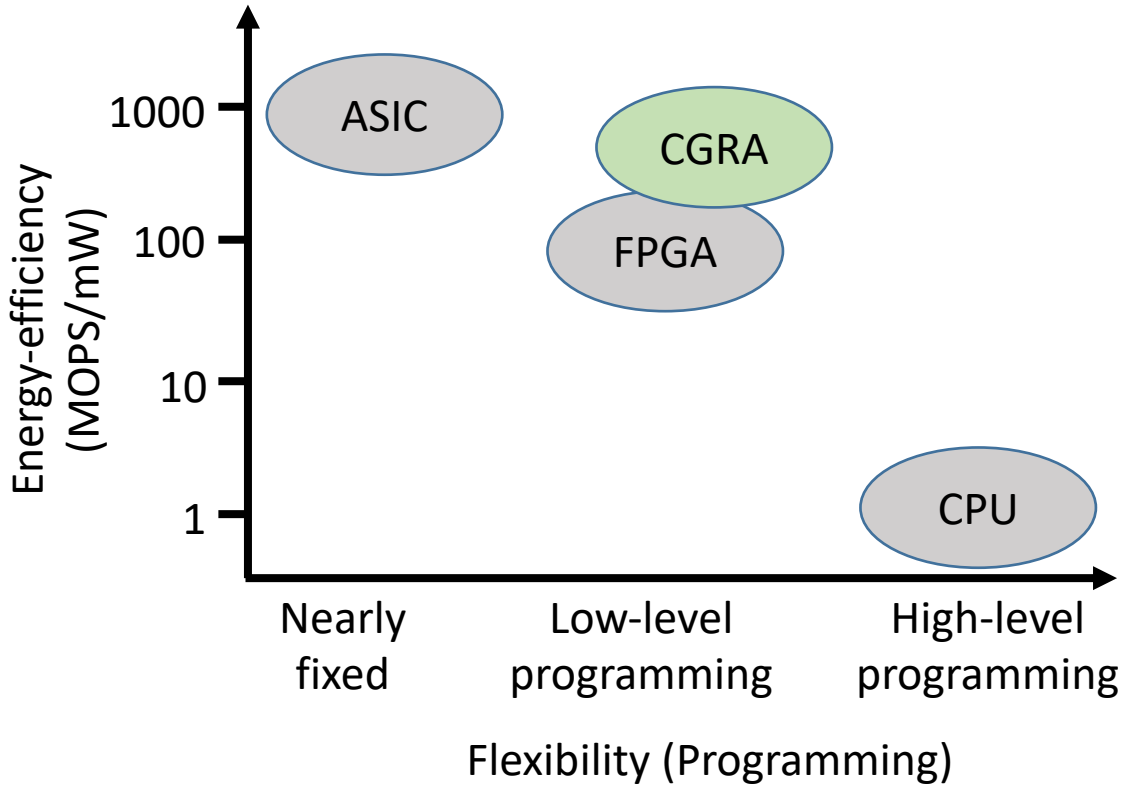

# Ultra-Fast CGRA Scheduling to Enable Run Time, Programmable CGRAs

Jinho Lee, Trevor E. Carlson

National University of Singapore

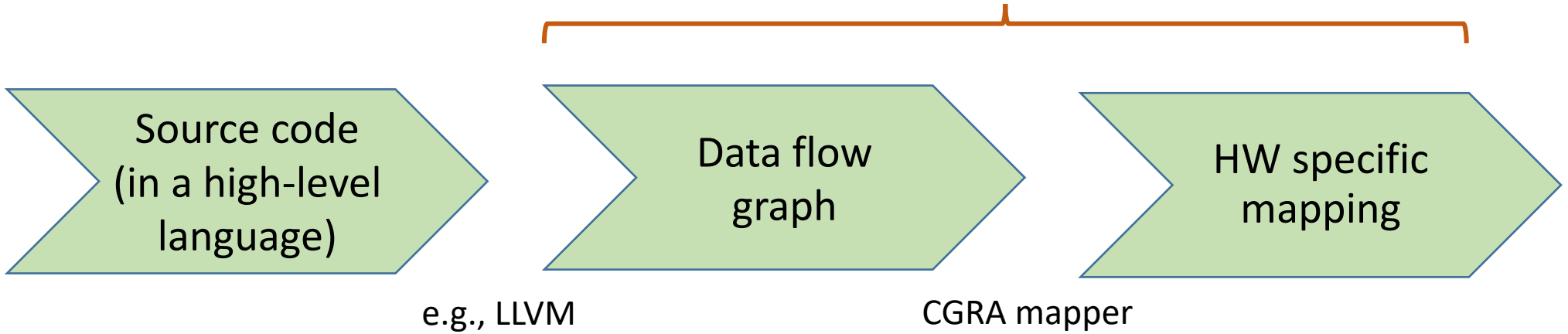
# Overview




- End of Dennard scaling
- CGRAs are a promising alternative
- CGRAs are limited to to lack of:
  - Flexibility
  - Adaptability
- Our Goal: Build flexible CGRA infrastructure

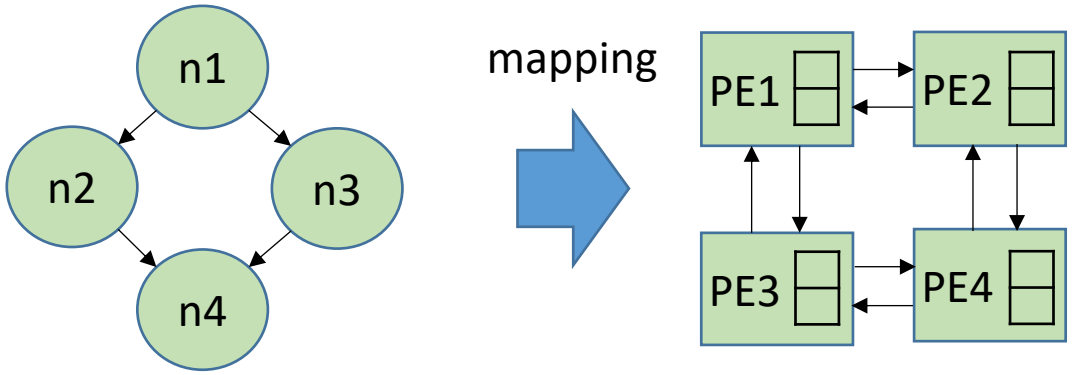
[1] Leibo Liu, Jianfeng Zhu, Zhaoshi Li, Yanan Lu, Yangdong Deng, Jie Han, Shouyi Yin, and Shaojun Wei. 2019. A Survey of Coarse-Grained Reconfigurable Architecture and Design: Taxonomy, Challenges, and Applications. *ACM Comput. Surv.* 52, 6, Article 118 (January 2020), 39 pages.

# Background: CGRA scheduling




- Compilation from data flow graph to HW specific configuration takes a long time (seconds to hours).
- A fast compiler can provide a path toward run-time scheduling.

# Background: Modulo Scheduling for CGRAs



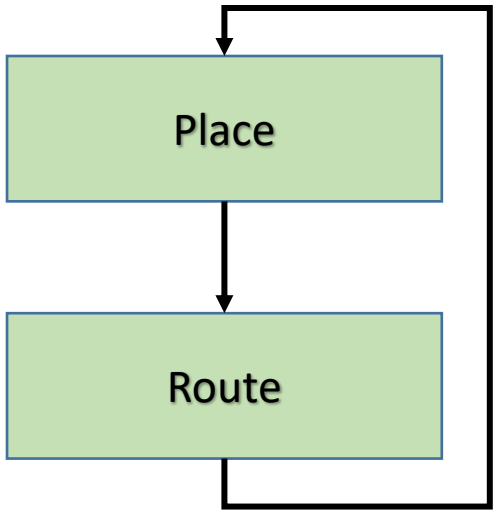

| | PE1 | PE2 | PE3 | PE4 |
|---|---|---|---|---|
| cycle 0 | | n1 | | |
| cycle 1 | n2 | n1 | n3 | |
| cycle 2 | n2 | | n3 | n4 |
| cycle 3 | | | | n4 |

- Inputs:
  - Data Flow Graph (DFG)
  - Processing Elements (PEs)
- Output:
  Mapping (or scheduling)
- Resulting performance:
  Iteration Interval (II)

[2] Bingfeng Mei, S. Vernalde, D. Verkest, H. De Man and R. Lauwereins, "Exploiting loop-level parallelism on coarse-grained reconfigurable architectures using modulo scheduling," *2003 Design, Automation and Test in Europe Conference and Exhibition*, 2003, pp. 296-301

# Challenge: scheduling is complex and time consuming

Place: looking for the best PE to place an instruction

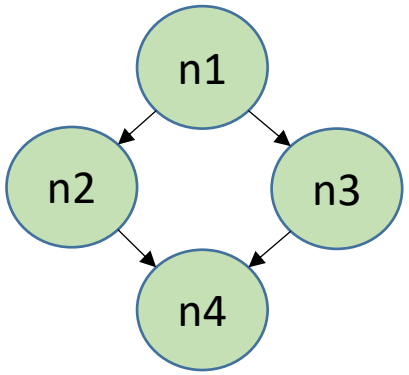


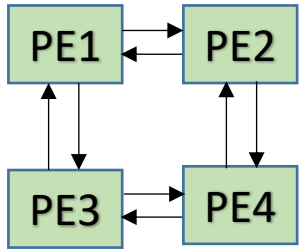

Route: allocate resources (e.g., wires, ports, etc.)

- For each instruction
  - Repeated Place & Route
- In the placement phase
  - Routing costs calculation to each PE
  - Comparison between them
  - Takes most of the time
- $O\left((V \times N)^8\right)$ where $V$ is the size of a DFG and $N$ is the size of a CGRA [3].

[3] Shail Dave, Mahesh Balasubramanian, and Aviral Shrivastava. 2018. RAMP: resource-aware mapping for CGRAs. In *Proceedings of the 55th Annual Design Automation Conference* (*DAC '18*)

# Observation

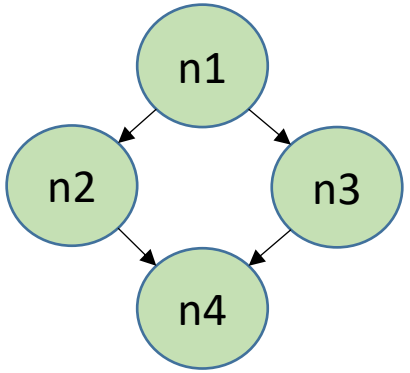


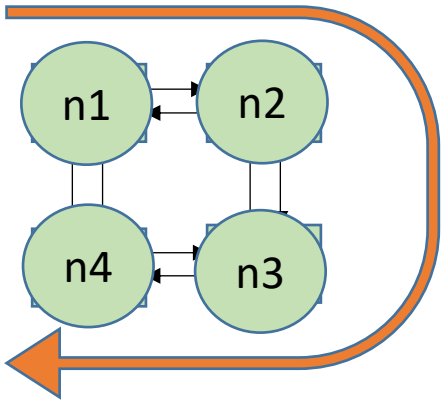




| | PE1 | PE2 | PE3 | PE4 |
|---|---|---|---|---|
| cycle 0 | n1 | | | |
| cycle 1 | | | | |

- Routing becomes difficult due to the single-hop data transfer constraint.
  - 3-dimensional problem (e.g., col, row, and time)
  - Ex) If n1 is placed at PE1 in cycle 0, where and when n2 should be mapped?
  - For each candidate, routing cost is calculated considering the time dimension.
- Too **frequent** shortest path searching in 3D.

# Our approach results in a significantly faster compile time




- Operations are placed following a topological order.
  - Parent instrs. tend to be followed by their children instrs.
  - We try to place them close to each other without calculating routing costs to all PEs.
- Placement is done in the fixed order of PEs in O(1) time.
- If multi-hop data transfer is allowed, Routing is just BFS in 2-dimensions.
  - Using multi-hop data transfer feature of state-of-the-art CGRA implementation [4].

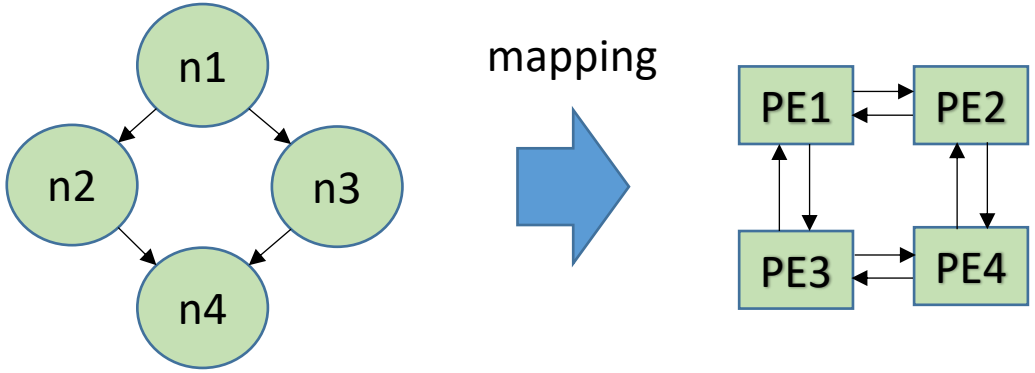
[4] M. Karunaratne, A. K. Mohite, T. Mitra and L. Peh, "HyCUBE: A CGRA with reconfigurable single-cycle multi-hop interconnect," *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2017

# Our approach: diagonal mapping

- For each target II, for each PE, we try to place instructions until the PE has II number of instructions in its configuration memory.
  - Find all paths from its already placed parent instructions to this PE.
  - If all the paths exist, allocate the resources. Otherwise, try to place at the next next candidate position & time slot.
- Instructions placed at N-th PE are executed following instructions at (N-1)-th PE.
- ex) If the II is 2 and all paths exist,

| | PE1 | PE2 | … |
|---|---|---|---|
| cycle 0 | n1 | n3 | |
| cycle 1 | n2 | n4 | |
| cycle 2 | n1 | n3 | |
| cycle 3 | n2 | n4 | |
| … | | | |

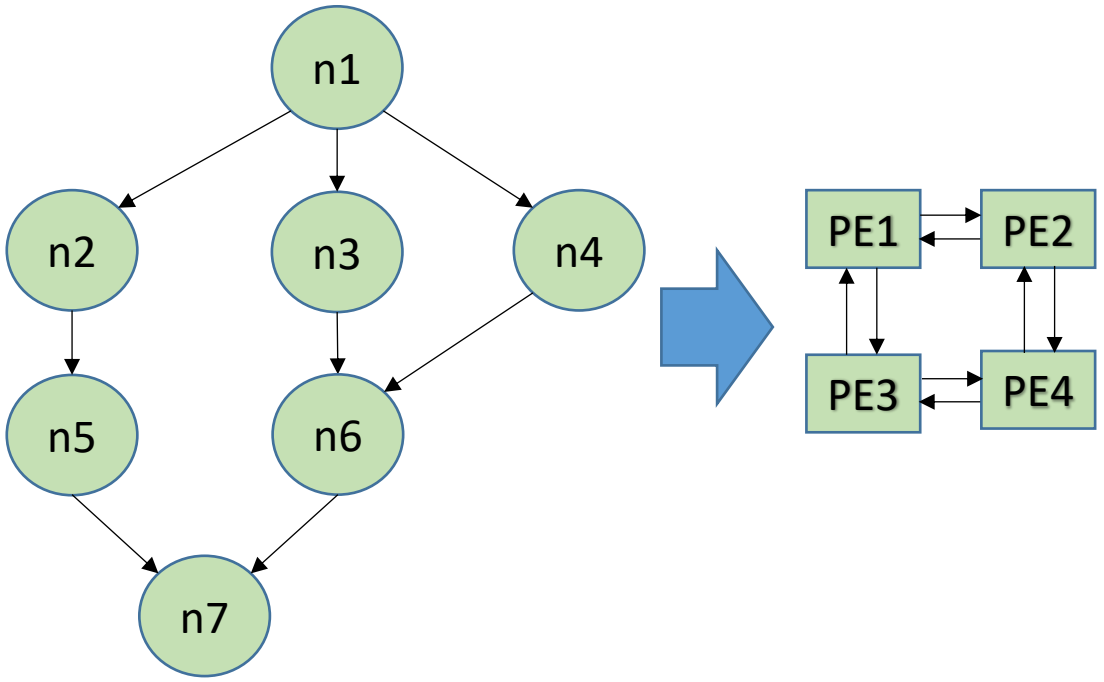
# Our approach: diagonal mapping




| | PE1 | PE2 | PE4 | PE3 |
|---|---|---|---|---|
| cycle 0 | n1 | | | |
| cycle 1 | n1 | n2 | | |
| cycle 2 | | n2 | n3 | |
| cycle 3 | | | n3 | n4 |

- PE order:
  - PE1 → PE2 → PE4 → PE3
- We achieve the same II (=1) with the previous example.

# Diagonal mapping: one more example




| | PE1 | PE2 | PE4 | PE3 |
|---|---|---|---|---|
| cycle 0 | n1 | | | |
| cycle 1 | n2 | n1 | | |
| cycle 2 | n1 | n3 | n2 | |
| cycle 3 | n2 | n4 | n3 | |
| cycle 4 | | n3 | n5 | n4 |
| cycle 5 | | n4 | n6 | n5 |
| cycle 6 | | | n5 | n7 |
| cycle 7 | | | n6 | |
| cycle 8 | | | | n7 |

# Experimental setup

- 4x4 CGRA with memory access only at the first column
- Single-hop anywhere data transfer
- Unlimited on PE register file to store the data reached too early
- Tested on benchmarks from various domains.

| benchmark | Nodes | Edges | Domain |
|---|---|---|---|
| adpcm en. | 103 | 167 | telecom |
| adpcm de. | 84 | 128 | telecom |
| dwt | 122 | 190 | signal |
| eq. of SF | 48 | 57 | physics |
| fft | 20 | 26 | telecom |
| gemm | 17 | 23 | lin. algebra |
| gsm | 88 | 110 | speech |
| hydro frag. | 22 | 25 | data parl. |
| jpeg | 90 | 142 | image |
| mpeg2 | 154 | 213 | video |
| 1st diff. | 17 | 21 | approx. |
| 2d- ehf | 52 | 79 | data parl. |


# Results

| Benchmark | Compilation Time (s) | | |
|---|---|---|---|
| | Baseline | Ours | Speedup |
| adpcm en. | 5,825.40 | 0.039 | 149,360 |
| adpcm de. | 4,716.91 | 0.016 | 294,806 |
| dwt | 20,226.81 | 0.055 | 367,760 |
| eq. of SF | 47.21 | 0.018 | 2,623 |
| fft | 18.84 | 0.016 | 1,177 |
| gemm | 20.54 | 0.045 | 455 |
| gsm | 15,638.50 | 0.019 | 823,078 |
| hydro frag. | 12.18 | 0.008 | 1,522 |
| jpeg | 2,433.08 | 0.036 | 67,585 |
| mpeg2 | 61,947.53 | 0.121 | 511,963 |
| 1st diff. | 13.92 | 0.016 | 870 |
| 2d-ehf | 1,497.16 | 0.020 | 74,857 |

- Up to 800,000x faster
- Within 2 IIs of the previous compiler


# Conclusion

- We propose a fast CGRA compilation with mapping quality comparable to state-of-the-art.
- CGRA scheduling now takes place in milliseconds, which takes hours with previous works.
- We hope that this technique can help to expand CGRA uses cases though run time scheduling.


# Thank you