ETH zürich

Programming FPGAs: a Software Programmer’s Perspective

Wenqi Jiang, Dario Korolija, Gustavo Alonso
Systems Group, Dept. of Computer Science, ETH Zürich

2023/06/23

Systems@ETH zürich

Roadmap

FPGAs and Non-Von-Neumann architectures

Fundamental differences between software and hardware programming

Hardware Description Languages (HDL)

High-Level Synthesis (HLS): programming FPGAs with C/C++

FPGAs and Non-Von-Neumann architectures

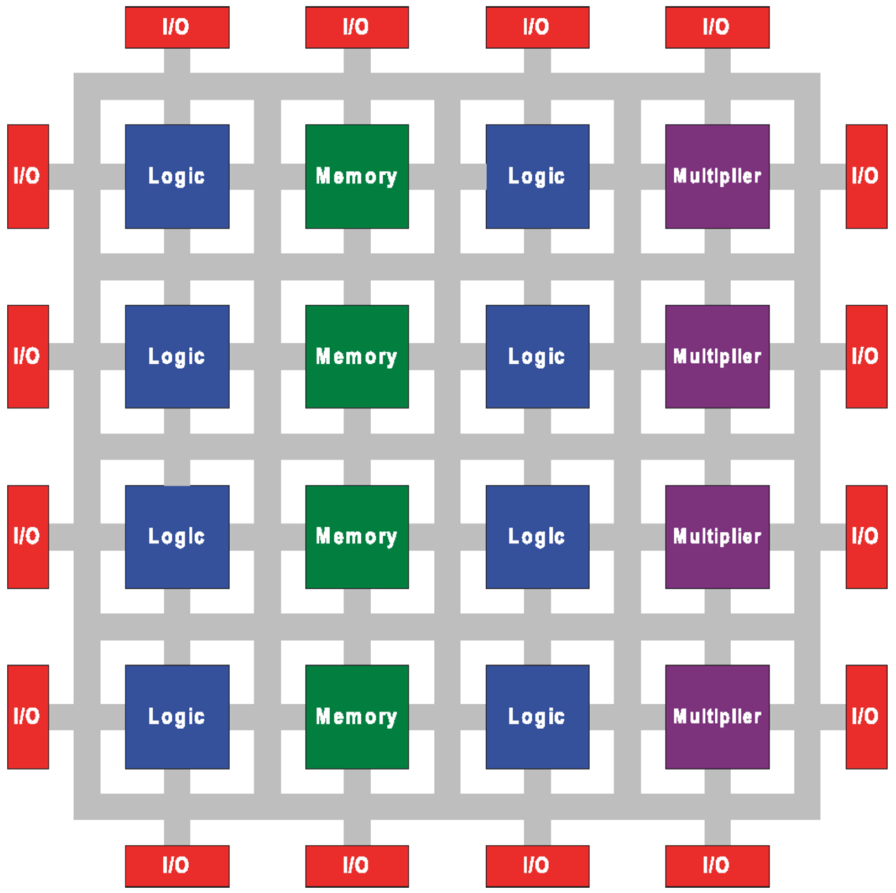
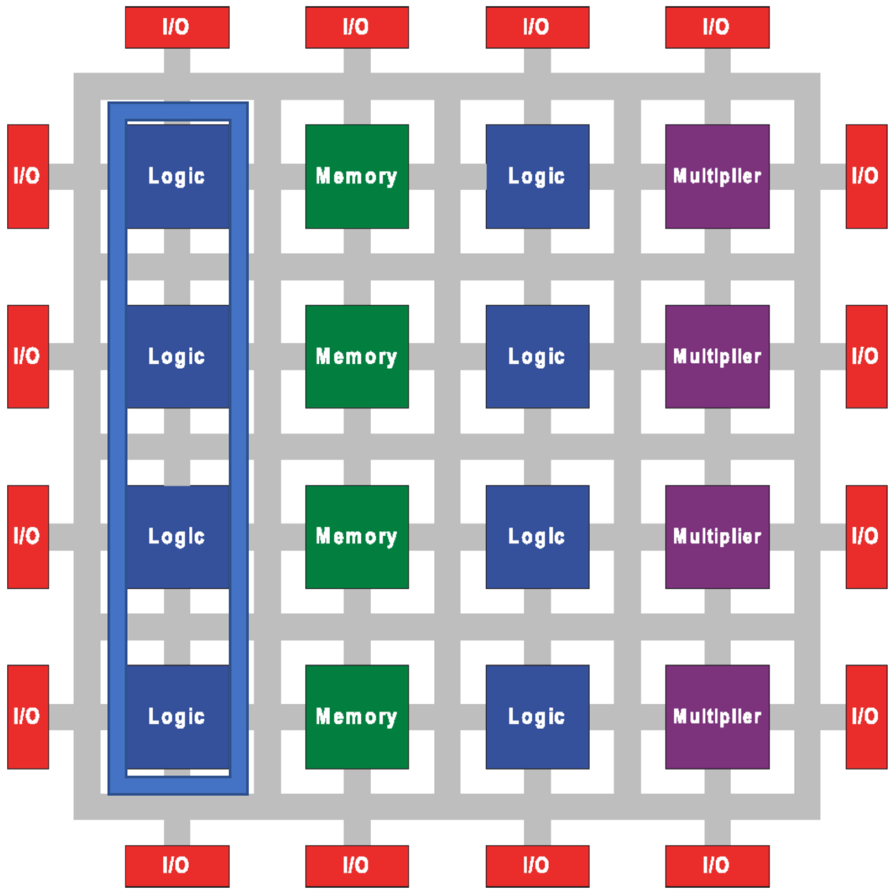


Figure source: Kuon, Ian, Russell Tessier, and Jonathan Rose. "FPGA architecture: Survey and challenges." *Foundations and Trends® in Electronic Design Automation* 2.2 (2008): 135-253.

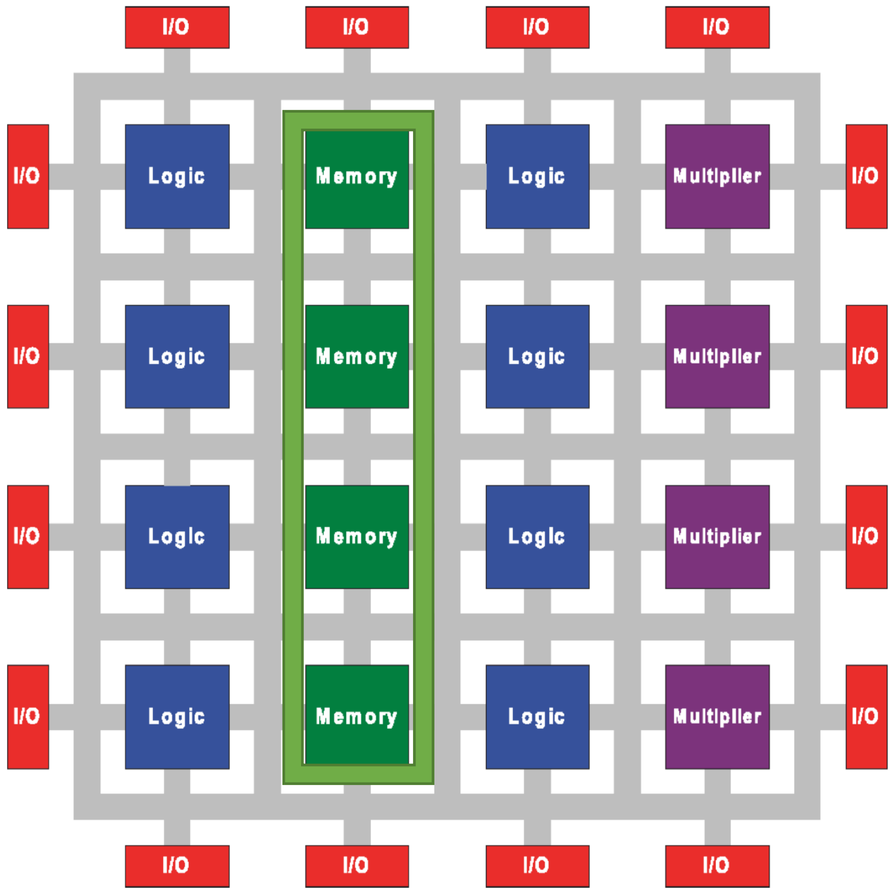
FPGAs and Non-Von-Neumann architectures



- Logic Block contains LUT and FF
- Look-up table (LUT): This element performs logic operations
- Flip-Flop (FF): This register element stores the result of the LUT

Figure source: Kuon, Ian, Russell Tessier, and Jonathan Rose. "FPGA architecture: Survey and challenges." *Foundations and Trends® in Electronic Design Automation* 2.2 (2008): 135-253.

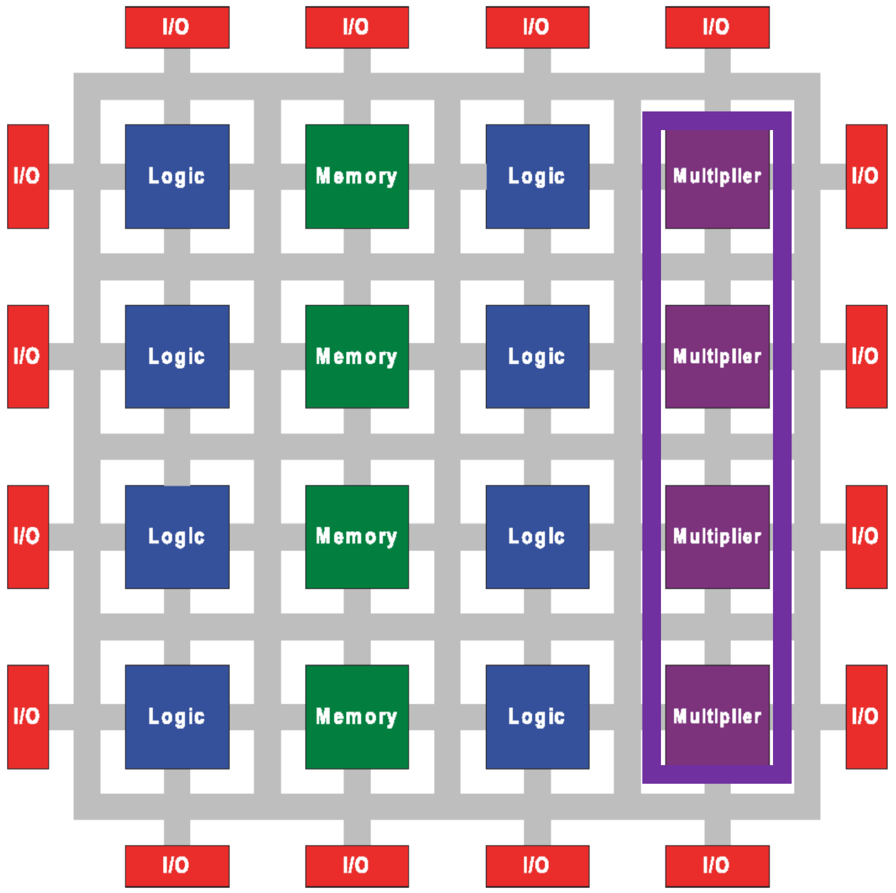
FPGAs and Non-Von-Neumann architectures



- Block-RAM (fast on-chip SRAM)
- Similar to CPU cache, but the user has full control of the BRAM behavior

Figure source: Kuon, Ian, Russell Tessier, and Jonathan Rose. "FPGA architecture: Survey and challenges." *Foundations and Trends® in Electronic Design Automation* 2.2 (2008): 135-253.

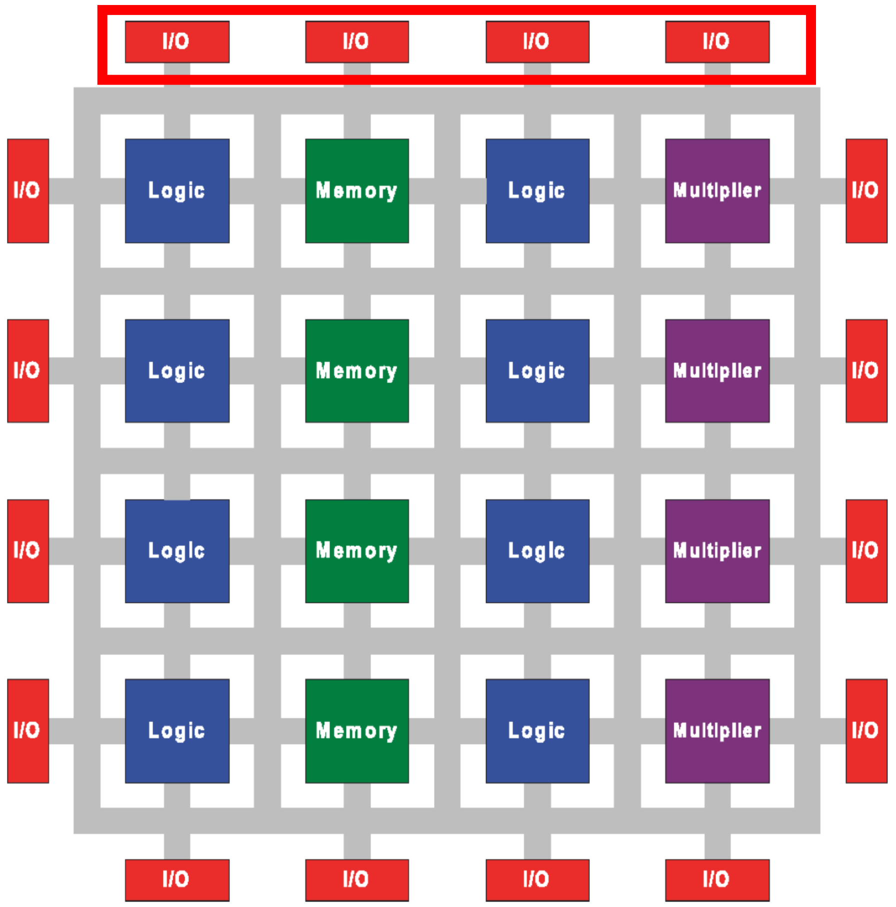
FPGAs and Non-Von-Neumann architectures



- DSP (multiplier): floating-point cores
- Implementing floating-point operations using LUT is less efficient

Figure source: Kuon, Ian, Russell Tessier, and Jonathan Rose. "FPGA architecture: Survey and challenges." *Foundations and Trends® in Electronic Design Automation* 2.2 (2008): 135-253.

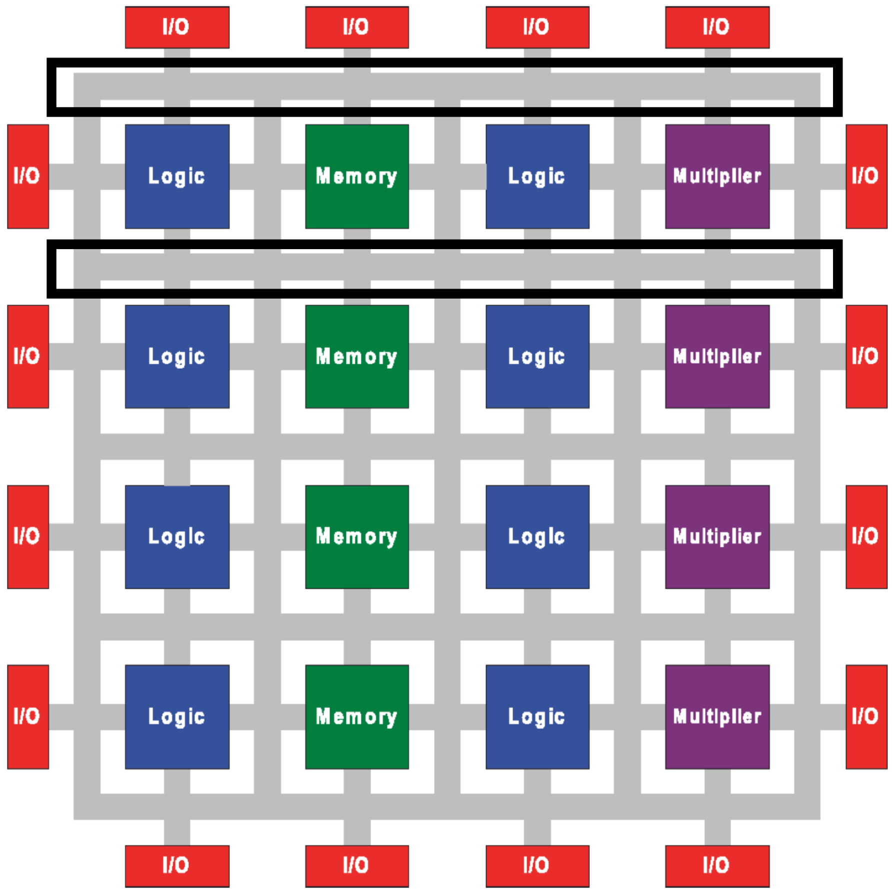
FPGAs and Non-Von-Neumann architectures



- Input/Output (I/O) pads: These physically available ports get signals in and out of the FPGA.
- For example: DRAM, network, PCIe, etc.

Figure source: Kuon, Ian, Russell Tessier, and Jonathan Rose. "FPGA architecture: Survey and challenges." *Foundations and Trends® in Electronic Design Automation* 2.2 (2008): 135-253.

FPGAs and Non-Von-Neumann architectures

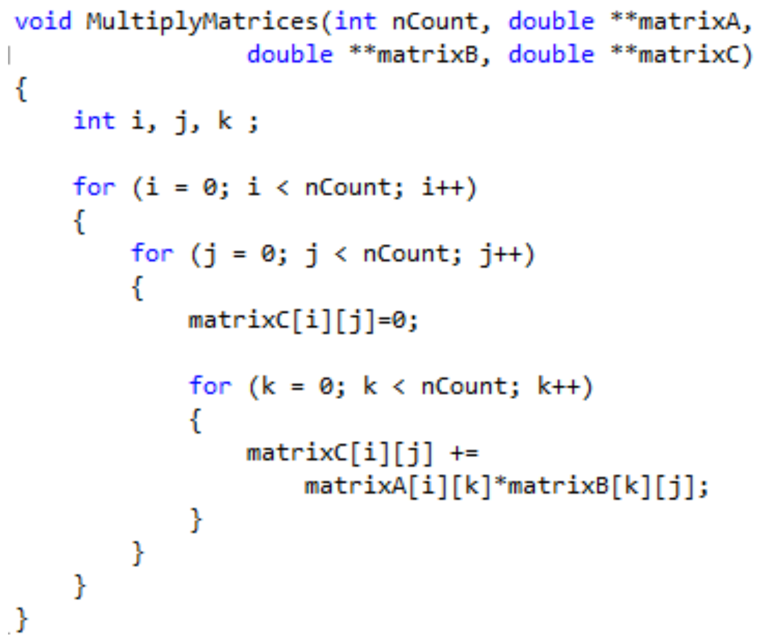


- Wires and switches: These elements connect logic blocks, on-chip memory, DSPs, and I/O blocks to each another

Figure source: Kuon, Ian, Russell Tessier, and Jonathan Rose. "FPGA architecture: Survey and challenges." *Foundations and Trends® in Electronic Design Automation* 2.2 (2008): 135-253.

Programming software vs FPGAs

- CPU: describing the program behavior using an imperative or functional programming language
- The program is compiled to assemblies and executed on a Von-Neumann architecture instruction by instruction
- Example: matrix multiplication



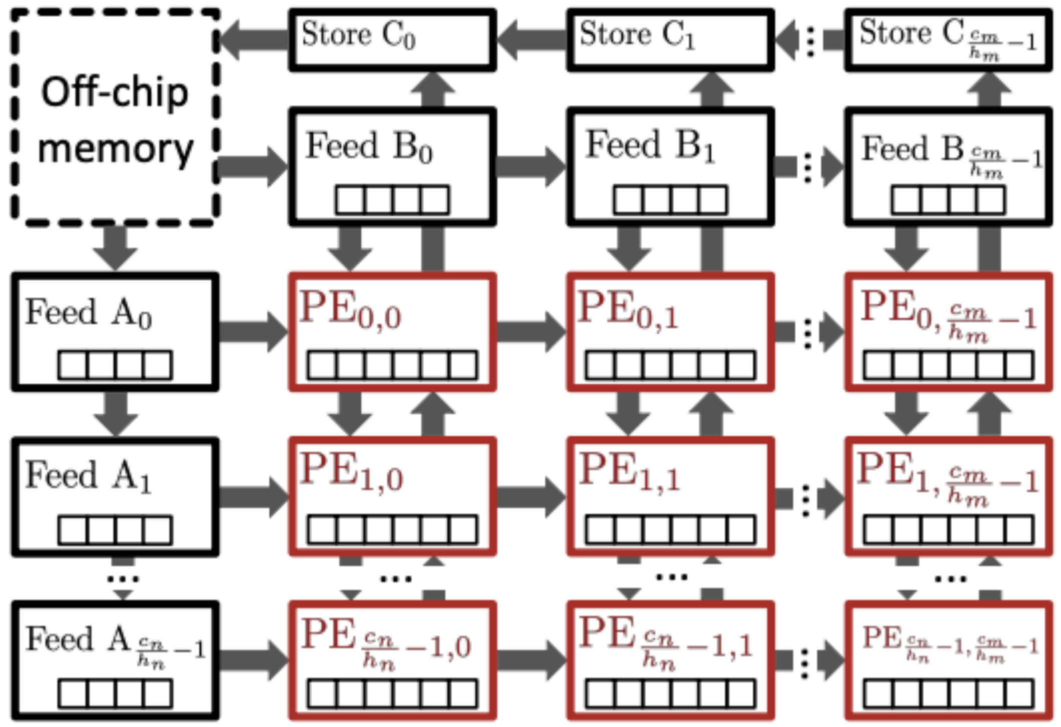
```
void MultiplyMatrices(int nCount, double **matrixA,
                double **matrixB, double **matrixC)
{
    int i, j, k ;

    for (i = 0; i < nCount; i++)
    {
        for (j = 0; j < nCount; j++)
        {
            matrixC[i][j]=0;

            for (k = 0; k < nCount; k++)
            {
                matrixC[i][j] +=
                    matrixA[i][k]*matrixB[k][j];
            }
        }
    }
}
```

Programming software vs FPGAs

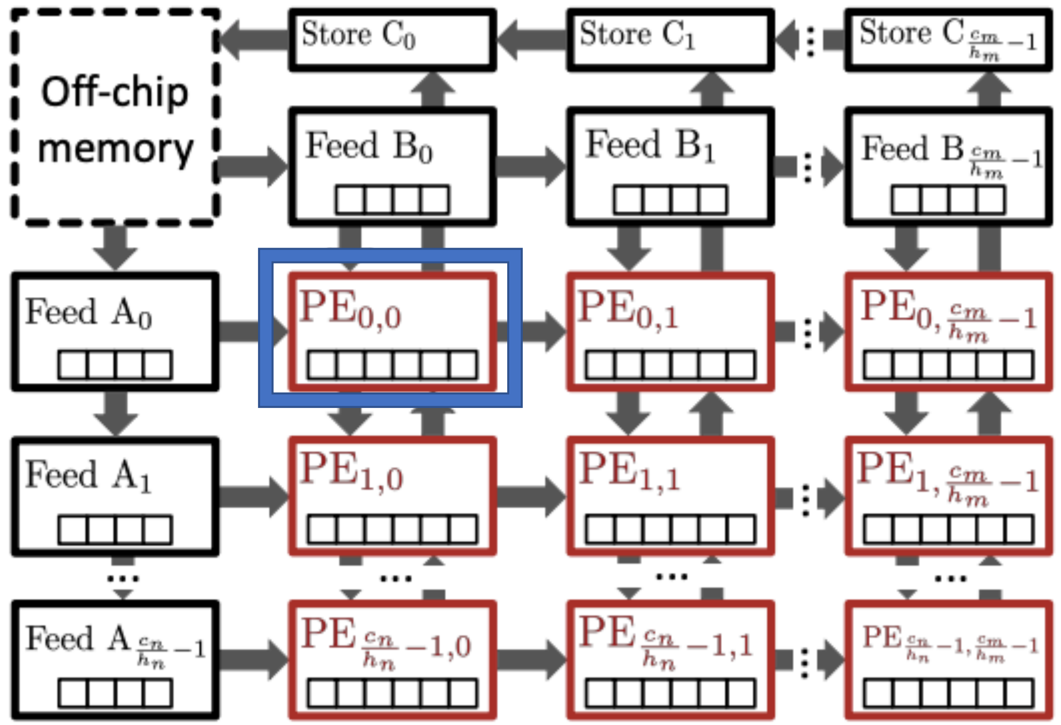
- FPGA: describing a hardware micro-architecture in your mind using some programming languages



Johannes de Fine Licht, Grzegorz Kwasniewski, and Torsten Hoefler. "Flexible communication avoiding matrix multiplication on FPGA with high-level synthesis." *Proceedings of the 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. 2020.

Programming software vs FPGAs

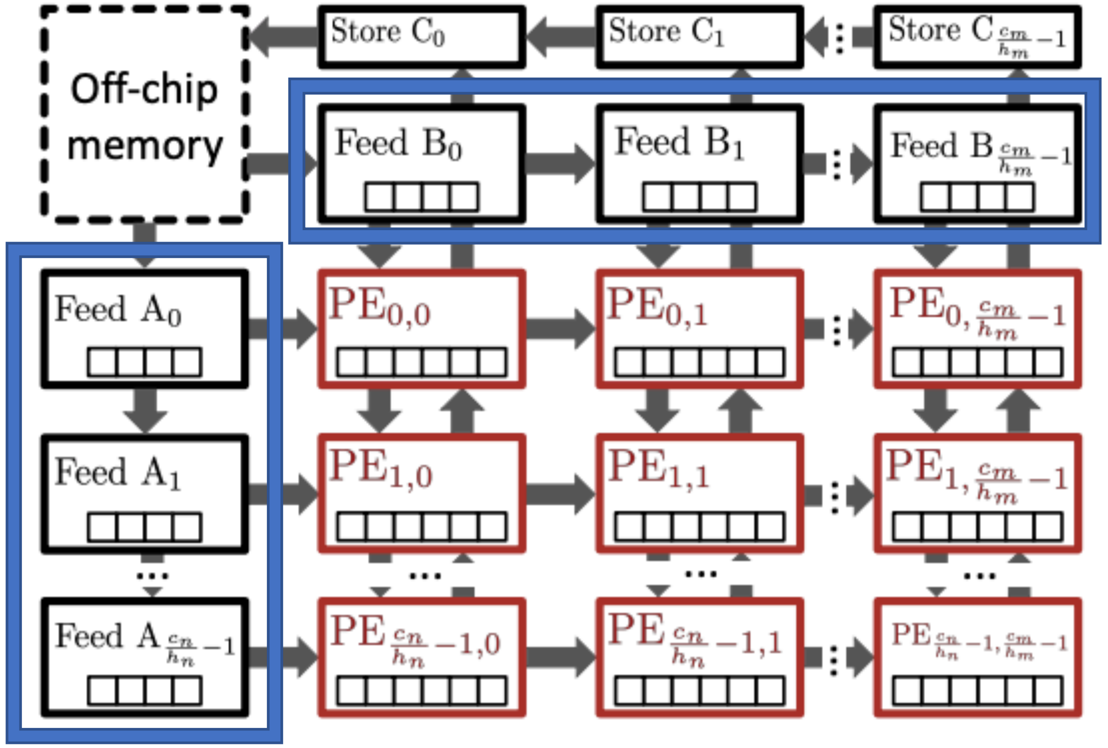
- FPGA: describing a hardware micro-architecture in your mind using some programming languages



A processing element (PE) is responsible for computing a part of the matrix multiplication

Programming software vs FPGAs

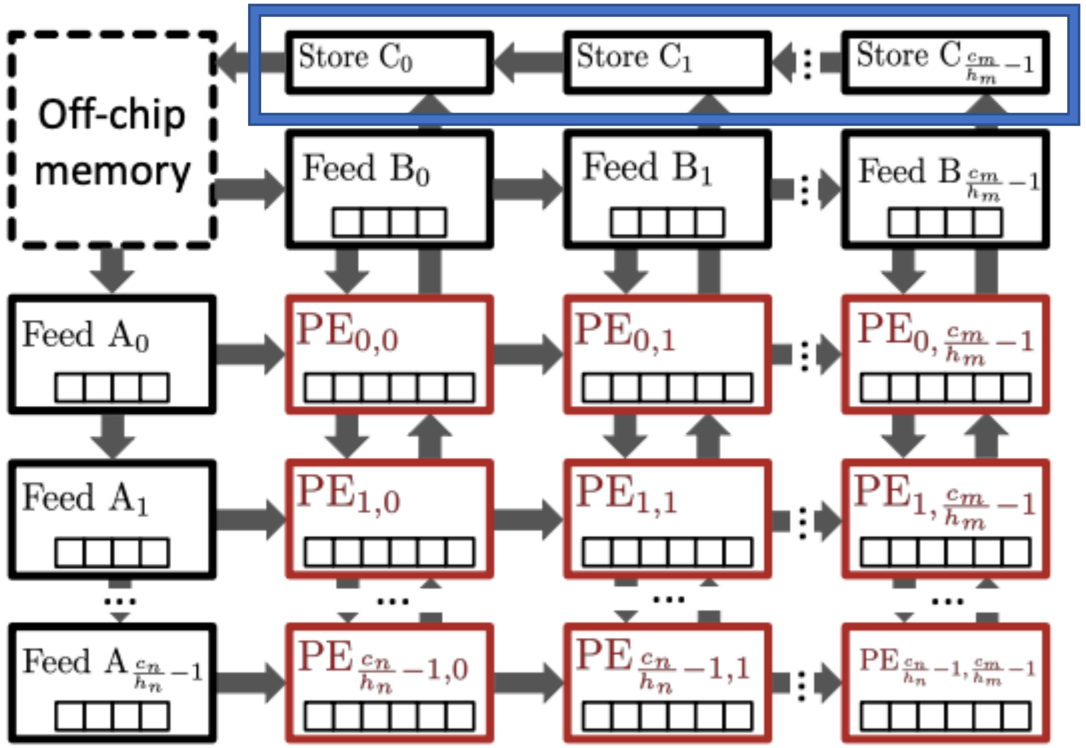
- FPGA: describing a hardware micro-architecture in your mind using some programming languages



Read units feed data (both matrices) from DRAM to the processing elements

Programming software vs FPGAs

- FPGA: describing a hardware micro-architecture in your mind using some programming languages

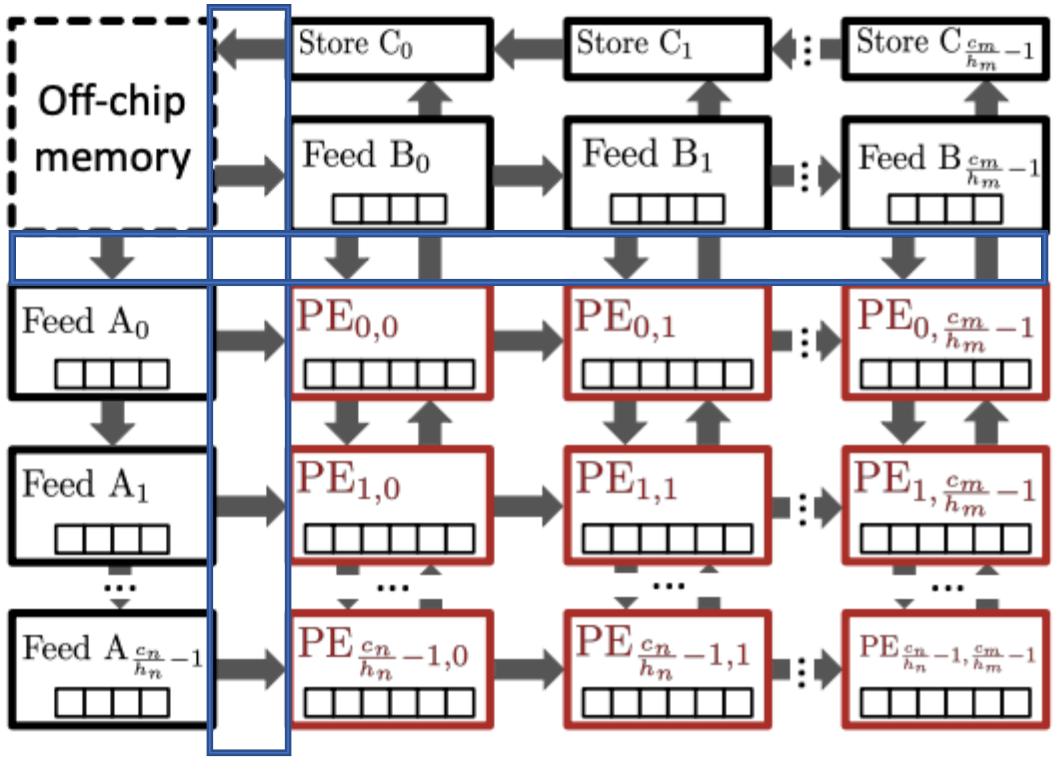


Off-chip memory
Store C_0
Store C_1
Store $C_{\frac{c_m}{h_m}-1}$
Feed B_0
Feed B_1
Feed $B_{\frac{c_m}{h_m}-1}$
Feed A_0
Feed A_1
Feed $A_{\frac{c_n}{h_n}-1}$
$PE_{0,0}$
$PE_{0,1}$
$PE_{0,\frac{c_m}{h_m}-1}$
$PE_{1,0}$
$PE_{1,1}$
$PE_{1,\frac{c_m}{h_m}-1}$
$PE_{\frac{c_n}{h_n}-1,0}$
$PE_{\frac{c_n}{h_n}-1,1}$
$PE_{\frac{c_n}{h_n}-1,\frac{c_m}{h_m}-1}$

Write units store the computed results back to DRAM

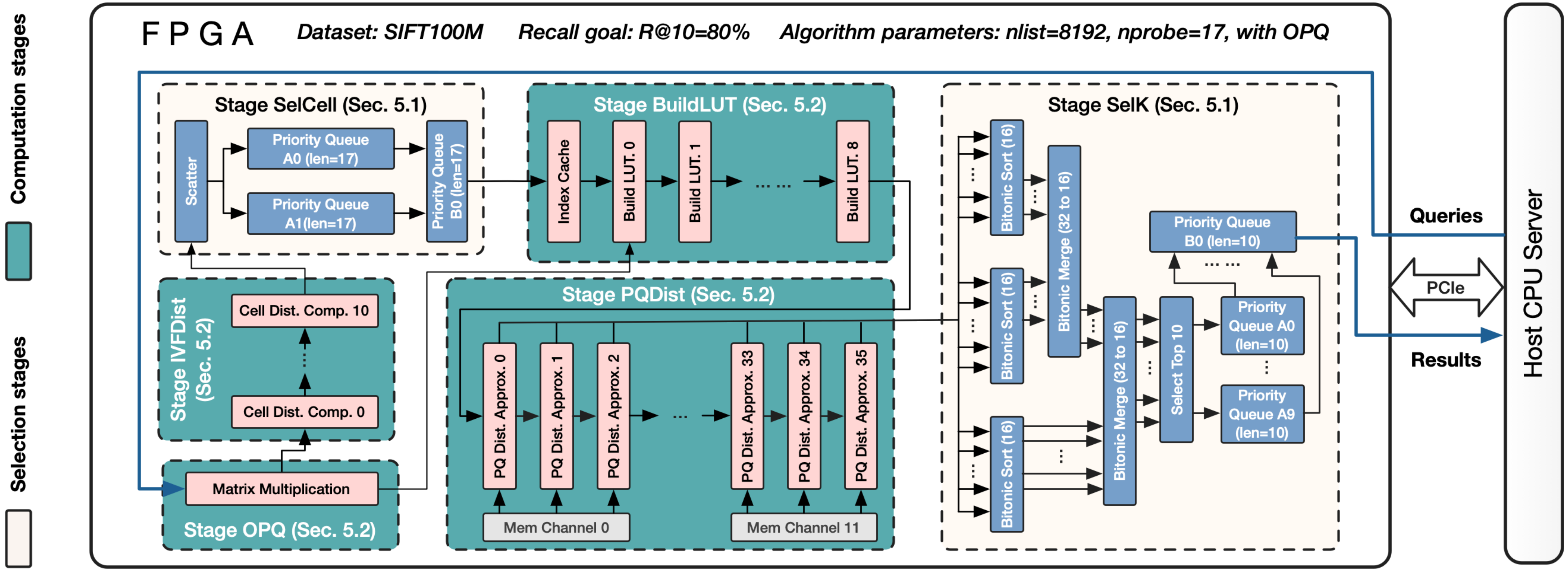
Programming software vs FPGAs

- FPGA: describing a hardware micro-architecture in your mind using some programming languages



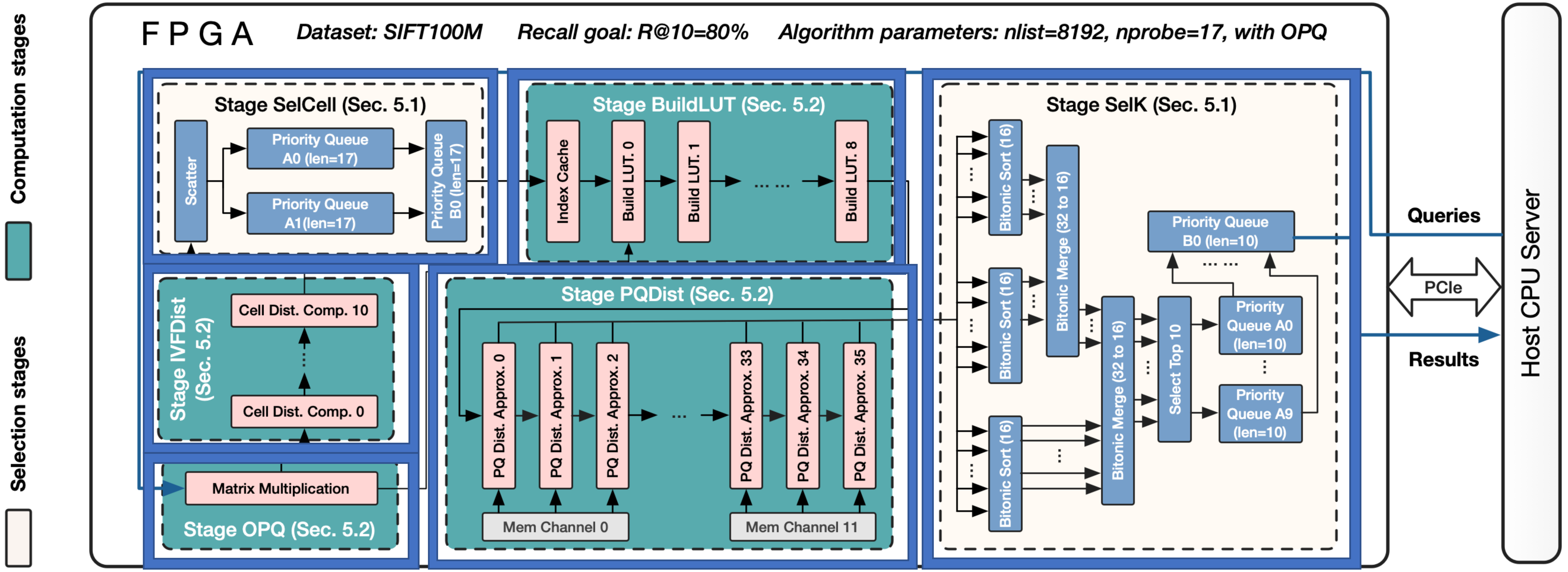
On-chip data movements are implemented by FIFOs (pipes)

Programming FPGAs: a more complex example (ANNS)



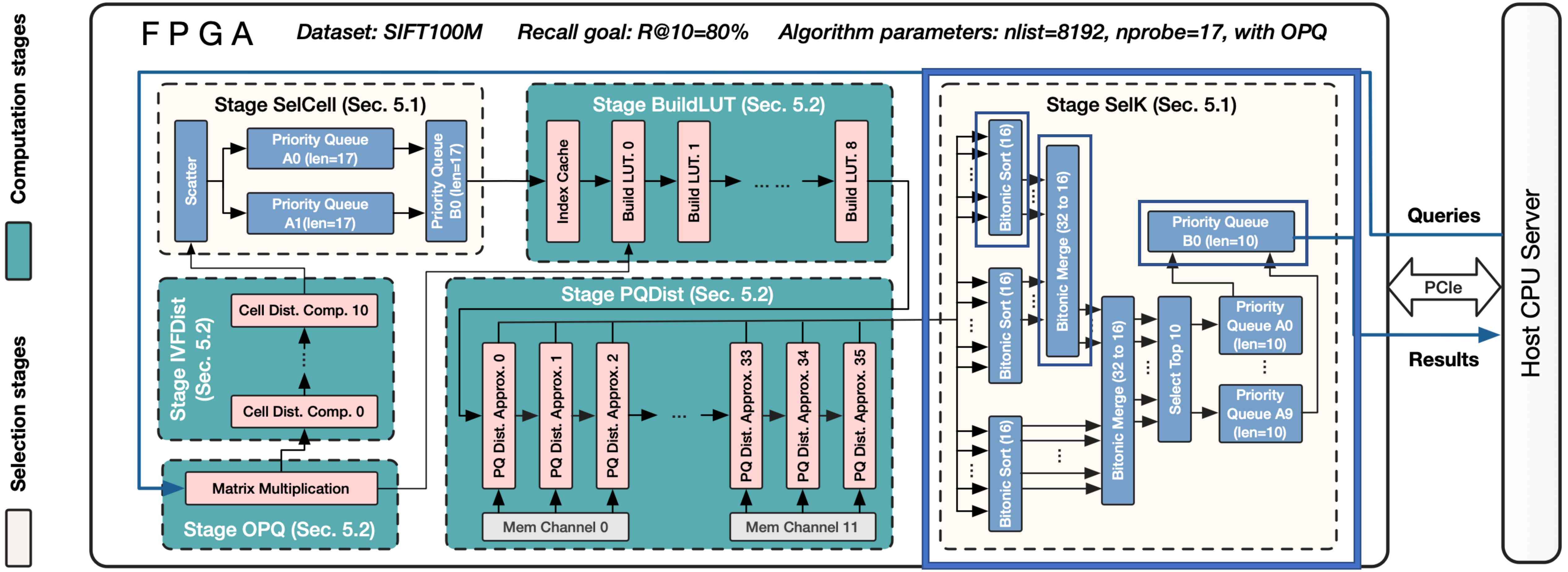
Wenqi Jiang et al. “Co-design Hardware and Algorithm for Vector Search”, the International Conference for High Performance Computing, Networking, Storage and Analysis (SC 2023).

Programming FPGAs: a more complex example (ANNS)



Six stages of different functionalities in the accelerator

Programming FPGAs: a more complex example (ANNS)



A single stage can consist of heterogeneous processing elements

What languages are used to describe the architecture?

- The traditional way: Hardware Description Languages (HDL)
- Describing the behavior of the architecture from two aspects:
    - The data path: what operations to do within a single clock cycle
    - The control flow: a finite state machine for state transfer control

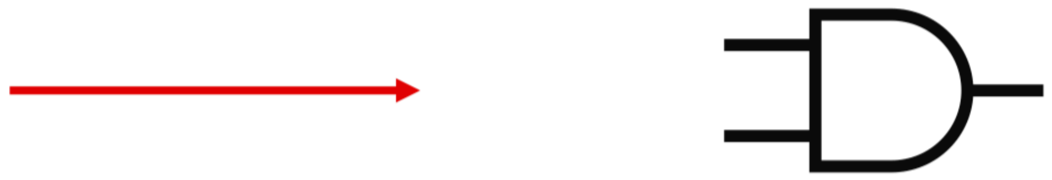
HDL (Hardware description languages): VHDL example

```
-------------------------------------
-- VHDL AND gate
-------------------------------------

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY AND_GATE IS
    PORT (
        x : IN std_logic;
        y : IN std_logic;
        f : OUT std_logic
    );
END AND_GATE;

ARCHITECTURE behaviour OF AND_GATE IS
BEGIN
    f <= x AND y;
END behaviour;
```



AND gate

Problems of HDL

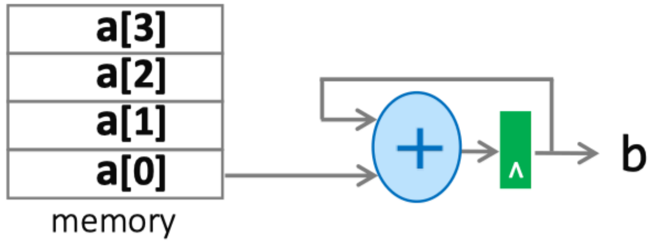
- Verbose grammars
  - Long development cycles
  - High architecture revisit cost
  - Especially problematic for research projects
- Hard to get start with for software programmers
- High-Level Synthesis (HLS) as a rescue: programming hardware with C/C++

High Level Synthesis

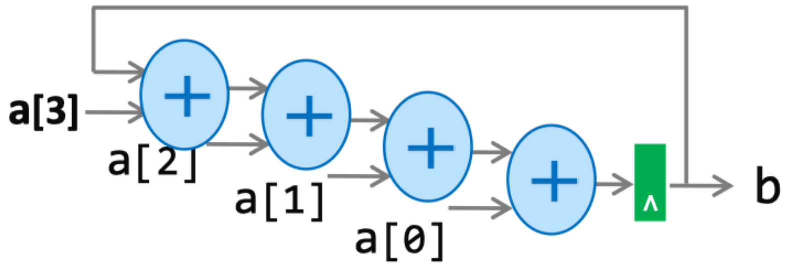
- Usually C based

```
void F (...)  {
...
for (i=0;i<=3;i++) {
     b = a[i] + b;
}
...
```

Different implementations possible



Serial execution

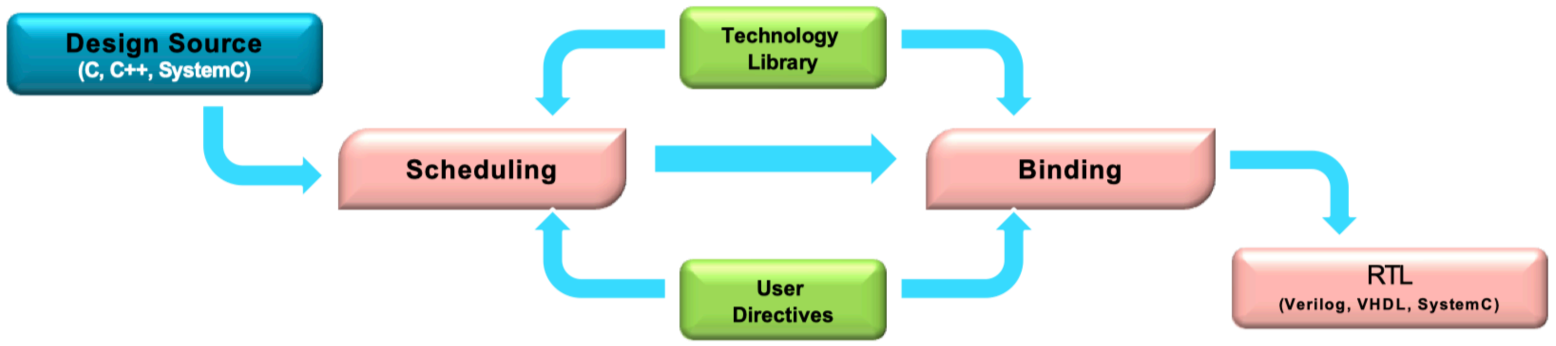


Parallel execution

More on HLS later

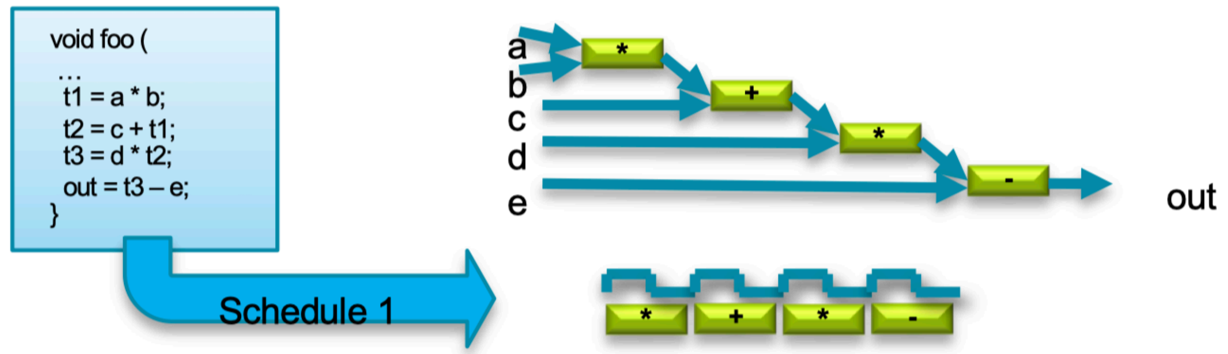
High-Level Synthesis: Scheduling & Binding

- Scheduling & Binding
  - Scheduling and Binding are at the heart of HLS
- Scheduling determines in which clock cycle an operation will occur
  - Takes into account the control, dataflow and user directives
  - The allocation of resources can be constrained
- Binding determines which library cell is used for each operation
  - Takes into account component delays, user directives



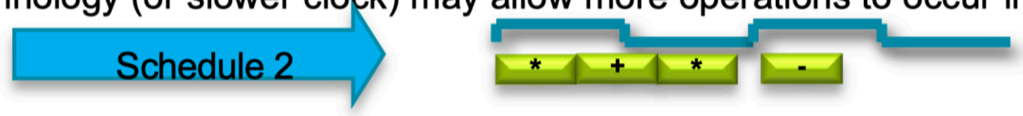
Scheduling

- The operations in the control flow graph are mapped into clock cycles



```
void foo (
 …
 t1 = a * b;
 t2 = c + t1;
 t3 = d * t2;
 out = t3 – e;
}
```

- The technology and user constraints impact the schedule
  - A faster technology (or slower clock) may allow more operations to occur in the same clock cycle

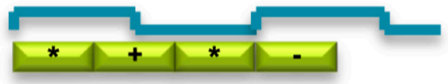


- The code also impacts the schedule
  - Code implications and data dependencies must be obeyed

Binding

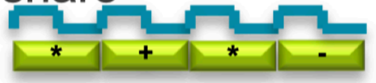
- Binding is where operations are mapped to cores from the hardware library
  - Operators map to cores

- Binding Decision: to share
  - Given this schedule:



    - Binding must use 2 multipliers, since both are in the same cycle
    - It can decide to use an adder and subtractor or *share* one addsub

- Binding Decision: or not to share
  - Given this schedule:



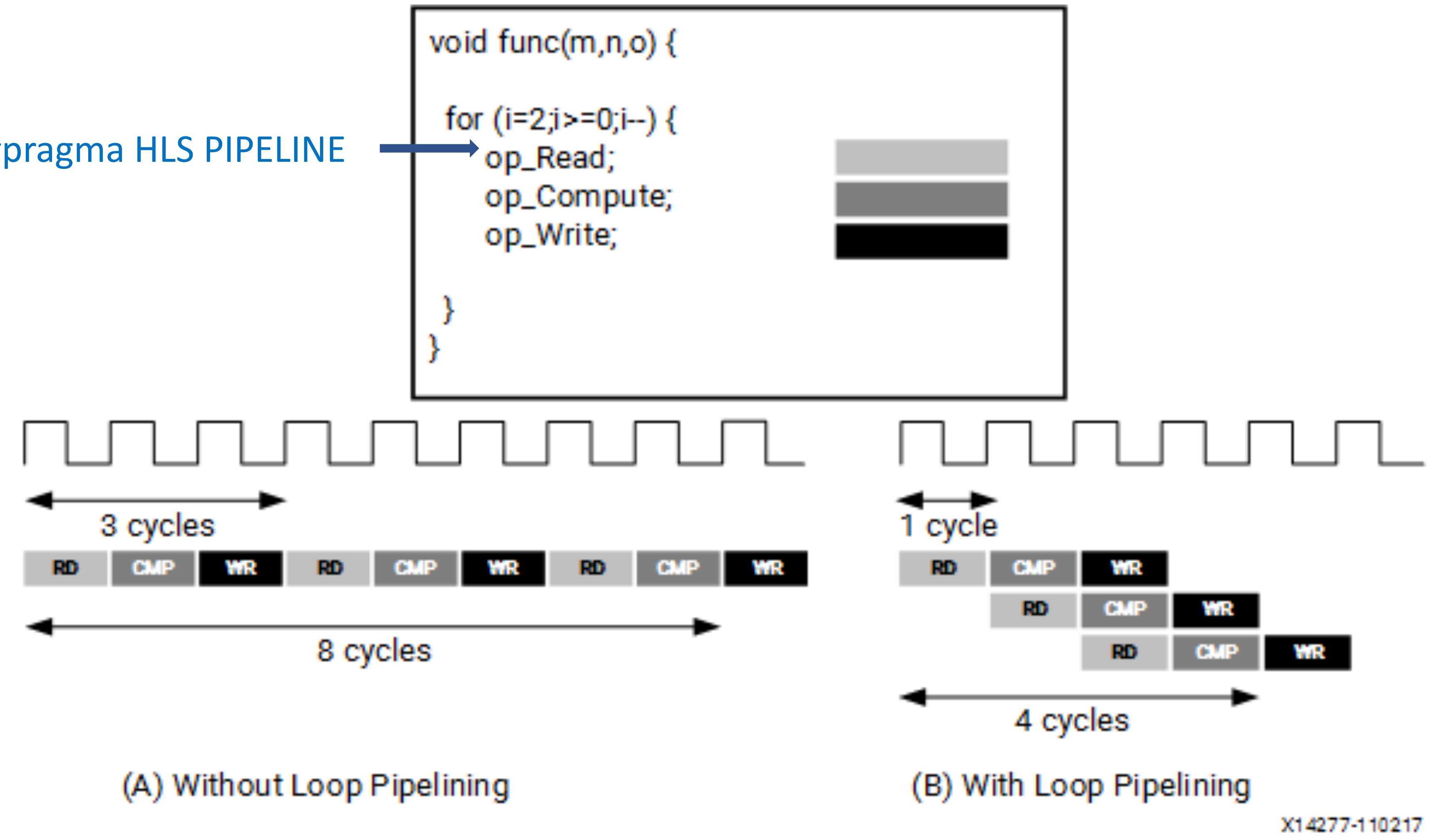
    - Binding may decide to share the multipliers (each is used in a different cycle)
    - Or it may decide the cost of sharing (muxing) would impact timing and it may decide *not to share* them
    - It may make this same decision in the first example above too

Pragmas: control the architecture behavior to an extent

- Control scheduling: what are the latency and throughput requirements within a given module?
- Control binding: which on-chip resources to use to implement the given functionalities?
- Not the HDL level of control, but give some hints to the HLS compiler

Common pragma: pipeline

#pragma HLS PIPELINE



```
void func(m,n,o) {

 for (i=2;i>=0;i--) {
   op_Read;
   op_Compute;
   op_Write;

 }
}
```

(A) Without Loop Pipelining

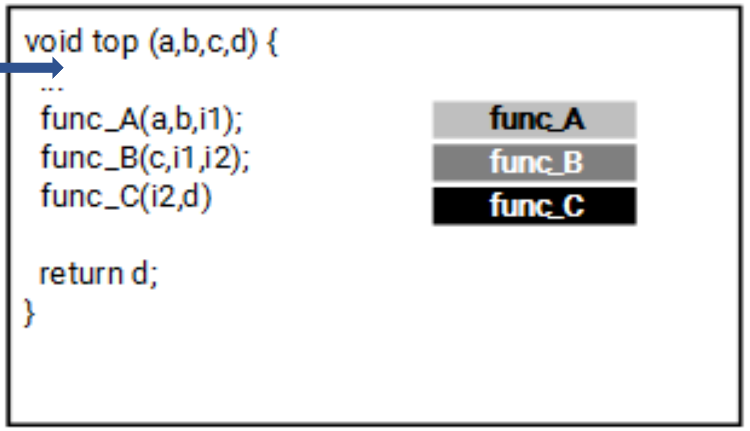
(B) With Loop Pipelining

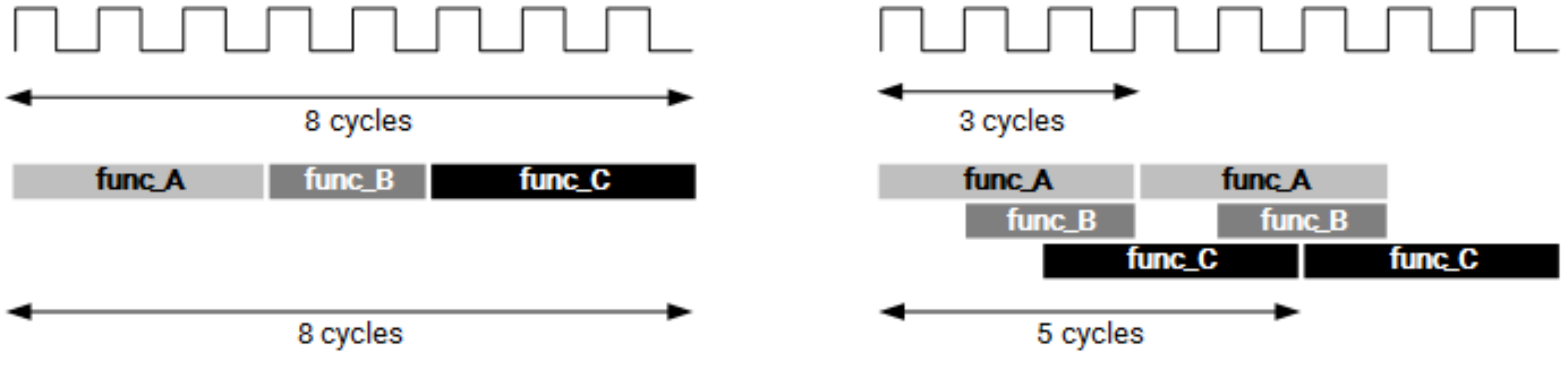
Common pragma: dataflow

#pragma HLS DATAFLOW

```
void top (a,b,c,d) {
 ...
  func_A(a,b,i1);
  func_B(c,i1,i2);
  func_C(i2,d)

  return d;
}
```





(A) Without Dataflow Pipelining

(B) With Dataflow Pipelining

X14266-110217

Common pragma: unroll

- Replicating the logic for higher performance

```
for(int i = 0; i < X; i++) {
  a[i] = b[i] + c[i];
}
```

Common pragma: unroll

- Replicating the logic for higher performance

```
for(int i = 0; i < X; i++) {
  pragma HLS unroll factor=2
  a[i] = b[i] + c[i];
}
```

What are HLS good and not good at?

- Pros
  - Fast prototyping – crucial for research
  - Easy to get start for a software programmers
- Cons
  - Not suitable for infrastructure development
    - e.g., memory controllers, network stack, etc.
  - No full control of the generated architecture
    - Can only indirectly control scheduling and binding using pragmas

More references for HLS programming

- Vitis HLS Programming Guide
  - https://github.com/Xilinx/Vitis-Tutorials
  - https://docs.xilinx.com/r/en-US/ug1399-vitis-hls
- Tutorial@SC: Productive Parallel Programming for FPGA with HLS
  - http://spcl.inf.ethz.ch/Teaching/hls-tutorial
  - By Johannes de Fine Licht and Torsten Hoefler @ETH Zurich
  - 3-hour hands-on tutorial with programming examples

Our tutorial slides are available at: https://systems.ethz.ch/research/data-processing-on-modern-hardware/hacc/sigmod-23-tutorial--data-processing-on-fpgas-with-modern-archite.html