

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

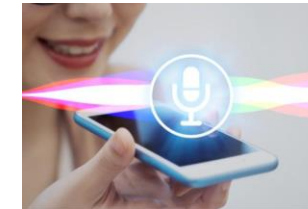
Several domain-specific accelerators plus multiple general-purpose cores on SoC are not tenable for such devices



Video



ML



Audio

Domain Specific Accelerators

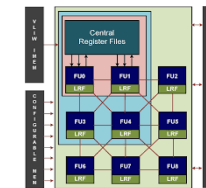


Apple iPhone 12's A14 Bionic SoC with
Domain Specific Accelerators

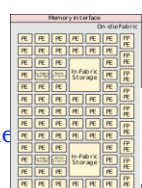
Reconfigurable Accelerators



Field Programmable Gate Arrays (FPGA)



SAMSUNG
Samsung
Reconfigurable
Processor

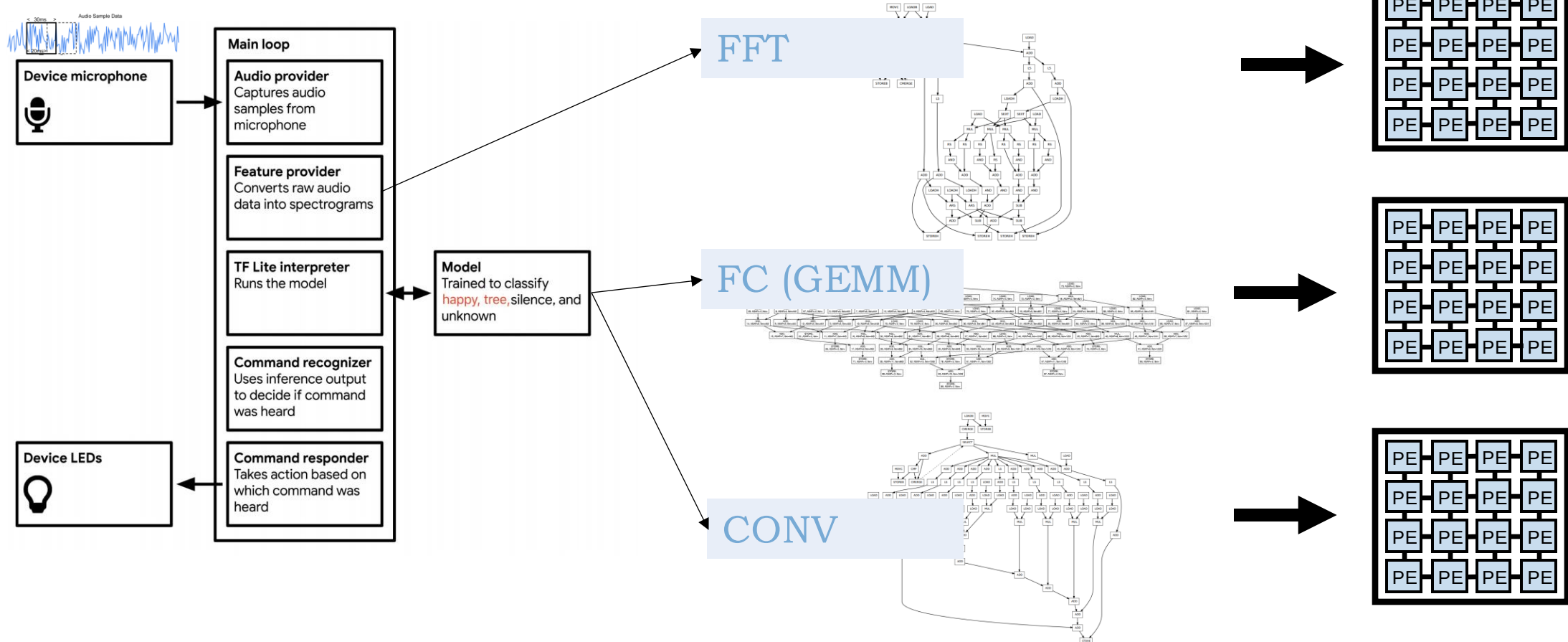


intel
Configurable
Spatial
Accelerator

Coarse-Grained Reconfigurable Array (CGRA)



CGRA: Supporting Diverse Dataflows



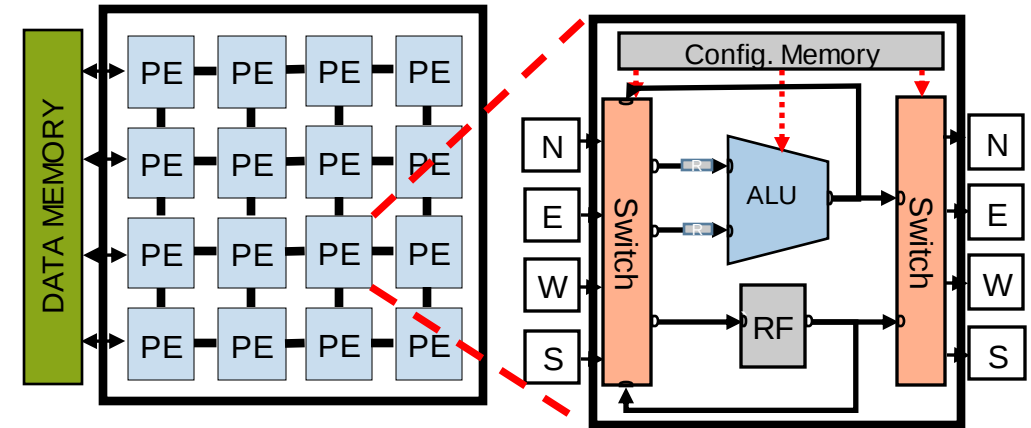
Speech Recognition Model

Application Kernels

Different Dataflows

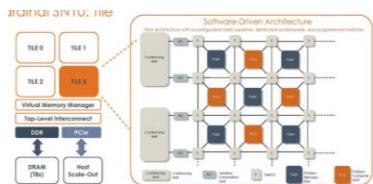
Coarse-Grained Reconfigurable Array (CGRA)

- **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)

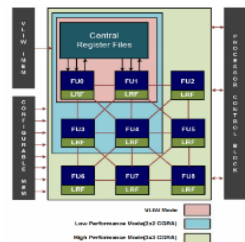
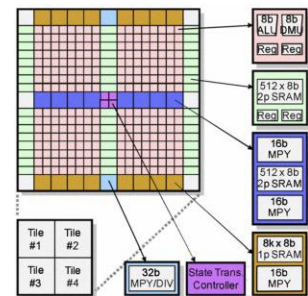
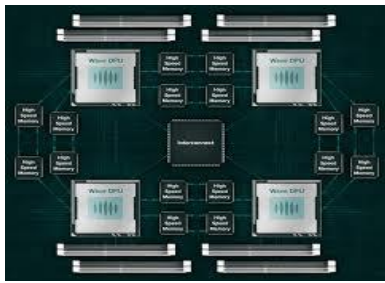


Figure 1. Structure of UIP-SRP

Samsung
Reconfigurable
Processor (SRP)

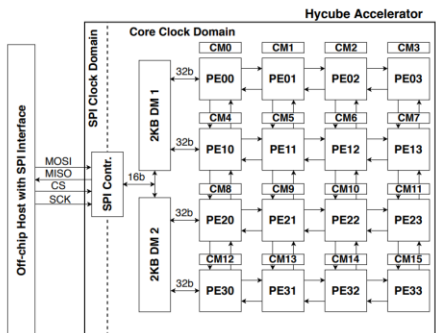


Renesas Dynamic
Reconfigurable Processor
(DRP)

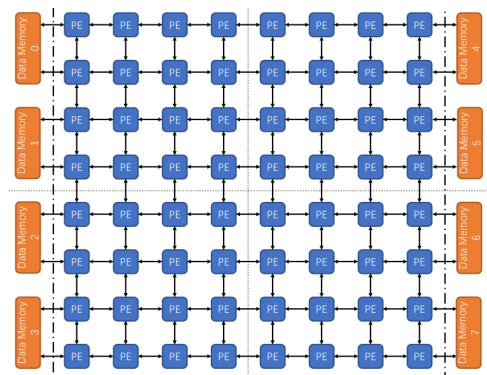
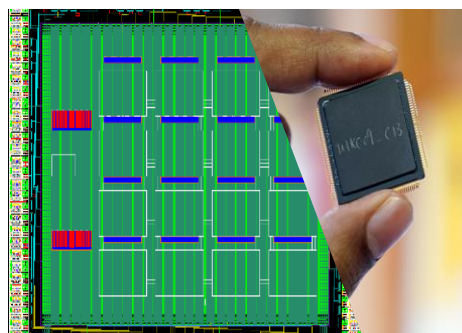


Wave DPU

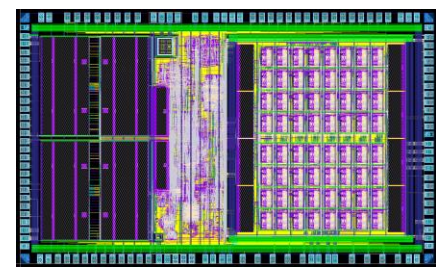
Commercial CGRAs



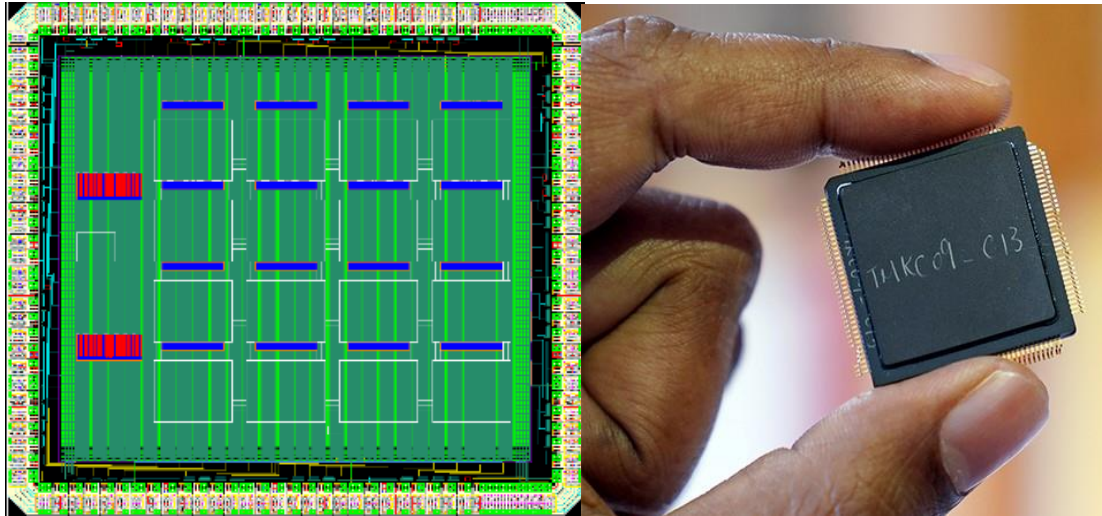
NUS HyCUBE



NUS PACE



HyCUBE CGRA @ NUS 2019



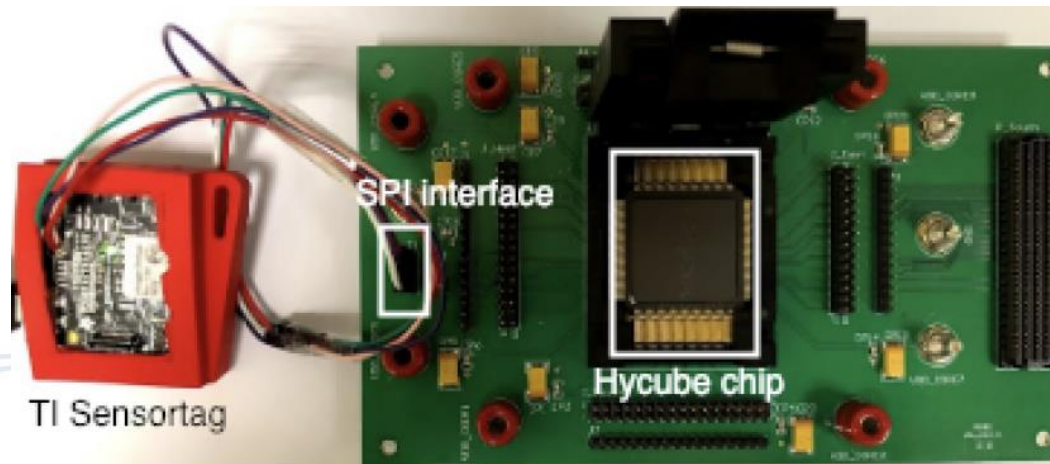
4x4 CGRA, TSMC 40nm

NUS HyCUBE: 90 MOPS/mW

Samsung SRP: 22 MOPS/mW

ARM CPU: 2.6 MOPS/mW

Xilinx FPGA: 25 MOPS/mW



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

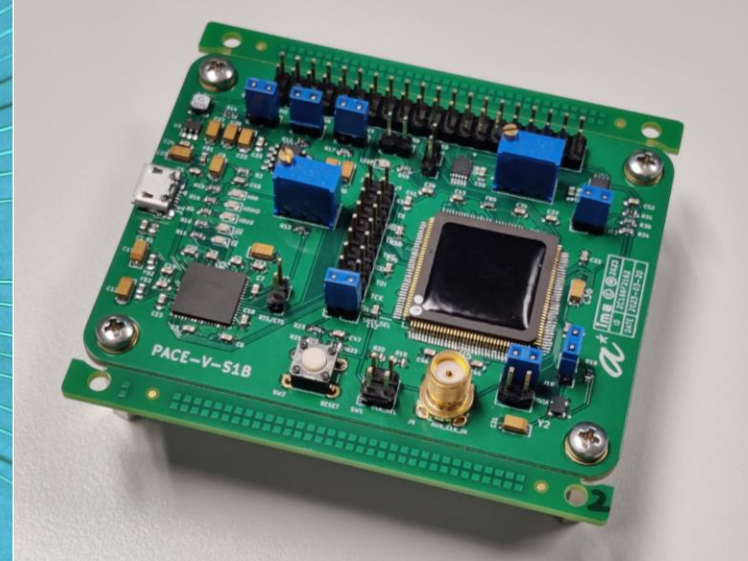
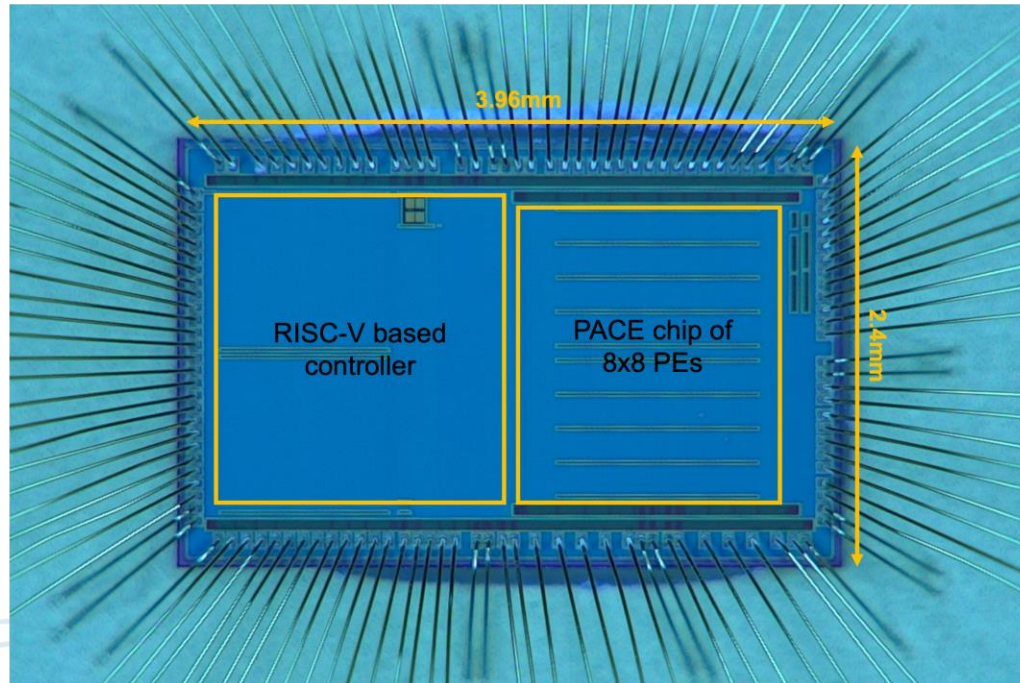


8x8 CGRA:

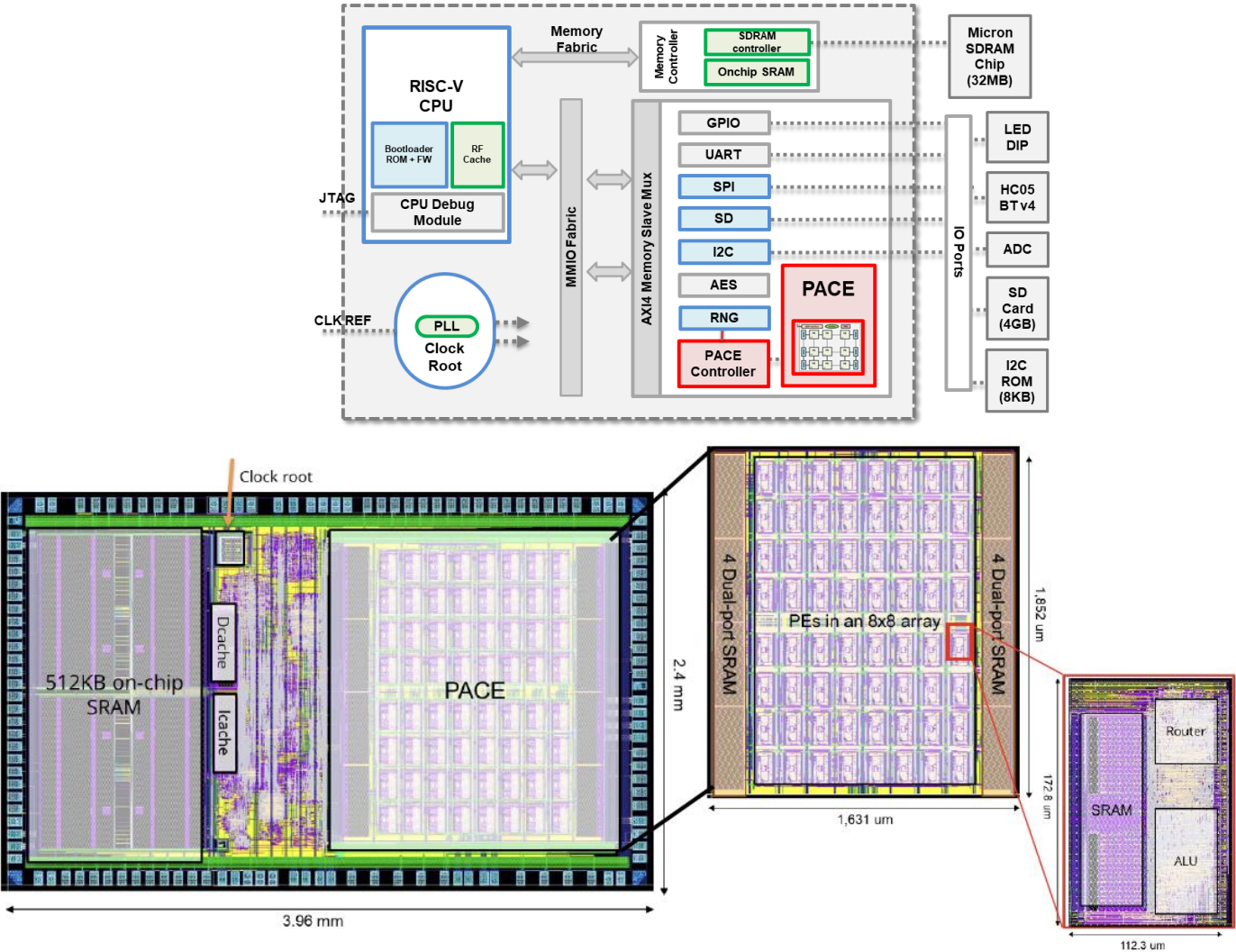
582 GOPS/W at 0.45V, 40nm ULP

1.1 mW at 10MHz

Estimated 1 TOPS/W at 22nm

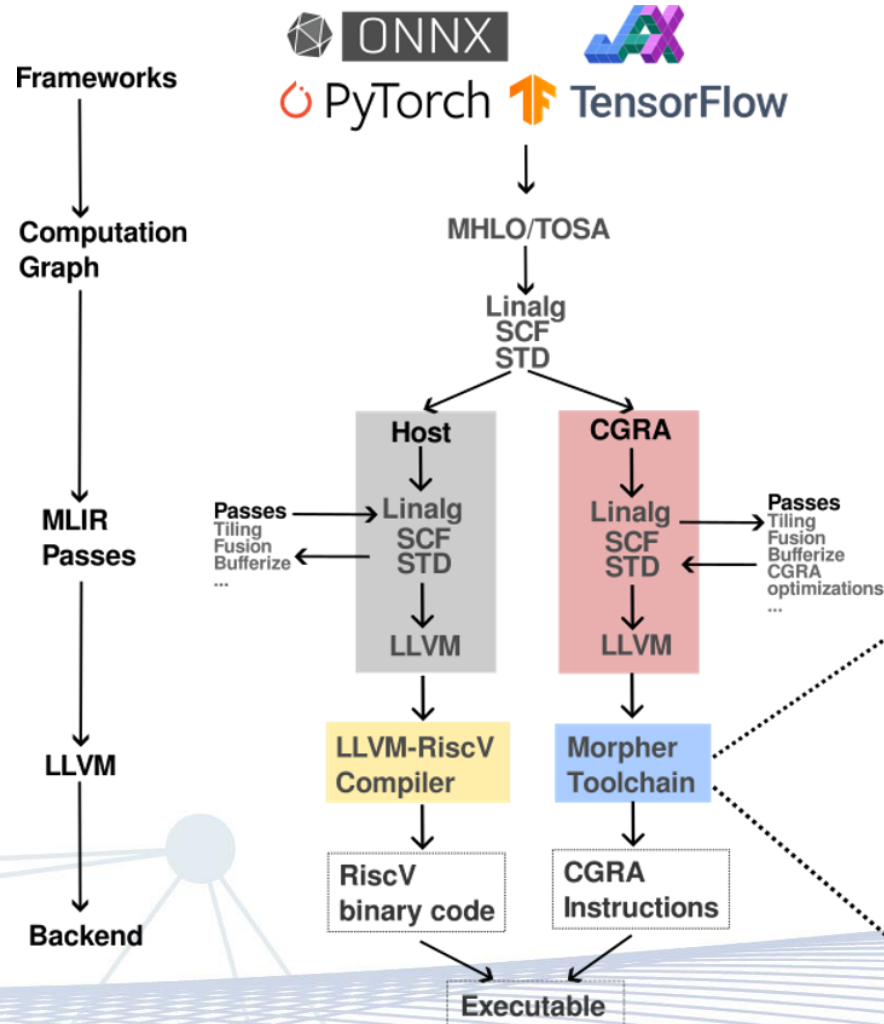


PACE CGRA SoC: NUS + IME

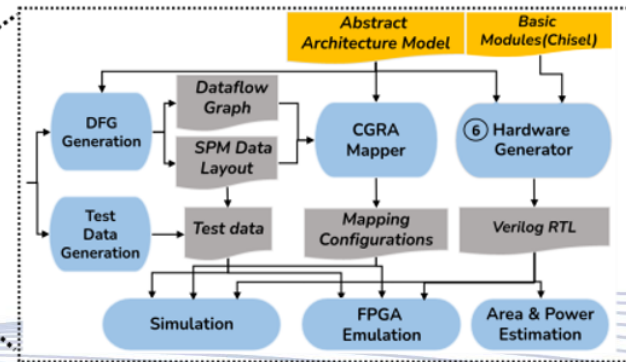


PACE SoC SPECIFICATIONS (SIMULATION)		
Tech. Node		UMC 40nm ULP
Area		3.96mm x 2.4mm (Die) 3.80mm x 2.0mm (Core)
Voltage		1V/3.3V
RISC-V based controller	Memory	512 KB on-chip memory 16KB ICache 16KB DCache
	Frequency	200/100 MHz
	Power	14mW @ 1.0V
PACE CGRA	#PEs	8x8 (17 instructions, including NOP)
	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 (Multi-level intermediate representation).
 - Support complex applications
 - Support heterogeneous system (RISC-V & CGRA) compilation
 - Leveraging MLIR builtin dialects and passes (linalg, std, scf...)

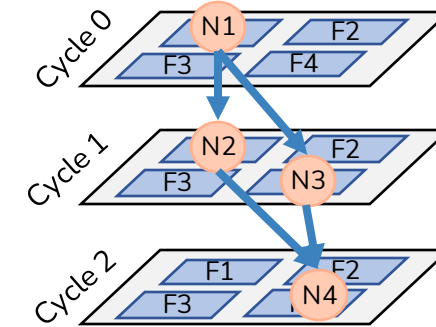
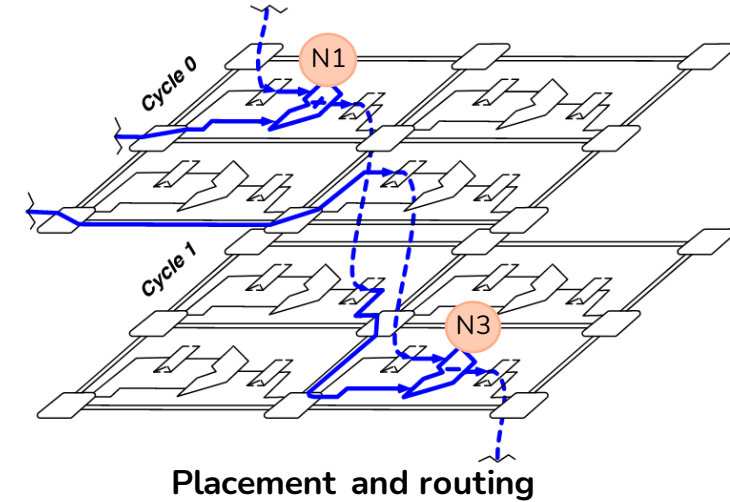
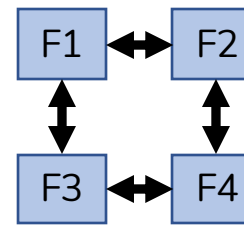
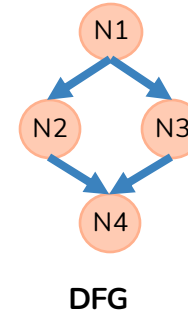
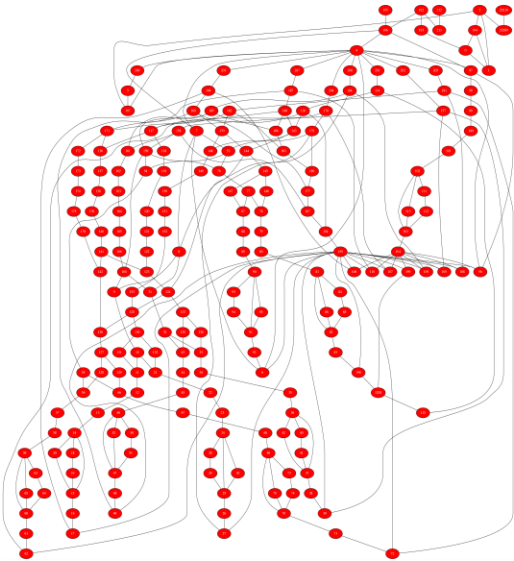


Application mapping on CGRA

- 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

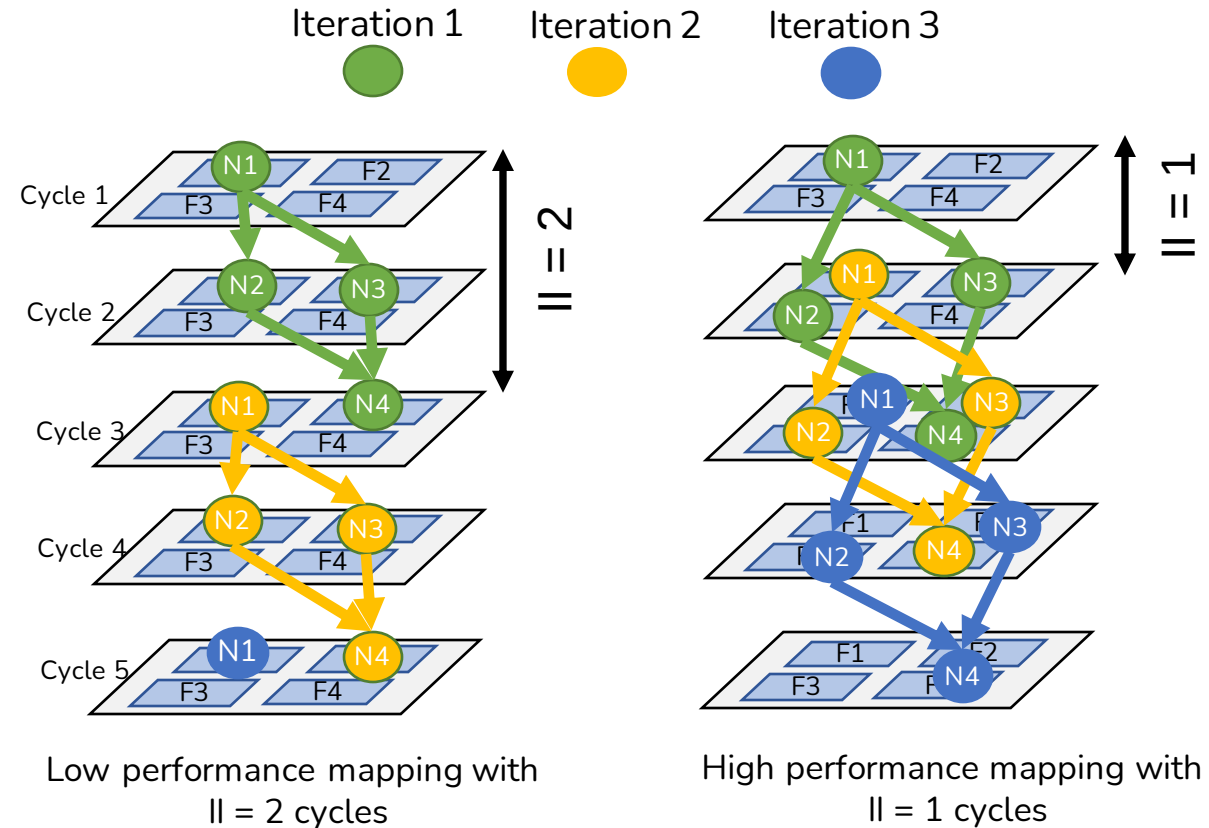
```

uint16_t read(void)
{
    uint16_t i;
    i = *pdc & 0xffff;
    for (i = 0; i < 8; i++)
    {
        #if (CPU == CM401)
            pldm_name_m0[i] =
                ReadPDC();
        #endif
        if (pdc[i] | pdc[i+1] | pdc[i+2] | pdc[i+3] | pdc[i+4] | pdc[i+5] | pdc[i+6] | pdc[i+7])
        {
            // Short circuit the 10 DCT if only the DC component is non-zero
            uint16_t s0 = 0;
            if (pdc[i] > 0) s0 = pdc[i];
            if (pdc[i+1] > 0) s0 = pdc[i+1];
            if (pdc[i+2] > 0) s0 = pdc[i+2];
            if (pdc[i+3] > 0) s0 = pdc[i+3];
            if (pdc[i+4] > 0) s0 = pdc[i+4];
            if (pdc[i+5] > 0) s0 = pdc[i+5];
            if (pdc[i+6] > 0) s0 = pdc[i+6];
            if (pdc[i+7] > 0) s0 = pdc[i+7];
        }
        else
        {
            uint16_t s0 = *pdc + 1;
            uint16_t s1 = *pdc + 3;
            uint16_t s2 = *pdc + 5;
            uint16_t s3 = *pdc + 7;
            uint16_t s4 = *pdc + 9;
            uint16_t s5 = *pdc + 11;
            uint16_t s6 = *pdc + 13;
            uint16_t s7 = *pdc + 15;
            uint16_t s8 = *pdc + 17;
            uint16_t s9 = *pdc + 19;
            uint16_t s10 = *pdc + 21;
            uint16_t s11 = *pdc + 23;
            uint16_t s12 = *pdc + 25;
            uint16_t s13 = *pdc + 27;
            uint16_t s14 = *pdc + 29;
            uint16_t s15 = *pdc + 31;
            uint16_t s16 = *pdc + 33;
            uint16_t s17 = *pdc + 35;
            uint16_t s18 = *pdc + 37;
            uint16_t s19 = *pdc + 39;
            uint16_t s20 = *pdc + 41;
            uint16_t s21 = *pdc + 43;
            uint16_t s22 = *pdc + 45;
            uint16_t s23 = *pdc + 47;
            uint16_t s24 = *pdc + 49;
            uint16_t s25 = *pdc + 51;
            uint16_t s26 = *pdc + 53;
            uint16_t s27 = *pdc + 55;
            uint16_t s28 = *pdc + 57;
            uint16_t s29 = *pdc + 59;
            uint16_t s30 = *pdc + 61;
            uint16_t s31 = *pdc + 63;
            uint16_t s32 = *pdc + 65;
            uint16_t s33 = *pdc + 67;
            uint16_t s34 = *pdc + 69;
            uint16_t s35 = *pdc + 71;
            uint16_t s36 = *pdc + 73;
            uint16_t s37 = *pdc + 75;
            uint16_t s38 = *pdc + 77;
            uint16_t s39 = *pdc + 79;
            uint16_t s40 = *pdc + 81;
            uint16_t s41 = *pdc + 83;
            uint16_t s42 = *pdc + 85;
            uint16_t s43 = *pdc + 87;
            uint16_t s44 = *pdc + 89;
            uint16_t s45 = *pdc + 91;
            uint16_t s46 = *pdc + 93;
            uint16_t s47 = *pdc + 95;
            uint16_t s48 = *pdc + 97;
            uint16_t s49 = *pdc + 99;
            uint16_t s50 = *pdc + 101;
            uint16_t s51 = *pdc + 103;
            uint16_t s52 = *pdc + 105;
            uint16_t s53 = *pdc + 107;
            uint16_t s54 = *pdc + 109;
            uint16_t s55 = *pdc + 111;
            uint16_t s56 = *pdc + 113;
            uint16_t s57 = *pdc + 115;
            uint16_t s58 = *pdc + 117;
            uint16_t s59 = *pdc + 119;
            uint16_t s60 = *pdc + 121;
            uint16_t s61 = *pdc + 123;
            uint16_t s62 = *pdc + 125;
            uint16_t s63 = *pdc + 127;
            uint16_t s64 = *pdc + 129;
            uint16_t s65 = *pdc + 131;
            uint16_t s66 = *pdc + 133;
            uint16_t s67 = *pdc + 135;
            uint16_t s68 = *pdc + 137;
            uint16_t s69 = *pdc + 139;
            uint16_t s70 = *pdc + 141;
            uint16_t s71 = *pdc + 143;
            uint16_t s72 = *pdc + 145;
            uint16_t s73 = *pdc + 147;
            uint16_t s74 = *pdc + 149;
            uint16_t s75 = *pdc + 151;
            uint16_t s76 = *pdc + 153;
            uint16_t s77 = *pdc + 155;
            uint16_t s78 = *pdc + 157;
            uint16_t s79 = *pdc + 159;
            uint16_t s80 = *pdc + 161;
            uint16_t s81 = *pdc + 163;
            uint16_t s82 = *pdc + 165;
            uint16_t s83 = *pdc + 167;
            uint16_t s84 = *pdc + 169;
            uint16_t s85 = *pdc + 171;
            uint16_t s86 = *pdc + 173;
            uint16_t s87 = *pdc + 175;
            uint16_t s88 = *pdc + 177;
            uint16_t s89 = *pdc + 179;
            uint16_t s90 = *pdc + 181;
            uint16_t s91 = *pdc + 183;
            uint16_t s92 = *pdc + 185;
            uint16_t s93 = *pdc + 187;
            uint16_t s94 = *pdc + 189;
            uint16_t s95 = *pdc + 191;
            uint16_t s96 = *pdc + 193;
            uint16_t s97 = *pdc + 195;
            uint16_t s98 = *pdc + 197;
            uint16_t s99 = *pdc + 199;
            uint16_t s100 = *pdc + 201;
            uint16_t s101 = *pdc + 203;
            uint16_t s102 = *pdc + 205;
            uint16_t s103 = *pdc + 207;
            uint16_t s104 = *pdc + 209;
            uint16_t s105 = *pdc + 211;
            uint16_t s106 = *pdc + 213;
            uint16_t s107 = *pdc + 215;
            uint16_t s108 = *pdc + 217;
            uint16_t s109 = *pdc + 219;
            uint16_t s110 = *pdc + 221;
            uint16_t s111 = *pdc + 223;
            uint16_t s112 = *pdc + 225;
            uint16_t s113 = *pdc + 227;
            uint16_t s114 = *pdc + 229;
            uint16_t s115 = *pdc + 231;
            uint16_t s116 = *pdc + 233;
            uint16_t s117 = *pdc + 235;
            uint16_t s118 = *pdc + 237;
            uint16_t s119 = *pdc + 239;
            uint16_t s120 = *pdc + 241;
            uint16_t s121 = *pdc + 243;
            uint16_t s122 = *pdc + 245;
            uint16_t s123 = *pdc + 247;
            uint16_t s124 = *pdc + 249;
            uint16_t s125 = *pdc + 251;
            uint16_t s126 = *pdc + 253;
            uint16_t s127 = *pdc + 255;
            uint16_t s128 = *pdc + 257;
            uint16_t s129 = *pdc + 259;
            uint16_t s130 = *pdc + 261;
            uint16_t s131 = *pdc + 263;
            uint16_t s132 = *pdc + 265;
            uint16_t s133 = *pdc + 267;
            uint16_t s134 = *pdc + 269;
            uint16_t s135 = *pdc + 271;
            uint16_t s136 = *pdc + 273;
            uint16_t s137 = *pdc + 275;
            uint16_t s138 = *pdc + 277;
            uint16_t s139 = *pdc + 279;
            uint16_t s140 = *pdc + 281;
            uint16_t s141 = *pdc + 283;
            uint16_t s142 = *pdc + 285;
            uint16_t s143 = *pdc + 287;
            uint16_t s144 = *pdc + 289;
            uint16_t s145 = *pdc + 291;
            uint16_t s146 = *pdc + 293;
            uint16_t s147 = *pdc + 295;
            uint16_t s148 = *pdc + 297;
            uint16_t s149 = *pdc + 299;
            uint16_t s150 = *pdc + 301;
            uint16_t s151 = *pdc + 303;
            uint16_t s152 = *pdc + 305;
            uint16_t s153 = *pdc + 307;
            uint16_t s154 = *pdc + 309;
            uint16_t s155 = *pdc + 311;
            uint16_t s156 = *pdc + 313;
            uint16_t s157 = *pdc + 315;
            uint16_t s158 = *pdc + 317;
            uint16_t s159 = *pdc + 319;
            uint16_t s160 = *pdc + 321;
            uint16_t s161 = *pdc + 323;
            uint16_t s162 = *pdc + 325;
            uint16_t s163 = *pdc + 327;
            uint16_t s164 = *pdc + 329;
            uint16_t s165 = *pdc + 331;
            uint16_t s166 = *pdc + 333;
            uint16_t s167 = *pdc + 335;
            uint16_t s168 = *pdc + 337;
            uint16_t s169 = *pdc + 339;
            uint16_t s170 = *pdc + 341;
            uint16_t s171 = *pdc + 343;
            uint16_t s172 = *pdc + 345;
            uint16_t s173 = *pdc + 347;
            uint16_t s174 = *pdc + 349;
            uint16_t s175 = *pdc + 351;
            uint16_t s176 = *pdc + 353;
            uint16_t s177 = *pdc + 355;
            uint16_t s178 = *pdc + 357;
            uint16_t s179 = *pdc + 359;
            uint16_t s180 = *pdc + 361;
            uint16_t s181 = *pdc + 363;
            uint16_t s182 = *pdc + 365;
            uint16_t s183 = *pdc + 367;
            uint16_t s184 = *pdc + 369;
            uint16_t s185 = *pdc + 371;
            uint16_t s186 = *pdc + 373;
            uint16_t s187 = *pdc + 375;
            uint16_t s188 = *pdc + 377;
            uint16_t s189 = *pdc + 379;
            uint16_t s190 = *pdc + 381;
            uint16_t s191 = *pdc + 383;
            uint16_t s192 = *pdc + 385;
            uint16_t s193 = *pdc + 3
```



Application mapping on CGRA

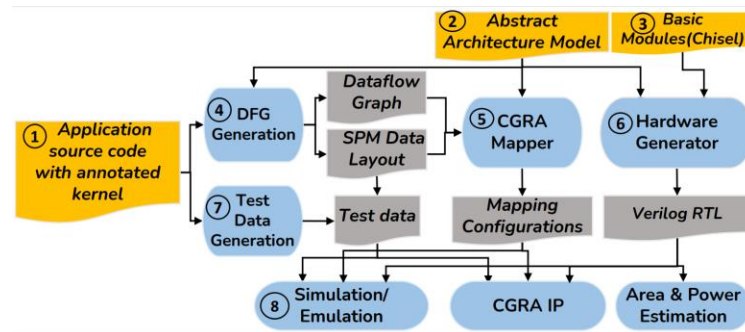
- Software pipelined schedule
- Goal: Mapping with minimum initiation interval
- Initiation interval (II) = cycle difference between initiation of consecutive iterations
- Low II $\rightarrow$ High performance



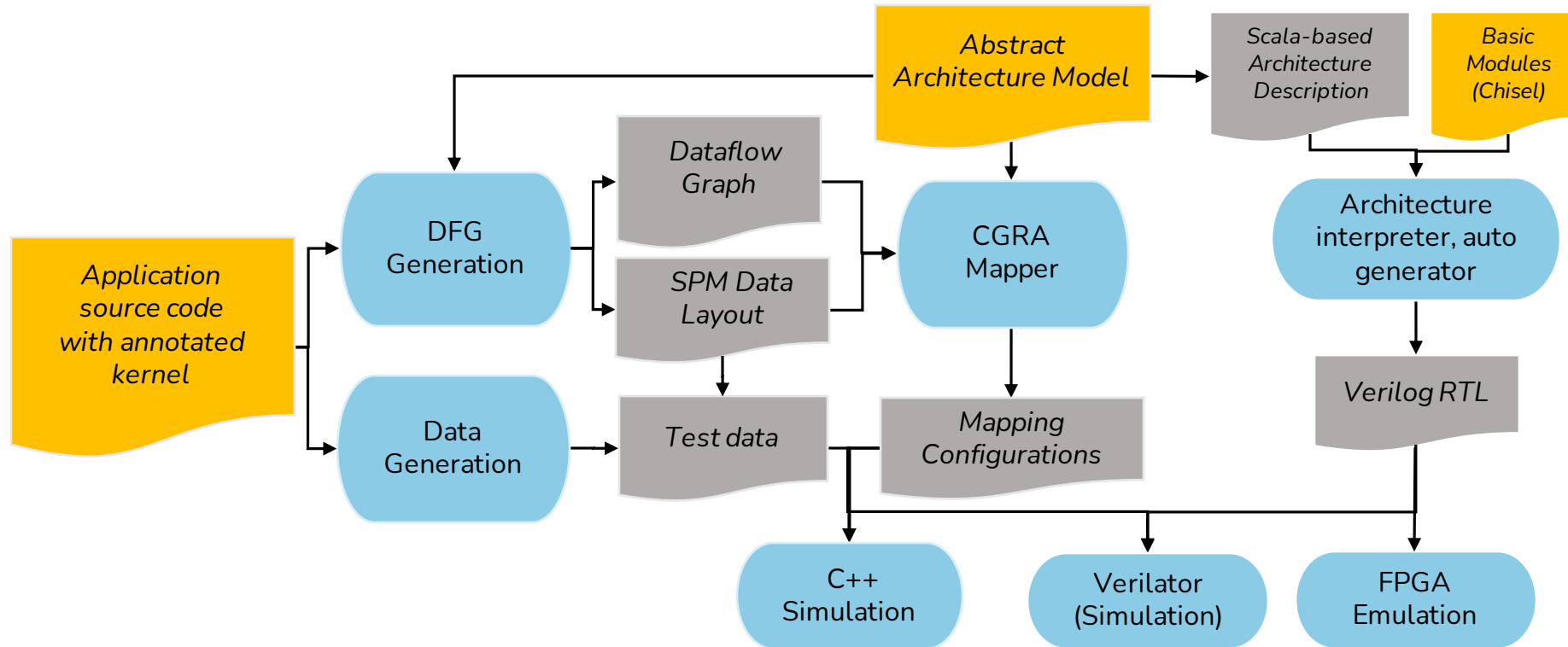
Morpher: An Integrated Compilation and Simulation Framework for CGRA

- 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

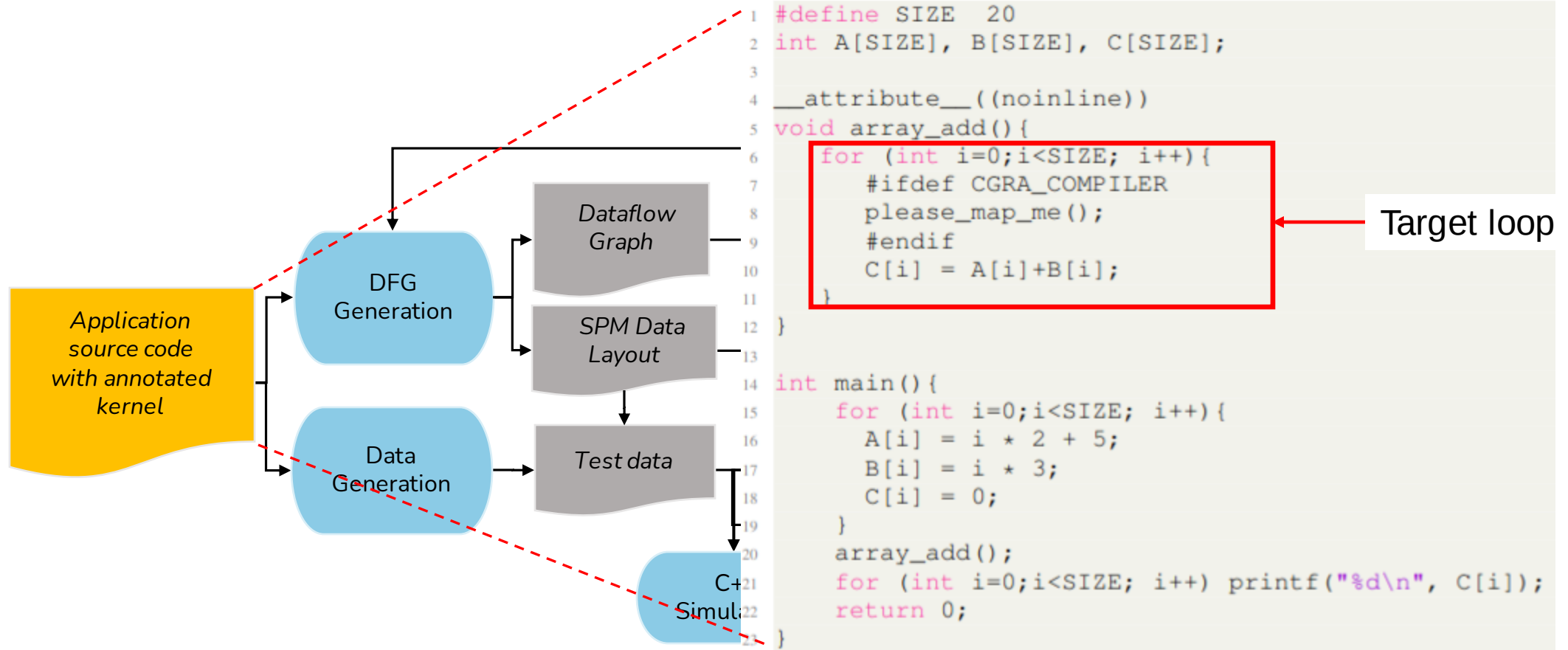
Features		CGRA-ME	Pillars	Open-CGRA	CCF	Morpher
DFG Generation	Models control divergence	X	X	✓	✓	✓
	Recurrence edges	X	X	✓	✓	✓
Architecture Modelling	Adapt user defined architectures	✓	✓	✓	X	✓
	Multi-hop connections	X	X	X	X	✓
	Different memory organizations	X	X	✓	X	✓
P&R Mapper	Architecture adaptive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



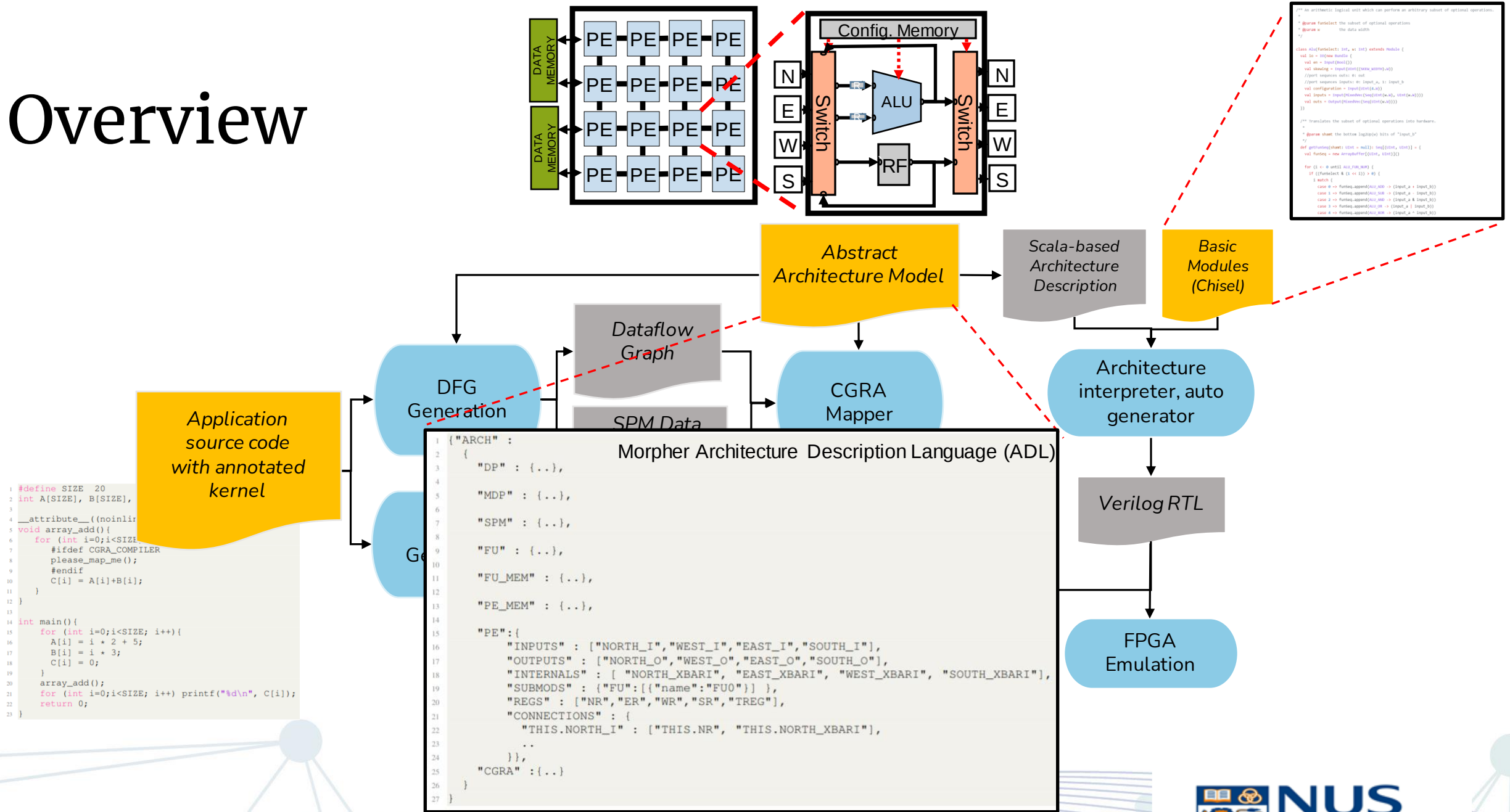
Overview



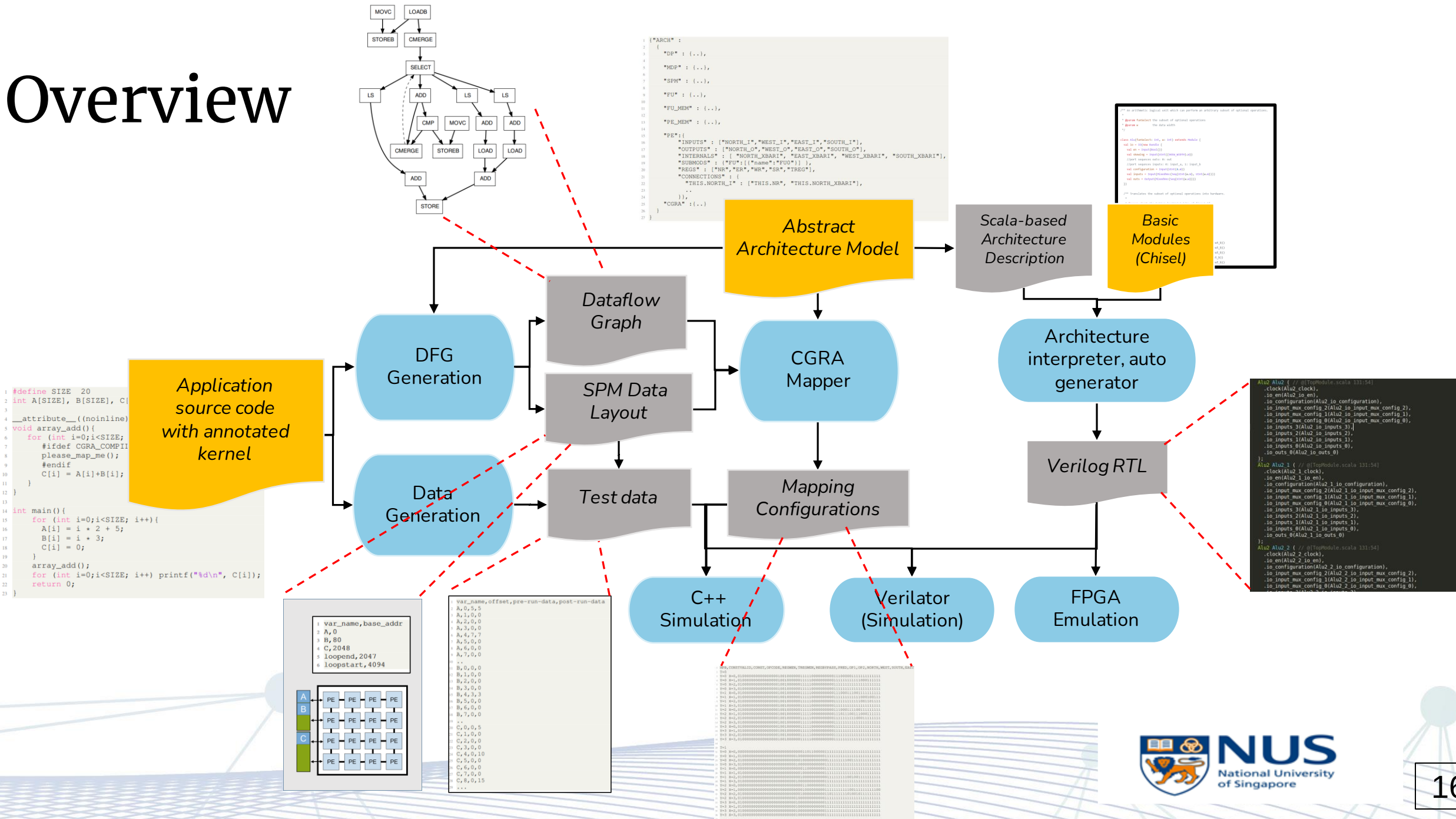
Overview



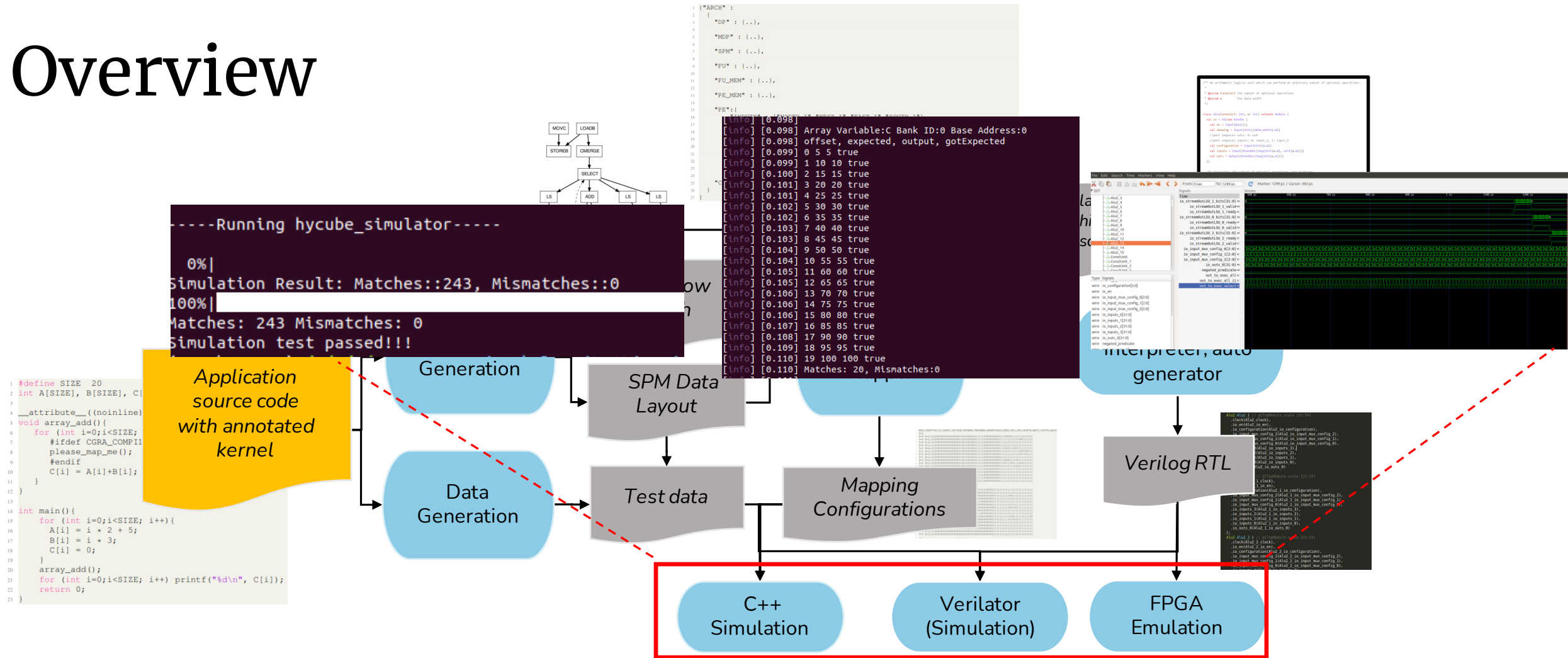
Overview



Overview

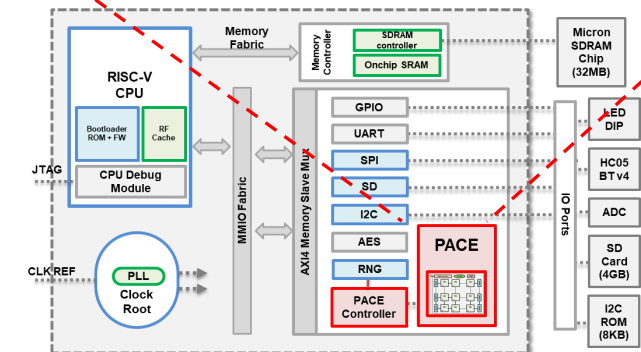
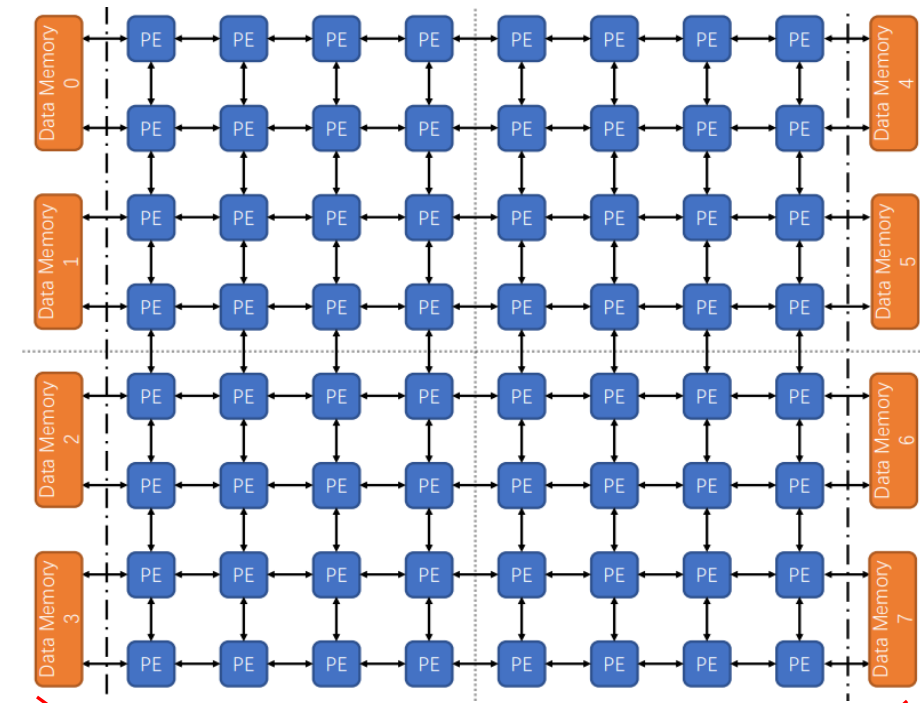


Overview



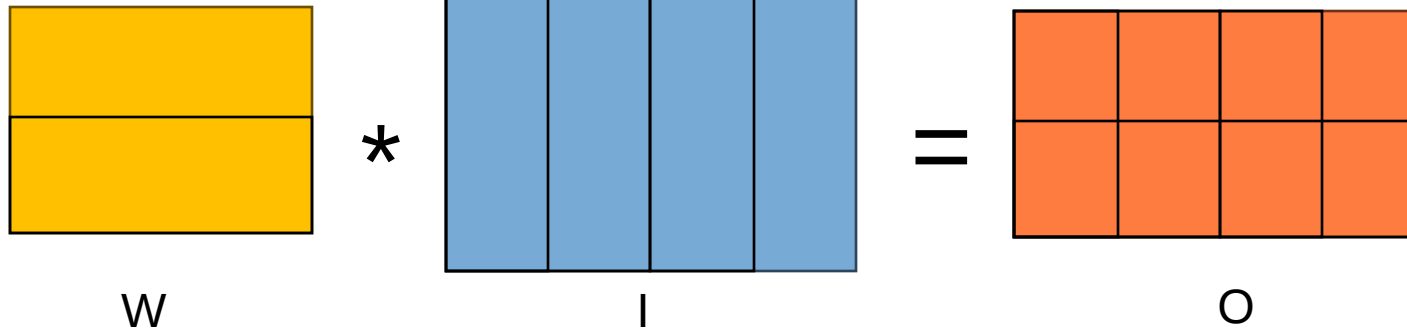
Experimental Study

- 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 $64 \times 64 \times 64$; Tile Size: $64 \times 16 \times 64$
 - CONV: Dimension of $64 \times 64 \times 64 \times 3 \times 3$; Tile Size: $64 \times 64 \times 1 \times 3 \times 3$



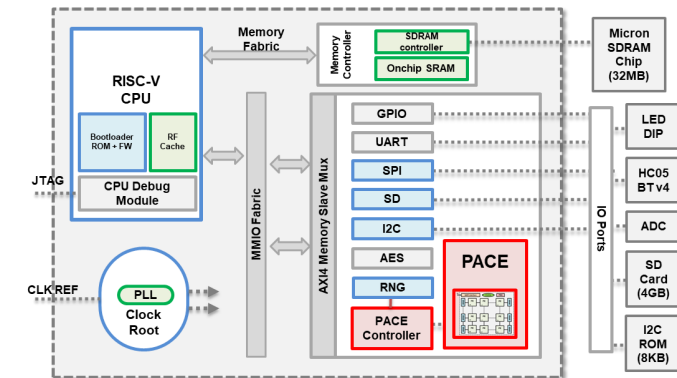
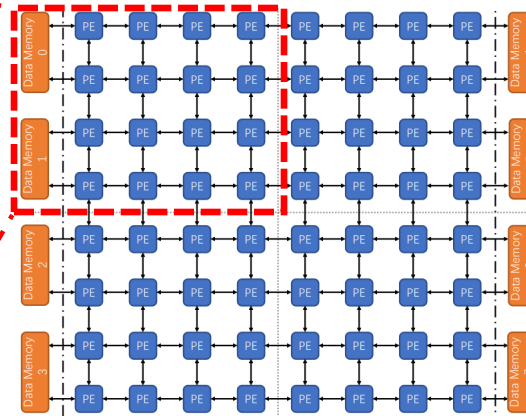
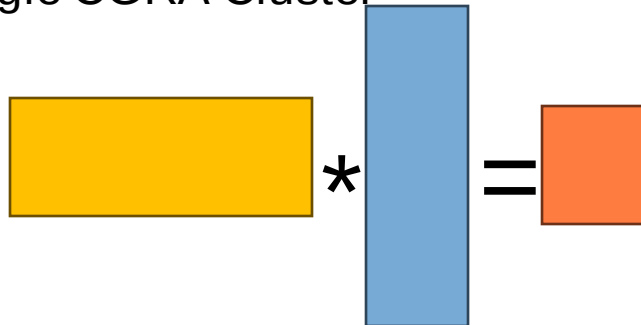
GEMM Mapping

GEMM Computation



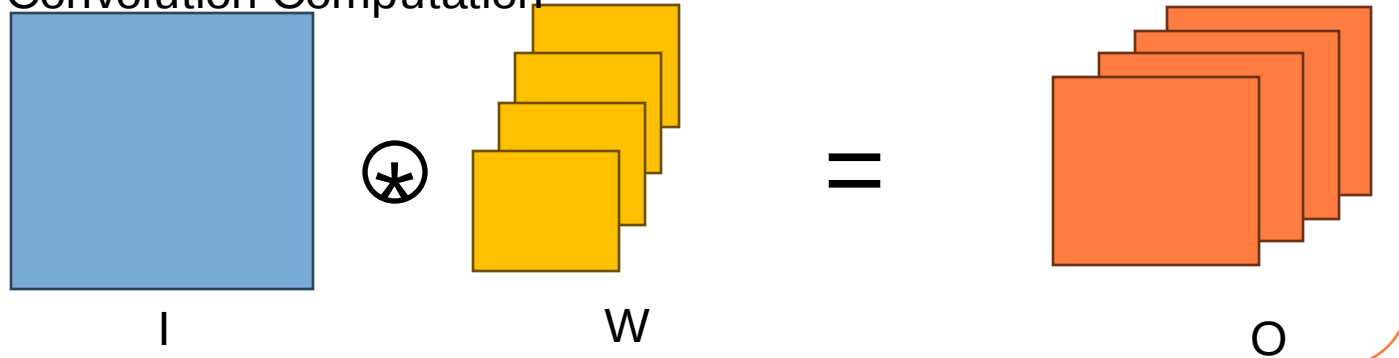
```
1 // Sequential loop: from off-chip to on-chip
2 for m in range(M/(TI*X)):
3   for n in range(N/(TJ*Y)):
4     for k in range(K/(TK)):
5       // Parallel loop: CGRA clusters
6       for x in range(X):
7         for y in range(Y):
8           // Single CGRA level
9           for i in range(TI):
10            for j in range(TJ):
11              for k in range(TK): //map this
12                O[i][j] += W[i][x] * I[x][j];
```

Single CGRA Cluster



Convolution Mapping

Convolution Computation

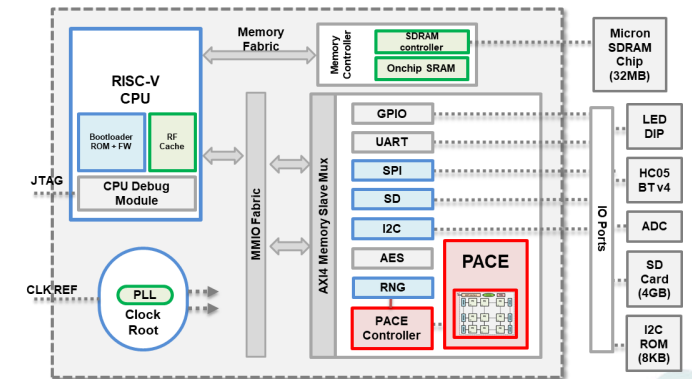
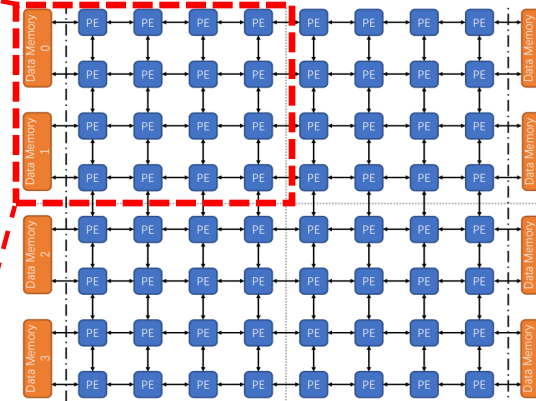
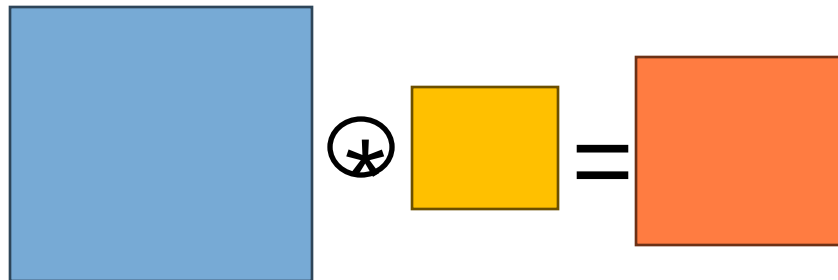


```

1 //Sequential loop: from off-chip to on-chip
2 for i.0 in range (O1/ X*TO1):
3   for j.0 in range (O2/ Y*TO2):
4     for c.0 in range(Co/ TCo):
5       // Parallel loop: CGRA clusters
6       for x in range(X):
7         for y in range(Y):
8           // Single CGRA level
9           for i in range(TO1):
10            for j in range(TO2):
11              for c in range(TCo):
12                temp = 0;
13                for k1 in range(K):
14                  for k2 in range(K): // map this:
15                    temp += I[] * W[];
16                O[] = temp;

```

Single CGRA Cluster



Micro kernel mapping on GEMM

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

Unrolling

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

Coalescing

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

DFG Nodes in
the inner Loop

26



58



79

CGRA PE
Utilization

40.6%



60.1%



61.7%

Inner Loop
Invocations

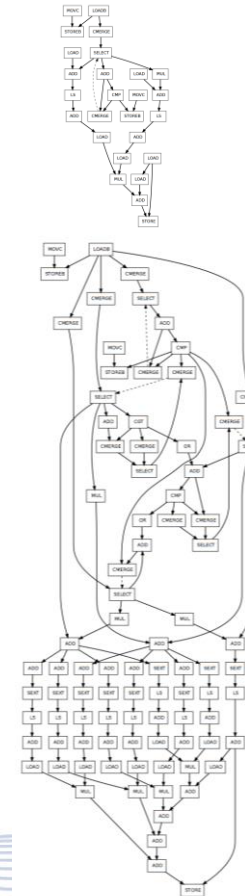
4096 (TI x TJ)



4096 (TI x TJ)



1



Micro kernel mapping on Convolution

```
1 for (i=0; i<TO1; i++)
2 for (j=0; j<TO2; j++)
3 for (c=0; c<TCO; c++){
4   temp = 0;
5   for(k1 = 0; k1<K; k1++)
6     for(k2 = 0; k2<K; k2++){ //map this
7       temp += I[] * W[]
8     }
9   O[] = temp;}
```

Unrolling

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

Coalescing

```
1 for (int ijk=0;ijk<TCO*TO1*TO2; ijk++){ //map this
2   O[] = I[] * W[] + I[] * W[] + I[] * W[]
3   + I[] * W[] + I[] * W[] + I[] * W[]
4   + I[] * W[] + I[] * W[] + I[] * W[];
5   j = j + 1;
6   if(j+1 > O2){j=0;++i;}
7   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)

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>





Thank you!