2022 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION JAPAN

IEEE2804 SHIM: Software-Hardware Interface for Multi-Many-Core

名古屋大学大学院情報学研究科
枝廣　正人

Multicore Tools
SHIM
Multicore HW

accellera SYSTEMS INITIATIVE

アウトライン

- SHIMについて
- SHIMの構造
- SHIMの利用・応用
- SHIMを用いた性能見積技術

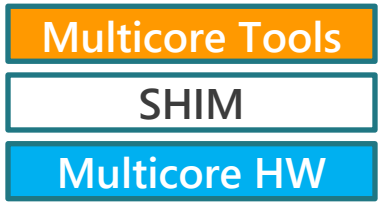
SHIM (Spacer)

シム （Wikipediaより）

- 詰め物、くさび、などを意味する英語
- 機械、構築物の高さ・隙間調整
  のために用いるスペーサー、ライナー

IEEE2804 SHIMとは?

- Software-Hardware Interface for Multi-Many-Core
  - 現バージョンはSHIM2.0 (IEEE2804-2019)
  - https://standards.ieee.org/ieee/2804/7477/
  - ハードウェアとソフトウェアの間をつなぐもの



- マルチコア(マルチプロセッサ)向け設計ツール(例:自動並列化ツール)やOS/ミドルウェアは、ハードウェアの詳細を隠蔽し、アプリケーションがマルチコア上で最適動作することを支援する
- しかしながらツール自体はハードウェア情報を知っている必要がある
- SHIMは、XML形式で標準化された記述により、機械的に読める形でハードウェア情報を設計ツールに提供する

SHIMが解決する課題

- 課題
  - マルチコアやメニーコアアーキテクチャの自由度は高い
  - アーキテクチャ情報は、設計ツールや実行環境が並列化、最適化、解析、管理するために極めて重要である
  - アーキテクチャ情報は独自形式、多くの場合自然言語（何百～何千頁ものドキュメント）となっている
  - このことが、新しいハードウェアにとって、高いツールサポートコスト、低いツール利用可能性、時に誤解のもとになっており、最新ハードウェアの利用阻害要因になっている

- 解決法
  - マルチコアアーキテクチャ情報に対する標準的記述の定義
  - ベンダがXML形式でハードウェア情報を提供 ⇒ ツールベンダが利用

SHIMのメリット

- メリット
  - チップベンダ、システムベンダ、プログラム開発者にとってより多くの設計ツールが利用可能になる
  - エンドユーザーにとってtime-to-marketがより早くなる
  - ハードウェアとソフトウェアのベンダ間におけるより正確なインターフェースとドキュメントを提供する

SHIMは・・・

- XMLスキーマで定義されたインタフェース
  - スキーマにもとづき記述もしくは生成されたXML形式のハードウェア記述
- マルチコア設計ツールが利用すると想定されるハードウェア情報を抽出
  - プロセッサコアの種類、数、同期メカニズム、コア間通信、メモリシステム等
- ソフトウェア視点で記述されたハードウェアモデル

- ハードウェアの動作仕様ではない – 構成や遅延に関する説明である
- 100%の説明ではない – ソフトウェアが利用すると想定される情報のみ
- ツールそのものではない – SHIMを利用するツールが様々なベンダにより開発される

SHIMができること・できないこと

- ツールによるソフトウェア性能の概算見積を支援する
- ハードウェア情報により、ツール自身のコンフィギュレーションや、コンフィギュレーションUIを自動生成することを支援する
- デバイスドライバやハードウェア仮想レイヤ (HAL) のコンフィギュレーションを支援する

- 100%精度のソフトウェア性能見積をするものではない
  - おおよそ±20%程度(80～120%)の精度と考えられている
- ハードウェア仮想化レイヤ (HAL) の自動生成ではない

SHIMの位置づけ

SHIM XML

.xml

.xml

.xml

自動並列化
コンパイラ

並列支援ツール

性能解析ツール

OS/ミドルウェア
コンフィギュレータ

SHIMのユースケース

- 性能見積
  - ほとんどの設計支援ツールにおいて重要
  - 例：自動並列化コンパイラ、並列化支援ツール、性能解析ツール等
- システムコンフィギュレーション
  - OS、ミドルウェア、ランタイムライブラリではコンフィギュレーション時に基本的なアーキテクチャ情報が必要
  - 上述の設計支援ツールでも同様
- ハードウェアモデル
  - シミュレータ等におけるハードウェアモデル
  - アーキテクチャ探索にも有用である可能性

SHIMの生い立ち

- 2012-2014: NEDOプロジェクトにおいて仕様策定
  - NEDO戦略的省エネルギー技術革新プログラム／実用化開発「多様なマルチ・メニーコアの高度な活用を可能にする標準プラットフォーム開発とエコシステム構築による省エネルギー技術の実用化」
  - 下部委員会(ルネサス、eSOL、TOPS、東芝、CATS、NEC、早大、名大)にて策定
- 2014.10: 組込みマルチコアコンソーシアム設立
- 2015.2: Multicore Association (MCA)からSHIM1.0をリリース
- 2015.6: MCAからSHIM1.0日本語版をリリース
- 2016頃からSilexica社を中心にSHIM2.0検討開始
- 2019.1: MCAからSHIM2.0をリリース
- 2020.1: IEEEからSHIM2.0を標準化 (IEEE2804-2019)

アウトライン

- SHIMについて
- SHIMの構造
- SHIMの利用・応用
- SHIMを用いた性能見積技術

トップレベル

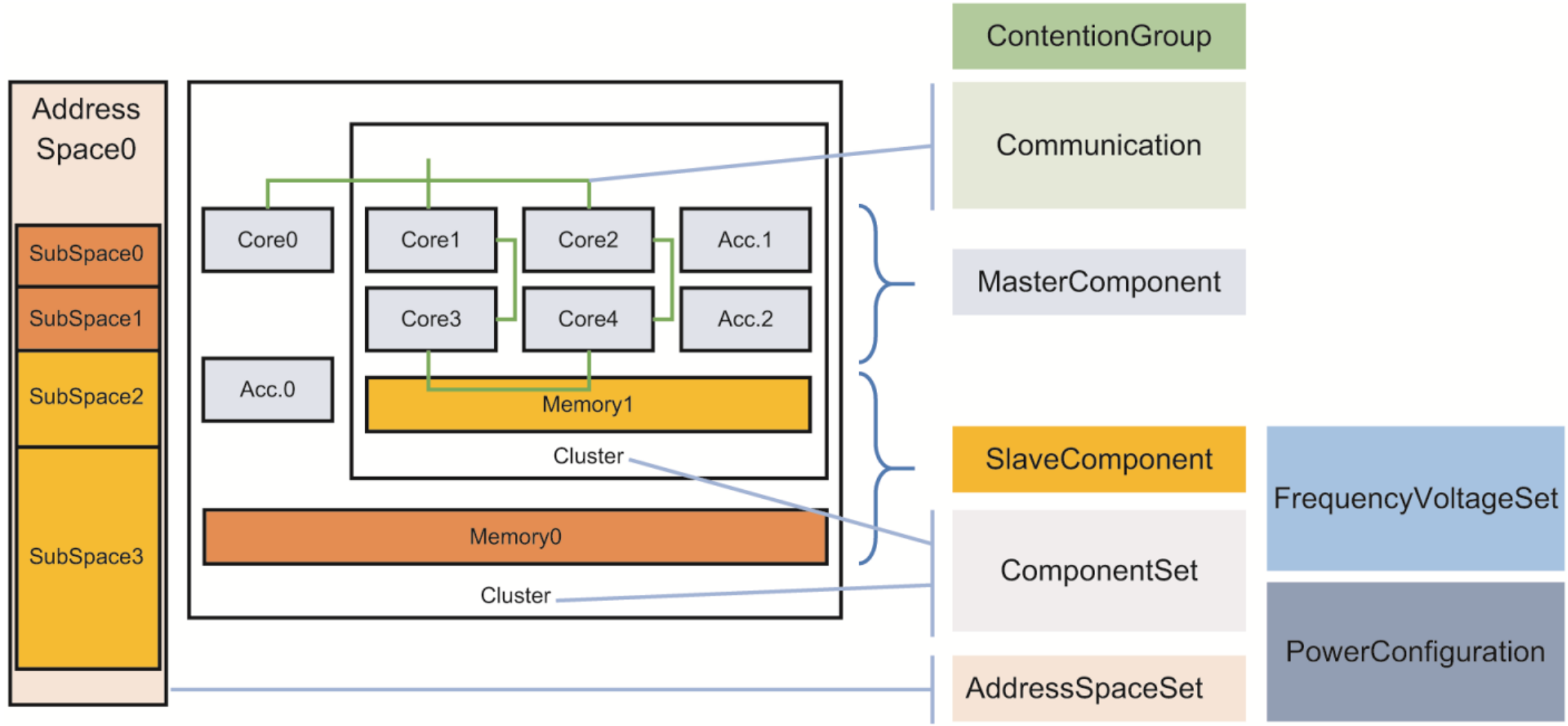
- SHIM
  - SystemConfiguration（次頁）
  - PowerConfiguration
    - Component(s)の動作ポイント（後述）ごとの電力
  - VendorExtension
    - ベンダ固有ライブラリのような形でSystemConfigurationとPowerConfigurationの組を定義可能
  - FunctionalUnitSet
    - 後述。プロセッサ内部の機能ユニット（命令遅延等）をライブラリのように定義可能。並列プロセッサにおいて記述を一回に

SystemConfiguration

- SystemConfiguration
  - ComponentSet（階層化可能、クラスタ）
    - MasterComponent（プロセッサ）
    - SlaveComponent（メモリ）
    - Cache
  - CommunicationSet（Master間通信）
  - AddressSpaceSet（アドレス空間）
    - Master－Slave間遅延もここに記述
  - FrequencyVoltageSet（動作周波数/電圧。動作ポイントを定義）
  - ContentionGroupSet
    - バス等での複数Master-Slave間共有を記述

注意：すべてを記載しているわけではない

マルチコアアーキテクチャとの対応



Note that not all relations are illustrated

MasterComponent

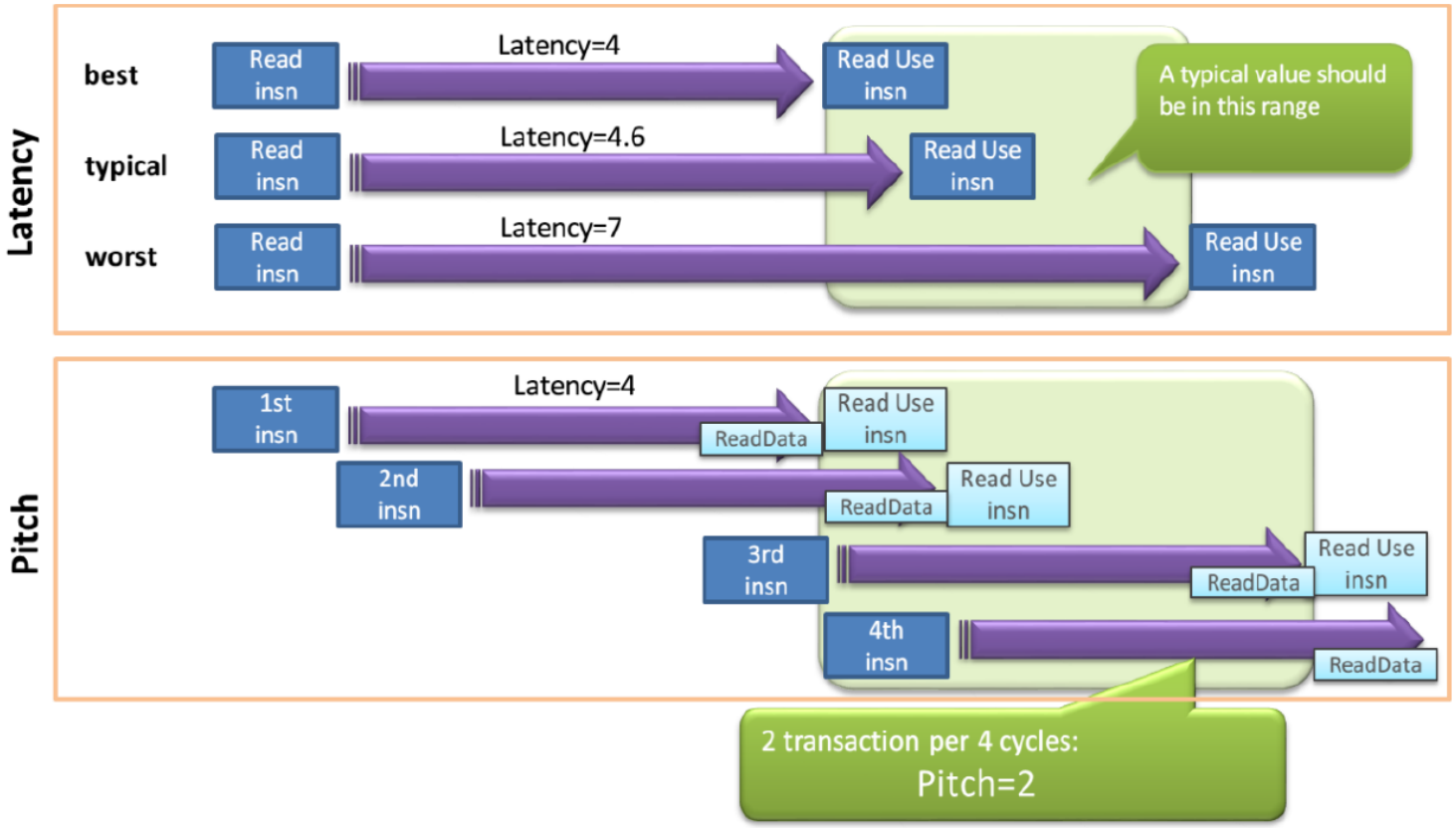
- MasterComponent（プロセッサ）
  - CommonInstructionSet
    - FunctionalUnitSet
      - FunctionalUnit
        - Instruction
          - 命令名
          - Performance（命令レイテンシ）
        - CustomInstruction
          - 命令名
          - Performance

基本命令セットは
LLVM-IR
（LLVMコンパイラ基盤の中間表現。addなどアセンブラレベルの表現）

注意：すべてを記載しているわけではない

Performance

- Performance
  - Latency
    - 1命令の実行サイクル
    - Best（最良）, Typical（最頻）, Worst（最悪）
  - Pitch
    - 連続命令間の実質実行サイクル
    - Best（最良）, Typical（最頻）, Worst（最悪）



CommunicationSet

- CommunicationSet
  - SharedRegisterCommunication
  - SharedMemoryCommunication
  - EventCommunication
  - FIFOCommunication
  - InterruptCommunication

注意：すべてを記載しているわけではない

  - 各Communicationでは、
    - 送信側MasterComponent、受信側MasterComponentを指定
    - 一度に送信可能なデータサイズ等を指定

AddressSpaceSet

- AddressSpaceSet（複数のアドレス空間を定義可能）
  - AddressSpace（一つのアドレス空間）
    - SubSpace（アドレス区間）
      - 開始アドレス、終了アドレス
      - MasterSlaveBindingSet
        - MasterSlaveBinding
          - SlaveComponentを参照
          - Accessor
            - MasterComponentを参照
            - PerformanceSet
              - アクセス種類（例：4 byte read）ごとにPerformanceを記載

注意：すべてを記載しているわけではない

FrequencyVoltageSet

- FrequencyVoltageSet
  - FrequencyDomain（名前とid）
  - VoltageDomain（名前とid）
  - OperatingPointSet
    - 名前とid
    - OperatingPoint（動作ポイント）
      - 名前、id、周波数値、電圧値

注意：すべてを記載しているわけではない

- 各(Master, Slave, Cache)Componentには、FrequencyDomain、VoltageDomain、OperatingPointSetへの参照を記載でき、当該Componentが含まれる周波数ドメインや電圧ドメイン、実行可能な動作ポイントを記述可能

ContentionGroupSet

- ContentionGroupSet
  - ContentionGroup
    - Throughput [byte/cycle]またはDataRate [byte/sec]
    - PerformanceSetRef
      - AddressSpace中のMasterSlaveBindingで記述されたPerformanceSetを参照

注意：すべてを記載しているわけではない

- 各ContentionGroupは一箇所のハードウェア（例:バス)に対応し、最大スループット（またはDataRate）と、そのハードウェアを共有するMasterSlaveBindingの集合を記述する

SHIM記述（一部）

1. All SystemConfiguration root elements

```
1   <?xml version="1.0" encoding="UTF-8"?>
2   <shim:Shim
3       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4       xmlns:shim="http://www.multicore-association.org/2017/SHIM2.0/"
5       xsi:schemaLocation="http://www.multicore-association.org/2017/SHIM2.0/ ../schemas/shim20.xsd"
6       name="MySystem" shimVersion="2.0">
7
8   <SystemConfiguration>
9       <ComponentSet name="CS_GENPLAT_MODULE" id="CS_GENPLAT_MODULE"> [135 lines]
145     <CommunicationSet> [21 lines]
167     <AddressSpaceSet> [161 lines]
329     <FrequencyVoltageSet> [30 lines]
360     <ContentionGroupSet> [37 lines]
398 </SystemConfiguration>
399 </shim:Shim>
400
```

2. ComponentSet expanded

```
5       <ComponentSet name="CS_GENPLAT_MODULE" id="CS_GENPLAT_MODULE">
6           <ComponentSet name="CS_GENPLAT_CLUSTER0" id="CS_GENPLAT_CLUSTER0">
7               <ComponentSet name="CS_GENPLAT_CLUSTER0_CPU" id="CS_GENPLAT_CLUSTER0_CPU">
8                   <MasterComponent masterType="PU" arch="GENERIC_RISC_CPU" name="MC_GENPLA
38                  <MasterComponent masterType="PU" arch="GENERIC_RISC_CPU" name="MC_GENPLA
68                  <MasterComponent masterType="PU" arch="GENERIC_RISC_CPU" name="MC_GENPLA
98                  <MasterComponent masterType="PU" arch="GENERIC_RISC_CPU" name="MC_GENPLA
128                 <Cache name="CA_GENPLAT_CLUSTER0_L2" id="CA_GENPLAT_CLUSTER0_L2" cacheTy
129             </ComponentSet>
130         </ComponentSet>
131         <SlaveComponent name="SC_GENPLAT_EXTMEM_DDR" id="SC_GENPLAT_EXTMEM_DDR" size="1"
132     </ComponentSet>
```

3. Performance under AddressSpaceSet

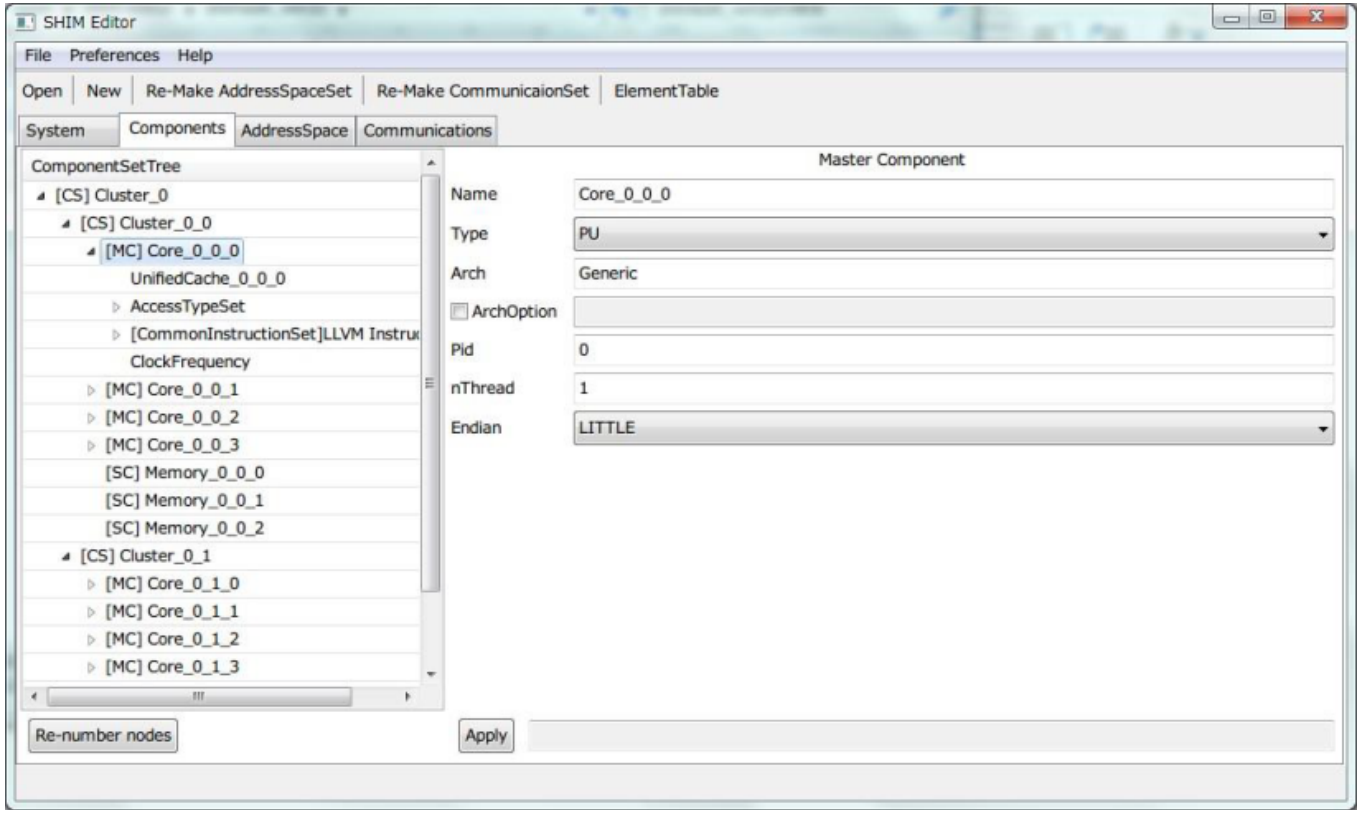
```
155     <AddressSpaceSet>
156         <AddressSpace name="AS_GENPLAT_MAIN" id="AS_GENPLAT_MAIN">
157             <SubSpace name="SS_GENPLAT_DDR" id="SS_GENPLAT_DDR" start="0" end="1073741824" >
158                 <MasterSlaveBindingSet>
159                     <MasterSlaveBinding slaveComponentRef="SC_GENPLAT_EXTMEM_DDR">
160                         <Accessor masterComponentRef="MC_GENPLAT_CLUSTER0_CPU0">
161                             <PerformanceSet id="PS_GENPLAT_CLUSTER0_DDR_CPU0_RDHITL1" cacheRef="
162                                 <Performance accessTypeRef="AT_GENPLAT_CLUSTER0_CPU0_RD64">
163                                     <Pitch   best="1.0" typical="1.0" worst="1.0" />
164                                     <Latency best="1.0" typical="1.0" worst="1.0" />
165                                 </Performance>
166                             </PerformanceSet>
167                             <PerformanceSet id="PS_GENPLAT_CLUSTER0_DDR_CPU0_RDHITL2" cacheRef="
168                                 <Performance accessTypeRef="AT_GENPLAT_CLUSTER0_CPU0_RD64">
```

アウトライン

- SHIMについて
- SHIMの構造
- SHIMの利用・応用
- SHIMを用いた性能見積技術

SHIM作成

- SHIM Editor
  - https://github.com/openshim/shim2/tree/master/tools



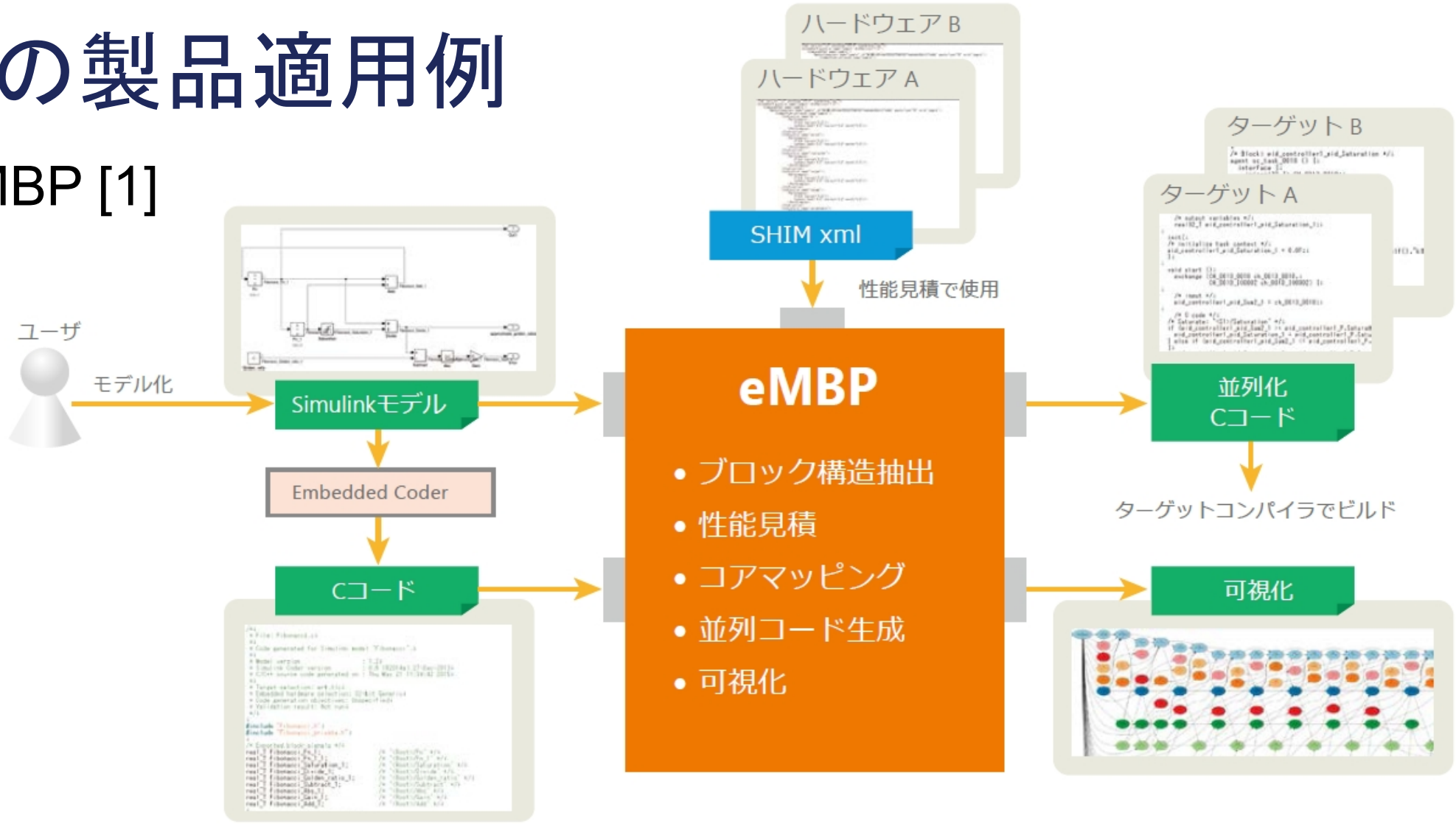
SHIMの作成は誰が？

- チップベンダのようだが
  - プロセッサからメモリへのレイテンシは、使っているメモリや、ハードウェアの作りに依存するため、チップベンダが記述するのは難しい
  - プロセッサ内の命令レイテンシもチップだけの話ではない（後の話題）
  - 安易なチップ比較や責任問題になることをチップベンダは好まない

- プラットフォームベンダやユーザによる作成も
  - 実行環境や開発環境を搭載したSoCプラットフォーム環境に対し、プラットフォームベンダが当該環境上での計測値をSHIM XMLとして作成
  - ユーザにも利用可能な生成ツール
  - 参考：SHIM XMLファイルの命名に関する推奨は domainName.hardwarePlatformName.platformVersion.compilerName.compilerVersion.xml

SHIMの製品適用例

- eSOL MBP [1]



[1] https://www.esol.co.jp/embedded/mbp.html

SHIMの製品適用例

- Silexica SLX [2]

Silexica、最新版SLXをリリース｜業界初FPGAのHW/SW分割機能を搭載

SLXとは･･･

SLXは、複雑なマルチコア向けプログラミングを自動化するための、並列ソフトウェア開発ツール・スイート（SLX Tool Suite）です。SHIM（Software-Hardware Interface for Multi-many-core）で記述された抽象度の高いハードウェア・プラットフォーム情報を別途用意することで、シングル・スレッドで記述されたC/C++コードを並列化します。
またAutosarで記述されたコードをマルチコアに分割する機能や、FPGAへの自動実装機能も提供しています。

以下、SLXの中でも人気の高い2製品SLX for C/C++とSLX for FPGAに関して詳しく紹介させていただきます。

1． SLX for C/C++: 既存C/C++コードの並列化

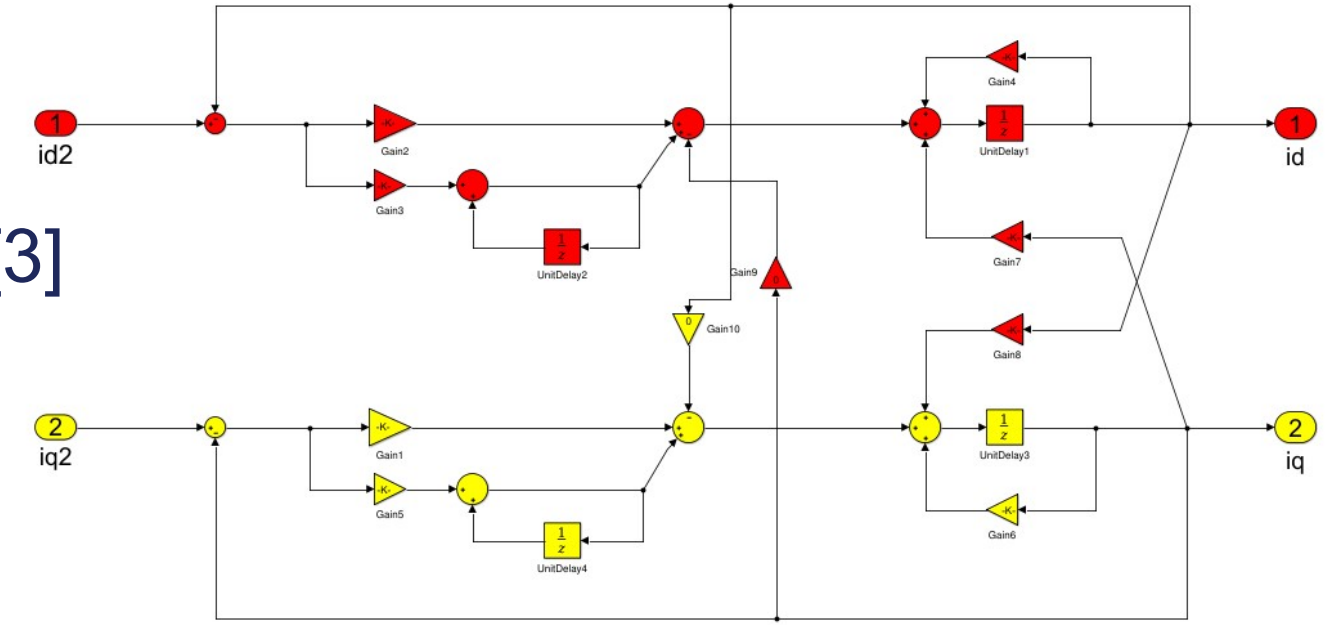
[2] https://www.eda-express.com/hot_topix/2019/01/silexicaslxfpgahwsw-1.html

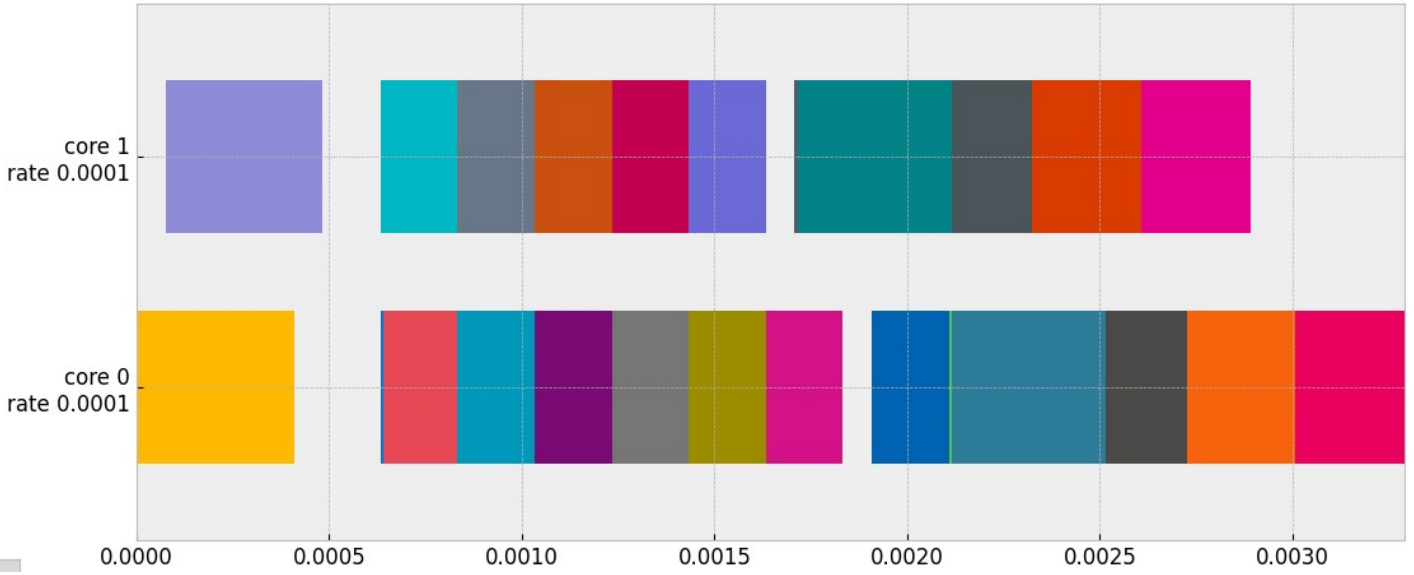
SHIM利用研究例（名古屋大学）

1. モデルベース並列化（MBP）
2. UnitDelay追加によるモデルレベル性能改善
3. SHIMulator
4. 性能見積技術（見積手法とSHIMに記載する遅延値について）

研究例1：モデルベース並列化(MBP) [3]

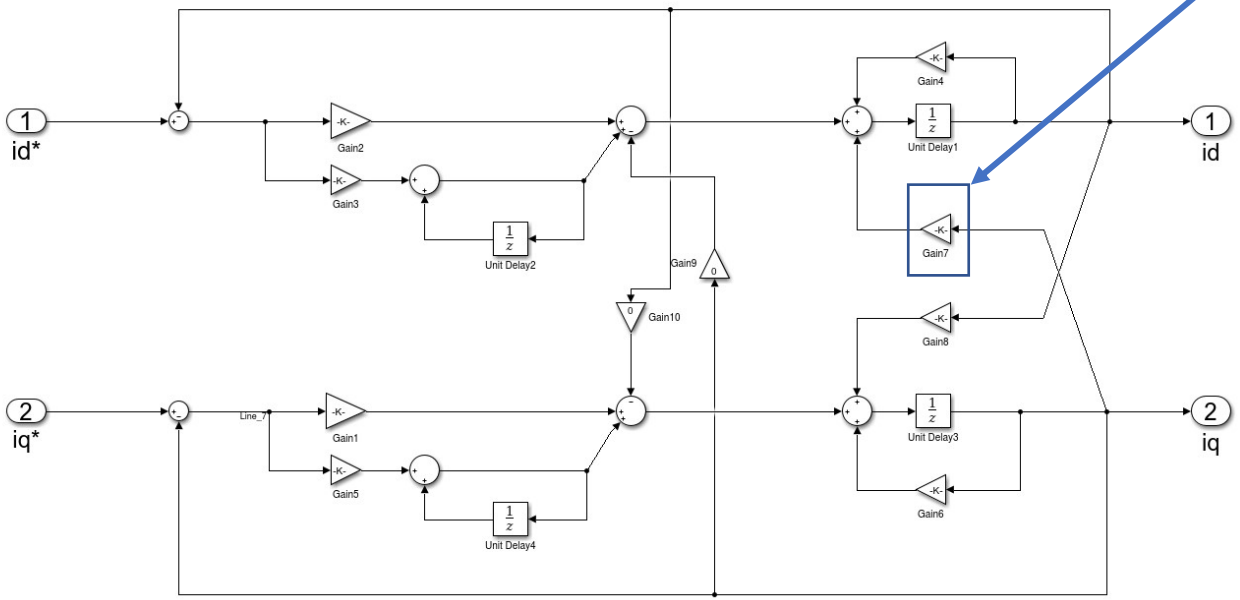
- モデルレベルで並列化
- モデルレベルでの並列性設計が重要
  - モデルレベルで並列性がなければコードレベルでの並列性抽出は困難





[3] 山口他. “Simulinkモデルからのブロックレベル並列化, ” 情報処理学会ESS2015（ポスター）.

SHIMの必要性（モデルベース並列化）



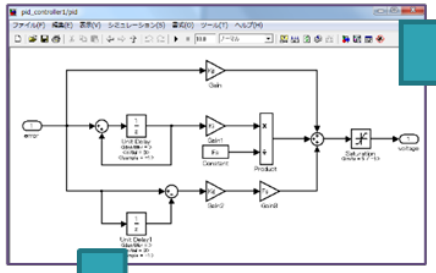
高い並列度を出すために<u>ブロック粒度(処理時間)</u>を考慮する必要がある

実機で計測すると高精度だが
・実機がないと計測出来ない
保有していても
・それぞれの設定工数が大きい
・使いこなすために知識と時間が必要

要望
・実機がない状態で測定したい
・ボードの特徴を最大限に利用したい
・様々なボードに対応してほしい

モデルレベルでどう性能評価するのか？

Simulinkモデル



②ブロックレベル
構造抽出
(CSPグラフ構造)

①コード
生成

```
/* Saturate: '<S1>/Saturation' */
if (pid_controller1_pid_Sum2_1 >= p
  pid_controller1_pid_Saturation_1
} else if (pid_controller1_pid_Sum2
{
  pid_controller1_pid_Saturation_1
} else {
  pid_controller1_pid_Saturation_1
}

/* End of Saturate: '<S1>/Saturatio

/* Sum: '<S1>/Sum' incorporates:
 *  Inport: '<Root>/error'
```

③ブロック
処理コード
抽出

```
<block blocktype="Saturate" name="pid_controller1_pid_
  <input line="pid_controller1_pid_Sum2_1" port="pid_
    <connect block="pid_controller1_pid_Sum2" port="p
  </input>
  <output line="pid_controller1_pid_Saturation_1" port
    <connect block="pid_controller1_pid_voltage" port
  </output>
  <var line="pid_controller1_pid_Sum2_1" mode="input"
  <var line="pid_controller1_pid_Saturation_1" mode="
  <param name="Saturation_UpperSat" storage="pid_contr
  <param name="Saturation_LowerSat" storage="pid_contr
  <code file="models/pid/pid_controller1_ert_rtw/pid_
    /* Saturate: '&lt;S1&gt;/Saturation' */
    if (pid_controller1_pid_Sum2_1 &gt;= pid_controlle
      pid_controller1_pid_Saturation_1 = pid_controll
    } else if (pid_controller1_pid_Sum2_1 &lt;= pid_co
    {
      pid_controller1_pid_Saturation_1 = pid_controlle
    } else {
      pid_controller1_pid_Saturation_1 = pid_controlle
    }
    /* End of Saturate: '&lt;S1&gt;/Saturation'
  </code>
  <code file="models/pid/pid_controller1_e
    pid_controller1_pid_Saturation_1 = 0
  </code>
  <performance best="26" type="task" typica
  <performance best="15" type="init" typical=
  <forward block="pid_controller1_voltage" type=
    <var line="pid_controller1_pid_1" mode="input"
  </forward>
  <backword block="pid_controller1_pid_Sum2" type="dat
    <var line="pid_controller1_pid_Sum2_1" mode="outpu
  </backword>
</block>
```

④抽出コード
処理量見積

```
<ComponentSet name="Board_B0">↓
  <ComponentSet name="Cluster_B0C0">↓
    <ComponentSet name="PE_B0C0P1">↓
      <SlaveComponent name="LRAM_B0C0P1"
      <MasterComponent name="CPU_B0C0P1"
        <CommonInstructionSet name="
          <Instruction name="ret">
            <Latency best="10.0"  ypical="10.
            <Pitch best="10.0" typical="10.0"
          </Instruction>↓
          <Instruction name="br">↓
```

SHIM

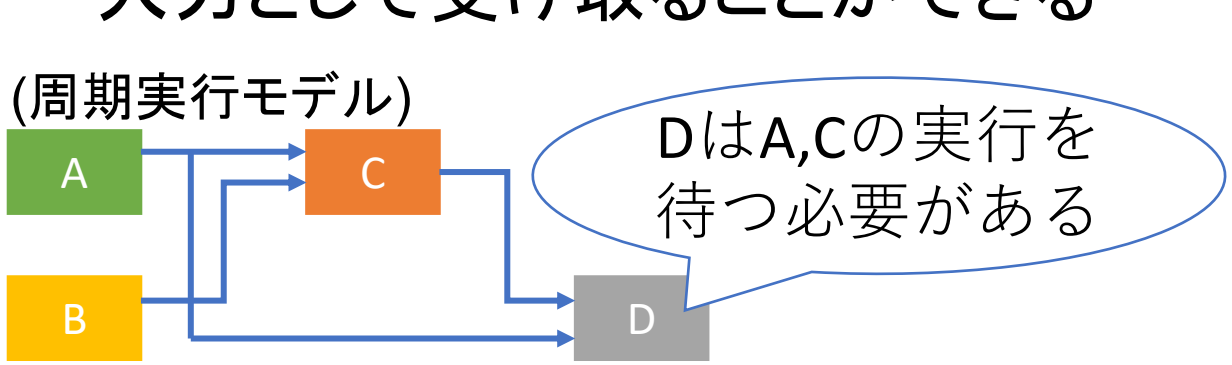
ブロックレベル
構造XML (BLXML)

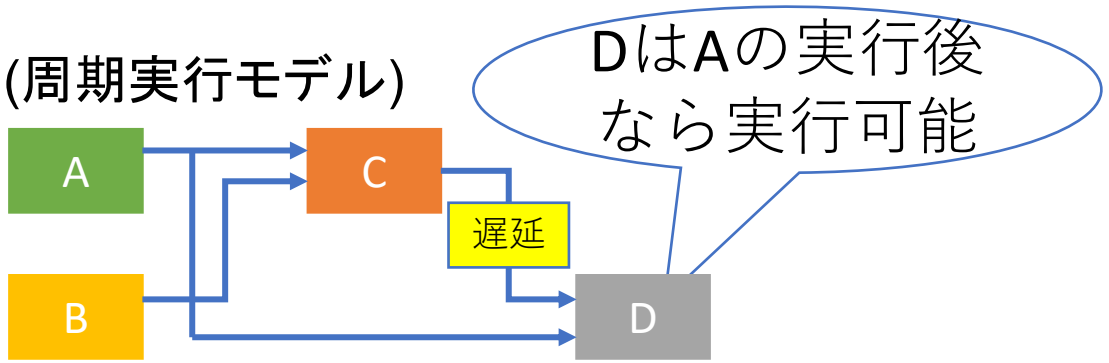
モデル（内の各ブロック）に対応するコードをLLVM中間表現に変換し、SHIMを用いて性能見積を行う

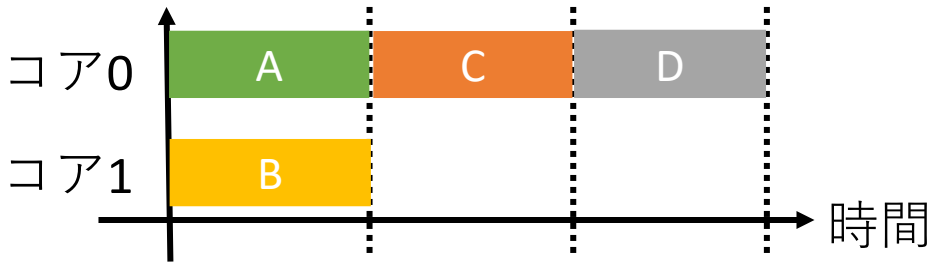
研究例2：UnitDelay追加によるモデルレベル性能改善 [4]

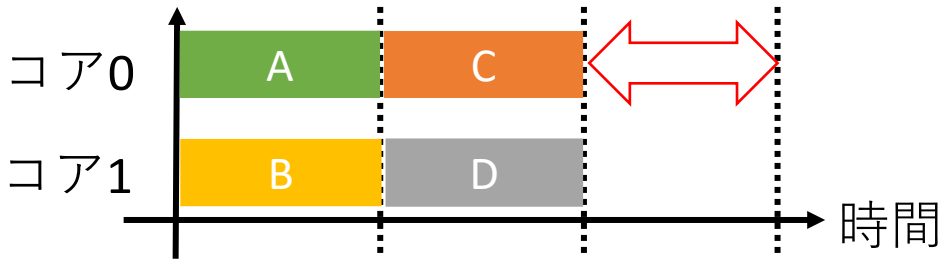
- UnitDelayブロック
  - 入力データを1周期遅らせて出力するブロック
    - ✓周期の始めに1周期前の入力データを出力
  - UnitDelayブロックの後続のブロックは周期の始めに（前周期の値を）入力として受け取ることができる

[4] 池田他. “制御モデルに内在する遅延を用いた並列化,” 情報処理学会ETNET2018.

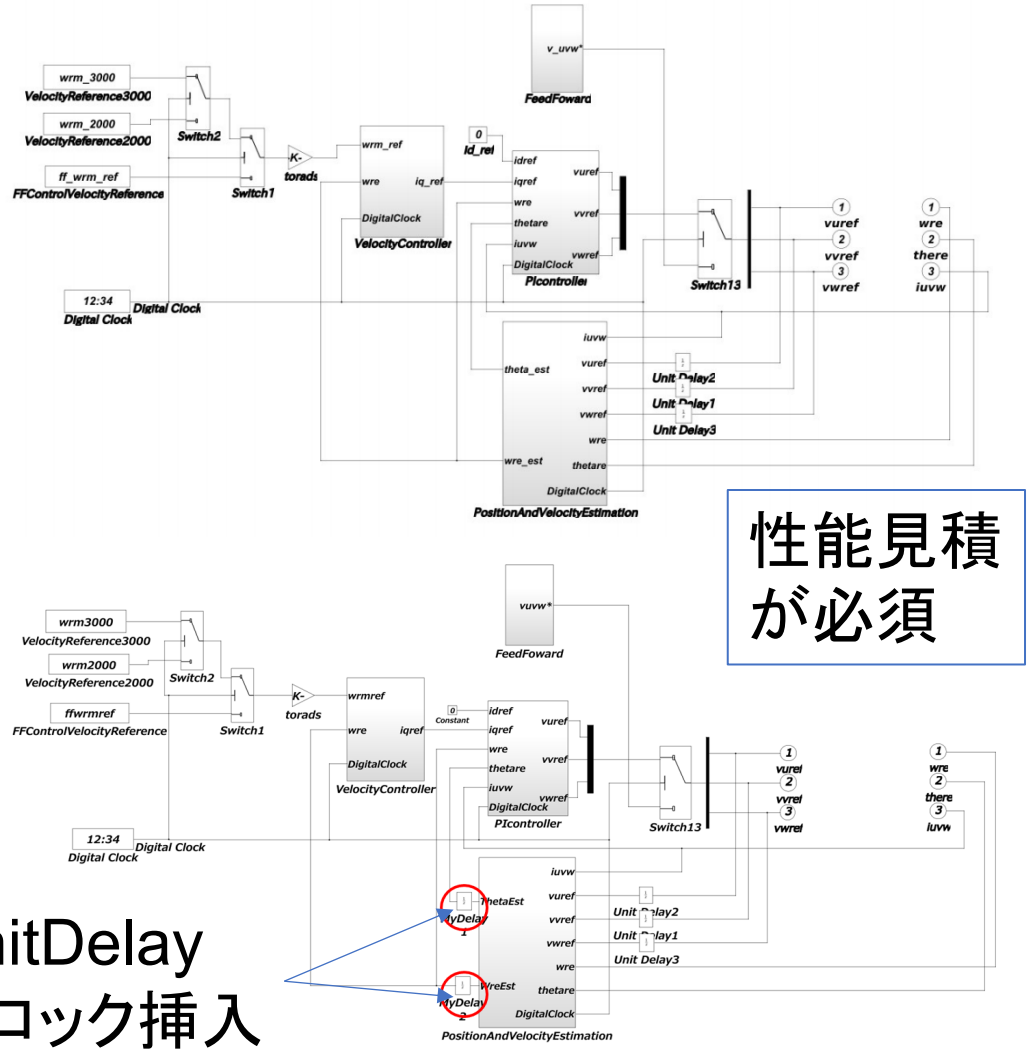






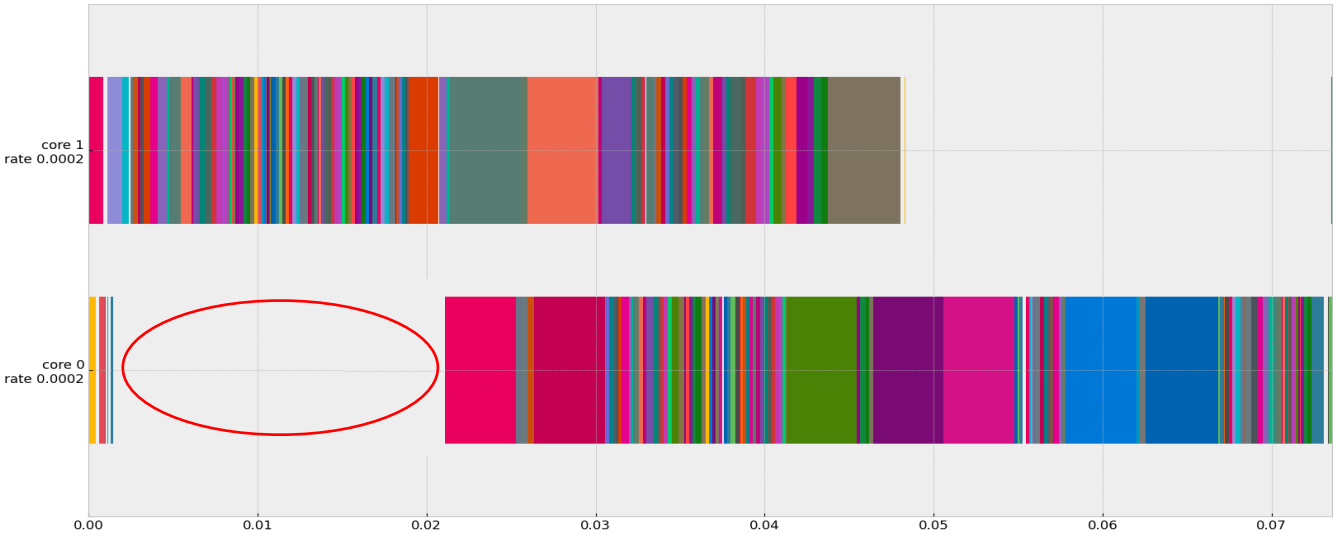


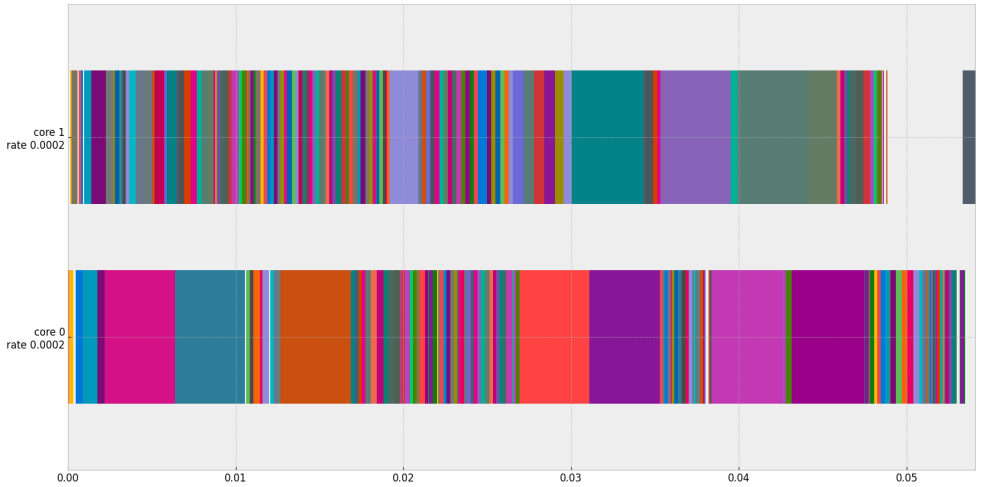
モータのベクトル制御モデル（457ブロック）



性能見積が必須

UnitDelay
ブロック挿入



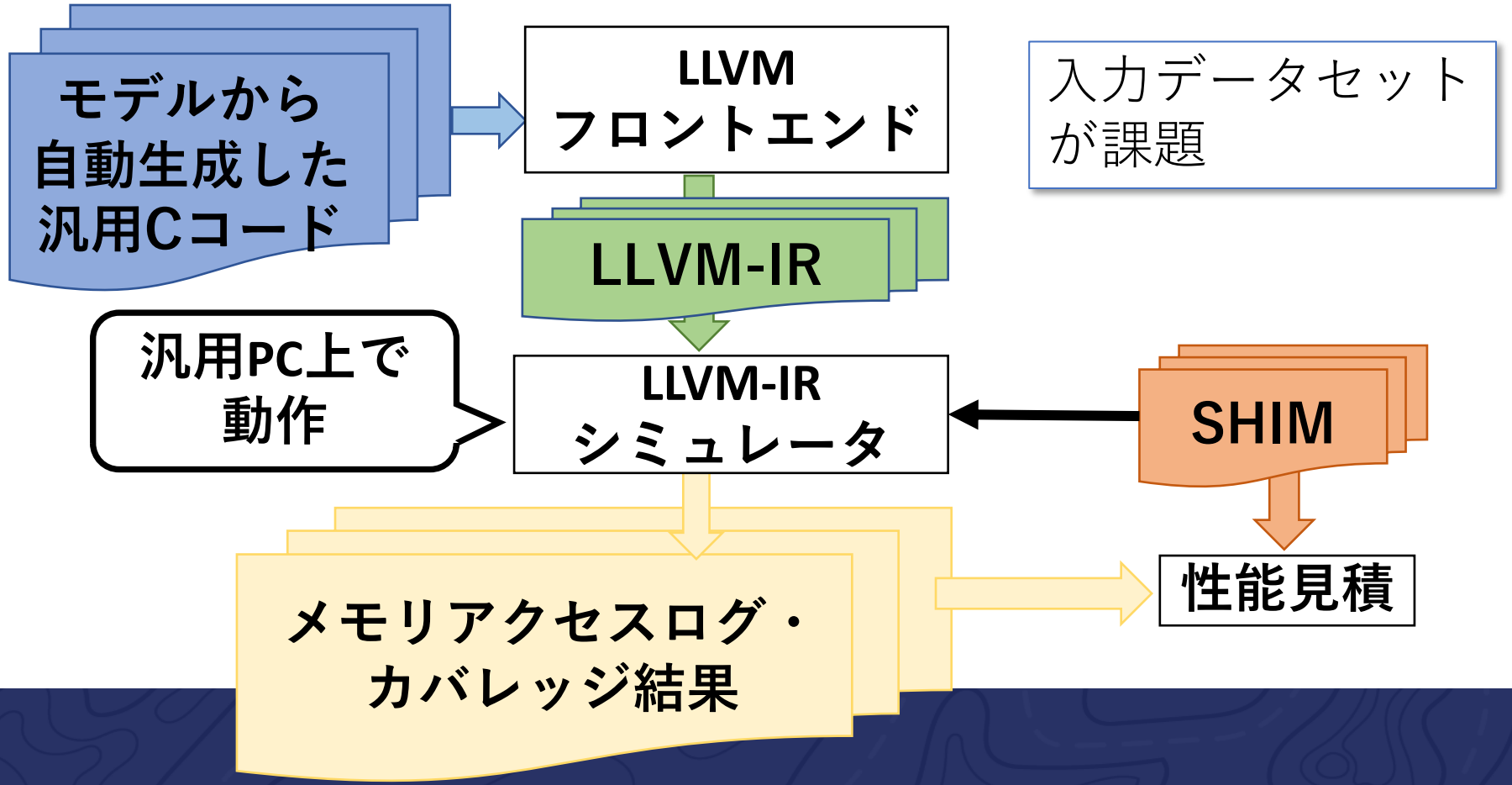


並列性能
向上
1.87倍

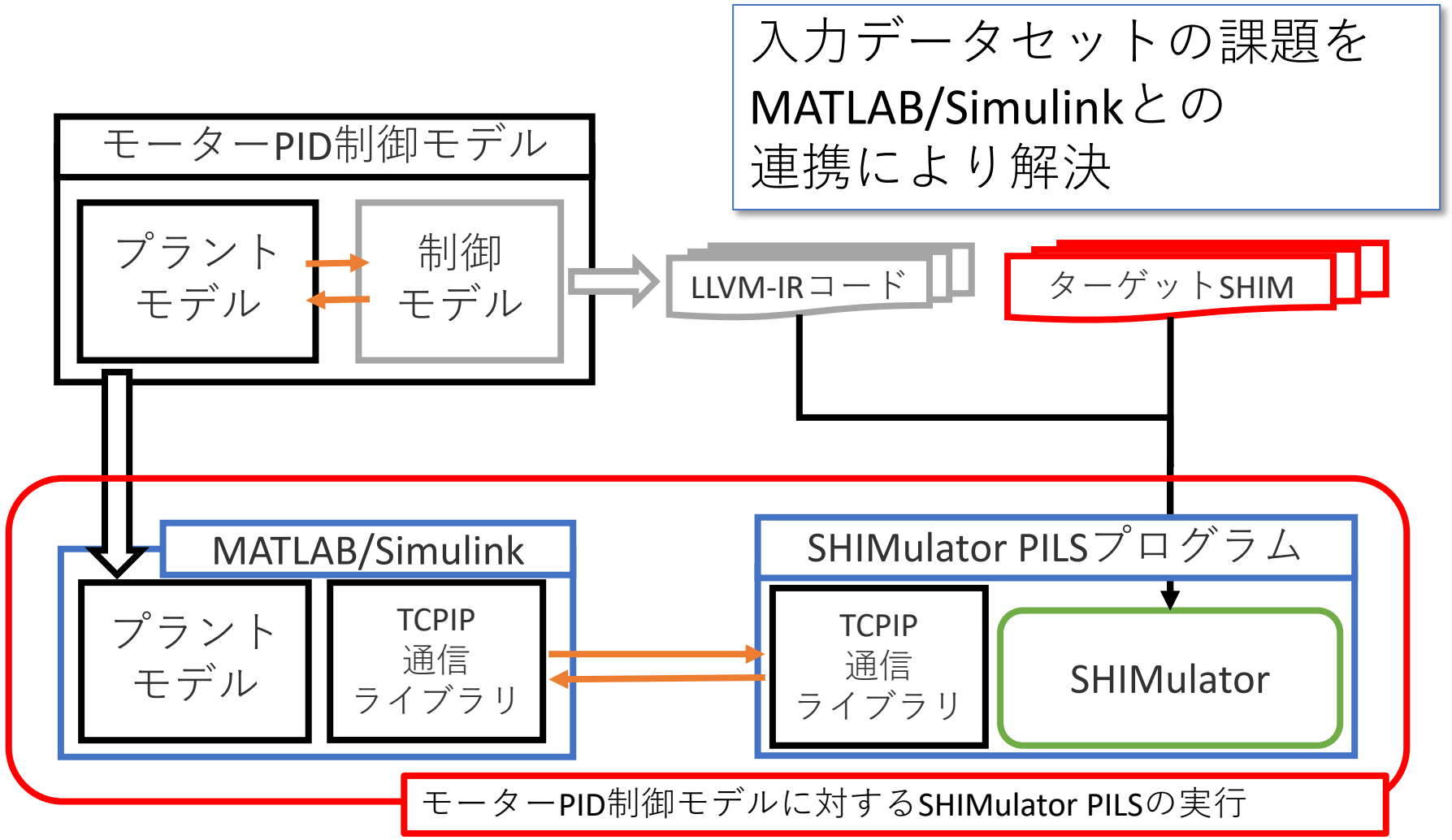
研究例3：SHIMulator [5]

- SHIMのアーキテクチャ情報、性能情報を使って動作するISS（命令レベルシミュレータ）
- 動的見積、静的見積の双方に利用可能

[5] 佐合他. “ハードウェア抽象化記述SHIMとSHIMulatorによるソフトウェア動的性能見積手法, ” 情報処理学会ETNET2019.



SHIMulator PILS



アウトライン

- SHIMについて
- SHIMの構造
- SHIMの利用・応用
- SHIMを用いた性能見積技術 [6]

[6] H. Mikami, et al., “LLVM Instruction Latency Measurement for Software-Hardware Interface for Multi/many-core,” IJCT, Vol.22 (2022), pp.50-63. (https://rajpub.com/index.php/ijct/article/view/9231)

LLVM-IR

- **LLVMで使用される中間表現**
  - 複数のプログラミング言語に対応
  - アーキテクチャに依存しない
  - 無限のレジスタを想定


```
%0 = load i32, i32* %a, align 4, !dbg !17
%1 = load i32, i32* %b, align 4, !dbg !17
%add = add nsw i32 %0, %1, !dbg !17
store i32 %add, i32* %c, align 4, !dbg !17
```

a と b を加算してcに代入するLLVM-IR

SHIMとLLVM-IRとの関係

SHIMは性能見積にLLVM-IR命令レイテンシを使う

- 多種の言語に対応しているので汎用性が高まる
- 性能見積精度は記述されたレイテンシの影響を受ける

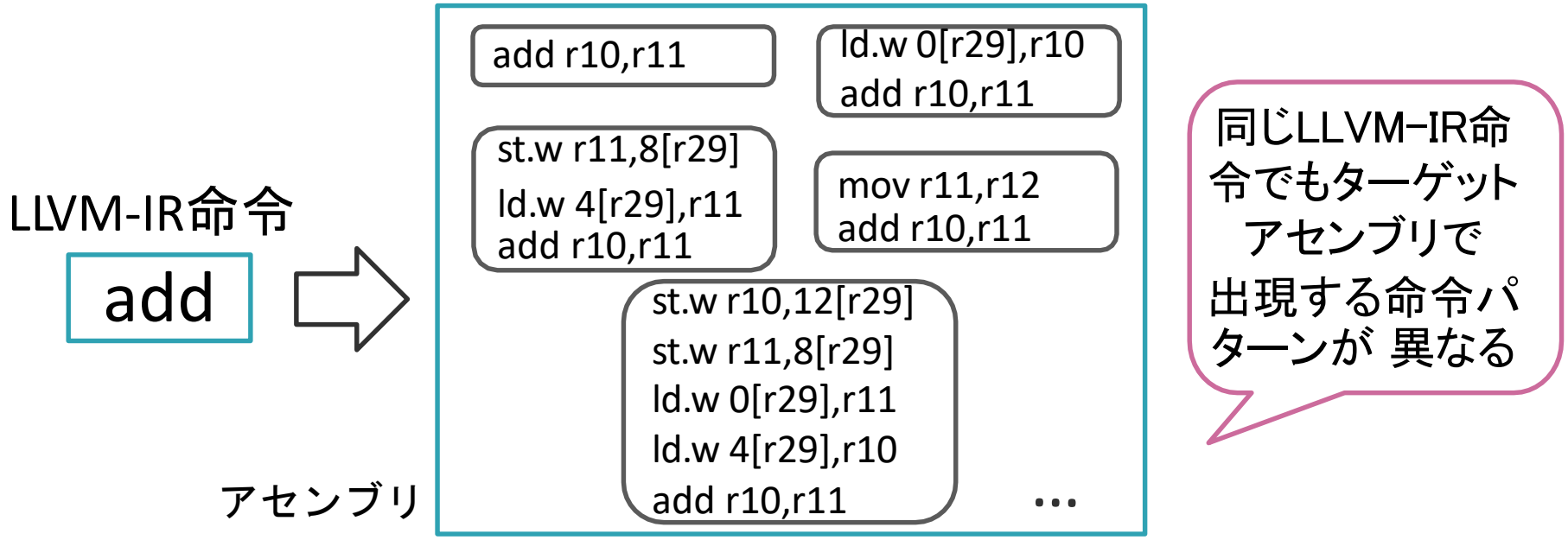
✓ LLVM-IRは無限のレジスタを想定している
　≫ レジスタスピルが起こることがない

✓ 単一のLLVM命令から複数のターゲットコードが生成される

SHIMによる性能見積手法、LLVM-IR命令性能値計測手法はどうあるべき？

- ➢ レジスタスピルや複数種類ターゲットコードを考慮した性能見積手法
- ➢ 多種のハードウェアに適用でき、ユーザも実施可能なLLVM-IR命令性能値計測

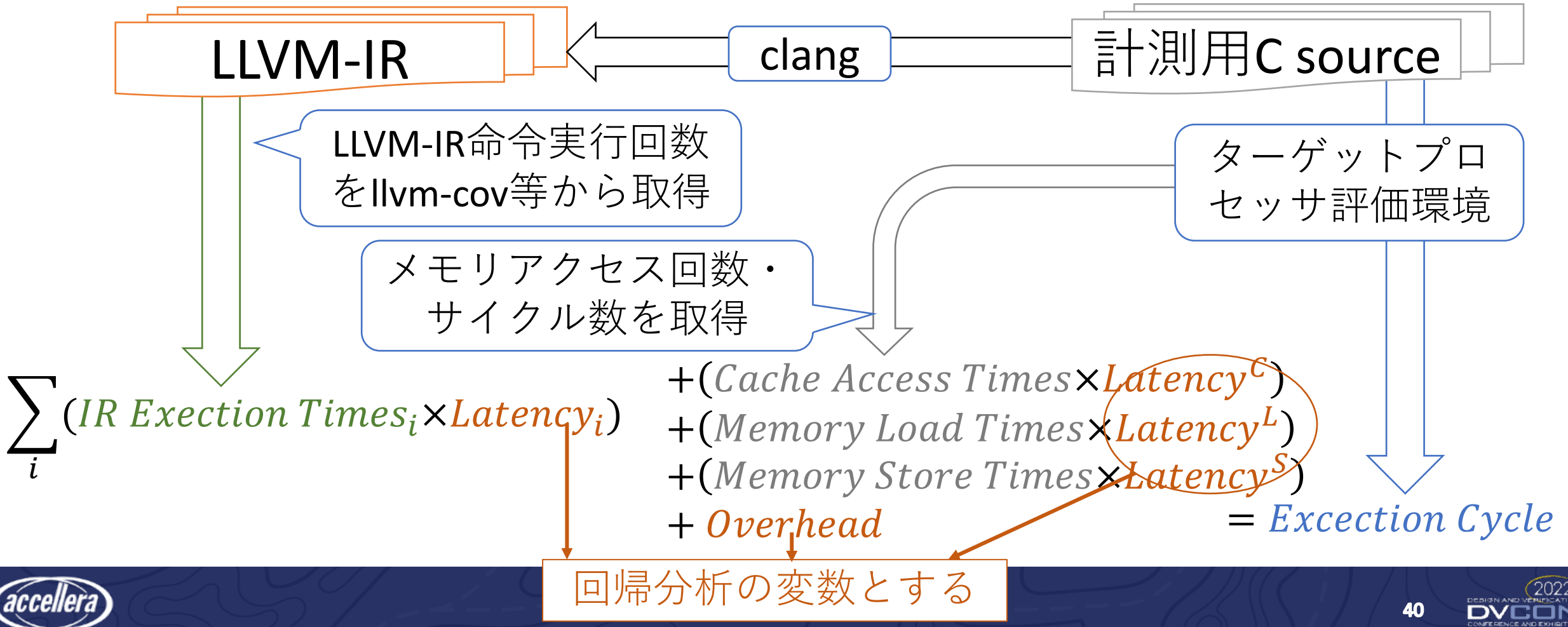
単一のLLVM-IR命令から複数種類のターゲットコードが生成される（しかもコンパイラ・最適化依存）



命令レイテンシの誤差だけではなく、各パターンの出現頻度の推定誤差も生じる

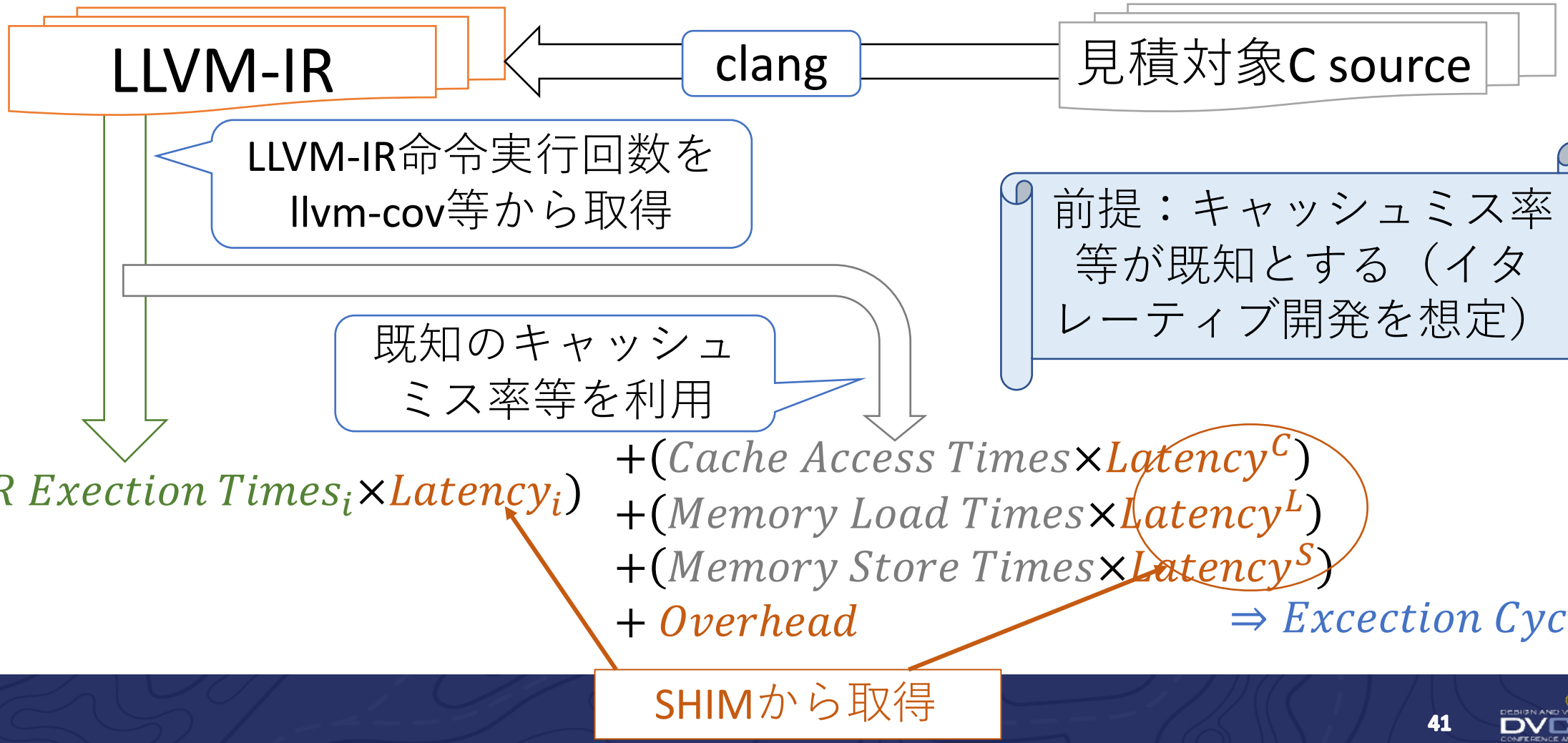
SHIMに入れる性能値計測の提案

- ターゲットプロセッサ評価環境を用い、回帰分析により性能値計測

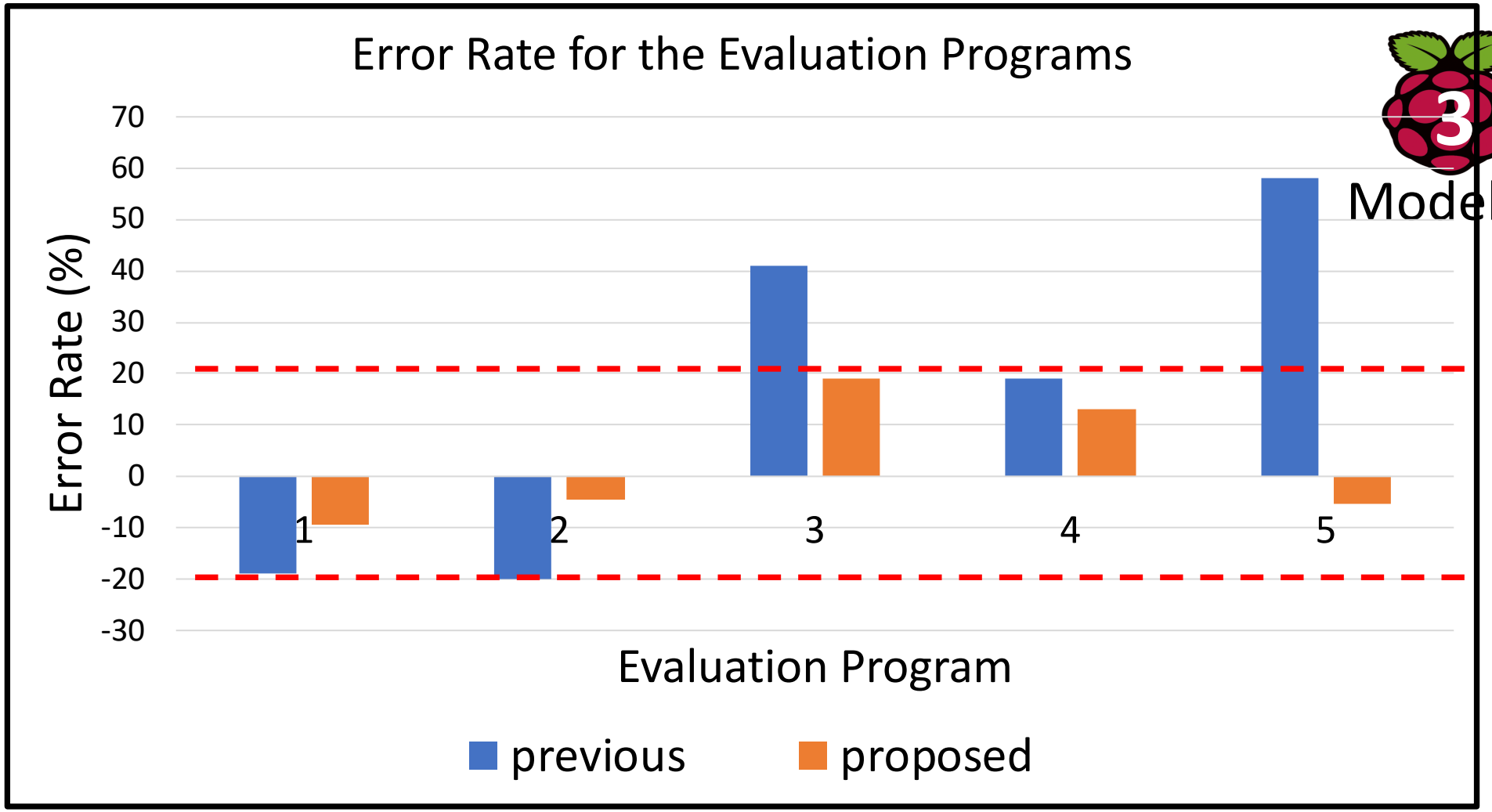


SHIMを用いたソフトウェア性能見積手法の提案

- ターゲットプロセッサ評価環境を用いず、上流設計で性能見積



Raspberry Pi 3 Model B+を用いた評価



Model B+

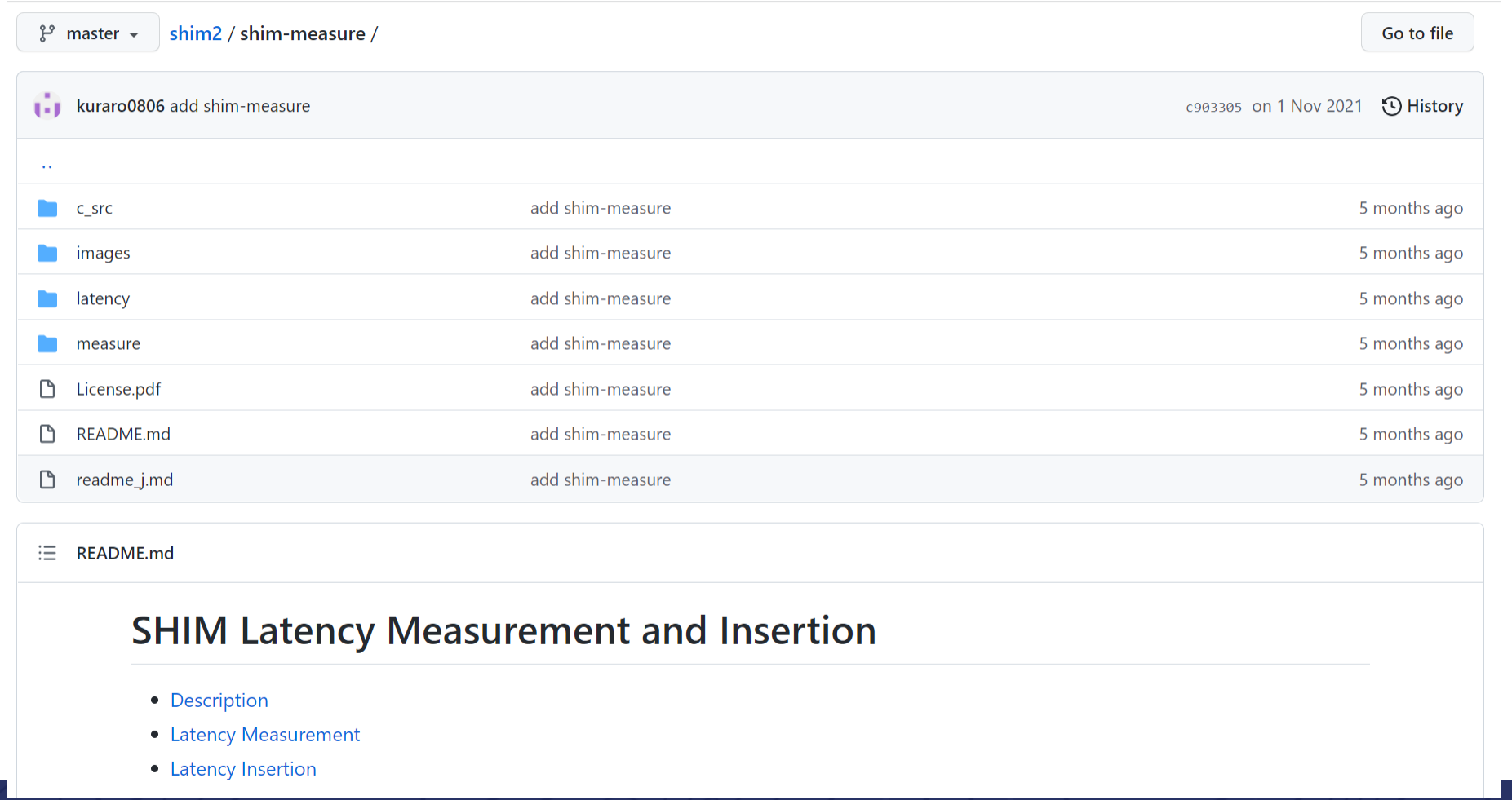
1.4GHz

Cortex-A53

42

SHIM性能値計測ツールを公開

- https://github.com/openshim/shim2/tree/master/shim-measure



43

まとめ

- SHIM (IEEE2804-2019)の概要、活用事例を紹介
  - Software-Hardware Interface for Multi-Many-Core
  - ソフトウェア視点で記述されたハードウェアモデル
  - XML記述により標準化することにより設計支援ツール等で利用可能
  - マルチコア・メニーコアプロセッサを有効活用するには、上流設計段階からハードウェアを意識した開発が必要だが、上流設計者はハードウェアの詳細を隠蔽したツールを使いたい
  - システムの設計生産性、競争力向上にはシステムベンダからチップベンダまでのエコシステム構築が必須⇒SHIMの役割は重要

- SHIM 3.0に向けて
  - より上流へ:プラットフォーム上での性能見積において、現状の通信記述が低レベルで、より上位のソフトウェアプリミティブに関する記述に期待

質疑応答

- 参考情報（SHIM関連URL）
  - SHIM2.0 (IEEE2804-2019)
    https://standards.ieee.org/ieee/2804/7477/
  - SHIM1.0
    https://www.embeddedmulticore.org/the-multicore-association-specifications/
  - OpenSHIM (SHIM Schema、関係ツール)
    https://github.com/openshim
- 謝辞
  - 日頃議論をいただいているIEEE SHIM WG Chair 権藤 正樹様（イーソル）、Secretary 安積 卓也先生（埼玉大）、組込みマルチコアコンソーシアムの皆様、共同研究先の皆様を始め関係各位に深く感謝します