Matrix Multiplication

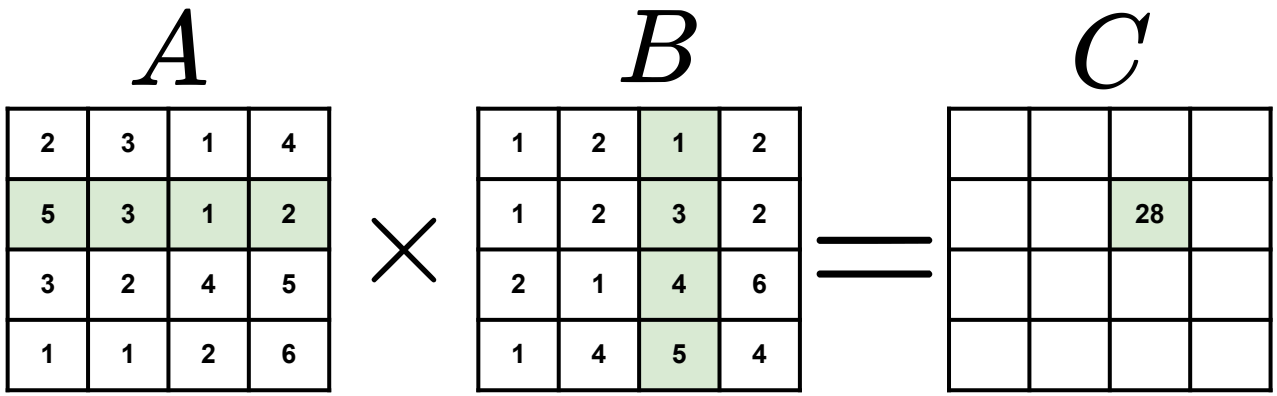
Speaker: Jia-Ming Lin

Outline

- Matrix Multiplication and Applications
- Complete Matrix Multiplication
  - Smaller size, e.g. 32x32
- Block Matrix Multiplication
  - Larger size, e.g. 1024x1024

Matrix Multiplication and Applications

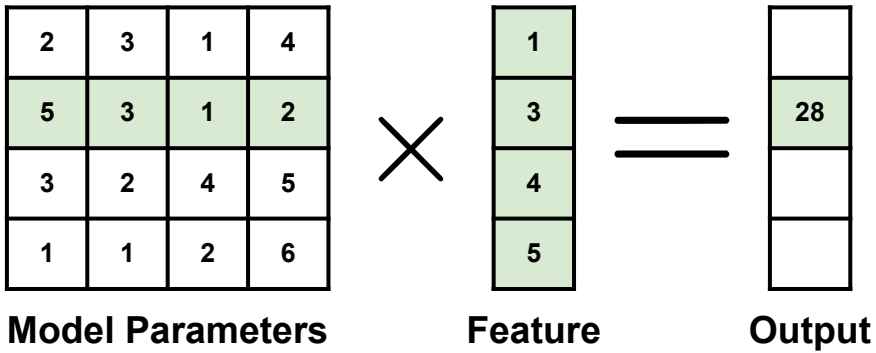
Matrix Multiplication



C[i][j] = inner product of A[i][...] and B[...][j]

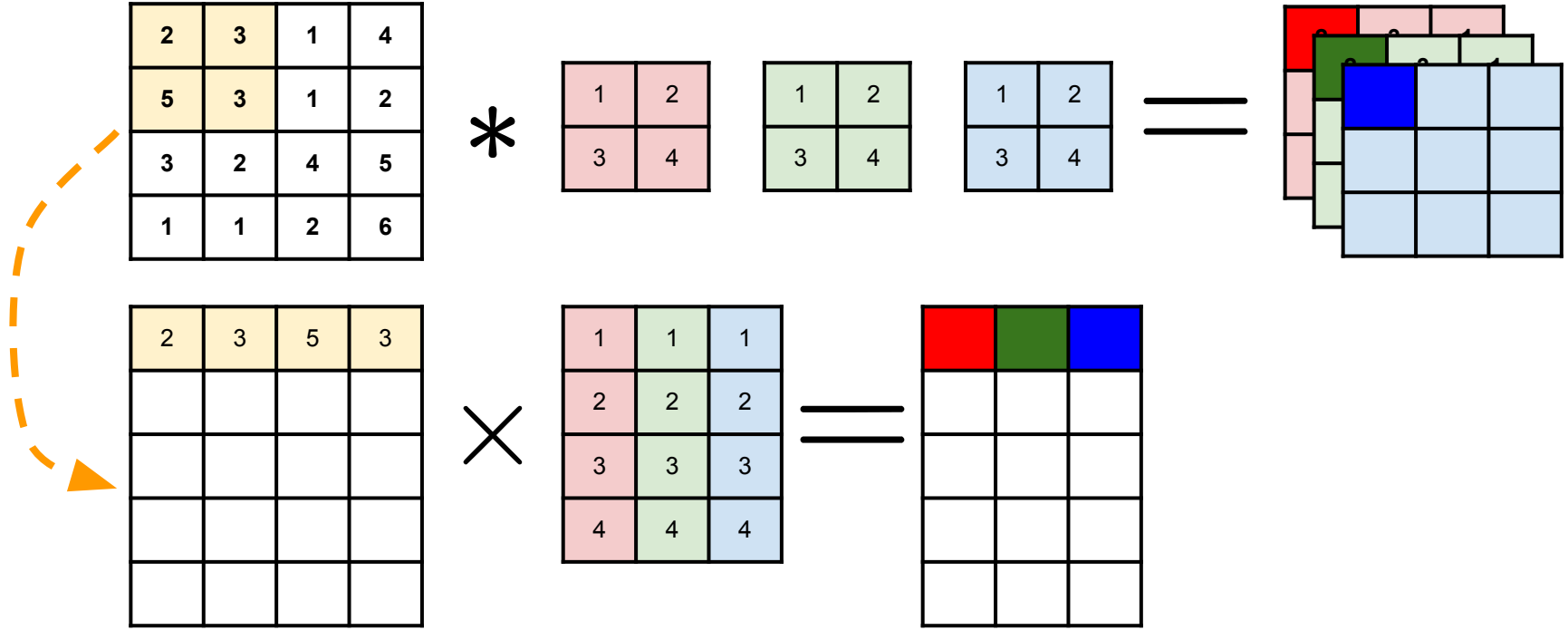
Application

- Computation of Neural Network
  - Fully connected layer



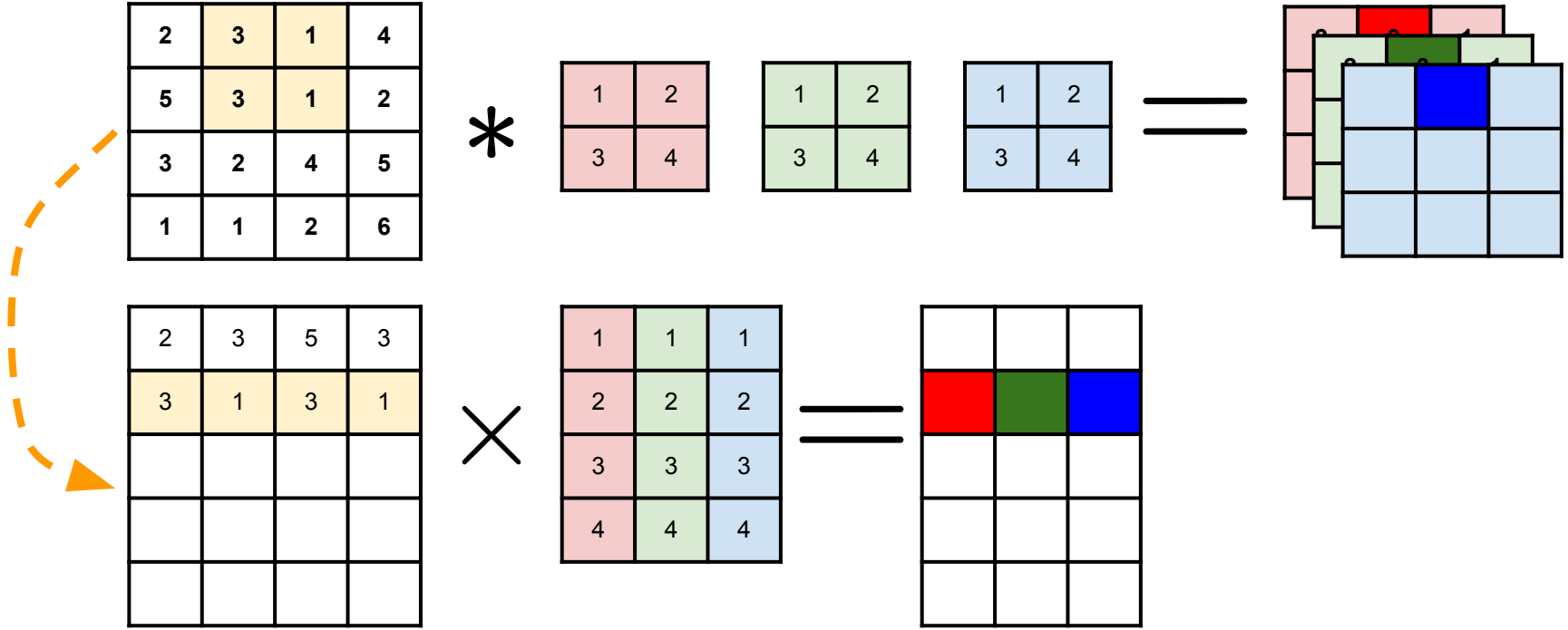
Application

- Computation of Neural Network
  - Convolution Layer



Application

- Computation of Neural Network
  - Convolution Layer



Memory Hierarchy

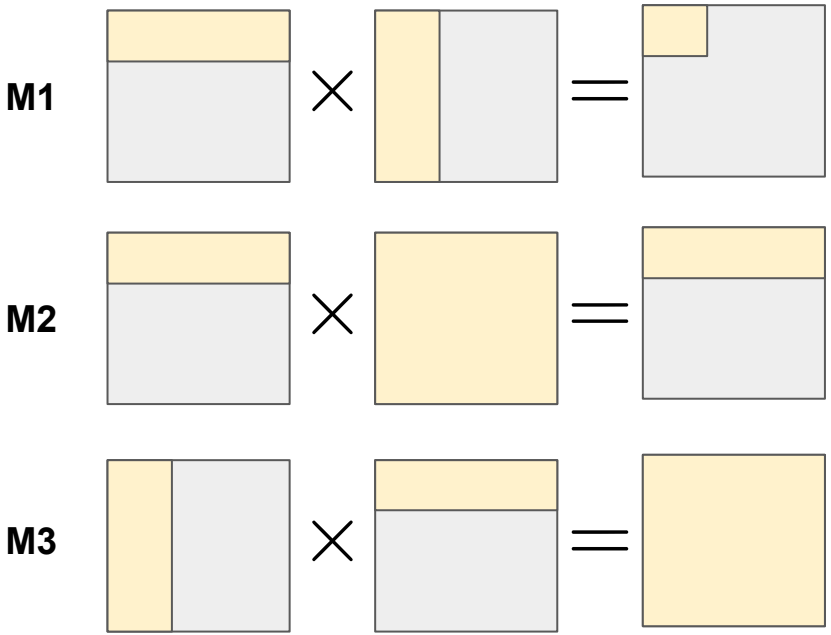
	External Memory	BRAM	FFs
count	1-4	thousands	millions
size	GBytes	KBytes	Bits
total size	GBytes	MBytes	100s of KBytes
width	8-64	1-16	1
total bandwidth	GBytes/sec	TBytes/sec	100s of TBytes/sec

- **External Memory**: highest density, lowest bandwidth
- **FFs**: highest total bandwidth, limited amount of data storage capability.
- **BRAM**: intermediate value between external memory and FFs

Complete Matrix Multiplication

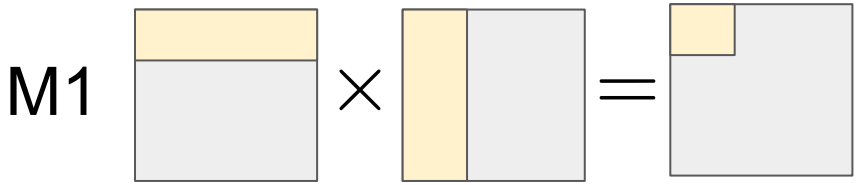
Methodologies

- Comparison
  - Latency, Resource Consumption

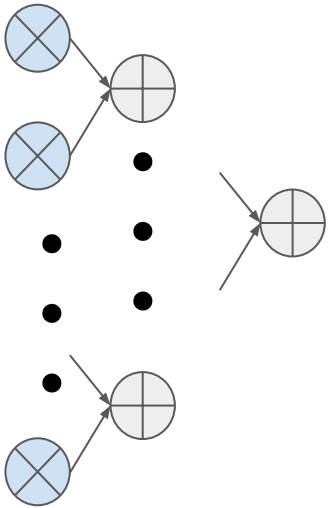


Suppose

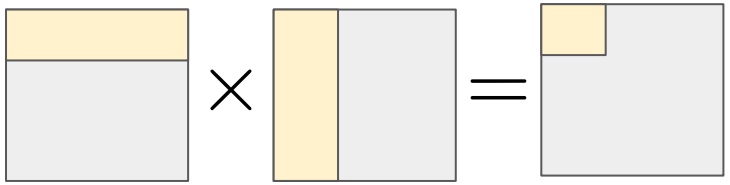
- Matrix size = 32x32
- Numbers in 16-bit integer
- Partial sum in 32-bit integer



- Compute one inner product in adder tree
  - One output: need log(32) = 5 cycles

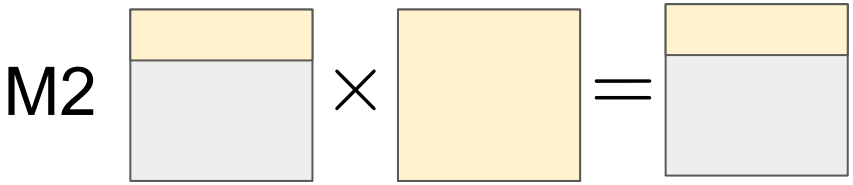


```
void matrixmul(int A[N][M], int B[M][P], int AB[N][P]) {
    #pragma HLS ARRAY_RESHAPE variable=A complete dim=2
    #pragma HLS ARRAY_RESHAPE variable=B complete dim=1
    /* for each row and column of AB */
    row: for(int i = 0; i < N; ++i) {
        col: for(int j = 0; j < P; ++j) {
            #pragma HLS PIPELINE II=1
            /* compute (AB)i,j */
            int ABij = 0;
            product: for(int k = 0; k < M; ++k) {
                ABij += A[i][k] * B[k][j];
            }
            AB[i][j] = ABij;
        }
    }
}
```

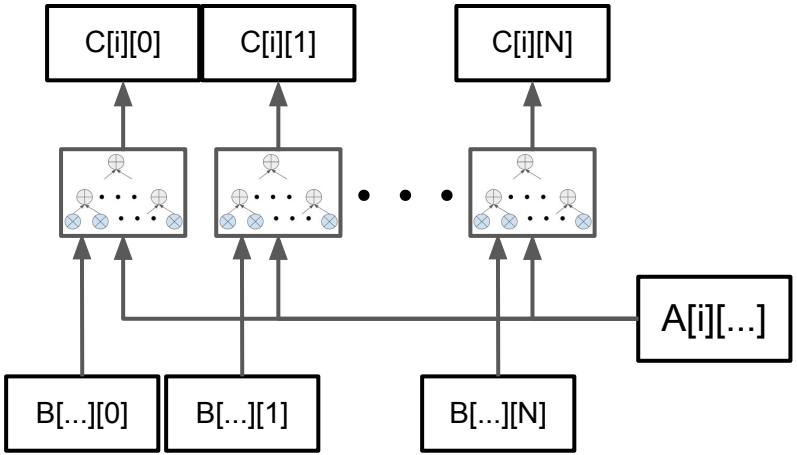
M1 

- Latency:
  - Pipeline Depth = 5
  - # of Iterations = 32x32
  - Initialization Interval = 1
  - Latency = 1024+4 = 1028
- Resource Consumption
  - DSP = 32

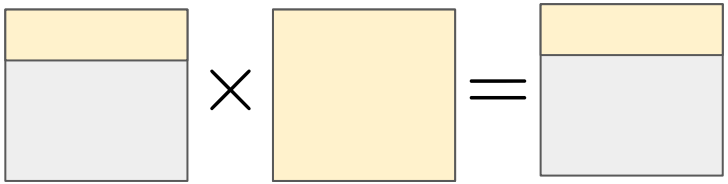
```
void matrixmul(int A[N][M], int B[M][P], int AB[N][P]) {
    #pragma HLS ARRAY_RESHAPE variable=A complete dim=2
    #pragma HLS ARRAY_RESHAPE variable=B complete dim=1
    /* for each row and column of AB */
    row: for(int i = 0; i < N; ++i) {
        col: for(int j = 0; j < P; ++j) {
            #pragma HLS PIPELINE II=1
            /* compute (AB)i,j */
            int ABij = 0;
            product: for(int k = 0; k < M; ++k) {
                ABij += A[i][k] * B[k][j];
            }
            AB[i][j] = ABij;
        }
    }
}
```



- Multiple parallel adder trees
  - One output: need log(32) = 5 cycles

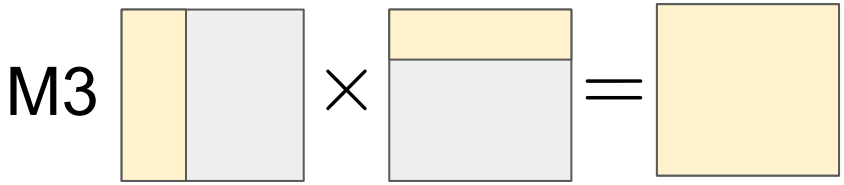



```
void inner_product(A_i, B_j, C_ij){
#pragma HLS INLINE off
        for(k = 0...N){ // Unroll
                C_ij += A_i[k] * B_j[k]
        }
}

void mat_mul(A[N][N], B[N][N], C[N][N]){
#pragma HLS ALLOCATION instances=inner_product
limit=32 function
        for(i = 0…N){
                for(j = 0...N){ // Unroll
                        inner_product(A[i][...], B[...][j], C[i][j])
                }
        }
}
```

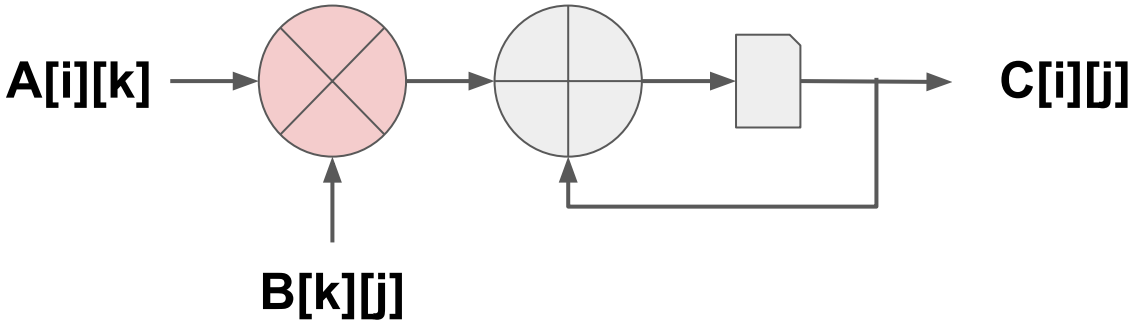
M2 

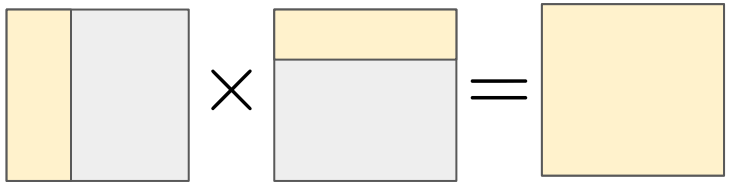
- Latency:
  - Pipeline Depth = 5
  - # of Iterations = 32
  - Initialization Interval = 5
  - Latency = 32 * 5 = 160 (cycles)
- Resource Consumption
  - DSP = 32*32 = 1024

```
void inner_product(A_i, B_j, C_ij){
#pragma HLS INLINE off
        for(k = 0...N){ // Unroll
                C_ij += A_i[k] * B_j[k]
        }
}

void mat_mul(A[N][N], B[N][N], C[N][N]){
#pragma HLS ALLOCATION instances=inner_product
limit=32 function
        for(i = 0…N){
                for(j = 0...N){ // Unroll
                        inner_product(A[i][...], B[...][j], C[i][j])
                }
        }
}
```

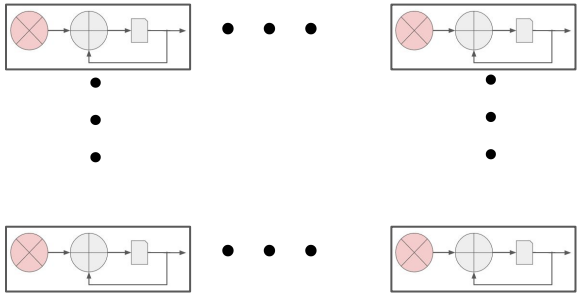


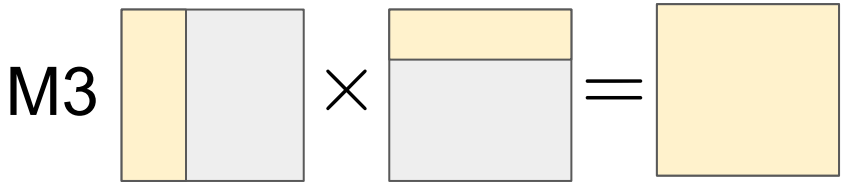
- Parallelize Partial Sum Computations
  - Parallelized outer product



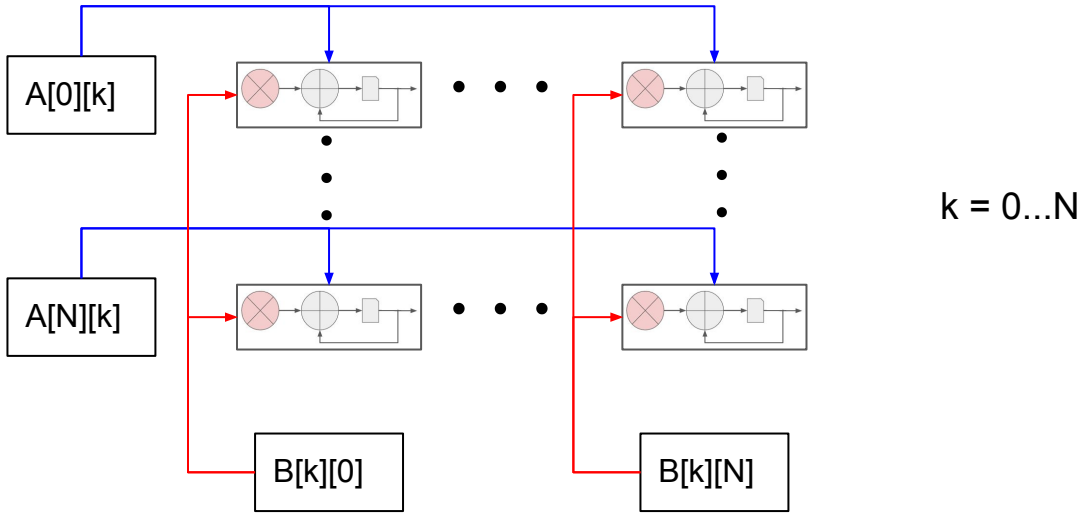
M3 

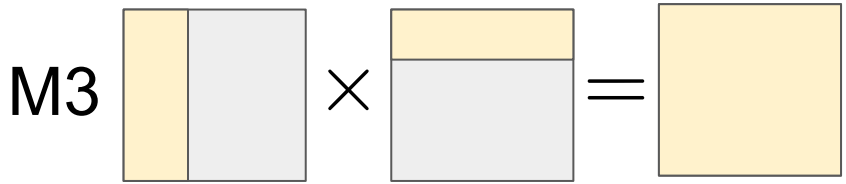
- Parallelize Partial Sum Computations



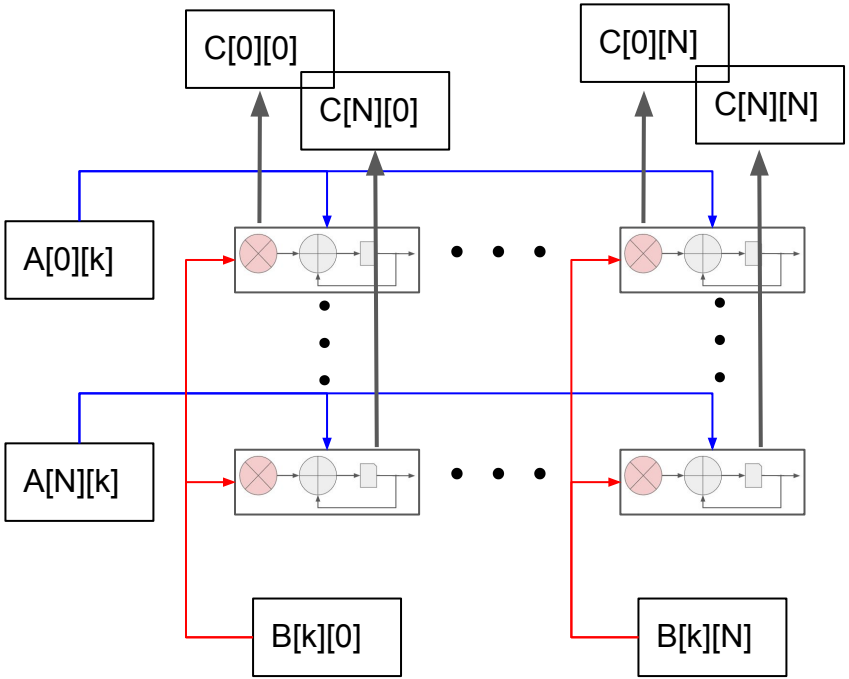


- Parallelize Partial Sum Computations

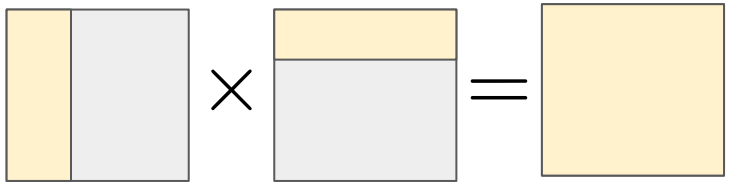




- Parallelize Partial Sum Computations




```
int32 PE(a, b, c){
        return a*b+c
}

void mat_mul(A[N][N], B[N][N], C[N][N]){
#pragma HLS ALLOCATION instances=PE limit=1024 function
        for(k=0...N){ // pipeline
                A_col = A[...][k];
                B_row = B[k][...];
                for(i = 0...N){ // unroll
                        for(j = 0...N){ // unroll
                                C[i][j] = PE(A_col[i], B_row[j], C[i][j]);
                        }
                }
        }
}
```

M3 

- Latency:
  - Pipeline Depth = 2
  - # of iterations = 32
  - Initialize interval = 1
  - Latency = 1+32 = 33 (cycles)
- Resource Consumption
  - DSP = 32*32 = 1024

```
int32 PE(a, b, c){
        return a*b+c
}

void mat_mul(A[N][N], B[N][N], C[N][N]){
#pragma HLS ALLOCATION instances=PE limit=1024 function
        for(k=0...N){ // pipeline
                A_col = A[...][k];
                B_row = B[k][...];
                for(i = 0...N){ // unroll
                        for(j = 0...N){ // unroll
                                C[i][j] = PE(A_col[i], B_row[j], C[i][j]);
                        }
                }
        }
}
```