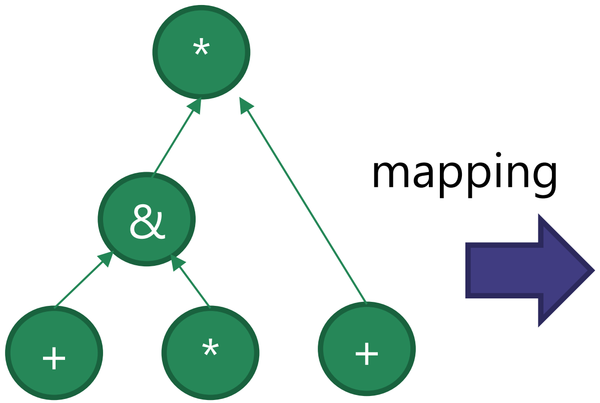
An Architecture-Independent CGRA Compiler enabling OpenMP Applications

Takuya Kojima[†], Carlos Cesar Cortes Torres[‡], Boma Adhi[‡], Yiyu Tan[‡], Kentaro Sano[‡]

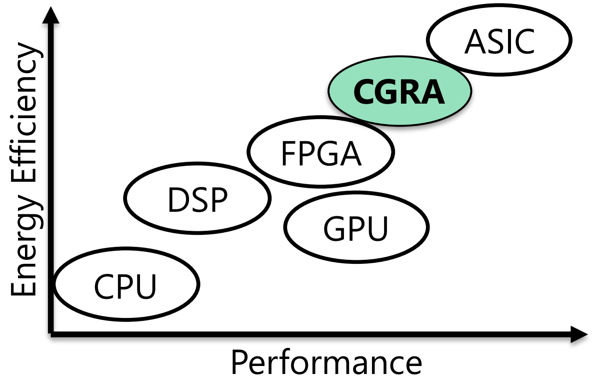
[†] The University of Tokyo, Japan

[‡] RIKEN, Japan

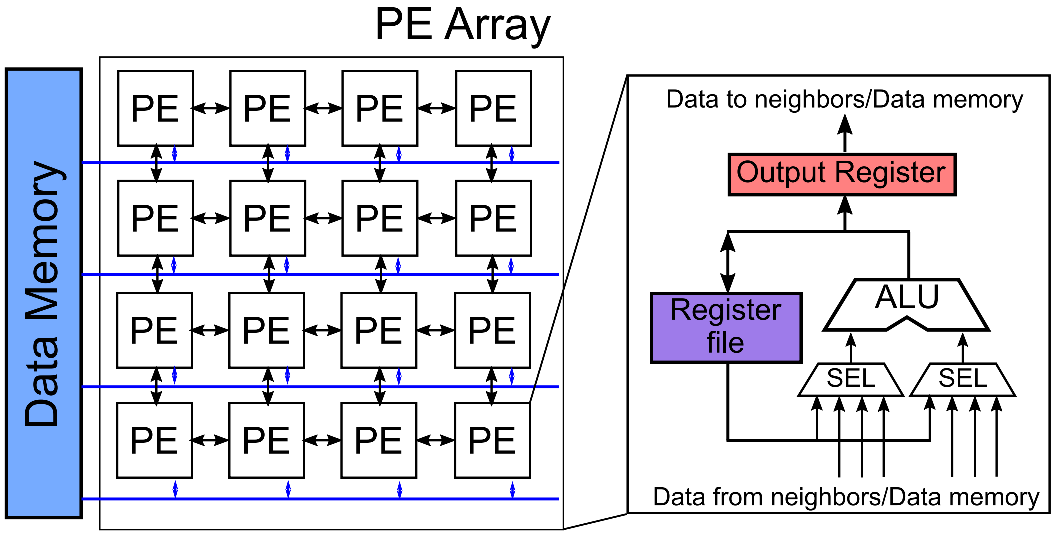
Coarse-grained reconfigurable architectures



A data flow graph
as an application



Comparison with other architectures[2]


PE Array
Data Memory
PE
Data to neighbors/Data memory
Output Register
ALU
Register file
SEL
SEL
Data from neighbors/Data memory

General structure of the CGRAs

- Coarse-Grained Reconfigurable Architecture (CGRA)
  - Composed of an array of Processing Elements (PEs)
  - Providing a word-level reconfigurability (e.g., 32-bit)
    - Smaller energy-overhead than FPGAs (bit-level)
  - Generally used as an accelerator

[2] Liu, Leibo, et al. "A survey of coarse-grained reconfigurable architecture and design: Taxonomy, challenges, and applications." *ACM Computing Surveys (CSUR)* 52.6 (2019): 1-39.

Purpose & Proposal

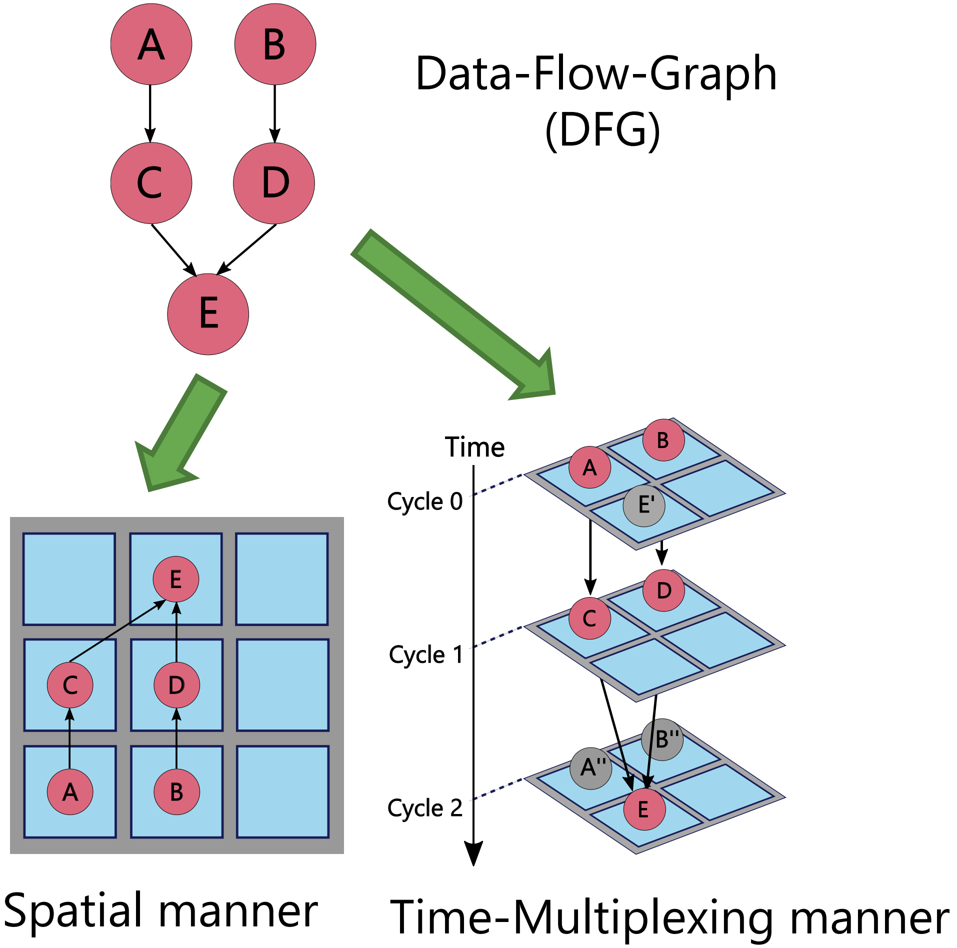
- Trend in CGRA research: design space exploration framework
  - Highly customizability
  - Possibilities of domain-specific architecture
  - Recent work: CGRA-ME [3], OpenCGRA [4], RIKEN CGRA [5]
    DSAGEN [6] (ISCA 2020), SNAFU [7] (ISCA 2021)
- Challenge
  - No general-purpose & architecture-independent compiler frontend for CGRAs
  - Needs of abstracting hardware layer for software programmers
- Our proposal
  - A compiler framework enabling OpenMP *offloading* to CGRAs
  - A case study: implementation for RIKEN CGRA

Wide Variety of Design Choices

- Characteristics of CGRA Design
  - Reconfiguration style
  - PE array size
  - Interconnection topology
  - **Operational capabilities in PE**
  - **Ability to handle control flows**

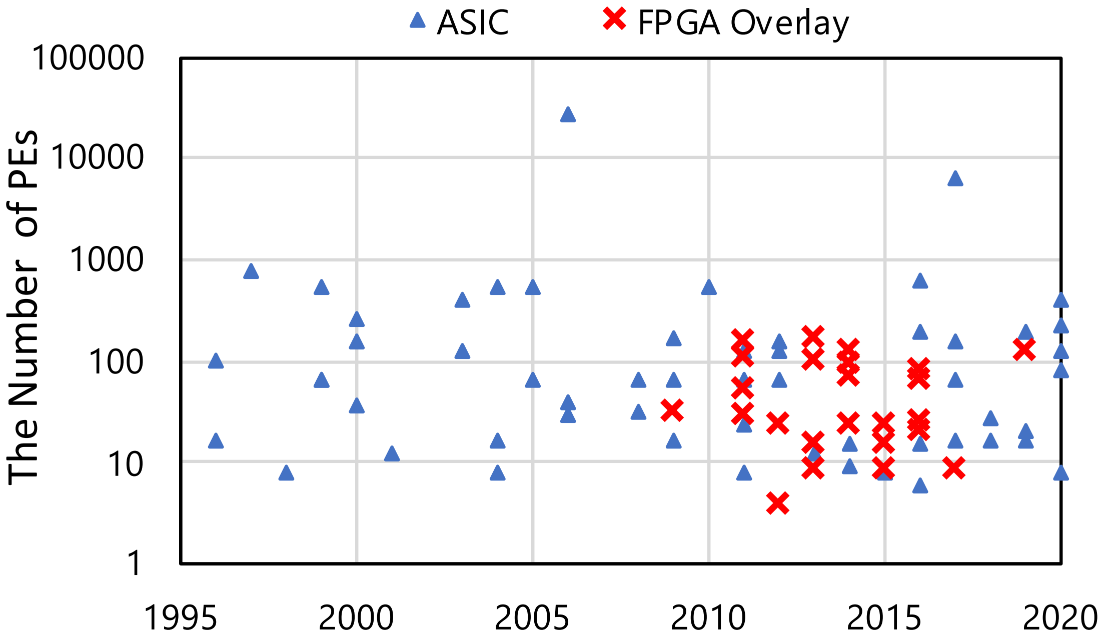
Wide Variety of Design Choices

- Characteristics of CGRA Design
  - Reconfiguration style
    - Time-Multiplexing manner
    - Spatial one
  - PE array size
  - Interconnection topology
  - Operational capabilities in PE
  - Ability to handle control flows



Wide Variety of Design Choices

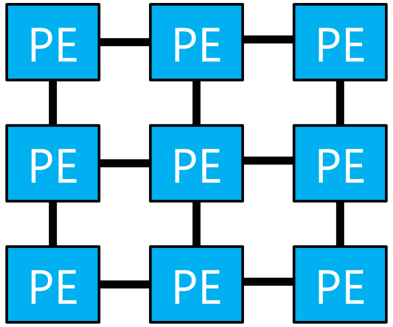
- Characteristics of CGRA Design
  - Reconfiguration style
  - **PE array size**
    - Ranges 10-10^4
  - Interconnection topology
  - Operational capabilities in PE
  - Ability to handle control flows



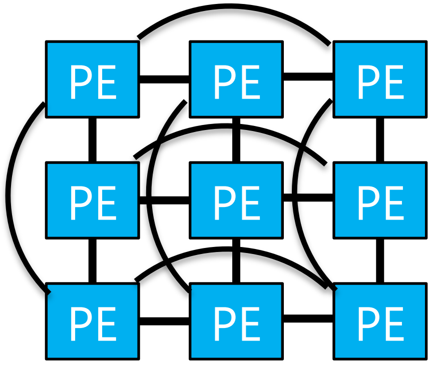
Distribution of the PE array size [8]

Wide Variety of Design Choices

- Characteristics of CGRA Design
  - Reconfiguration style
  - PE array size
  - <u>Interconnection topology</u>
    - Mesh
    - Meshplus
    - Torus, etc
  - Operational capabilities in PE
  - Ability to handle control flows



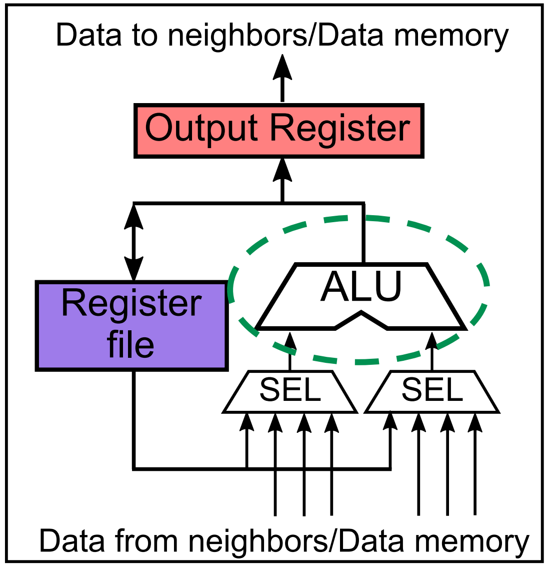
mesh



mesh-plus

Wide Variety of Design Choices

- Characteristics of CGRA Design
  - Reconfiguration style
  - PE array size
  - Interconnection topology
  - ➢ <u>Operational capabilities in PE</u>
    - Bit-width
    - Floating point
    - SIMD
    - Custom instruction (e.g., ReLU, sigmoid for ML)
  - Ability to handle control flows


Data to neighbors/Data memory
Output Register
Register file
ALU
SEL
SEL
Data from neighbors/Data memory

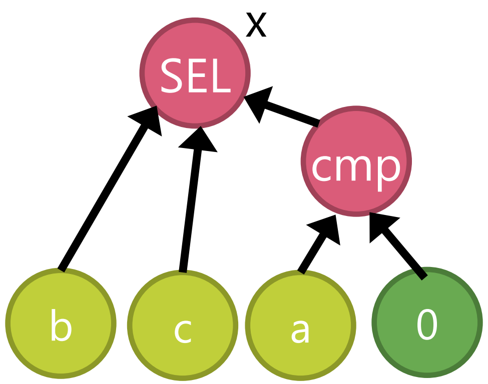
32bit? 64bit?
int? float? fixed?

Wide Variety of Design Choices

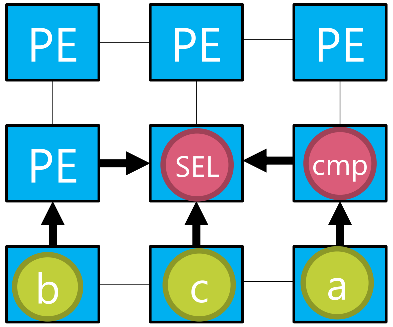
- Characteristics of CGRA Design
  - Reconfiguration style
  - PE array size
  - Interconnection topology
  - Operational capabilities in PE
  - Ability to handle control flows
    - Conditionals are supported or not ?
    - Loop-carried dependence is allowed or not ?

```
for (...) {
  if (a > 0) {
    x = b;
  } else {
    x = c;
  }
}
```

Loop containing conditional parts

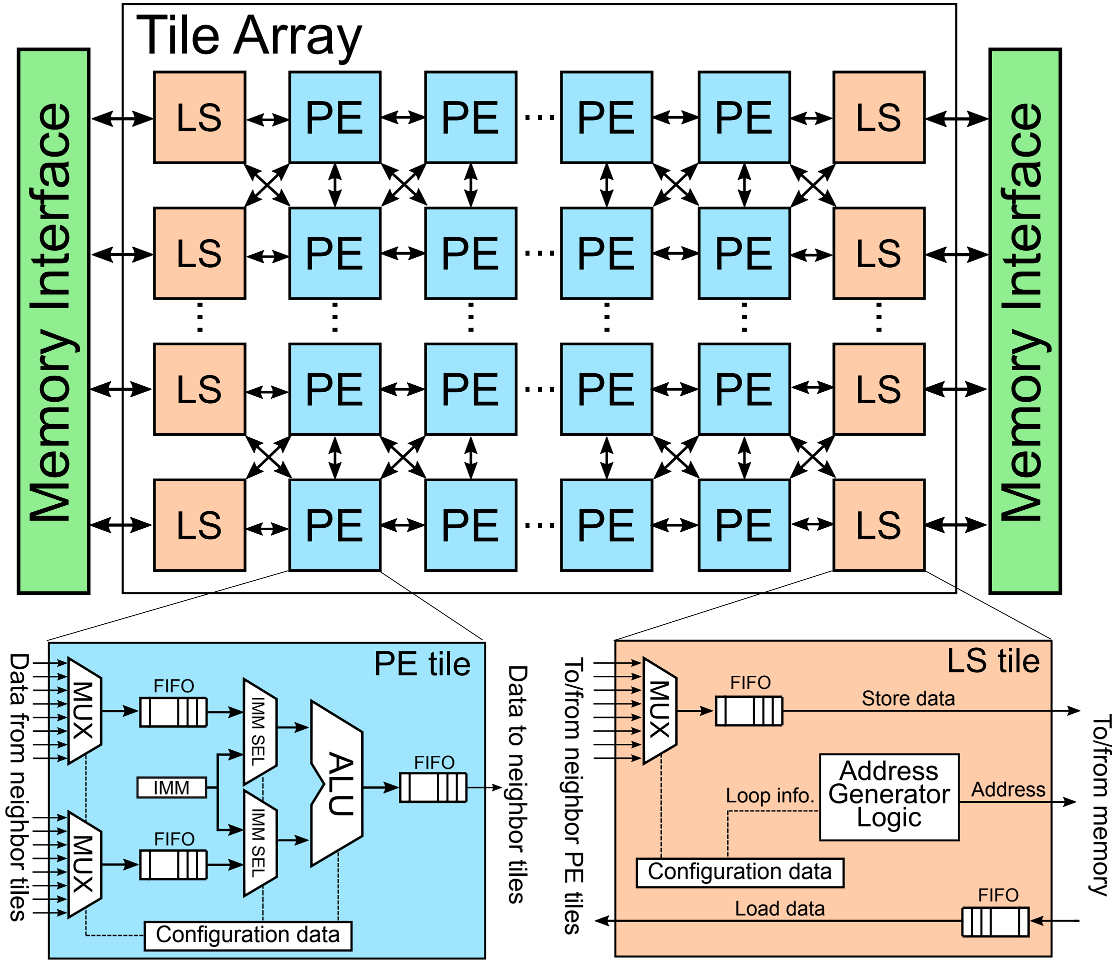


DFG with Partial Predication



Partial predication on CGRAs

A case of CGRA: RIKEN CGRA [5]

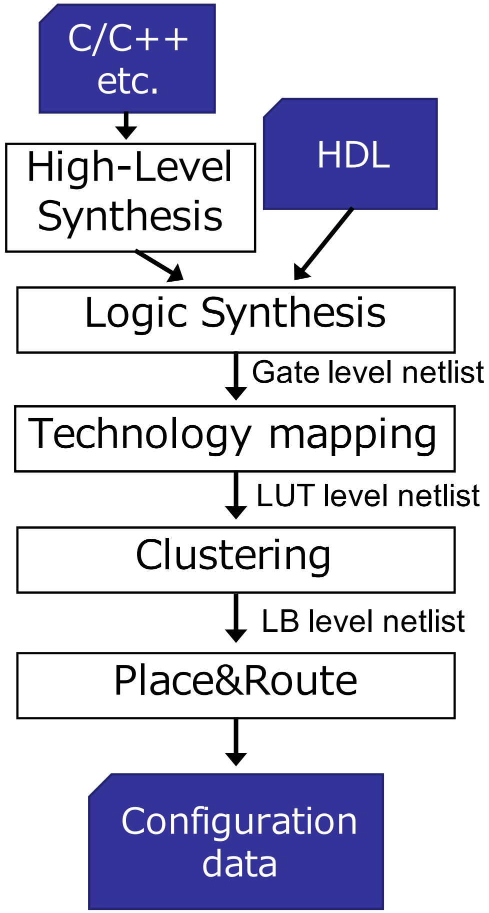


Overview of RIKEN CGRA

- Design template for design space exploration
  - Implemented with SystemVerilog
- Two types of tiles
  1. LS (Load/Store)
     - Data access according to loop control info.
  2. PE
     - Computation
- Reconfiguration style: Spatial
  - FIFO buffers allow operands to arrive at different times

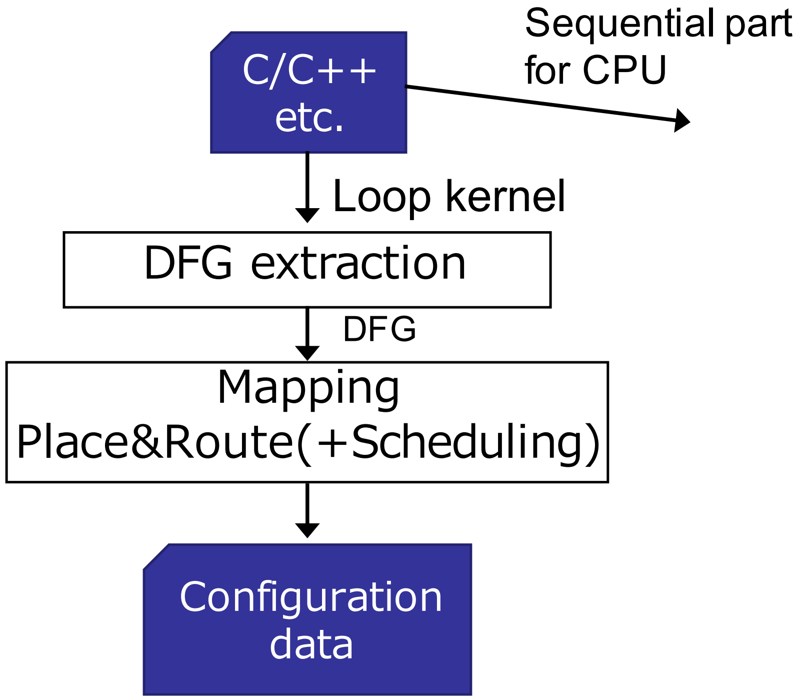
Limitations of Existing Compilers

CGRA vs FPGA in compilation flow



For FPGAs

Coarse granularity mitigates compilation complexity



For CGRAs

Existing compilers for CGRAs

Methods	Frontend	Targets	
		Architecture	Reconf.
Musketeer (STP Tool) [9]	C	Renesas STP (DRP)	TM
PipeRench Compiler [10]	C-like DSL (DIL)	PipeRench	SP (TM)
PACT XPP-VC [11]	C	PACT XPP	TM
SambaFlow™ [12]	PyTorch, TensorFlow, etc	SambaNova RDA	SP
BlackDiamond [13]	C-like DSL	Parameterized	TM/SP
CCF [14]	C (pragma)	ADRES[16]-like	TM
Kim, Hee-Seok, *et al* [15]	OpenCL	SRP	TM
MENTAI [16]	C	Cool Mega Array	SP
CGRA-ME [3]	C	Parameterized	TM/SP
OpenCGRA [4]	C, Python DSL	Parameterized	TM
DSAGEN [6]	C (pragma)	Parameterized	SP

Commercial products

TM: Time-Multiplexing, SP: Spatial

Existing compilers for CGRAs

Methods	Frontend	Targets	
		Architecture	Reconf.
Musketeer (STP Tool) [9]	C	Renesas STP (DRP)	TM
PipeRench Compiler [10]	C-like DSL (DIL)	PipeRench	SP (TM)
PACT XPP-VC [11]	C	PACT XPP	TM
SambaFlow™ [12]	PyTorch, TensorFlow, etc	SambaNova RDA	SP
BlackDiamond [13]	C-like DSL	Parameterized	TM/SP
CCF [14]	C (pragma)	ADRES[16]-like	TM
Kim, Hee-Seok, *et al* [15]	OpenCL	SRP	TM
MENTAI [16]	C	Cool Mega Array	SP
CGRA-ME [3]	C	Parameterized	TM/SP
OpenCGRA [4]	C, Python DSL	Parameterized	TM
DSAGEN [6]	C (pragma)	Parameterized	SP
This work	**OpenMP**	**Parameterized**	**TM/SP**

Commercial products

TM: Time-Multiplexing, SP: Spatial

Existing compilers for CGRAs

Methods	Frontend	Targets	
		Architecture	Reconf.
Musketeer (STP Tool) [9]	C	Renesas STP (DRP)	TM
PipeRench Compiler [10]	C-like DSL (DIL)	PipeRench	SP (TM)
PACT XPP-VC [11]	C	PACT XPP	TM
SambaFlow^{TM} [12]	PyTorch, TensorFlow, etc	SambaNova RDA	SP
[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]
Ki[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]
DSAGEN [6]	C (pragma)	Parameterized	SP
This work	**OpenMP**	**Parameterized**	**TM/SP**

Commercial products

Design space exploration
Fair comparison between various types of CGRAs

Reuse of source codes
Minimizing efforts to modify the codes

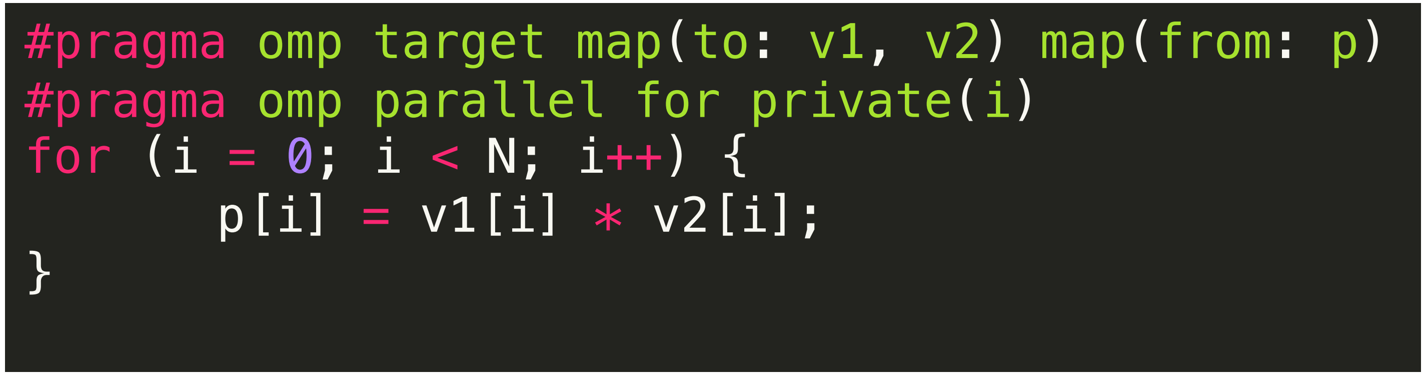
Easy to compare other architectures
Comparing to GPU, many-core CPU with the same kernel

TM: Time-Multiplexing, SP: Spatial

Our Proposal: CGRA OpenMP

Target directive

- Accelerator offloading features
    - Added since OpenMP 4.0
    - Similar concept to OpenACC
    - Mainly supporting GPU offloading
    - Explicit data transfer between hosts and device (map clause)

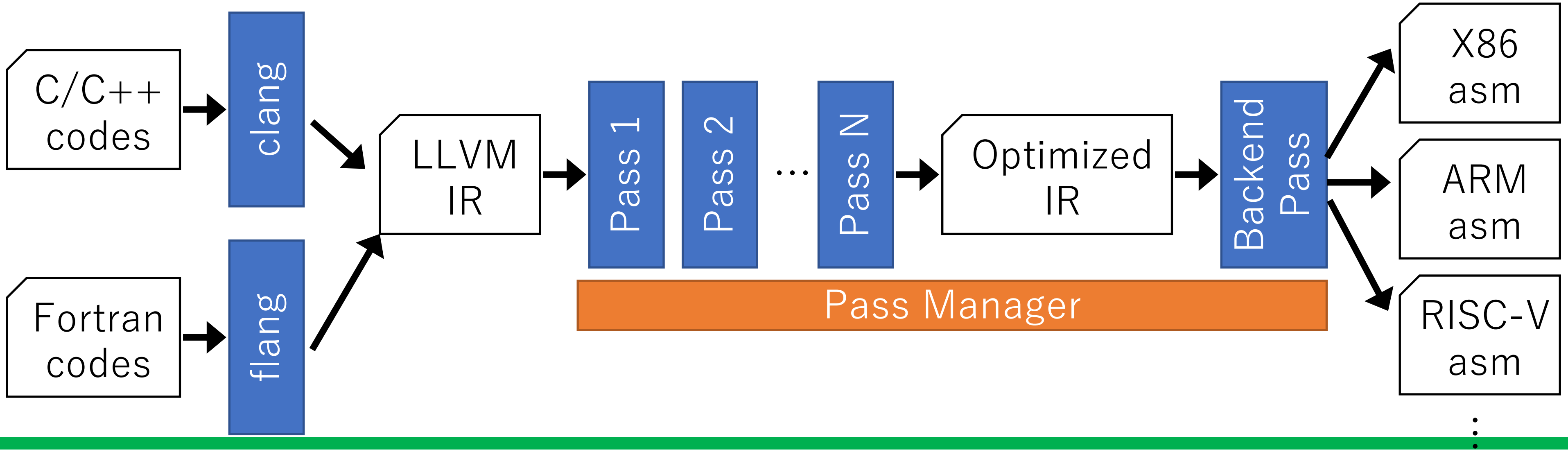


```
#pragma omp target map(to: v1, v2) map(from: p)
#pragma omp parallel for private(i)
for (i = 0; i < N; i++) {
      p[i] = v1[i] * v2[i];
}
```

Code snippet with target directive [17]

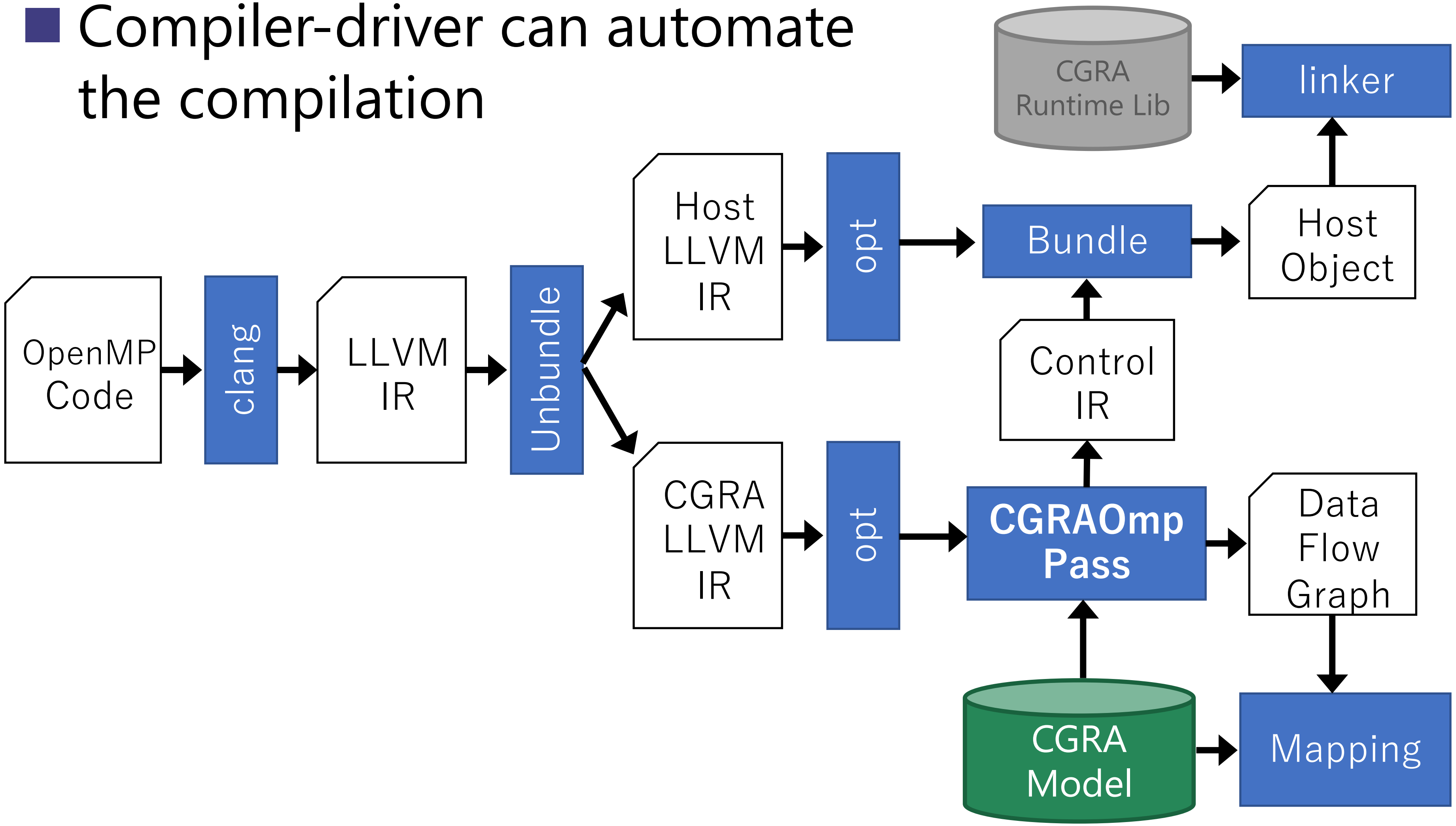
Implementation based on LLVM

- LLVM: An open-source compiler framework
  - LLVM-IR: target-independent intermediate representation
  - Common optimization and analysis algorithms (Pass)
  - Official sub-projects
    - C frontend Clang, Fortran frontend Flang, OpenMP, etc



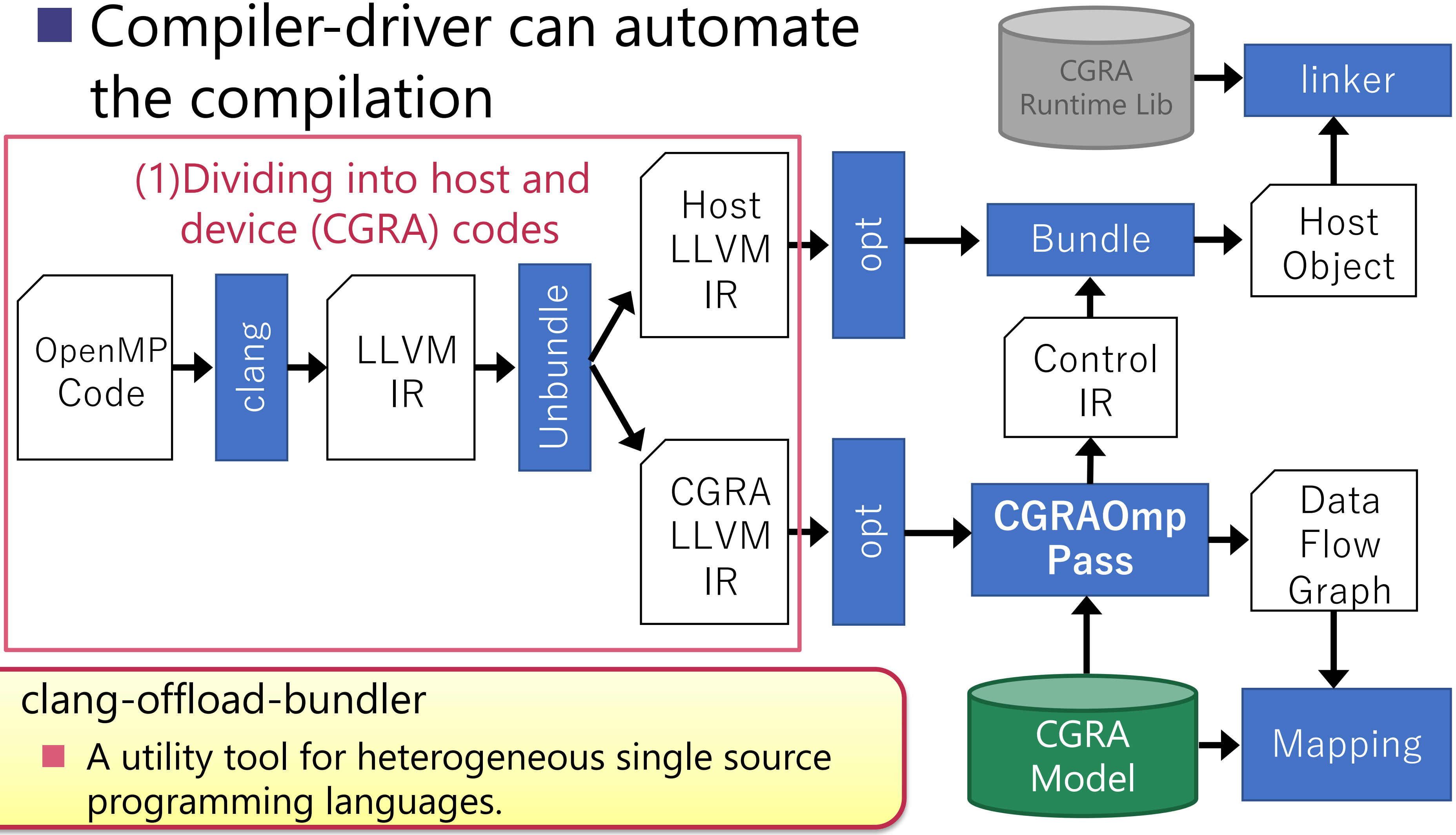
Compilation flow in CGRA OpenMP

- Compiler-driver can automate the compilation



Compilation flow in CGRA OpenMP

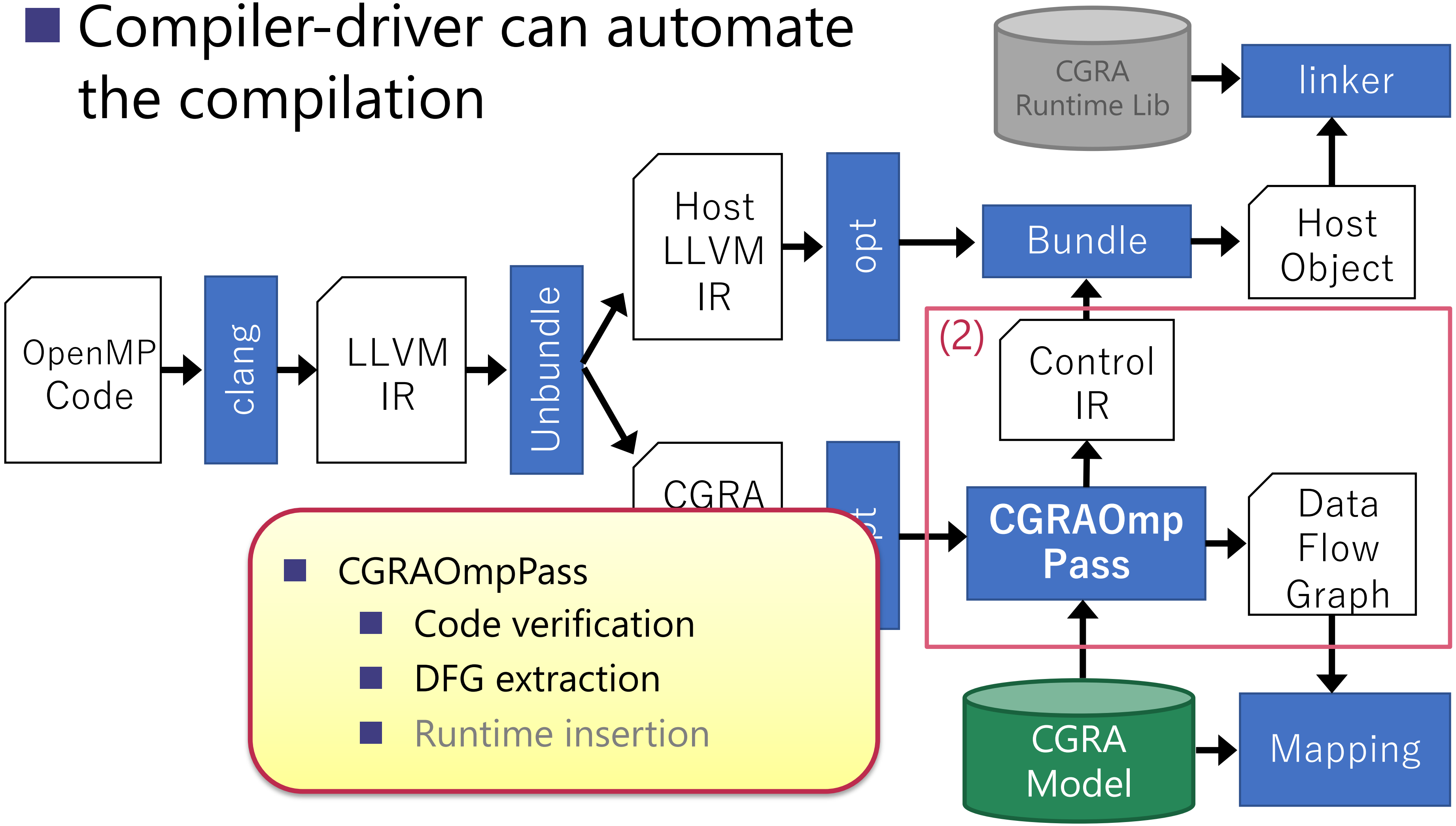
- Compiler-driver can automate the compilation



- clang-offload-bundler
  - A utility tool for heterogeneous single source programming languages.

Compilation flow in CGRA OpenMP

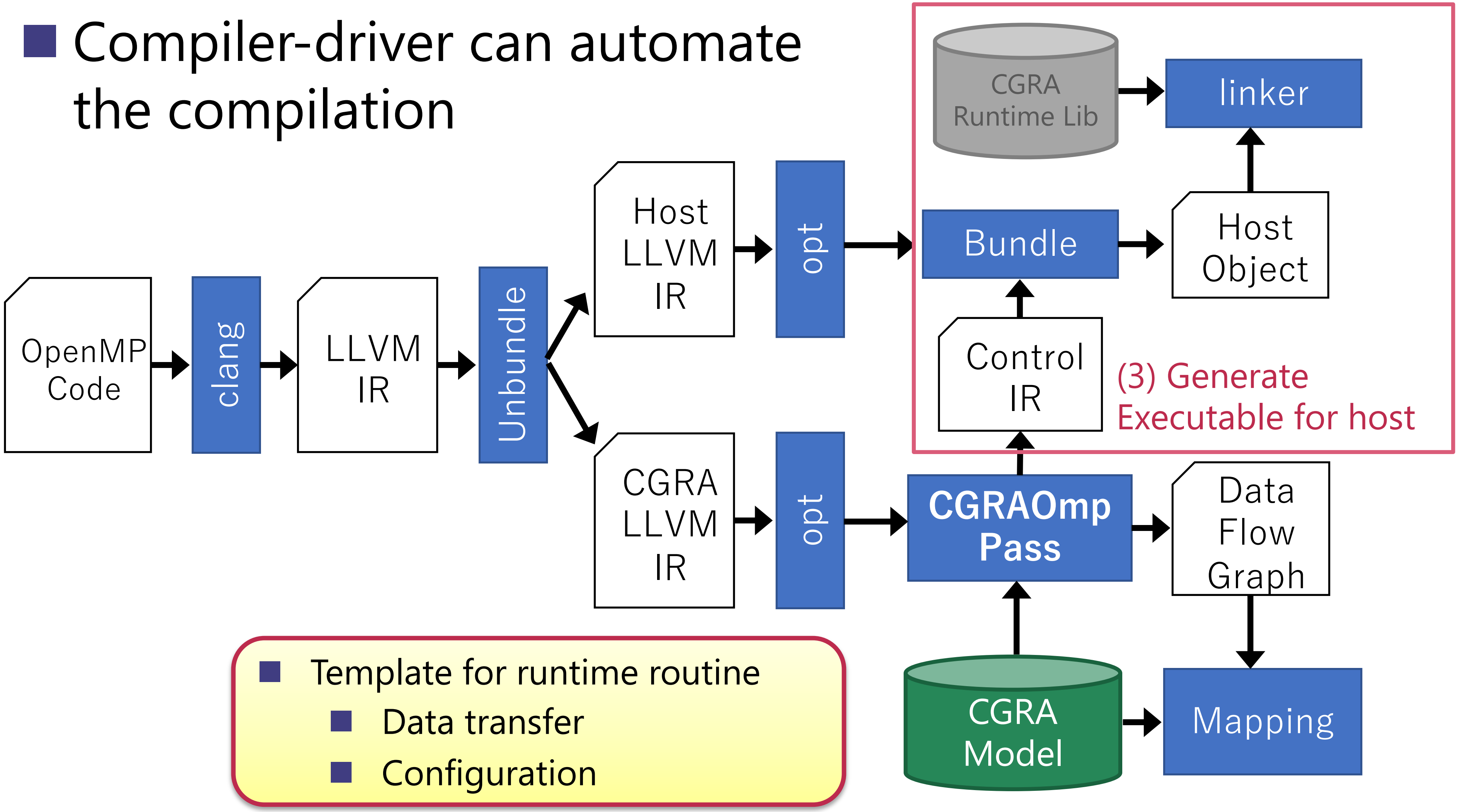
- Compiler-driver can automate the compilation



- CGRAOmpPass
  - Code verification
  - DFG extraction
  - Runtime insertion

Compilation flow in CGRA OpenMP

- Compiler-driver can automate the compilation

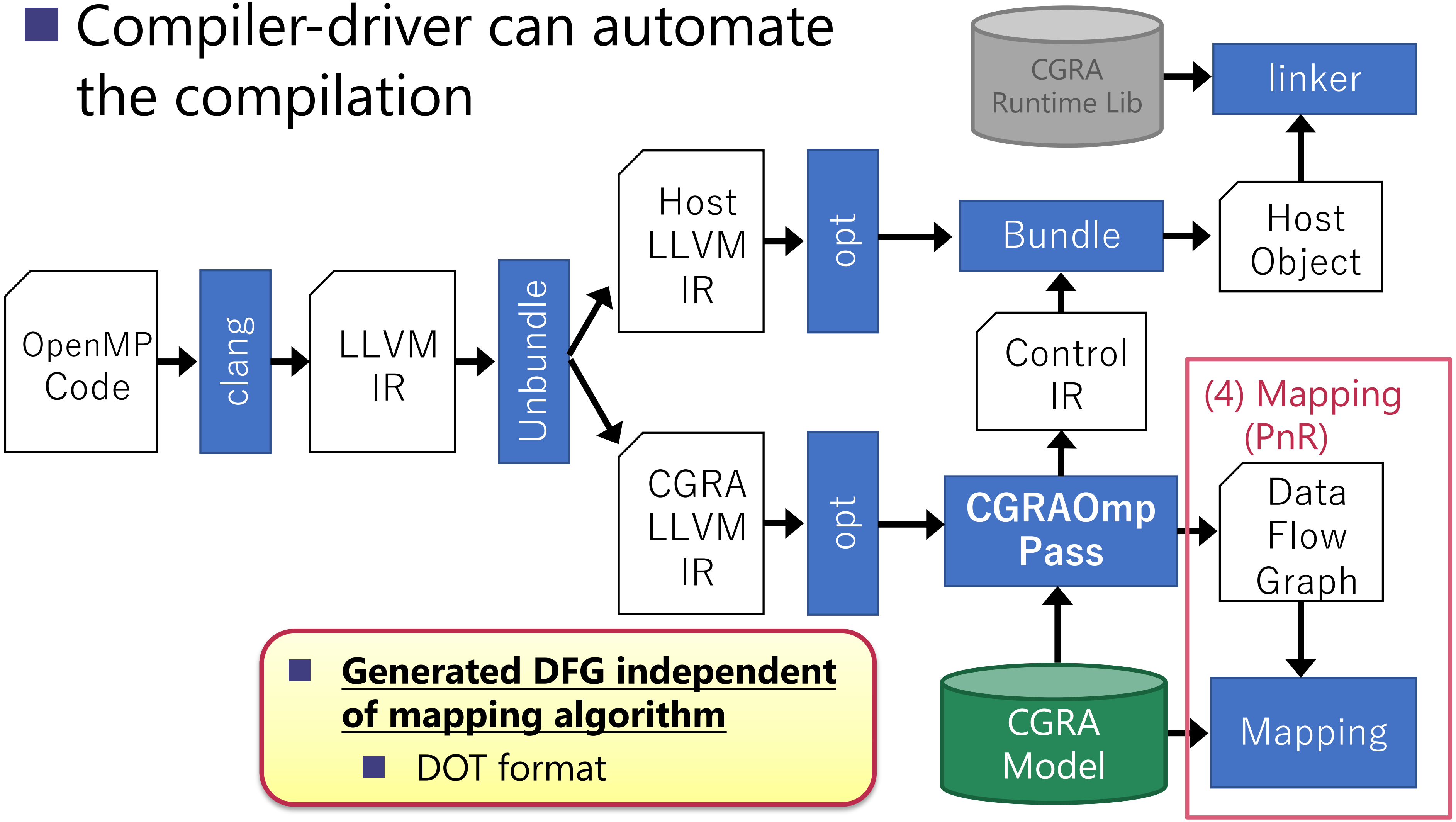


(3) Generate Executable for host

- Template for runtime routine
  - Data transfer
  - Configuration

Compilation flow in CGRA OpenMP

- Compiler-driver can automate the compilation

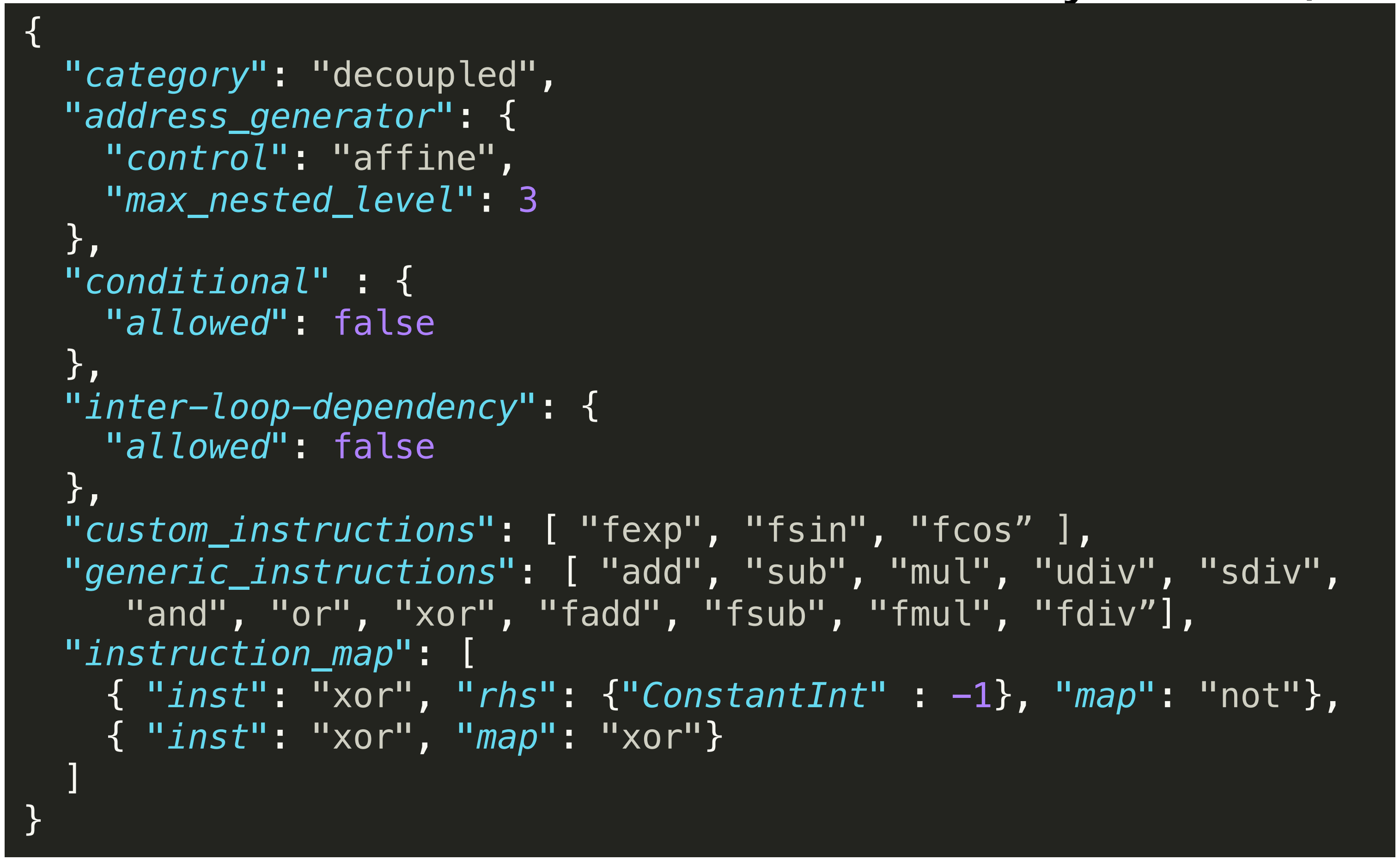


- **Generated DFG independent of mapping algorithm**
  - DOT format

CGRA Model Description

CGRA model description

- The model defines CGRA execution style, etc (JSON)

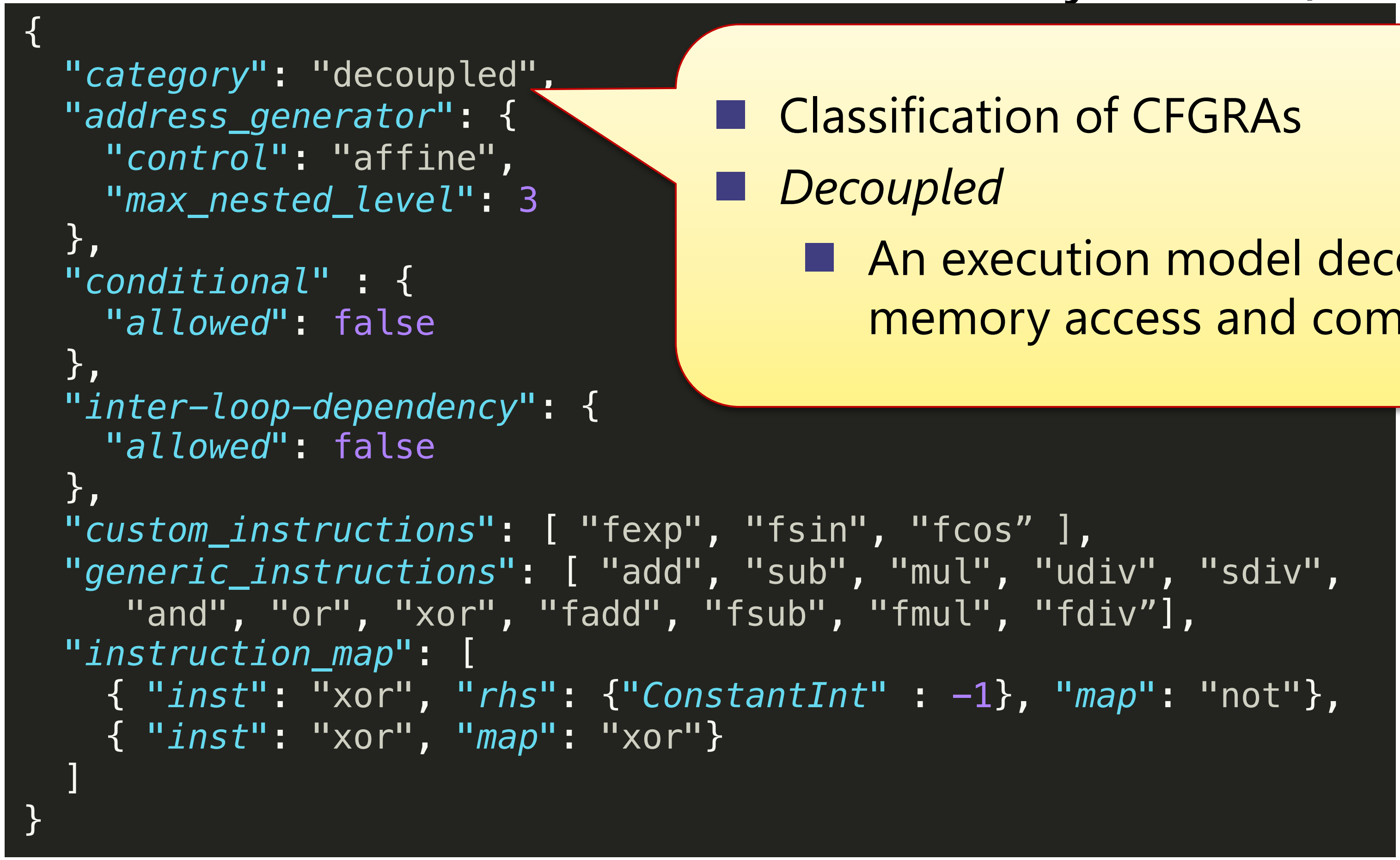


```
{
  "category": "decoupled",
  "address_generator": {
    "control": "affine",
    "max_nested_level": 3
  },
  "conditional" : {
    "allowed": false
  },
  "inter-loop-dependency": {
    "allowed": false
  },
  "custom_instructions": [ "fexp", "fsin", "fcos” ],
  "generic_instructions": [ "add", "sub", "mul", "udiv", "sdiv",
     "and", "or", "xor", "fadd", "fsub", "fmul", "fdiv”],
  "instruction_map": [
    { "inst": "xor", "rhs": {"ConstantInt" : -1}, "map": "not"},
    { "inst": "xor", "map": "xor"}
  ]
}
```

An example: the case of RIKEN CGRA

CGRA model description

- The model defines CGRA execution style, etc (JSON)



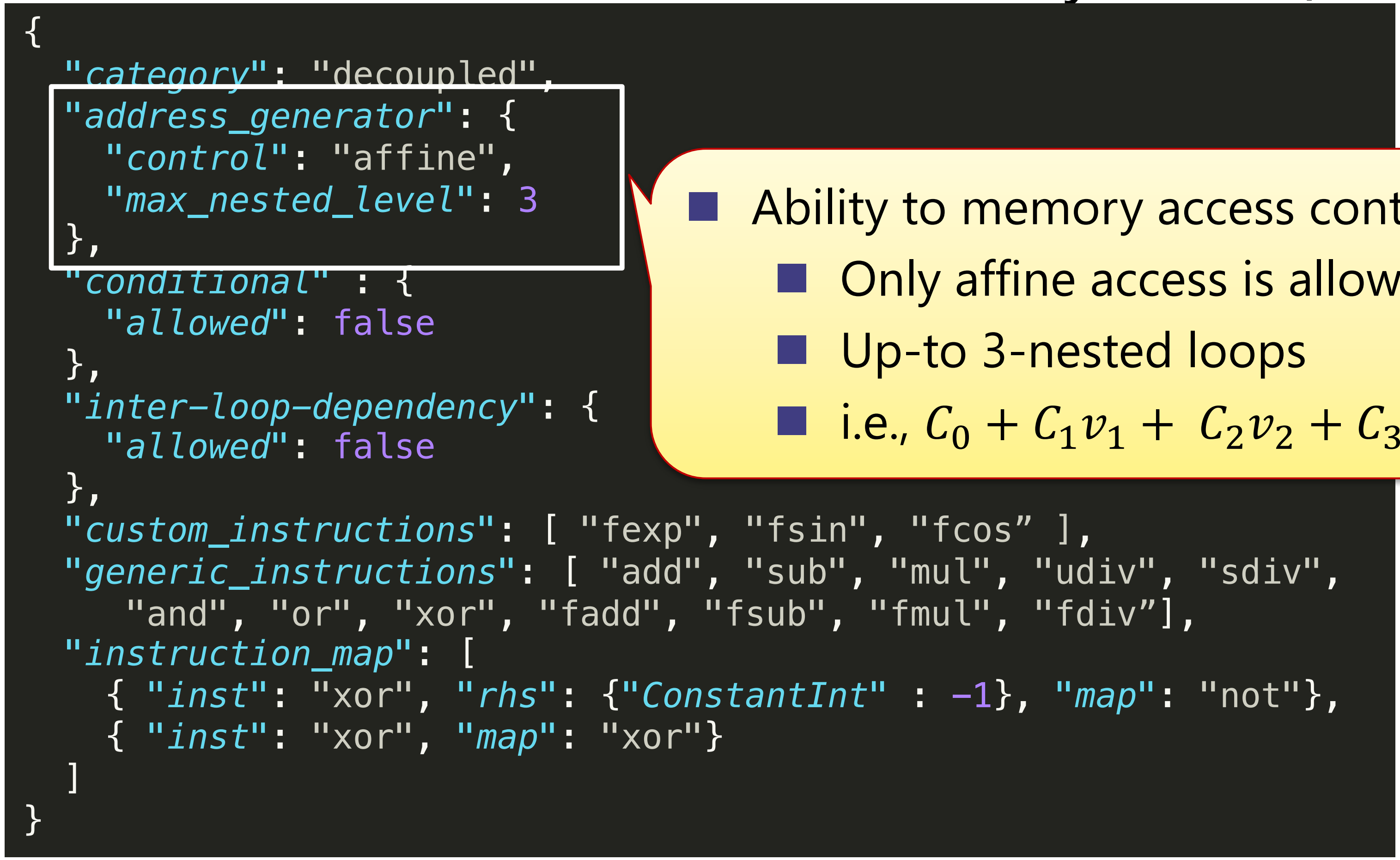
```
{
  "category": "decoupled",
  "address_generator": {
    "control": "affine",
    "max_nested_level": 3
  },
  "conditional" : {
    "allowed": false
  },
  "inter-loop-dependency": {
    "allowed": false
  },
  "custom_instructions": [ "fexp", "fsin", "fcos” ],
  "generic_instructions": [ "add", "sub", "mul", "udiv", "sdiv",
     "and", "or", "xor", "fadd", "fsub", "fmul", "fdiv”],
  "instruction_map": [
    { "inst": "xor", "rhs": {"ConstantInt" : -1}, "map": "not"},
    { "inst": "xor", "map": "xor"}
  ]
}
```

- Classification of CFGRAs
- *Decoupled*
  - An execution model decoupling memory access and computation [6]

An example: the case of RIKEN CGRA

CGRA model description

- The model defines CGRA execution style, etc (JSON)



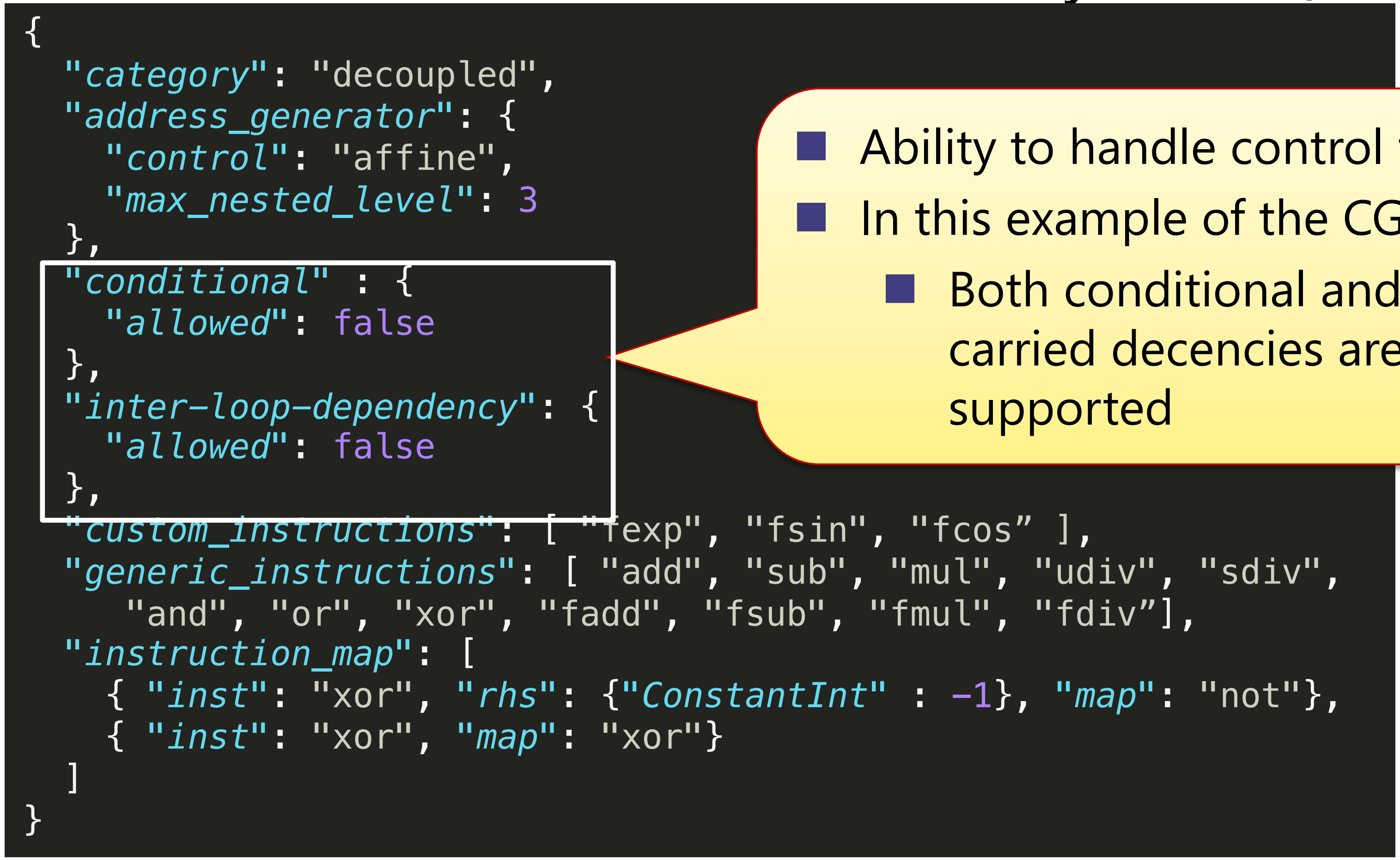
```
{
  "category": "decoupled",
  "address_generator": {
    "control": "affine",
    "max_nested_level": 3
  },
  "conditional" : {
    "allowed": false
  },
  "inter-loop-dependency": {
    "allowed": false
  },
  "custom_instructions": [ "fexp", "fsin", "fcos” ],
  "generic_instructions": [ "add", "sub", "mul", "udiv", "sdiv",
     "and", "or", "xor", "fadd", "fsub", "fmul", "fdiv”],
  "instruction_map": [
    { "inst": "xor", "rhs": {"ConstantInt" : -1}, "map": "not"},
    { "inst": "xor", "map": "xor"}
  ]
}
```

- Ability to memory access control
  - Only affine access is allowd
  - Up-to 3-nested loops
  - i.e., $C_0 + C_1 v_1 + \ C_2 v_2 + C_3 v_3$

An example: the case of RIKEN CGRA

CGRA model description

- The model defines CGRA execution style, etc (JSON)



```
{
  "category": "decoupled",
  "address_generator": {
    "control": "affine",
    "max_nested_level": 3
  },
  "conditional" : {
    "allowed": false
  },
  "inter-loop-dependency": {
    "allowed": false
  },
  "custom_instructions": [ "fexp", "fsin", "fcos” ],
  "generic_instructions": [ "add", "sub", "mul", "udiv", "sdiv",
     "and", "or", "xor", "fadd", "fsub", "fmul", "fdiv”],
  "instruction_map": [
    { "inst": "xor", "rhs": {"ConstantInt" : -1}, "map": "not"},
    { "inst": "xor", "map": "xor"}
  ]
}
```

- Ability to handle control flow
- In this example of the CGRA
  - Both conditional and loop-carried decencies are not supported

An example: the case of RIKEN CGRA

CGRA model description

- The model defines CGRA execution style, etc (JSON)

```
{
  "category
  "address_
    "contro
    "max_ne
  },
  "conditio
    "allowe
  },
  "inter-loo
    "allowed": false
  },
  "custom_instructions": [ "fexp", "fsin", "fcos” ],
  "generic_instructions": [ "add", "sub", "mul", "udiv", "sdiv",
     "and", "or", "xor", "fadd", "fsub", "fmul", "fdiv”],
  "instruction_map": [
    { "inst": "xor", "rhs": {"ConstantInt" : -1}, "map": "not"},
    { "inst": "xor", "map": "xor"}
  ]
}
```

- What kind of instructions are supported in ALU
  - ***custom_instructions***: instructions not in LLVM-IR
    - The Same function name should be used in codes
  - ***generic_instructions***: Corresponding LLVM IR instructions
  - ***instruction_map***: mapping LLVM IR instr. to ALU opcode
    - Some mapping conditions are available

An example: the case of RIKEN CGRA

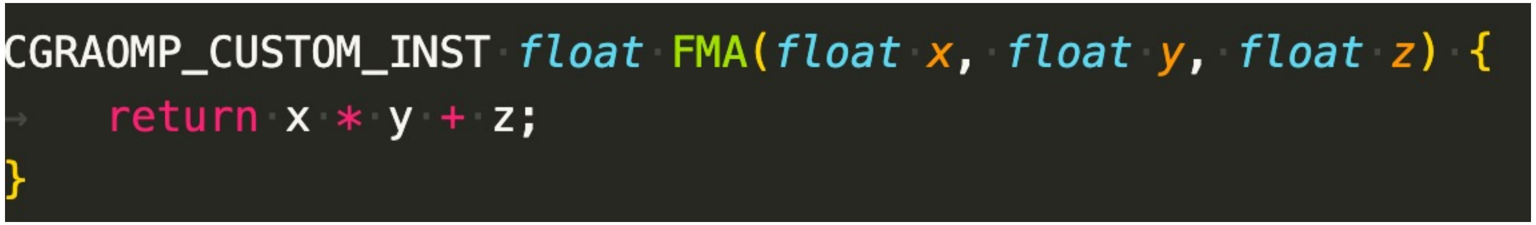
CGRA model description

- The model defines CGRA execution style, etc (JSON)

```
{
  "category
  "address_
    "contro
    "max_ne
  },
  "conditio
    "allowe
  },
  "inter-loo
```

```
"sdiv",
,
"not"},
```

- What kind of instructions are supported in ALU
  - ***custom_instructions***: instructions not in LLVM-IR
    - The Same function name should be used in codes
  - ***generic_instructions***: Corresponding LLVM IR instructions
  - ***instruction_map***: mapping LLVM IR instr. to ALU opcode
    - Some mapping conditions are available

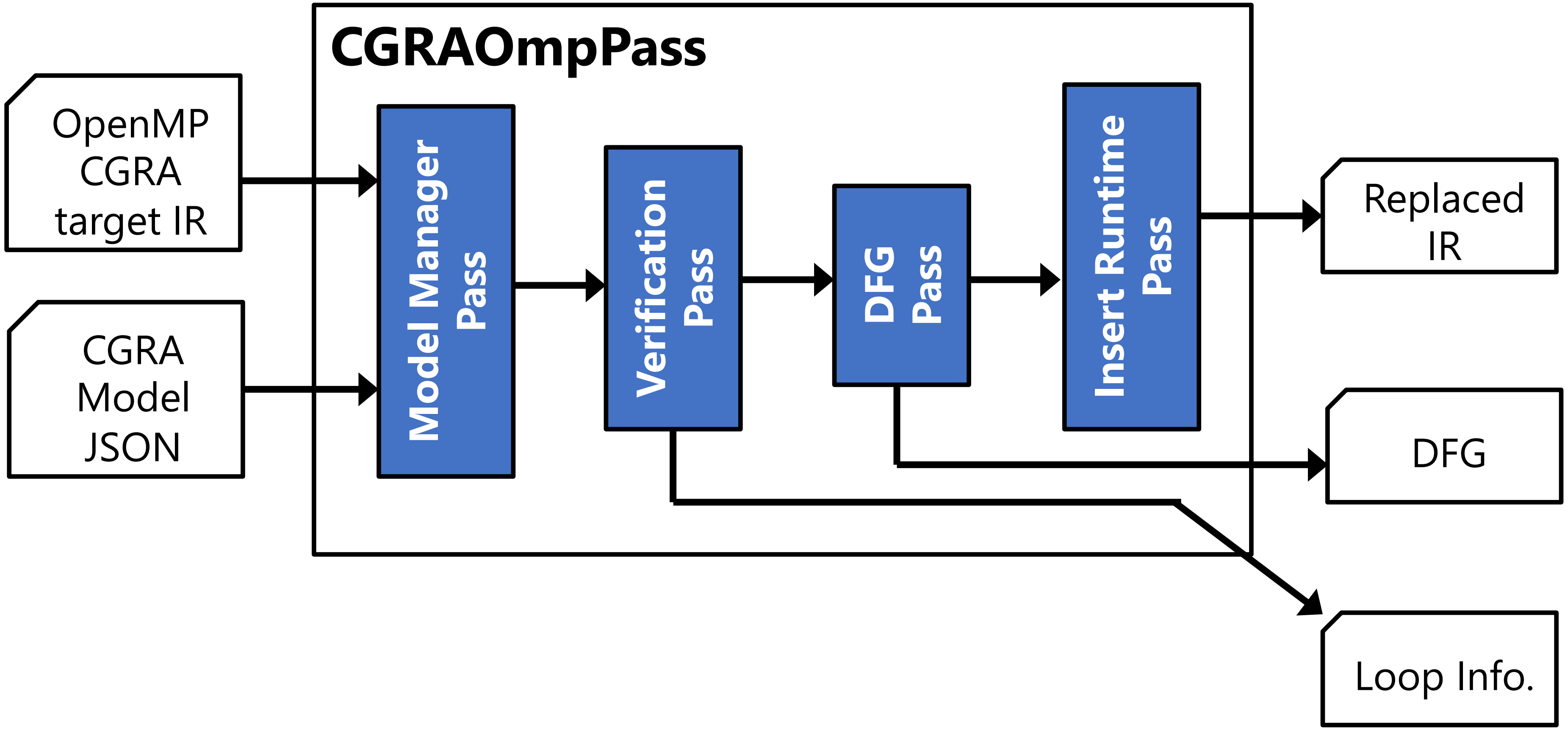


```
CGRAOMP_CUSTOM_INST float FMA(float x, float y, float z) {
    return x * y + z;
}
```

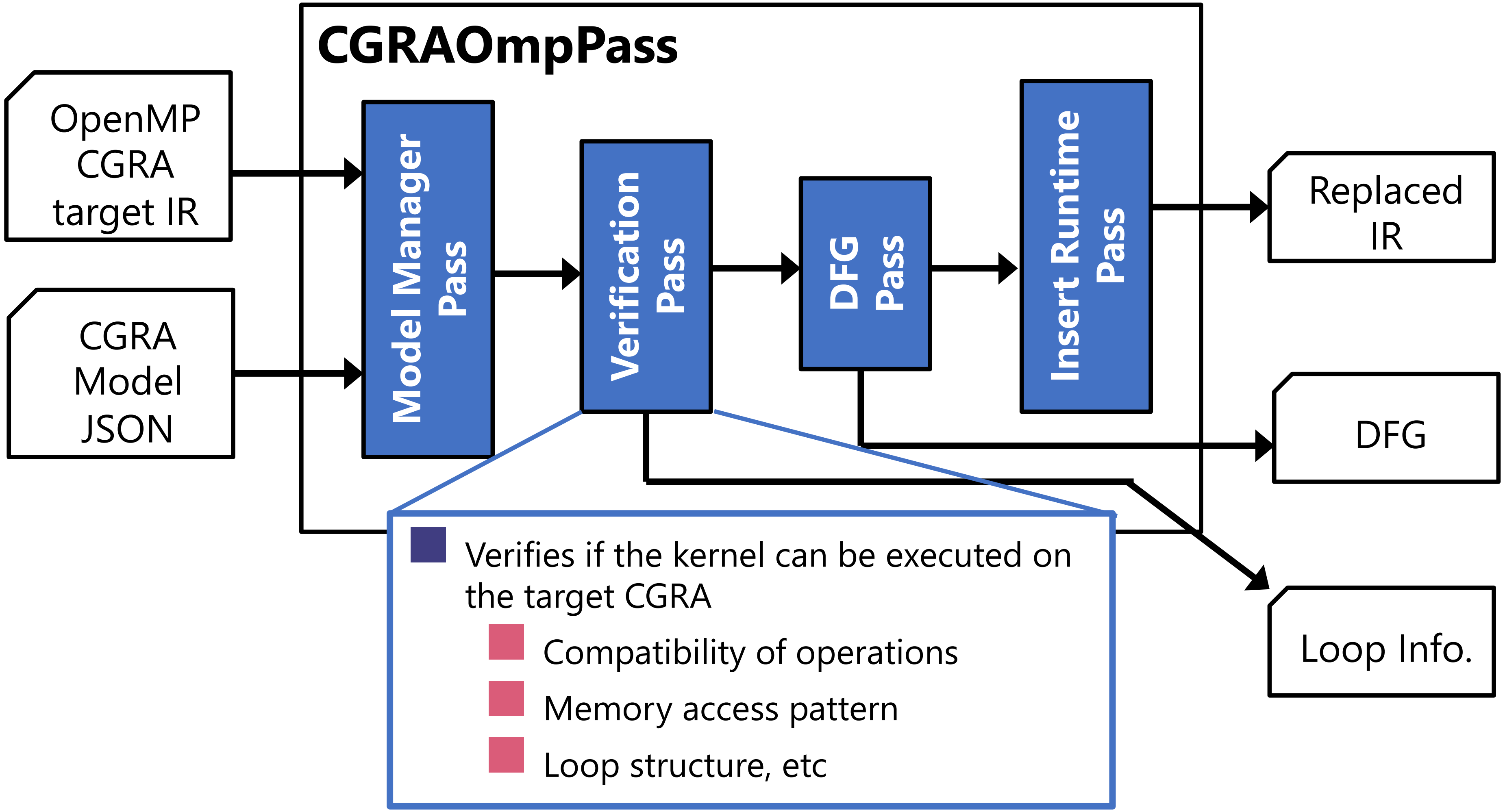
function declaration in source codes for the custom instruction

An example: the case of RIKEN CGRA

Flow of CGRAOmpPass

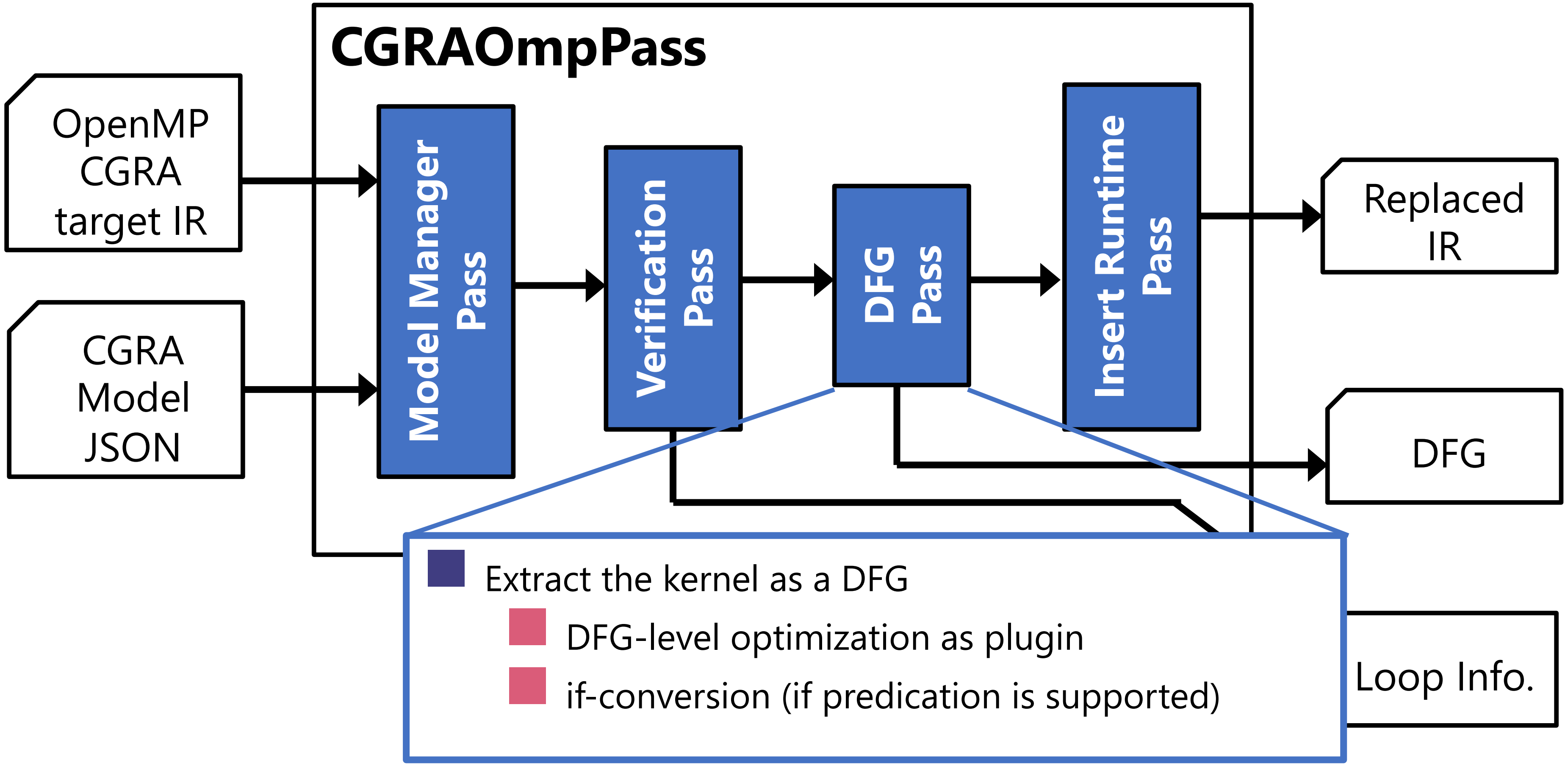


Flow of CGRAOmpPass



- Verifies if the kernel can be executed on the target CGRA
  - Compatibility of operations
  - Memory access pattern
  - Loop structure, etc

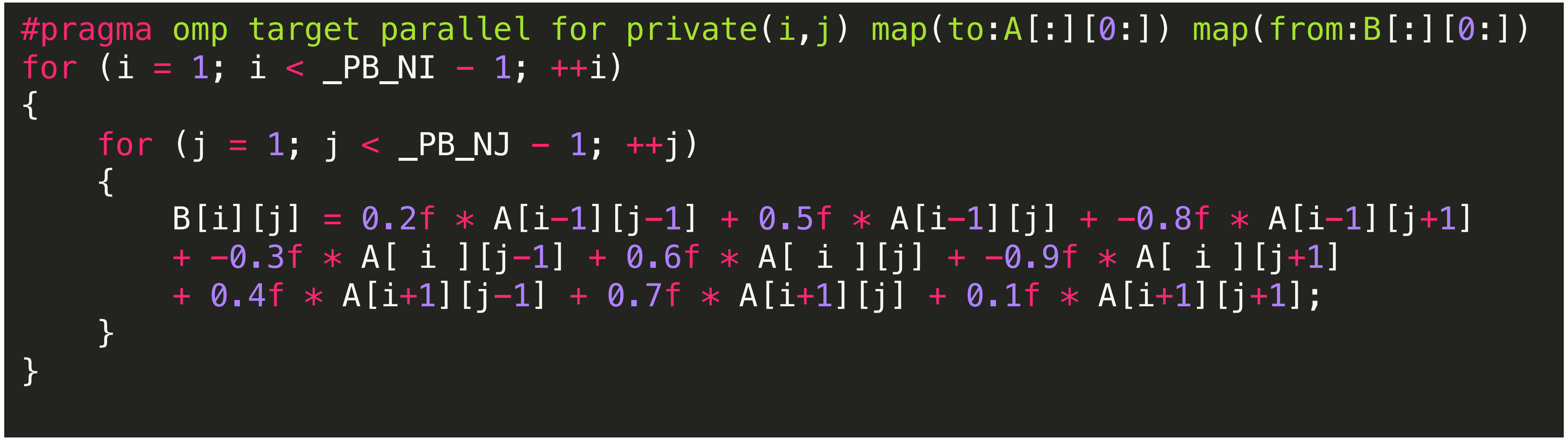
Flow of CGRAOmpPass



- Extract the kernel as a DFG
  - DFG-level optimization as plugin
  - if-conversion (if predication is supported)

Code exmaple: 3x3 convolution

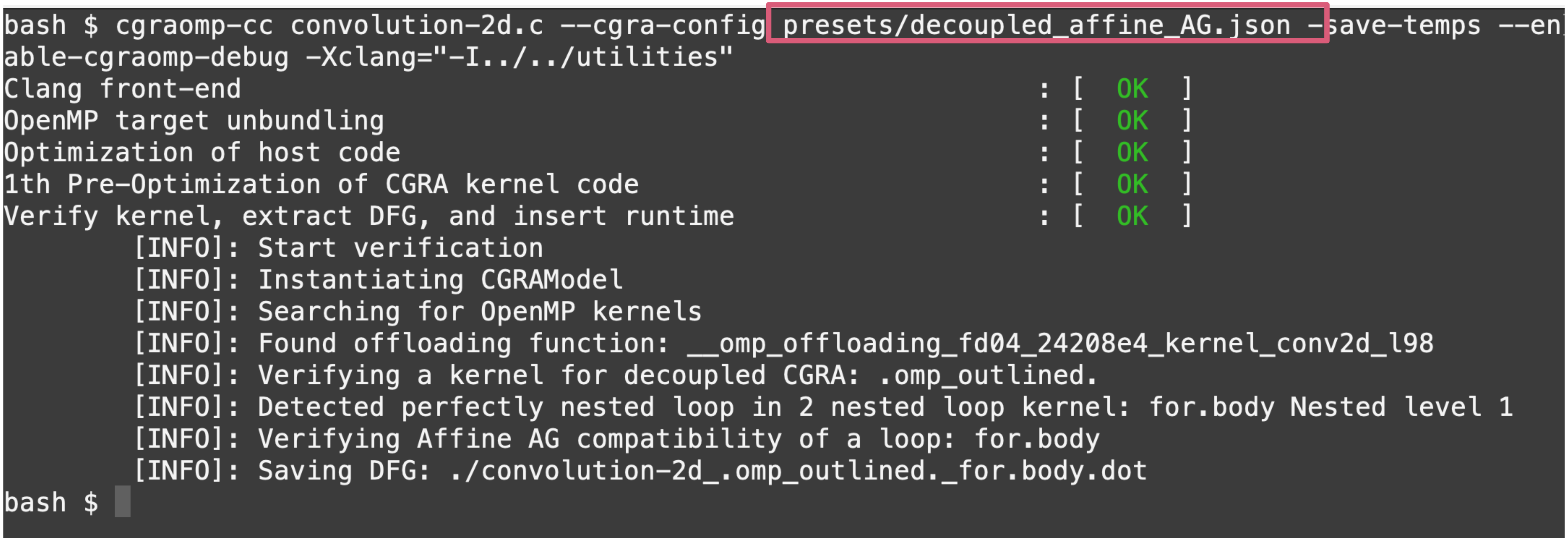
- convolution-2d.c from PolyBench-ACC



```
#pragma omp target parallel for private(i,j) map(to:A[:][0:]) map(from:B[:][0:])
for (i = 1; i < _PB_NI - 1; ++i)
{
    for (j = 1; j < _PB_NJ - 1; ++j)
    {
        B[i][j] = 0.2f * A[i-1][j-1] + 0.5f * A[i-1][j] + -0.8f * A[i-1][j+1]
        + -0.3f * A[ i ][j-1] + 0.6f * A[ i ][j] + -0.9f * A[ i ][j+1]
        + 0.4f * A[i+1][j-1] + 0.7f * A[i+1][j] + 0.1f * A[i+1][j+1];
    }
}
```

← Only this pragma is inserted

Demonstration of the compiler driver

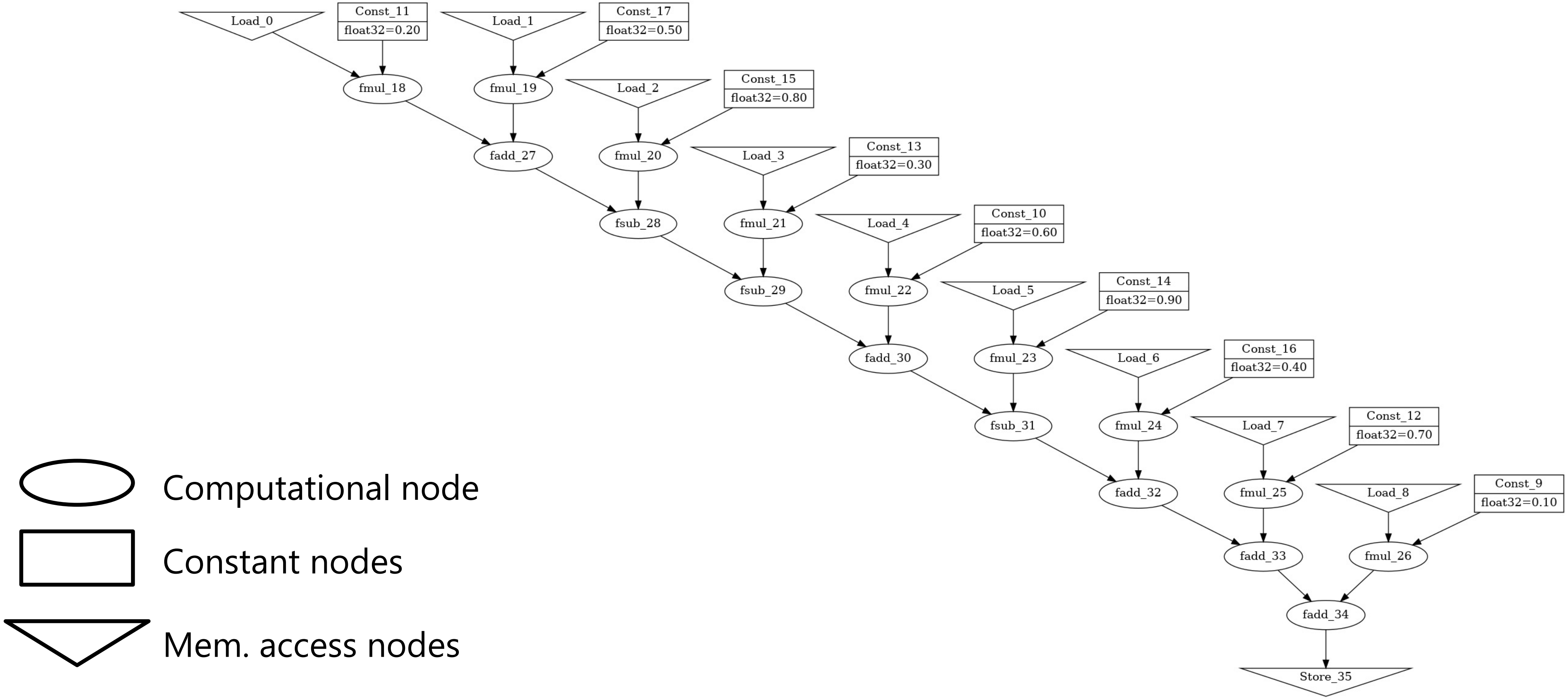


```
bash $ cgraomp-cc convolution-2d.c --cgra-config presets/decoupled_affine_AG.json -save-temps --en
able-cgraomp-debug -Xclang="-I../../utilities"
Clang front-end                                                      : [   OK   ]
OpenMP target unbundling                                             : [   OK   ]
Optimization of host code                                            : [   OK   ]
1th Pre-Optimization of CGRA kernel code                             : [   OK   ]
Verify kernel, extract DFG, and insert runtime                       : [   OK   ]
        [INFO]: Start verification
        [INFO]: Instantiating CGRAModel
        [INFO]: Searching for OpenMP kernels
        [INFO]: Found offloading function: __omp_offloading_fd04_24208e4_kernel_conv2d_l98
        [INFO]: Verifying a kernel for decoupled CGRA: .omp_outlined.
        [INFO]: Detected perfectly nested loop in 2 nested loop kernel: for.body Nested level 1
        [INFO]: Verifying Affine AG compatibility of a loop: for.body
        [INFO]: Saving DFG: ./convolution-2d_.omp_outlined._for.body.dot
bash $ 
```

DFG Optimization after extraction

Generated DFG

- Data dependencies in LLVM-IR cause unbalanced DFG



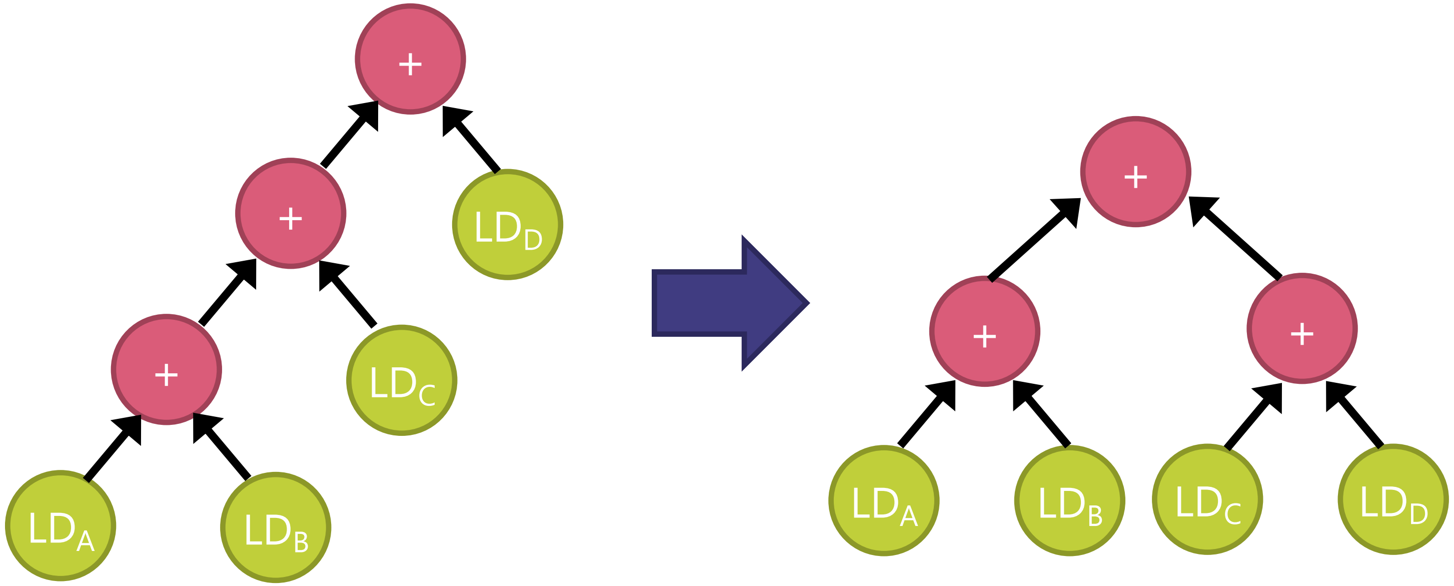
Computational node

Constant nodes

Mem. access nodes

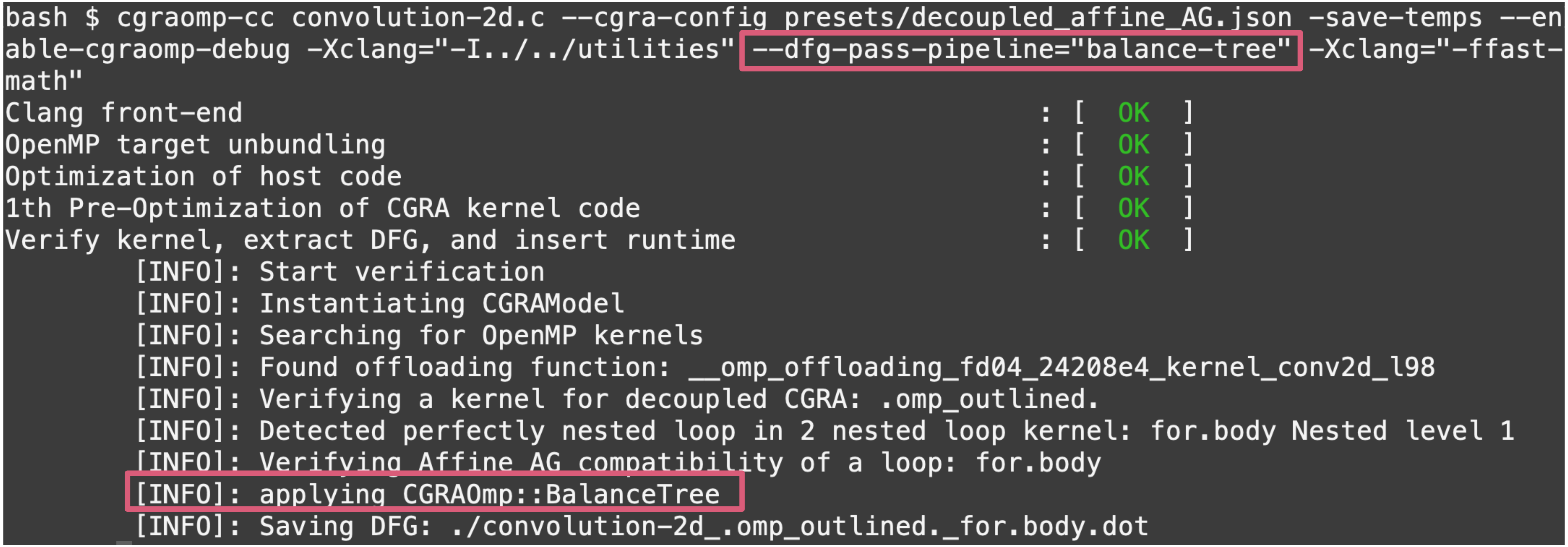
DFG-level optimization: Tree-Height-Reduction

- An important optimization for LSI design and High-level synthesis [18]
  - Graph transformation based on commutativity & associativity of operators
    - e.g., addition (+), multiplication (*)
  - This work integrates Huffman code-based algorithm [19] as a built-in pass



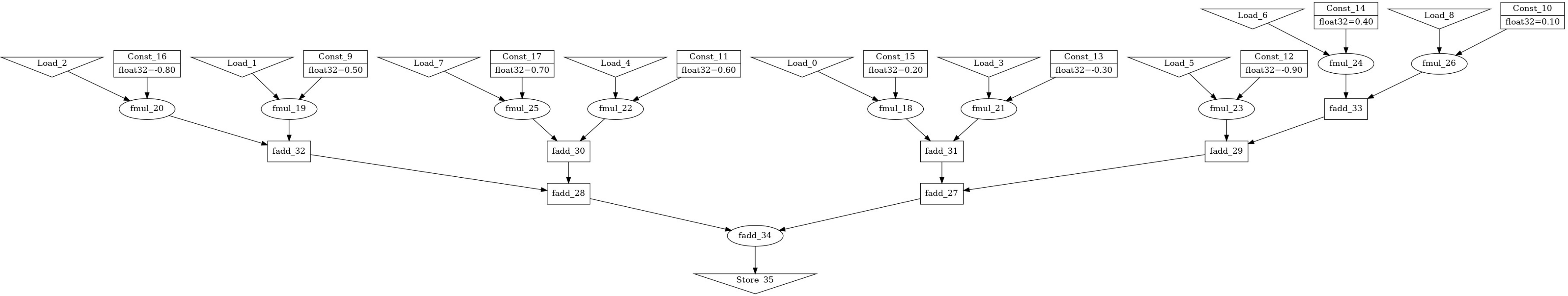
Applying Tree-Height-Reduction

- Easy to custom pass pipeline for DFG optimization



```
bash $ cgraomp-cc convolution-2d.c --cgra-config presets/decoupled_affine_AG.json -save-temps --en
able-cgraomp-debug -Xclang="-I../../utilities" --dfg-pass-pipeline="balance-tree" -Xclang="-ffast-
math"
Clang front-end                                                   : [  OK  ]
OpenMP target unbundling                                          : [  OK  ]
Optimization of host code                                         : [  OK  ]
1th Pre-Optimization of CGRA kernel code                          : [  OK  ]
Verify kernel, extract DFG, and insert runtime                    : [  OK  ]
        [INFO]: Start verification
        [INFO]: Instantiating CGRAModel
        [INFO]: Searching for OpenMP kernels
        [INFO]: Found offloading function: __omp_offloading_fd04_24208e4_kernel_conv2d_l98
        [INFO]: Verifying a kernel for decoupled CGRA: .omp_outlined.
        [INFO]: Detected perfectly nested loop in 2 nested loop kernel: for.body Nested level 1
        [INFO]: Verifying Affine AG compatibility of a loop: for.body
        [INFO]: applying CGRAOmp::BalanceTree
        [INFO]: Saving DFG: ./convolution-2d_.omp_outlined._for.body.dot
```

DFG after optimization



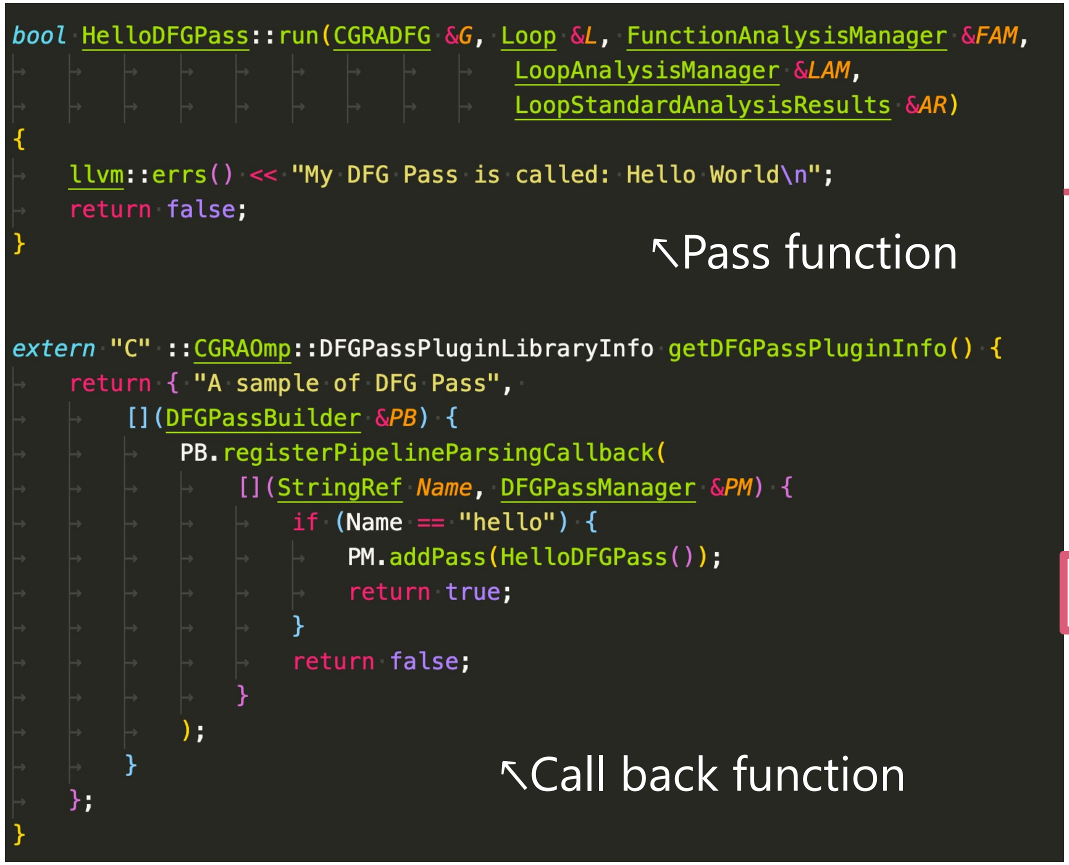
⬭ Computational node

▭ Constant nodes

▽ Mem. access nodes

DFG Pass Plugin

- Easy to create and enable your own custom pass in similar manner in LLVM

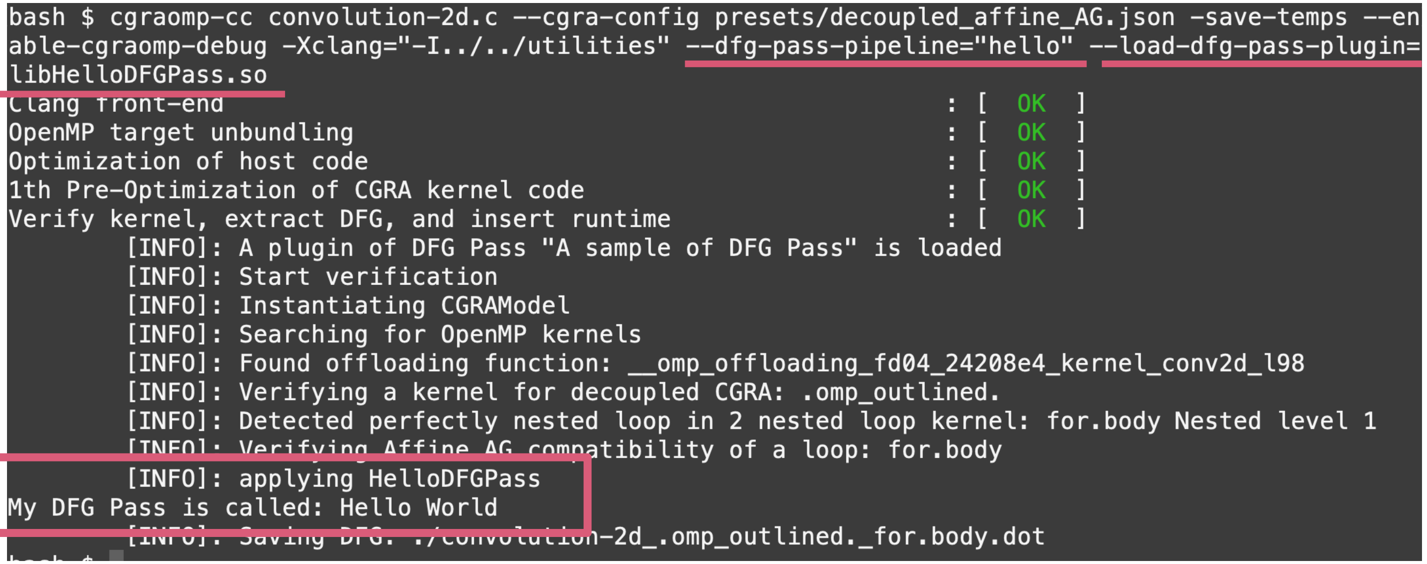


```cpp
bool HelloDFGPass::run(CGRADFG &G, Loop &L, FunctionAnalysisManager &FAM,
                         LoopAnalysisManager &LAM,
                         LoopStandardAnalysisResults &AR)
{
    llvm::errs() << "My DFG Pass is called: Hello World\n";
    return false;
}


extern "C" ::CGRAOmp::DFGPassPluginLibraryInfo getDFGPassPluginInfo() {
    return { "A sample of DFG Pass",
        [](DFGPassBuilder &PB) {
            PB.registerPipelineParsingCallback(
                [](StringRef Name, DFGPassManager &PM) {
                    if (Name == "hello") {
                        PM.addPass(HelloDFGPass());
                        return true;
                    }
                    return false;
                }
            );
        }
    };
}
```

↖Pass function

↖Call back function

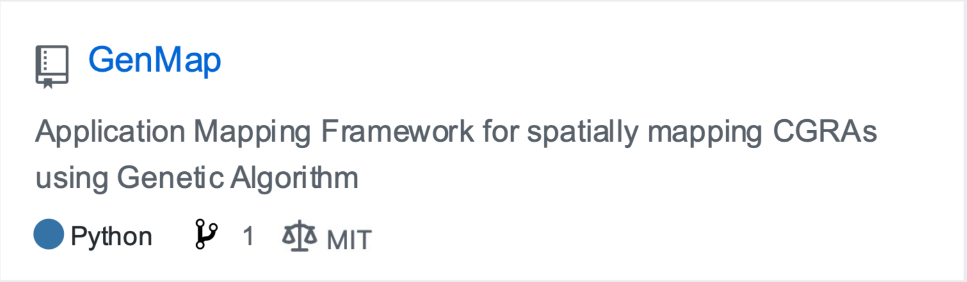


```
bash $ cgraomp-cc convolution-2d.c --cgra-config presets/decoupled_affine_AG.json -save-temps --en
able-cgraomp-debug -Xclang="-I../../utilities" --dfg-pass-pipeline="hello" --load-dfg-pass-plugin=
libHelloDFGPass.so
Clang front-end                                                    : [  OK  ]
OpenMP target unbundling                                           : [  OK  ]
Optimization of host code                                          : [  OK  ]
1th Pre-Optimization of CGRA kernel code                           : [  OK  ]
Verify kernel, extract DFG, and insert runtime                     : [  OK  ]
        [INFO]: A plugin of DFG Pass "A sample of DFG Pass" is loaded
        [INFO]: Start verification
        [INFO]: Instantiating CGRAModel
        [INFO]: Searching for OpenMP kernels
        [INFO]: Found offloading function: __omp_offloading_fd04_24208e4_kernel_conv2d_l98
        [INFO]: Verifying a kernel for decoupled CGRA: .omp_outlined.
        [INFO]: Detected perfectly nested loop in 2 nested loop kernel: for.body Nested level 1
        [INFO]: Verifying Affine AG compatibility of a loop: for.body
        [INFO]: applying HelloDFGPass
My DFG Pass is called: Hello World
        [INFO]: Saving DFG: ./convolution-2d_.omp_outlined._for.body.dot
bash $
```

Evaluation

Experimental setup

- LLVM version 12.0.1
- CGRA design
  - RIKEN CGRA
  - 8x10 array (8x8 PE tiles + 8+8LS tiles)
- Benchmark: 3x3 convolution
- Backend (mapping algorithm)
  - GenMap[20] currently supports RIKEN CGRA
  - Genetic algorithm-based mapping



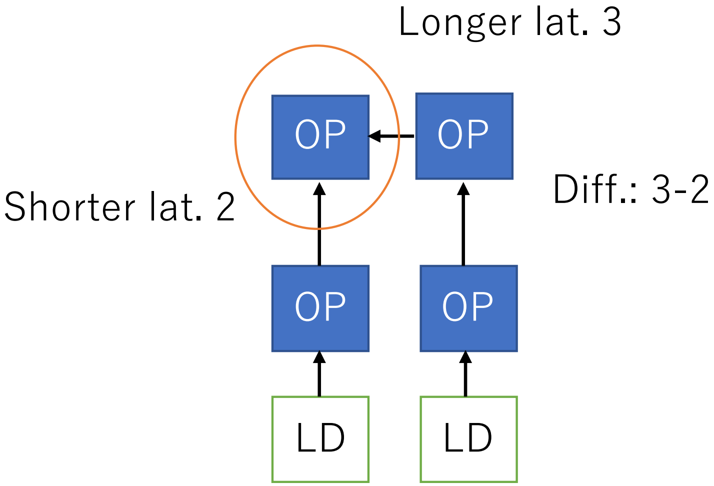
GenMap

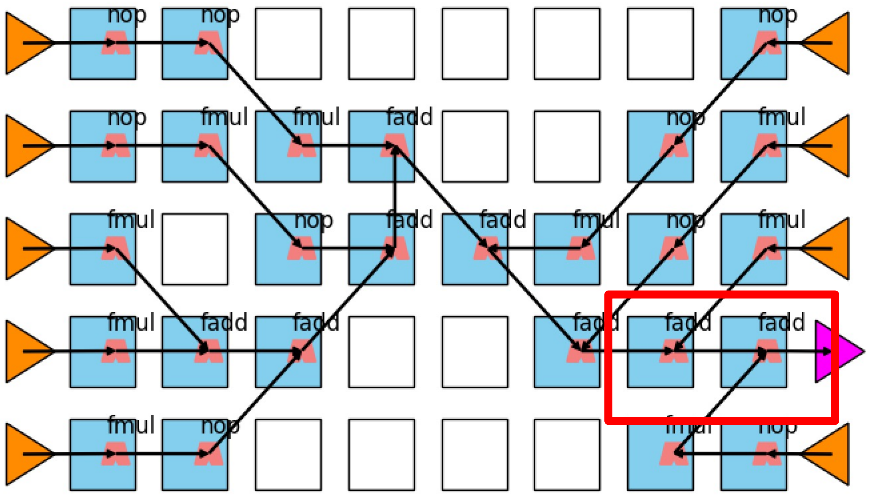
Application Mapping Framework for spatially mapping CGRAs using Genetic Algorithm

https://github.com/hungalab/GenMap

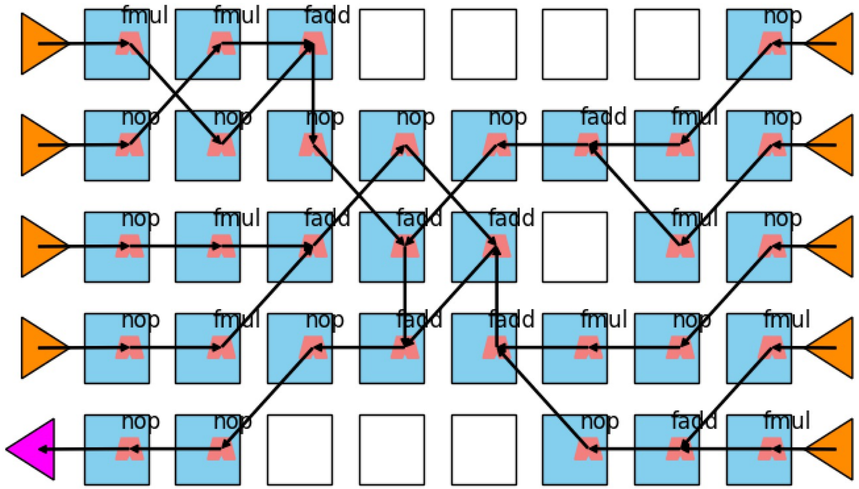
Mapping results

	priority	Total wire length	Map area	Latency diff.
naïve DFG	area	40.38	40 (5 x 8)	6
	latency balance	50.56	56 (7 x 8)	2
Optimized DFG		46.21	40 (5 x 8)	0




nop nop nop
nop fmul fmul fadd nop fmul
fmul nop fadd fadd fmul nop fmul
fmul fadd fadd fadd fadd fadd
fmul nop fmul nop

naïve DFG 5x8 mapping


fmul fmul fadd nop
nop nop nop nop nop fadd fmul nop
nop fmul fadd fadd fadd fmul nop
nop fmul nop fadd fadd fmul nop fmul
nop nop nop fadd fmul

Optimized DFG 5x8 mapping

Conclusion & Future work

- This work
  - Proposes a CGRA compiler designated to handle the same source code regardless of the target architecture
  - Uses OpenMP offloading
- Future work
  - To extend verification and analysis for other types of CGRAs
  - To Implement runtime insertion
  - To make it work together with CGRA simulators or FPGA overlays

References

[1] Hennessy, John L., and David A. Patterson. "A new golden age for computer architecture." *Communications of the ACM* 62.2 (2019): 48-60.

[2] Liu, Leibo, et al. "A survey of coarse-grained reconfigurable architecture and design: Taxonomy, challenges, and applications." *ACM Computing Surveys (CSUR)* 52.6 (2019): 1-39.

[3] Anderson, Jason, et al. "CGRA-ME: An Open-Source Framework for CGRA Architecture and CAD Research." *2021 IEEE 32nd International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. IEEE, 2021.

[4] Tan, Cheng, et al. "OpenCGRA: An Open-Source Unified Framework for Modeling, Testing, and Evaluating CGRAs." *2020 IEEE 38th International Conference on Computer Design (ICCD)*. IEEE, 2020.

[5] Podobas, Artur, Kentaro Sano, and Satoshi Matsuoka. "A template-based framework for exploring coarse-grained reconfigurable architectures." *2020 IEEE 31st International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. IEEE, 2020.

[6] Weng, Jian, et al. "Dsagen: Synthesizing programmable spatial accelerators." *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2020.

References

[7] Gobieski, Graham, et al. "Snafu: an ultra-low-power, energy-minimal CGRA-generation framework and architecture." *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2021.

[8] Podobas, Artur, Kentaro Sano, and Satoshi Matsuoka. "A survey on coarse-grained reconfigurable architectures from a performance perspective." *IEEE Access* 8 (2020): 146719-146743.

[9] Renesas Electronics Corporation, “Dynamically Reconfigurable Processor (DRP) Technology Development | Renesas”, https://www.renesas.com/us/en/application/key-technology/artificial-intelligence/voice-face-recognition/drp-development, access 2022.

[10] Chou, Yuan, et al. "Piperench implementation of the instruction path coprocessor." *Proceedings of the 33rd annual ACM/IEEE international symposium on Microarchitecture*. 2000.

[11] Cardoso, Joao MP, and Markus Weinhardt. "XPP-VC: AC compiler with temporal partitioning for the PACT-XPP architecture." *International Conference on Field Programmable Logic and Applications*. Springer, Berlin, Heidelberg, 2002.

References

[12] SambaNova, Accelerated Computing with a Reconfigurable Dataflow Architecture. https://sambanova.ai/wp-content/uploads/2021/04/SambaNova_RDA_Whitepaper.pdf, access 2022

[13] Tunbunheng, Vasutan, and Hideharu Amano. "Black-diamond: A retargetable compiler using graph with configuration bits for dynamically reconfigurable architectures." *Proc. 14th Workshop on Synthesis and System Integration of Mixed Information Technologies (SASIMI)*. 2007.

[14] S. Dave and A. Shrivastava, “CCF: A CGRA compilation framework,” https://github.com/MPSLab-ASU/ccf, access 2022

[15] Kim, Hee-Seok, et al. "Design evaluation of opencl compiler framework for coarse-grained reconfigurable arrays." *2012 International Conference on Field-Programmable Technology*. IEEE, 2012.

[16] Ohwada, Ayaka, Takuya Kojima, and Hideharu Amano. "MENTAI: A Fully Automated CGRA Application Development Environment that Supports Hardware/Software Co-design."

[17] OpenMP, ” OpenMP Application Programming Interface Examples”, 2016.

References

[18] D.L. Kuck, Structure of Computers and Computations, John Wiley & Sons, Inc., 1978.

[19] K.E. Coons, W. Hunt, B.A. Maher, D. Burger, and K.S. McKin- ley, Optimal huffman tree-height reduction for instruction-level parallelism, Computer Science Department, University of Texas at Austin, 2008.

[20] Kojima, Takuya, Nguyen Anh Vu Doan, and Hideharu Amano. "GenMap: A Genetic Algorithmic Approach for Optimizing Spatial Mapping of Coarse-Grained Reconfigurable Architectures." *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 28.11 (2020): 2383-2396.