Standalone Nested Loop Acceleration on CGRAs for Signal Processing Applications

Chilankamol Sunny, *Satyajit Das, Kevin Martin, Philippe Coussy*

112004004@smail.iitpkd.ac.in

Jan 19, 2024

SCC Munich, Germany

dasip

The Workshop on Design and Architectures for Signal and Image Processing

OVERVIEW

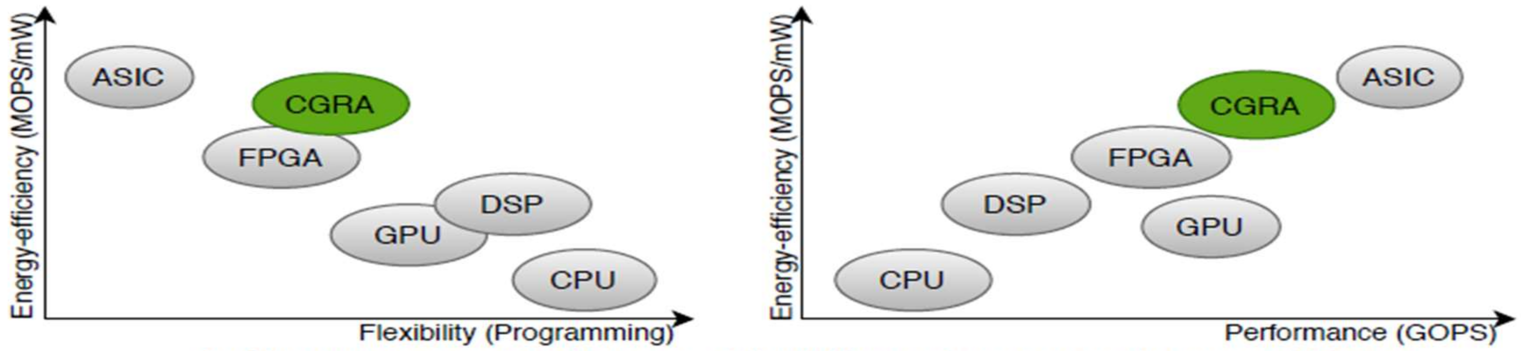
OVERVIEW

- **INTRODUCTION**
- MOTIVATION & BACKGROUND
- PROPOSED APPROACH
- RESULTS & DISCUSSION
- CONCLUSION

INTRODUCTION

- High performance computing within stringent power budgets

- Coarse-Grained Reconfigurable Array (CGRA) Architecture as accelerator
  - Near-ASIC energy efficiency and performance
  - Software-like programmability



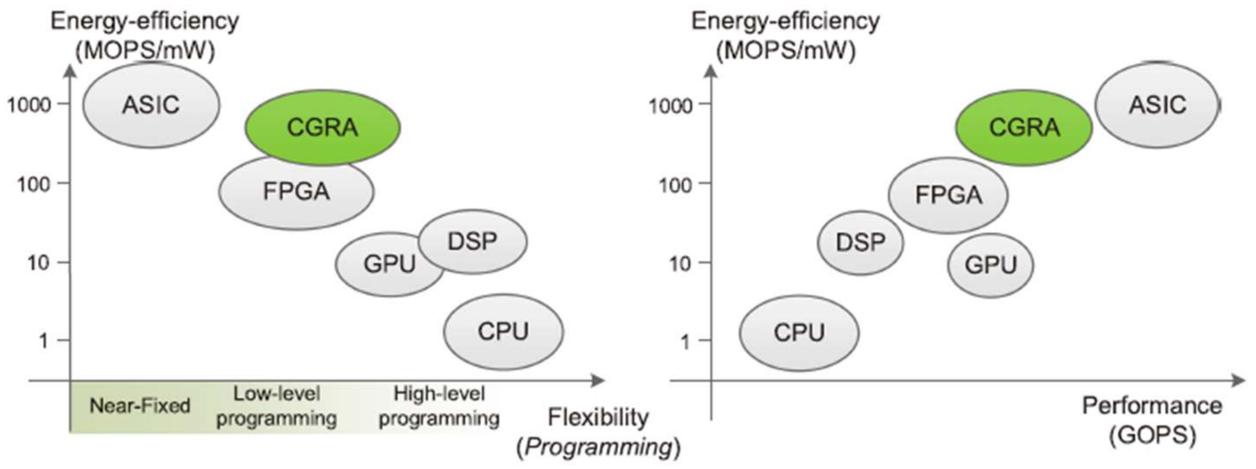
Architecture comparison in terms of flexibility, performance and energy efficiency

[A survey of coarse-grained reconfigurable architecture and design: Taxonomy, challenges, and applications, Liu Leibo et al., CSUR, 2019]

INTRODUCTION

- High performance computing within stringent power budgets

- Coarse-Grained Reconfigurable Array (CGRA) Architecture as accelerator
  - Near-ASIC energy efficiency and performance
  - Software-like programmability



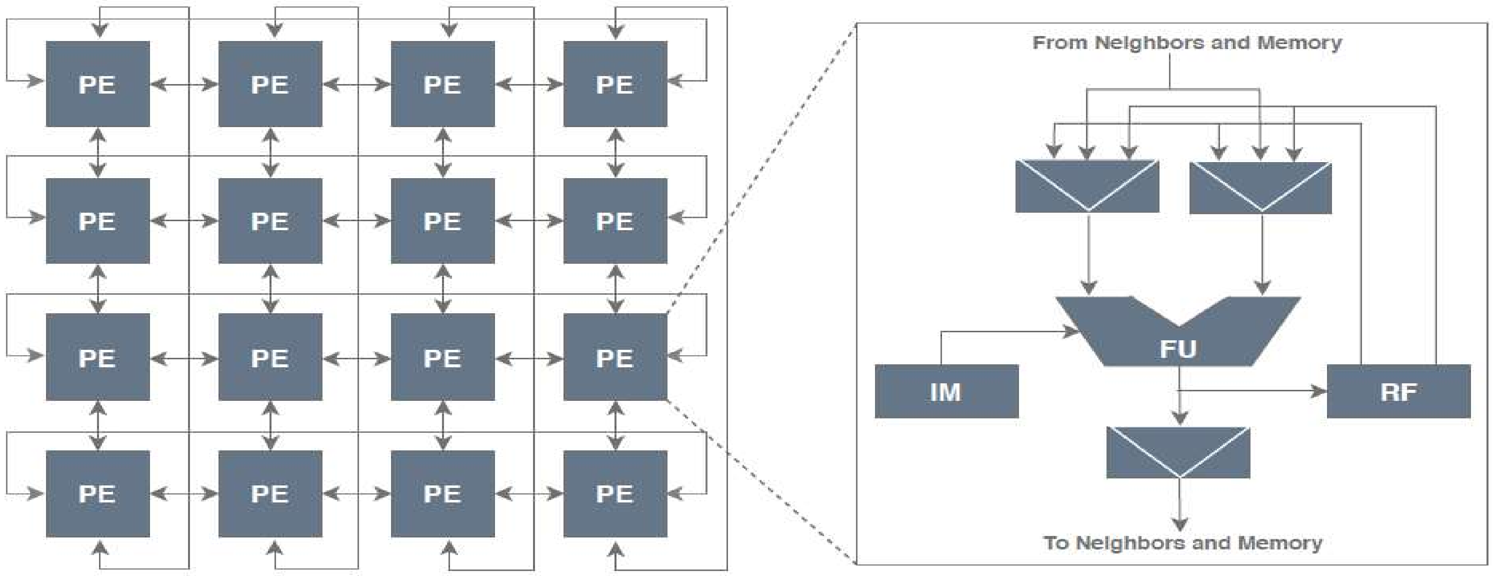
Architecture comparison in terms of flexibility, performance, and energy efficiency.

[A survey of coarse-grained reconfigurable architecture and design: Taxonomy, challenges, and applications, Liu Leibo et al., CSUR, 2019]

INTRODUCTION

What is a CGRA?

- **Array** of interconnected Processing Elements (PEs)
- **Reconfigurable**
  - PEs configurable to perform different operations
- **Coarse-Grained**
  - Support for higher-level applications like multiplication on multi-bit data
  - Word-level configurability



INTRODUCTION

Problem of Interest

- Hardware efficiency comes at the cost of hard programming
  - Automate mapping process

- Design optimizations and mapping techniques to improve the performance of CGRAs

- Focus on the optimized execution of the innermost loop
  - Outer loops executed on the host processor
  - Increases synchronization overhead
  - Diminishes the benefits of acceleration provided by the CGRA

- Optimized mapping techniques and improved architectural designs **NOT Sufficient** to guarantee the best performance

Loop Execution Model

Defines how loops are distributed between CGRA and host processor

INTRODUCTION

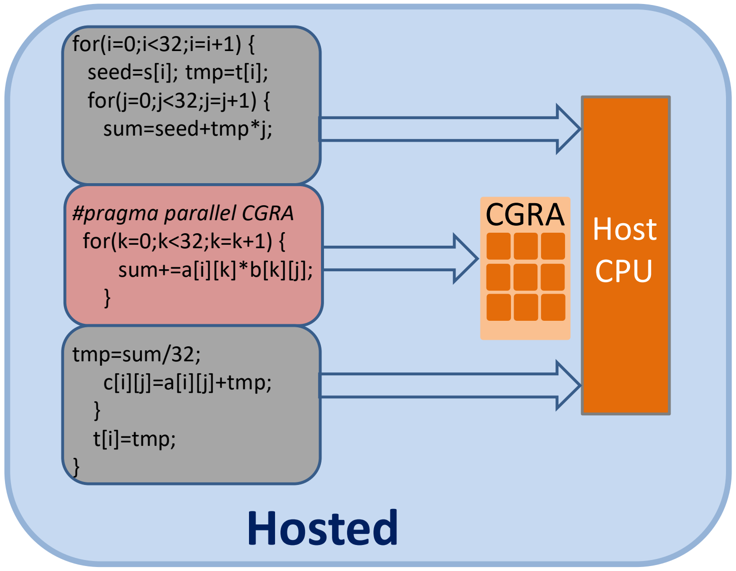
Loop Execution Model

Hosted

- Executes the innermost loop on CGRA
- Outer loops on host CPU

Standalone

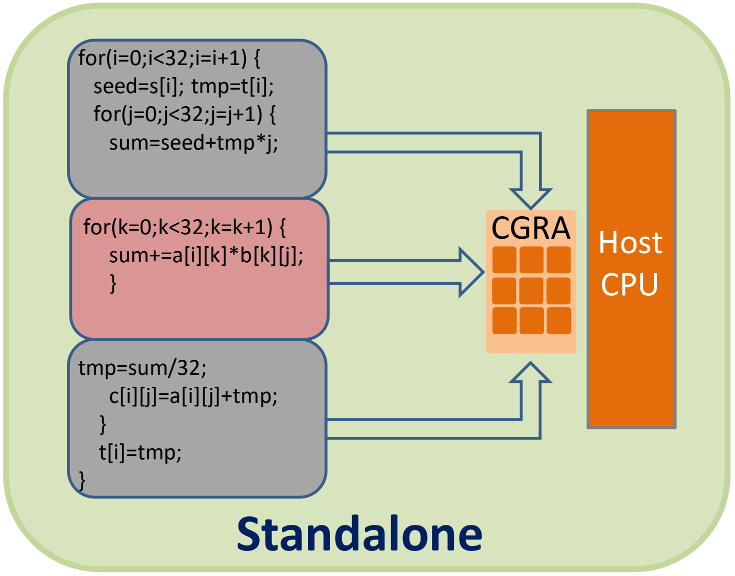
- Executes outer loops as well as the innermost loop on CGRA



```
for(i=0;i<32;i=i+1) {
  seed=s[i]; tmp=t[i];
  for(j=0;j<32;j=j+1) {
    sum=seed+tmp*j;
```

```
#pragma parallel CGRA
  for(k=0;k<32;k=k+1) {
      sum+=a[i][k]*b[k][j];
    }
```

```
tmp=sum/32;
    c[i][j]=a[i][j]+tmp;
  }
  t[i]=tmp;
}
```



```
for(i=0;i<32;i=i+1) {
  seed=s[i]; tmp=t[i];
  for(j=0;j<32;j=j+1) {
    sum=seed+tmp*j;
```

```
for(k=0;k<32;k=k+1) {
    sum+=a[i][k]*b[k][j];
  }
```

```
tmp=sum/32;
    c[i][j]=a[i][j]+tmp;
  }
  t[i]=tmp;
}
```

INTRODUCTION

Major Contributions

- Explorative study of different execution models
  - Impact in determining the performance and energy efficiency of CGRAs

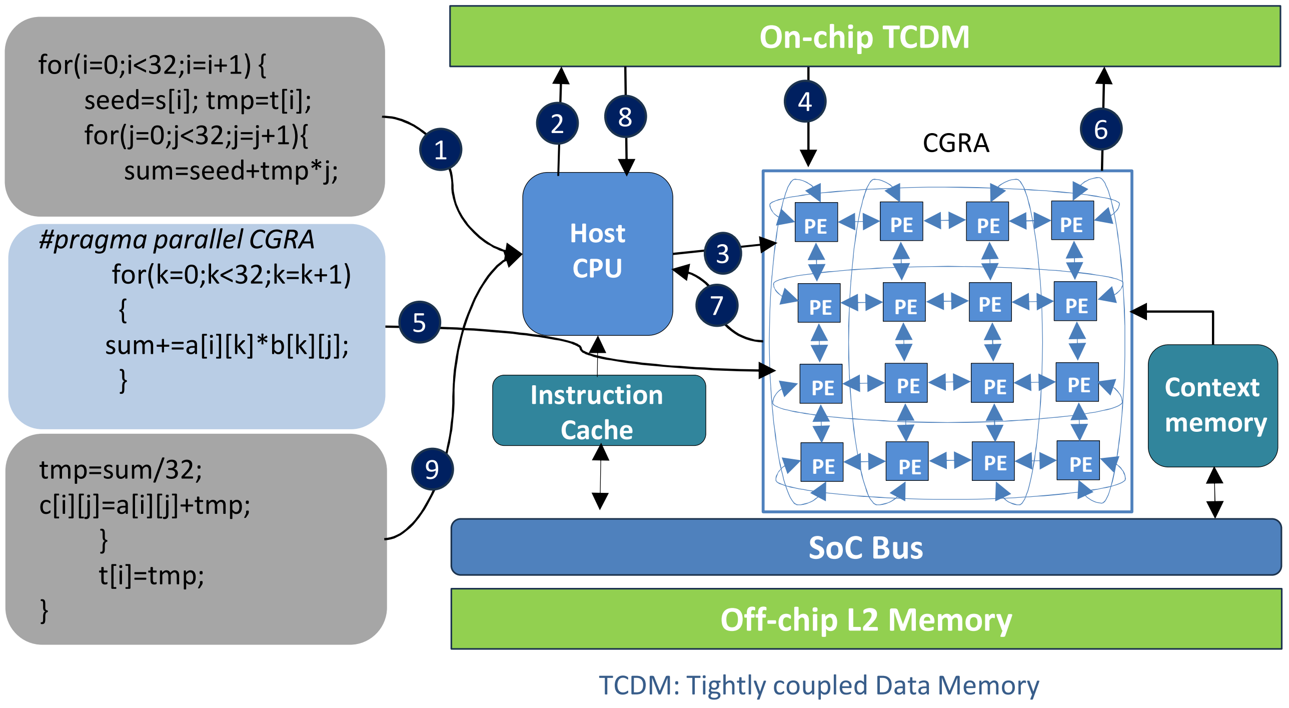
- Compilation flow supporting the standalone execution of nested loops

OVERVIEW

- INTRODUCTION
- **MOTIVATION & BACKGROUND**
- PROPOSED APPROACH
- RESULTS AND DISCUSSION
- CONCLUSION

MOTIVATION & BACKGROUND

Hosted Loop Execution Model



```
for(i=0;i<32;i=i+1) {
      seed=s[i]; tmp=t[i];
      for(j=0;j<32;j=j+1){
            sum=seed+tmp*j;
```

```
#pragma parallel CGRA
          for(k=0;k<32;k=k+1)
          {
         sum+=a[i][k]*b[k][j];
          }
```

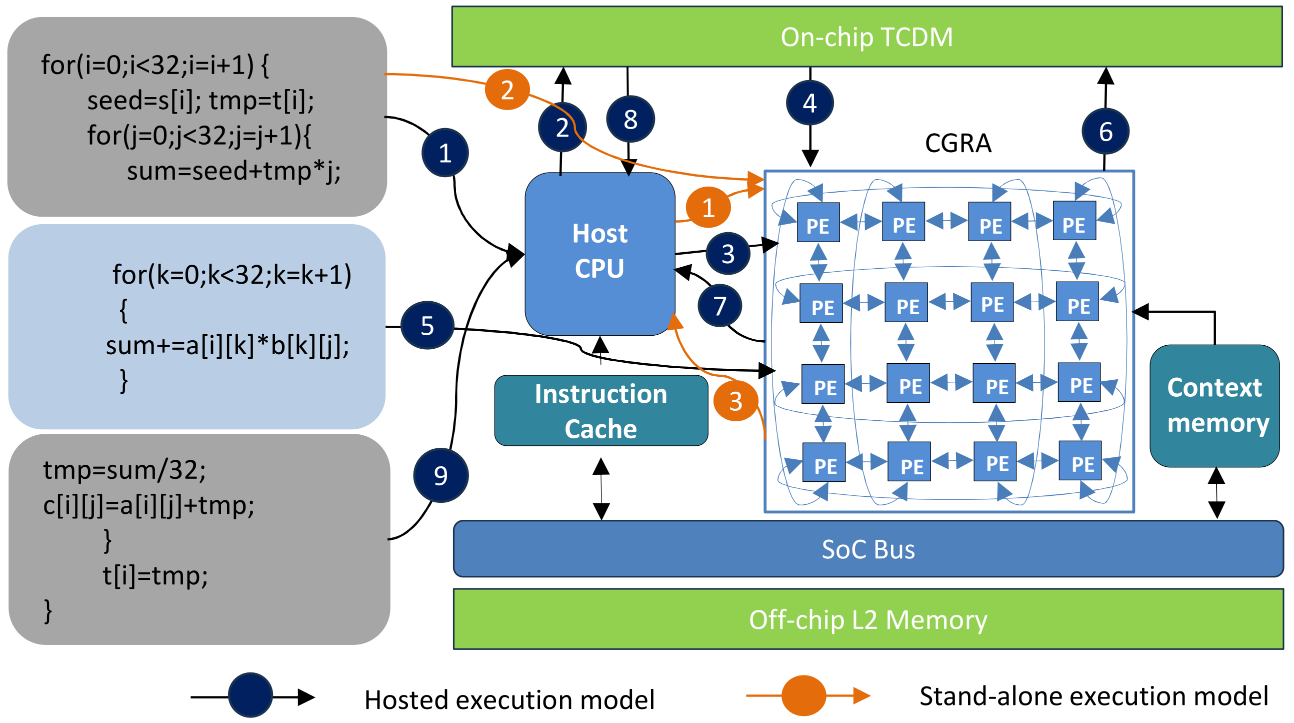
```
tmp=sum/32;
c[i][j]=a[i][j]+tmp;
        }
        t[i]=tmp;
}
```

TCDM: Tightly coupled Data Memory

1. Outer loop start execution
2. Live-in variables store from CPU
3. CGRA start execution
4. Live-in variables load in CGRA
5. CGRA execution - innermost loop
6. Live-out variables store from CGRA
7. CGRA end execution
8. Live-out variables load in CPU
9. Outer loop end execution

- Live-in and live-out variables are transferred through shared memory
  - Live-in Variables: variables needed for the CGRA to execute the innermost loop
  - Live-out Variables: variables the processor needs from CGRA to execute the outer loops

MOTIVATION & BACKGROUND



Hosted Loop Execution Model

1. Outer loop start execution
2. Live-in variables store from CPU
3. CGRA start execution
4. Live-in variables load in CGRA
5. CGRA execution - innermost loop
6. Live-out variables store from CGRA
7. CGRA end execution
8. Live-out variables load in CPU
9. Outer loop end execution

Standalone Execution Model

Standalone Loop Execution Model

1. CGRA start execution
2. CGRA execution - outer loops + innermost loop
3. CGRA end execution

Overhead of Hosted Execution Model : Extra memory operations and communication for synchronization

Ideal Execution Model : Standalone

MOTIVATION & BACKGROUND

Existing Solutions - Hosted

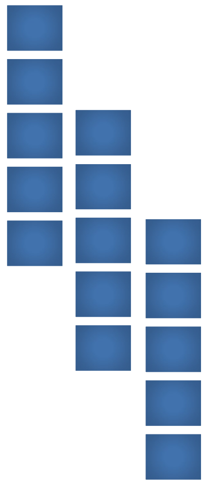
```
for(i=0;i<32;i=i+1){
        sum=c[i];
        for(j=0;j<32;j=j+1) {
            sum+=a[i][j]+b[i][j];
        }
       c[i]=sum;
     }
```

- Most of the CGRA implementations follow the hosted loop execution model
  - Synchronization overhead with host processor
- Optimizes execution of the innermost loop execution
  - Modulo scheduling, loop unrolling, loop flattening
  - Modulo scheduling, the most commonly used loop optimization technique

Standalone Execution Model
Modulo Scheduling

Modulo Scheduling

- Software pipelining technique
  - Overlapped execution of different iterations of the innermost loop
- Finds a schedule of operations from different iterations
  - Repeated in a short interval called initiation interval (II)



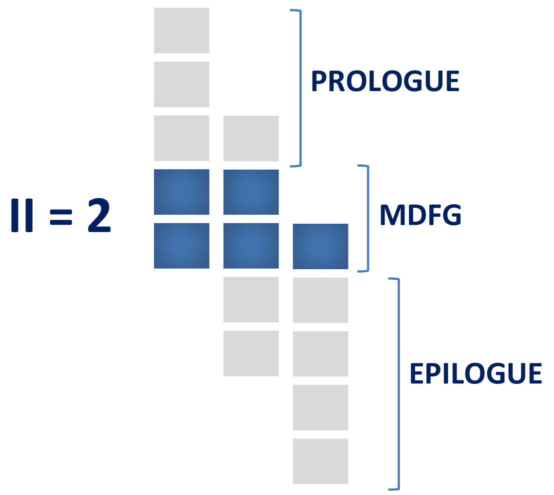
[Himap: Fast and scalable high-quality mapping on cgra via hierarchical abstraction, Wijerathne, D. et al., IEEE TCAD, 2021]
[Ramp: Resource-aware mapping for cgras, Dave, S. et al., DAC, 2018]
[Flattening-based mapping of imperfect loop nests for cgras, Lee, J. et al., CODES, 2014]
[Polyhedral model based mapping optimization of loop nests for cgras, Liu D., et al., DAC, 2013]
[Epimap: Using epimorphism to map applications on cgras, Hamzeh M. et al., DAC, 2012]
[Edge-centric modulo scheduling for coarse-grained recongurable architectures, Park H. et al., PACT, 2008]

MOTIVATION & BACKGROUND

Existing Solutions - Hosted

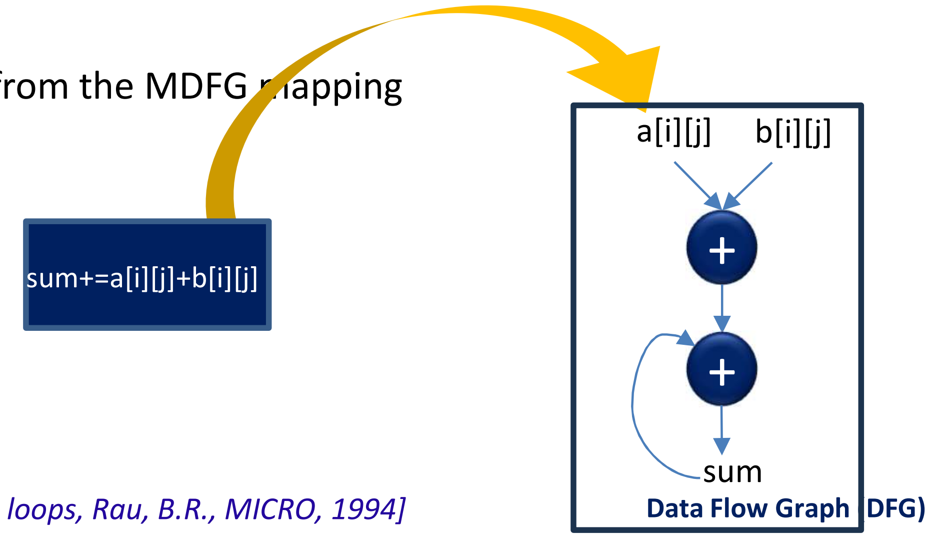
Modulo Scheduling

- MDFG (Modulo Data Flow Graph)
  - DFG formed by the repeating schedule of length II



- Prologue and Epilogue
  - DFGs formed by the set of operations executed before and after the MDFG
  - Modulo DFG Trio (MDT)
- Mapping Problem
  - Considers application mapping a DFG mapping problem
  - Maps MDFG
  - Prologue and epilogue mappings prepared from the MDFG mapping

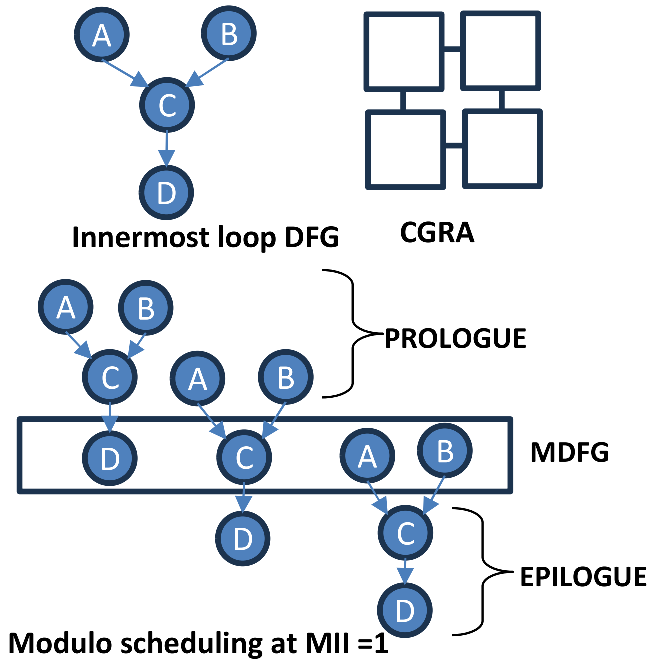
```
for(i=0;i<32;i=i+1){
        sum=c[i];
        for(j=0;j<32;j=j+1) {
            sum+=a[i][j]+b[i][j];
        }
        c[i]=sum;
    }
```



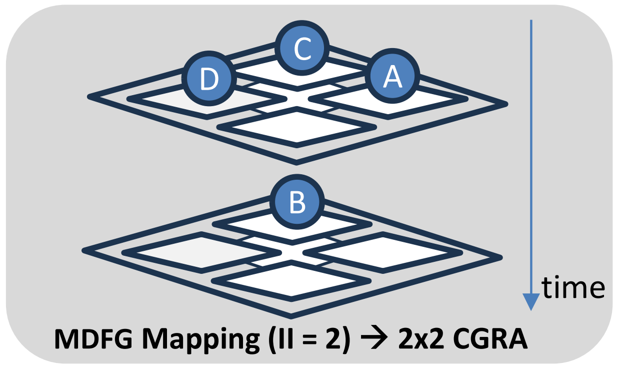
[Iterative modulo scheduling: An algorithm for software pipelining loops, Rau, B.R., MICRO, 1994]

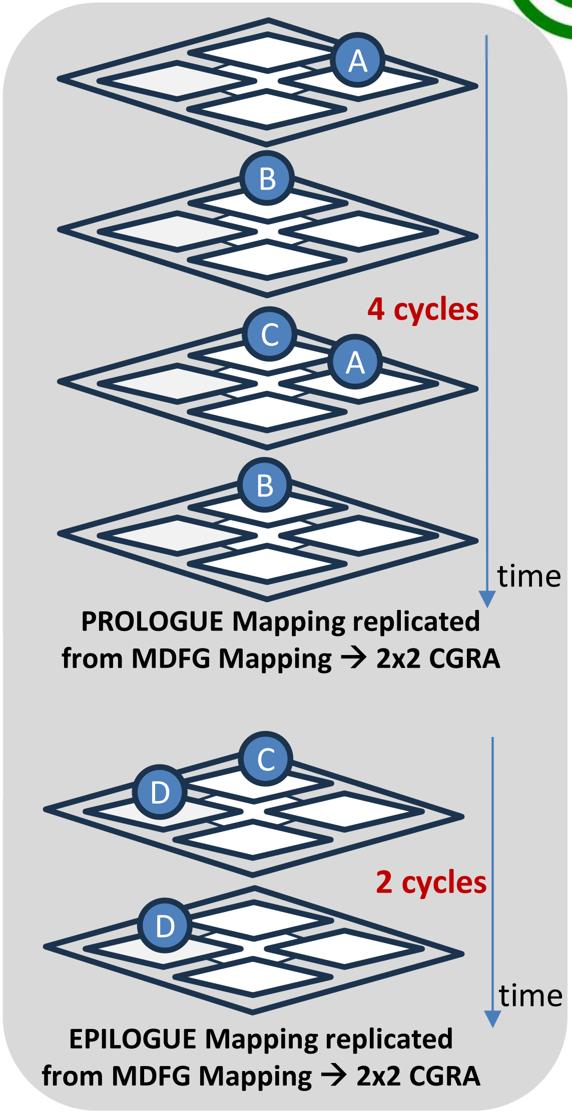
MOTIVATION & BACKGROUND

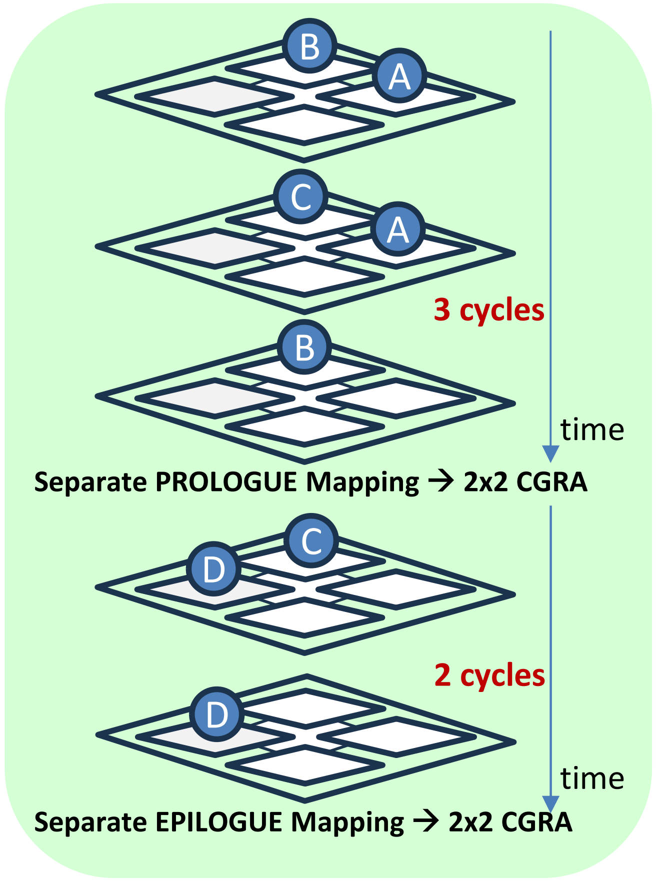
Modulo Scheduling



Standalone Execution Model
Modulo Scheduling
Separate mapping of MDFG, prologue, epilogue




A
B
C
4 cycles
time
PROLOGUE Mapping replicated from MDFG Mapping → 2x2 CGRA
D
2 cycles
time
EPILOGUE Mapping replicated from MDFG Mapping → 2x2 CGRA


B
A
C
3 cycles
time
Separate PROLOGUE Mapping → 2x2 CGRA
D
2 cycles
time
Separate EPILOGUE Mapping → 2x2 CGRA

Mapping = Scheduling + Placement

Separate mapping of MDFG, prologue, and epilogue considering each as an individual DFG improves performance

MOTIVATION & BACKGROUND

Existing Solutions - Standalone

Cheng et al.

- Flattens loop nests into a single-nested loop to facilitate DFG mapping
- Modulo schedules the resultant DFG
- Inflated DFG when the number of loops gets increased
  - Increased II and high energy consumption

Integrated Programmable Array (IPA) [Das et al.]

- Employs direct CDFG mapping
  - Register allocation-based mapping
- Does not support modulo scheduling

[Opencgra: An open-source unified framework for modeling, testing, and evaluating cgras., Tan, C. et al., ICCD, 2020]
[Efficient mapping of cdfg onto coarse-grained reconfigurable array architectures, Das, S. et al., ASP-DAC, 2017]

OVERVIEW

- INTRODUCTION
- MOTIVATION & BACKGROUND
- **PROPOSED APPROACH**
- RESULTS & DISCUSSION
- CONCLUSION

PROPOSED APPROACH

Standalone Execution Model
Modulo Scheduling
Separate mapping of MDFG, prologue, epilogue

A novel compilation flow supporting:

- Standalone execution of the entire loop nest
  - Direct CDFG Mapping
- Modulo scheduling of the innermost loop

PROPOSED APPROACH

Standalone Execution Model
Modulo Scheduling
Separate mapping of MDFG, prologue, epilogue

A novel compilation flow supporting:

- Standalone execution of the entire loop nest
    - Direct CDFG Mapping
- Modulo scheduling of the innermost loop

Combines modulo scheduling with CDFG mapping

Separate mapping of MDFG, prologue and epilogue DFGs (MDT)

PROPOSED APPROACH

Direct CDFG Mapping

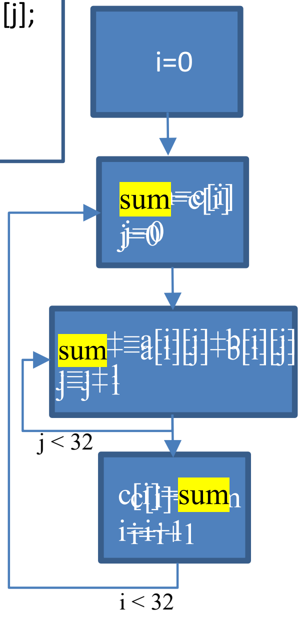
Register Allocation-Based Constraint-Aware Placement [Das et al.]

```
for(i=0;i<32;i=i+1){
          sum=c[i];
          for(j=0;j<32;j=j+1) {
              sum+=a[i][j]+b[i][j];
          }
         c[i]=sum;
      }
```

- Each Basic block (BB) in the CDFG mapped individually
- Control flow mapping by supporting JMP instruction

- Maintains data integrity between BB mappings
  - Symbol variables : variables that are used in multiple BBs

- Constraint-Aware Placement
  - Impose register allocation-based constraints in mapping the data nodes (variables)



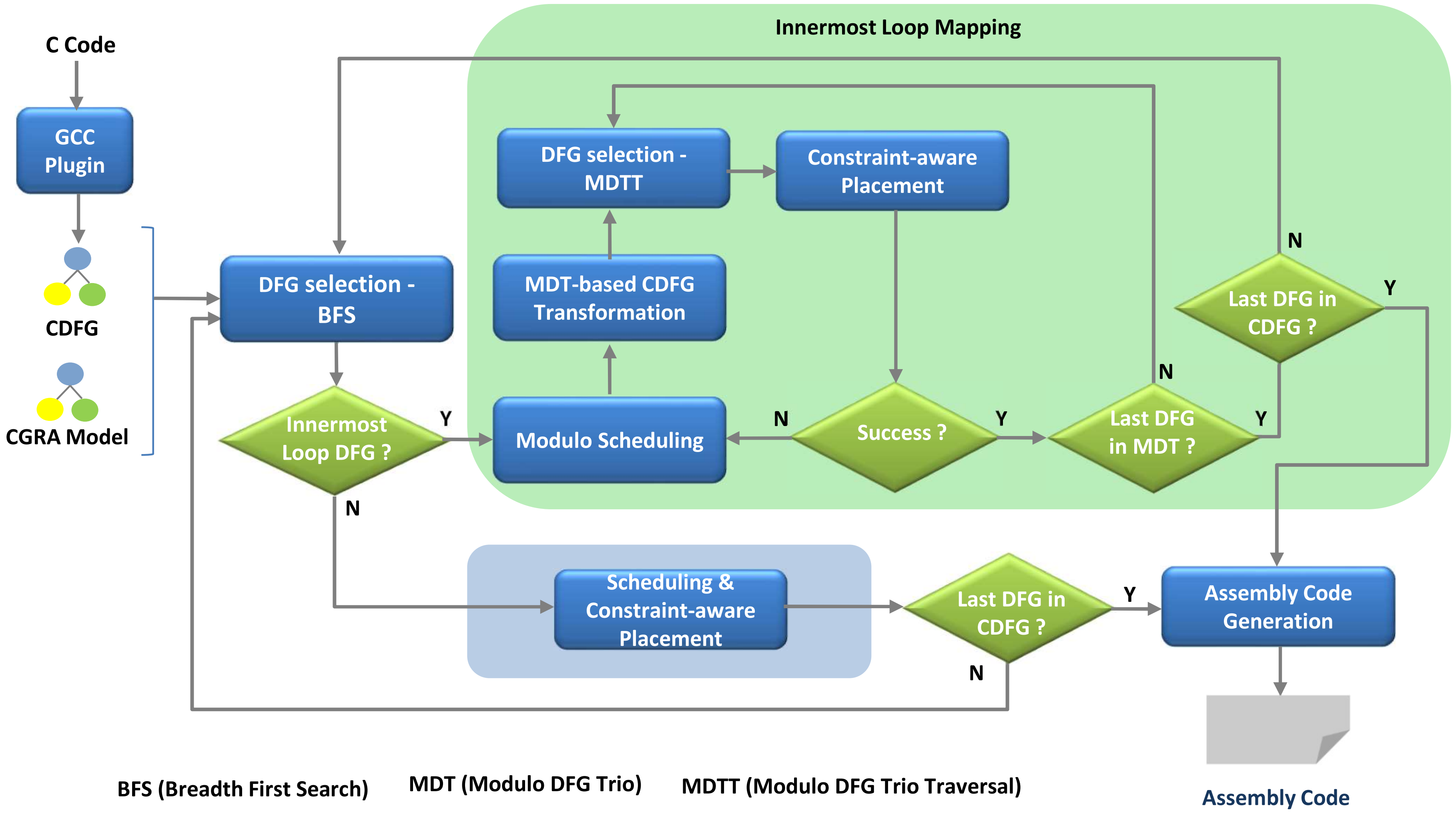
Control and Data Flow Graph (CDFG)

- Number of constraints depends on DFG selection criteria
  - Breadth-first search (BFS) induces the least number of constraints
  - Mapping BBs with a greater number of symbol nodes early helps to reduce the number of constraints

[Efficient mapping of cdfg onto coarse-grained reconfigurable array architectures, Das, S. et al., ASP-DAC, 2017]

PROPOSED APPROACH

Proposed Compilation Flow



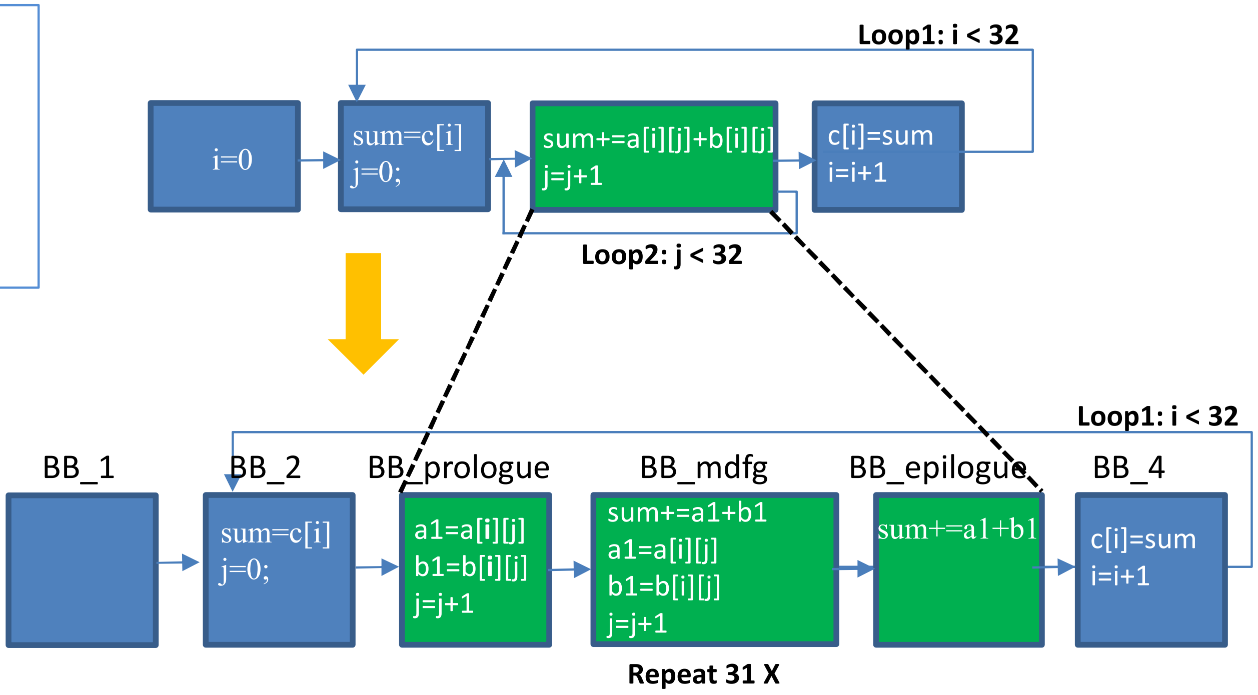
BFS (Breadth First Search) MDT (Modulo DFG Trio) MDTT (Modulo DFG Trio Traversal)

PROPOSED APPROACH

MDT-based CDFG transformation

- Innermost loop DFG replaced with a CDFG formed by Modulo DFG Trio (MDT)
  - Nested CDFG mapping problem

```
Loop1: for(i=0;i<32;i=i+1){
          sum=c[i];
Loop2:        for(j=0;j<32;j=j+1) {
              sum+=a[i][j]+b[i][j];
          }
          c[i]=sum;
       }
```

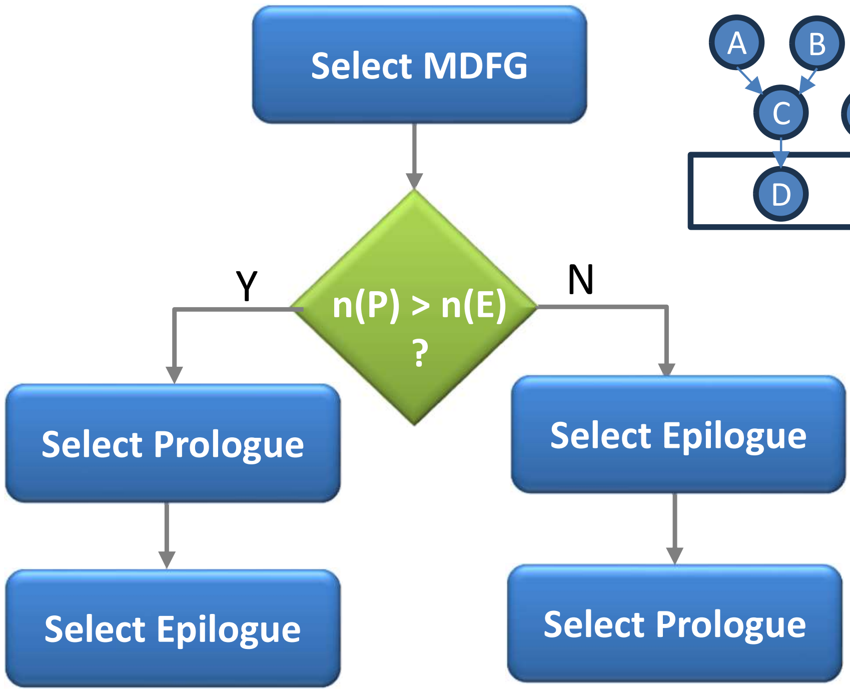


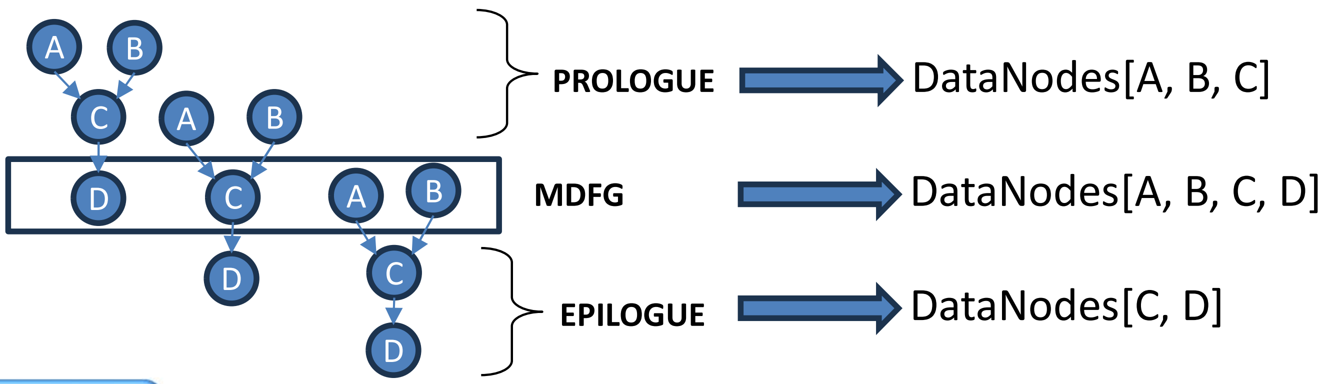
PROPOSED APPROACH

DFG selection by Modulo DFG Trio Traversal (MDTT)

- DFG (BB) selection affects the number of constraints
  - Mapping BBs with a greater number of symbol nodes early helps to reduce the number of target location constraints

- MDFG contains the highest number of symbol variables among the DFGs in MDT





n(P) : no. of variables in prologue
n(E) : no. of variables in epilogue

Let P, M and E be the set of all variables in prologue, MDFG and epilogue respectively and S be the set of all symbol variables (variables used in multiple BBs)

M = P U E
S = (P ∩ M) U (M ∩ E) U (P ∩ E)
P ∩ M = P ; M ∩ E = E; (P ∩ E) ⊆ M
Therefore S = P U E U (P ∩ E) = M

No. of symbol variables in prologue = n(S ∩ P) = n(M ∩ P) = n(P)
No. of symbol variables in MDFG = n(S ∩ M) = n(M ∩ M) = n(M)
No. of symbol variables in epilogue = n(S ∩ E) = n(M ∩ E) = n(E)
n(M) = n(P) + n(E) – n(P ∩ E) => n(M) > n(P) & n(M) > n(E)

OVERVIEW

RESULTS & DISCUSSION

Experimental Setup

- Target CGRA: 4×4 PE array configuration of IPA architecture
  - Loosely coupled with host CPU (RISCV)
- RTL synthesis: Cadence Genus
  - 90nm CMOS technology library
- Placement & Routing : Cadence Innovus
- Power Analysis : Cadence Voltus
- RTL/ netlist simulation: Questasim
- A set of loop-intensive signal processing kernels
  - Including those from PolyBench benchmark suite

[An energy-efficient integrated programmable array accelerator and compilation flow for near-sensor ultralow power processing, Das, S. et al., IEEE TCAD, 2018]
[http://www-roc.inria.fr/ pouchet/software/polybench]

RESULTS & DISCUSSION

PHASE 1: Performance Comparison of Different Execution Models

- Standalone [Proposed] vs Hosted
- Modulo Scheduling Technique: Epimap [Hamzeh et al.]

Execution Latency (cycles)

Kernel	Hosted	Standalone	Speed-up
Matrix Multiplication	464 159	113 310	4.10x
Histogram Equalization	25 225	15 484	1.63x
2D Non-Sep Filter	2 783 615	225 768	12.33x
FIR Filter	43 365	6,308	6.87x
DCT	14 450	2 813	5.14x
Bicg	12 452	6 451	1.93x
2D Convolution	1 352 406	126 446	10.70x
Sobel Filter	2 534 676	2 23 844	11.32x
Average			**6.75x**

A maximum of **12.33x** and an average of **6.75x** speed-up

[Epimap: Using epimorphism to map applications on cgras, Hamzeh M. et al., DAC 2012]

RESULTS & DISCUSSION

PHASE 1: Performance Comparison of Different Execution Models

- Standalone [Proposed] vs Hosted

Throughput (Mpbs)

Kernel	Hosted	Standalone	Gain
Matrix Multiplication	1.24	5.07	4.10x
Histogram Equalization	106.83	174.03	1.63x
2D Non-Sep Filter	0.97	11.94	12.33x
FIR Filter	2.59	17.80	6.87x
DCT	2.49	12.77	5.14x
Bicg	2.89	5.57	1.93x
2D Convolution	1.00	10.66	10.70x
Sobel Filter	0.91	10.27	11.32x
Average			**6.75x**

A maximum of **12.33x** and an average of **6.75x** gain in Throughput

RESULTS & DISCUSSION

PHASE 1: Energy Results on Different Execution Models

- Standalone [Proposed] vs Hosted

Energy (μJoule)

Kernel	Hosted	Standalone	Gain
Matrix Multiplication	287.64	58.97	4.88x
Histogram Equalization	15.31	8.06	1.90x
2D Non-Sep Filter	1702.74	117.49	14.49x
FIR Filter	26.37	3.28	8.03x
DCT	8.79	1.46	6.01x
Bicg	7.56	3.36	2.25x
2D Convolution	845.47	65.80	12.85x
Sobel Filter	1583.88	116.49	13.60x
Average			**8.00x**

Memory operations performed in the live-in and live-out phases of the hosted execution significantly increase energy consumption

A maximum of **14.49x** and an average of **8.00x** reduction in energy consumption

RESULTS & DISCUSSION

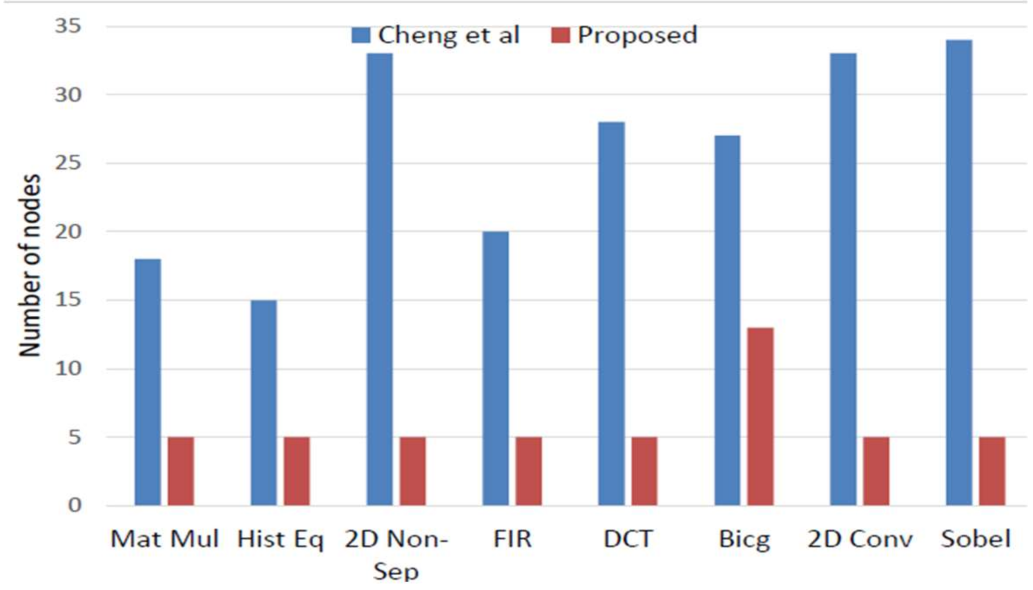
PHASE 2: Comparison with state-of-the-art Standalone Solution

- Proposed vs state-of-the-art solution employing Loop Flattening [Cheng et al.]

```
Loop1: for (i=0; i<M ; i++){
            sum=0;
Loop2: for (j=0; j<N ; j++)
            sum += array_in[i][j];
array_out[i] = sum;
}
```

```
Loop1: for (n=0; n < M * N ; n++){
            i = n / N ;
            j = n % N ;
            if (j==0)
                        sum=0;
            sum += array_in[i][j];
            if (j==N -1)
                        array_out[i] = sum;
}
```

No. of nodes in the modulo scheduled DFG



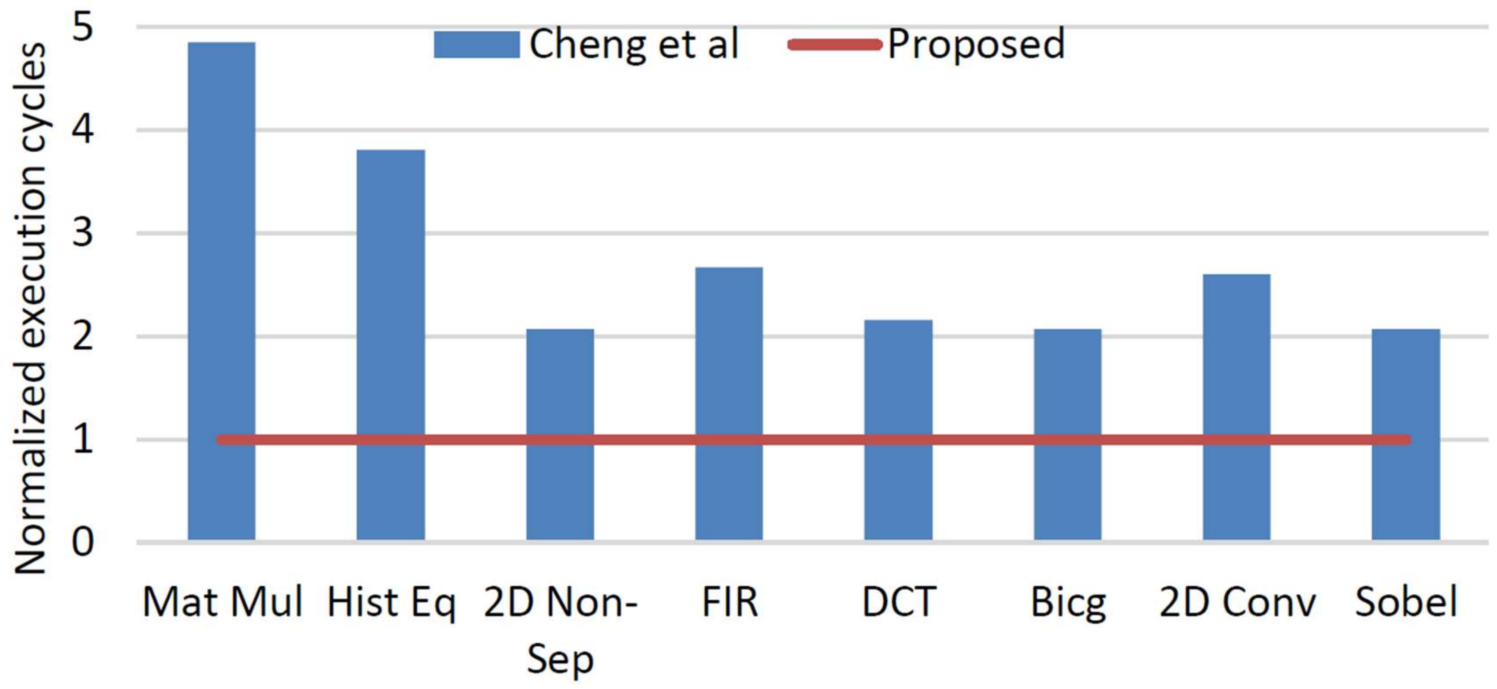
[Opencgra: An open-source unified framework for modeling, testing, and evaluating cgras., Tan, C. et al., ICCD, 2020]

RESULTS & DISCUSSION

PHASE 2: Comparison with state-of-the-art Standalone Solution

- Proposed vs state-of-the-art solution employing Loop Flattening [Cheng et al.]

Execution Latency (cycles)



A maximum of **4.80x** and an average of **2.80x** speed-up

[Opencgra: An open-source unified framework for modeling, testing, and evaluating cgras., Tan, C. et al., ICCD, 2020] 30

OVERVIEW

- INTRODUCTION
- MOTIVATION & BACKGROUND
- PROPOSED APPROACH
- RESULTS & DISCUSSION
- **CONCLUSION**

CONCLUSION

- Explorative study on the impact of execution model on performance and energy efficiency of CGRAs
- A novel compilation flow for standalone nested loop acceleration on CGRA
- Combines modulo scheduling with direct CDFG mapping
  - Modulo schedules the innermost loop
  - Maps prologue, MDFG, and epilogue DFGs separately
- A maximum of **12.33×** and an average of **6.75×** speed-up

  Up to **14.49x** and an average of **8.00x** reduction in energy over hosted model
- Up to **4.80×** and an average of **2.80×** speed-up over the state-of-the-art standalone solution

THANK YOU

Questions?

112004004@smail.iitpkd.ac.in

REFERENCES

[1] Liu, L., Zhu, J., Li, Z., Lu, Y., Deng, Y., Han, J., Yin, S.,Wei, S.: A survey of coarsegrained reconfigurable architecture and design: Taxonomy, challenges, and applications.ACM Comput. Surv. 52(6) (Oct 2019). https://doi.org/10.1145/3357375

[2] Hamzeh, M., Shrivastava, A., Vrudhula, S.: Epimap: Using epimorphism to map applications on cgras. In: Proceedings of the 49th Annual Design Automation Conference (2012)

[3] Dave, S., Balasubramanian, M., Shrivastava, A.: Ramp: Resource-aware mapping for cgras. In: Proceedings of the 55th Annual Design Automation Conference (2018)

[4] Rau, B.R.: Iterative modulo scheduling: An algorithm for software pipelining loops. In: Proceedings of the 27th annual international symposium on Microarchitecture (1994)

[5] Tan, C., Xie, C., Li, A., Barker, K.J., Tumeo, A.: Opencgra: An open-source unified framework for modeling, testing, and evaluating cgras. In: 2020 IEEE 38th International Conference on Computer Design (ICCD). IEEE (2020)

[6] Das, S., Martin, K.J., Coussy, P., Rossi, D., Benini, L.: Efficient mapping of cdfg onto coarse-grained
reconfigurable array architectures. In: 2017 22nd Asia and South Pacic Design Automation Conference (ASPDAC). pp. 127{132. IEEE (2017)

[7] Das, S., Martin, K.J., Rossi, D., Coussy, P., Benini, L.: An energy-efficient integrated programmable array accelerator and compilation ow for near-sensor ultralow power processing. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 38(6), 1095{1108 (2018)

[8] Wijerathne, D., Li, Z., Pathania, A., Mitra, T., Thiele, L.: Himap: Fast and scalable high-quality mapping on cgra via hierarchical abstraction. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 41(10) (2021)

[9] Pouchet, L.N., Grauer-Gray, S.: Polybench: The polyhedral benchmark suite, 2012 (2012), http://www-roc.inria.fr/pouchet/software/polybench