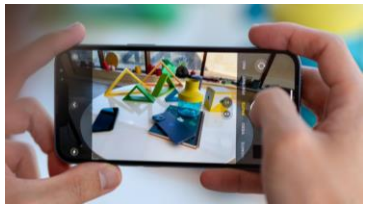
Accelerating Edge AI with Morpher: An Integrated Design, Compilation and Simulation Framework for CGRAs

Dhananjaya Wijerathne, Zhaoying Li , Tulika Mitra
School of Computing, National University of Singapore
{dmd, zhaoying, tulika}@comp.nus.edu.sg

Domain Specific vs Reconfigurable Accelerators

Tiny IoT devices have ultra-low area and power footprint need

Several domain-specific accelerators plus multiple general-purpose cores on SoC are not tenable for such devices



Video

ML

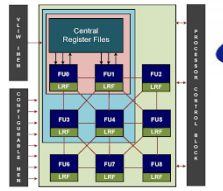
Audio

Domain Specific Accelerators

Apple iPhone 12's A14 Bionic SoC with Domain Specific Accelerators

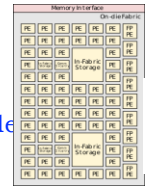
Reconfigurable Accelerators

Field Programmable Gate Arrays (FPGA)



SAMSUNG

Samsung Reconfigurable Processor



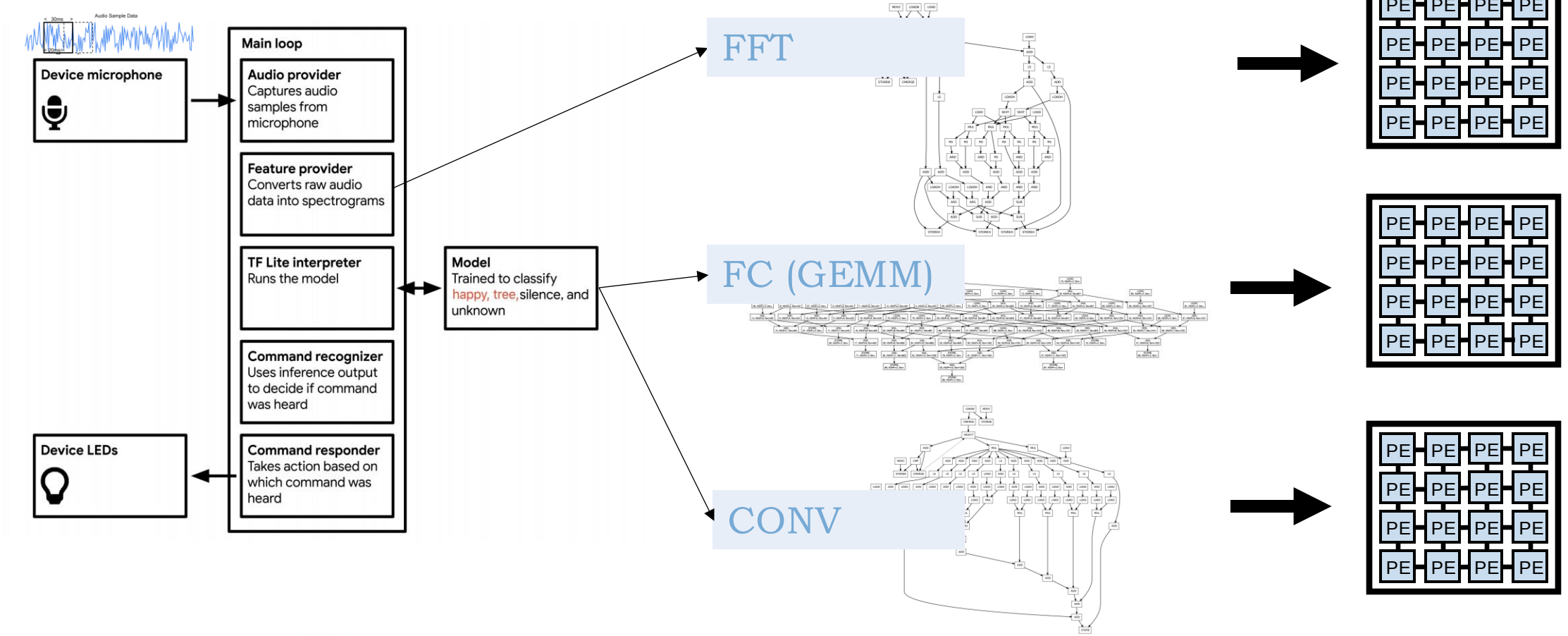
intel

Configurable Spatial Accelerator

Coarse-Grained Reconfigurable Array (CGRA)

NUS National University of Singapore

CGRA: Supporting Diverse Dataflows



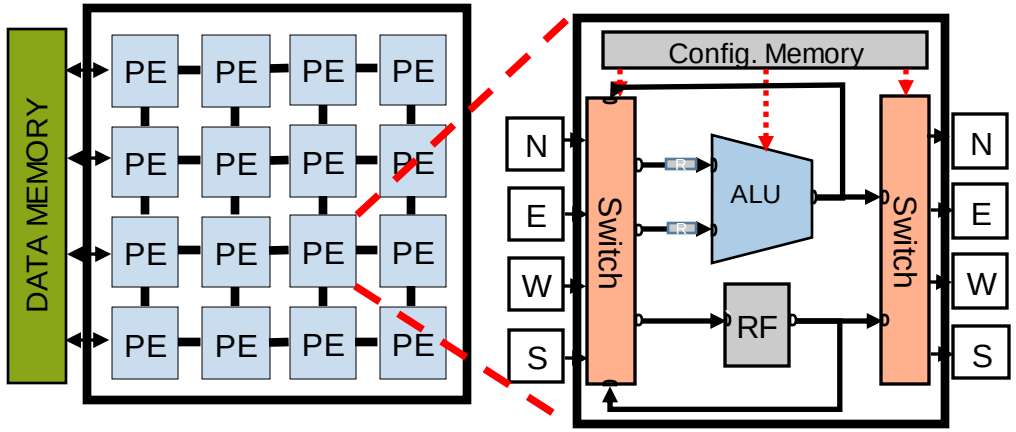
Speech Recognition Model

Application Kernels

Different Dataflows

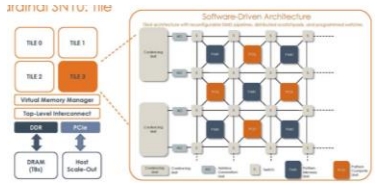
Coarse-Grained Reconfigurable Array (CGRA)

- **CGRA Architecture**: A power-efficient mesh of processing elements, each equipped with an ALU, register file, configuration memory, switches, and on-chip memory.
- **Execution Model**: CGRAs' flexibility lies in the compiler-generated execution schedules and configurations, adapting to different application kernels mapped onto the CGRAs.

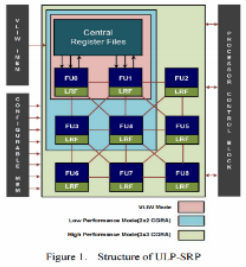


CGRA Architecture

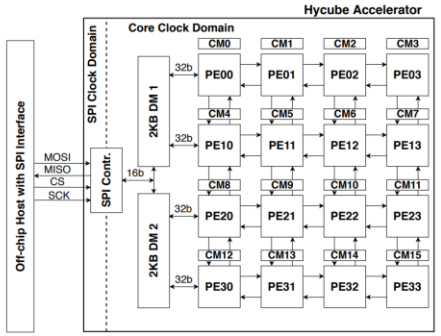
Coarse-Grained Reconfigurable Array (CGRA)

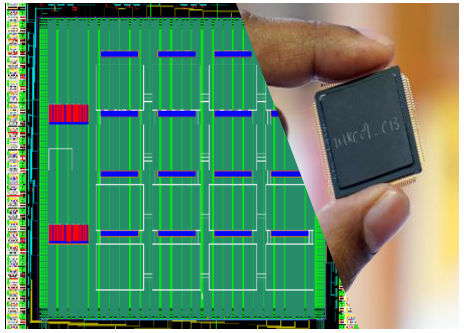


Sambanova Reconfigurable Dataflow Unit (RDU)

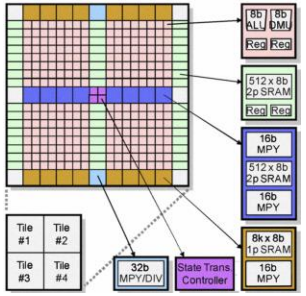


Samsung Reconfigurable Processor (SRP)

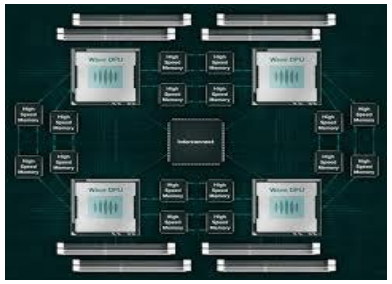




NUS HyCUBE

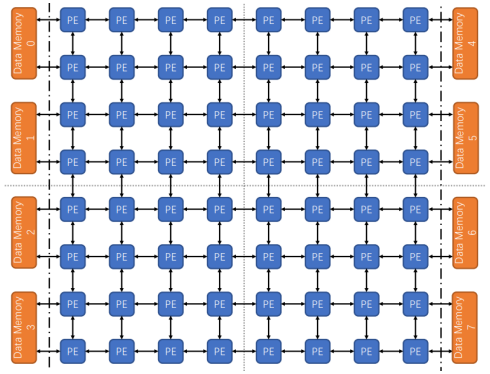


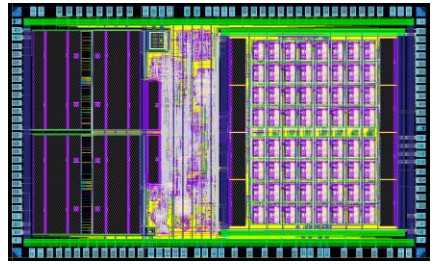
Renesas Dynamic Reconfigurable Processor (DRP)



Wave DPU

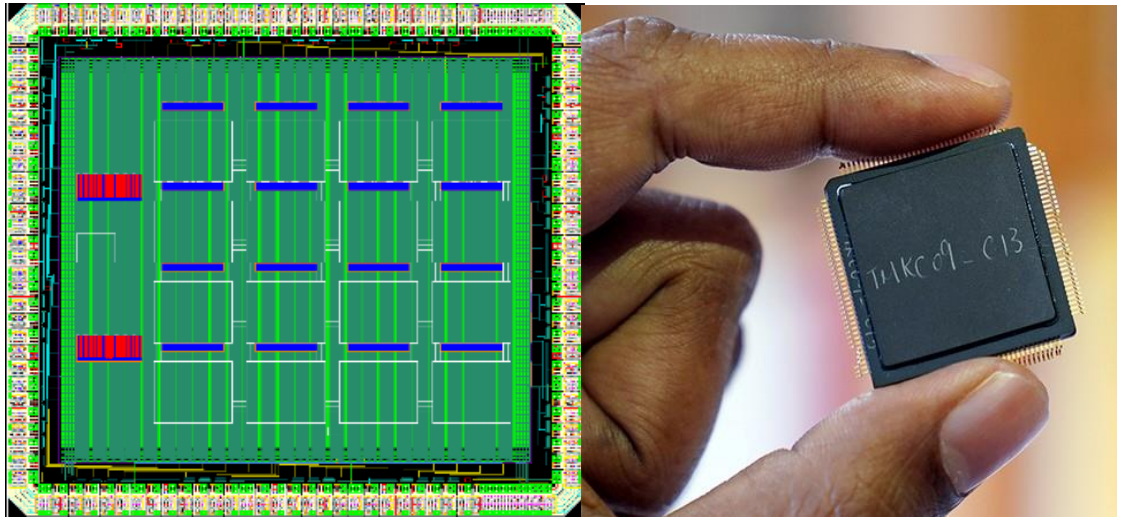
Commercial CGRAs

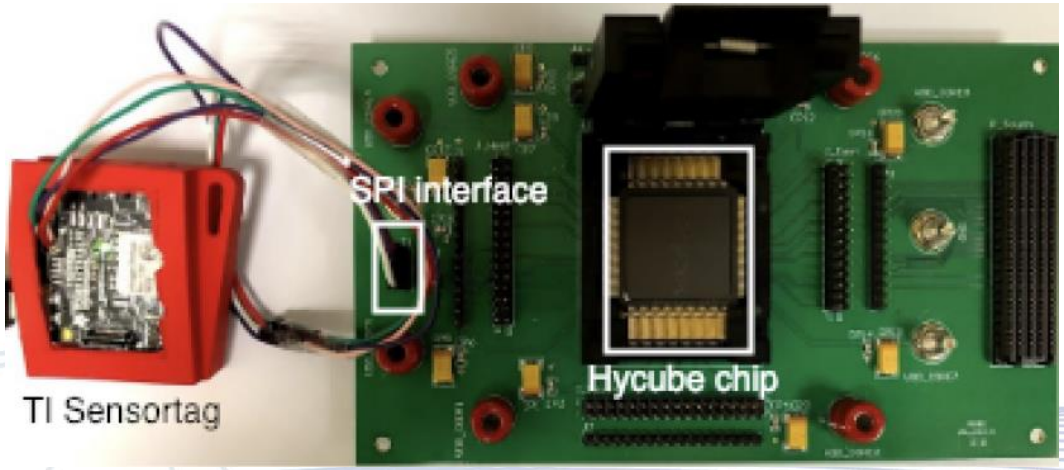




NUS PACE

HyCUBE CGRA @ NUS 2019





4x4 CGRA, TSMC 40nm

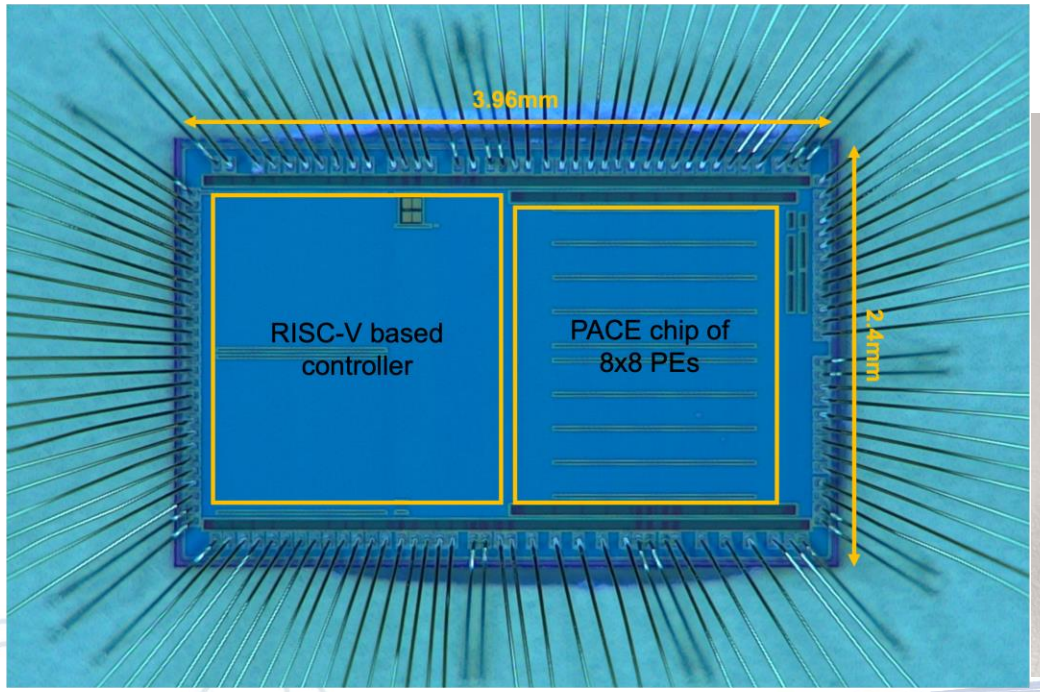
NUS HyCUBE: 90 MOPS/mW
Samsung SRP: 22 MOPS/mW
ARM CPU: 2.6 MOPS/mW
Xilinx FPGA: 25 MOPS/mW

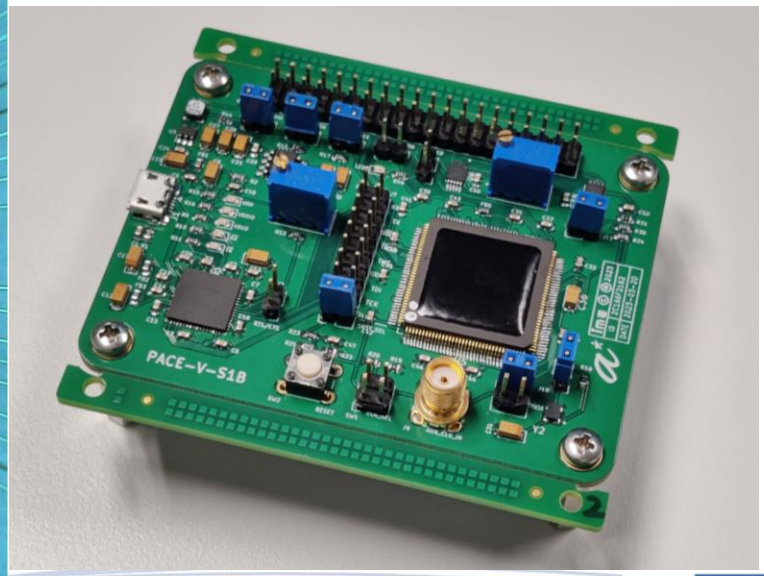
NUS National University of Singapore

PACE CGRA+RISC-V SoC: NUS + IME 2023

Institute of Microelectronics
A*STAR

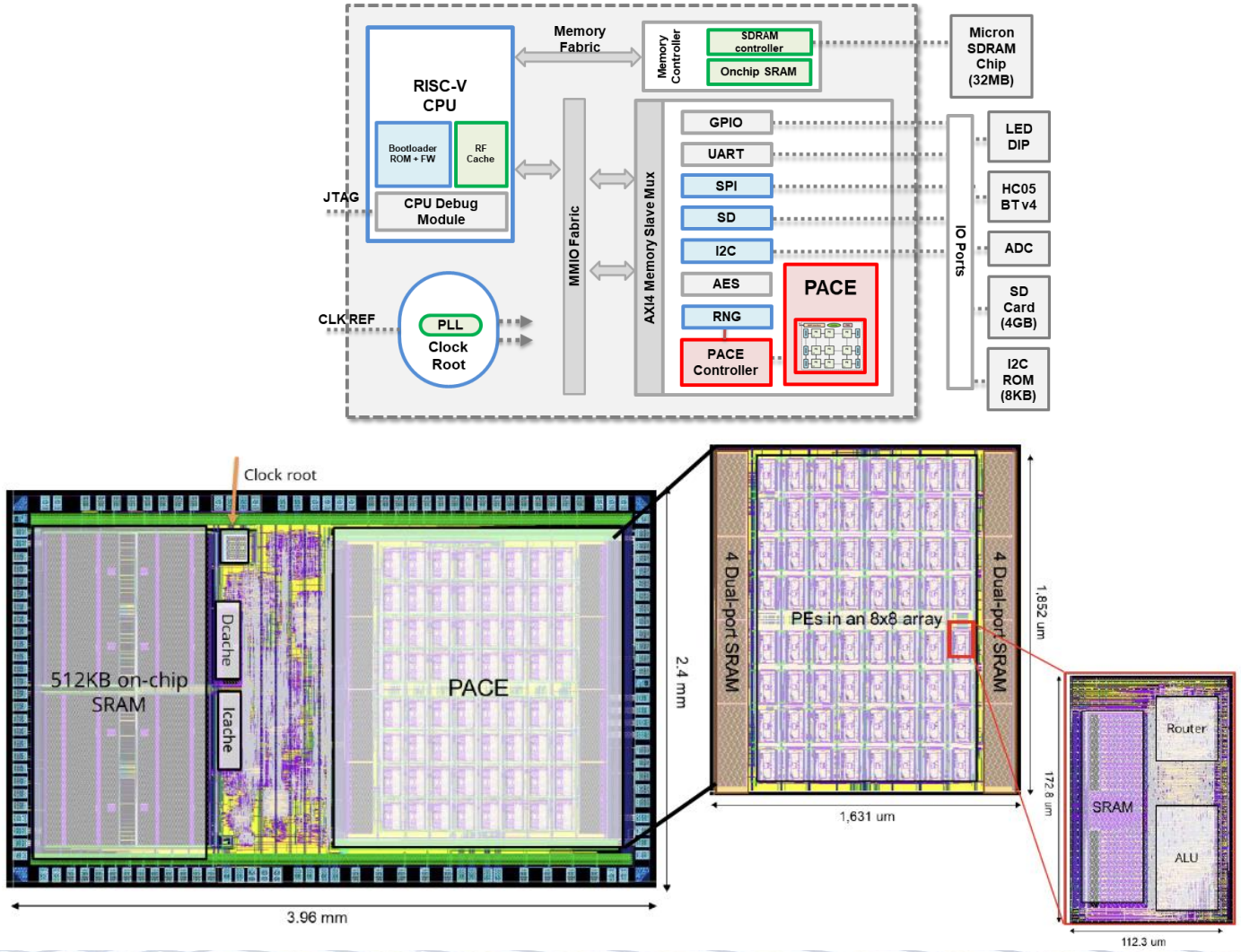
8x8 CGRA:
582 GOPS/W at 0.45V, 40nm ULP
1.1 mW at 10MHz
Estimated 1 TOPS/W at 22nm





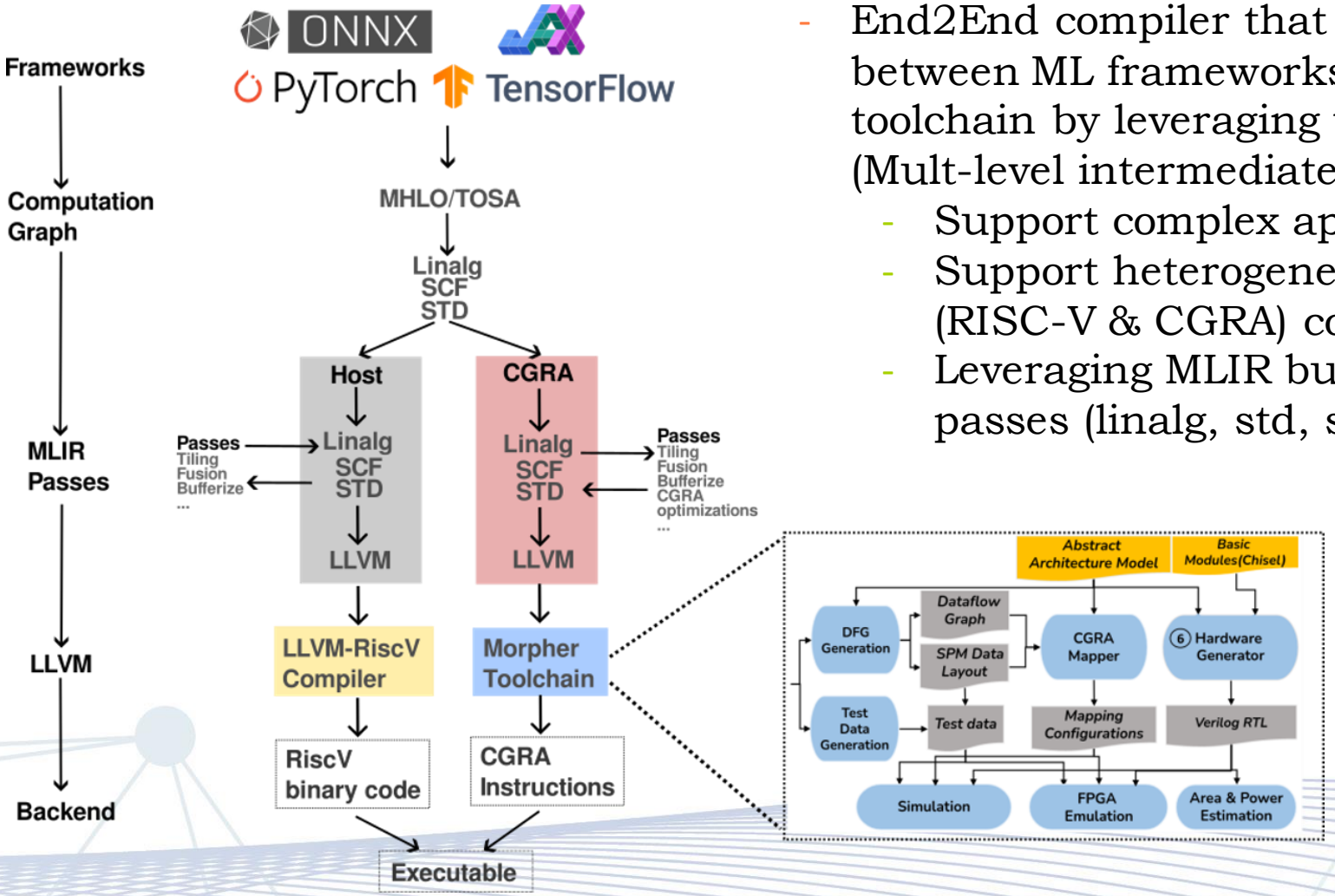
NUS National University of Singapore

PACE CGRA SoC: NUS + IME



PACE SoC Specifications (Simulation)		
Tech. Node		UMC 40nm ULP
Area		3.96mm x 2.4mm (Die) 3.80mm x 2.0mm (Core)
Voltage		1V/3.3V
RISC-V based controller	**Memory**	512 KB on-chip memory 16KB ICache 16KB DCache
	Frequency	200/100 MHz
	Power	14mW @ 1.0V
PACE CGRA	**#PEs**	8x8 (17 instructions, including NOP)
	Frequency	10 MHz
	Memory	8 x 8KB data memory 64 x 0.25KB config memory
	Power	1.1 mW @ 0.45V
	Efficiency	582 TOPS/W

NUS MLIR-Based End-to-End Compiler

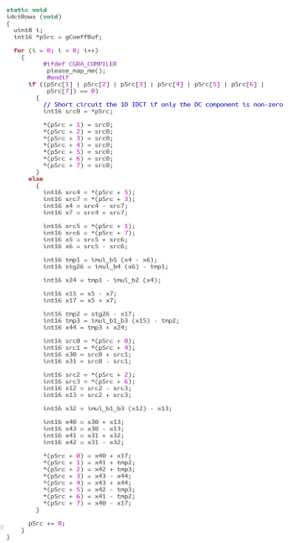


- End2End compiler that bridges the gap between ML frameworks and the Morpher toolchain by leveraging the power of MLIR (Mult-level intermediate representation).
  - Support complex applications
  - Support heterogeneous system (RISC-V & CGRA) compilation
  - Leveraging MLIR builtin dialects and passes (linalg, std, scf…)

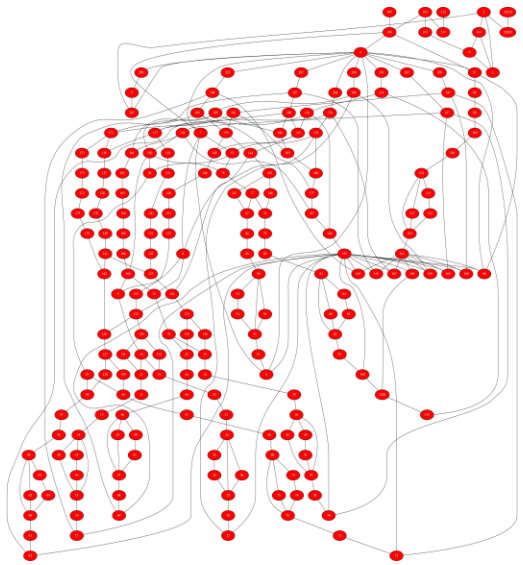
NUS National University of Singapore

Application mapping on CGRA

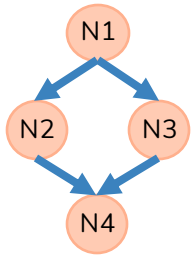
- Target: a loop kernel from applications
- Mapping the dataflow graph (DFG) of the loop body on to the CGRA
  - Placement: assigning DFG operations to ALUs
  - Routing: mapping data signals using wires and registers



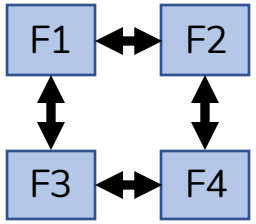
Loop kernel c code



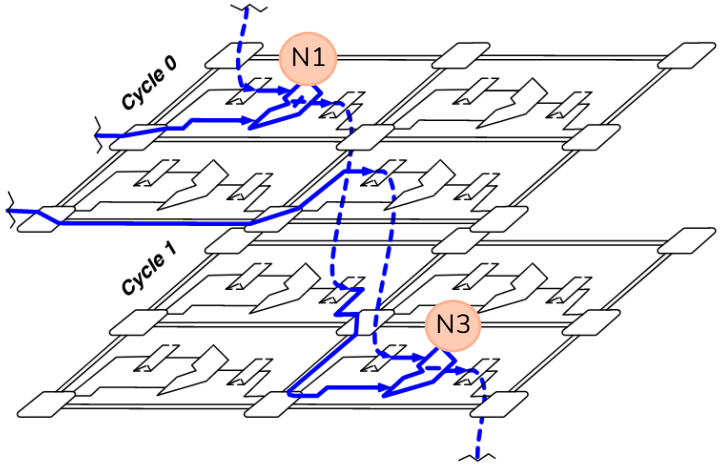
Dataflow graph (DFG) of the loop body



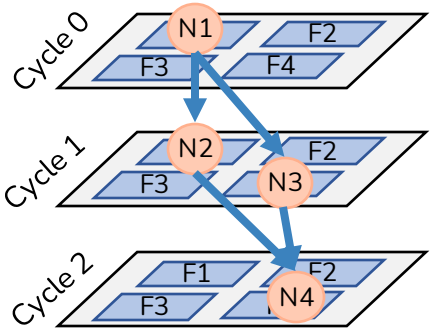
DFG



2x2 CGRA



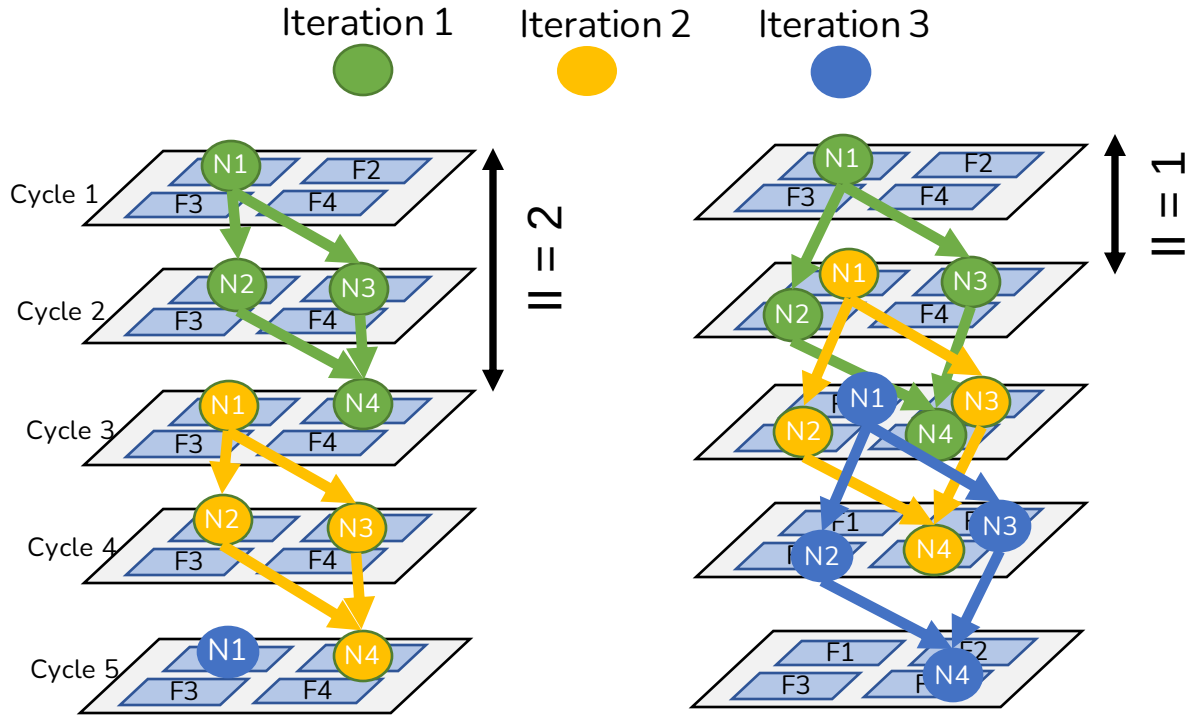
Placement and routing



Spatio-temporal mapping

Application mapping on CGRA

- Software pipelined schedule
- Goal: Mapping with minimum initiation interval
- Initiation interval (II) = cycle difference between initiation of consecutive iterations
- Low II -> High performance



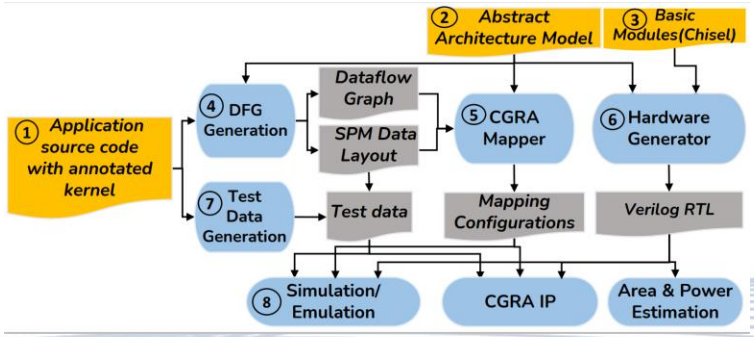
Low performance mapping with II = 2 cycles

High performance mapping with II = 1 cycles

Morpher: An Integrated Compilation and Simulation Framework for CGRA

- Fully automated end-to-end CGRA compilation, architecture generation and simulation framework
  - Flexible architecture specification language
  - Efficient mapping algorithms
  - Automatic RTL generation & cycle-accurate simulation to validate the compilation results
  - Fully open-source with easily modifiable modular code base

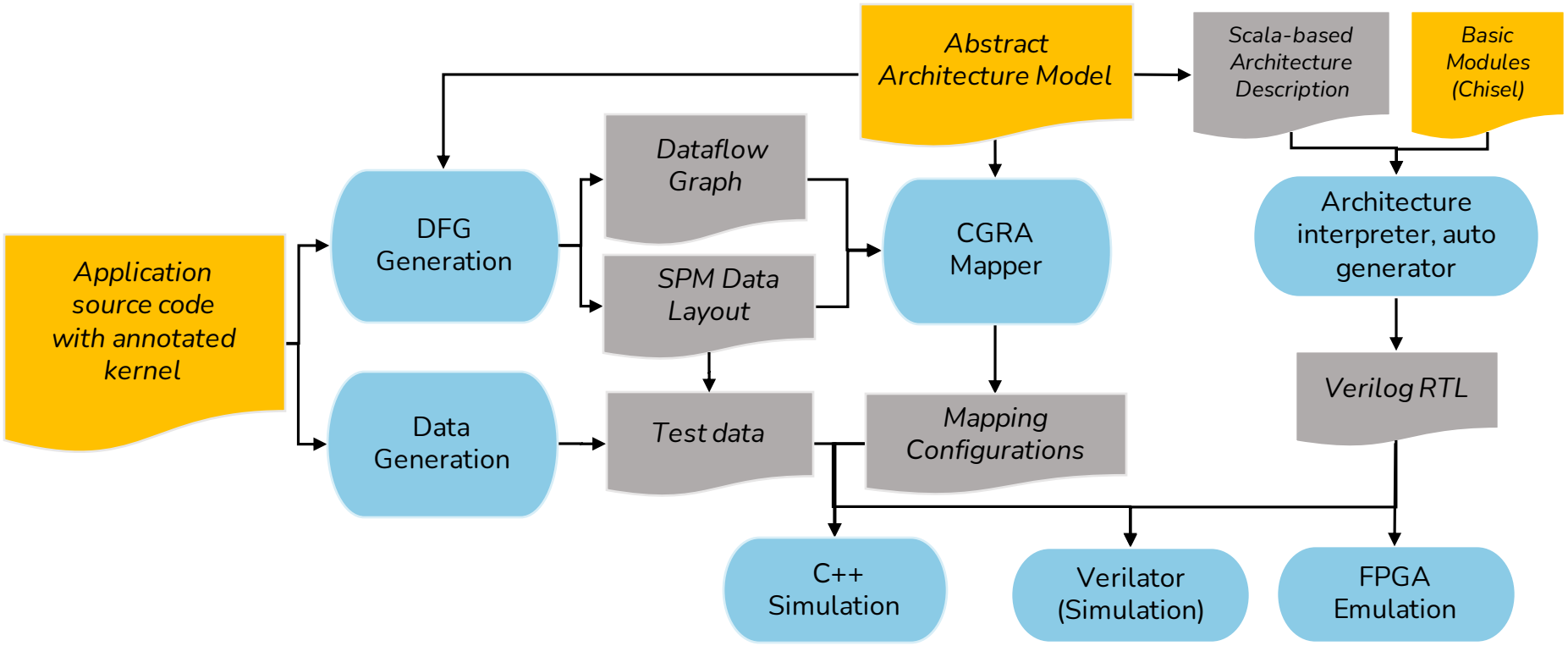
Features		CGRA-ME	Pillars	Open-CGRA	CCF	Morpher
DFG Generation	Models control divergence	✗	✗	✓	✓	✓
	Recurrence edges	✗	✗	✓	✓	✓
Architecture Modelling	Adapt user defined architectures	✓	✓	✓	✗	✓
	Multi-hop connections	✗	✗	✗	✗	✓
	Different memory organizations	✗	✗	✓	✗	✓
P&R Mapper	Architecture adaptive mapping	✓	✓	✗	✗	✓
	Data layout aware mapping	✗	✗	✗	✗	✓
	Recurrence aware mapping	✗	✗	✓	✓	✓
Simulation & validation	Cycle accurate simulation	✗	✓	✓	✓	✓
	Test data generation	✗	✗	✗	✗	✓
	Validation against test data	✗	✗	✗	✗	✓
Hardware Generation	Generate RTL	✓	✓	✓	✗	✓
	Infer control paths	✗	✓	✓	✗	✓
	Infer multiplexers	✗	✗	✗	✗	✓



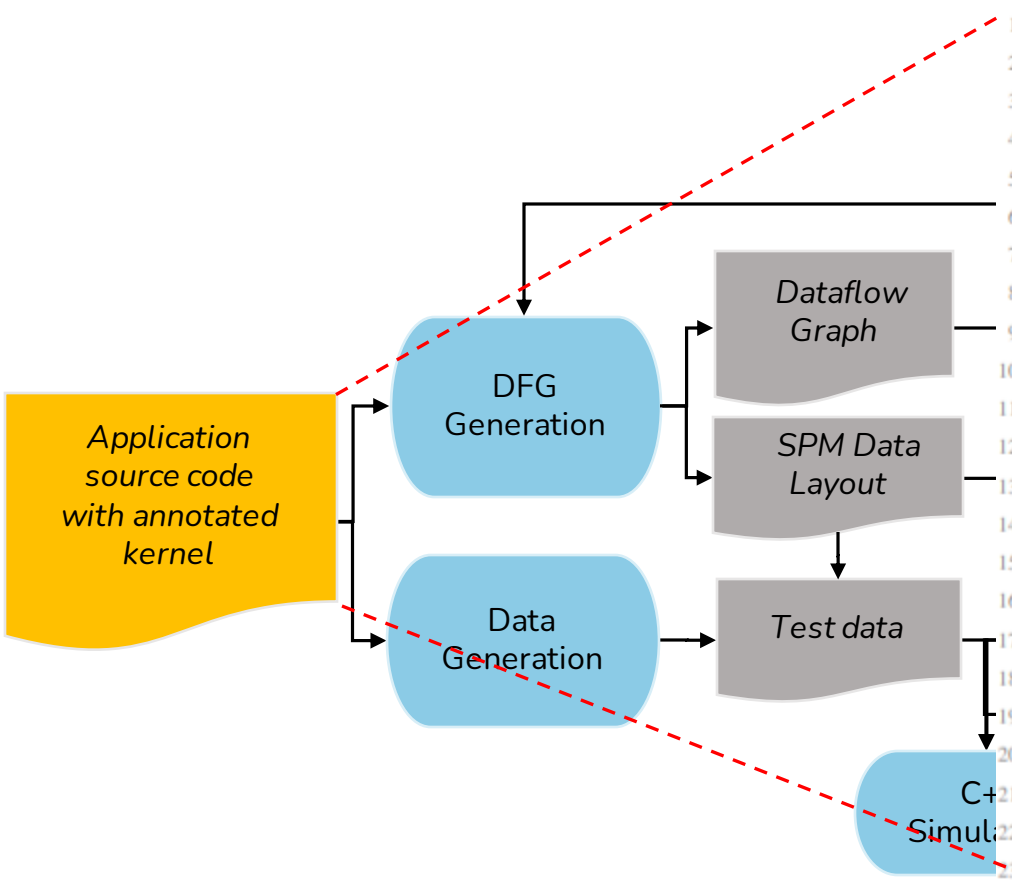
SCAN ME

NUS National University of Singapore

Overview


Abstract Architecture Model
Scala-based Architecture Description
Basic Modules (Chisel)
Application source code with annotated kernel
DFG Generation
Dataflow Graph
SPM Data Layout
CGRA Mapper
Architecture interpreter, auto generator
Data Generation
Test data
Mapping Configurations
Verilog RTL
C++ Simulation
Verilator (Simulation)
FPGA Emulation

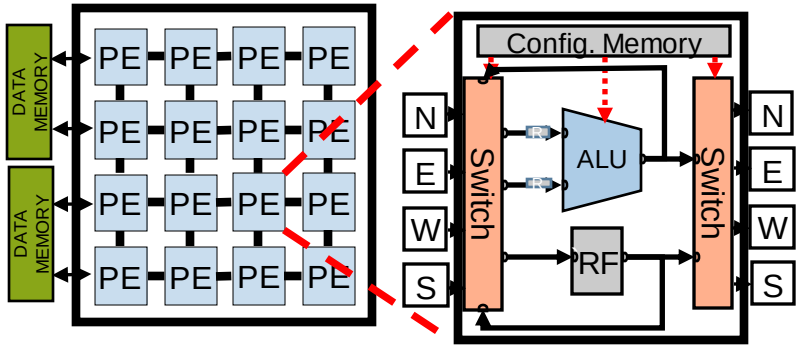
Overview




```
#define SIZE  20
int A[SIZE], B[SIZE], C[SIZE];

__attribute__((noinline))
void array_add(){
   for (int i=0;i<SIZE; i++){
      #ifdef CGRA_COMPILER
      please_map_me();
      #endif
      C[i] = A[i]+B[i];
   }
}

int main(){
    for (int i=0;i<SIZE; i++){
      A[i] = i * 2 + 5;
      B[i] = i * 3;
      C[i] = 0;
    }
    array_add();
    for (int i=0;i<SIZE; i++) printf("%d\n", C[i]);
    return 0;
}
```

Target loop

Overview



Abstract Architecture Model

Scala-based Architecture Description

Basic Modules (Chisel)

Dataflow Graph

DFG Generation

CGRA Mapper

Architecture interpreter, auto generator

Application source code with annotated kernel

Verilog RTL

FPGA Emulation

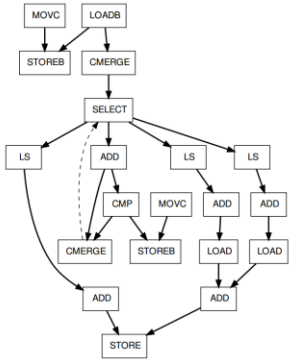
```
#define SIZE  20
int A[SIZE], B[SIZE],

__attribute__((noinli
void array_add(){
   for (int i=0;i<SIZE
      #ifdef CGRA_COMPILER
      please_map_me();
      #endif
      C[i] = A[i]+B[i];
   }
}

int main(){
    for (int i=0;i<SIZE; i++){
      A[i] = i * 2 + 5;
      B[i] = i * 3;
      C[i] = 0;
    }
    array_add();
    for (int i=0;i<SIZE; i++) printf("%d\n", C[i]);
    return 0;
}
```

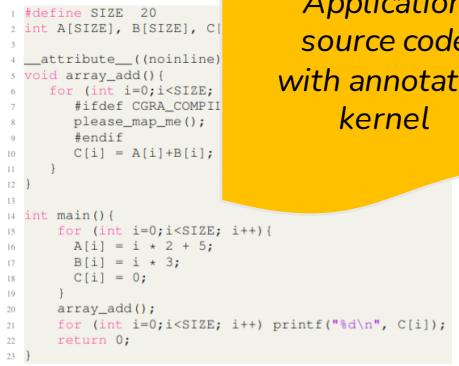
Morpher Architecture Description Language (ADL)

```
{"ARCH" :
  {
    "DP" : {..},

    "MDP" : {..},

    "SPM" : {..},

    "FU" : {..},

    "FU_MEM" : {..},

    "PE_MEM" : {..},

    "PE":{
        "INPUTS" : ["NORTH_I","WEST_I","EAST_I","SOUTH_I"],
        "OUTPUTS" : ["NORTH_O","WEST_O","EAST_O","SOUTH_O"],
        "INTERNALS" : [ "NORTH_XBARI", "EAST_XBARI", "WEST_XBARI", "SOUTH_XBARI"],
        "SUBMODS" : {"FU":[{"name":"FU0"}] },
        "REGS" : ["NR","ER","WR","SR","TREG"],
        "CONNECTIONS" : {
          "THIS.NORTH_I" : ["THIS.NR", "THIS.NORTH_XBARI"],
          ..
        }},
    "CGRA" :{..}
  }
}
```

NUS National University of Singapore

Overview





Application source code with annotated kernel

DFG Generation

Data Generation

Dataflow Graph

SPM Data Layout

Test data

Abstract Architecture Model

CGRA Mapper

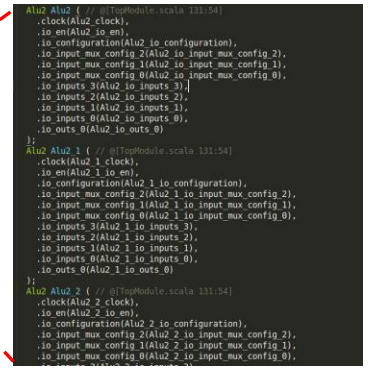
Mapping Configurations

Scala-based Architecture Description

Basic Modules (Chisel)

Architecture interpreter, auto generator

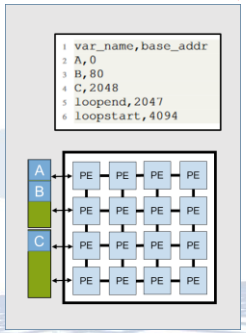
Verilog RTL

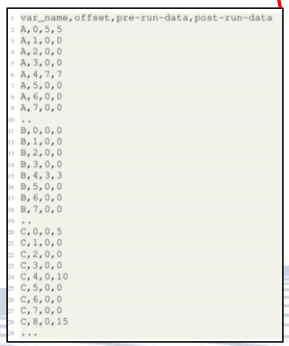


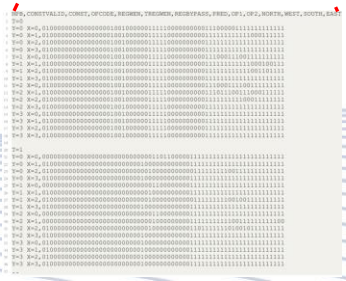
C++ Simulation

Verilator (Simulation)

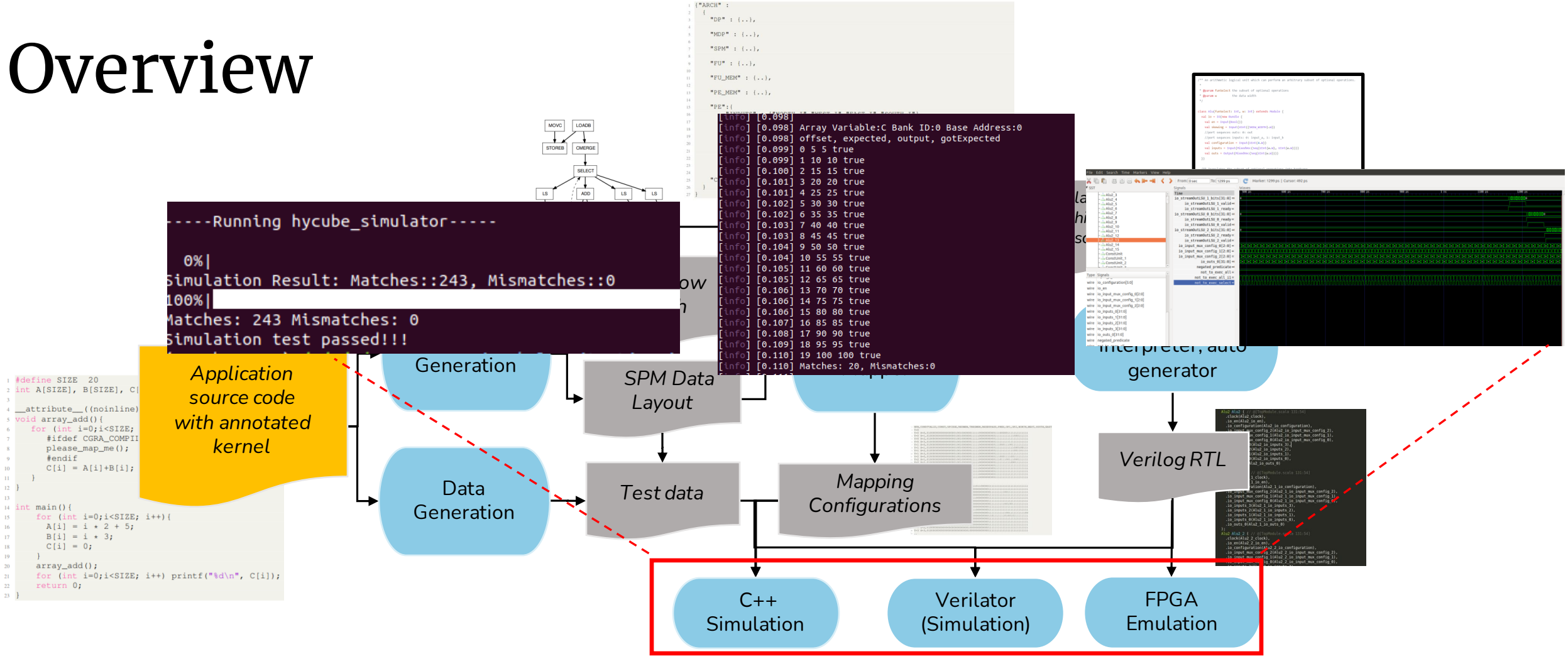
FPGA Emulation





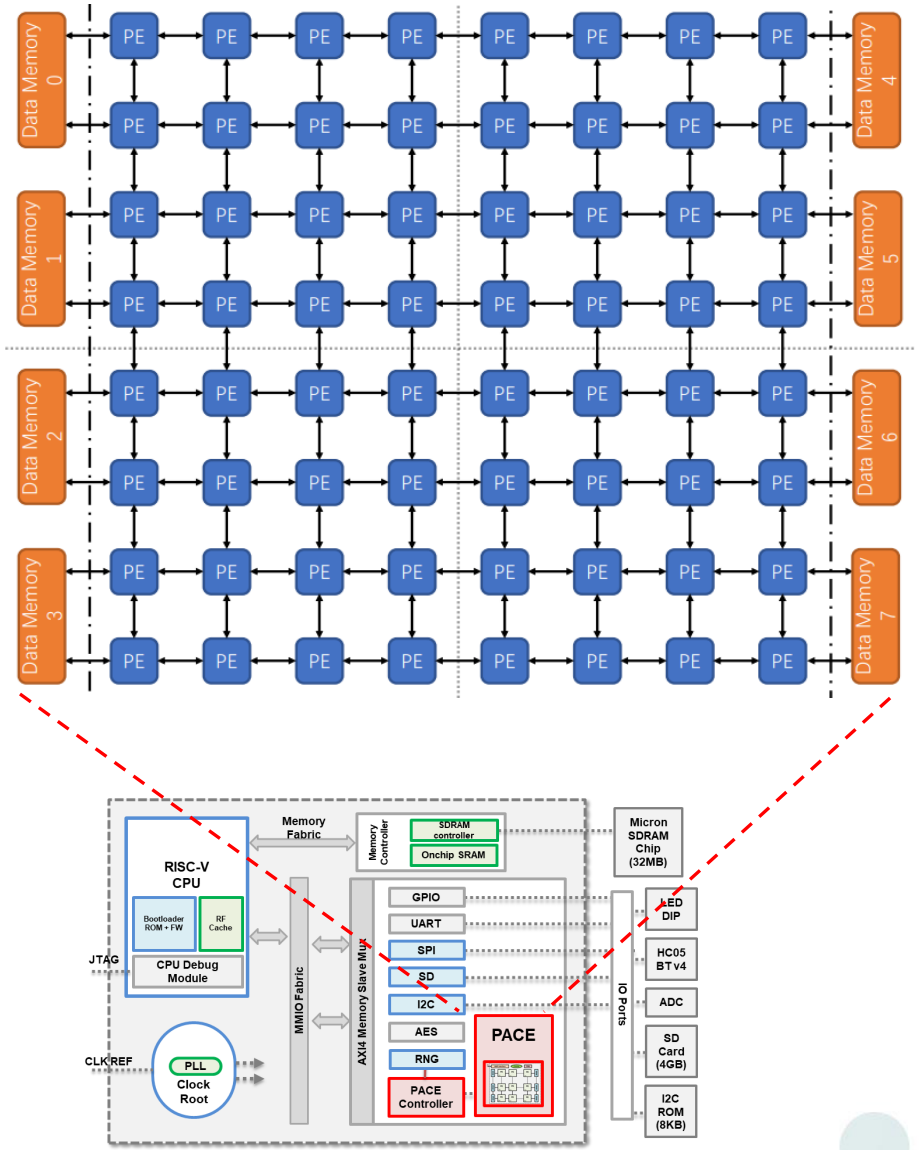


Overview

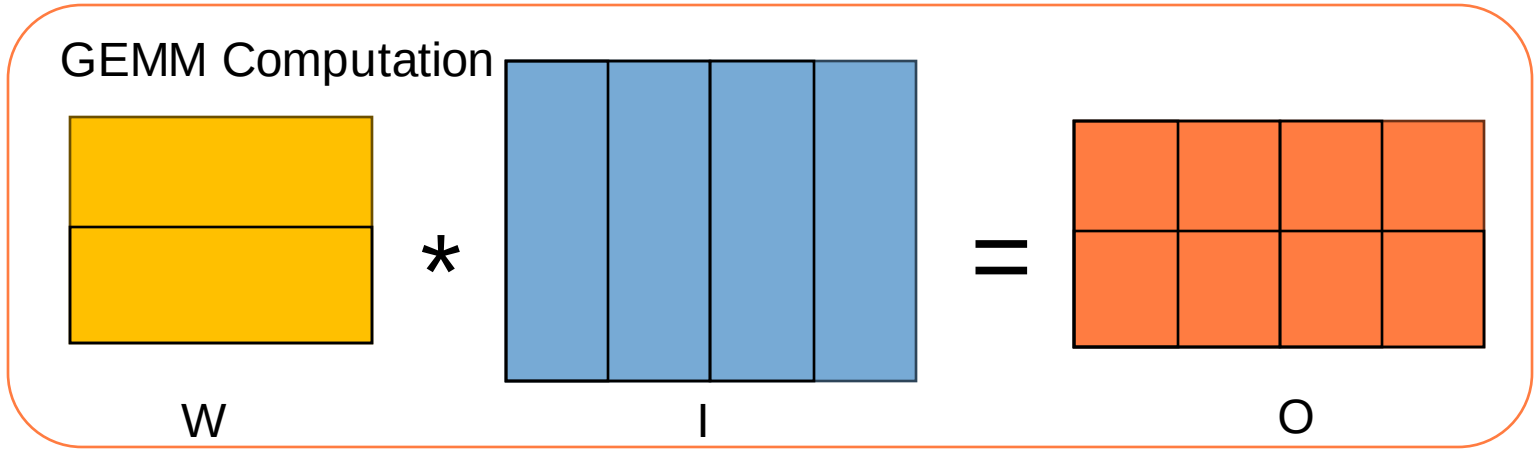

-----Running hycube_simulator-----
0%|
Simulation Result: Matches::243, Mismatches::0
100%|
Matches: 243 Mismatches: 0
Simulation test passed!!!
[info] [0.098] Array Variable:C Bank ID:0 Base Address:0
[info] [0.098] offset, expected, output, gotExpected
[info] [0.099] 0 5 5 true
[info] [0.099] 1 10 10 true
[info] [0.100] 2 15 15 true
[info] [0.101] 3 20 20 true
[info] [0.101] 4 25 25 true
[info] [0.102] 5 30 30 true
[info] [0.102] 6 35 35 true
[info] [0.103] 7 40 40 true
[info] [0.103] 8 45 45 true
[info] [0.104] 9 50 50 true
[info] [0.104] 10 55 55 true
[info] [0.105] 11 60 60 true
[info] [0.105] 12 65 65 true
[info] [0.106] 13 70 70 true
[info] [0.106] 14 75 75 true
[info] [0.106] 15 80 80 true
[info] [0.107] 16 85 85 true
[info] [0.108] 17 90 90 true
[info] [0.109] 18 95 95 true
[info] [0.110] 19 100 100 true
[info] [0.110] Matches: 20, Mismatches:0
MOVC
LOADB
STOREB
CMERGE
SELECT
LS
ADD
LS
LS
Application source code with annotated kernel
Generation
SPM Data Layout
Data Generation
Test data
Mapping Configurations
generator
Verilog RTL
C++ Simulation
Verilator (Simulation)
FPGA Emulation

Experimental Study

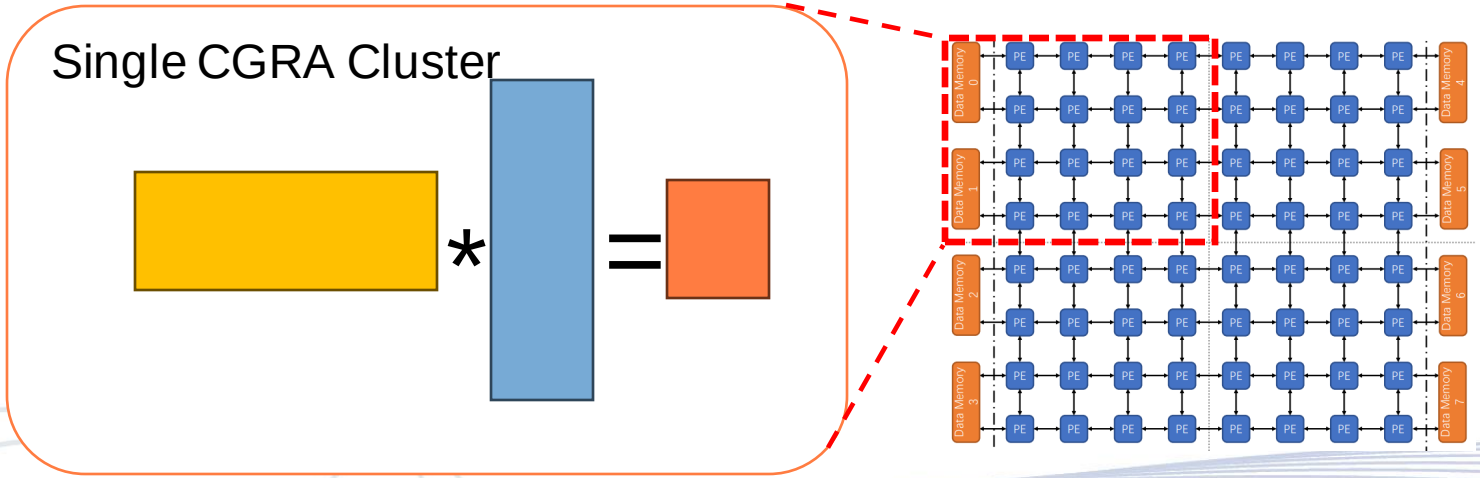
- Target CGRA Design: 8x8 PE array with 8 data memories on boundary PEs
  - Logically divided into four clusters:
  - Each with a 4x4 PE array and two 8kB memory banks
- Accelerating ML Workloads:
  - Focus on GEMM and CONV Kernels
  - Kernel Dimensions & Tile Sizes:
    - GEMM: Dimension of 64 x 64 x 64; Tile Size: 64 × 16 × 64
    - CONV: Dimension of 64 x 64 x 64 × 3 x 3; Tile Size: 64 x 64 × 1 × 3 x 3

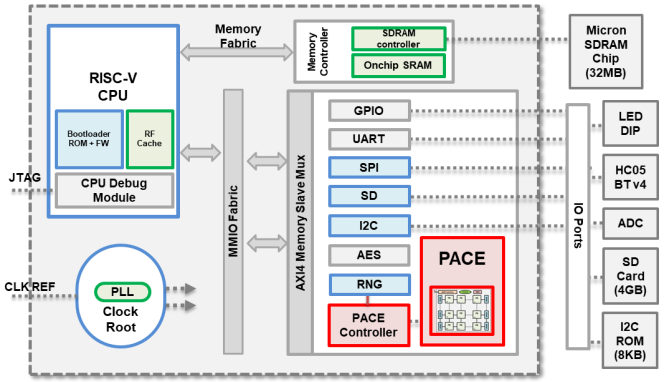


GEMM Mapping

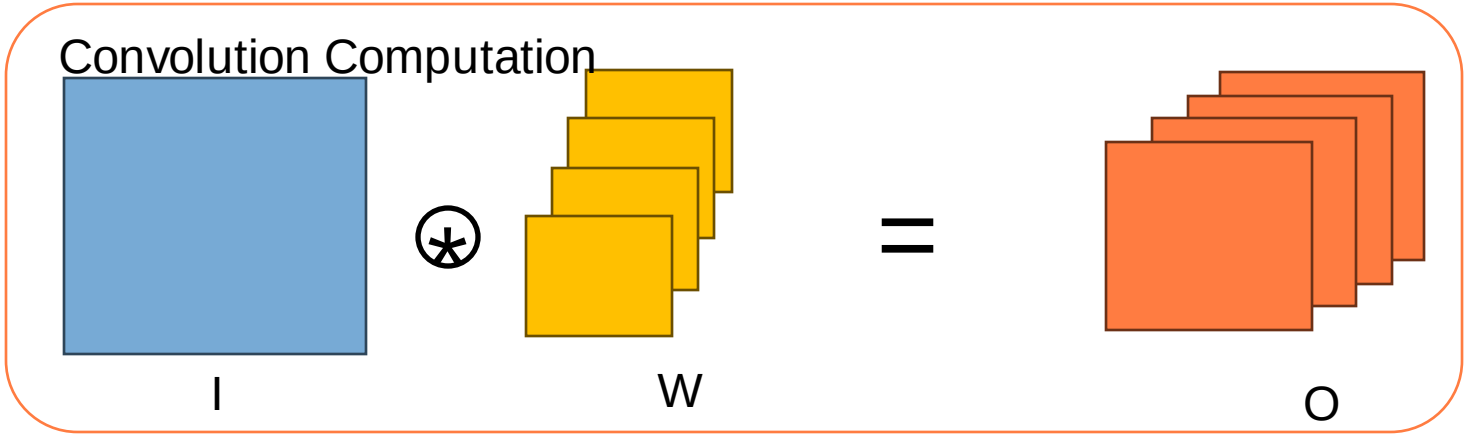



```
// Sequential loop: from off-chip to on-chip
for m in range(M/(TI*X)):
for n in range(N/(TJ*Y)):
for k in range(K/(TK)):
// Parallel loop: CGRA clusters
  for x in range(X):
  for y in range(Y):
// Single CGRA level
    for i in range(TI):
    for j in range(TJ):
    for k in range(TK)://map this
      O[][] += W[][]* I[][];
```

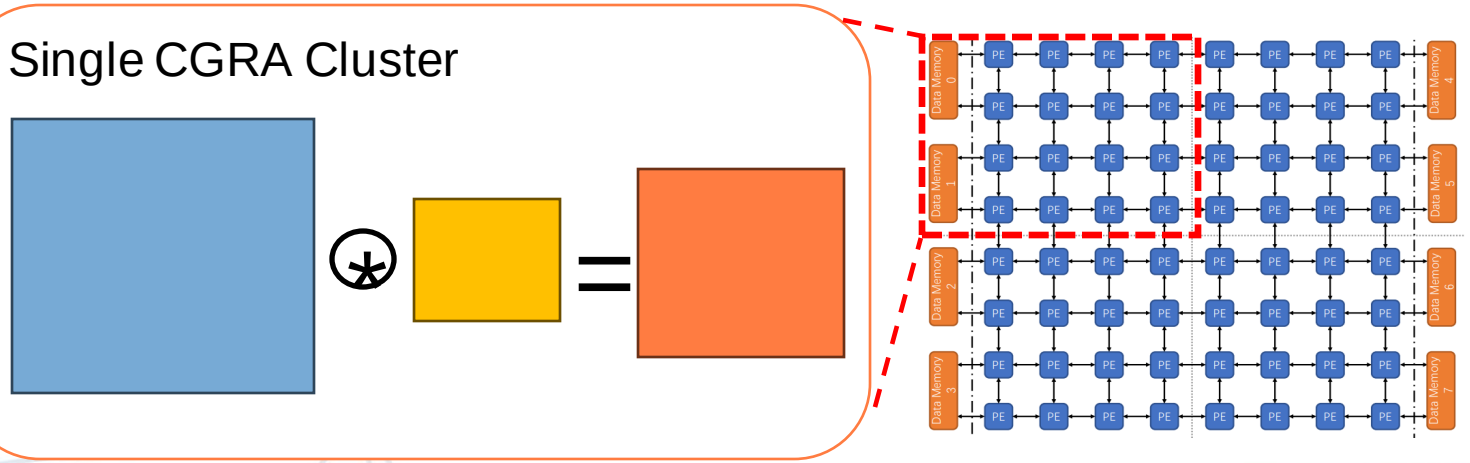


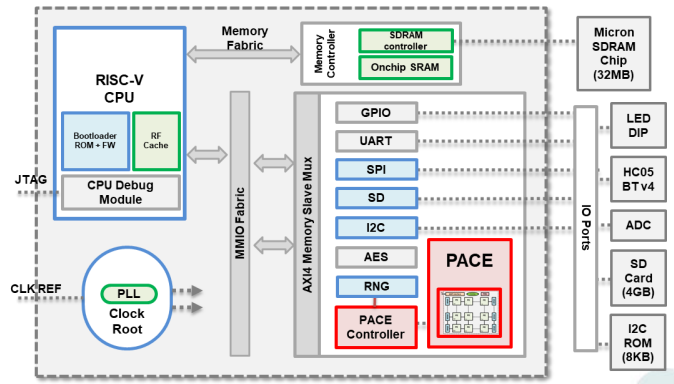


Convolution Mapping




```
//Sequential loop: from off-chip to on-chip
for i.0 in range (O1/ X*TO1):
for j.0 in range (O2/ Y*TO2):
for c.0 in range(Co/ TCo):
// Parallel loop: CGRA clusters
  for x in range(X):
  for y in range(Y):
// Single CGRA level
    for i in range(TO1):
    for j in range(TO2):
    for c in range(TCo):
        temp = 0;
        for k1 in range(K):
        for k2 in range(K):// map this:
            temp += I[] * W[];
    O[] = temp;
```





Micro kernel mapping on GEMM

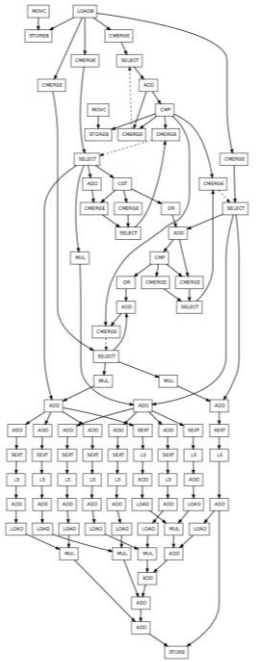
```
for (i=0;i<TI; i++)
for (j=0;j<TJ; j++)
for (k=0;k<TK; k++): //map this
  O[i][j] += W[i][k]* I[k][j];
```

Unrolling

```
for (i=0;i<TI; i++)
for (j=0;j<TJ; j++)
for (k=0;k<TK; k=k+4): //map this
  O[i][j] += W[i][k]* I[k][j]+ W[i][k+1]* I[k+1][j]
    W[i][k+2]* I[k+2][j]+W[i][k+3]* I[k+3][j];
```

Coalescing

```
for (n=0;i=0;j=0;k=0;n<TI*TJ*TK; n++){: //map this
  O[i][j] += W[i][k]*I[k][j]+W[i][k+1]*I[k+1][j]
  W[i][k+2]*I[k+2][j]+W[i][k+3]*I[k+3][j];k = k + 4;
  if(k+1 >= TK) {k=0; ++j;}
  if(j == TJ) {j=0; ++i;}}
```

DFG Nodes in the inner Loop		CGRA PE Utilization	Inner Loop Invocations
26		40.6%	4096 (TI x TJ)
58		60.1%	4096 (TI x TJ)
79		61.7%	1

Micro kernel mapping on Convolution

```
for (i=0; i<TO1; i++)
for (j=0; j<TO2; j++)
for (c=0; c<TCo; c++){
    temp = 0;
    for(k1 = 0; k1<K; k1++)
    for(k2 = 0; k2<K; k2++){: //map this
        temp += I[] * W[]
    }
    O[] = temp;}
```

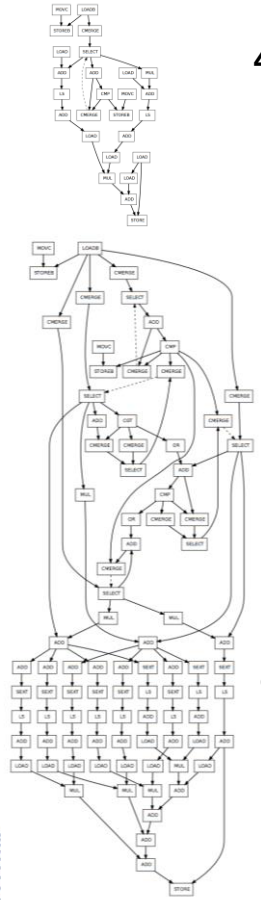
Unrolling

```
for (i=0; i<TO1; i++)
for (j=0; j<TO2; j++){//map this
    O[] = I[] * W[] + I[] * W[] + I[] * W[]
  + I[] * W[] + I[] * W[] + I[] * W[]
  + I[] * W[] + I[] * W[] + I[] * W[];}
```

Coalescing

```
for (int ijk=0;ijk<TCo*TO1*TO2; ijk++){: //map this
  O[] = I[] * W[] + I[] * W[] + I[] * W[]
  + I[] * W[] + I[] * W[] + I[] * W[]
  + I[] * W[] + I[] * W[] + I[] * W[];
  j = j + 1;
  if(j+1 > O2){j=0;++i;}
  if(i == O1){i=0;++c;}}
```

DFG Nodes in the inner Loop	CGRA PE Utilization	Inner Loop Invocations
27	42.2%	64x64x64x3 (TO1xTO2xTcoxK)
100	52%	4096 (TO1 x TO2)
153	86.9%	1



Performance comparison

Kernel	Nodes	II (MII)	Utilization	Compute time (ms)*	Data transfer time (ms)*	Total execution time (ms)*	Speedup
GEMM	26	4 (4)	40.63%	0.56	2.13	2.69	1×
GEMM-U	58	6 (4)	60.42%	0.25	2.13	2.38	1.1×
GEMM-U-C	79	8 (8)	61.72%	0.27	0.49	0.76	3.5×
CONV	27	4 (4)	42.19%	8.32	306.38	314.7	1×
CONV-U-C-1	100	12 (7)	52.08%	1.53	12.75	14.28	22×
CONV-U-C-2	153	11 (10)	86.93%	1.26	11.19	12.45	25.2×

* At 100 MHz CGRA frequency, 50 MBps host-to-CGRA data rate (GPIO)

NUS National University of Singapore

Conclusion

- Morpher CGRA compilation and simulation framework
  - Flexible to model modern CGRA architectures
  - Map complex workloads with a higher mapping quality at a shorter compilation time
  - Automatic RTL generation & cycle-accurate simulation to validate the compilation results
- Fully open-source
  - Modular codebase
  - Easy to modify
- Open source repository:
  - https://github.com/ecolab-nus/morpher-v2

SCAN ME

NUS National University of Singapore

Thank you!

NUS National University of Singapore