HPC向けRIKEN CGRAのための コンパイル環境整備と予備評価

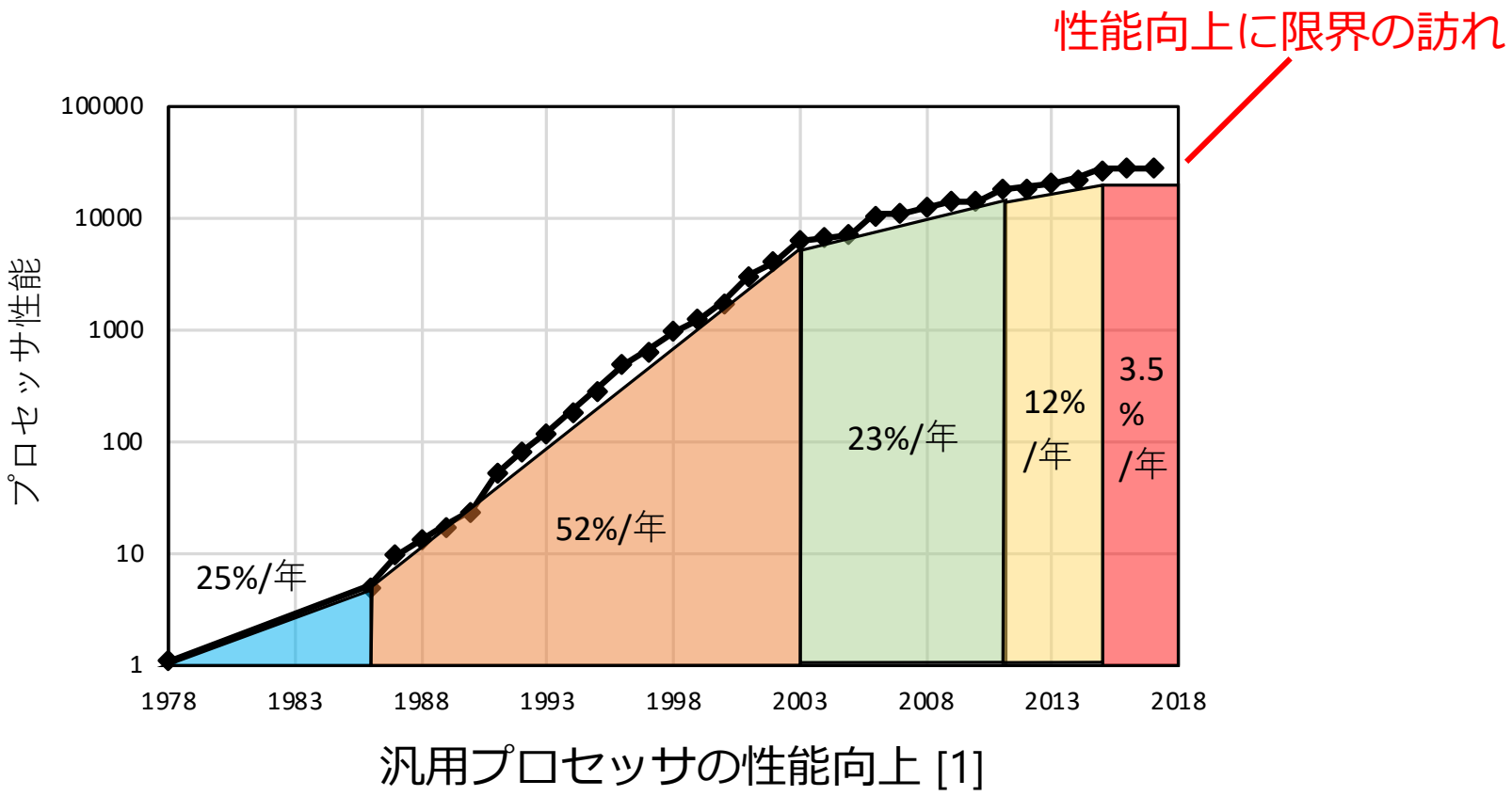
小島拓也[†‡], Carlos Cesar Cortes Torres[‡], Boma Adhi[‡], Yiyu Tan[‡], 佐野健太郎[‡]

† 東京大学大学院 情報理工学系研究科

‡ 理化学研究所

本研究の背景

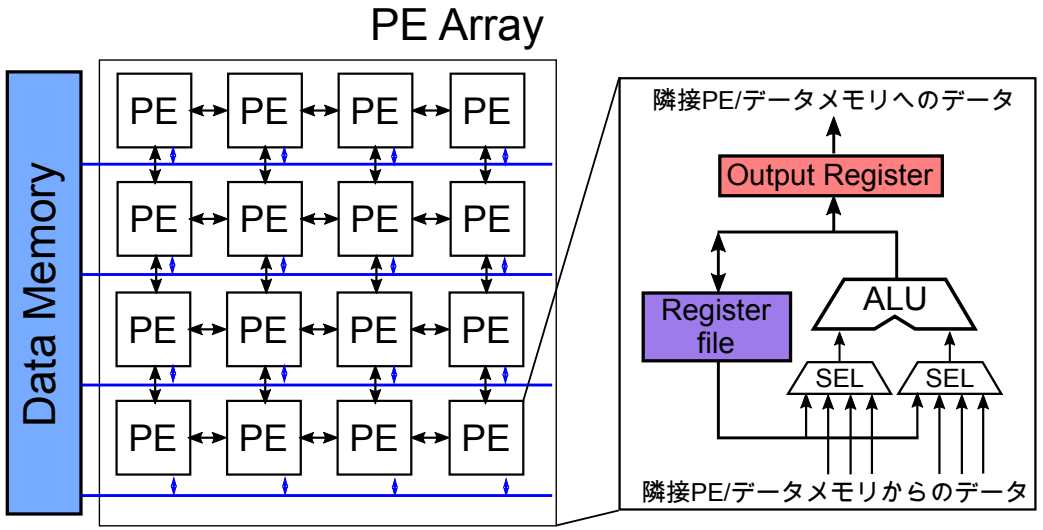
汎用プロセッサの限界



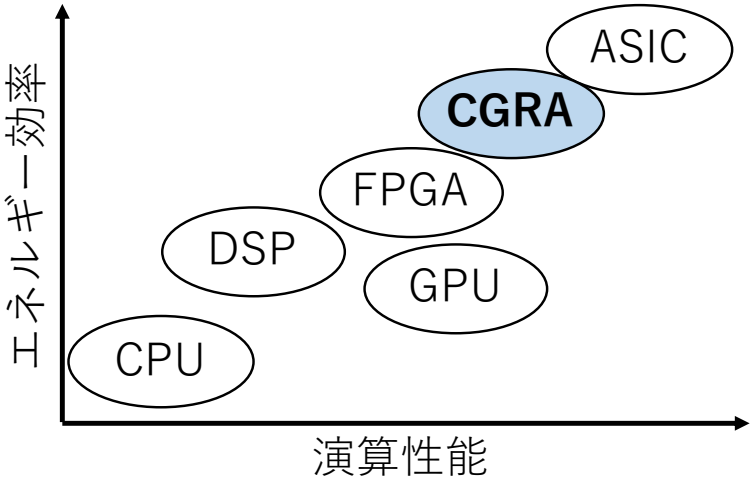
汎用プロセッサの性能向上 [1]

汎用プロセッサに代わり**性能スケーリング**を
維持する新たなアーキテクチャが必要

データフロー型計算機 CGRA



一般的なCGRAの構成



他のアーキテクチャとの比較 [2]

- Coarse Grained Reconfigurable Architecture
- 再構成可能アーキテクチャの一種
  - 再構成可能なPEがアレイ状に配置
  - 粗粒度な再構成のための電力/性能のオーバーヘッドが小さい
  - 一般にCPUの代わりにループ部を実行する (アクセラレータ)

本研究の目的と提案

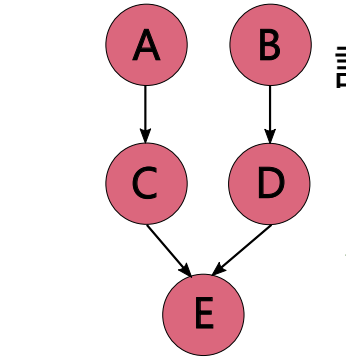
- CGRA研究のトレンド: 設計探索フレームワーク
  - 高いカスタマイズ性
  - 領域特化アーキテクチャの実現
  - 近年の報告: CGRA-ME [3], OpenCGRA [4], RIKEN CGRA [5]
    DSAGEN [6] (ISCA 2020), SNAFU [7] (ISCA 2021)
- 課題
  - CGRA向け汎用コンパイラの不在
- 本研究の提案
  - CGRAへのOpenMP offloadingを可能にするコンパイラ
  - 汎用化に向けて: RIKEN CGRAへの対応
  - 予備評価の実施

多様なアーキテクチャ

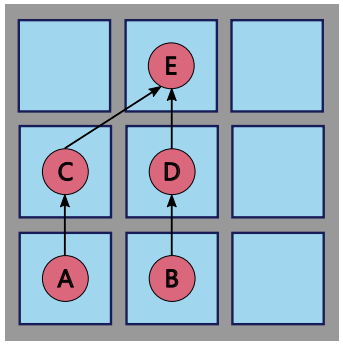
- アーキテクチャを特徴づける多数のパラメータ
  - 再構成方式
  - PEアレイサイズ
  - ネットワークトポロジ
  - 演算能力
  - オンチップの制御フロー
  - メモリアーキテクチャ

多様なアーキテクチャ

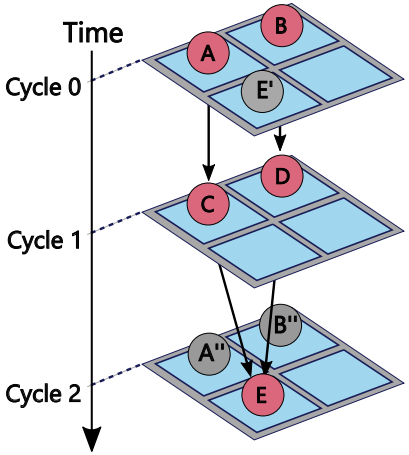
- アーキテクチャを特徴づける多数のパラメータ
  - 再構成方式
    - Time-Multiplexed
    - Spatial
  - PEアレイサイズ
  - ネットワークトポロジ
  - 演算能力
  - オンチップの制御フロー
  - メモリアーキテクチャ



計算データフローグラフ
(DFG)



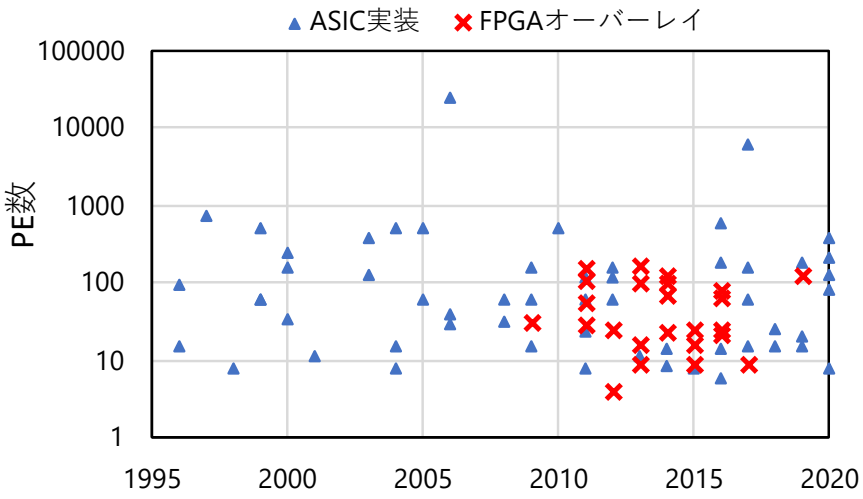
Spatial方式



Time-Multiplexed方式

多様なアーキテクチャ

- アーキテクチャを特徴づける多数のパラメータ
  - 再構成方式
  - ➢PEアレイサイズ
  - ネットワークトポロジ
  - 演算能力
  - オンチップの制御フロー
  - メモリアーキテクチャ

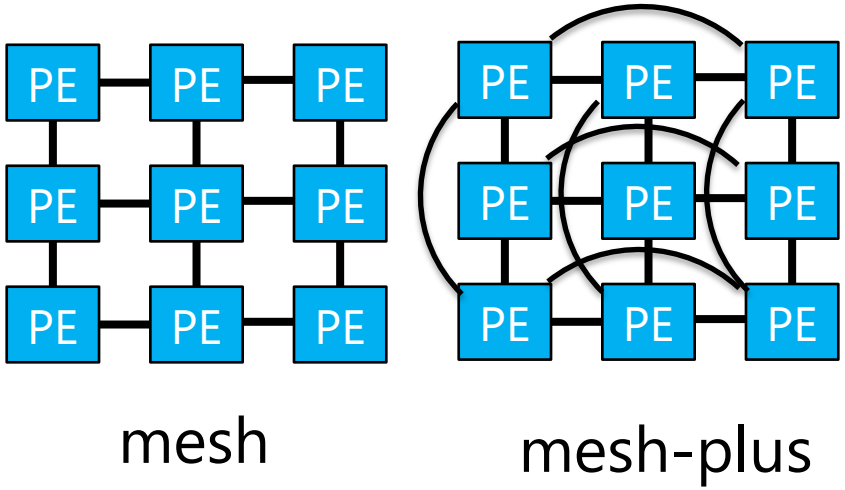


PE数の分布 [8]より

数個ー数万PEの実装が存在

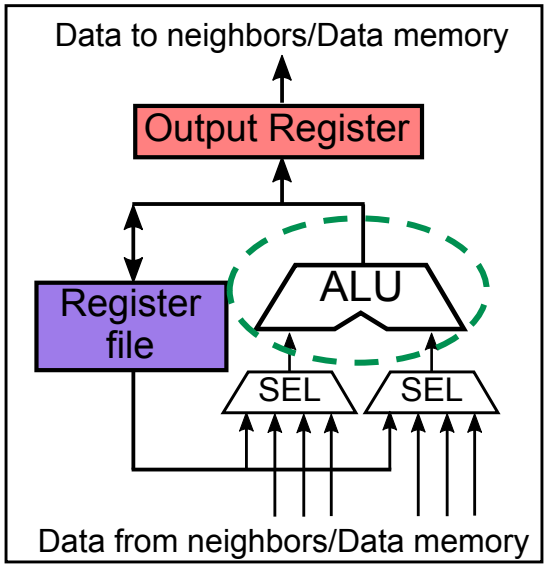
多様なアーキテクチャ

- アーキテクチャを特徴づける多数のパラメータ
  - 再構成方式
  - PEアレイサイズ
  - ネットワークトポロジ
    - Mesh
    - Meshplus
    - Torus, etc
  - 演算能力
  - オンチップの制御フロー
  - メモリアーキテクチャ



多様なアーキテクチャ

- アーキテクチャを特徴づける多数のパラメータ
  - 再構成方式
  - PEアレイサイズ
  - ネットワークトポロジ
  - 演算能力
    - ビット幅
    - 浮動小数点演算
    - SIMD
    - カスタム命令
      - ReLU, sigmoid, etc for ML
  - オンチップの制御フロー
  - メモリアーキテクチャ



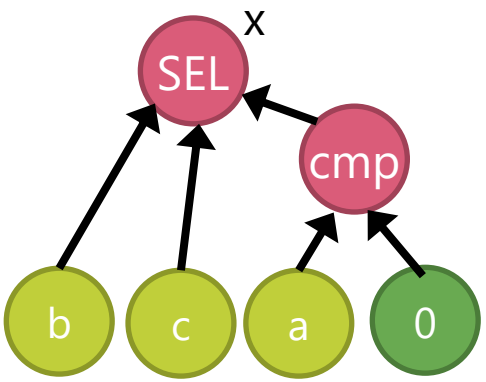
32bit? 64bit?
int? float? fixed?

多様なアーキテクチャ

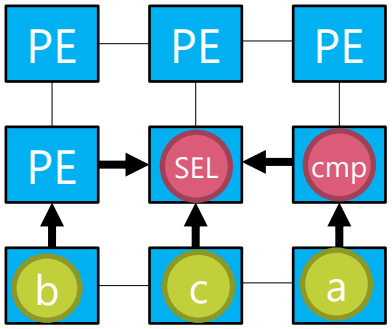
- アーキテクチャを特徴づける多数のパラメータ
  - 再構成方式
  - PEアレイサイズ
  - ネットワークトポロジ
  - 演算能力
  - ➢ オンチップの制御フロー
    - 条件分岐の可否
    - ループ間依存の可否
  - メモリアーキテクチャ

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

条件分岐を含むループ

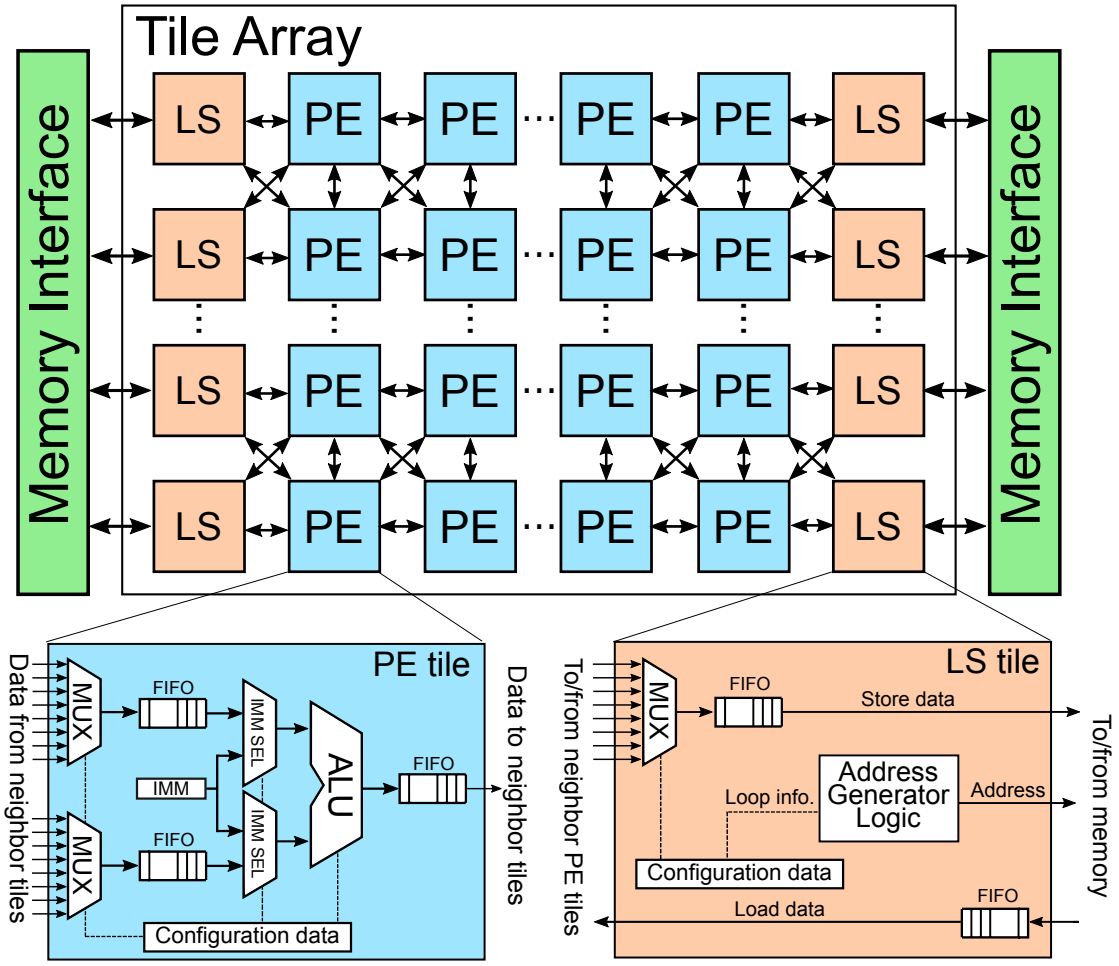


Partial Predicationによる DFG



Predicate実行可能なCGRA

RIKEN CGRA [5]

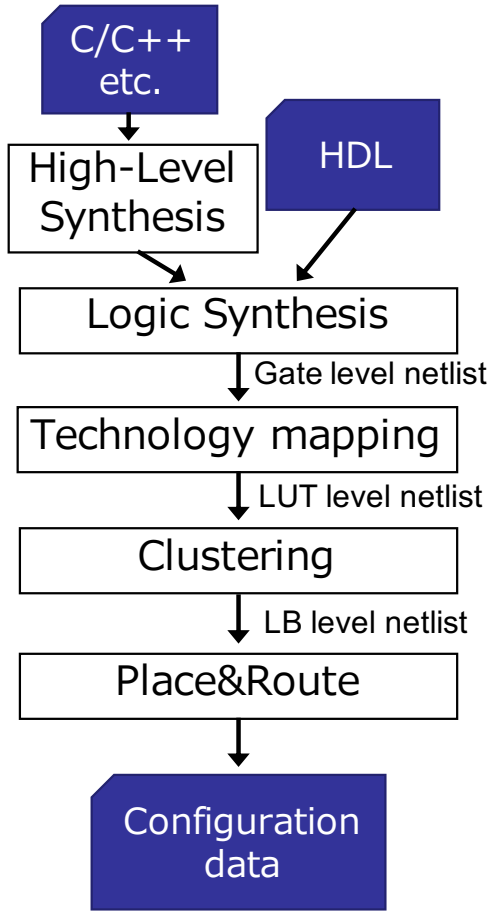


RIKEN CGRAの概要

- 設計探索を目的として設計テンプレート
  - SystemVerilogで実装
- 2種のタイルで構成
  1. LS (Load/Store)
     - ループ情報を基にデータアクセス
  2. PE
     - 各種演算
- Spatial方式の再構成
  - FIFOで遅延差を吸収

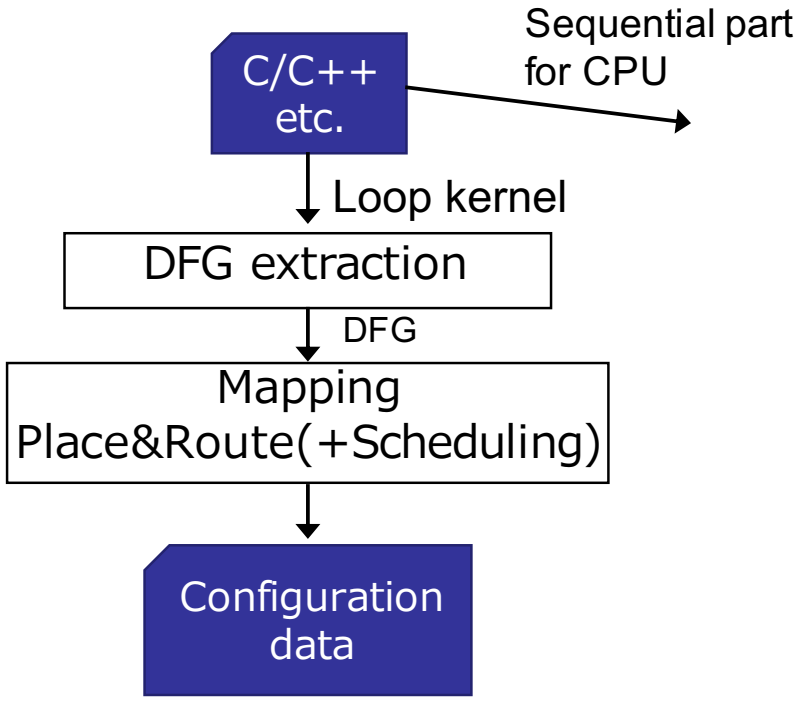
既存コンパイラ技術とその制約

CGRA vs FPGA コンパイルフロー



FPGAの場合

粗な再構成粒度がCADフローを軽減



CGRAの場合

既存のコンパイラ比較

手法、ツール	フロントエンド	ターゲット	
		アーキテクチャ	再構成方式
Musketeer (STP Tool) [9]	C言語	Renesas STP (DRP)	TM
PipeRench Compiler [10]	C-like DSL (DIL)	PipeRench	SP (TM)
PACT XPP-VC [11]	C言語	PACT XPP	TM
SambaFlow™ [12]	PyTorch, TensorFlow, etc	SambaNova RDA	SP
BlackDiamond [13]	C-like DSL	Parameterized	TM/SP
CCF [14]	C言語 (pragma)	ADRES[16]-like	TM
Kim, Hee-Seok, *et al* [15]	OpenCL	SRP	TM
MENTAI [16]	C言語	Cool Mega Array	SP
CGRA-ME [3]	C言語	Parameterized	TM/SP
OpenCGRA [4]	Python DSL	Parameterized	TM
DSAGEN [6]	C言語 (pragma)	Parameterized	SP

商用製品

TM: Time-Multiplexed, SP: Spatial

提案: CGRA向けOpenMPコンパイラ

手法、ツール	フロントエンド	ターゲット	
		アーキテクチャ	再構成方式
Musketeer (STP Tool) [9]	C言語	Renesas STP (DRP)	TM
PipeRench Compiler [10]	C-like DSL (DIL)	PipeRench	SP (TM)
PACT XPP-VC [11]	C言語	PACT XPP	TM
SambaFlow™ [12]	PyTorch, TensorFlow, etc	SambaNova RDA	SP
BlackDiamond [13]	C-like DSL	Parameterized	TM/SP
CCF [14]	C言語 (pragma)	ADRES[16]-like	TM
Kim, Hee-Seok, *et al* [15]	OpenCL	SRP	TM
MENTAI [16]	C言語	Cool Mega Array	SP
CGRA-ME [3]	C言語	Parameterized	TM/SP
OpenCGRA [4]	Python DSL	Parameterized	TM
DSAGEN [6]	C言語 (pragma)	Parameterized	SP
本研究	**OpenMP**	**Parameterized**	**TM/SP**

商用製品

TM: Time-Multiplexed, SP: Spatial

提案: CGRA向けOpenMPコンパイラ

手法、ツール	フロントエンド	ターゲット	
		アーキテクチャ	再構成方式
Musketeer (STP Tool) [9]	C言語	Renesas STP (DRP)	TM
PipeRench Compiler [10]	C-like DSL (DIL)	PipeRench	SP (TM)
PACT XPP-VC [11]	C言語	PACT XPP	TM
SambaFlow™ [12]	PyTorch, TensorFlow, etc	SambaNova RDA	SP
BlackDia			SP
CC			
Kim, Hee-S			
MEN			
CGR			P
Open			
DSAGEN [6]	C言語 (pragma)	Parameterized	SP
本研究	**OpenMP**	**Parameterized**	**TM/SP**

商用製品

設計探索
さまざまな方式をより公平に評価
ソースコード再利用性
最小限のコード修正
異種アーキテクチャとの比較
メニーコアCPUやGPUと同一カーネルで比較

TM: Time-Multiplexed, SP: Spatial

提案手法: CGRA OpenMP

Targetディレクティブを採用

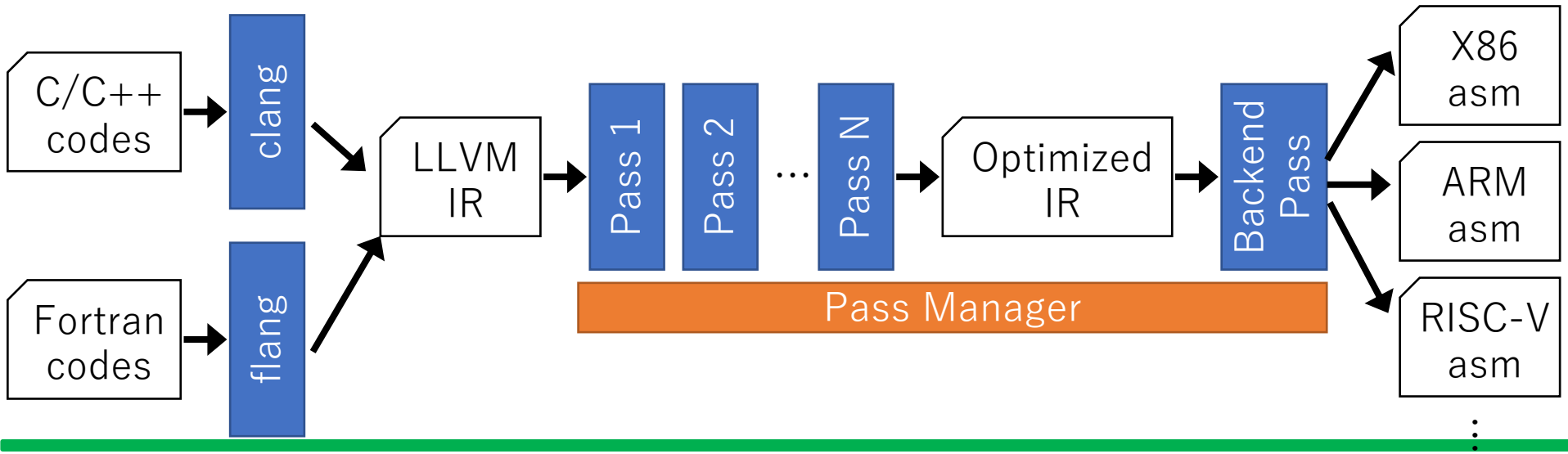
- OpenMP 4.0から追加された
  アクセラレータオフロード機能
  - OpenACCと類似するコンセプト
  - 主にGPUを対象とした実装が利用可能
  - ホスト-デバイス間で明示的なデータ転送 (map節)

```
#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];
}
```

Targetディレクティブを用いたサンプルコード [17]

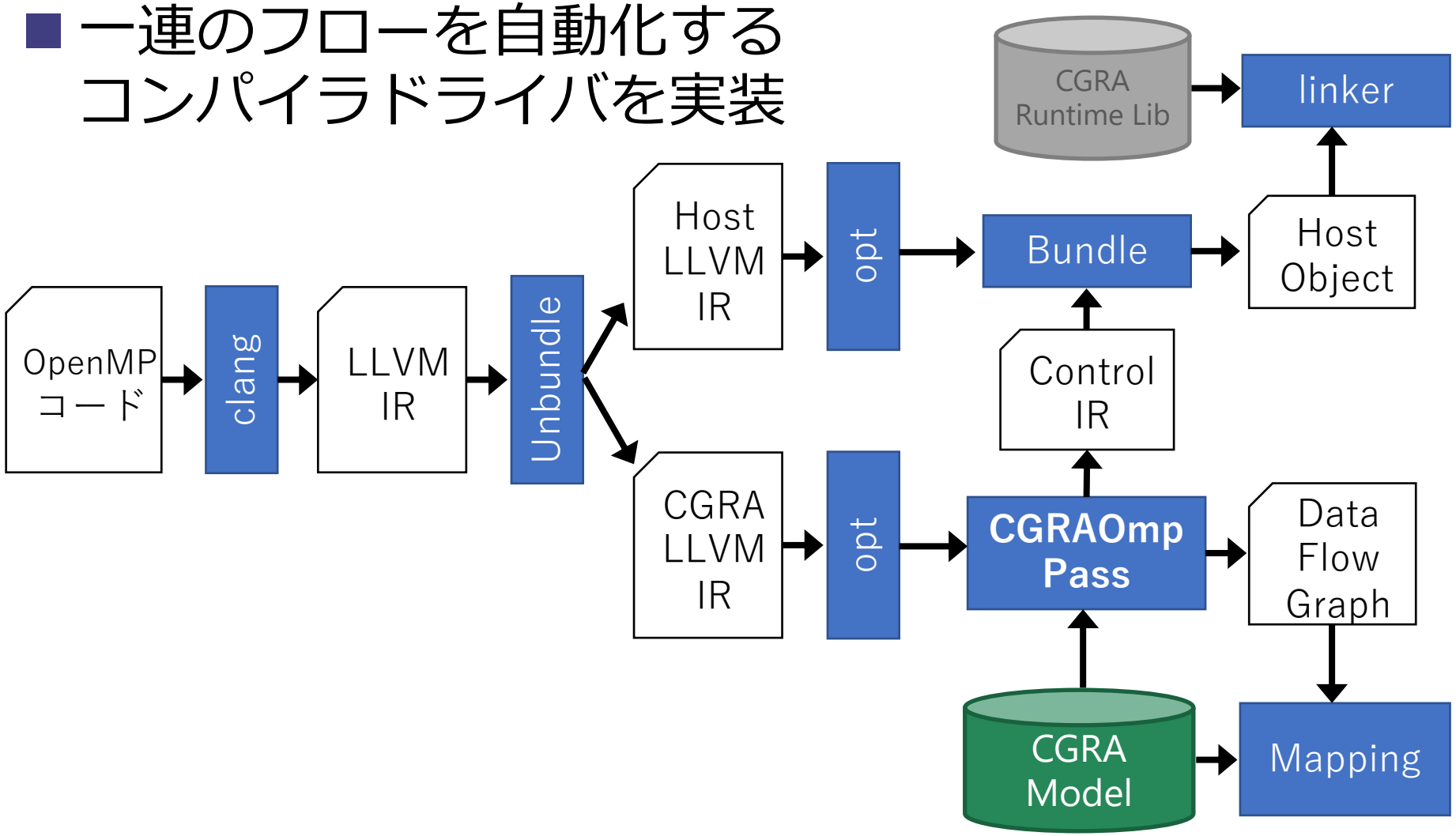
LLVMベースの開発

- LLVM: オープンソースコンパイラ基盤
  - マシン非依存な中間表現 LLVM-IR
  - コンパイラに共通の最適化、解析機能 (Pass)
  - さまざまな公式サブプロジェクト
    - Cフロントエンド Clang, FortranフロントエンドFlang, OpenMP, etc
  - 外部プロジェクトも多数



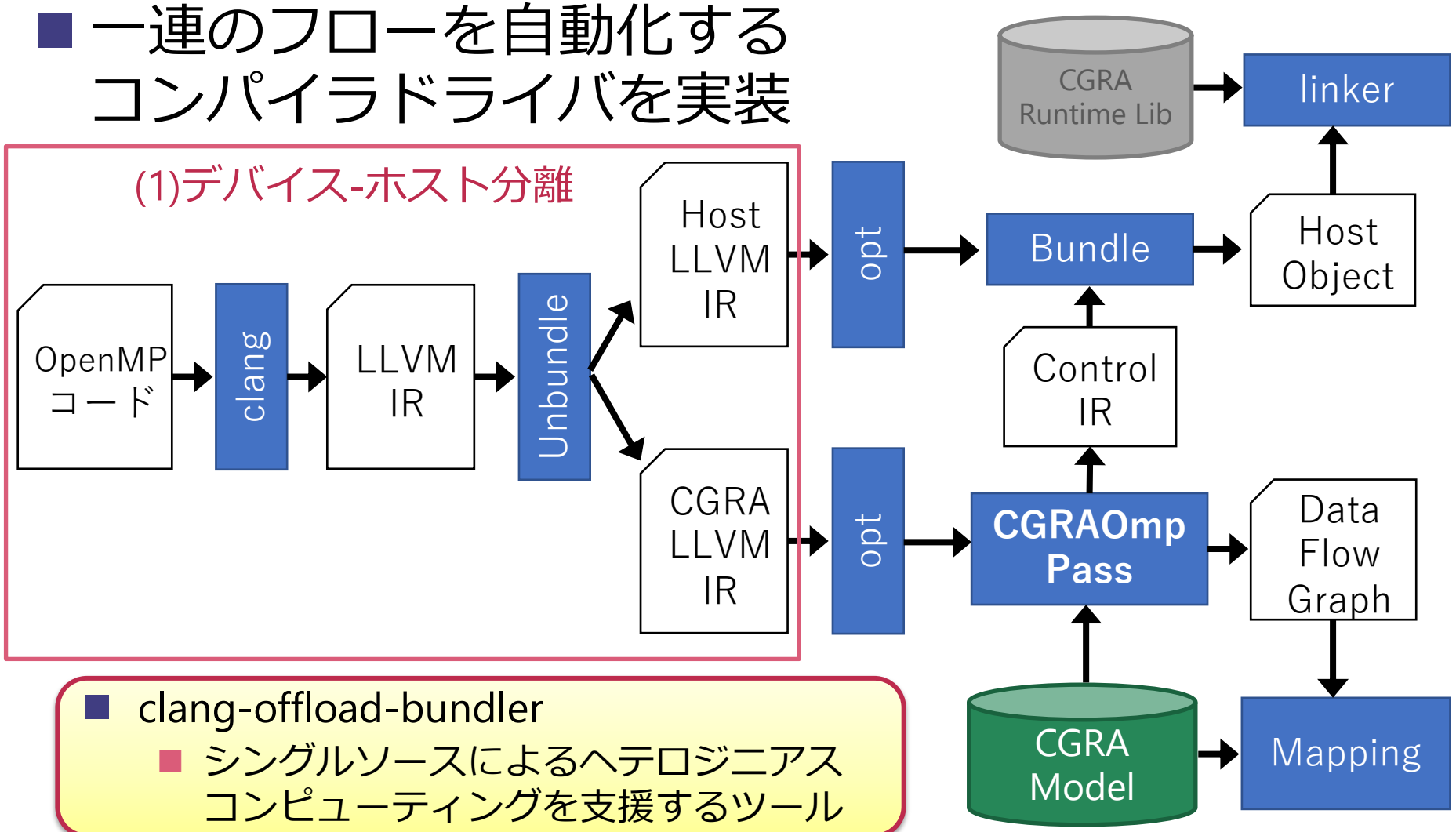
CGRA OpenMPのコンパイルフロー

- 一連のフローを自動化する
  コンパイラドライバを実装



CGRA OpenMPのコンパイルフロー

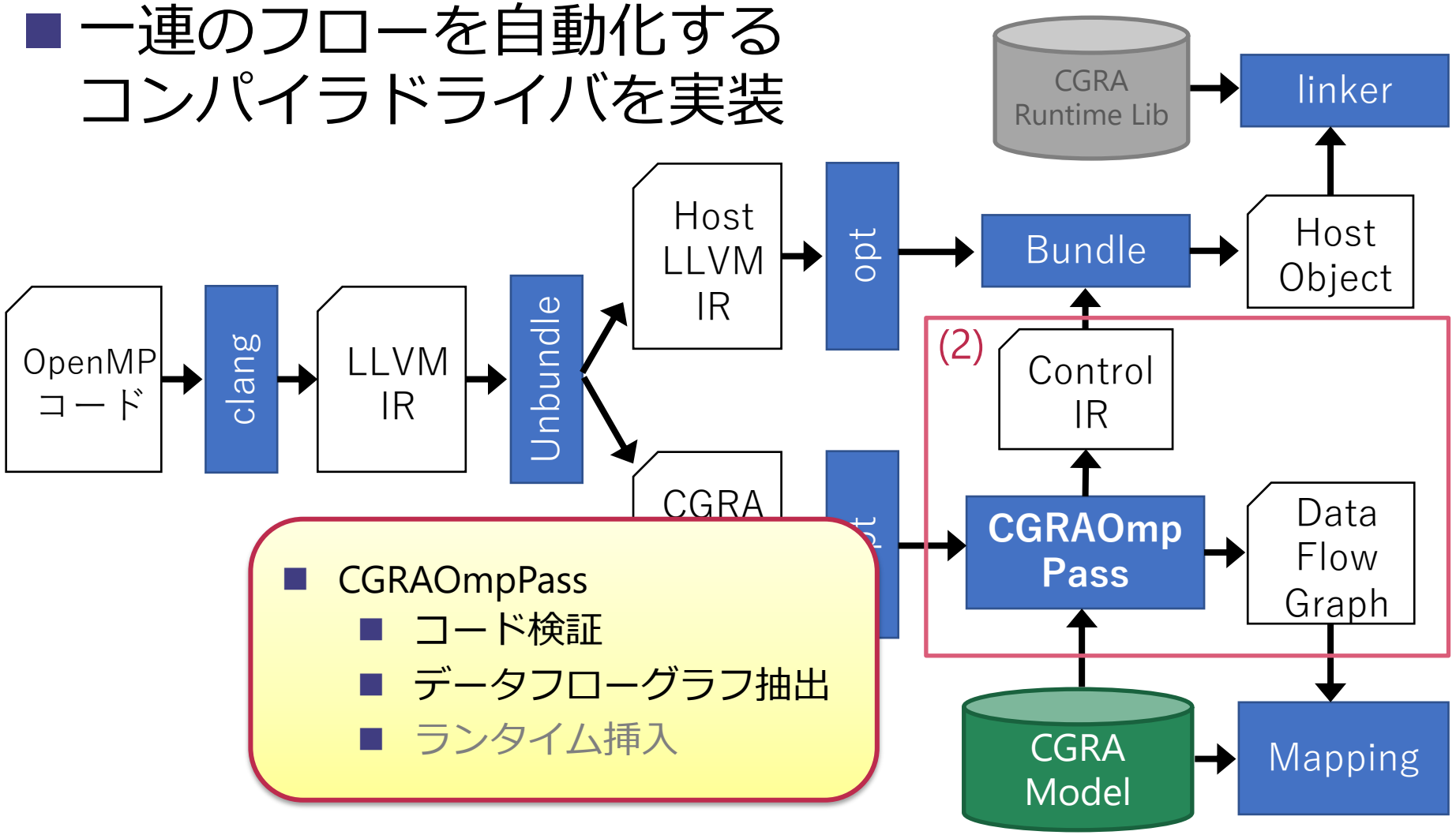
- 一連のフローを自動化する
  コンパイラドライバを実装



- clang-offload-bundler
  - シングルソースによるヘテロジニアス
    コンピューティングを支援するツール

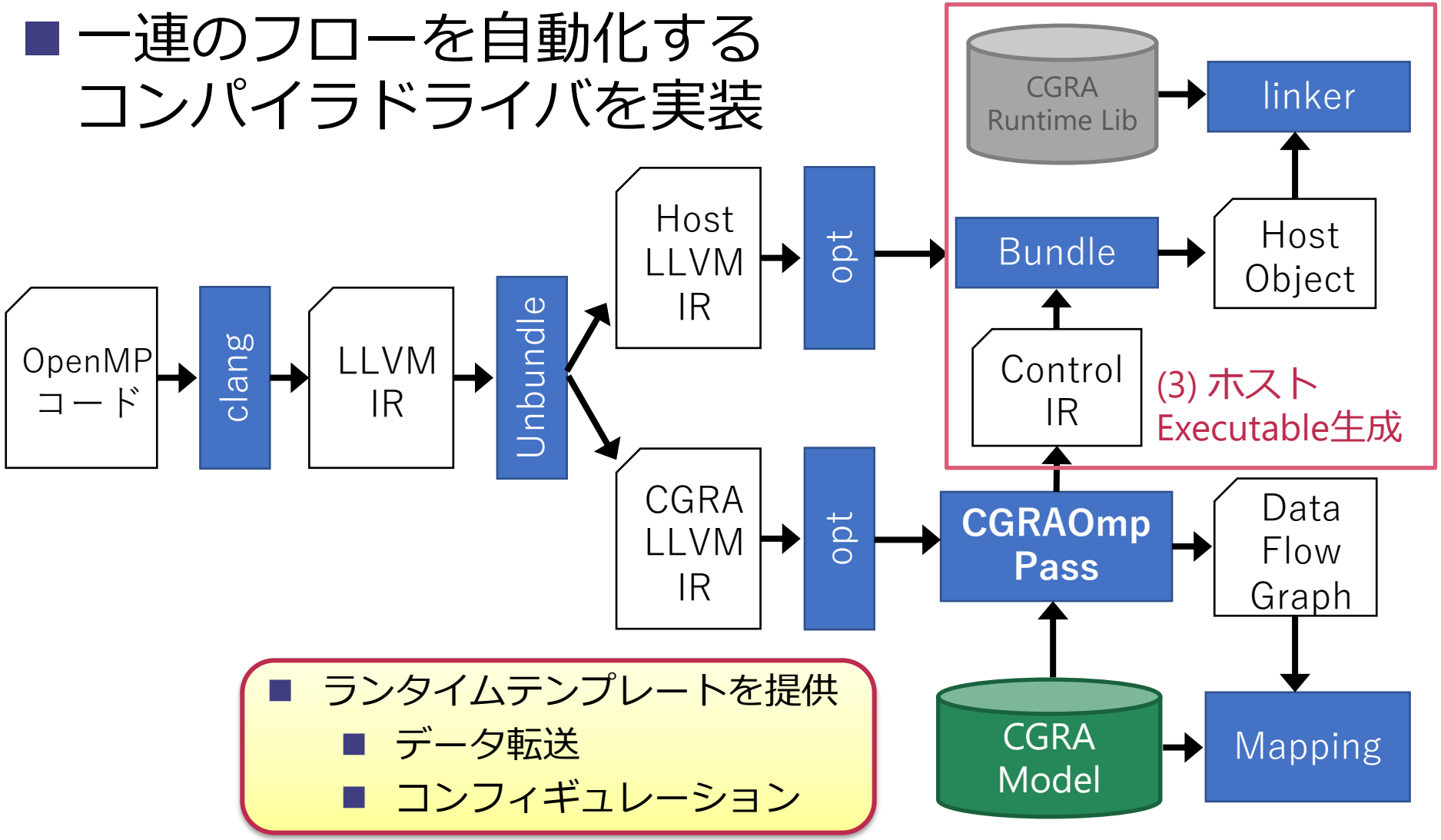
CGRA OpenMPのコンパイルフロー

- 一連のフローを自動化する
  コンパイラドライバを実装



CGRA OpenMPのコンパイルフロー

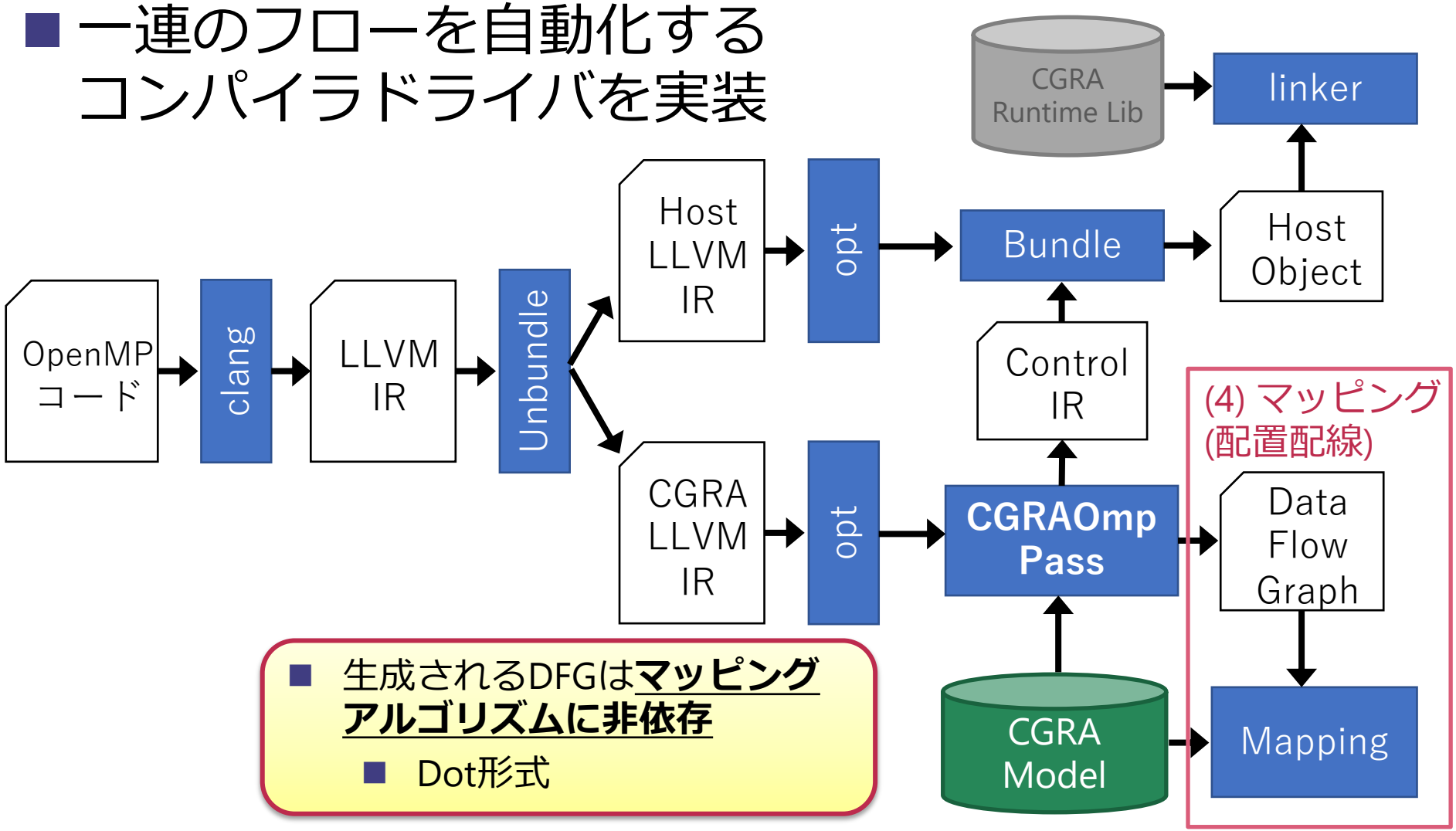
- 一連のフローを自動化する
  コンパイラドライバを実装



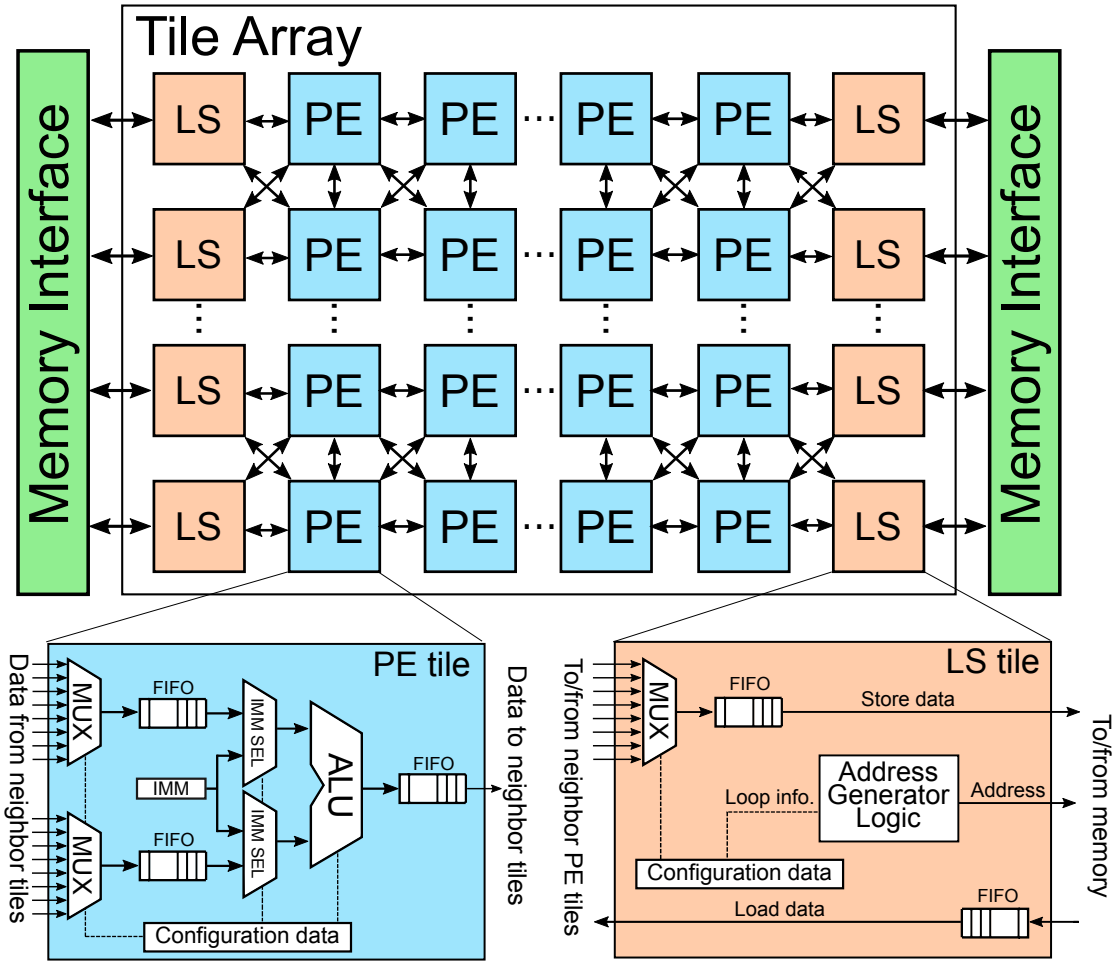
- ランタイムテンプレートを提供
  - データ転送
  - コンフィギュレーション

CGRA OpenMPのコンパイルフロー

- 一連のフローを自動化する
  コンパイラドライバを実装



RIKEN CGRA [4] (再掲)



RIKEN CGRAの概要

- 設計探索を目的として設計テンプレート
  - SystemVerilogで実装
- 2種のタイルで構成
  1. LS (Load/Store)
     - ループ情報を基にデータアクセス
  2. PE
     - 各種数値演算
- Spatial方式の再構成
  - FIFOで遅延差を吸収

CGRAモデル

- CGRAの実行モデル、再構成方式などを記述(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"}
  ]
}
```

RIKEN CGRA用の記述

CGRAモデル

- CGRAの実行モデル、再構成方式などを記述(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"}
  ]
}
```

- CGRAの分類
- Decoupled
  - メモリアクセスと計算パートを分離した実行モデル[6]

RIKEN CGRA用の記述

CGRAモデル

- CGRAの実行モデル、再構成方式などを記述(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"}
  ]
}
```

- メモリアクセスの表現可能パターン
  - アフィンアクセス
  - 3重ループ
  - $C_0 + C_1 v_1 + C_2 v_2 + C_3 v_3$

RIKEN CGRA用の記述

CGRAモデル

- CGRAの実行モデル、再構成方式などを記述(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"}
  ]
}
```

- 対応可能な制御フロー
- この例では
  - 条件分岐不可
  - ループ間依存不可

RIKEN CGRA用の記述

CGRAモデル

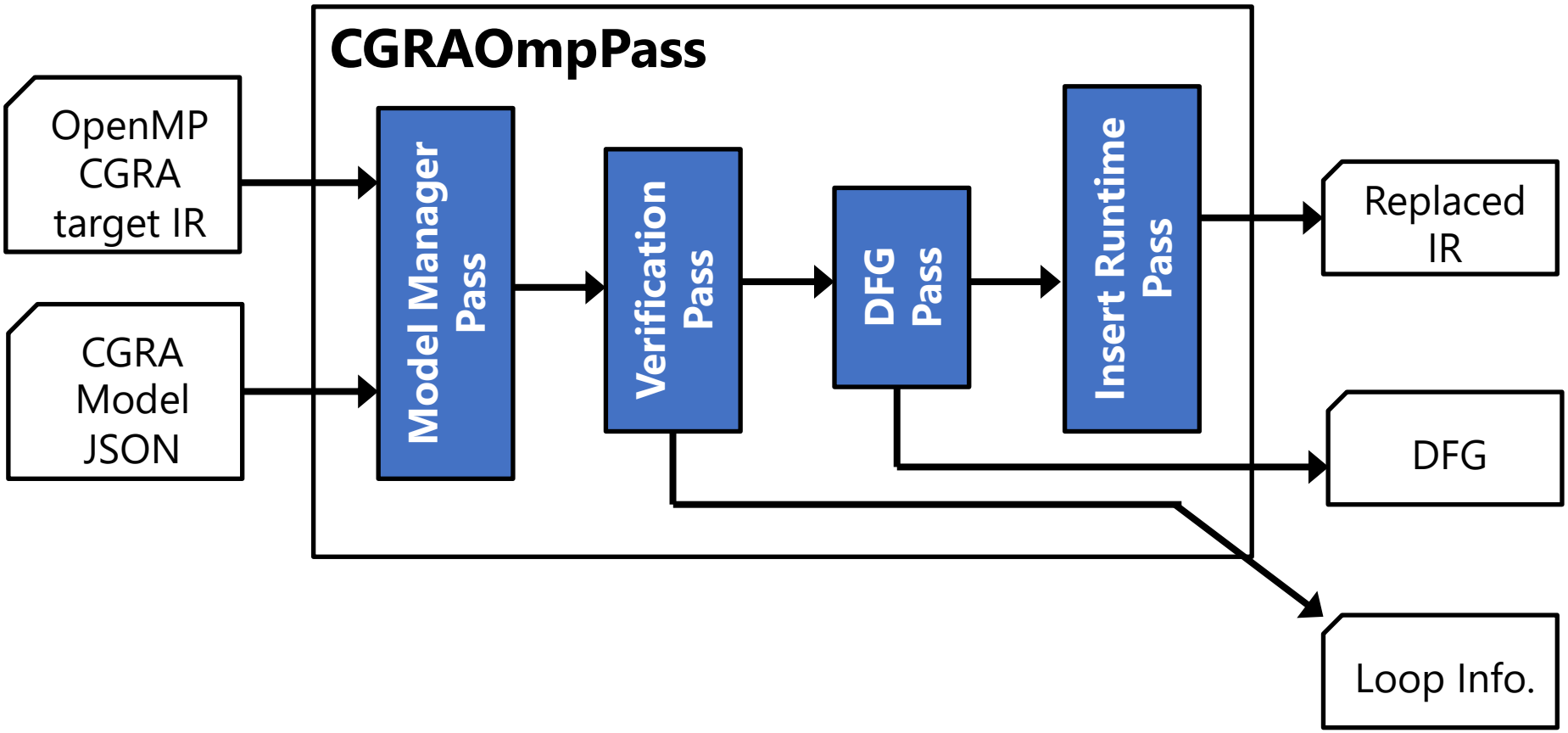
- CGRAの実行モデル、再構成方式などを記述(JSON)

```
{
  "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"}
  ]
}
```

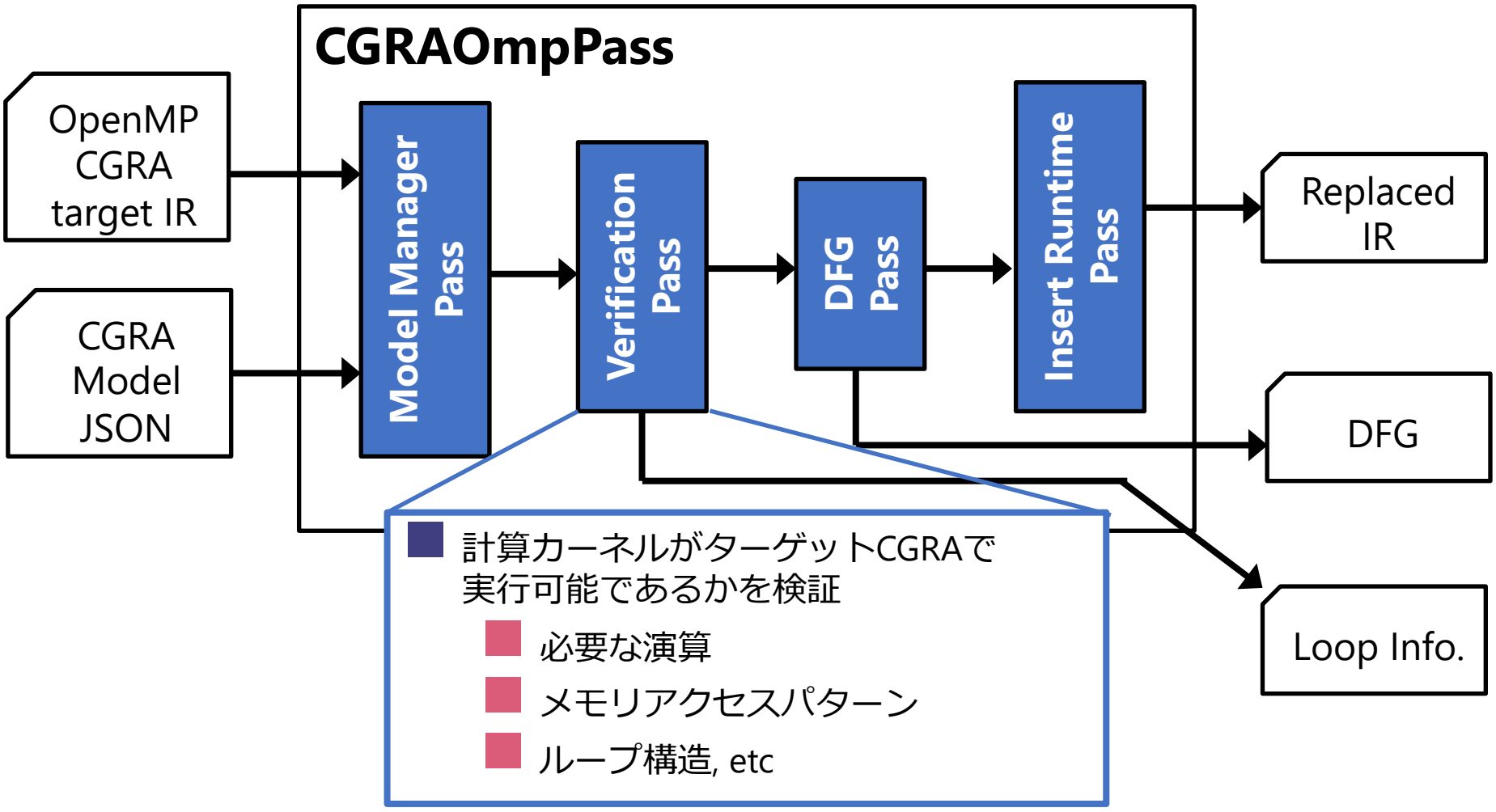
- ALUがサポートする命令情報
- custom_instructions: LLVM IRに存在しない命令
  - ソースコード中では同名の関数呼び出し
- generic_instructions: LLVM IRに対応する命令
- instruction map: LLVM IR命令->ALU opへの対応
  - 変換条件を指定可能

RIKEN CGRA用の記述

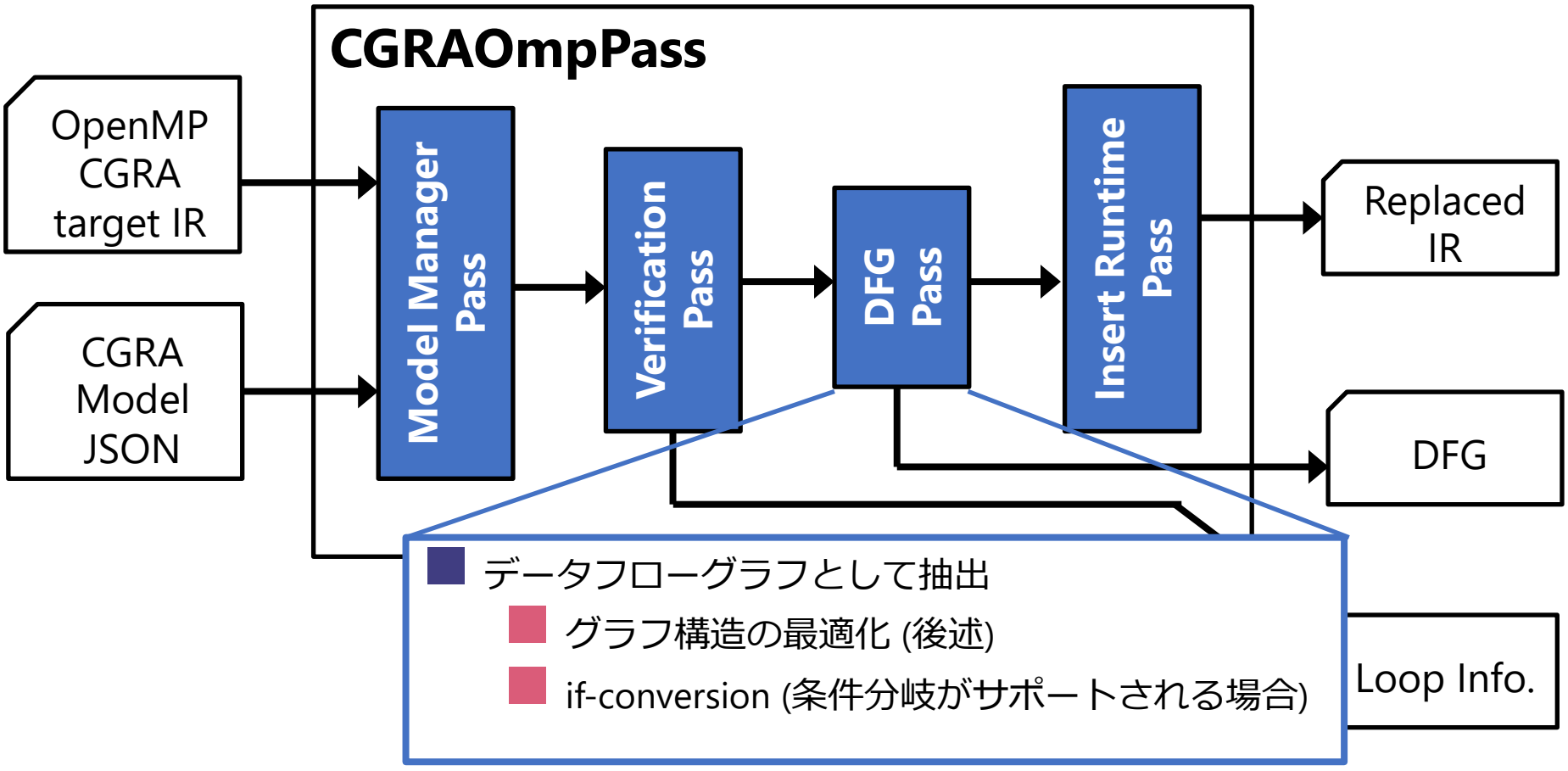
CGRAOmpPassのフロー



CGRAOmpPassのフロー



CGRAOmpPassのフロー



コード例: 3x3 covolution

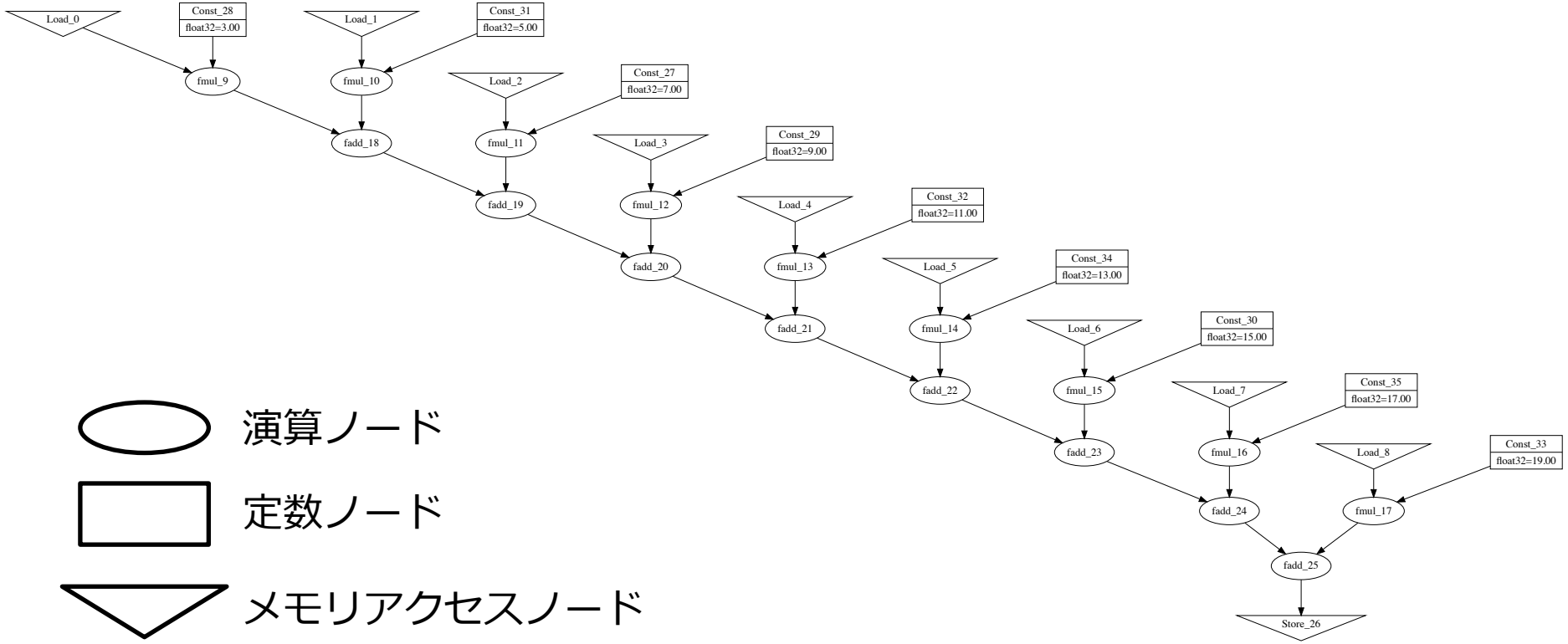
```
void conv(float * indata, float * outdata)
{
  int irow, icol, orow, ocol, krow, kcol;
  float tmp;
  orow = 0;
  #pragma omp target parallel for map(to:indata) map(from:outdata)
    private(icol, ocol, orow, krow, kcol)
  for (irow = 0; irow <= (H-K_H); irow += STRIDE) {
    for (icol = 0, ocol = 0; icol <= (W-K_W); icol += STRIDE, ocol++) {
      const int weight[] = {3, 5, 7, 9, 11, 13, 15, 17, 19};
      tmp = ZERO;
      #pragma unroll
      for (krow = 0; krow < K_H; krow++) {
        #pragma unroll
        for (kcol = 0; kcol < K_W; kcol++) {
          tmp += indata[(irow + krow) * W + icol + kcol] * weight[krow * K_W + kcol];
        }
      }
      outdata[orow + O_W + ocol] = tmp;
    }
  }
}
```

コンパイラドライバによる実行の様子

```
bash $clang-cgraomp test/conv.c -o test/conv -cc share/presets/decoupled_affine_AG.
json -save-temps --enable-cgraomp-debug
Clang front-end                                          : [  OK  ]
OpenMP target unbundling                                 : [  OK  ]
Optimization of host code                                : [  OK  ]
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_1d40416_conv_l178
        [INFO]: Verifying a kernel for decoupled CGRA: .omp_outlined.
        [INFO]: Detected perfectly nested loop: omp.inner.for.body Nested level 2
        [INFO]: Verifying Affine AG compatibility of a loop: omp.inner.for.body
        [INFO]: Verifying Affine AG compatibility of a loop: omp.inner.for.body
omp.inner.for.body
  %mul24 = fmul fast float %9, 3.000000e+00 1
  %mul24.1 = fmul fast float %11, 5.000000e+00 1
  %mul24.2 = fmul fast float %14, 7.000000e+00 1
  %mul24.114 = fmul fast float %17, 9.000000e+00 1
  %mul24.1.1 = fmul fast float %19, 1.100000e+01 1
  %mul24.2.1 = fmul fast float %21, 1.300000e+01 1
  %mul24.225 = fmul fast float %24, 1.500000e+01 1
  %mul24.1.2 = fmul fast float %26, 1.700000e+01 1
  %mul24.2.2 = fmul fast float %28, 1.900000e+01 1
  %add25.1 = fadd fast float %mul24.1, %mul24 1
  %add25.2 = fadd fast float %add25.1, %mul24.2 1
  %add25.115 = fadd fast float %add25.2, %mul24.114 1
  %add25.1.1 = fadd fast float %add25.115, %mul24.1.1 1
  %add25.2.1 = fadd fast float %add25.1.1, %mul24.2.1 1
  %add25.226 = fadd fast float %add25.2.1, %mul24.225 1
  %add25.1.2 = fadd fast float %add25.226, %mul24.1.2 1
  %add25.2.2 = fadd fast float %add25.1.2, %mul24.2.2 1
```

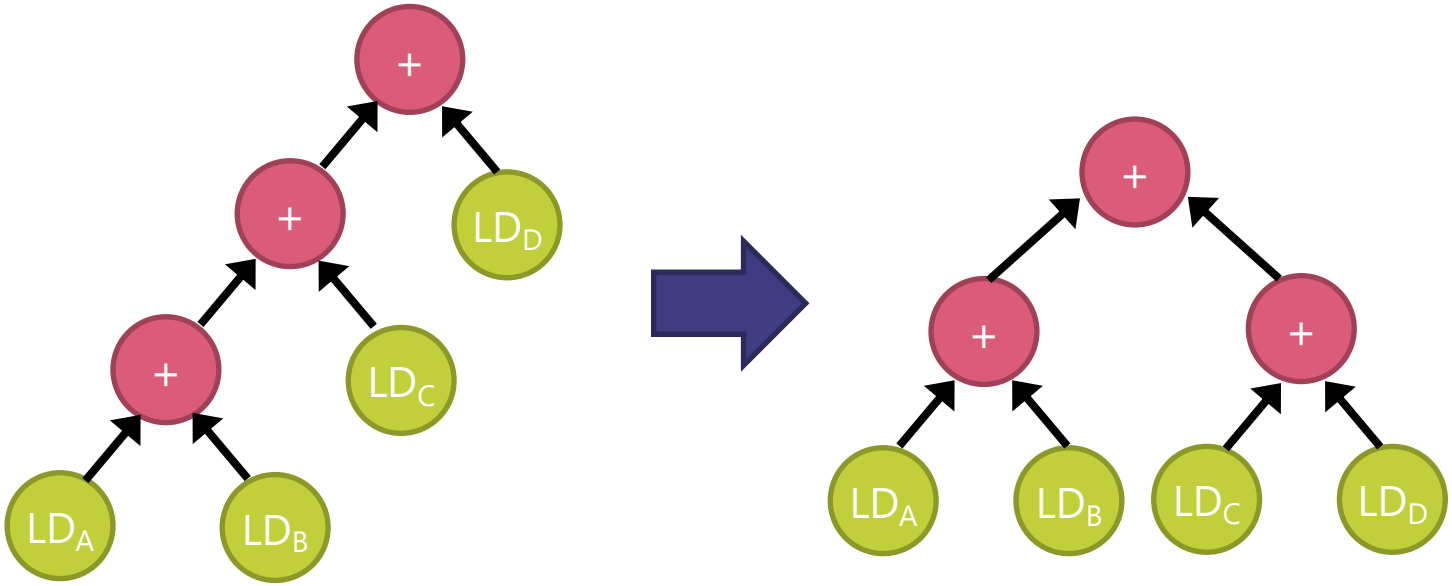
生成されたDFG

- LLVM IRにおけるデータ依存をそのまま利用すると木構造がアンバランスになる

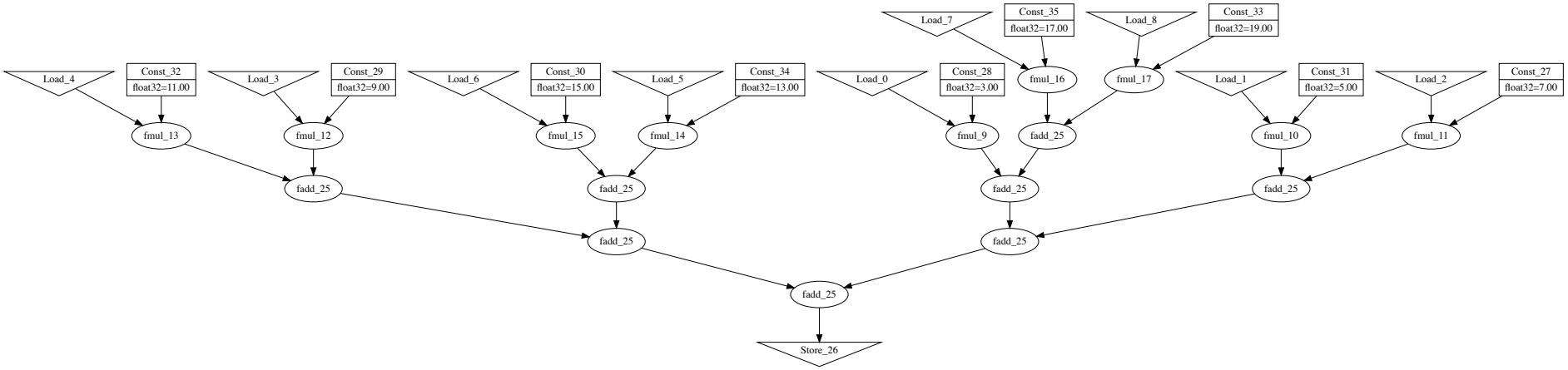


Tree-Height-Reductionを追加

- LSI設計やHDLの高位合成で広く用いられる[18]
  - 演算子の交換則、結合則に基づきグラフを変形
  - 本研究では動的ハフマン符号ベースの手法を実装[19]



変形後のDFG



演算ノード

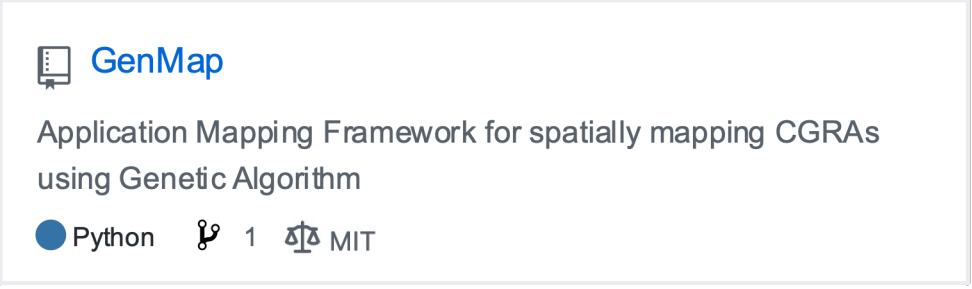
定数ノード

メモリアクセスノード

評価

評価環境

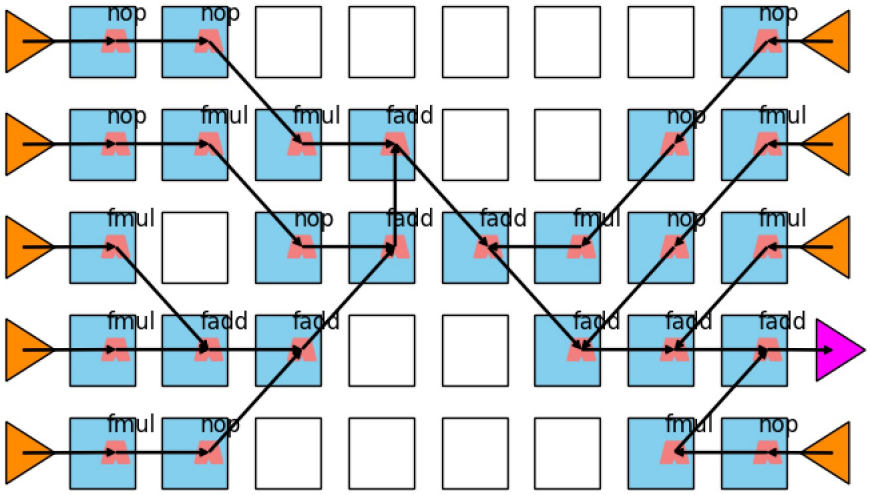
- LLVM version 12.0.1
- CGRA構成
  - 8x10アレイ (8x8 PEタイル + 8+8LSタイル)
- ベンチマーク: 3x3 convolution
- 配置配線バックエンド
  - GenMap[20]をRIKEN CGRAへ移植
  - 遺伝的アルゴリズムベースのマッピング最適化



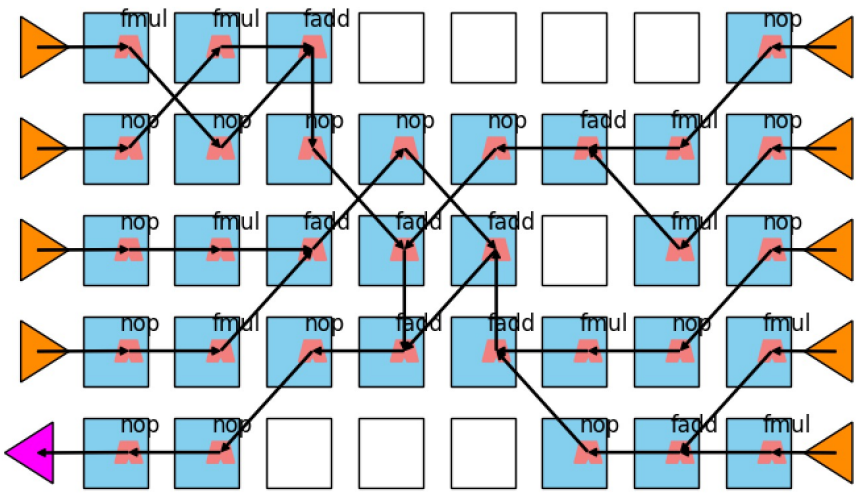
https://github.com/hungalab/GenMap

マッピング結果の比較

		総配線長	マップ面積	レイテンシ差
naïve DFG	面積優先	40.38	40 (5 x 8)	6
	レイテンシ優先	50.56	56 (7 x 8)	2
Optimized DFG		46.21	40 (5 x 8)	0

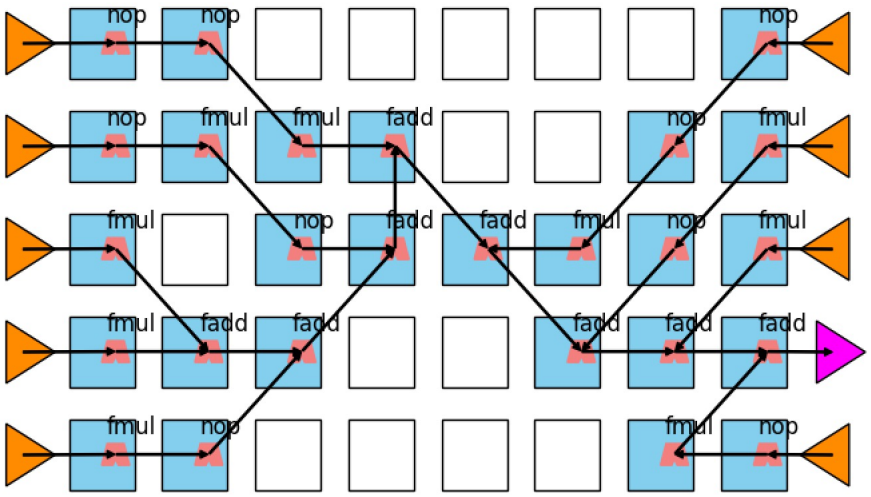


naïve DFG 5x8 mapping

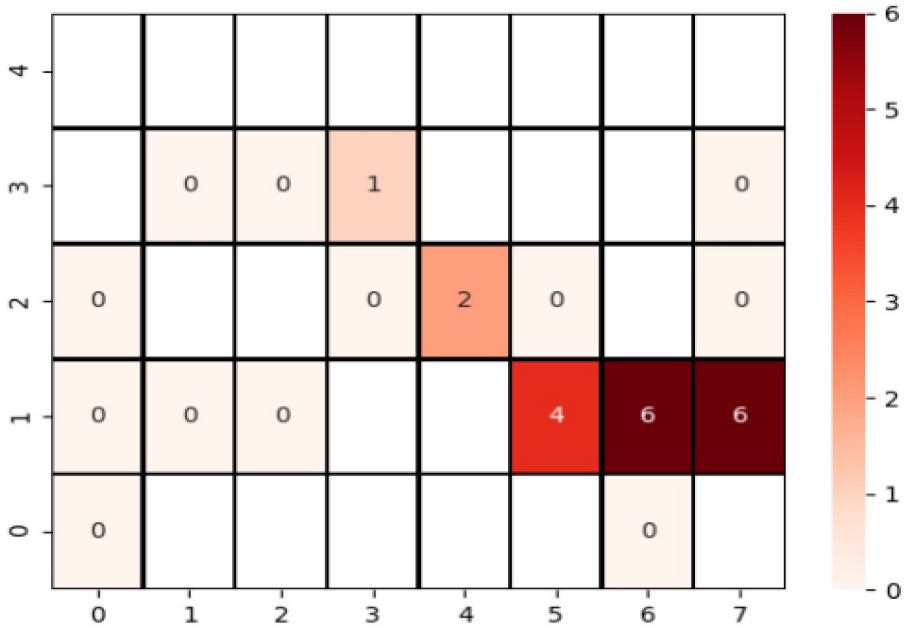


Optimized DFG 5x8 mapping

レイテンシ差の解析



naïve DFG 5x8 mapping



演算がマップされたPEにおける
レイテンシの差

結論と今後の展望

- 本研究では
  - CGRA向けOpenMPコンパイラの実装
  - データフローグラフに対してTree-Height-Reductionを適用
  - マッピング結果に与える影響の確認
- 今後の展望
  - 他方式のCGRAへの対応
  - ベンチマークセットを用いた評価
    - PolyBenchのOpenMP実装など
  - ランタイム部の実装
  - シミュレーター、FPGAオーバーレイとの連携

文献リスト

[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.

文献リスト

[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.

文献リスト

[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.

文献リスト

[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.