Compiler Framework for Spatial Mapping CGRA using LLVM

Keio Univ. JAPAN

Ayaka Ohwada, Takuya Kojima, and Hideharu Amano

Background: CGRA

Coarse-Grained Reconfigurable Architecture

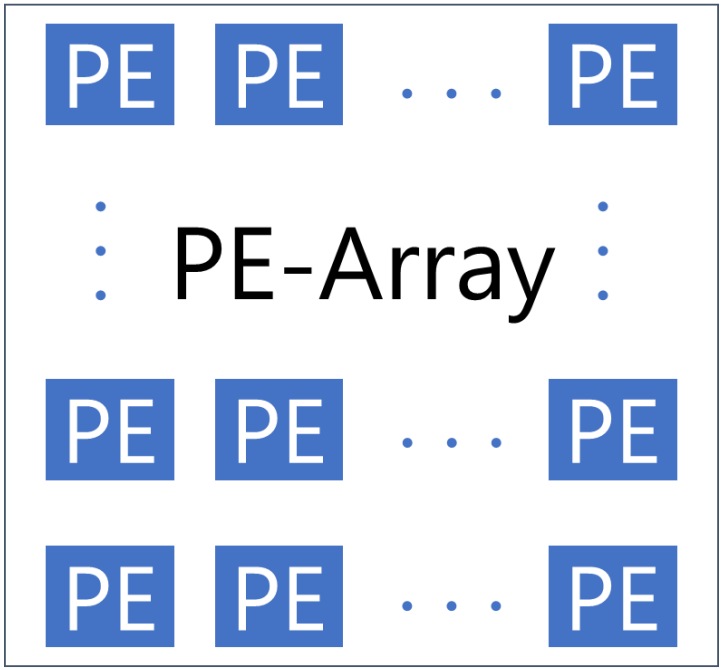
- **A kind of reconfigurable architecture**

- Has a PE array composed of multiple PEs (Processing Elements)

- Computation and interconnection of each PE are reconfigurable

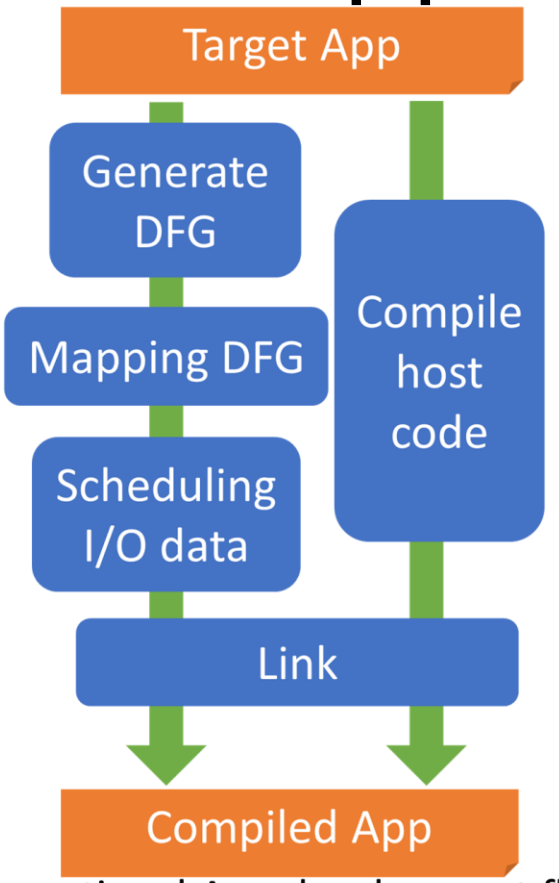
- Has **flexibility** and **high power efficiency**.

- Attracts attention with the spread of IoT devices.



PE Array

Background and Motivation: CGRA App Development Flow



Conventional App development flow

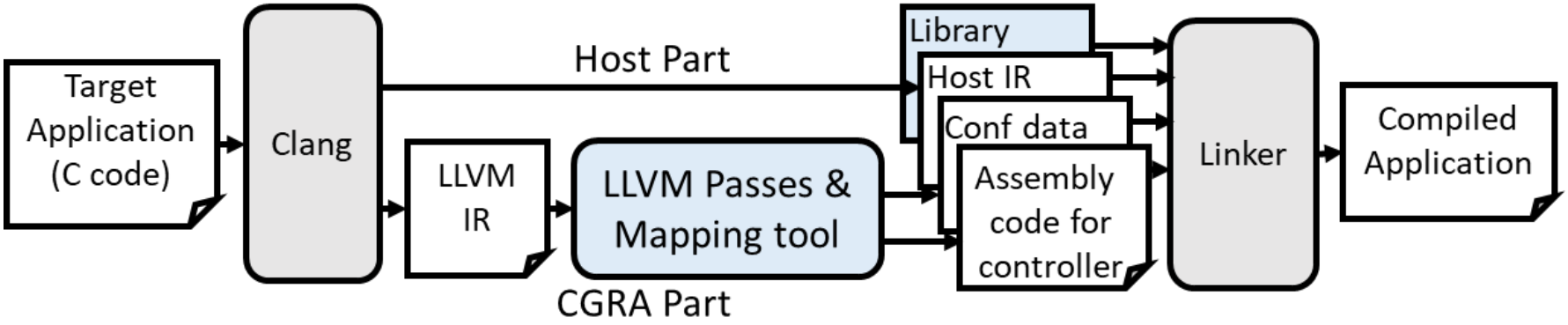
Each step is separate.
Some are manual.

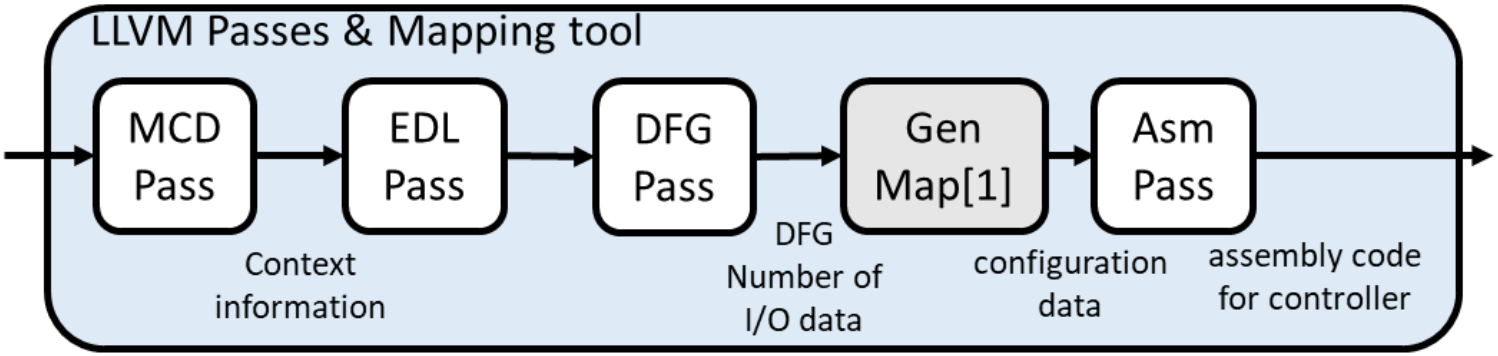
- automation and Integration of each process

Flow using the proposed tool

Our tool covers entire flow.

Proposed CGRA App Development Tool Flow





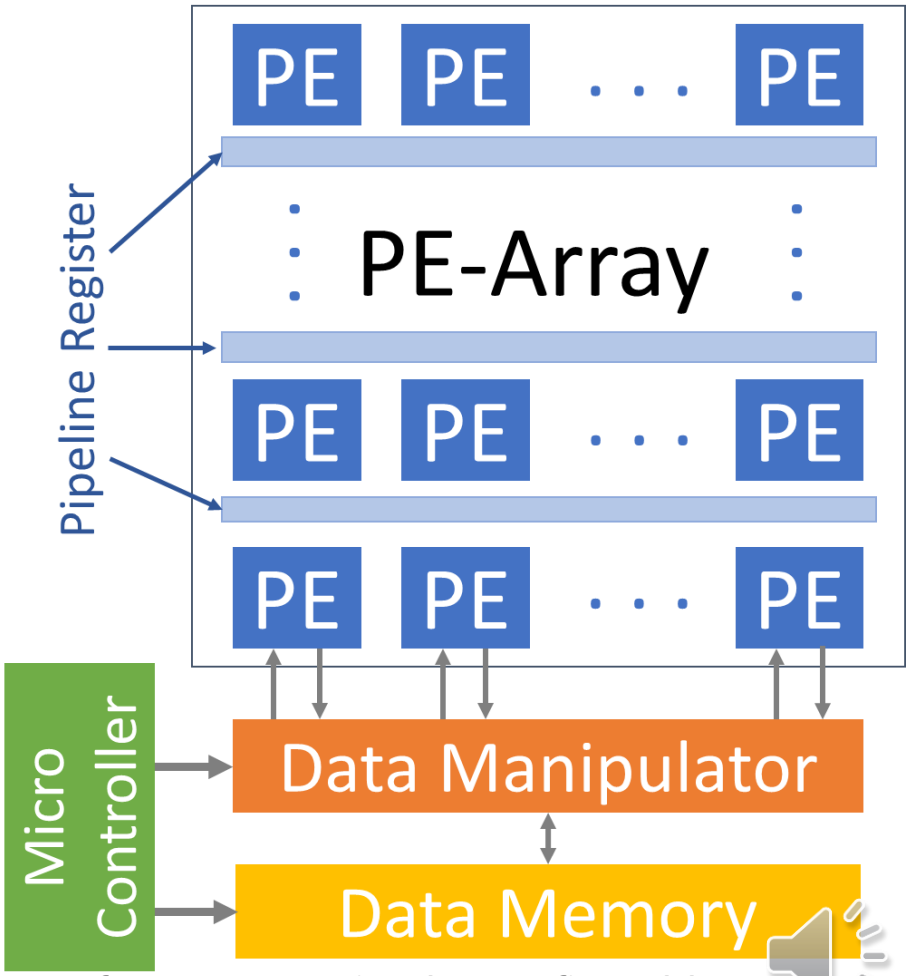
- Consists of multiple LLVM Pass.
- The input C code is processed separately into two parts, a CGRA part and a CPU part.

[1] Takuya Kojima, et al. "Real chip evaluation of a low power cgra with optimized application mapping," HEART 2018.

Case Study: VPCMA

Variable Pipelined Cool Mega Array[2]

- A kind of spatial mapping CGRA.
- Variable pipeline length.
- Data transfer is controlled by the micro controller.

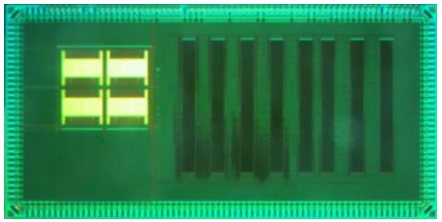


[2] Ando Naoki, et al. "Variable pipeline structure for coarse grained reconfigurable array CMA." FPT 2016.

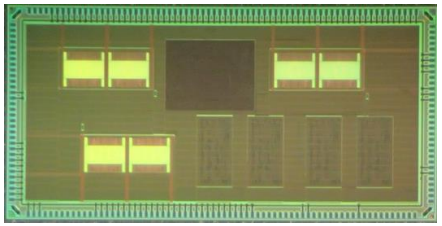
Evaluation

Evaluation Target

- **CCSOTB2@20MHz**
  - A real chip implementation of a VPCMA
- **GeyserTT@20MHz**
  - An embedded processor with a MIPS R3000 compatible CPU core.



CCSOTB2



GeyserTT

Evaluation Environment

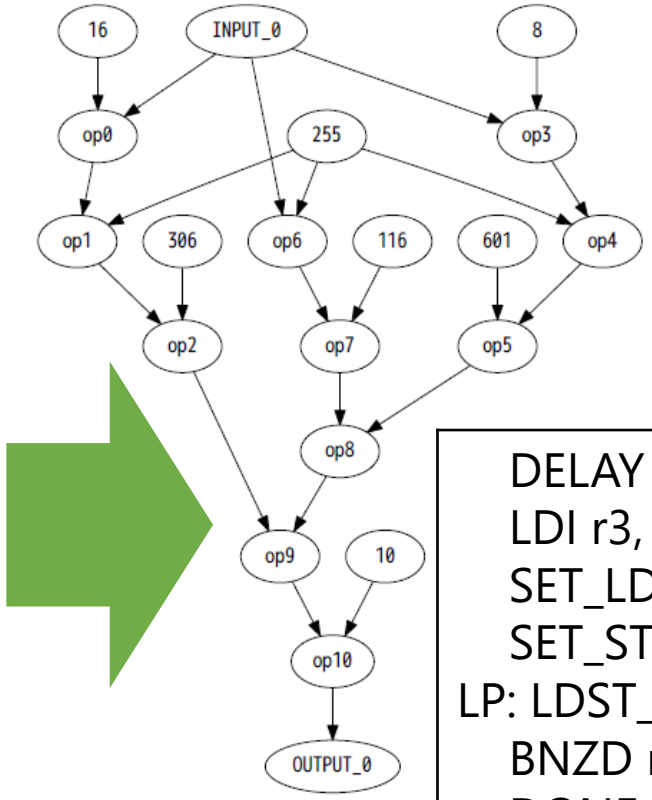
Design	Verilog HDL
Process	Renesas SOTB 65nm
Logic Synthesis	Synopsys Design Compiler 2016.03-SP4
Place and Route	Synopsys IC Compiler 2016.03-SP4
HDL Simulation	Cadence NC-Verilog 15.20-s020
Power-Supply Voltage	CCSOTB2:0.55V GeyserTT: 0.75V

Evaluation Result

- Four applications were chosen for the evaluation.
  - alpha: 24bit alpha blender
  - gray: 24bit gray scale
  - sepia: 24bit sepia filter
  - sf: 8bit sepia filter

```
void gray(int *in, int *out) {
    CGRA(1, 1);
    int i = 0;
    int maskr = 0x00ff0000;
    int maskg = 0x0000ff00;
    int maskb = 0x000000ff;

    out [i] = (((input[i] & maskr) >> 16) *
    306+ ((in [i] & maskg) >> 8) * 601 +
    (in [i] & maskb) * 117>> 10;
}
```

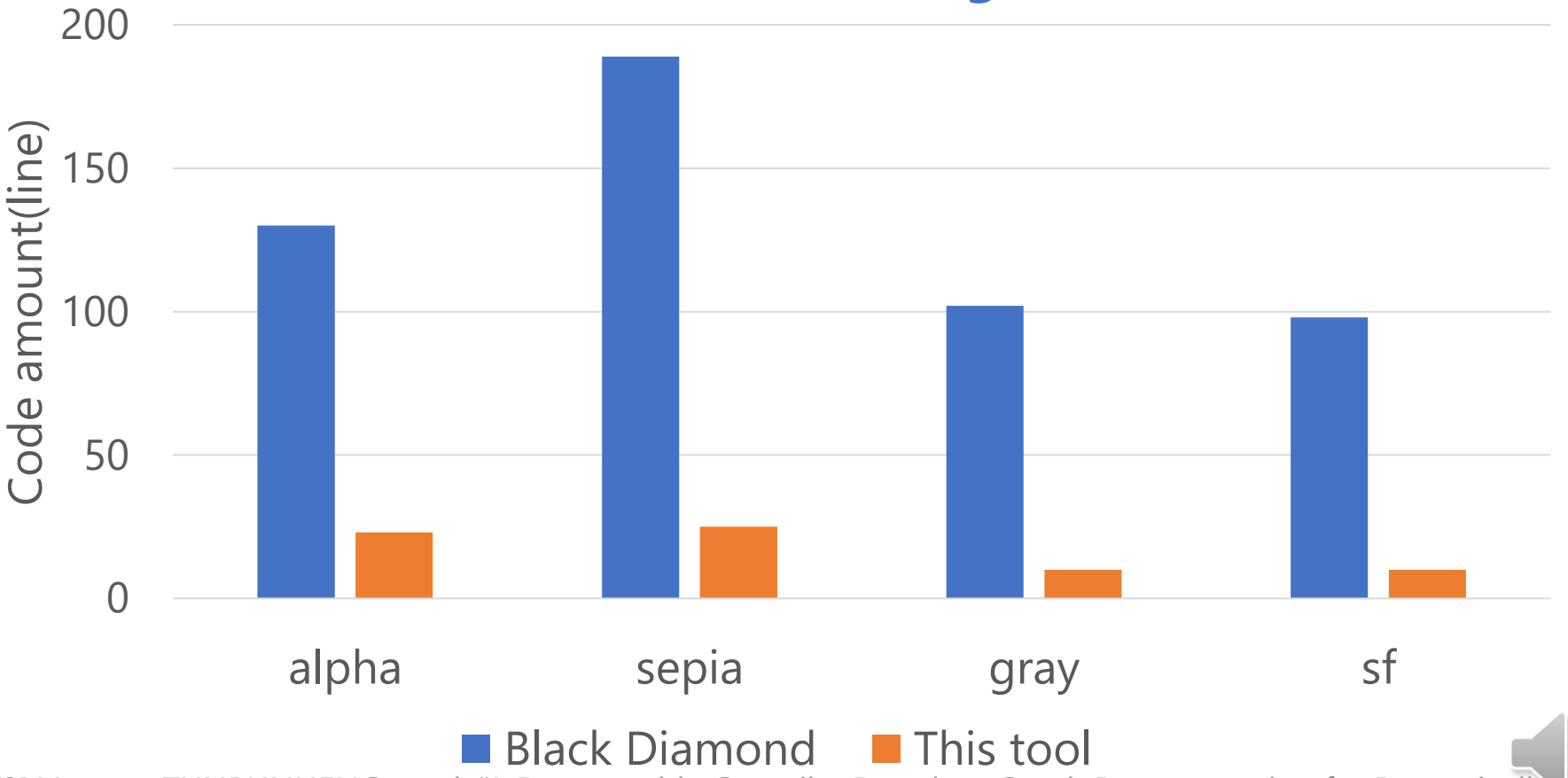



```
    DELAY 7
    LDI r3, #48
    SET_LD #0x0, #1
    SET_ST #0x30, #1
LP: LDST_ADD #0, #0
    BNZD r3, LP
    DONE
```

Evaluation: Code Amount

- Compared to the case described for Black Diamond compiler [3].
- Compare only the kernel part of the user-written code

87.2% code reduction on average.

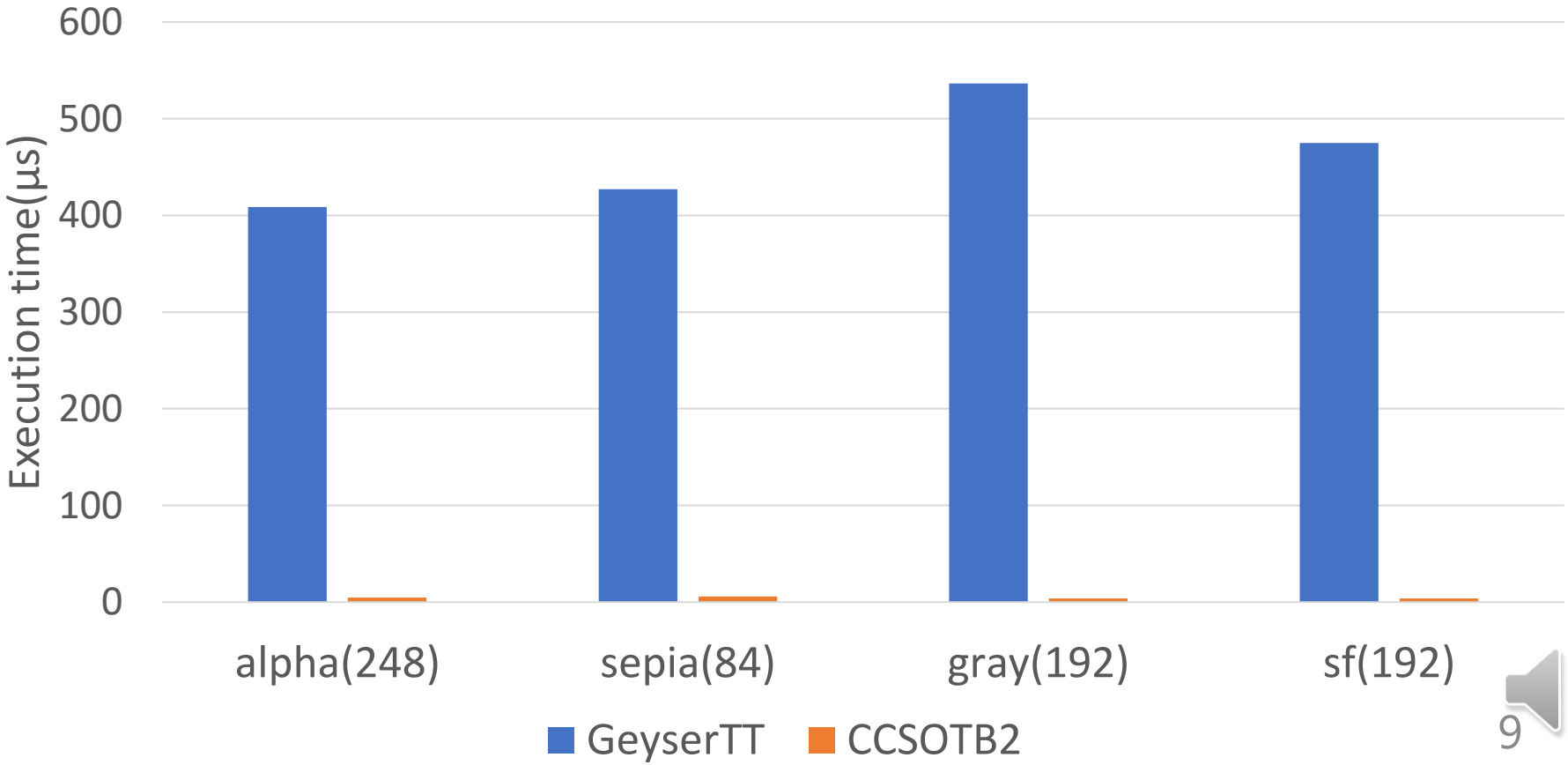


[3] Vasutan TUNBUNHENG, et al. "A Retargetable Compiler Based on Graph Representation for Dynamically Reconfigurable Processor Arrays", I EI CE Transactions on Information and Systems, 2008.

Evaluation: Execution Time

- Execution time is defined as "time required for calculation"
- The value in parentheses indicates the number of operation data.

99% reduction in execution time on average.



Conclusion

Implementation of CGRA app development tool using LLVM.

- Apps written in C language can be compiled into a form that can be executed on CGRA.
- Simultaneously generates code that can be executed on the CGRA side and the CPU side from the input C code.

- Run 4 applications on simulation
  - CCSOTB2 and GeyserTT
  - **All applications were confirmed to work properly.**

- The average code amount is reduced by **87.2%** compared to the conventional kernel description method.