Towards Efficient Data Parallelism on Spatial CGRA via Constraint Satisfaction and Graph Coloring

Authors: Yuan Dai, Xuchen Gao*, et al.

Presenter: Xuchen Gao
Institution: Fudan University, China
Email: xcgao22@m.fudan.edu.cn

Contents

- Introduction
- Problem Formulation
- Proposed Methodology
- Experiment
- Conclusion

Contents

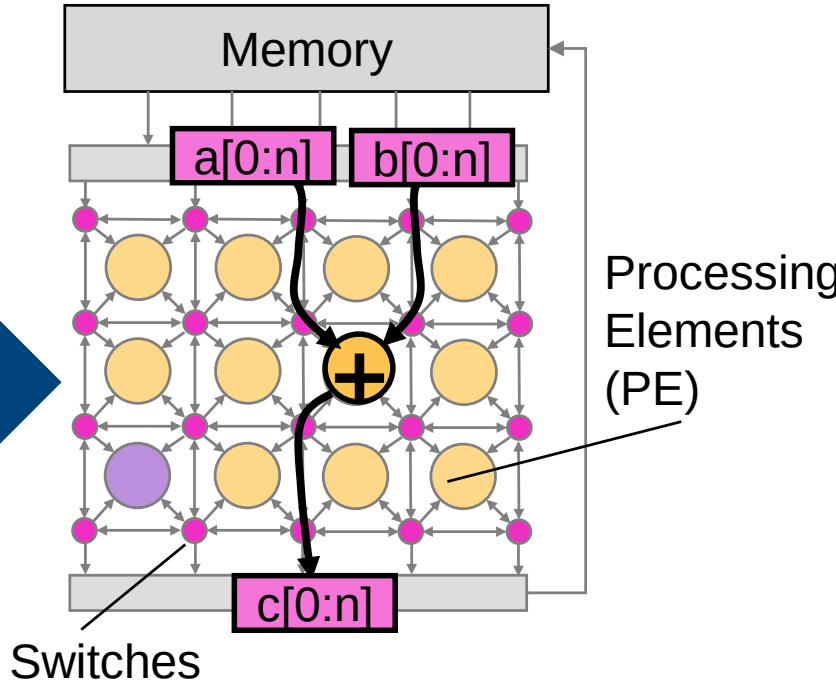
Background

- The Workflow of Coarse-Grained Reconfigurable Architecture (CGRA)

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

application code

DFG Generation
Scheduling
Mapping
.....



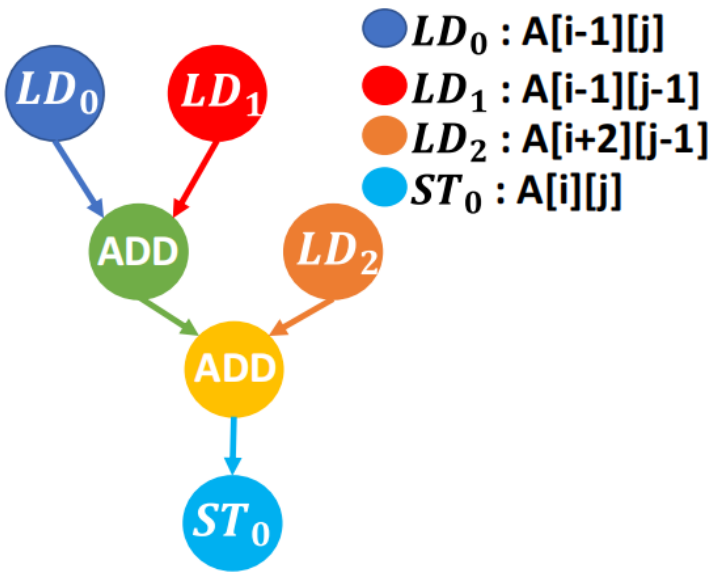
An example of CGRA

Motivation

- **Motivation Example**

```
#define N 32
int A[N][N];
void example ( ) {
      for( int i = 1; i < N ; i ++ )
            for(int j = 1; j < N ; j ++ )
                  A[i][j] = A[i-1][j] + A[i-1][j-1]
                  + A[i+2][j-1];
}
```

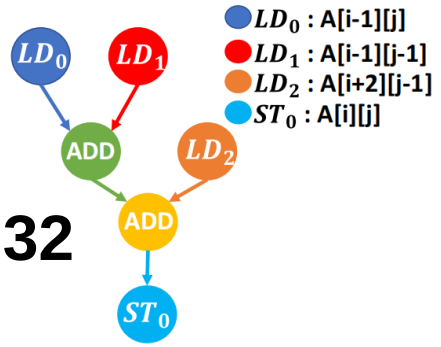
Motivating example code

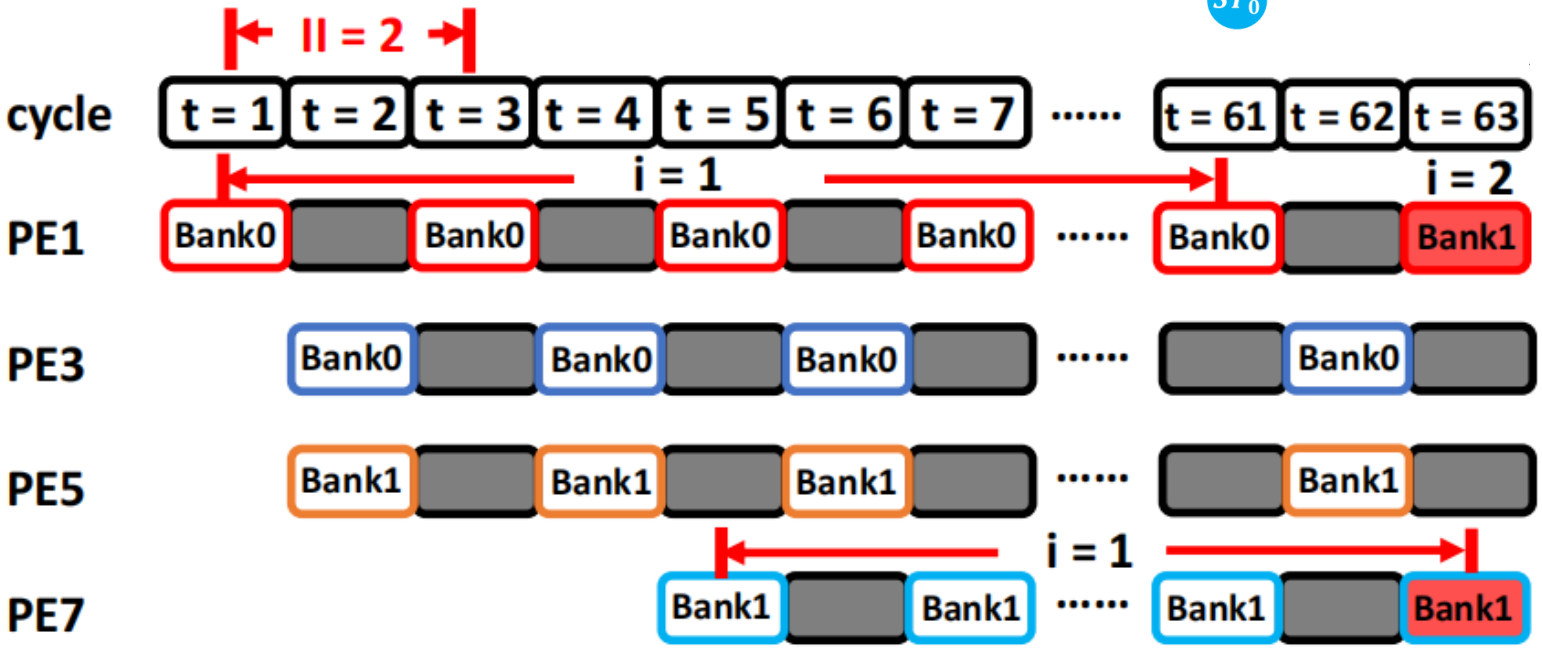


DFG of example code

Motivation

- **Motivation Example**
  - **Invalid scheme with N = 2 and B = 32**





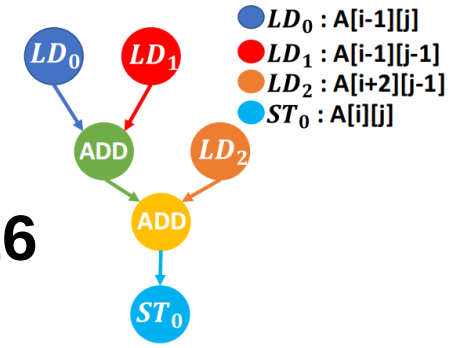
Bank0	Bank1
A[0][0 ~ 31]	A[1][0 ~ 31]
A[2][0 ~ 31]	A[3][0 ~ 31]
……	……

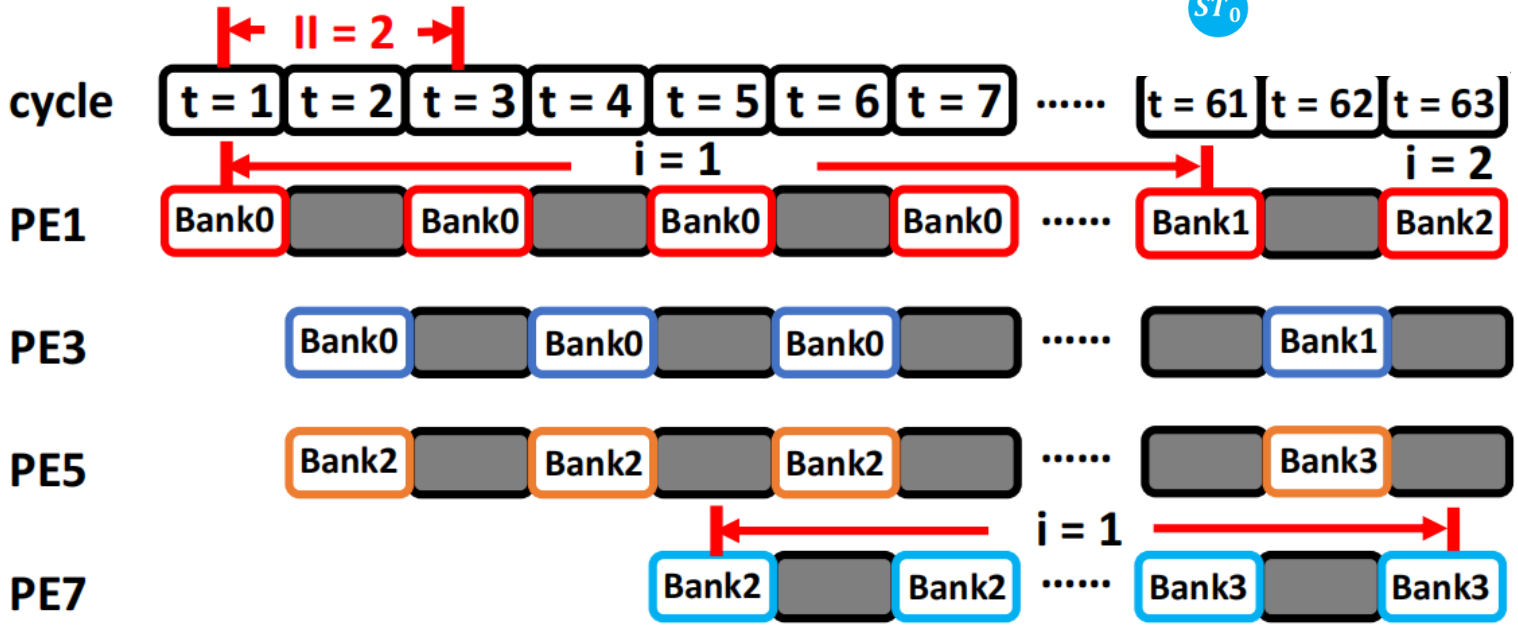
Bank index = $\left\lfloor \frac{x}{32} \right\rfloor \% 2$

x: the address of data

Motivation

- **Motivation Example**
  - **Valid scheme with N = 4 and B = 16**





Bank0	Bank1	Bank2	Bank3
A[0][0 ~ 15] A[2][0 ~ 15] ……	A[0][16 ~ 31] A[2][16 ~ 31] ……	A[1][0 ~ 15] A[3][0 ~ 15] ……	A[1][16 ~ 31] A[3][16 ~ 31] ……

Bank index = $\left\lfloor \frac{x}{16} \right\rfloor \% 4$

x: the address of data

Contents

- Introduction
- Problem Formulation
- Proposed Methodology
- Experiment
- Conclusion

Definitions

- Affine Address Access

$$\phi(\vec{i}) = [AS_0\ AS_1 \ldots\ AS_{l-1}] \begin{bmatrix} i_0 \\ i_1 \\ \ldots \\ i_{l-1} \end{bmatrix} + BA$$

- Access Pattern

$$P_A = \{\phi_1(\vec{i_1}), \ldots, \phi_m(\vec{i_m})\}$$

- Control Step Access Pattern

$$P_A^n = \{\phi_1(\vec{i_1}), \ldots, \phi_q(\vec{i_q})\}$$

q ($q \leq m$) memory accesses at n^{th} cycle

Definitions

- Memory Partition

$$f(\phi) = \lfloor \frac{\phi}{B} \rfloor \% N; \quad g(\phi) = \lfloor \frac{\phi}{N \cdot B} \rfloor \cdot B + \phi \% B$$

$f(\phi)$: bank function $g(\phi)$: offset function

- Iteration Distance

$$iter_d^{\alpha\beta} = \frac{data\_delay}{Mapped\_II}$$

Target

- ① Access II

$$AII = MAX(Con_i), 1 \le i \le N$$

- ② Iteration Count Vector

$$\forall \vec{i_\alpha}, \vec{i_\beta} \in M,\ \forall \phi_\alpha(\vec{i_\alpha}), \phi_\beta(\vec{i_\beta}) \in P_A^n, \alpha \neq \beta$$

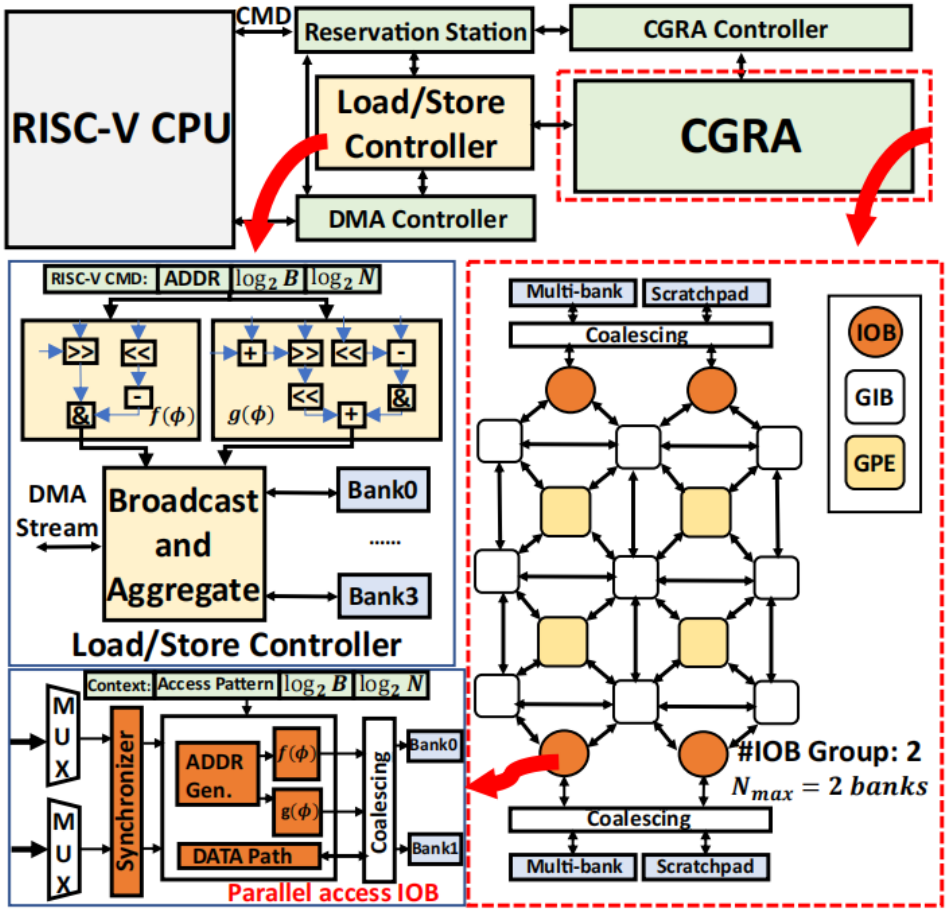
$$(\vec{i_\alpha} - \vec{i_\beta}) \cdot (1, ic_0, \ldots, \prod_{t=0}^{l-2} ic_t) = iter_d^{\alpha\beta},\ f(\phi_\alpha(\vec{i_\alpha})) \neq f(\phi_\beta(\vec{i_\beta}))$$

- Given an l-level loop in the iteration domain M with m affine memory accesses on the same array $P_A = \{\phi_1(\vec{i_1}), \ldots, \phi_m(\vec{i_m})\}$
  find the N and B such that:
  ※ Minimize AII in ① & Hold the equation ②

Contents

- Introduction
- Problem Formulation
- Proposed Methodology
- Experiment
- Conclusion

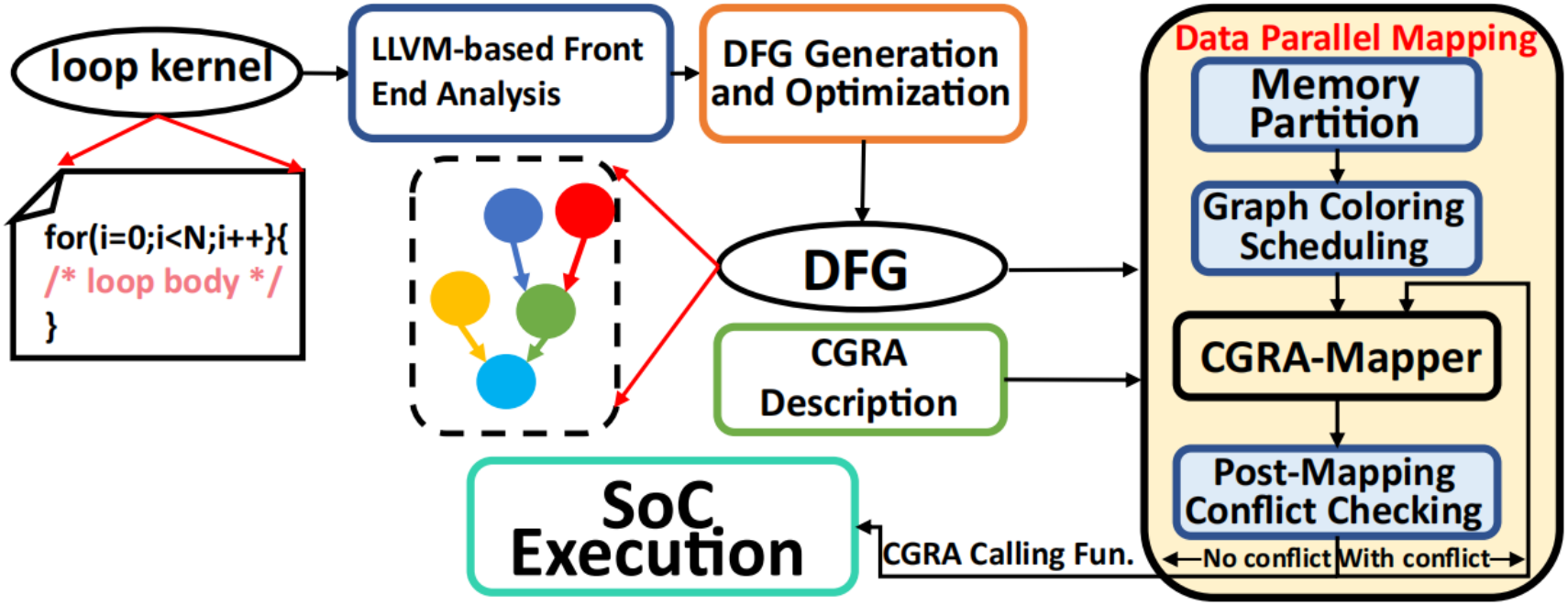
End-to-End Framework: Hardware



- Enhanced IOB to access different banks during execution
- Specific mechanism for loading/storing data from/to external memory

The SoC supports parallel memory access

End-to-End Framework: Software



Software flow with the proposed memory partition algorithm

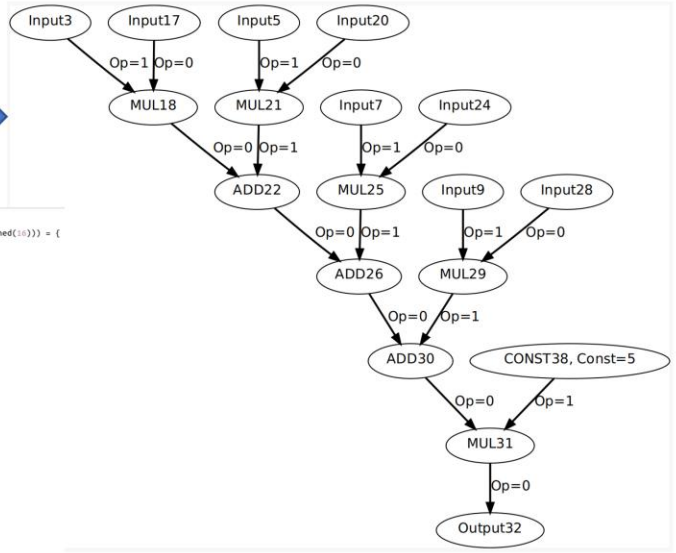
End-to-End Framework: Software

```
#define _PB_NI 32
#define _PB_NJ 32
#define _PB_NK 32
int C[_PB_NI][_PB_NI];
int A[_PB_NI][_PB_NI];
int B[_PB_NI][_PB_NI];
//kernel 23
void kernel() {
  #ifdef CGRA_COMPILER
    loop_begin();   ①: added pragma for front-end tool
  #endif
    int alpha = 5;
    for (int i = 0; i < _PB_NI; i++)
      for (int j = 0; j < _PB_NJ; j++)
      {
        for (int k = 0; k < _PB_NK; k= k + 4)
          C[i][j] = alpha * A[i][k] * B[k][j] + A[i][k+1] * B[k+1][j]+ A[i][k+2] * B[k+2][j]+ A[i][k+3] * B[k+3][j]);
      }
  #ifdef CGRA_COMPILER
    loop_end();   ①
  #endif
}
```

②: A[][] and B[][] have multiple memory accesses

Gemm Source Code

LLVM-based front-end tool



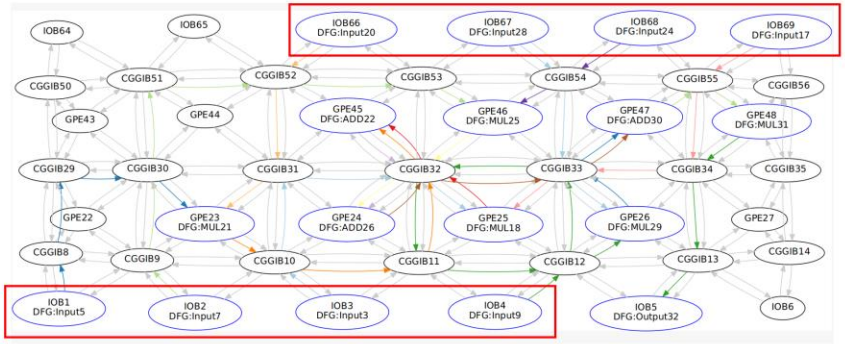
Generated DFG

```
Succeed to map DFG to ADG!<<<<<<
Final II = 1 !<<<<<<
*************** Memory partition and schedule results ***************
Array Name: A; N: 4; B: 1
 >> Step 0<<
Input3 Input5 Input7 Input9

Array Name: B; N: 4; B: 32
 >> Step 0<<
Input17 Input20 Input24 Input28

PE usage: 0.666667
IO usage: 0.75
DFG latency: 10
```

Mapper Report



Mapping Result

```
1 void cgra_execute(void** din_addr, void** dout_addr)
2 {
3       static unsigned short cin[80][8] __attribute__((aligned(16))) = {
4               {0x0000, 0x1000, 0x0000},
5               {0x0200, 0x0010, 0x0009},
6               {0x0000, 0x0010, 0x000a},
7               {0x0420, 0x0000, 0x000b},
8               {0x0002, 0x1000, 0x0010},
9               {0x0200, 0x0010, 0x0011},
10              {0x0000, 0x0010, 0x0012},
11              {0x0420, 0x0000, 0x0013},
12              {0x0001, 0x1000, 0x0018},
13              {0x0200, 0x0010, 0x0019},
14              {0x0000, 0x0010, 0x001a},
15              {0x0420, 0x0000, 0x001b},
16              {0x0003, 0x1000, 0x0020},
17              {0x0200, 0x0010, 0x0021},
18              {0x0000, 0x0010, 0x0022},
19              {0x0420, 0x0000, 0x0023},
20              {0x1000, 0x1000, 0x0028},
21              {0x0010, 0x0010, 0x0029},
22              {0x0000, 0x0910, 0x002a},
23              {0x0000, 0x0200, 0x002b},
24              {0x0000, 0x0000, 0x0040},
25              {0x0000, 0x0000, 0x0048},
26              {0xa008, 0x0000, 0x0050},
27              {0x6000, 0x0001, 0x0058},
28              {0x0100, 0x0000, 0x0060},
29              {0x0002, 0x0000, 0x0068},
30              {0xe023, 0x0000, 0x00b9},
31              {0x2021, 0x0001, 0x00c1},
32              {0xe003, 0x0000, 0x00c9},
33              {0x5843, 0x0000, 0x00d1},
34              {0x0000, 0x0000, 0x00e8},
35              {0x0000, 0x0180, 0x00f0},
36              {0x0002, 0x0030, 0x00f8},
37              {0x0000, 0x0000, 0x0100},
38              {0x80b1, 0x0000, 0x0108},
39              {0x0000, 0x0001, 0x0110},
40              {0x9021, 0x0000, 0x0169},
41              {0xe043, 0x0000, 0x0171},
42              {0x4a01, 0x0000, 0x0179},
43              {0x0005, 0x0008, 0x0180},
44              {0x1803, 0x0000, 0x0181},
45              {0x0000, 0x0001, 0x0190},
46              {0x0000, 0x0001, 0x01a0},
47              {0x2000, 0x0000, 0x01b0},
48              {0x0000, 0x0000, 0x01b8},
49              {0x1000, 0x1000, 0x0210},
50              {0x7e10, 0x0010, 0x0211},
51              {0x0000, 0x0010, 0x0212},
52              {0x8568, 0x0000, 0x0213},
53              {0x1060, 0x1000, 0x0218},
54              {0x7e10, 0x0010, 0x0219},
55              {0x0000, 0x0010, 0x021a},
56              {0x8560, 0x0000, 0x021b},
57              {0x1040, 0x1000, 0x0220},
58              {0x7e10, 0x0010, 0x0221},
59              {0x0000, 0x0010, 0x0222},
60              {0x8560, 0x0000, 0x0223},
61              {0x1020, 0x1000, 0x0228},
62              {0x7e10, 0x0010, 0x0229},
63              {0x0000, 0x0010, 0x022a},
64              {0x8560, 0x0000, 0x022b},
65      };
66
67      load_cfg((void*)cin, 0xc000, 384, 0, 0);
68      load_data(din_addr[0], 0x0, 3984, 1, 0, 0, 2, 0);
69      load_data(din_addr[0], 0x1000, 3984, 1, 0, 0, 2, 0);
70      load_data(din_addr[0], 0x2000, 3984, 1, 0, 0, 2, 0);
71      load_data(din_addr[0], 0x3000, 3984, 0, 0, 0, 2, 0);
72      load_data(din_addr[1], 0x8000, 512, 1, 0, 0, 2, 5);
73      load_data(din_addr[1], 0x9000, 512, 1, 0, 0, 2, 5);
74      load_data(din_addr[1], 0xa000, 512, 1, 0, 0, 2, 5);
75      load_data(din_addr[1], 0xb000, 512, 0, 0, 0, 2, 5);
76      config(0x0, 01, 0, 0);
77      execute(0xffff, 0, 0);
78      store(dout_addr[0], 0x4000, 4096, 0, 0, 0, 0, 0);
79 }
```

CGRA Calling Function

```
*Verdi* : Begin traversing the scopes, layer (0).
*Verdi* : Enable +all dumping.
*Verdi* : End of traversing.
*Verdi* : fsdbDumpon - All FSDB files at 0 ps.
Initialization finished!
CPU add finished!
It takes 30611 cycles for CPU to finish the task.
It takes 2432 cycles for CGRA to finish the task(1).
CGRA comput finished!
Checking results!
Done!
*Verdi* : fsdbDumpoff - All FSDB files at 862,145,500 ps.
```

Simulation Result

CSP formulation

- Constraint Satisfaction Problem(CSP)
  - **Iteration Constraint(pre-mapping)**: Before mapping, all the accesses in the P_A are regarded in the same iteration

$$\vec{i_\alpha} = \vec{i_\beta} = \vec{i};\ \vec{i} \in M$$

  - **Bank Constraint:** $\phi_\alpha(\vec{i_\alpha})$ and $\phi_\beta(\vec{i_\beta})$ access the same bank equation:

$$f(\phi_\alpha(\vec{i_\alpha})) = f(\phi_\beta(\vec{i_\beta})) \Leftrightarrow \lfloor \frac{\phi_\alpha(\vec{i_\alpha})}{B} \rfloor \%N = \lfloor \frac{\phi_\beta(\vec{i_\beta})}{B} \rfloor \%N$$

$$\Leftrightarrow \exists k \in \mathbb{Z},\ s.t.\ \lfloor \frac{\phi_\alpha(\vec{i_\alpha})}{B} \rfloor = \lfloor \frac{\phi_\beta(\vec{i_\beta})}{B} \rfloor + kN$$

$$\exists k \in \mathbb{Z},\ s.t.\ -B < \phi_\alpha(\vec{i_\alpha}) - \phi_\beta(\vec{i_\beta}) - kNB < B$$

CSP formulation

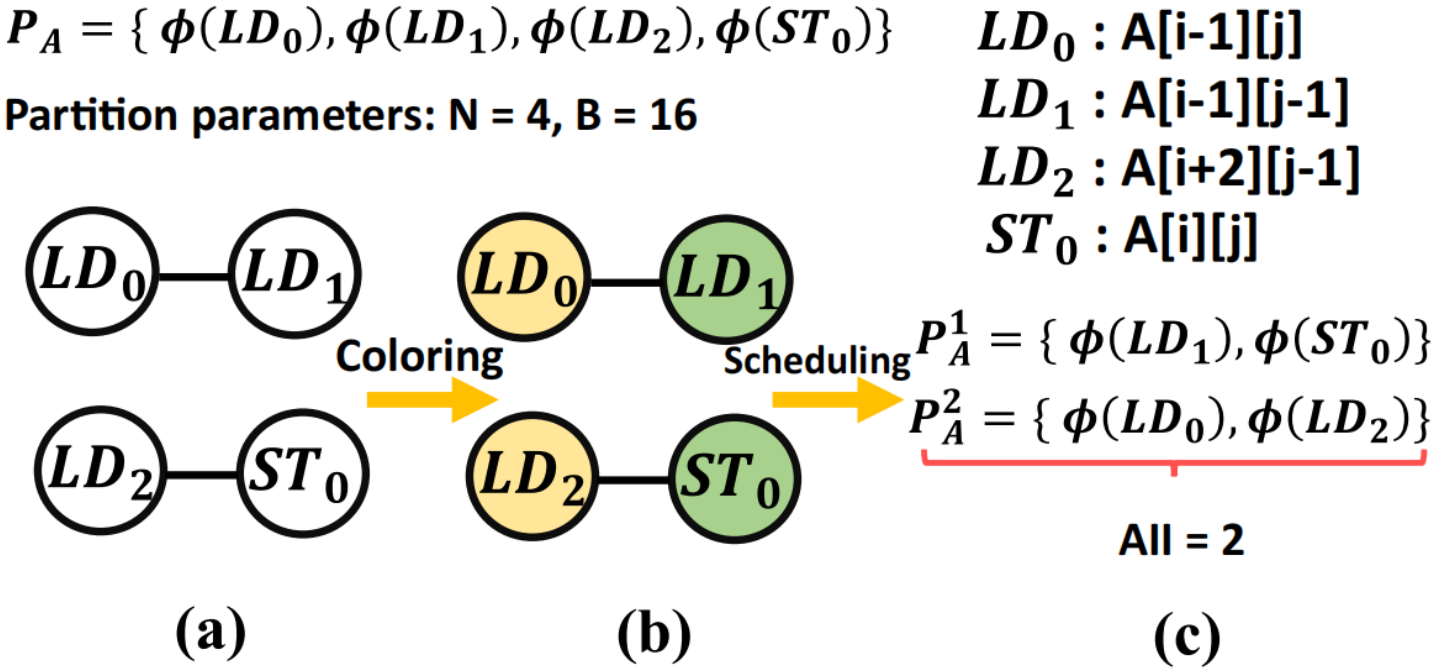
- Constraint Satisfaction Problem(CSP)

$$-B < \left[(AS_0^{\alpha} - AS_0^{\beta}) \ldots (AS_{l-1}^{\alpha} - AS_{l-1}^{\beta}) \right] \begin{bmatrix} i_0 \\ i_1 \\ \ldots \\ i_{l-1} \end{bmatrix} + (BA^{\alpha} - BA^{\beta}) - kNB < B$$

※**CSP problem:** the conflict exists when one solution is found under such the constraints

Graph Coloring-based Scheduling

- When failing to find a conflict-free solution
  - schedule the conflict accesses into different control steps
  - minimizing the *All*



Example of graph coloring and scheduling process

CSP-based Post-mapping Checking

- Iteration Constraint: now $iter_d^{\alpha\beta} \neq 0$

$$(\vec{i_\alpha} - \vec{i_\beta}) \cdot (1, ic_0, ..., \prod_{t=0}^{l-2} ic_t) = iter_d^{\alpha\beta}$$

It can be transformed into a CSP problem, where the conflict exists when one solution is found

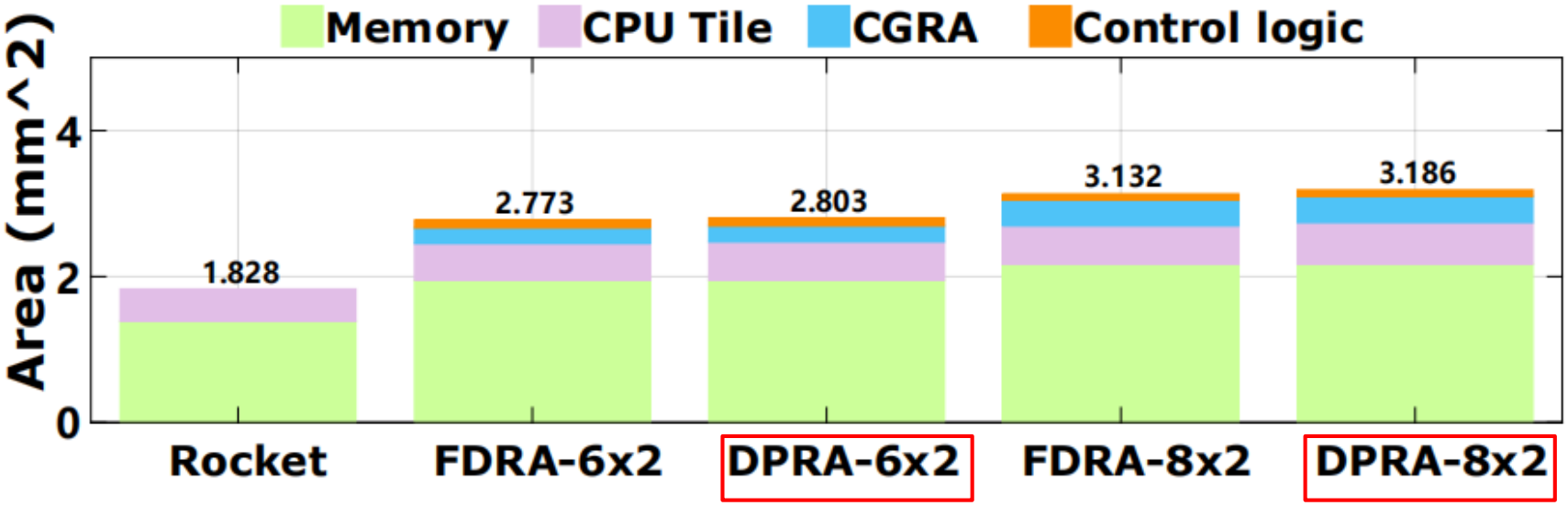
- CSP Check

$$\exists k \in \mathbb{Z},\ s.t.\ -B < \boxed{\phi_\alpha(\vec{i_\alpha}) - \phi_\beta(\vec{i_\beta}) - kNB} < B$$

Contents

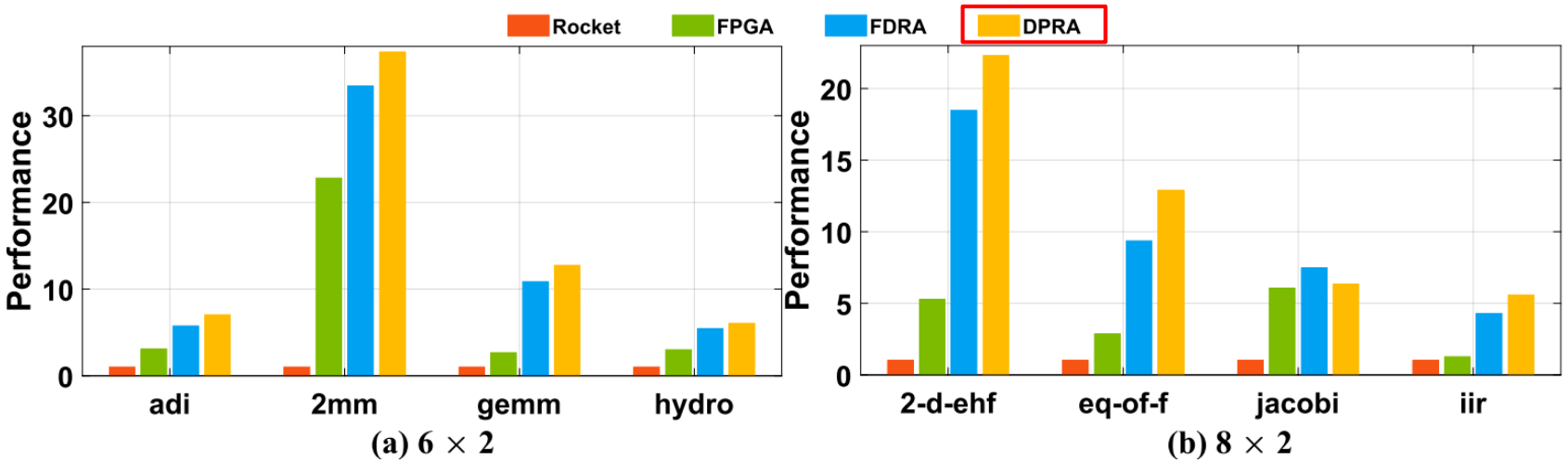
- Introduction
- Problem Formulation
- Proposed Methodology
- Experiment
- Conclusion

System Level Evaluation

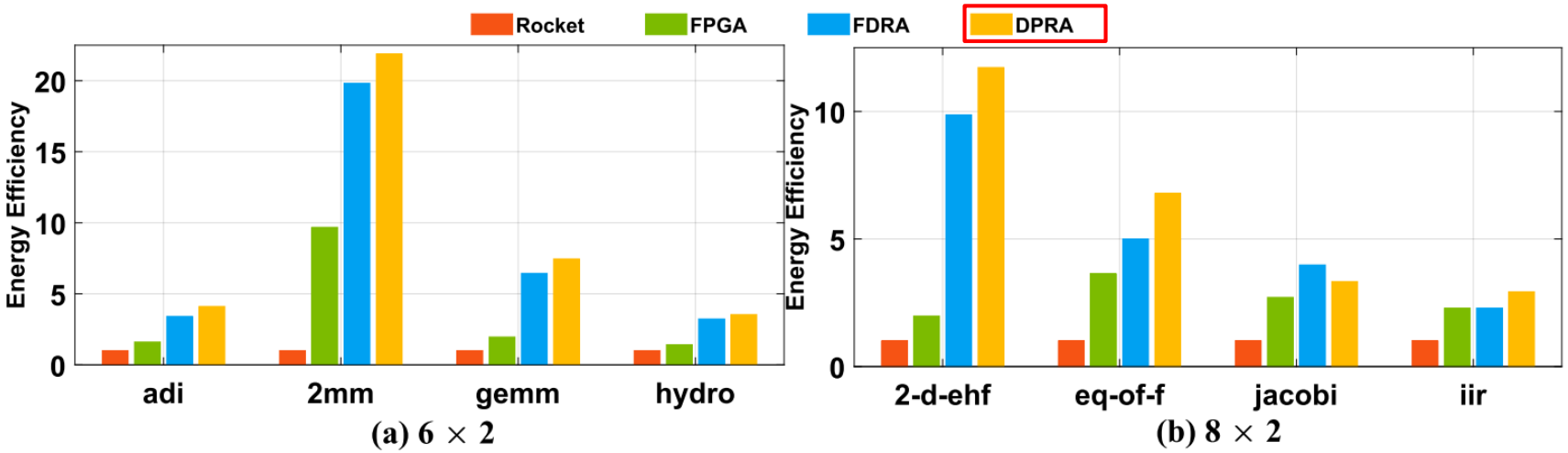


The area of each SoC

System Level Evaluation

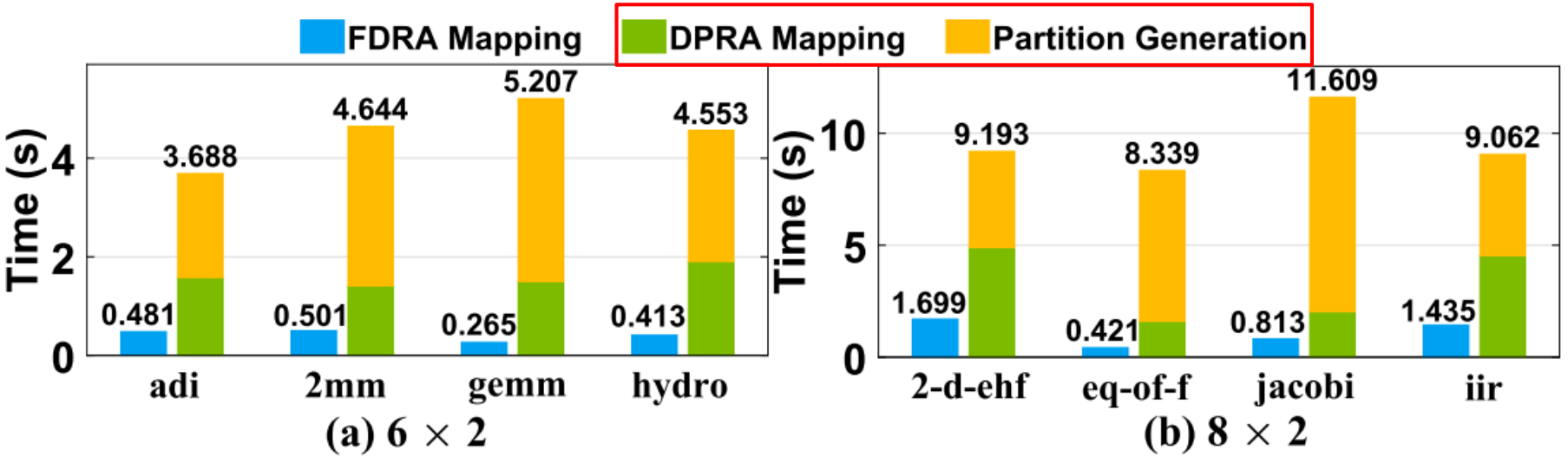


Comparison of performance



Comparison of energy efficiency

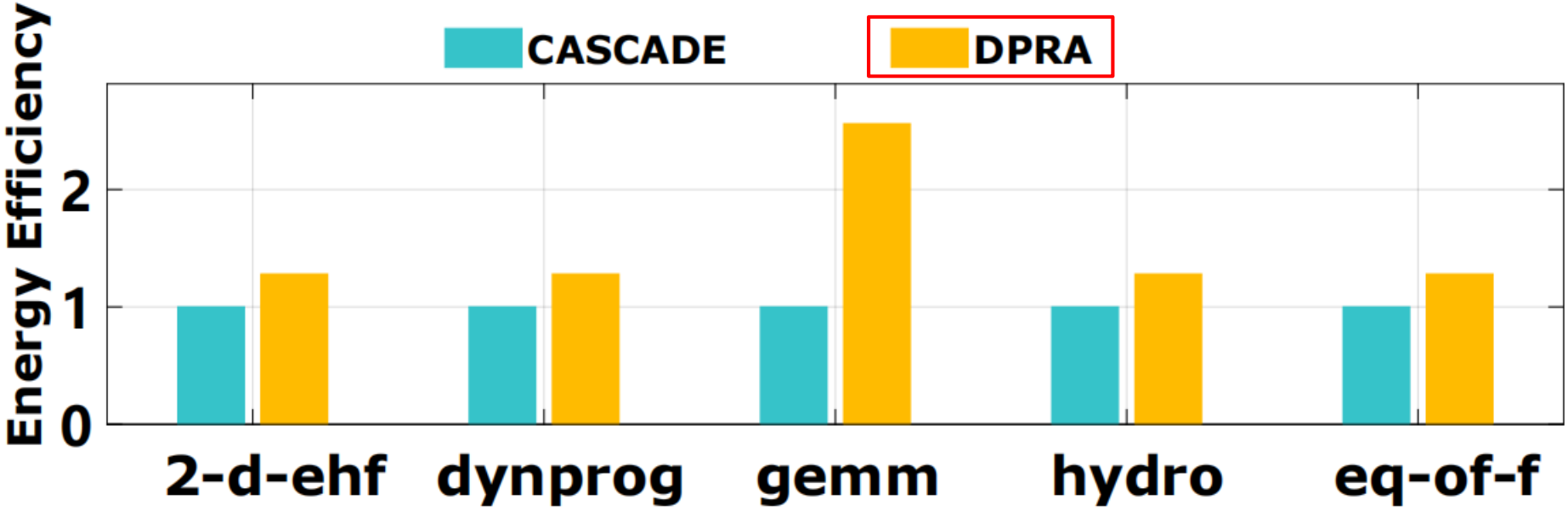
System Level Evaluation



Comparison of compilation time

CGRA Level Evaluation

- $Energy\ efficiency = \frac{Throughput}{Power\ consumption}$
- $Throughput = \frac{Frequency}{Mapped\_II}$



Comparison of energy efficiency with CASCADE

Contents

- Introduction
- Framework of our work
- Analyzing & transforming passes
- SoC runtime configuration of CGRA
- Experiment
- Conclusion

Conclusion

- A CSP-based memory conflict detection for memory partition
- A graph coloring-based approach for memory access scheduling during the partition process to improve performance
- An end-to-end CGRA framework, including the data parallel architecture and compiler with efficient data parallelism support

Thanks

Yuan Dai, Xuchen Gao, Chen Shen, Bingbing Peng,
Wenbo Yin, Wai-Shing Luk*, Lingli Wang*

State Key Laboratory of Integrated Chips and Systems
Fudan University, Shanghai, China
Email: xcgao22@m.fudan.edu.cn, *llwang@fudan.edu.cn