High Performance Computing
Lecture 31

Matthew Jacob

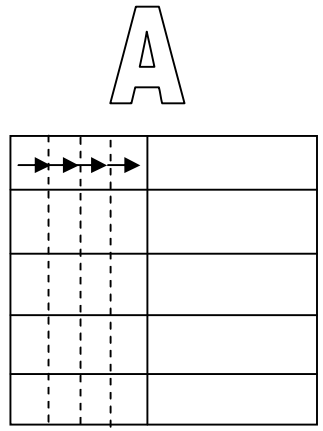
Indian Institute of Science

Example 4 with Loop Interchange

```
double A[1024][1024], B[1024][1024];
for (j=0; j<1024; j++)
for (i=0; i<1024; i++)
    B[i][j] = A[i][j] + B[i][j];
```

Example 4 with Loop Interchange.

```
double A[1024][1024], B[1024][1024];
for (i=0; i<1024; i++)
for (j=0; j<1024; j++)
    B[i][j] = A[i][j] + B[i][j];
```

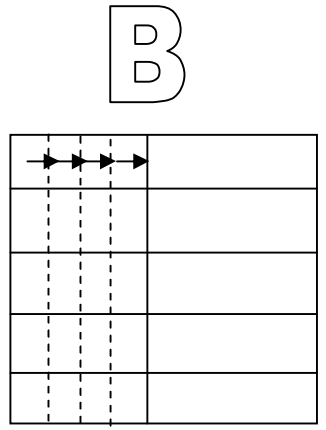


- Reference Sequence:

  load A[0,0] load B[0,0] store B[0,0]

  load A[0,1] load B[0,1] store B[0,1] **...**

- Hit ratio: 83.3%

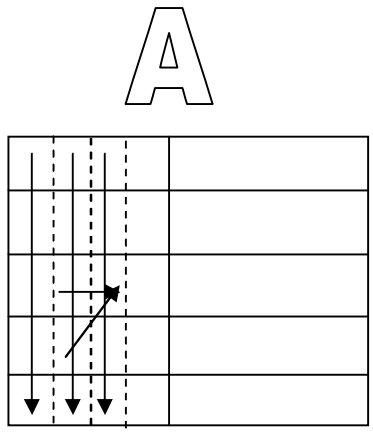


Is Loop Interchange Always Safe?

~~for (i=2047; i>1; i--)~~
~~for (i=1; i<2048; i++)~~

for (j=1; j<2048; j++)

A[i][j] = A[i+1][j-1] + A[i][j-1];

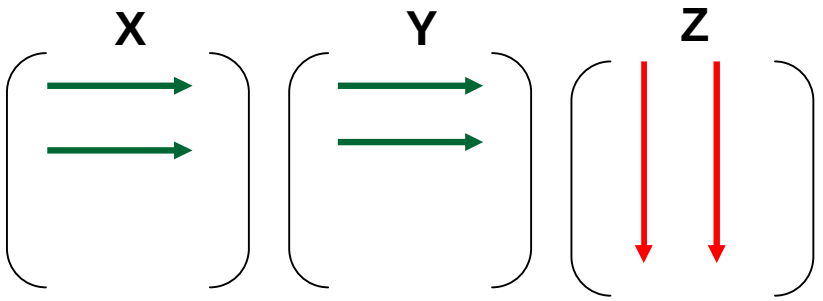


A[1,1] = A[2,0]+A[1,0]
A[2,1] = A[3,0]+A[2,0]
…
A[1,2] = A[2,1]+A[1,1]

A[1,1] = A[2,0]+A[1,0]
A[1,2] = A[2,1]+A[1,1]
…
A[2,1] = A[3,0]+A[2,0]

Example 5: Matrix Multiplication

```
double X[N][N], Y[N][N], Z[N][N];
for (i=0; i<N; i++)
  for (j=0; j<N; j++)
    for (k=0; k<N; k++)
      X[i][j] += Y[i][k] * Z[k][j];
                          / Dot product inner loop
```



Reference Sequence:

Y[0,0], Z[0,0], Y[0,1], Z[1,0], Y[0,2], Z[2,0] … X[0,0],
Y[0,0], Z[0,1], Y[0,1], Z[1,1], Y[0,2], Z[2,1] … X[0,1],
…
Y[1,0], Z[0,0], Y[1,1], Z[1,0], Y[1,2], Z[2,0] … X[1,0],

With Loop Interchanging

- Can interchange the 3 loops in any way
- Example: Interchange i and k loops

```
double X[N][N], Y[N][N], Z[N][N];
for (k=0; k<N; k++)
    for (j=0; j<N; j++)
        for (i=0; i<N; i++)
            X[i][j] += Y[i][k] * Z[k][j];
```

- For inner loop: Z[k][j] can be loaded into register once for each (k,j), reducing the number of memory references

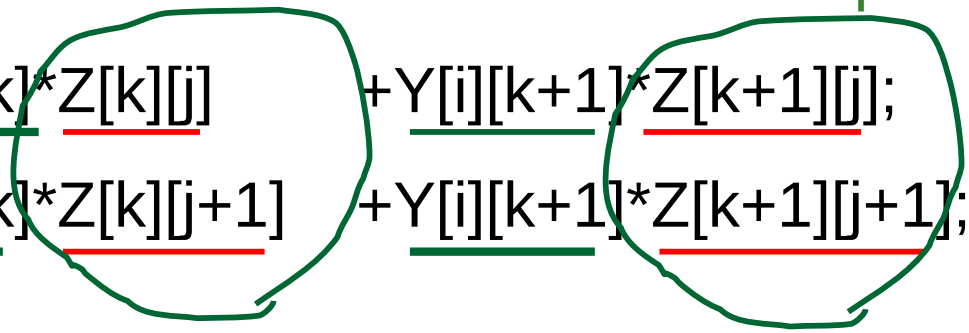
Let's try some Loop Unrolling Instead

```
double X[N][N], Y[N][N], Z[N][N];
for (i=0; i<N; i++)
    for (j=0; j<N; j++)
      for (k=0; k<N; k+=2)
          X[i][j] += Y[i][k]*Z[k][j] + Y[i][k+1]*Z[k+1][j];
```

Unroll k loop

Let's try some Loop Unrolling Instead

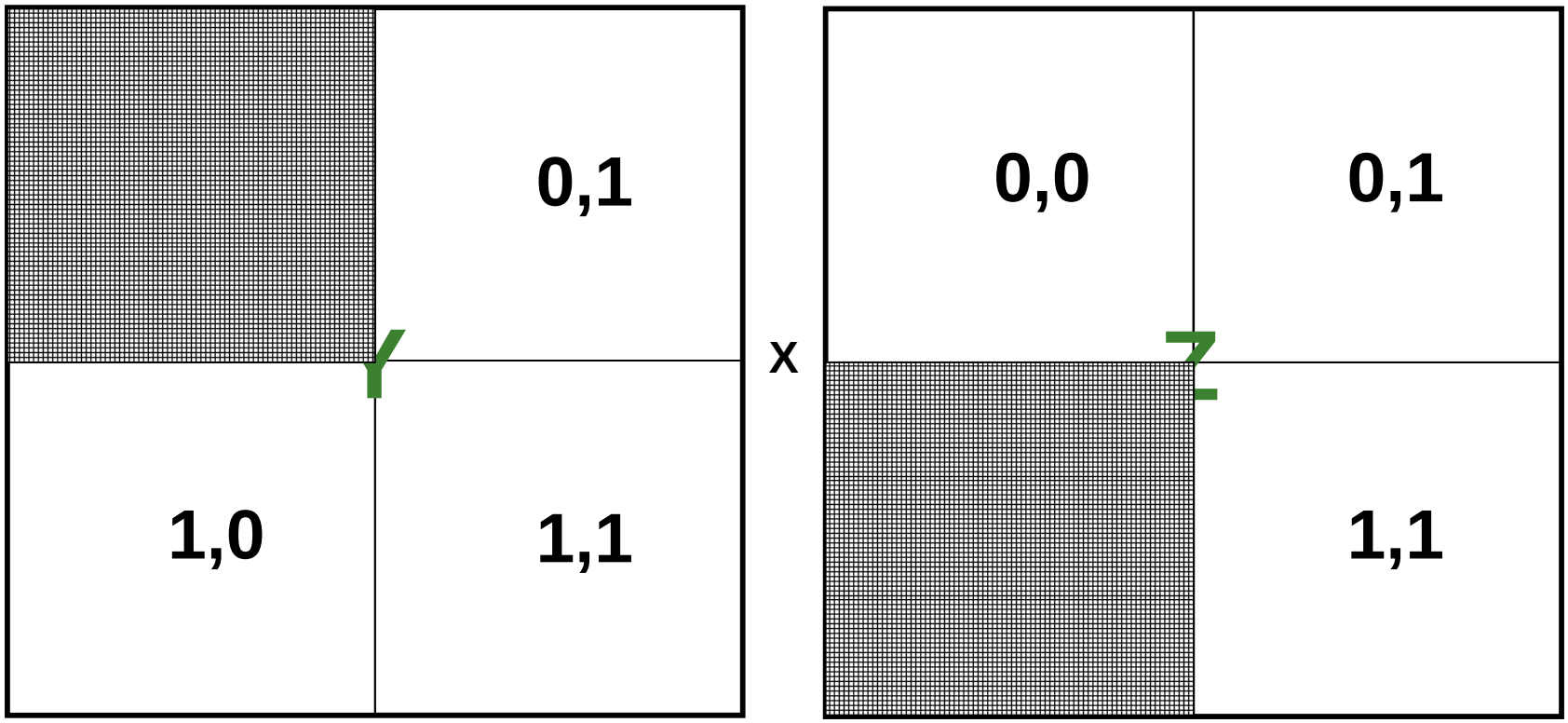
```
double X[N][N], Y[N][N], Z[N][N];
for (i=0; i<N; i++)
    for (j=0; j<N; j+=2)                                    Unroll j loop
      for (k=0; k<N; k+=2){                                 Unroll k loop
        X[i][j]     += Y[i][k]*Z[k][j]     +Y[i][k+1]*Z[k+1][j];
        X[i][[j+1]  += Y[i][k]*Z[k][j+1]   +Y[i][k+1]*Z[k+1][j+1];
      }
```



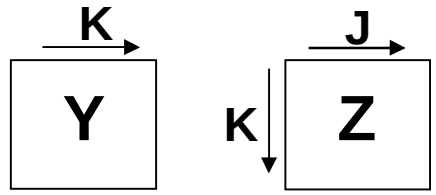
Blocking or Tiling

Blocking/Tiling

Idea: Since problem is with accesses to array Z, make full use of elements of Z when they are brought into the cache



Blocked Matrix Multiplication



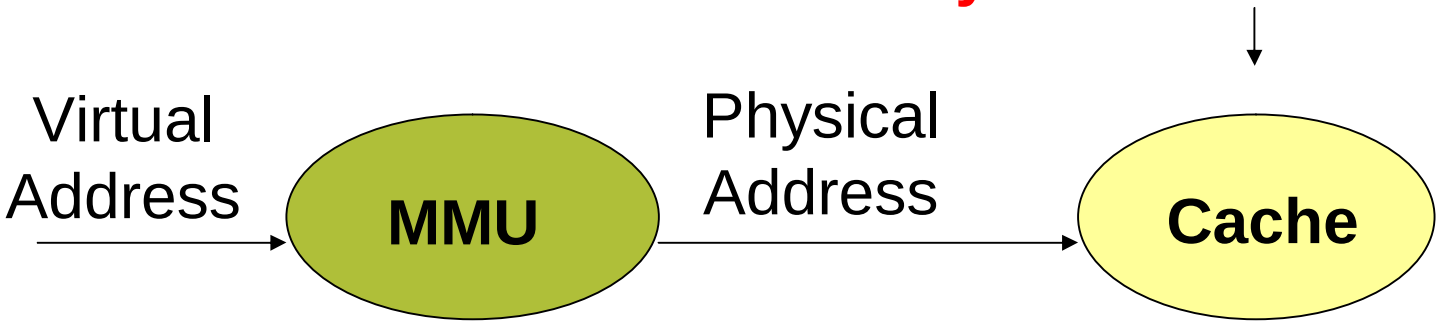

```
for (J=0; J<N; J+=B)
for (K=0; K<N; K+=B)
for (i=0; i<N; i++)
for (j=J; j<min(J+B,N); j++){
   for (k=K, r=0; k<min(K+B,N); k++)
      r += Y[i][k] * Z[k][j];
    X[i][j] += r;
}
```

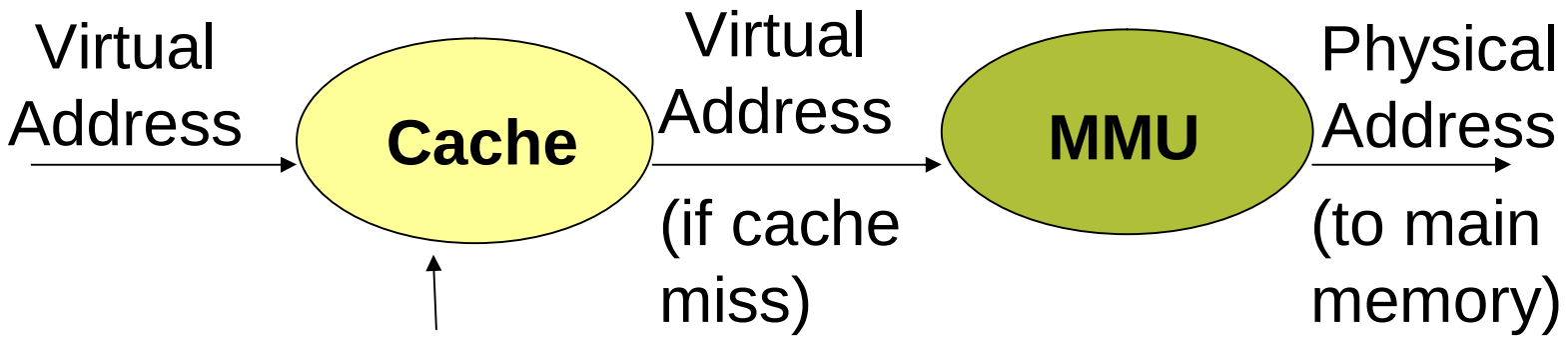
Reality Check

- Question 1: Are real caches built to work on virtual addresses or physical addresses?
- Question 2: Do modern processors use pipelining of the kind that we studied?

Q1: Caches and Address Translation

“Physical Addressed Cache”





“Virtual Addressed Cache”

Which is less preferable?

- Physical addressed cache
  - Hit time higher (cache access after translation)
- Virtual addressed cache
  - Data/instruction of different processes with same virtual address in cache at the same time …
    - Flush cache on context switch, or
    - Include Process id as part of each cache directory entry
  - Synonyms
    - Virtual addresses that translate to same physical address
    - More than one copy of a block in cache …