Vivado/Vitis HLS Optimizations

Daniel Jiménez-Gonzalez (UPC/BSC)
Xavier Martorell (UPC/BSC)

Daniel.jimenez-gonzalez@bsc.es

Programming Models and OS for FPGA-based Architectures team

UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

18/3/2026

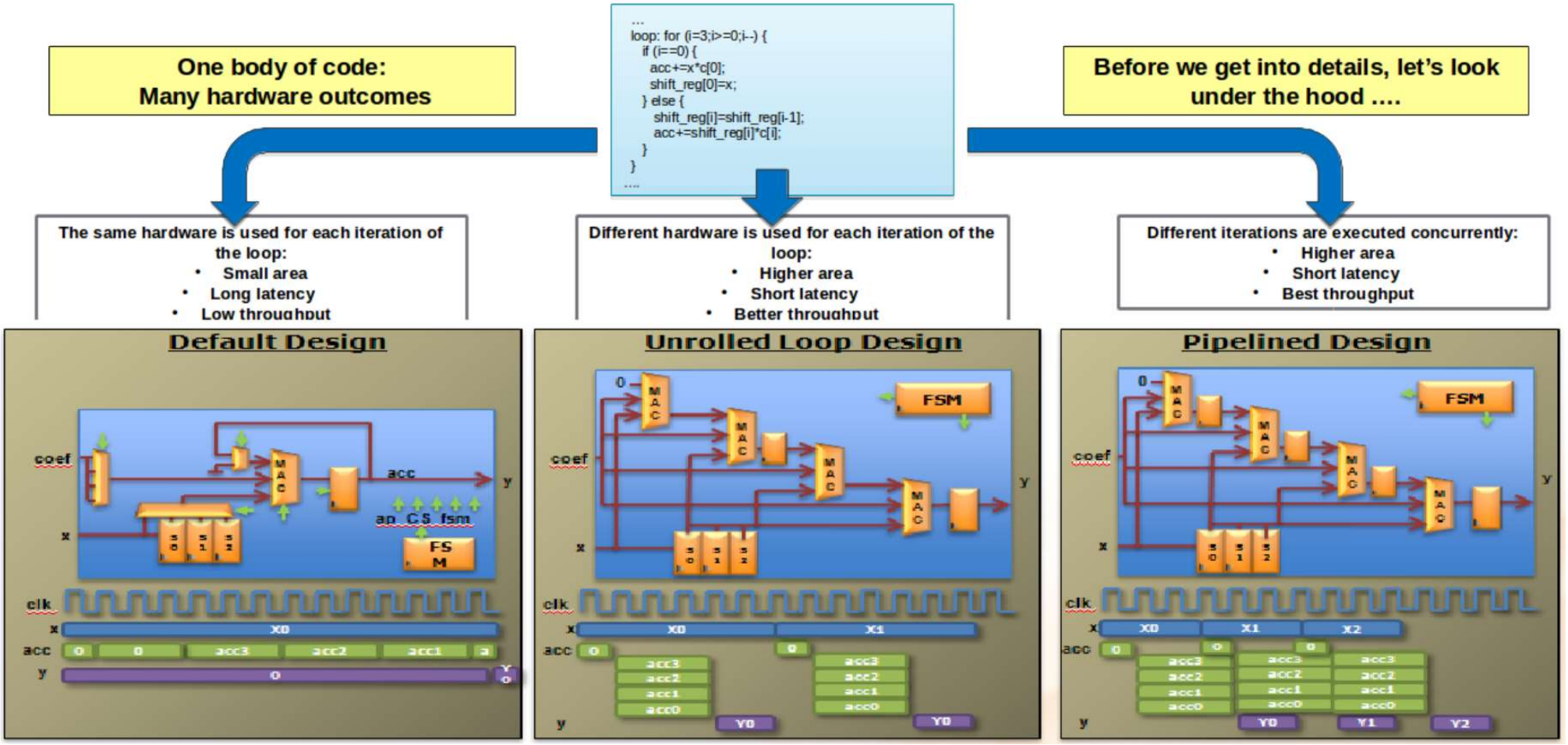
Agenda

- Vivado/Vitis HLS Optimizations
  - OmpSs Example and basic optimizations
    - Loop Unrolling
    - Pipelining
    - Array partition
    - Example: Tiled Matrix Multiplication
  - Porting Matrix Multiplication
  - FPGA wide port to increase data movement performance
  - Reverse offloading
  - Advanced optimizations

OmpSs Example: Vivado HLS

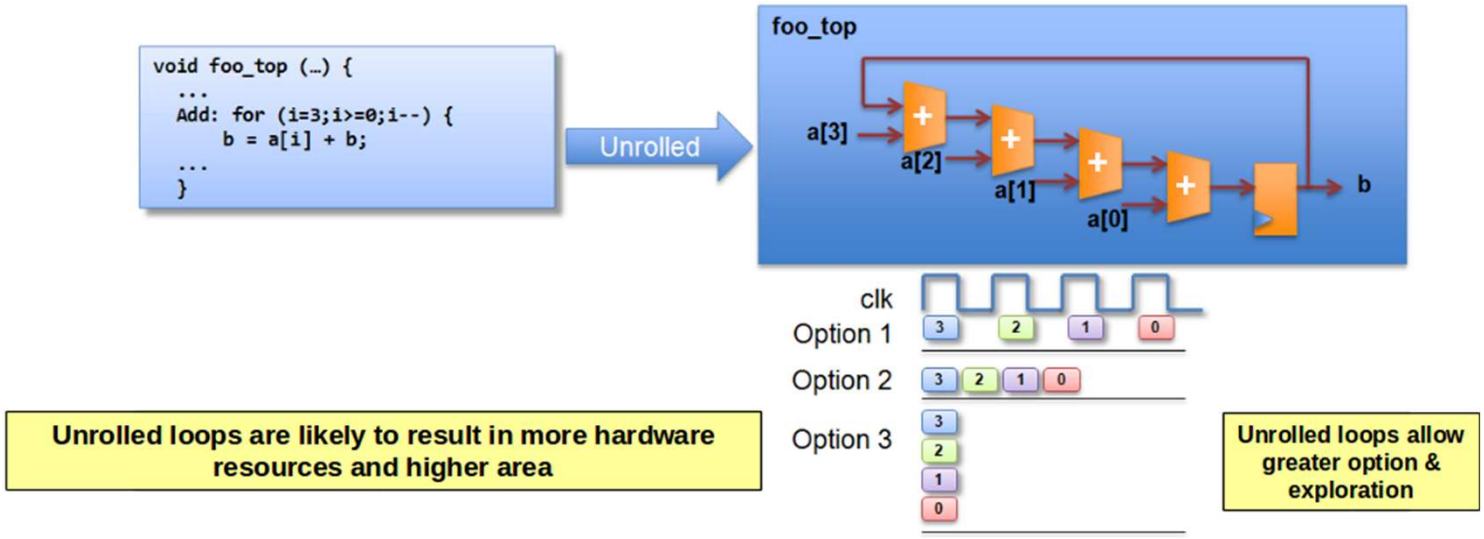
```
void matrix_multiply(float a[BS][BS], float b[BS][BS],float c[BS][BS])
{
#pragma HLS inline
  int const FACTOR = BS/2;
#pragma HLS array_partition variable=a block factor=FACTOR dim=2
#pragma HLS array_partition variable=b block factor=FACTOR dim=1
  // matrix multiplication of a A*B matrix
  for (int ia = 0; ia < BS; ++ia)
    for (int ib = 0; ib < BS; ++ib) {
#pragma HLS PIPELINE II=1
      float sum = 0;
      for (int id = 0; id < BS; ++id)
        sum += a[ia][id] * b[id][ib];
      c[ia][ib] += sum;
    }
}
```

Overall Overview*



* Some information and pictures are extracted from the Xilinx University Program Vivado HLS Presentations

Loop Unrolling

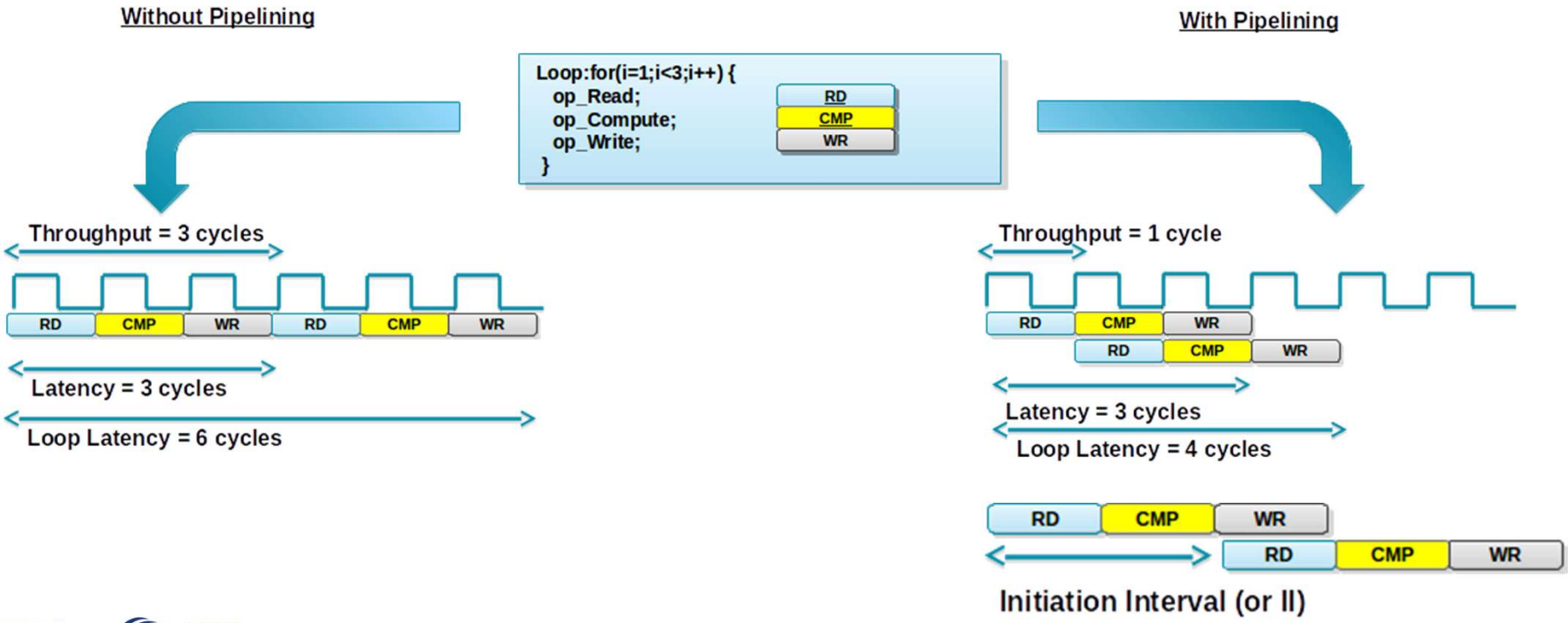


- Option 1: one iteration per cycle
- Option 2: several iterations per cycle in sequential
- Option 3: several iterations per cycle in parallel

- Be careful with the needed resources
- Memory conflicts!!
- #pragma HLS unroll **skip_exit_check** factor=UNROLL_FACTOR

Pipelining

- Several instances of the hardware can operate in pipeline
- Initiation Interval: Number of cycles before you can start a new instance of the hardware



Pipelining: Resources vs Parallelism

- Looking for pipeline
  - With few resources (Left)
- Looking for some parallelism and pipeline:
  - Full unroll of the inner-most loop. Resources needed and throughput of loop L1 (Middle)
- Looking for high parallelism
  - Full unroll of L1 and L2. Too much resources!!! (Right)

```
void foo(in1[ ][ ], in2[ ][ ], ...) {
...
  L1:for(i=1;i<N;i++) {
    L2:for(j=0;j<M;j++) {
#pragma HLS PIPELINE
      out[i][j] = in1[i][j] + in2[i][j];
    }
  }
}
```

1adder, 3 accesses

```
void foo(in1[ ][ ], in2[ ][ ], ...) {
...
  L1:for(i=1;i<N;i++) {
#pragma HLS PIPELINE
    L2:for(j=0;j<M;j++) {
      out[i][j] = in1[i][j] + in2[i][j];
    }
  }
}
```

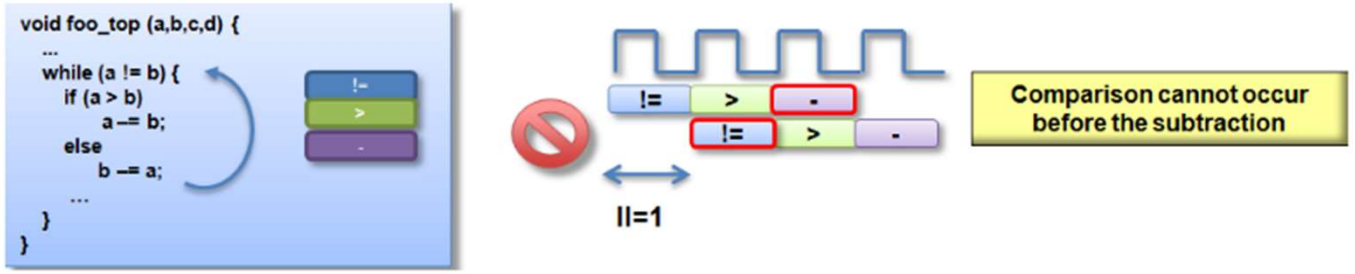
Unrolls L2
M adders, 3M accesses

```
void foo(in1[ ][ ], in2[ ][ ], ...) {
#pragma HLS  PIPELINE
  ...
  L1:for(i=1;i<N;i++) {
    L2:for(j=0;j<M;j++) {
      out[i][j] = in1[i][j] + in2[i][j];
    }
  }
}
```

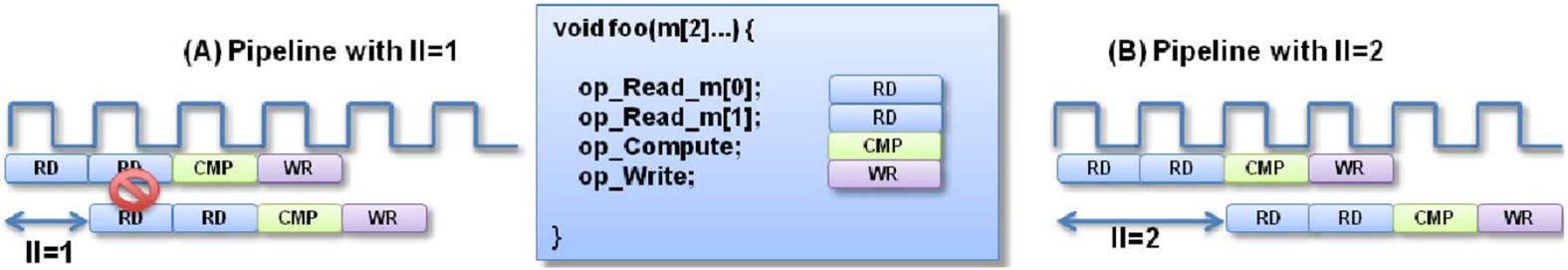
Unrolls L1 and L2
N*M adders, 3(N*M) accesses

Pipelining: Initiation Interval Limitations

- Constant Bounds help…
  - Example: comparison due non known limit…

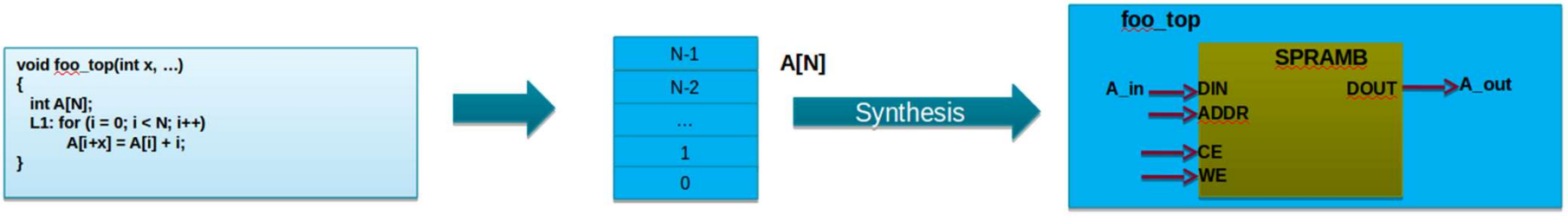


- Memory Accesses can have conflicts
  - Arra Partition can help

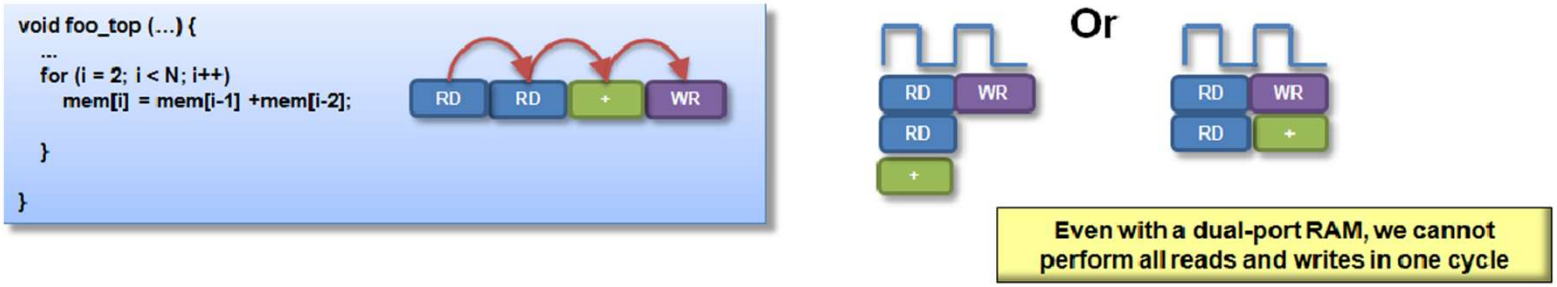


Exploring Data Conflicts: Array Synthesis

- Memory is synthesized (usually) to a BRAM by default

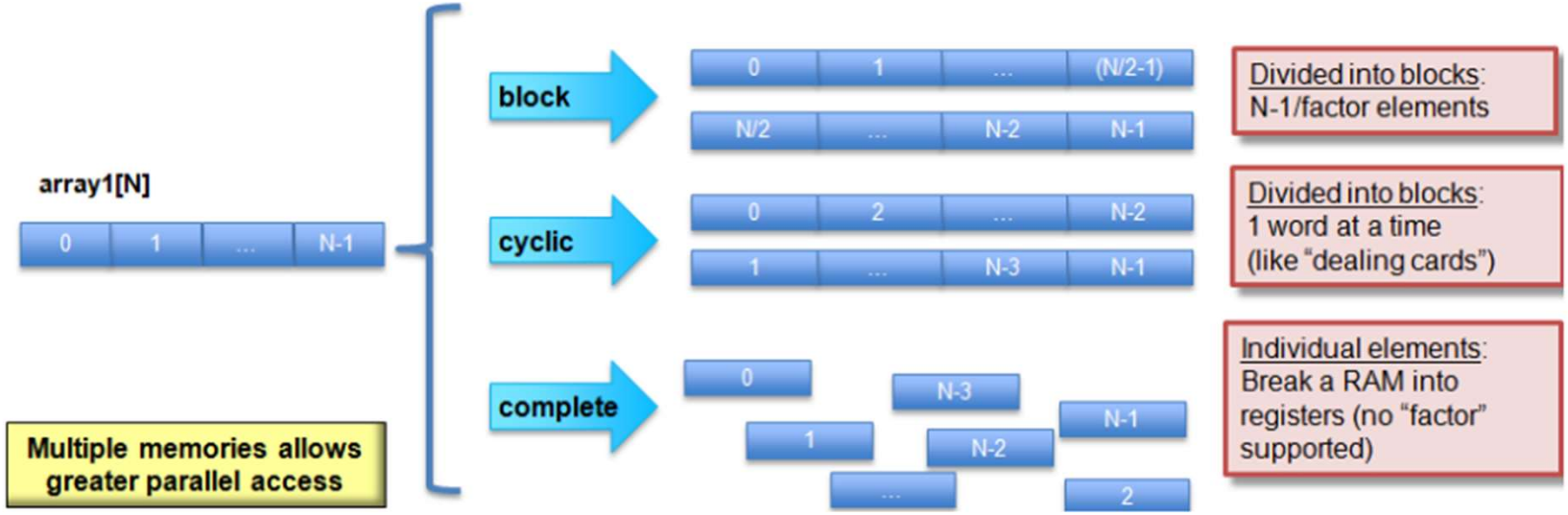


- Memory Accesses can have conflicts
  - Arra Partition can help



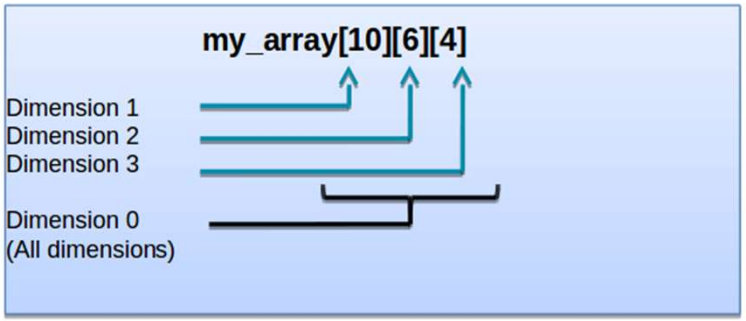
Array Partition

- Memory Accesses can have conflicts
  - Arra Partition can help
  - #pragma HLS ARRAY_PARTITION variable=name dim=X type factor=FACTOR



Array Partitioning: Array Dimensions





my_array[10][6][4] partition dimension 3

my_array_0[10][6]
my_array_1[10][6]
my_array_2[10][6]
my_array_3[10][6]

my_array[10][6][4] partition dimension 1

my_array_0[6][4]
my_array_1[6][4]
my_array_2[6][4]
my_array_3[6][4]
my_array_4[6][4]
my_array_5[6][4]
my_array_6[6][4]
my_array_7[6][4]
my_array_8[6][4]
my_array_9[6][4]

my_array[10][6][4] partition dimension 0 10x6x4 = 240 individual registers