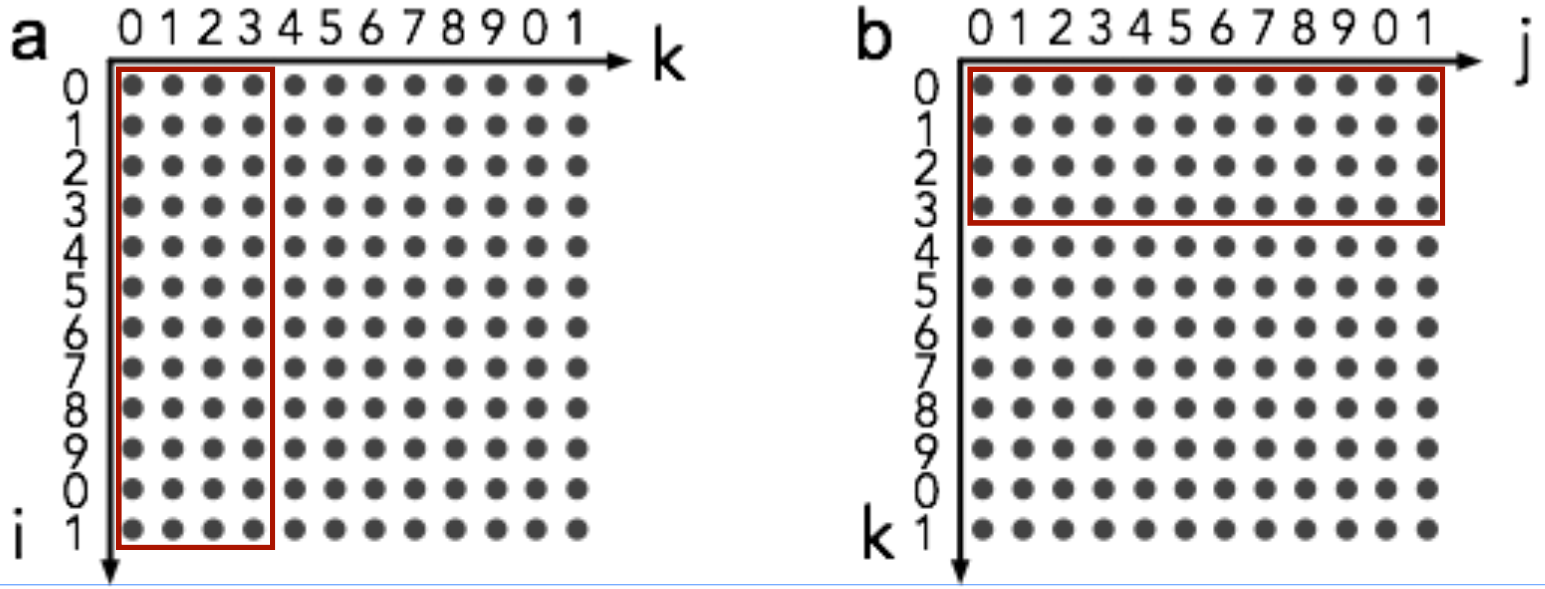
Tiling for Matrix Multiply (cont'd)

```
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];
```



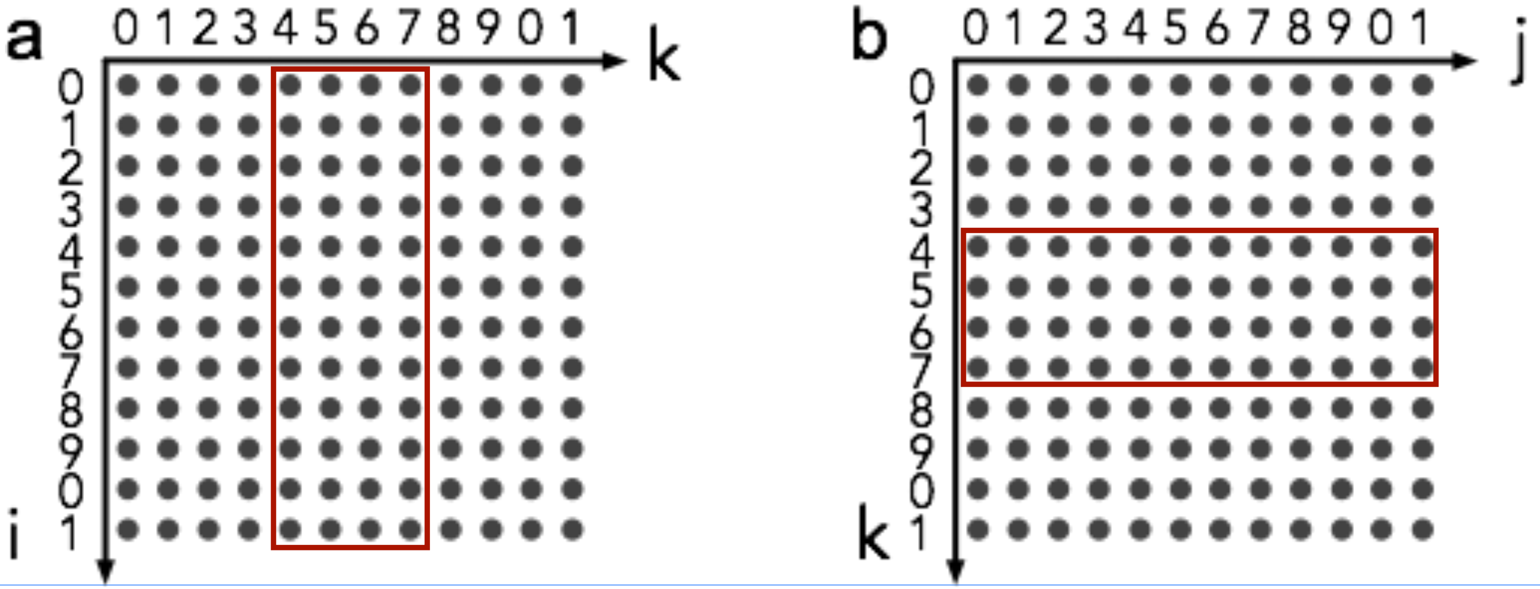
Tiling for Matrix Multiply (cont'd)

```
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];
```



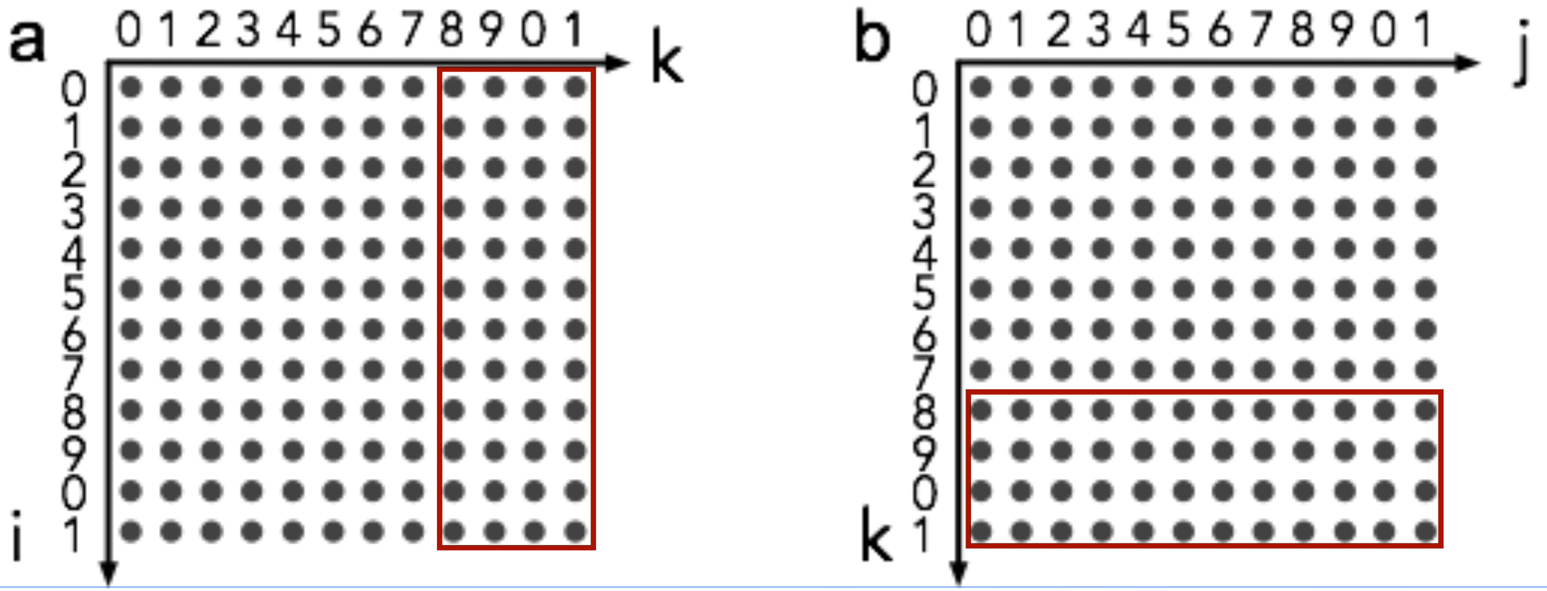
Tiling for Matrix Multiply (cont'd)

```
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];
```



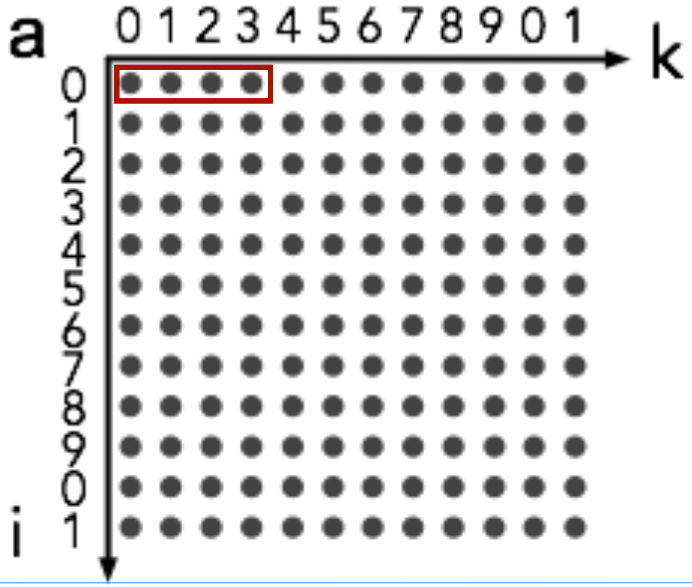
Tiling for Matrix Multiply (cont’d)

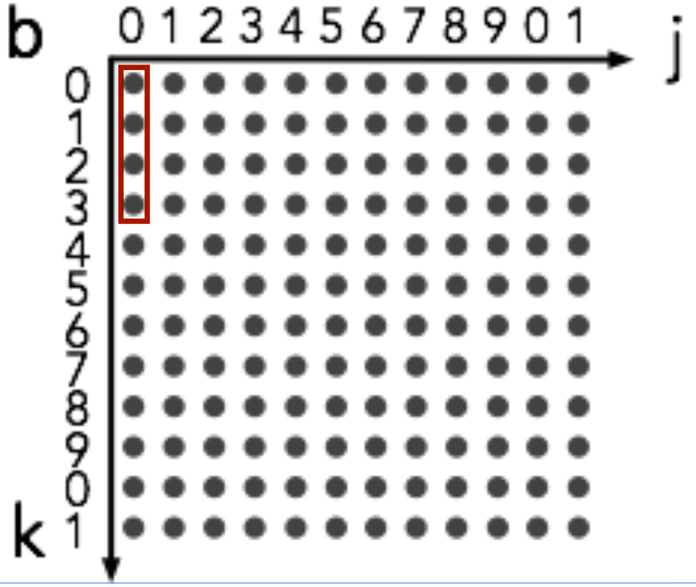
```
for (kk = 0; kk < N; kk += B)
    for (i = 0; i < N; i++)
        for (j = 0; j < N; j++)
            for (k = kk; k < MIN(kk+B, N); k++)
                c[i][j] += a[i][k] * b[k][j];
```



Tiling for Matrix Multiply (cont'd)

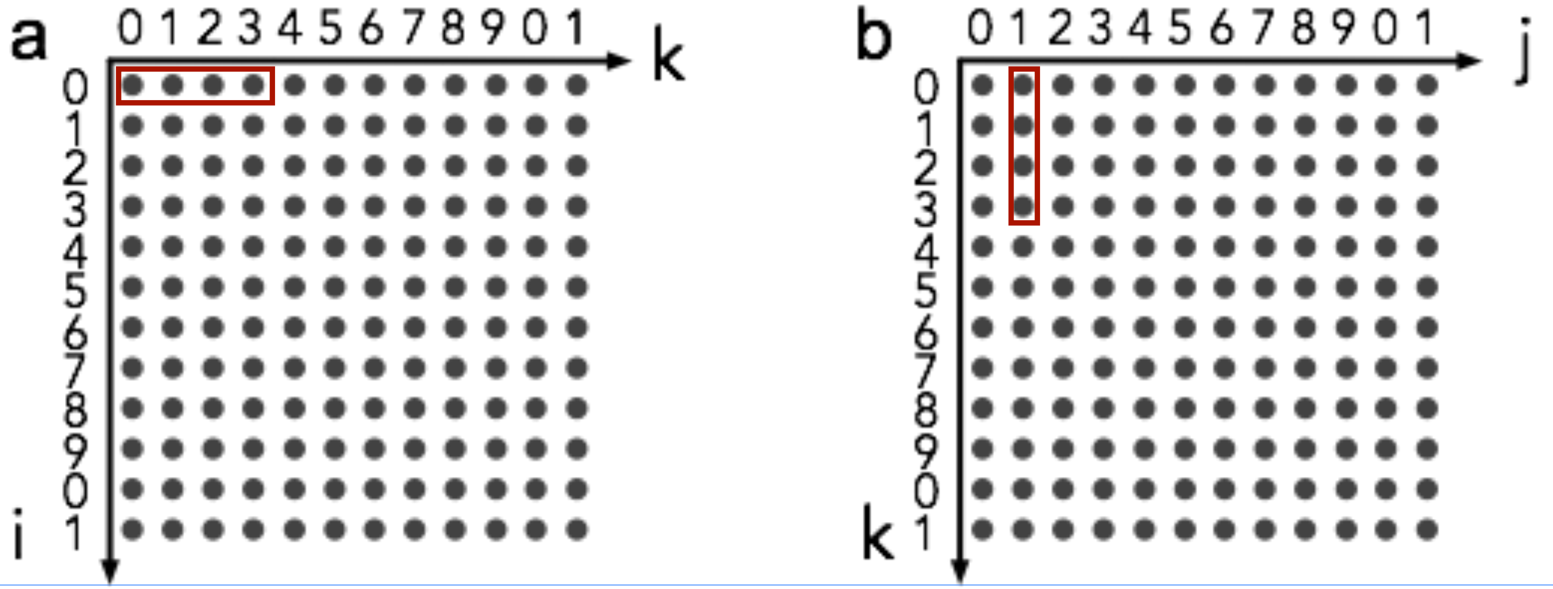
```
for (jj = 0; jj < N; jj += B)
    for (kk = 0; kk < N; kk += B)
        for (i = 0; i < N; i++)
            for (j = jj; j < MIN(jj+B, N); j++)
                for (k = kk; k < MIN(kk+B, N); k++)
                    c[i][j] += a[i][k] * b[k][j];
```





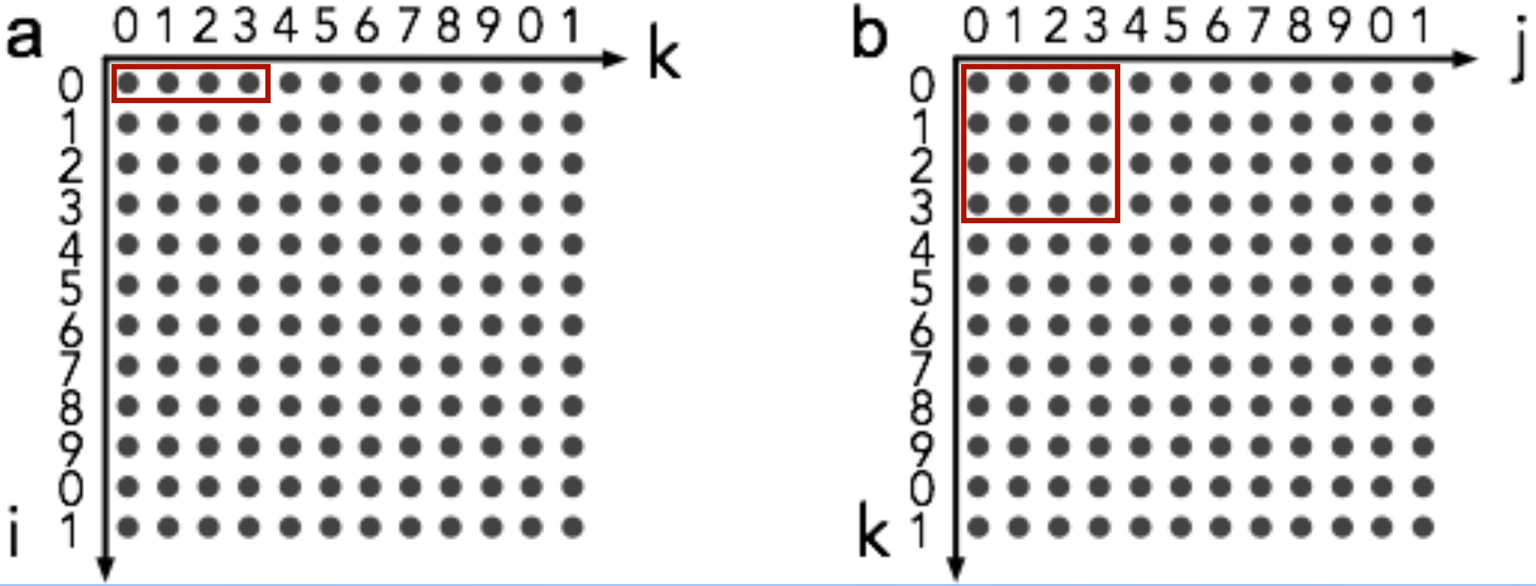
Tiling for Matrix Multiply (cont'd)

```
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];
```



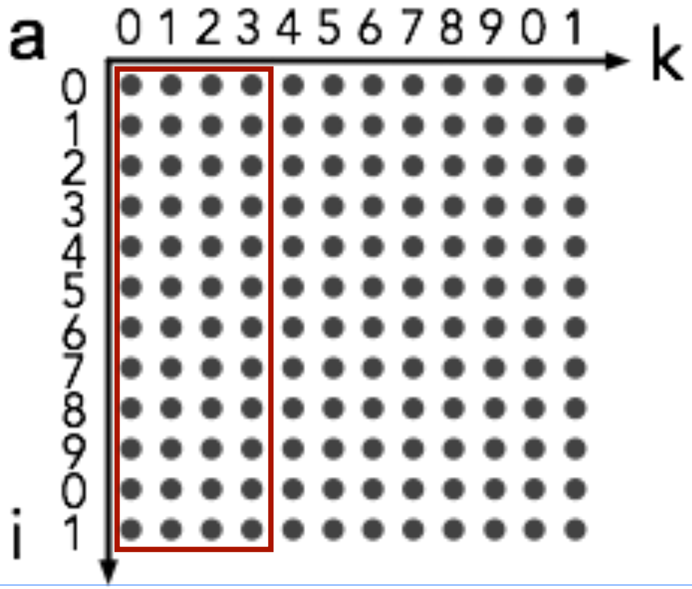
Tiling for Matrix Multiply (cont'd)

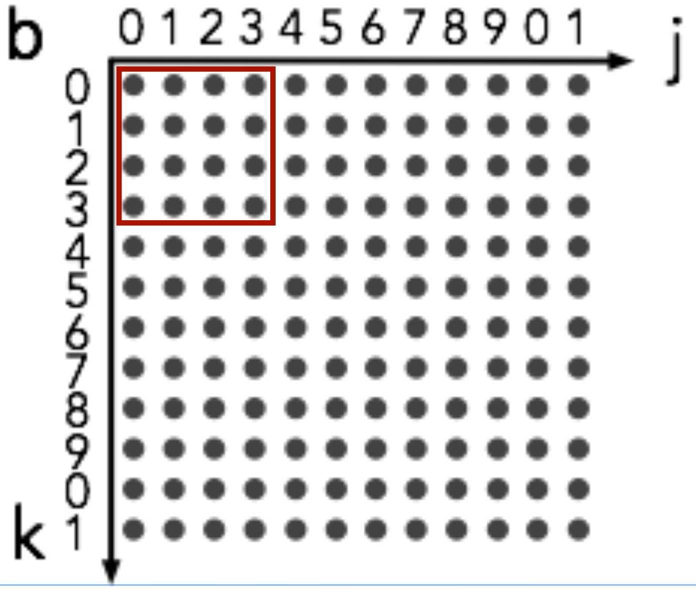
```
for (jj = 0; jj < N; jj += B)
    for (kk = 0; kk < N; kk += B)
        for (i = 0; i < N; i++)
            for (j = jj; j < MIN(jj+B, N); j++)
                for (k = kk; k < MIN(kk+B, N); k++)
                    c[i][j] += a[i][k] * b[k][j];
```



Tiling for Matrix Multiply (cont'd)

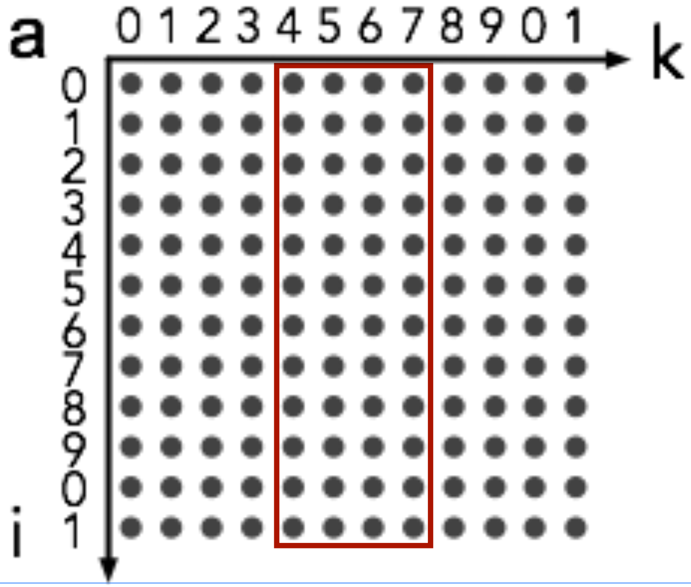
```
for (jj = 0; jj < N; jj += B)
    for (kk = 0; kk < N; kk += B)
        for (i = 0; i < N; i++)
            for (j = jj; j < MIN(jj+B, N); j++)
                for (k = kk; k < MIN(kk+B, N); k++)
                    c[i][j] += a[i][k] * b[k][j];
```

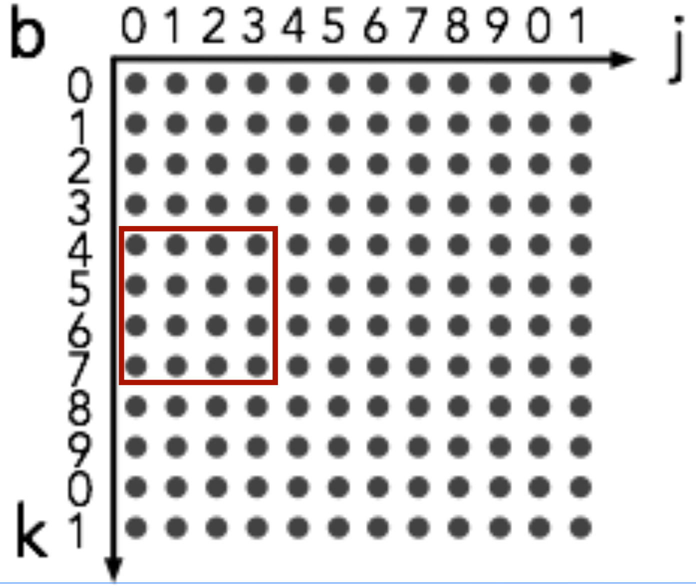




Tiling for Matrix Multiply (cont’d)

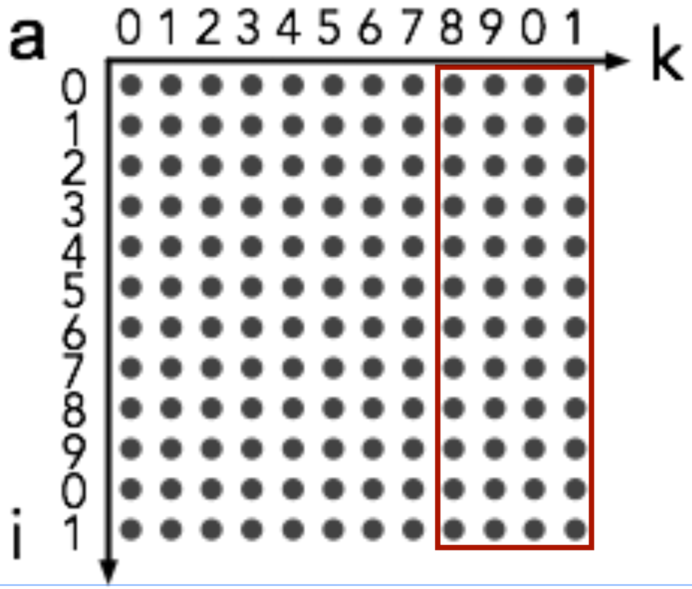
```
for (jj = 0; jj < N; jj += B)
    for (kk = 0; kk < N; kk += B)
        for (i = 0; i < N; i++)
            for (j = jj; j < MIN(jj+B, N); j++)
                for (k = kk; k < MIN(kk+B, N); k++)
                    c[i][j] += a[i][k] * b[k][j];
```

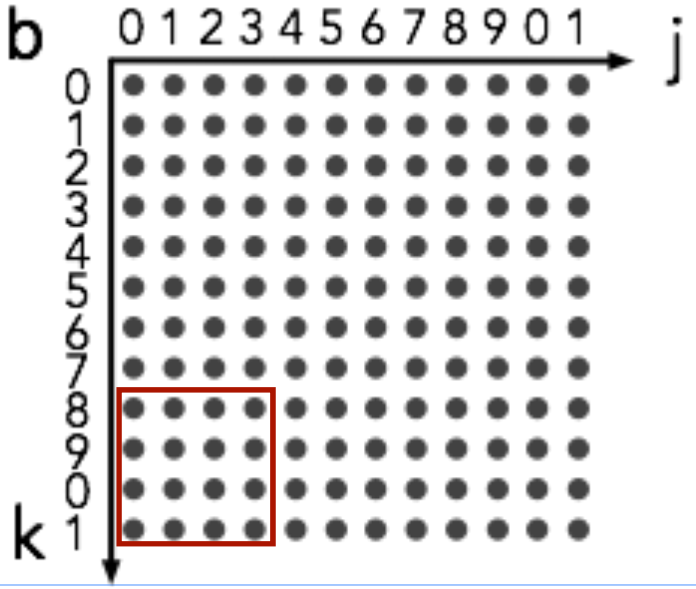




Tiling for Matrix Multiply (cont'd)

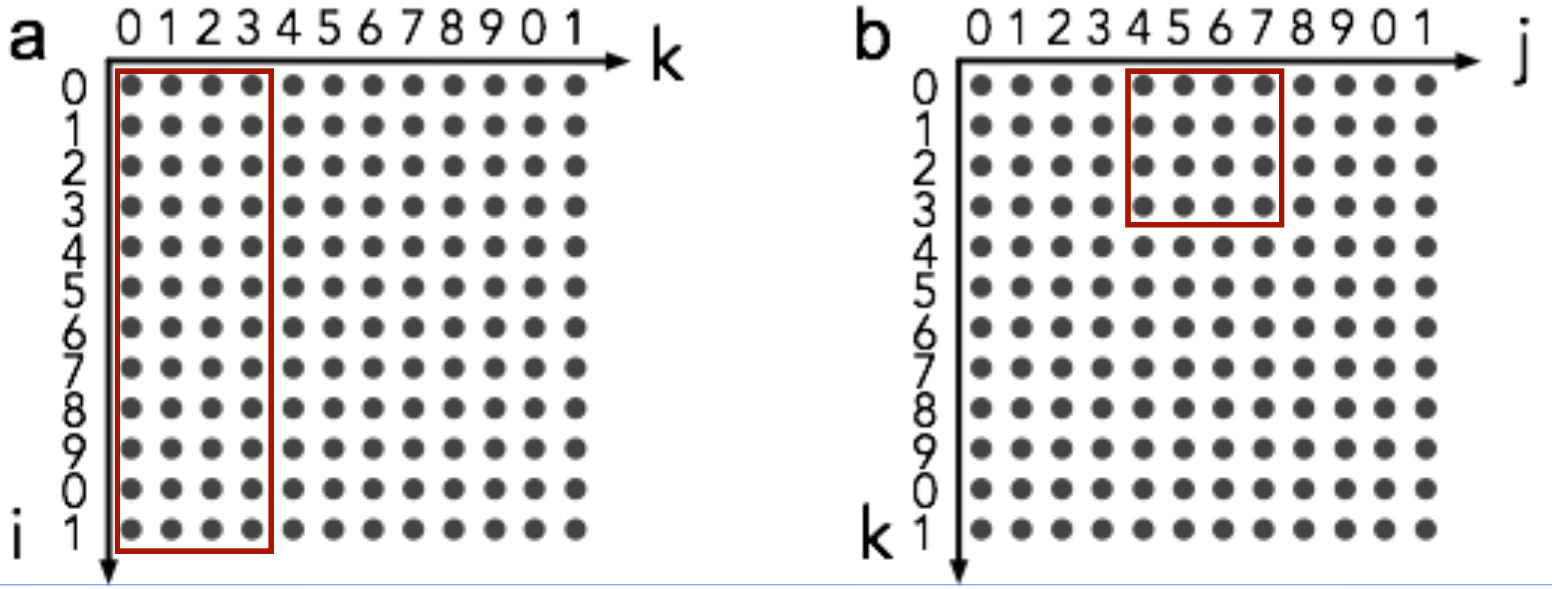
```
for (jj = 0; jj < N; jj += B)
    for (kk = 0; kk < N; kk += B)
        for (i = 0; i < N; i++)
            for (j = jj; j < MIN(jj+B, N); j++)
                for (k = kk; k < MIN(kk+B, N); k++)
                    c[i][j] += a[i][k] * b[k][j];
```





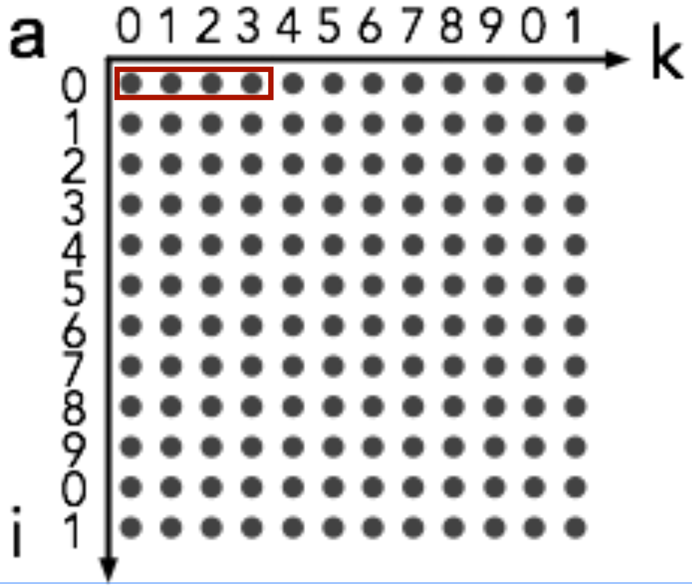
Tiling for Matrix Multiply (cont'd)

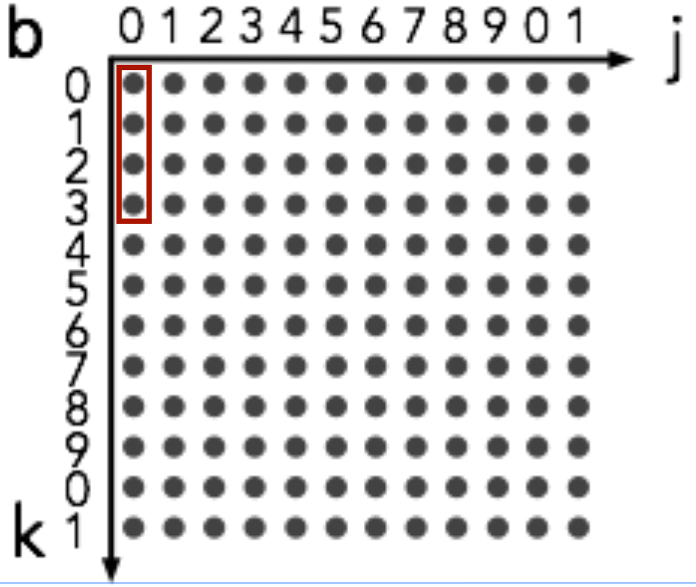
```
for (jj = 0; jj < N; jj += B)
    for (kk = 0; kk < N; kk += B)
        for (i = 0; i < N; i++)
            for (j = jj; j < MIN(jj+B, N); j++)
                for (k = kk; k < MIN(kk+B, N); k++)
                    c[i][j] += a[i][k] * b[k][j];
```



Tiling for Matrix Multiply (cont'd)

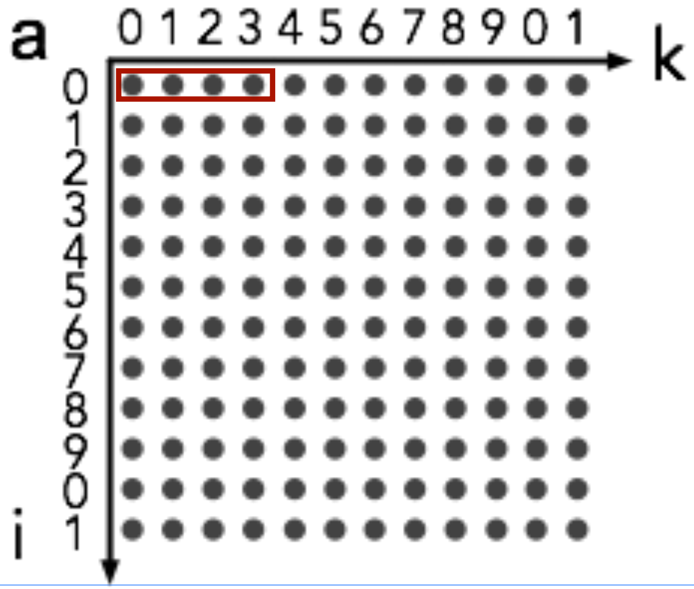
```
for (ii = 0; ii < N; ii += B)
    for (jj = 0; jj < N; jj += B)
        for (kk = 0; kk < N; kk += B)
            for (i = ii; i < MIN(ii+B, N); i++)
                for (j = jj; j < MIN(jj+B, N); j++)
                    for (k = kk; k < MIN(kk+B, N); k++)
                        c[i][j] += a[i][k]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

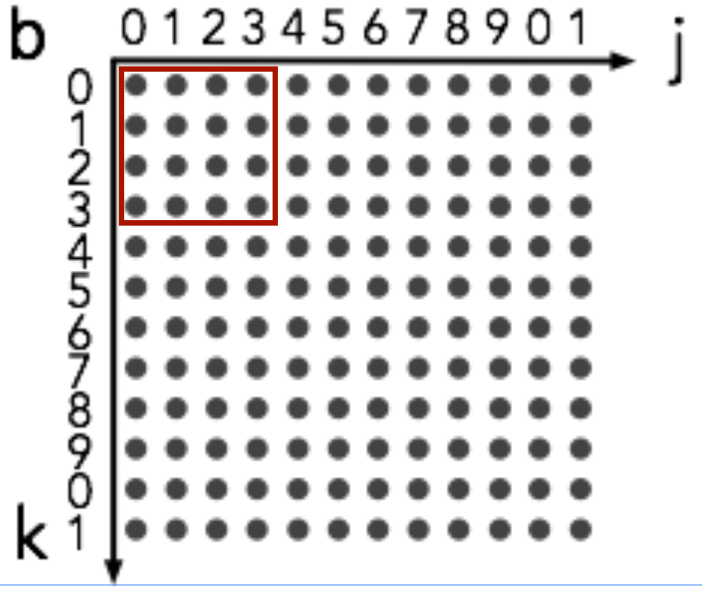




Tiling for Matrix Multiply (cont'd)

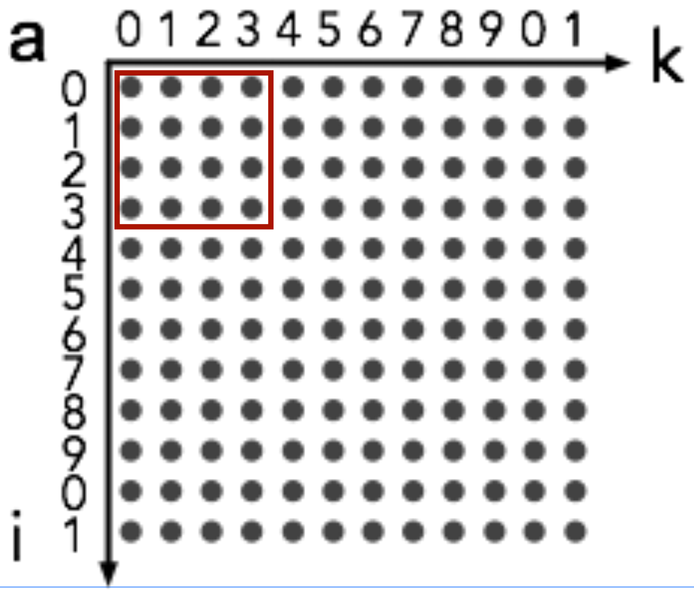
```
for (ii = 0; ii < N; ii += B)
    for (jj = 0; jj < N; jj += B)
        for (kk = 0; kk < N; kk += B)
            for (i = ii; i < MIN(ii+B, N); i++)
                for (j = jj; j < MIN(jj+B, N); j++)
                    for (k = kk; k < MIN(kk+B, N); k++)
                        c[i][j] += a[i][k]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

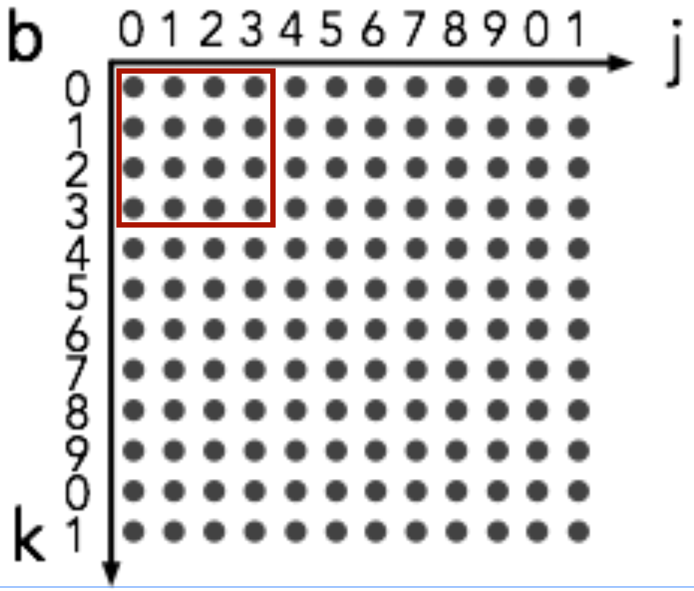




Tiling for Matrix Multiply (cont'd)

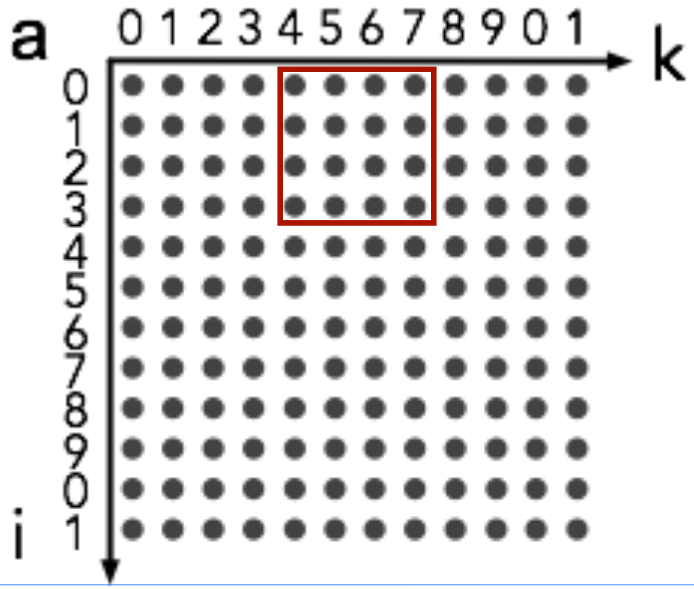
```
for (ii = 0; ii < N; ii += B)
    for (jj = 0; jj < N; jj += B)
        for (kk = 0; kk < N; kk += B)
            for (i = ii; i < MIN(ii+B, N); i++)
                for (j = jj; j < MIN(jj+B, N); j++)
                    for (k = kk; k < MIN(kk+B, N); k++)
                        c[i][j] += a[i][k]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

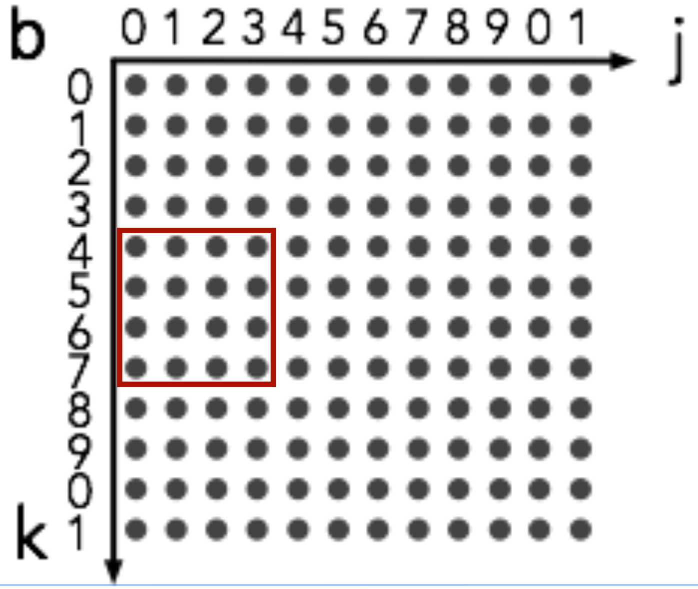




Tiling for Matrix Multiply (cont'd)

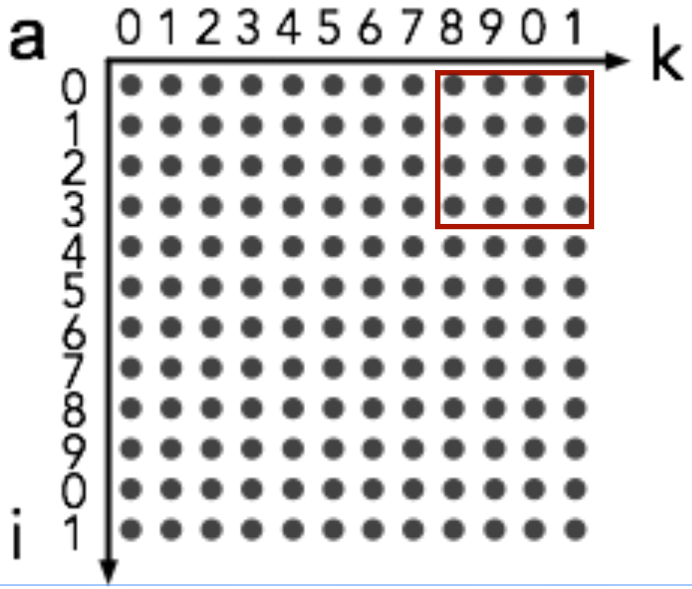
```
for (ii = 0; ii < N; ii += B)
    for (jj = 0; jj < N; jj += B)
        for (kk = 0; kk < N; kk += B)
            for (i = ii; i < MIN(ii+B, N); i++)
                for (j = jj; j < MIN(jj+B, N); j++)
                    for (k = kk; k < MIN(kk+B, N); k++)
                        c[i][j] += a[i][k]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

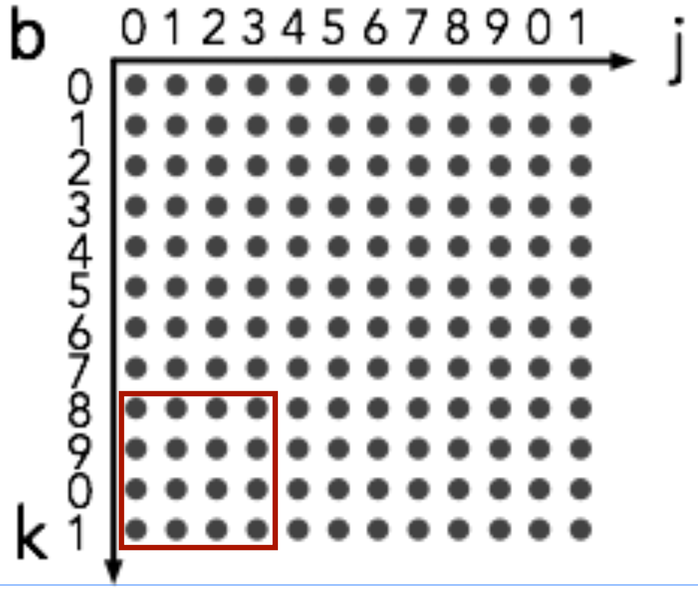




Tiling for Matrix Multiply (cont'd)

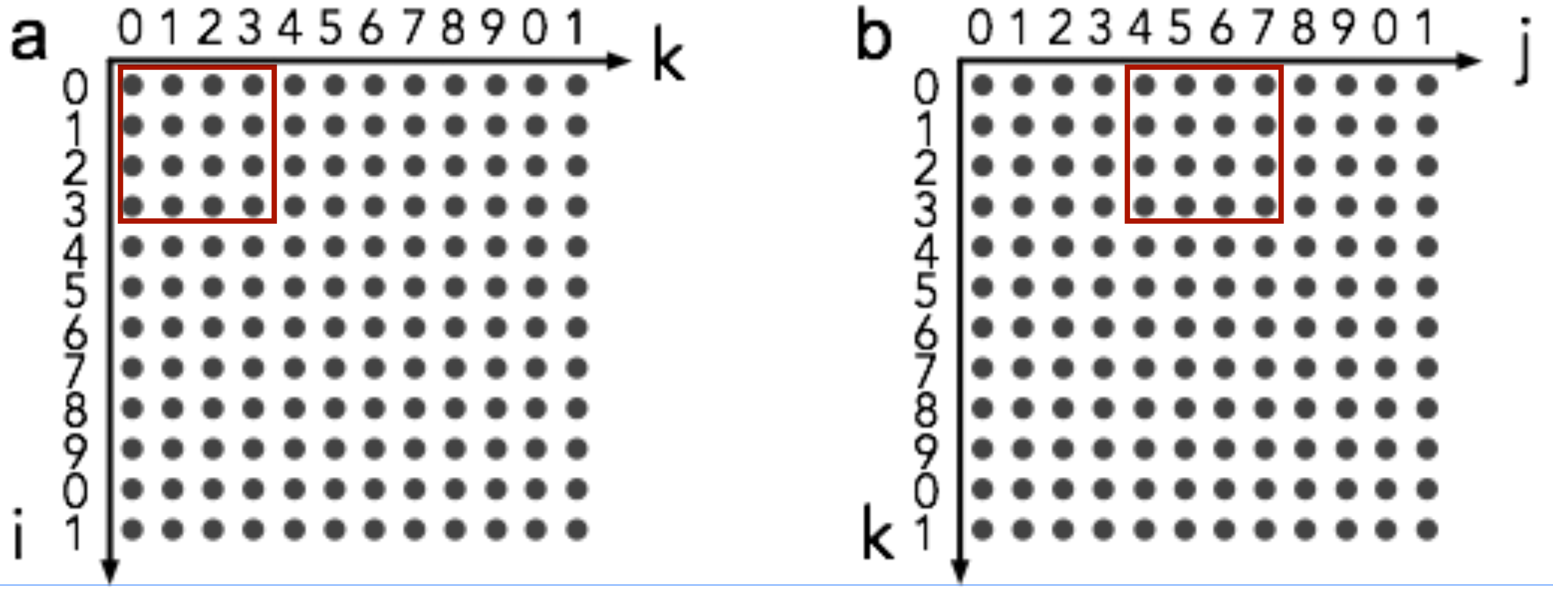
```
for (ii = 0; ii < N; ii += B)
    for (jj = 0; jj < N; jj += B)
        for (kk = 0; kk < N; kk += B)
            for (i = ii; i < MIN(ii+B, N); i++)
                for (j = jj; j < MIN(jj+B, N); j++)
                    for (k = kk; k < MIN(kk+B, N); k++)
                        c[i][j] += a[i][k]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





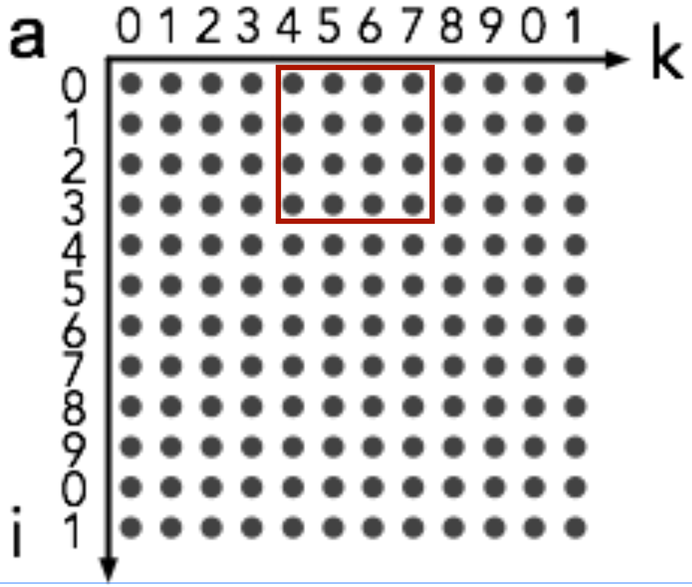
Tiling for Matrix Multiply (cont’d)

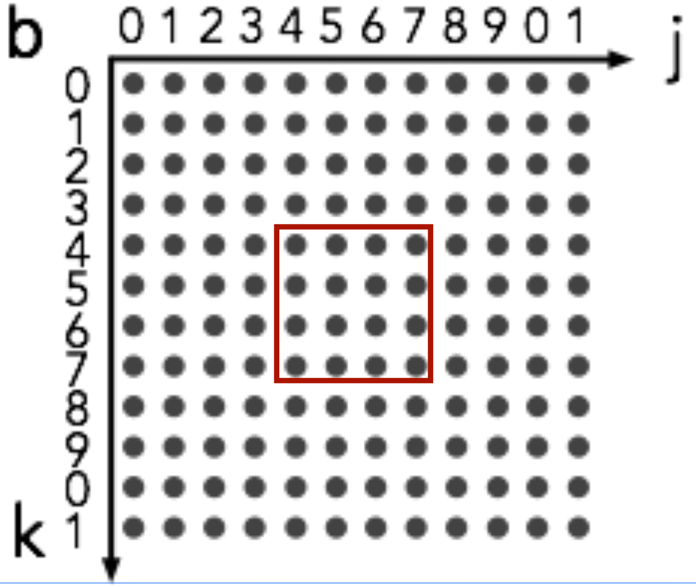
```
for (ii = 0; ii < N; ii += B)
    for (jj = 0; jj < N; jj += B)
        for (kk = 0; kk < N; kk += B)
            for (i = ii; i < MIN(ii+B, N); i++)
                for (j = jj; j < MIN(jj+B, N); j++)
                    for (k = kk; k < MIN(kk+B, N); k++)
                        c[i][j] += a[i][k]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



Tiling for Matrix Multiply (cont'd)

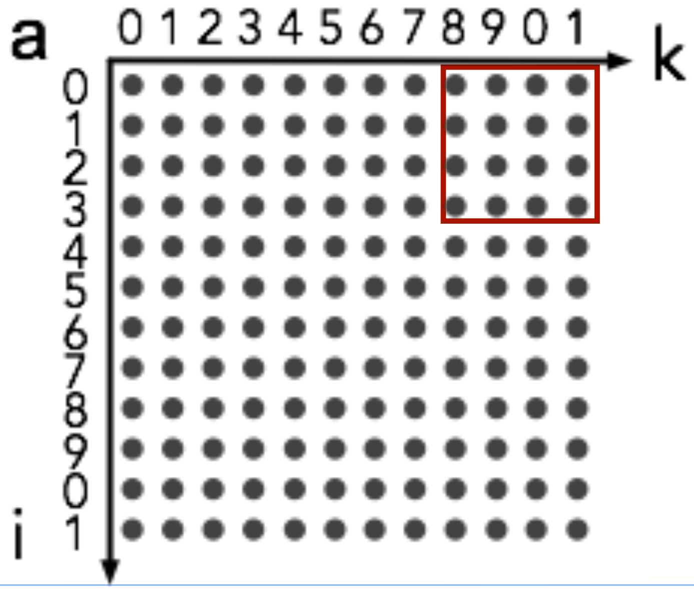
```
for (ii = 0; ii < N; ii += B)
    for (jj = 0; jj < N; jj += B)
        for (kk = 0; kk < N; kk += B)
            for (i = ii; i < MIN(ii+B, N); i++)
                for (j = jj; j < MIN(jj+B, N); j++)
                    for (k = kk; k < MIN(kk+B, N); k++)
                        c[i][j] += a[i][k]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

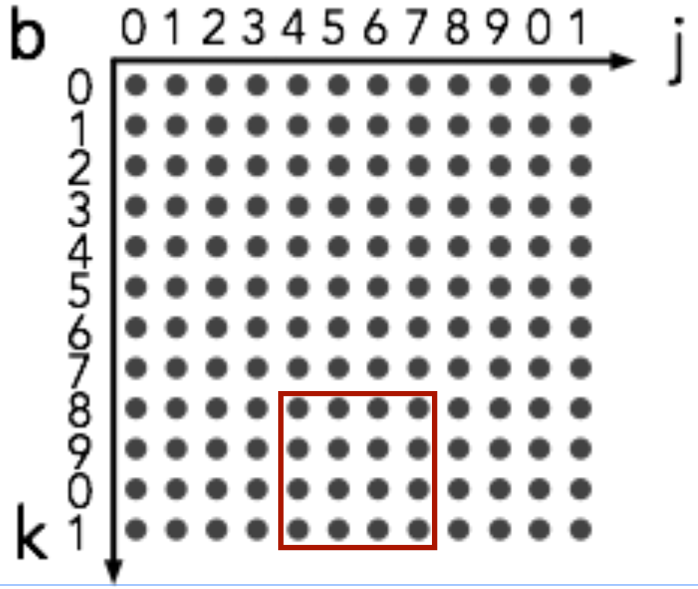




Tiling for Matrix Multiply (cont'd)

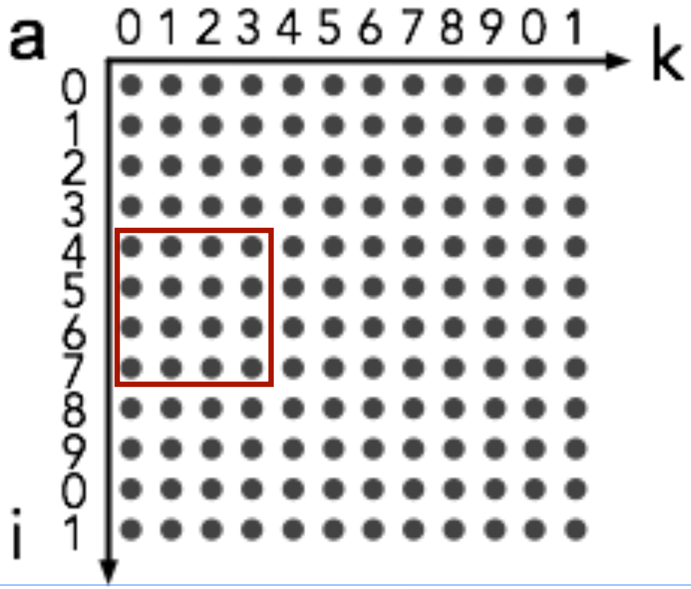
```
for (ii = 0; ii < N; ii += B)
    for (jj = 0; jj < N; jj += B)
        for (kk = 0; kk < N; kk += B)
            for (i = ii; i < MIN(ii+B, N); i++)
                for (j = jj; j < MIN(jj+B, N); j++)
                    for (k = kk; k < MIN(kk+B, N); k++)
                        c[i][j] += a[i][k]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

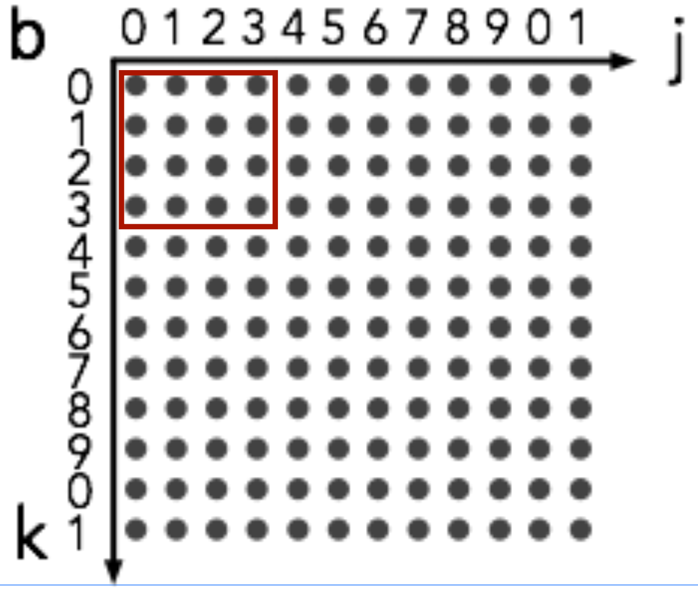




Tiling for Matrix Multiply (cont'd)

```
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

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

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

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

```
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];
```