Efficient Mapping of CDFG onto Coarse-Grained Reconfigurable Array Architectures

Satyajit Das[*†], Kevin J. M. Martin[*], Philippe Coussy[*], Davide Rossi[†], and Luca Benini[† ‡]

[*]Université de Bretagne-Sud, France,

[†]University of Bologna, Italy

[‡]Integrated Systems Laboratory, ETH Zurich, Switzerland,

ASP-DAC January 17, 2017

Outline

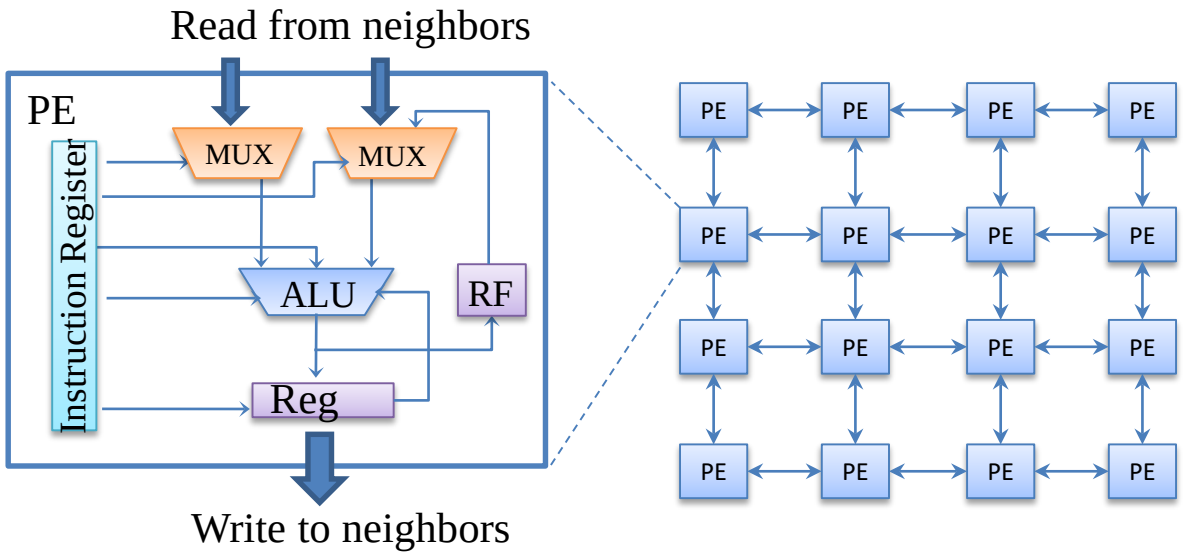
- Introduction & Background
- Contribution
  - Problem formulation
  - Proposed method
  - Proposed architecture
- Experimental results
- Conclusion

Outline

- Introduction & Background
- Contribution
  - Problem formulation
  - Proposed method
  - Proposed architecture
- Experimental results
- Conclusion

Introduction

- Coarse-grain reconfigurable architecture (CGRA)
  - Processing Elements (PEs) arranged as 2D structure
  - PE reconfiguration through Instruction register



Introduction

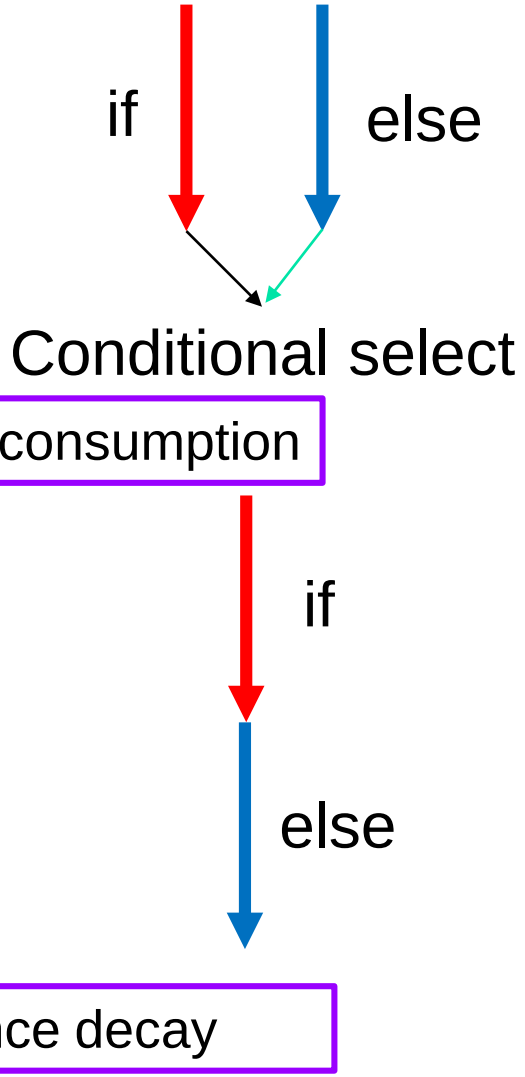
- CGRAs are coupled to a general purpose processor, where the processor executes the control parts and delegates data-flow parts to the CGRA
  - Communication overhead + memory operation overhead not suitable for low power solutions
- Existing CGRAs mostly rely on partial and full predication techniques to support conditional branches
  - Increased number of operations and performance decay lead to higher energy consumption

Background

- Control flow
  - Loops and conditionals
- Solutions for loops : Software pipelining for innermost loop only
  - No hardware support necessary
- Solutions for conditionals : partial and full predication
  - Hardware support necessary for the predication

Background

- Partial predication
  - Executes both (if and else) paths
  - Chooses the correct result later by using a predicated instruction (conditional move)
- Full predication
  - The operations that update the same variable are mapped to the same PE but at different times
  - The correct value of the output will be present in the PE after the maximum time



Outline

- Introduction & Background
- Contribution
  - Problem formulation
  - Proposed method
  - Proposed architecture
- Experimental results
- Conclusion

Contribution

- Register allocation based approach to execute **full control flow (branches & loops)** on CGRAs in an ultra-low-power (ULP) environment,
  - Helps mapping of any depth of loops and conditionals
  - Assists execution of kernels completely releasing the host processor from performing outer loops control
  - Addresses standalone execution of all the levels of the loops, and conditionals
- CGRA architectural template targeted to execute full kernels in ultra low power (ULP) environment, supporting execution of *jump* and *conditional jump* instructions

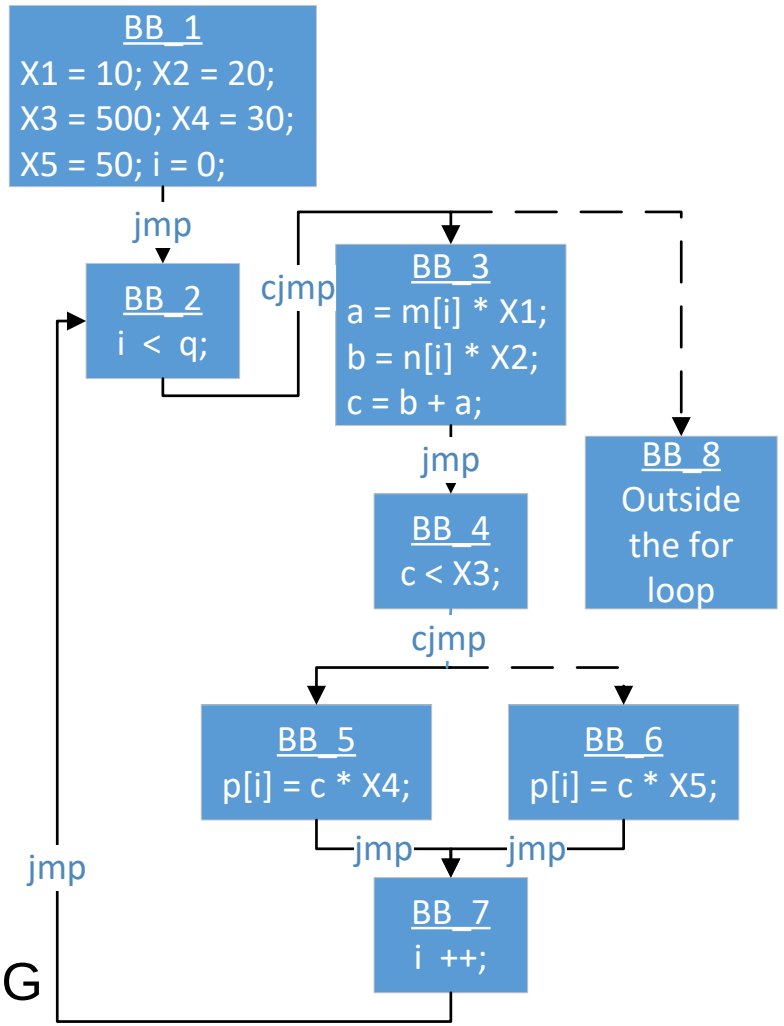
Outline

- Introduction & Background
- Contribution
  - Problem formulation
  - Proposed method
  - Proposed architecture
- Experimental results
- Conclusion

Problem formulation

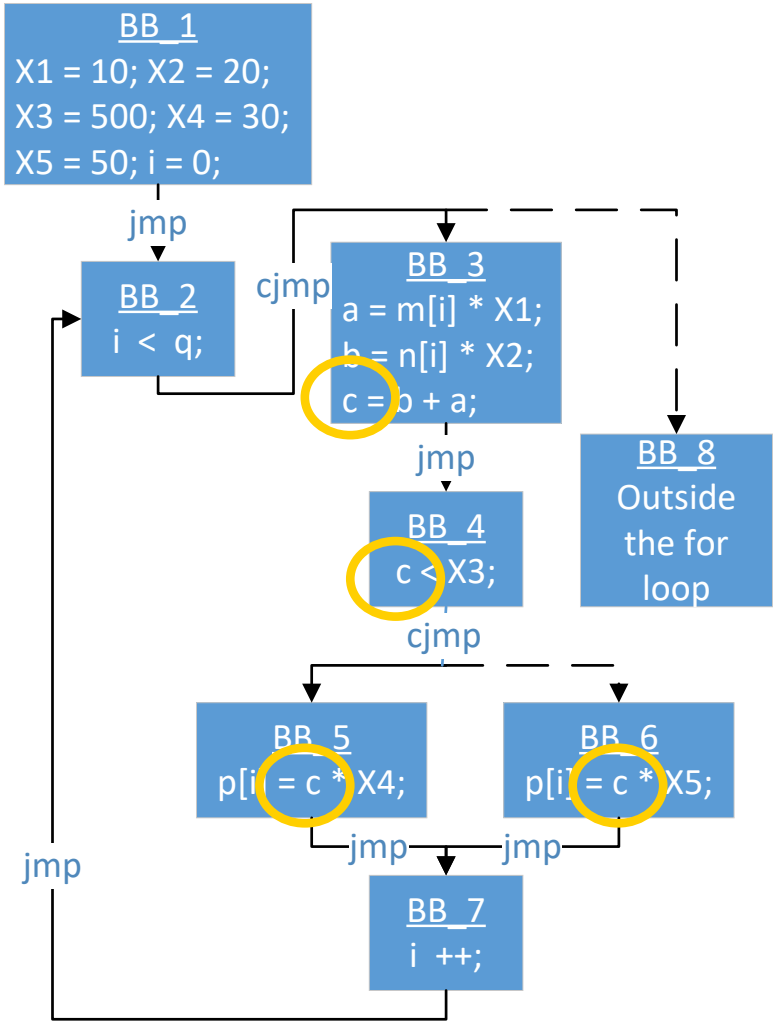
```
X1 = 10;
X2 = 20;
X3 = 500;
X4 = 30;
X5 = 50
for(i = 0; i < q; i++)
{
 a = m[i] * X1;
 b = n[i] * X2;
 c = b + a;
 if(c < X3)
          p[i] = c * X4;
 else
          p[i] = c * X5;
}
```

(a) Sample program



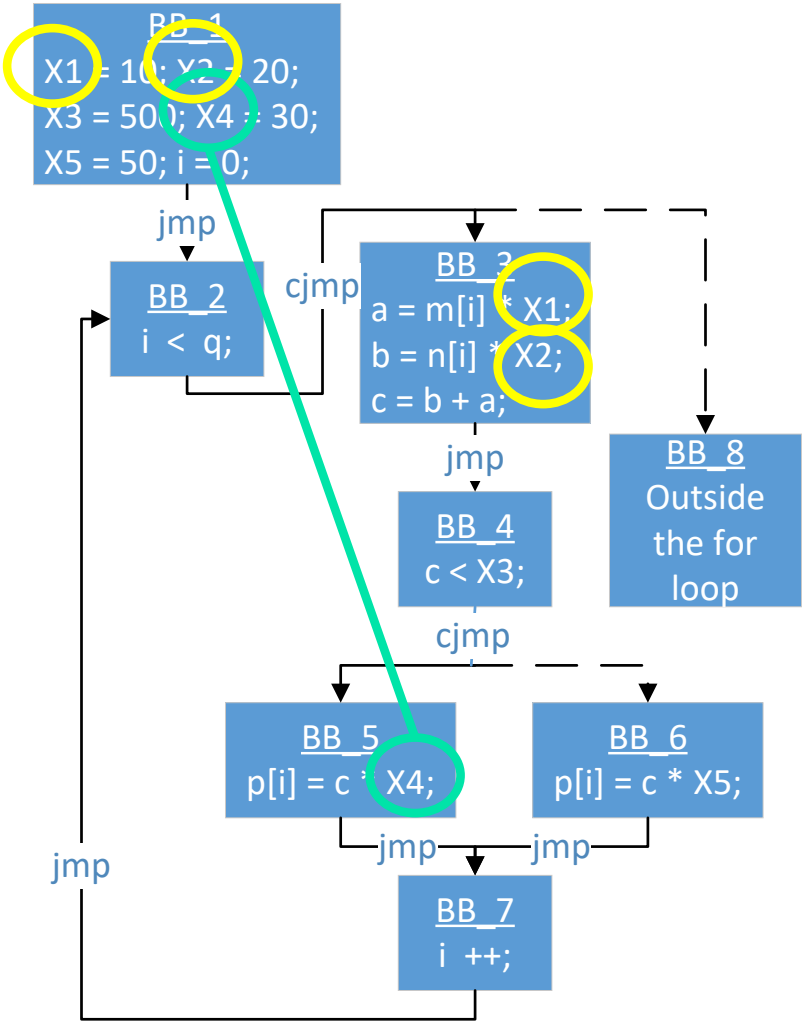
(b) Corresponding CDFG

Problem formulation



- Basic blocks => DFG
- Control flow => jmp, cjmp
- Execution of basic blocks mutually exclusive => discrete mapping of basic blocks
- Individual mapping of basic blocks creates placement constraints
  - Ex: placement of c (reg_c) in BB_3, BB_4, BB_5, BB_6 must be same
  - Variables with such placement constraints => Symbol variables

Problem formulation



- Behavioral classification of the placement constraints
  - Target location constraints (TLC)
    - Ex: For a mapping order of BB_1, BB_3: reg_X1, and reg_X2 in BB_3 is TLC
  - Reserved location constraints (RLC)
    - Ex: For a mapping order of BB_1, BB_3: reg_X3, reg_X4, reg_X5 in BB_3 are RLCs

Problem formulation

- Increase in the number of TLCs and RLCs induce complexity in mapping
- The number of TLCs and RLCs varies on traversal of the CDFG
- **Basic solution**: systematic load-store based approach
  - Introducing memory operation nullifies the placement problem
  - Not an energy efficient solutions (increased memory operation)

Outline

- Introduction & Background
- Contribution
  - Problem formulation
  - Proposed method
  - Proposed architecture
- Experimental results
- Conclusion

Proposed Method

- Register allocation based approach
  - Except array inputs and outputs, all the scalar variables are placed in the register files of the PEs

Proposed Method

- Register allocation based approach
  - Except array inputs and outputs, all the scalar variables are placed in the register files of the PEs

```
X1 = 10;
X2 = 20;
X3 = 500;
X4 = 30;
X5 = 50
for(i = 0; i < q; i++)
{
 a = m[i] * X1;
 b = n[i] * X2;
 c = b + a;
 if(c < X3)
        p[i] = c * X4;
 else
        p[i] = c * X5;
}
```

- In the sample program *m, n* are input arrays, and *p* is output array
  - Input arrays are loaded from memory
  - Output arrays are stored in memory
- All the other variables are placed in register files (no memory operations)

Proposed Method

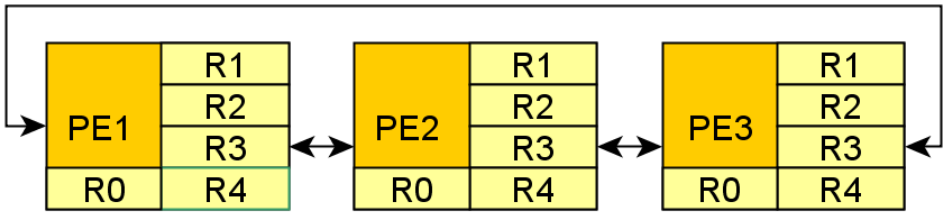
- Register allocation based approach
  - Except array inputs and outputs, all the scalar variables are placed in the register files of the PEs
  - Modified Forward traversal of CDFG to minimize the number of TLC and RLCs
    - Forward breadth first traversal helps the basic blocks with greater number of symbol nodes are mapped earlier => reduction in the number of constraints.

Proposed Method

- Register allocation based approach
  - Except array inputs and outputs, all the scalar variables are placed in the register files of the PEs
  - Modified Forward traversal of CDFG to minimize the number of TLC and RLCs
    - Forward breadth first traversal helps the basic blocks with greater number of symbol nodes are mapped earlier => reduction in the number of constraints.
  - Introduce routing to respect TLC or RLC, while mapping the basic blocks

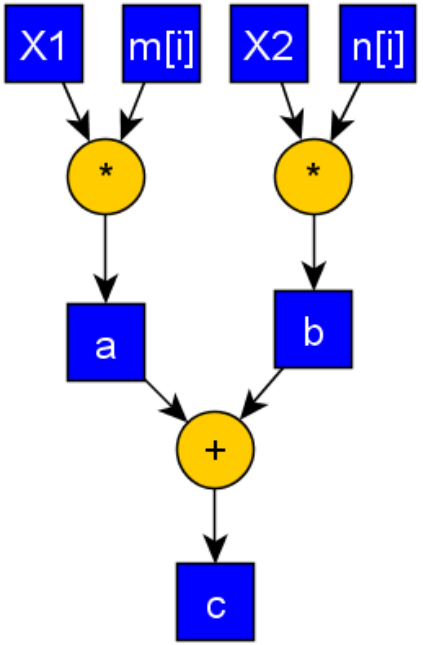
Proposed Method

- Register allocation based approach
  - Introduce routing to respect TLC or RLC, while mapping the basic blocks



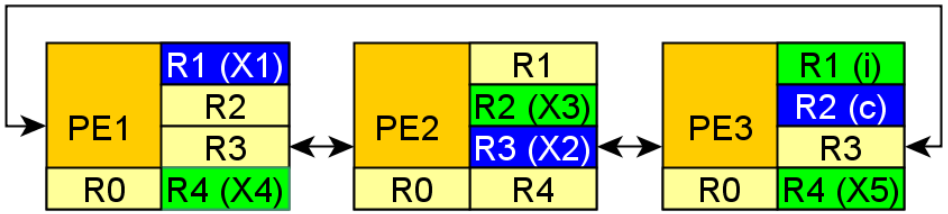

```
a = m[i] * X1;
b = n[i] * X2;
c = b + a;
```

BB_3



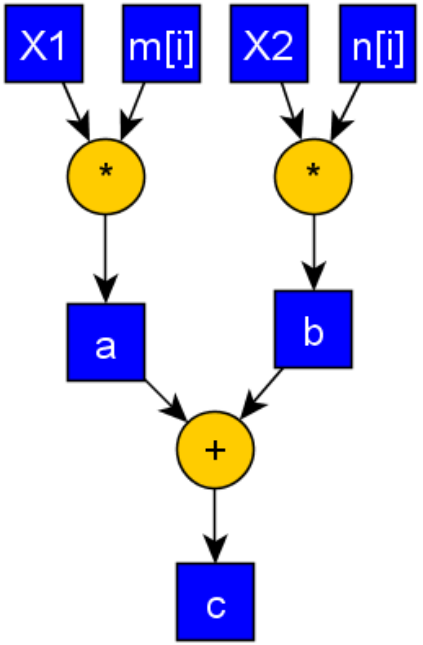
Proposed Method

- Register allocation based approach
  - Introduce routing to respect TLC or RLC, while mapping the basic blocks



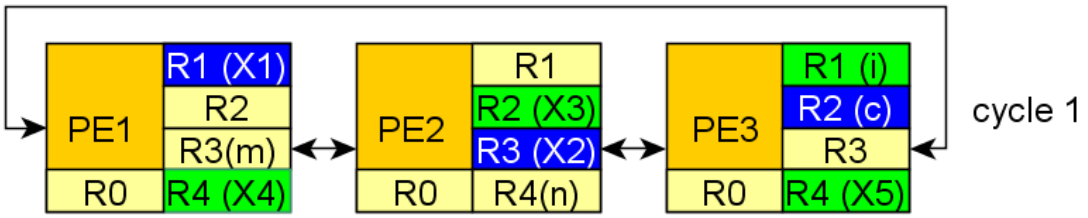

```
a = m[i] * X1;
b = n[i] * X2;
c = b + a;
```

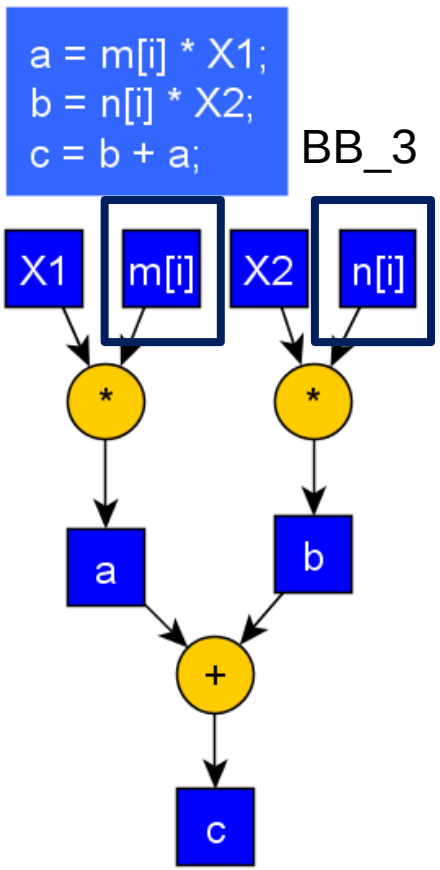
BB_3



Proposed Method

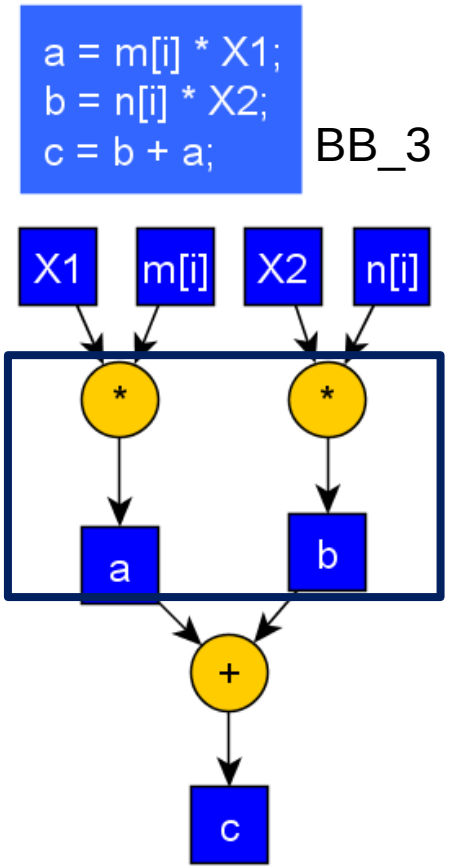
- Register allocation based approach
  - Introduce routing to respect TLC or RLC, while mapping the basic blocks

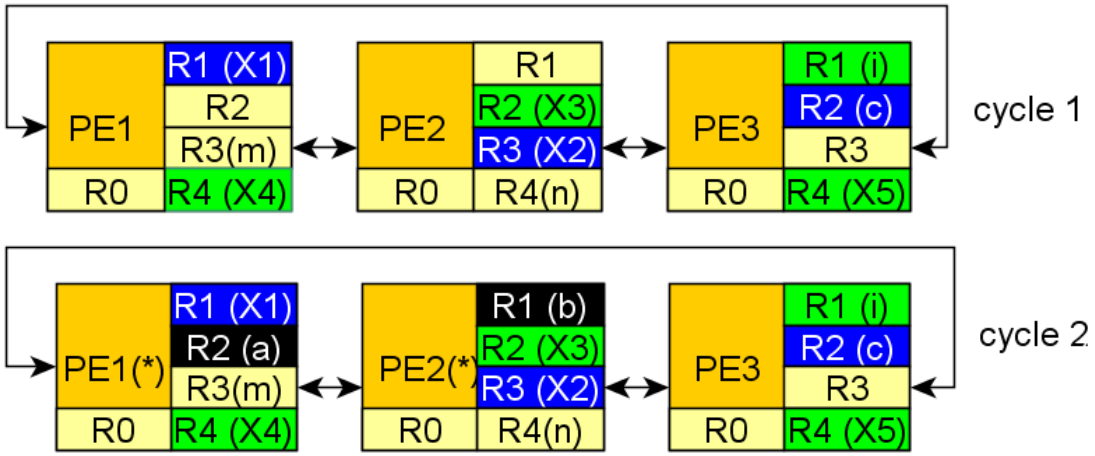




Proposed Method

- Register allocation based approach
  - Introduce routing to respect TLC or RLC, while mapping the basic blocks



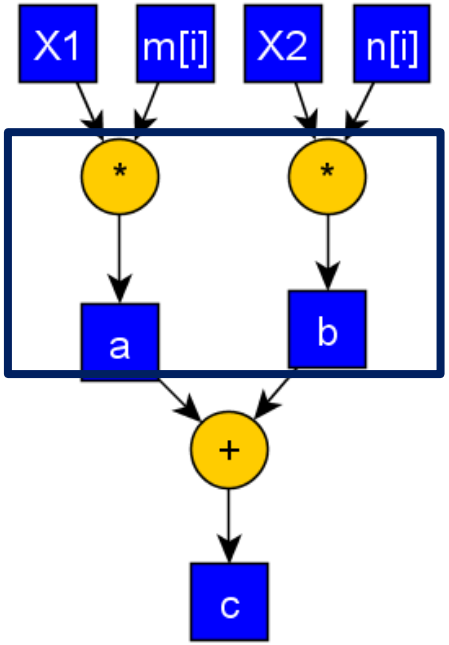


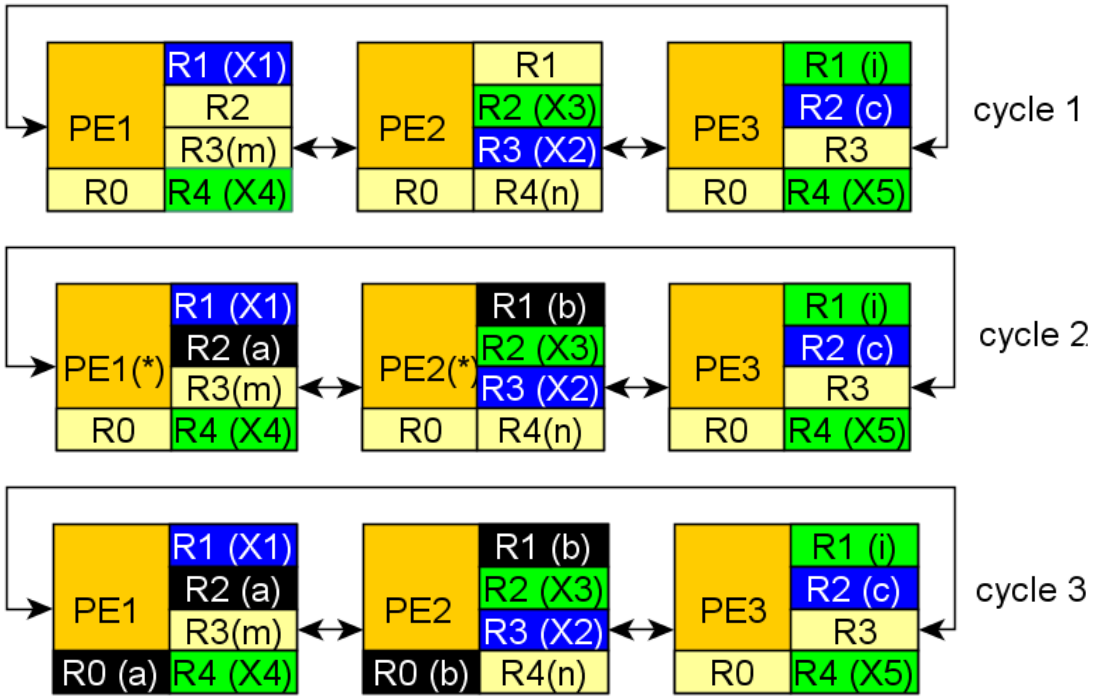
Proposed Method

- Register allocation based approach
  - Introduce routing to respect TLC or RLC, while mapping the basic blocks


```
a = m[i] * X1;
b = n[i] * X2;
c = b + a;
```

BB_3



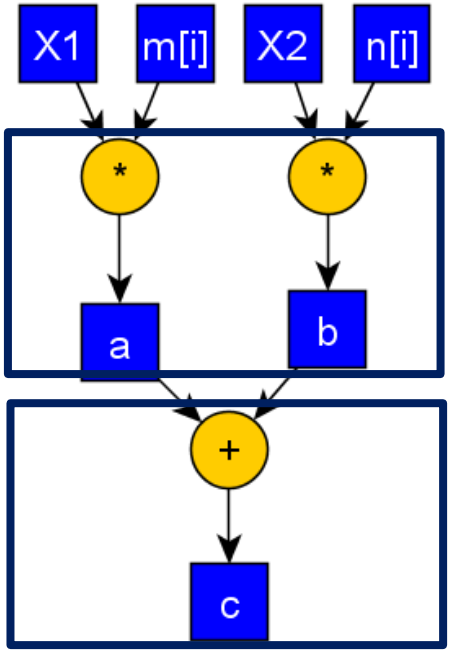


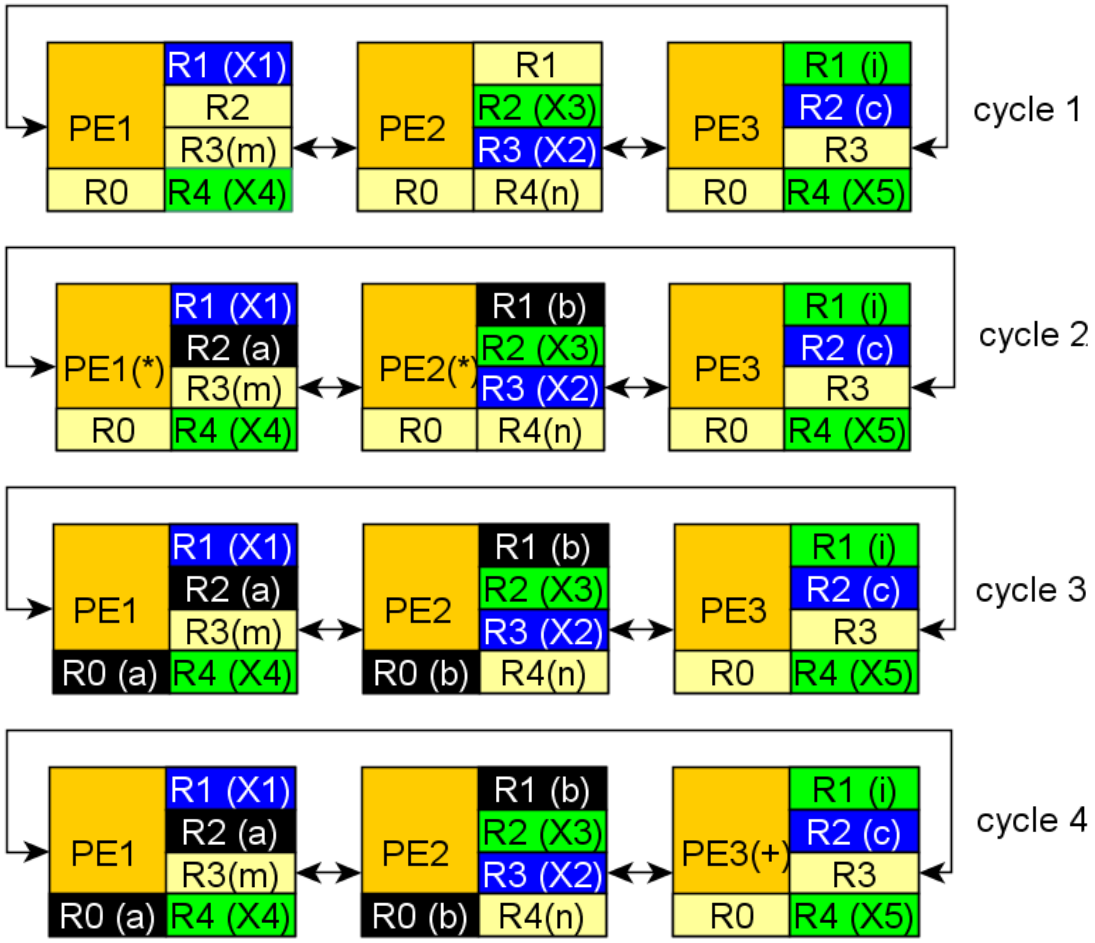
Proposed Method

- Register allocation based approach
  - Introduce routing to respect TLC or RLC, while mapping the basic blocks


```
a = m[i] * X1;
b = n[i] * X2;
c = b + a;
```

BB_3



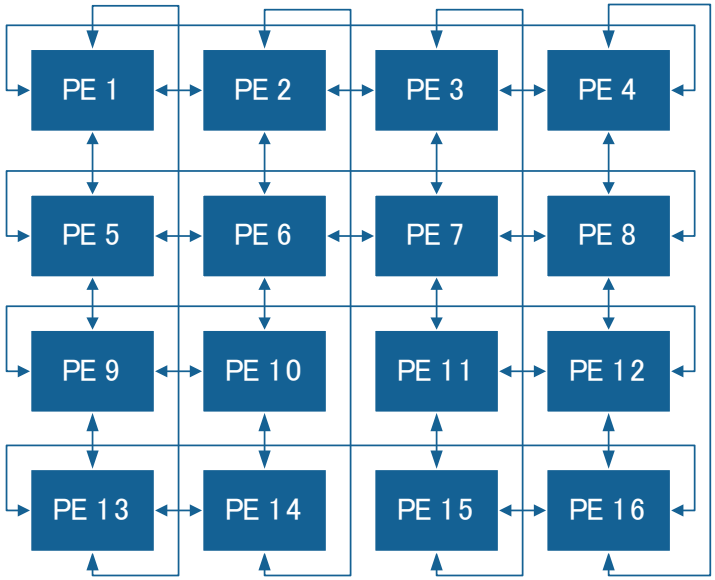


Outline

- Introduction & Background
- Contribution
  - Problem formulation
  - Proposed method
  - Proposed architecture
- Experimental results
- Conclusion

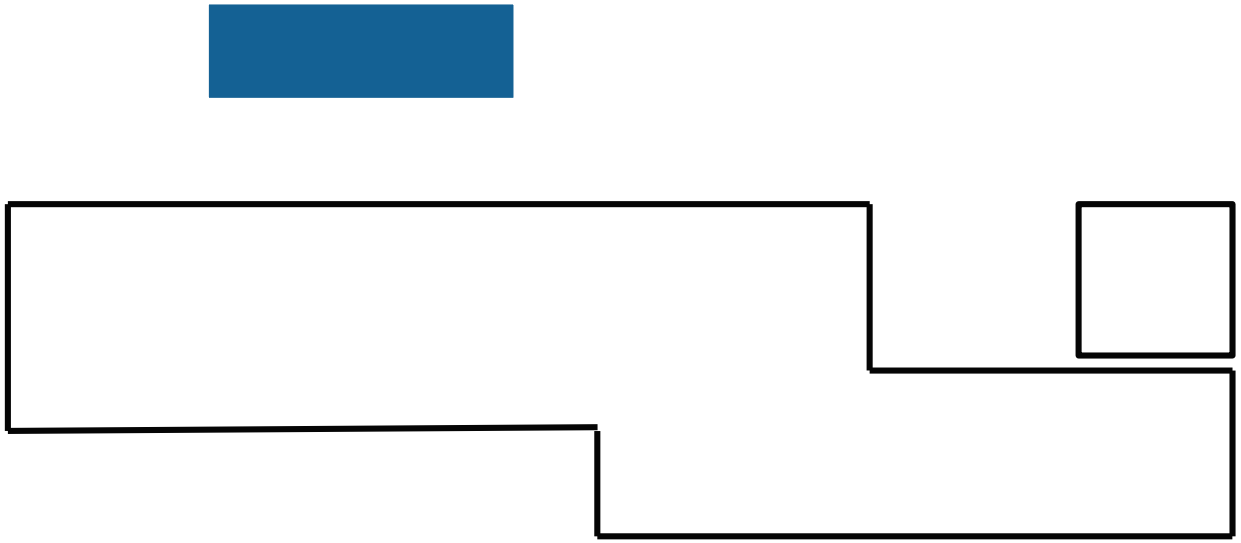
Proposed architecture: the grid

- 4x4 PE array
- Each PE is connected with 4 neighbours
- Supports MIMD execution model



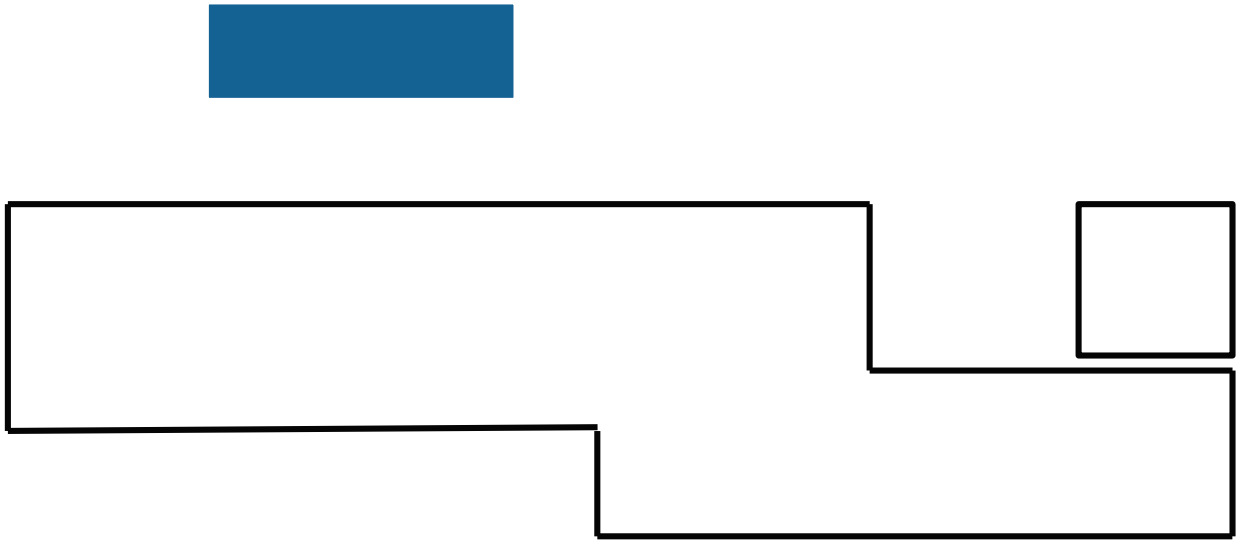
4x4 CGRA with mesh torus interconnect network

Proposed architecture: the PE



- 32 bit ALU, shifter and 16x16 bit multiplier
- Optional load-store unit (LSU)
- Regular register file (RRF) and output register (OR) to store temporary variables
- Constant register file (CRF) for constants
- Controller for selecting the address of next instruction

Proposed architecture: the PE



- Cond Register is one bit register, contains:
  - 0 => true conditions
  - 1 => false conditions
- The boolean OR of all the control bits from all the PEs gives the indication of false condition execution
  - In next cycle, offset address of the basic block from false path is fetched.
- Stall signals indicate memory access congestions

Outline

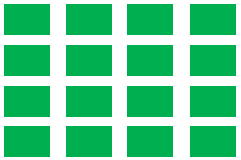
- Introduction & Background
- Contribution
  - Problem formulation
  - Proposed method
  - Proposed architecture
- Experimental results
- Conclusion

Experimental results

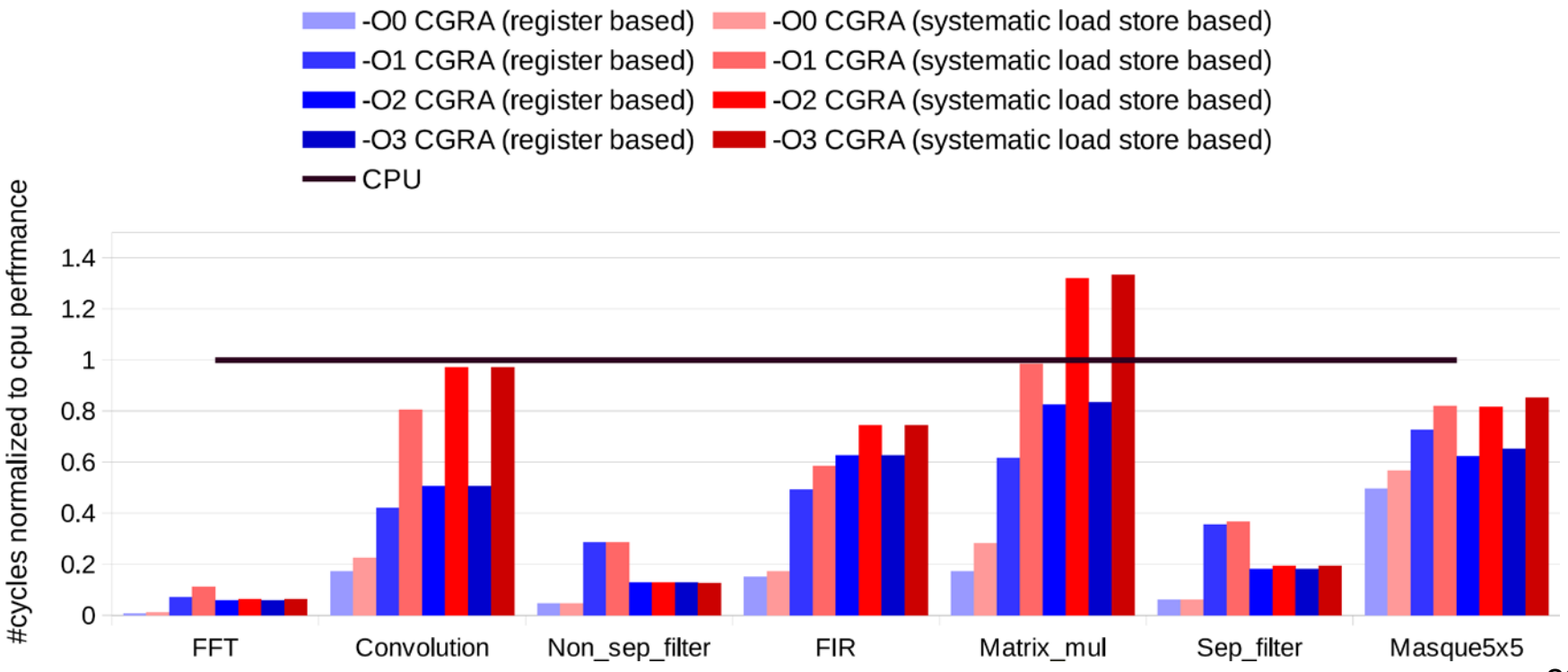
- Experimental setup
  - Fully automated mapping through a software tool implemented by using Java and Eclipse Modeling Framework (EMF). GCC 4.8 is used to generate CDFGs from applications described in C language
  - Performance comparison of the proposed register based approach and the basic systematic load-store based approach with respect to CPU
    - A cycle accurate model of the CGRA architecture is implemented in C++
    - OR1K is the chosen CPU. Instruction set simulations are done with the toolchain provided by OVPsim

Experimental results

- Performance comparison of the proposed register based approach and the basic systematic load-store based approach in a CGRA with infinite memory bandwidth with respect to CPU

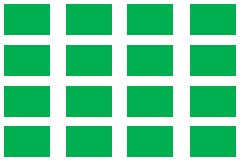


Memory Access

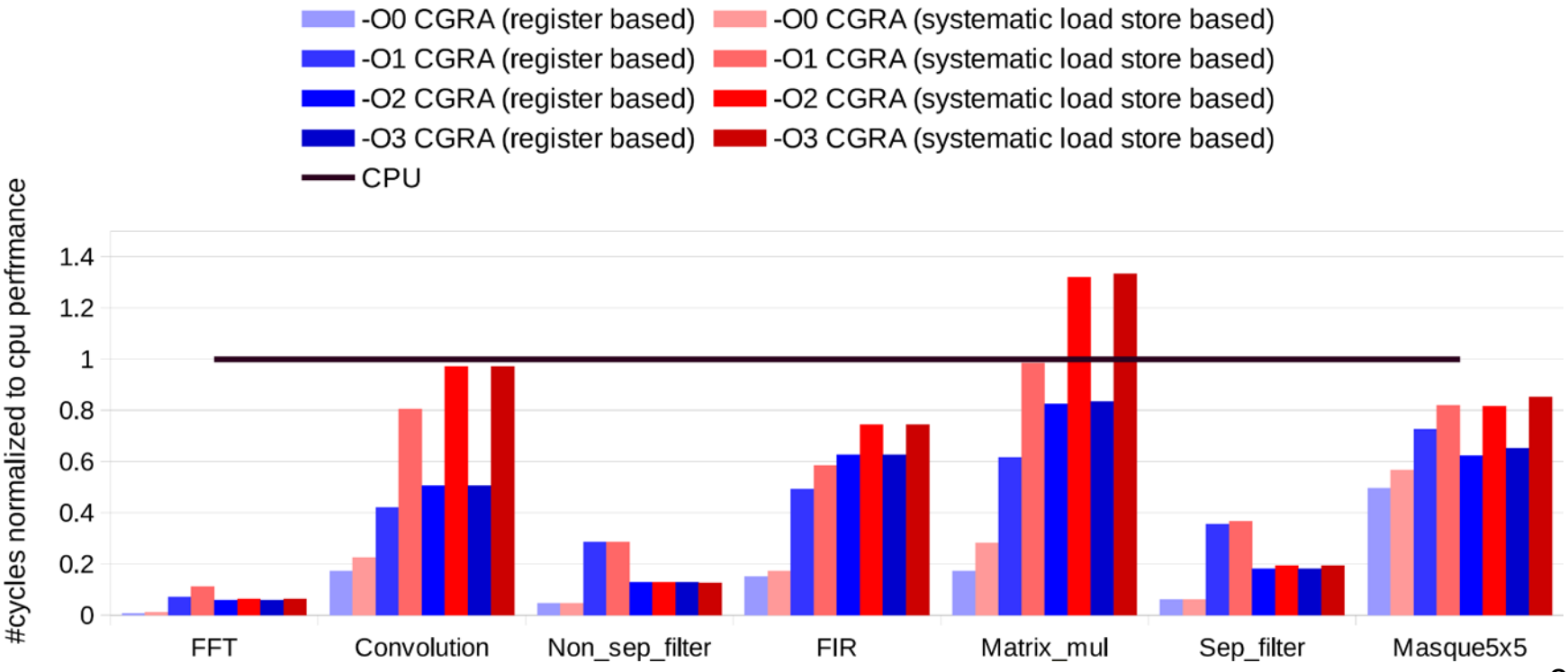


Experimental results

- The speed-up decreases with the compilation option
  - very aggressive optimizations on loops are performed on the cpu with higher optimization options.

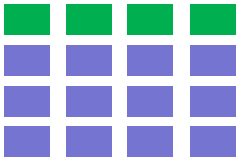


Memory Access

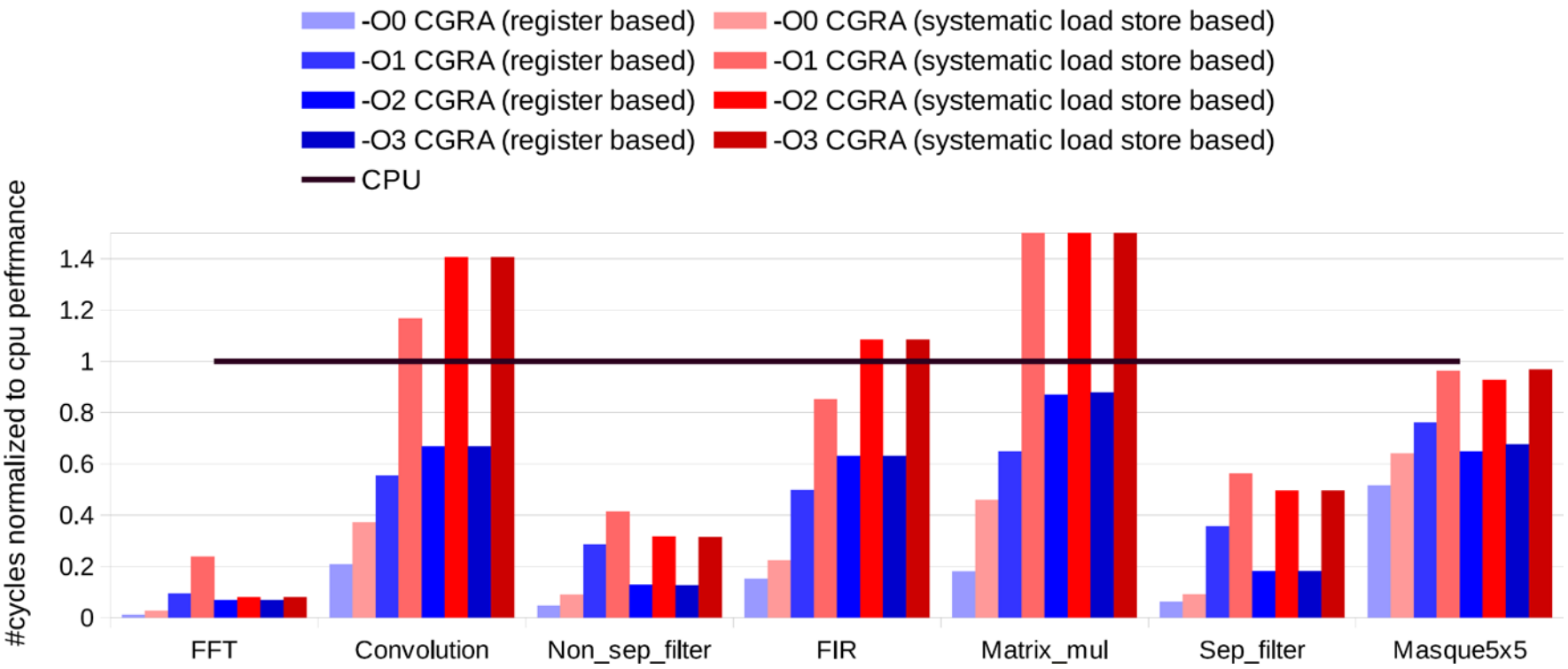


Experimental results

- Performance comparison of the proposed register based approach and the basic systematic load-store based approach in a CGRA with limited bandwidth (top row LSU) with respect to CPU

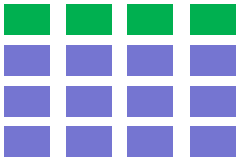


Memory Access

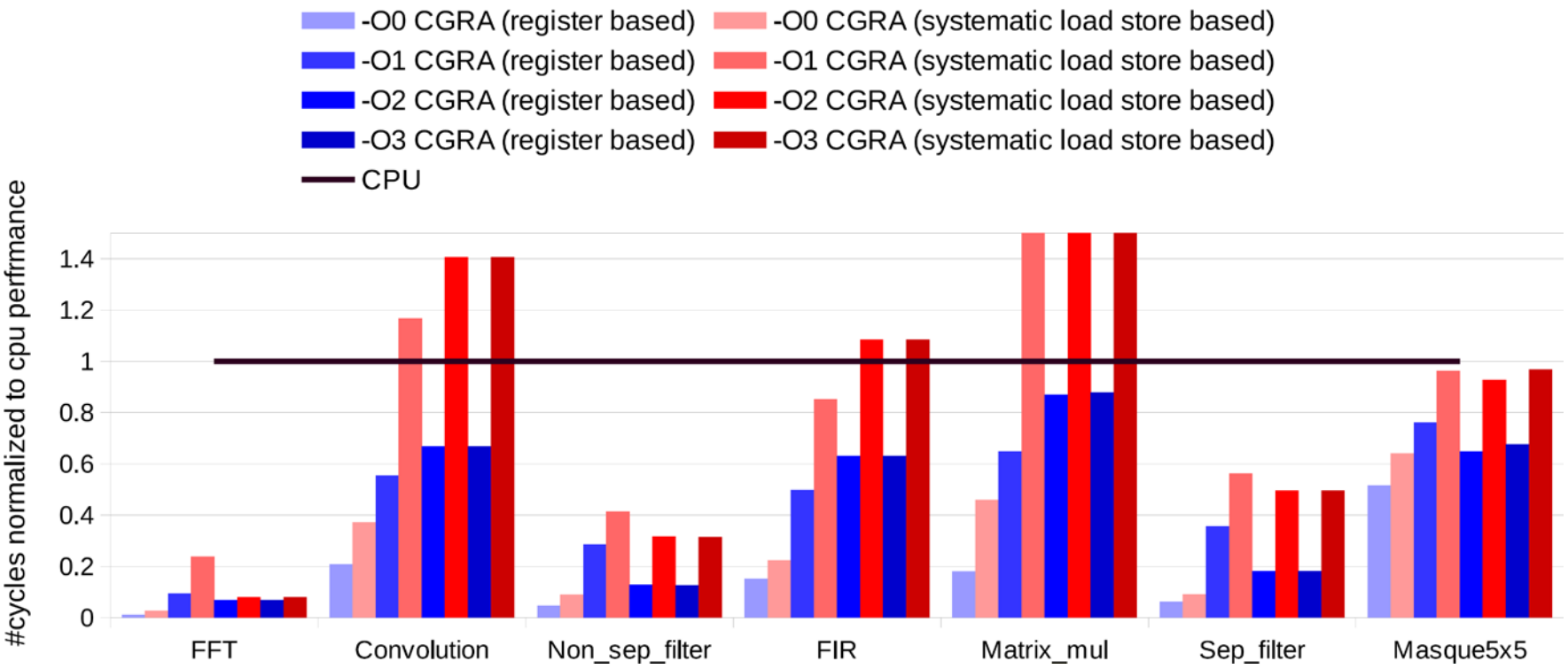


Experimental results

- Register based approach performed best all the time with an average speed up of 21x compared to that of CPU



Memory Access



Experimental results

- Area and energy consumption comparisons
  - Post synthesis implementation of CGRA and CPU in STMicroelectronics 28nm UTBB FD-SOI technology.
  - Synopsys Design Compiler => synthesis, and PrimePower => power analysis
    - @ 0.6V supply voltage at the temperature of 25° C.
      - In this operating point, the processor can achieve 45 MHz, with a power density of 3.54 W/MHz, while a 4x4 CGRA achieves 100 MHz and a power density of 1.78 W/MHz

Experimental results

- Area comparison of different size of CGRA with CPU
  - 4x4 CGRA is used for our experiments, which comes with an 2.7x area overhead compared to CPU

CPU	Combinational (%)	Sequential (%)	Total overhead compared to CPU
CGRA 2x2	56.47	43.53	0.8x
CGRA 3x3	52.07	47.93	1.6x
CGRA 4x4	49.62	50.38	2.7x

Experimental results

- Energy consumption comparison for several kernel execution in a 4x4 CGRA and a CPU
  - An average of 50x gain is achieved in CGRA compared to that of CPU

	Opt -O0	opt -O1	opt -O2	opt -O3	
kernels	CGRA (register based)	CGRA (register based)	CGRA (register based)	CGRA (register based)	CPU
FFT	238.86x	28.05x	32.33x	32.33x	1
Convolution	11.54x	4.73x	3.93x	3.93x	1
Non_sep_filter	42.34x	6.96x	15.47x	15.6x	1
FIR	13.18x	4.04x	3.18x	3.18x	1
Matrix_mul	11x	4x	3x	3x	1
Sep_filter	32.74x	5.59x	10.96x	10.97x	1
Masque5x5	3.33x	2x	2x	2x	1

Experimental results

- Energy consumption (µJ) for several kernel execution using register based approach and the state of the art predication techniques
  - an average of 1.44x and 1.6x energy improvement is achieved compared to partial and full predication respectively

	Gain compared to	
Kernels	**Partial pred**	**Full pred**
FFT	2.43x	2.71x
Convolution	1.11x	1.17x
Non_sep_filter	1.39x	1.47x
FIR	1.26x	1.32x
Matrix_mul	1.5x	2x
Sep_filter	1.29x	1.45x
Masque5x5	1x	1.33x

Outline

- Introduction & Background
- Contribution
  - Problem formulation
  - Proposed method
  - Proposed architecture
- Experimental results
- Conclusion

Conclusion

- CGRA architecture and mapping approach is presented to implement full control flow onto a CGRA in an ultra-low-power environment
- The proposed approach overcomes limitations and inefficiencies of state of the art predications methods, achieving 1.44x and 1.6x energy gain over partial and full predication techniques respectively
- The proposed approach achieves average speed-up of 50x and an energy improvement of 21x with respect to an embedded CPU with an area overhead of 2.7x

THANK YOU

References

- K. Han, J. Ahn, and K. Choi. Power-efficient predication techniques for acceleration of control flow execution on cgra. ACM Trans. Archit. Code Optim., 10(2):8:1–8:25, May 2013.
- K. Han, J. K. Paek, and K. Choi. Acceleration of control flow on cgra using advanced predicated execution. In Field-Programmable Technology (FPT), 2010 International Conference on, pages 429–432, Dec 2010..