Compilation and Optimization Techniques for Coarse-Grained Reconfigurable Architectures

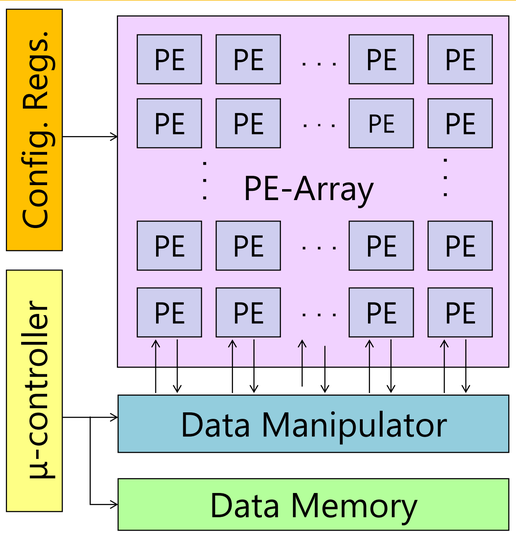
Takuya Kojima*†, *Keio University, Japan, †RIKEN, Japan

Introduction

CGRA (Coarse-Grained Reconfigurable Architecture) is a promising computation platform thanks to its programmability and high energy efficiency. Instead of bit-level reconfigurability like FPGAs, CGRAs have a coarser one (i.e., word-level), which mitigates the complexity of the compiler's task. However, the compilation process for CGRAs, including optimization, is still challenging since each CGRA targets different applications and needs different optimization policies. In this presentation, we introduce our toolchain to compile and optimize CGRA applications and show some case studies targeting two different CGRAs, one for IoT and embedded systems and the other for HPC workloads.

Examples of Coarse-Grained Reconfigurable Architectures (CGRAs)

Cool Mega Array (CMA)

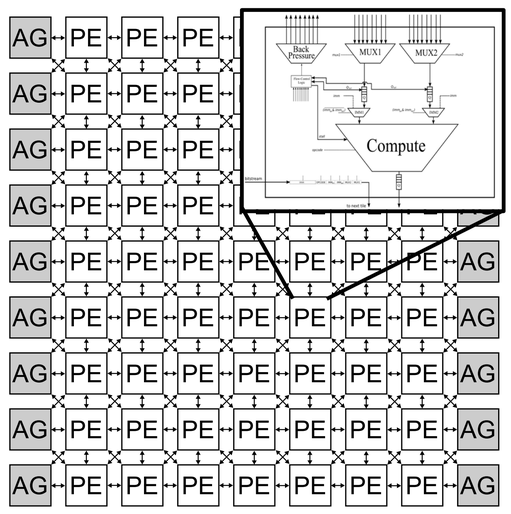


[N.Ozaki, 2011]

- **A low power CGRA**
- **PE (Processing Element)**
  - Composed of
    1. Simple ALU
    2. Switching Element
  - No Register file

  → No need of clock signal
  - Straightforward data-flow
- **µ-controller**
  - Controls data transfer b/w data memory & PE array

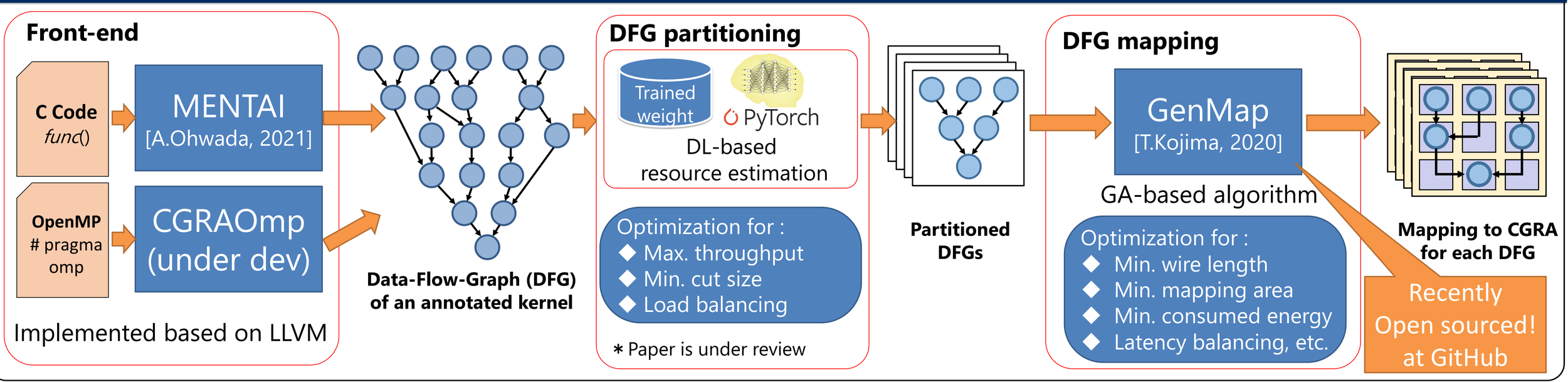
Riken High-Performance CGRA (RHP-CGRA)



[A. Podobas, 2020]

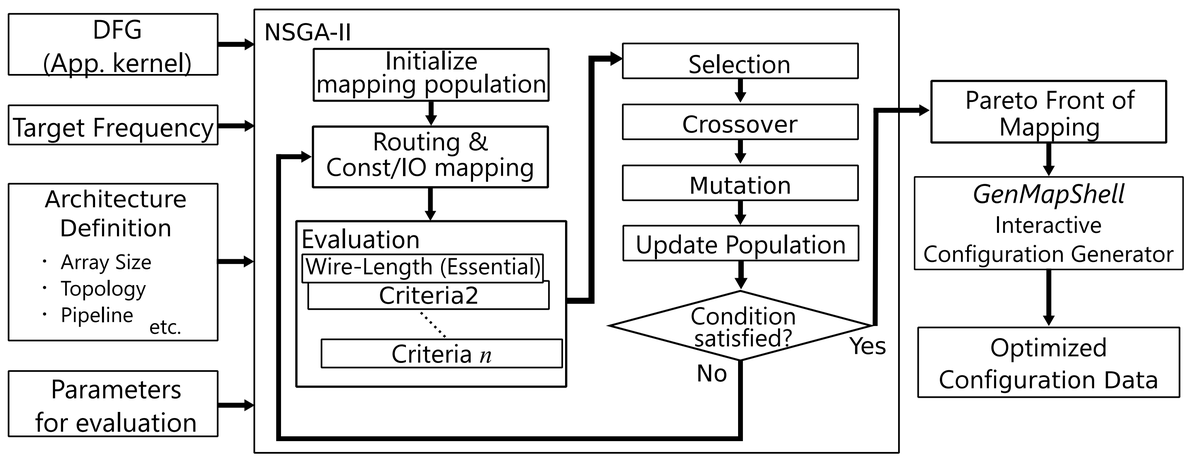
- **An architecture template**
  - For design space exploration
  - Targeting HPC workloads
- **A token-based CGRA**
  - FIFO buffers in each PE
  - Executing an operation in an ALU when required operands arrive
- **AG (Address generator)**
  - Handling data input/output to/from data memory

Our compilation & optimization toolchain for CGRAs



Details of GenMap

Optimization flow based on a genetic algorithm

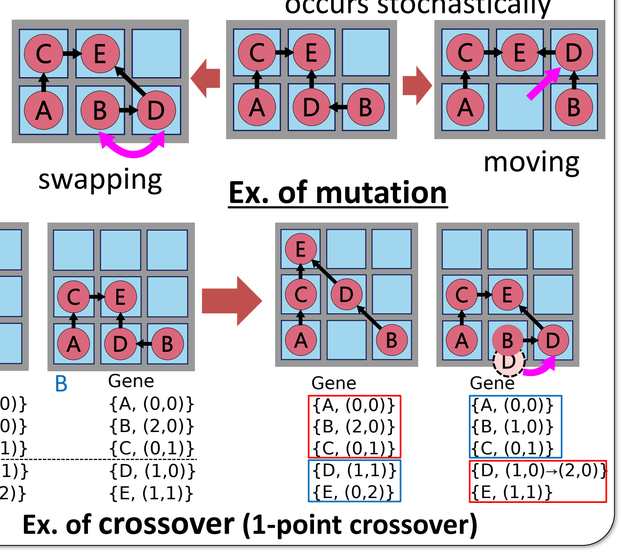


- Can be applied to various arch. with custom fitness func.
- Based on NSGA-II, a multi-objective GA

Genetic operations

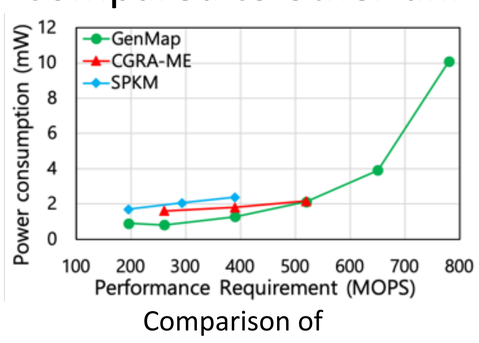
- Representing a mapping as chromosomes
  - Not including routing results
- Two types of genetic ops

  1). crossover & 2). mutation



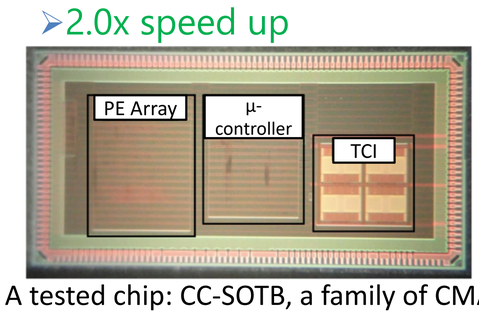
Case study: combining body bias optimization

- Body biassing can
  - boost computation performance at the cost of leakage
  - reduce leakage of PEs on non-critical paths
- Compared to other uni-objective mapping algorithms



Comparison of
power and maximum performance

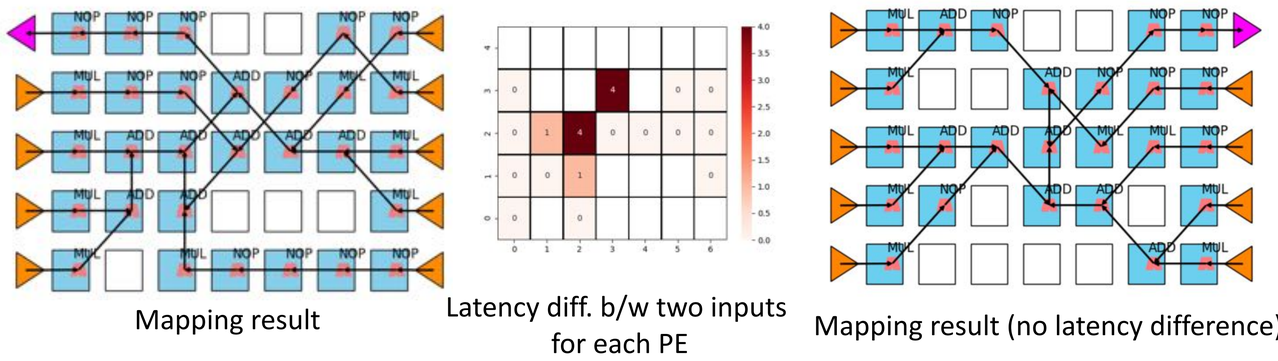
- 40+% energy reduction
- 2.0x speed up



A tested chip: CC-SOTB, a family of CMA

Case study: latency balancing for RHP-CGRA

- Targeting 3x3 convolution filter



Mapping result

Latency diff. b/w two inputs
for each PE

Mapping result (no latency difference)

Without optimization of latency balancing **With optimization of latency balancing**

- By adding a fitness function to evaluate maximum latency difference
  - A perfectly balanced DFG mapping can be obtained

- N.Ozaki, *et al.*, *IEEE Micro* 31.6 (2011), pp. 6-18.
- T.Kojima, *et al.*, *TVLSI* 28.11 (2020), pp. 2383-2396.
- A.Podobas, *et al., In Proc of ASAP 2020,* pp. 1-8
- A.Ohwada, *et al., In Proc of SASIMI 2021,* pp. 19-24.
- [GenMap open-source]: https://github.com/hungalab/GenMap