CACHE BLOCKING

Slides based on: Computer Architecture Book (6th Edition, page 107) by Hennessey Patterson

Slides adapted by: Dr Sparsh Mittal

Matrix-multiplication before blocking

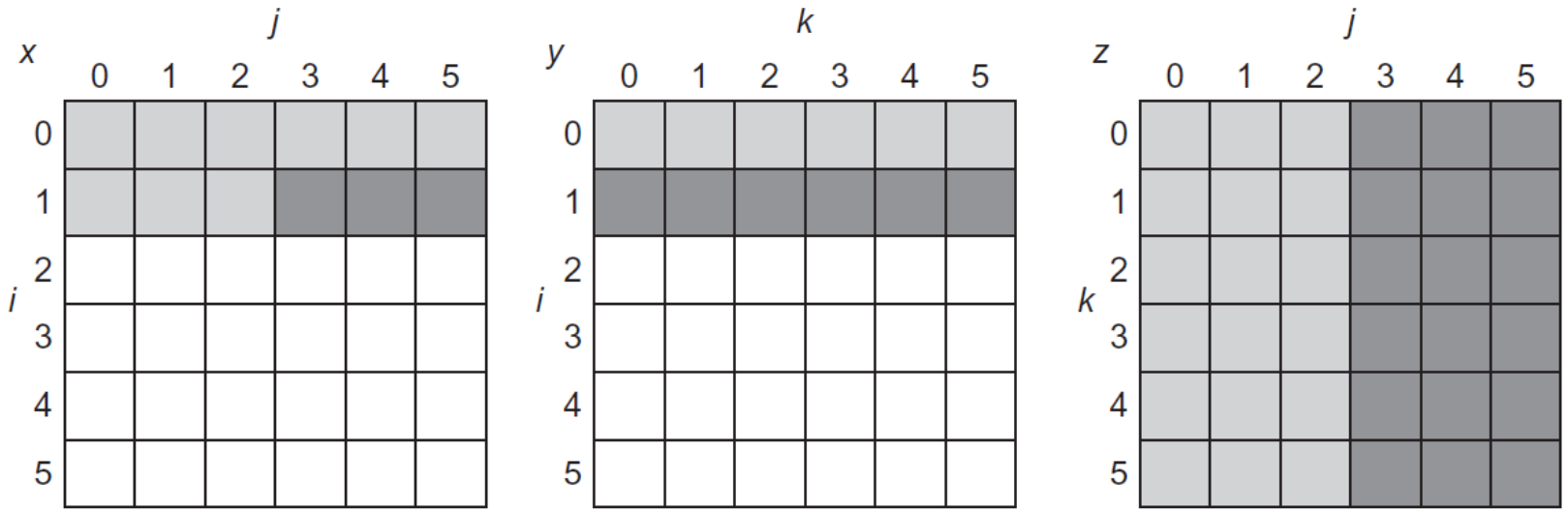
```
for (i = 0; i < N; i = i + 1)
        for (j = 0; j < N; j = j + 1)
                {r = 0;
                 for (k = 0; k < N; k = k + 1)
                         r = r + y[i][k]*z[k][j];
                 x[i][j] = r;
                };
```

(Assume x is initialized to zero.)

Access pattern: accessing multiple arrays such that some arrays are accessed by rows and some by columns.

Assume: cache is fully-associative => no conflict miss

Snapshot of arrays when N=6 and i=1



The age of accesses to the array elements is indicated by shade:
white means not yet touched,
light means older accesses, and
dark means newer accesses.

The elements of y and z are read repeatedly to calculate new elements of x.

The variables i, j, and k are shown along the rows or columns used to access the arrays.

Limitation of approaches discussed earlier

- Storing the arrays row by row (row major order) or column by column (column major order) does not solve the problem because both rows and columns are used in every loop iteration.
- Loop interchange still leaves room fo improvement.

Matrix-multiplication before blocking

```
for (i = 0; i < N; i = i + 1)
        for (j = 0; j < N; j = j + 1)
                {r = 0;
                 for (k = 0; k < N; k = k + 1)
                         r = r + y[i][k]*z[k][j];
                 x[i][j] = r;
                };
```

The two inner loops:

- read all N-by-N elements of z
- read the same N elements in a row of y repeatedly, and
- write one row of N elements of x

Cache miss analysis (before blocking)

- The number of capacity misses depends on N and the size of the cache.
- If it can hold all three N-by-N matrices, then everything is OK.
- If the cache can hold one N-by-N matrix and one row of N, then at least the i^{th} row of y and the array z may stay in the cache.
- If cache has less size than that, then misses may occur for both x and z.

Cache blocking: key idea

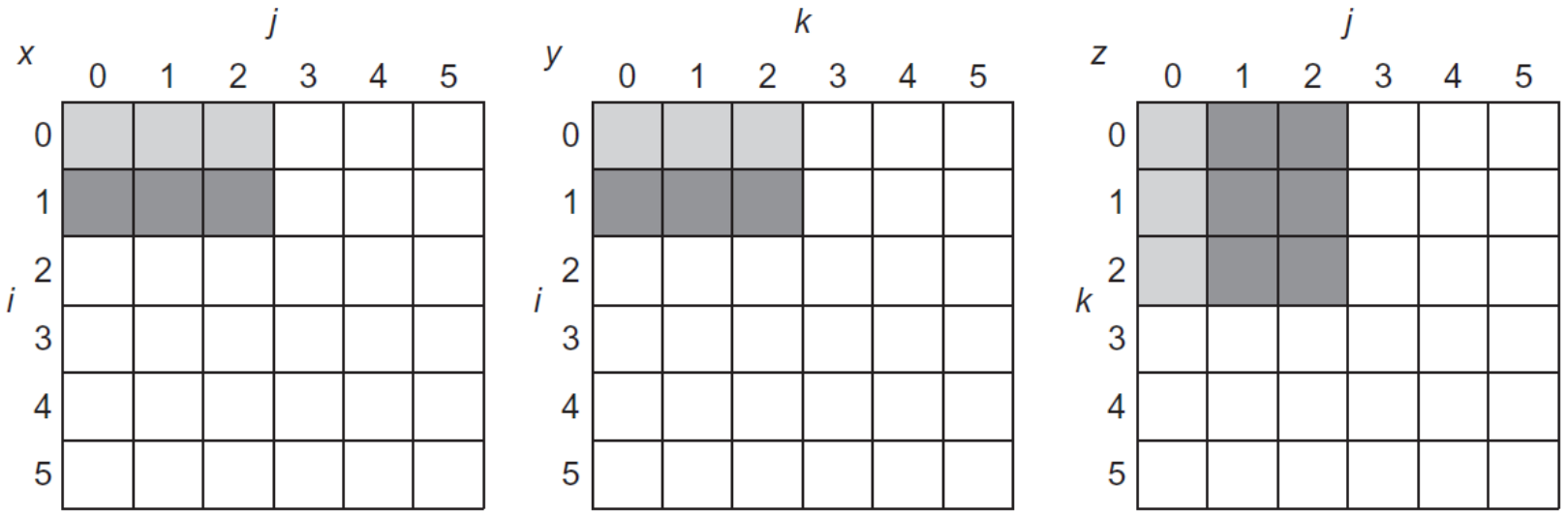
- Instead of operating on entire rows or columns of an array, blocked algorithms operate on submatrices or blocks.
- The goal is to maximize accesses to the data loaded into the cache before the data are replaced.
- Divide loops operating on arrays into computation chunks so that each chunk can hold its data in the cache
- Avoids cache conflicts between different chunks of computation
- Essentially: Divide the working set so that each piece fits in the cache

Matrix-multiplication after blocking

```
for (jj = 0; jj < N; jj = jj + B)
for (kk = 0; kk < N; kk = kk + B)
for (i = 0; i < N; i = i + 1)
        for (j = jj; j < min(jj + B,N); j = j + 1)
            {r = 0;
             for (k = kk; k < min(kk + B,N); k = k + 1)
                 r = r + y[i][k]*z[k][j];
             x[i][j] = x[i][j] + r;
            };
```

- We change code to compute on a submatrix of size B by B
- Two inner loops now compute in steps of size B rather than the full length of x and z. B is called the blocking factor.

Snapshot of arrays when N=6 and i=1



The age of accesses to the arrays x, y, and z when B=3.

Compared to original code, less elements are accessed. Reuse distance of elements is reduced, so cache misses are reduced.

Blocking: summary

- Blocking exploits a combination of spatial and temporal locality:
  - y benefits from spatial locality
  - z benefits from temporal locality.

- Cache blocking is absolutely necessary to get good performance from cache-based processors running applications using matrices as the primary data structure