モデルベース並列化と関連技術

２０２１年１月

名古屋大学大学院情報学研究科
枝廣　正人

“youtube masato edahiro”で検索していただくと、関連動画が見られます

本資料の一部画像は各製品販売企業WWWサイトから引用させていただきました。

講演概要

- モデルベース開発とマルチコア化が独立して普及する中、モデルレベルでターゲットを考慮して開発を進めることが必要不可欠になっています。本セミナーにおいては、名古屋大学で進めているモデルレベルでの並列化、その際にターゲット情報を取得するための国際標準SHIM、およびそれらの関連技術について紹介します。

目次

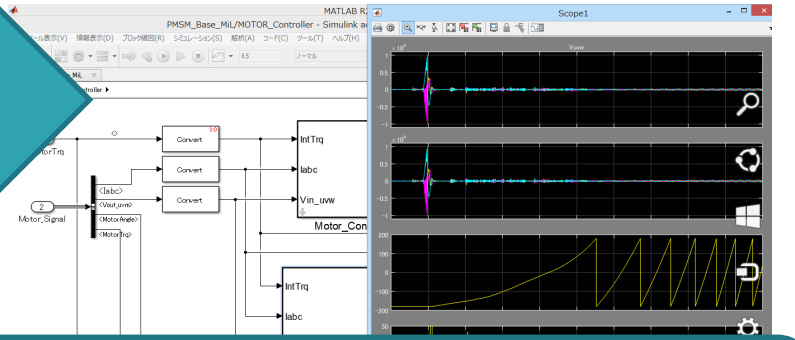
- どうしてモデルレベルでの並列化なのか
- 課題と研究状況
  - 並列化
  - 性能見積
  - 検証
- 組込みマルチコアコンソーシアムについて

モデルベース開発(MBD)

制御対象
（プラント）

MATLAB/Simulink

1．数理モデル化



2．制御器の数理モデル設計
（シミュレーションベース）

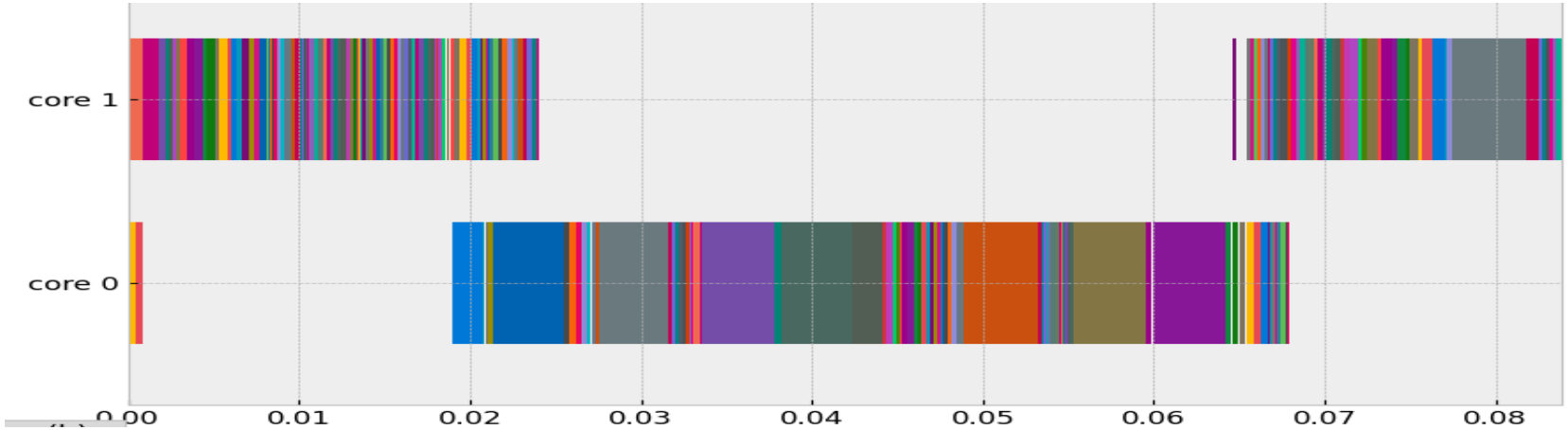
制御器（コントローラ）

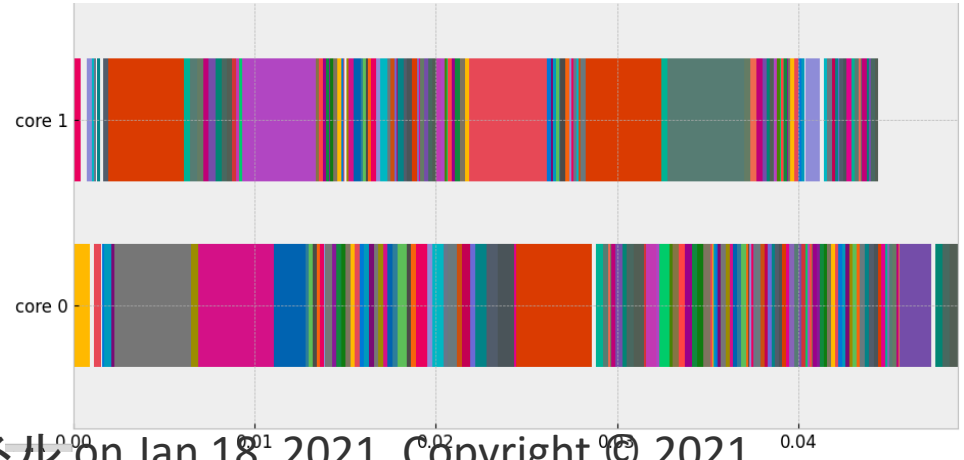
3．実装コード化

制御

制御と並列動作

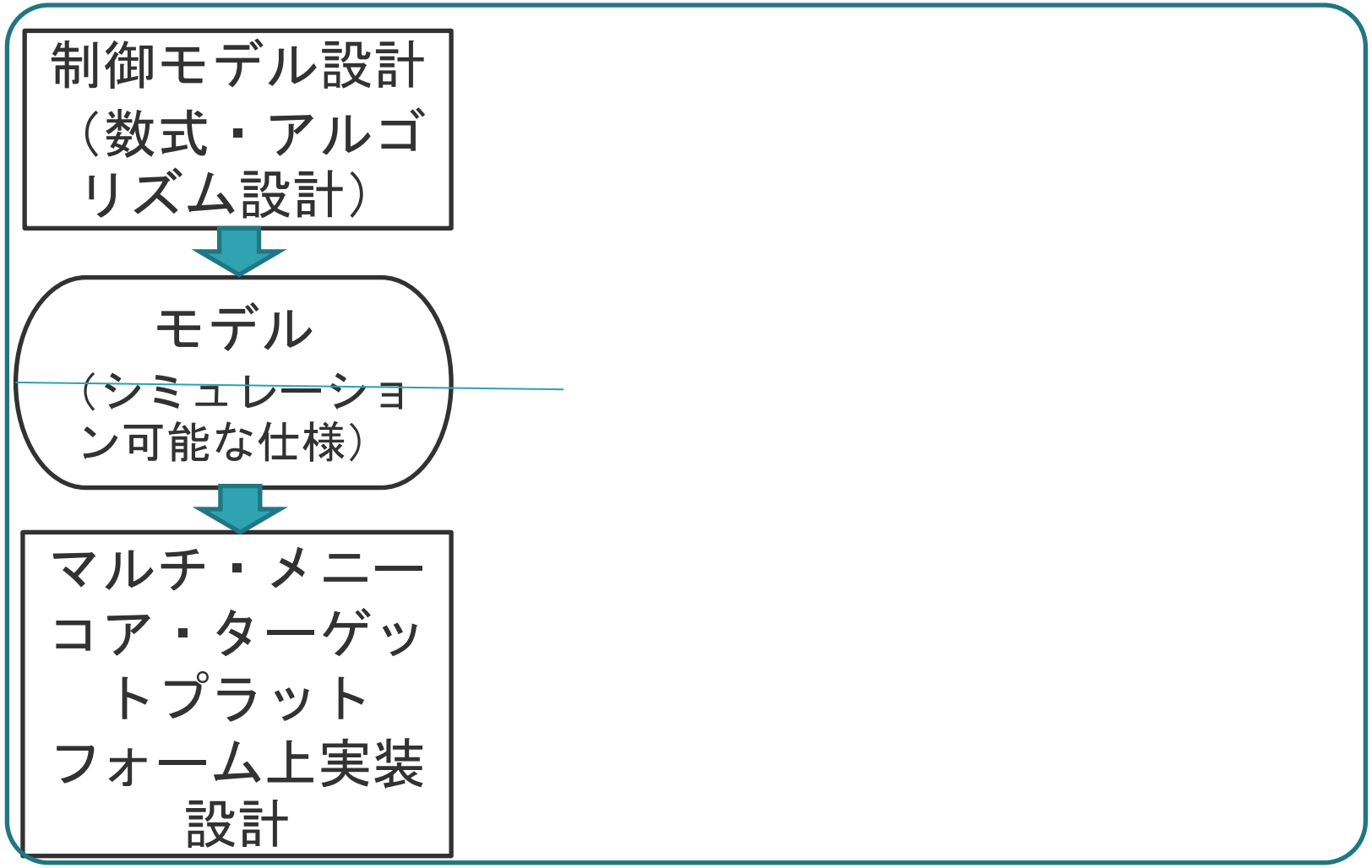
ほぼ同じ２種の制御プログラムの並列動作例



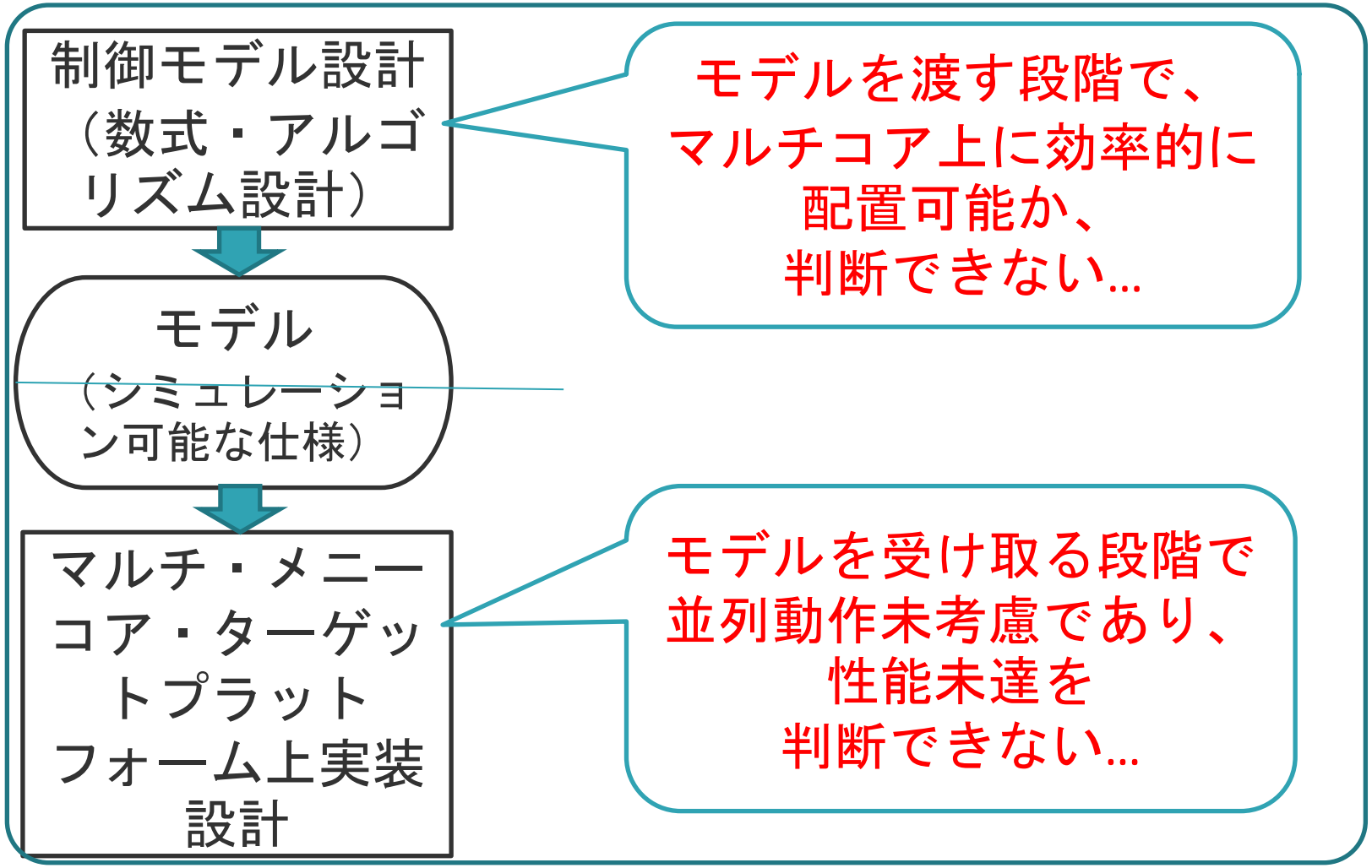


制御が少し異なるだけで、性能が大きく異なる

MBDの現状



モデルレベルで考えることができないと、、、



モデルレベルで考える際の課題

- 並列化手法
  - モデルレベルでどう並列化するのか？
- 性能見積
  - モデルレベルでどう性能評価するのか？
- 検証
  - モデルレベルでどう検証するのか？

目次

- どうしてモデルレベルでの並列化なのか

- 課題と研究状況
  - 並列化
  - 性能見積
  - 検証

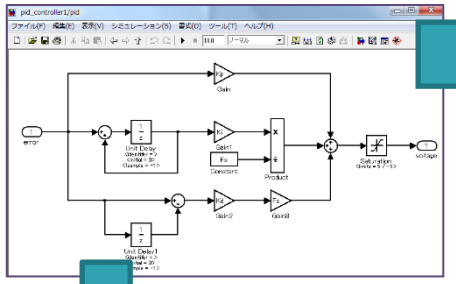
- 組込みマルチコアコンソーシアムについて

モデルレベルでどう並列化するのか？

- モデル（ブロック線図）のレベルで、ブロックをコアに割当
  - すべてのコアにおける負荷を同一にしつつコア間通信を最小化（いわゆるmin-cut手法）
- 重要：各ブロックに性能情報が付いていること
  - 性能見積の話題で
- 重要：すべての依存関係がブロック線図上の線として表現されていること
  - 実際には例外あり。SimulinkではData Store Memory
    →現状では依存関係をつけるか、同一Data Store Memoryに対するすべてのアクセスブロックを同じコアに配置して生成コードの順序を変えないことで対応
- 重要：並列動作時に逐次動作とふるまいを変えないこと
  - 検証の話題で

モデルからの情報抽出

Simulinkモデル



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'
```

②ブロックレベル構造抽出 (CSPグラフ構造)

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" por
    <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_cont
  <param name="Saturation_LowerSat" storage="pid_cont
  <code file="models/pid/pid_controller1_ert_rtw/pid_
    /* Saturate: '&lt;S1&gt;/Saturation' */
    if (pid_controller1_pid_Sum2_1 &gt;= pid_controll
      pid_controller1_pid_Saturation_1 = pid_controll
    } else if (pid_controller1_pid_Sum2_1 &lt;= pid_c
    {
      pid_controller1_pid_Saturation_1 = pid_controll
    } else {
      pid_controller1_pid_Saturation_1 = pid_controll
    }
    /* End of Saturate: '&lt;S1&gt;/Saturation' */
  </code>
  <code file="models/pid/pid_controller1_ert_rtw/pid_
    pid_controller1_pid_Saturation_1 = 0.0F;
  </code>
  <performance best="26" type="task" typical="31" wor
  <performance best="15" type="init" typical="15" wor
  <forward block="pid_controller1_voltage" type="port
    <var line="pid_controller1_pid_1" mode="input" na
  </forward>
  <backword block="pid_controller1_pid_Sum2" type="da
    <var line="pid_controller1_pid_Sum2_1" mode="outp
  </backword>
</block>
```

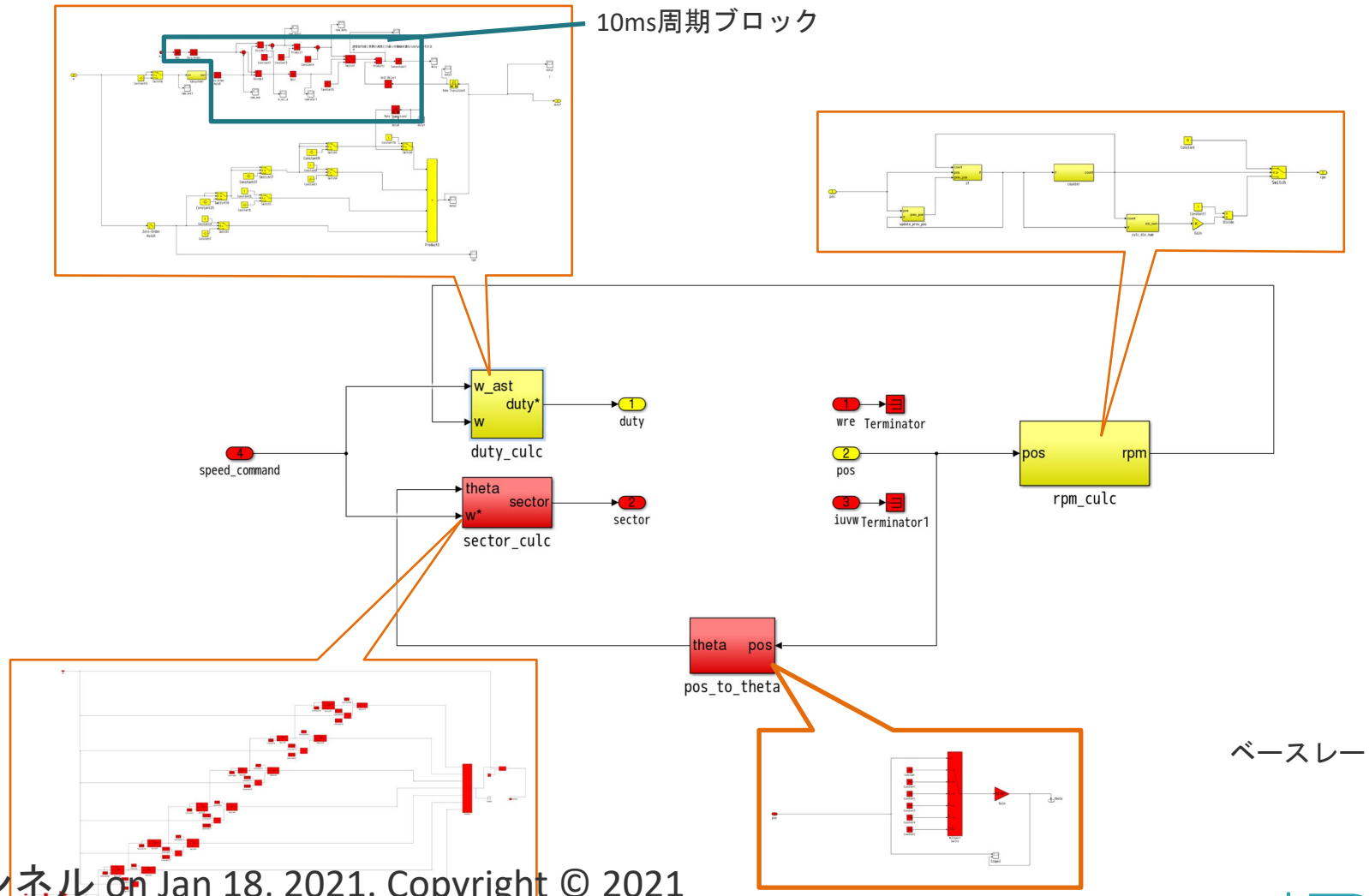
ブロックレベル構造XML (BLXML)

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" typical="10.
                        <Pitch best="10.0" typical="10.0"
                    </Instruction>↓
                    <Instruction name="br">↓
```

SHIM

２コア配置の例（色分けはコア割当）



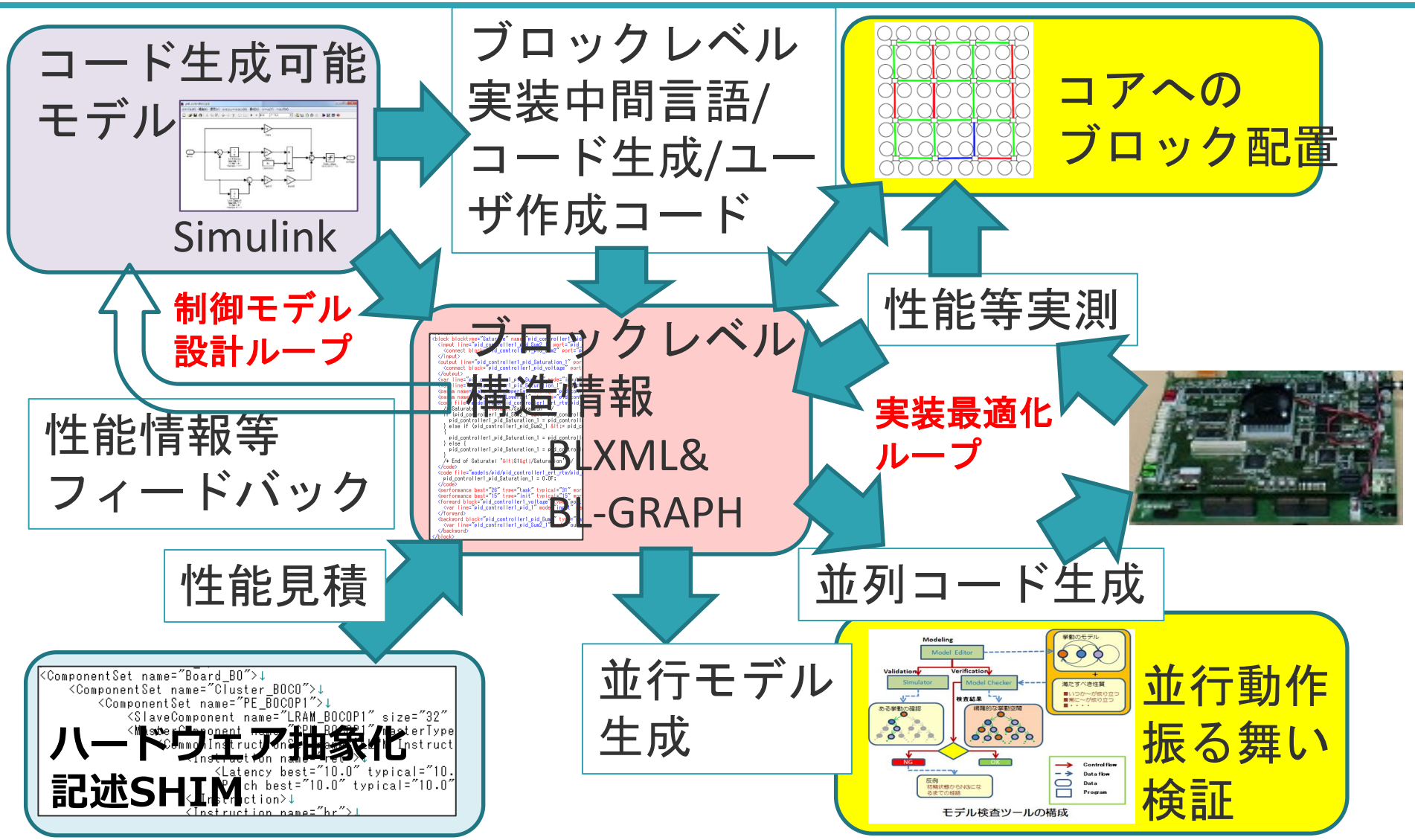
ベースレート：100us

自動コード生成

- 同一コアに割り当てられたブロックに対応するコードを順に並べ、必要に応じてコード間にコア間通信を配置

```
/* Block: pid_controller1_pid_Saturation */
agent sc_task_0010 () {
  interface {
    in<real32_T> CH_0013_0010;

    out<real32_T> CH_0010_I00002;

    spec {
      CH_0013_0010;
      CH_0010_I00002;
    };
  }

  map {
    SigmaC_agent_setUnitType(SigmaC_agent_self(),"k1-cluster");
  }

    /* params */
    struct {
      real32_T Saturation_UpperSat;
      real32_T Saturation_LowerSat;
    } pid_controller1_P = {
  5.0F,                                 /* Computed Parameter: S
                                         * Referenced by: '<S1>/
                                         */
  -5.0F                                 /* Computed Parameter: S
                                         * Referenced by: '<S1>/
                                         */
    };

    /* input variables */
    real32_T pid_controller1_pid_Sum2_1;

    /* output variables */
    real32_T pid_controller1_pid_Saturation_1;

  init{
  /* initialize task context */
  pid_controller1_pid_Saturation_1 = 0.0F;
  }

  void start ()
    exchange (CH_0013_0010 ch_0013_0010,
              CH_0010_I00002 ch_0010_I00002) {

    /* input */
    pid_controller1_pid_Sum2_1 = ch_0013_0010;

    /* C code */
  /* Saturate: '<S1>/Saturation' */
  if (pid_controller1_pid_Sum2_1 >= pid_controller1_P.Saturat
    pid_controller1_pid_Saturation_1 = pid_controller1_P.Satu
  } else if (pid_controller1_pid_Sum2_1 <= pid_controller1_P.
  {
    pid_controller1_pid_Saturation_1 = pid_controller1_P.Satu
  } else {
    pid_controller1_pid_Saturation_1 = pid_controller1_pid_Su
  }

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


    /* output */
    ch_0010_I00002 = pid_controller1_pid_Saturation_1;
  }
}
```

並列化全体像



事例

モデルベース開発からTOPPERS搭載システムへのクロスレイヤ自動設計を利用したマルチコアモータ制御実装

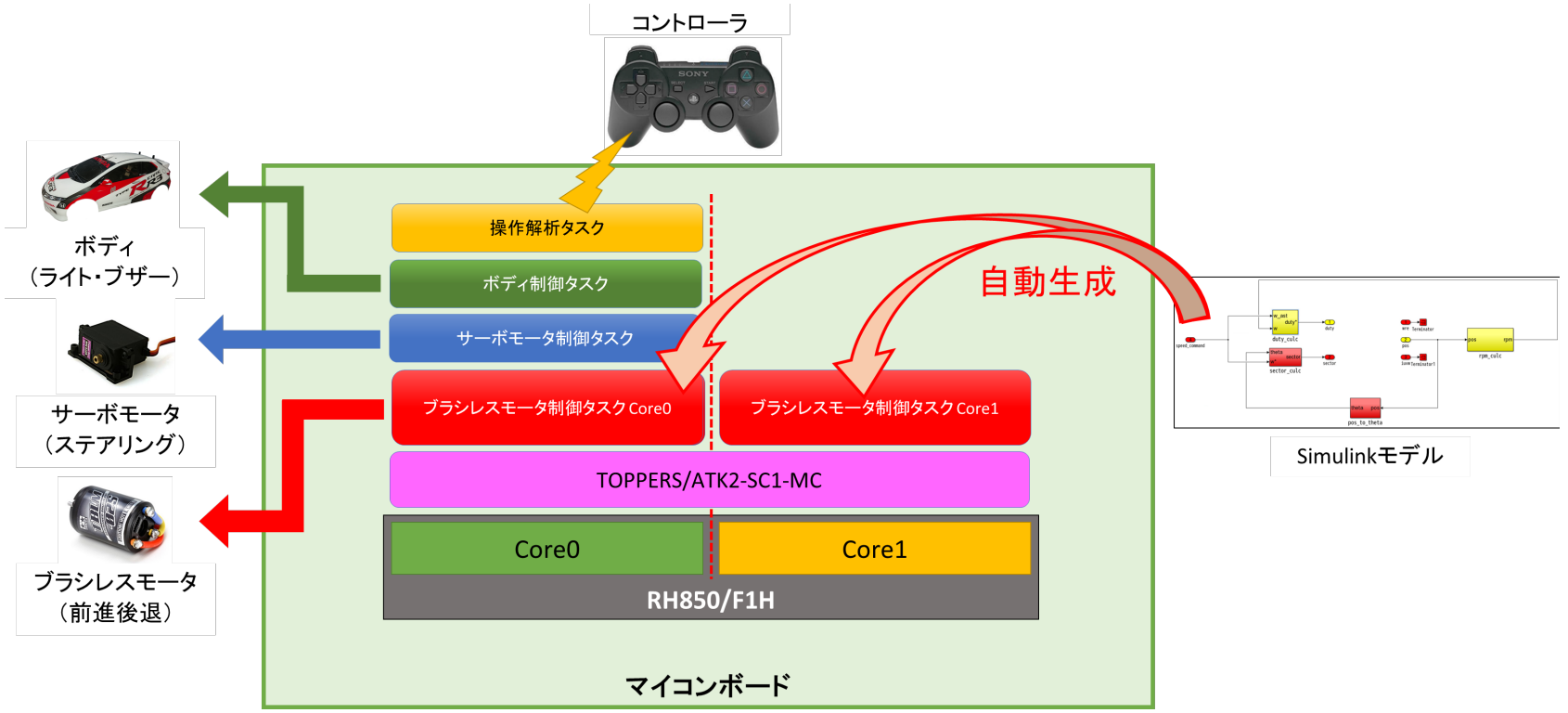
～第7回TOPPERS活用アイデア・アプリケーション開発コンテスト 銅賞

～ 組込みシステムシンポジウム2017 優秀ポスター賞

デモ動画: https://youtu.be/eszVE9wc__c

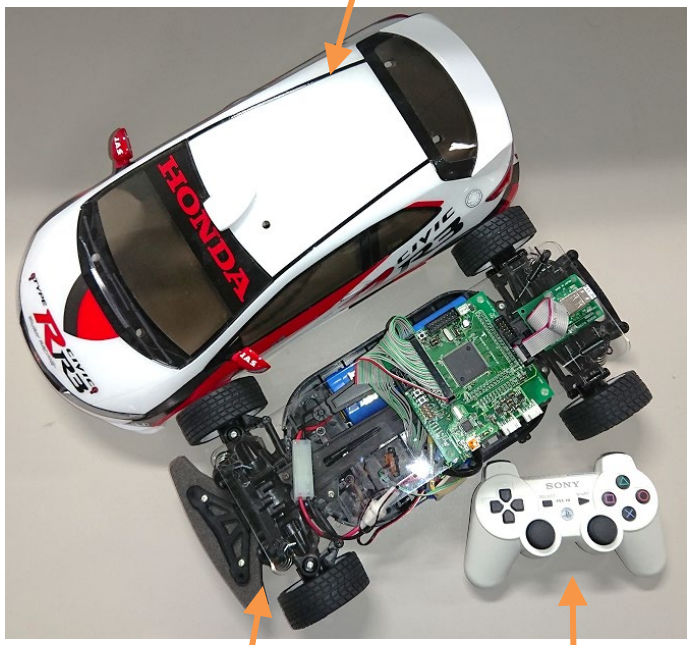
システム全体イメージ

- PS3コントローラで操作
- RH850/F1H(デュアルコア)を利用
- TOPPERS/ATK2-SC1-MC上で動作
- ブラシレスモータ制御はクロスレイヤ設計手法によってモデルから自動生成された並列タスクで実行



ハードウェア

ボディ
（ヘッドライト，ブレーキランプ，ブザー等
が搭載）

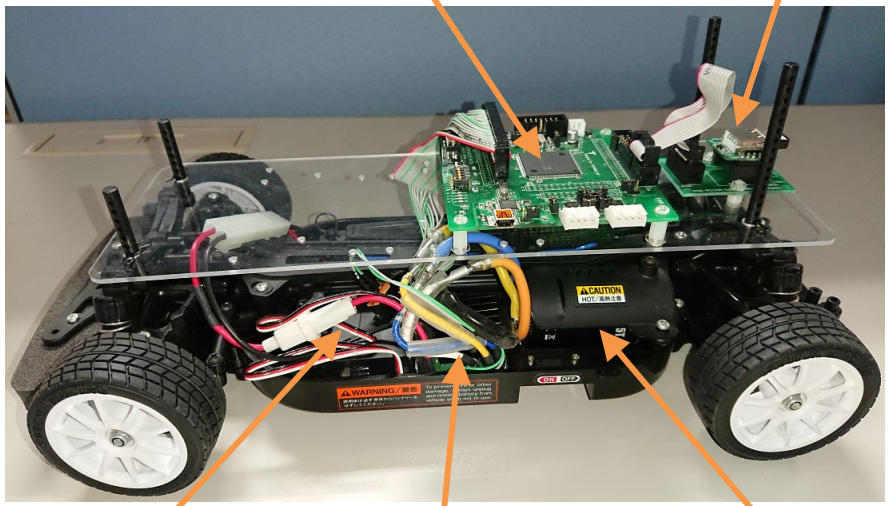


本体

PS3コントローラ

F1Hボード
(HSBRH850F1H176)

Bluetoothモジュール
(SBDBT5V)

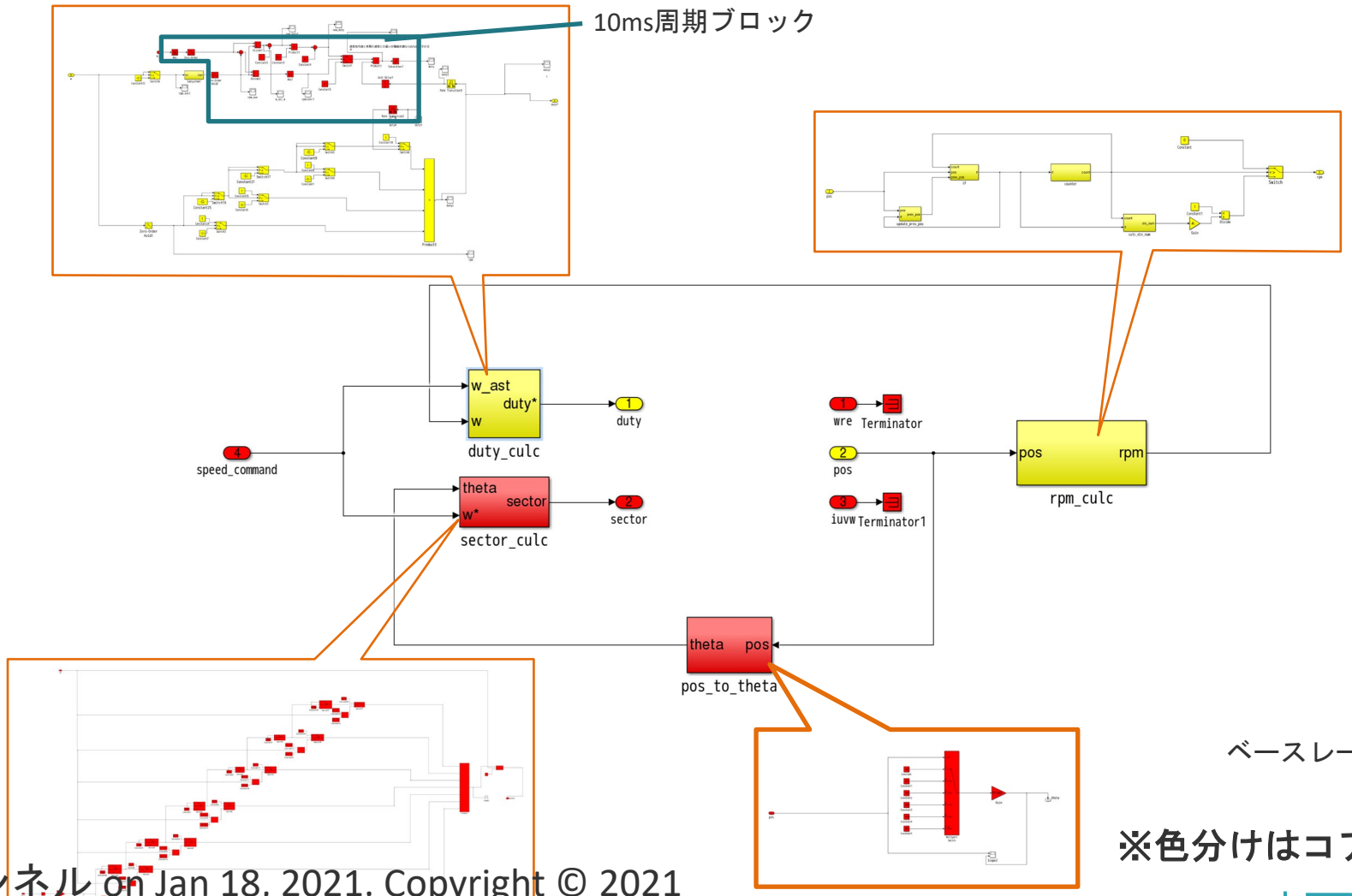


サーボモータ

組込用小型
モータドライバ
ボード

ブラシレスモータ

ブラシレスモータ制御モデル

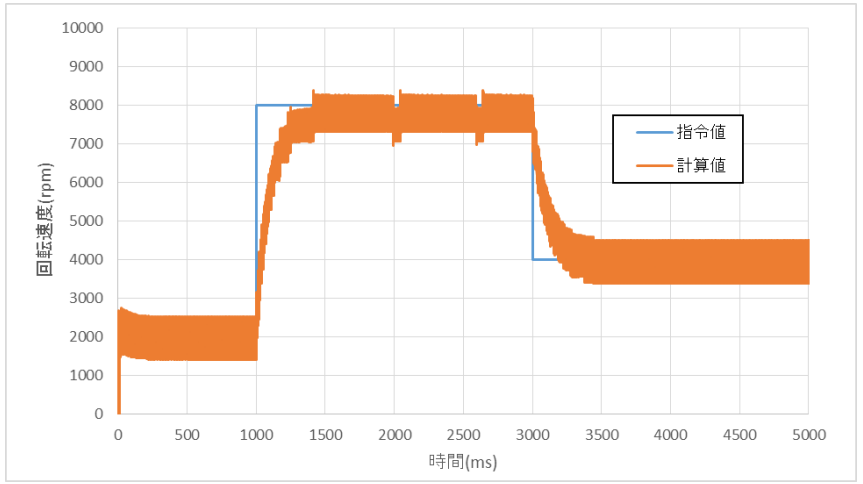


ベースレート：100us

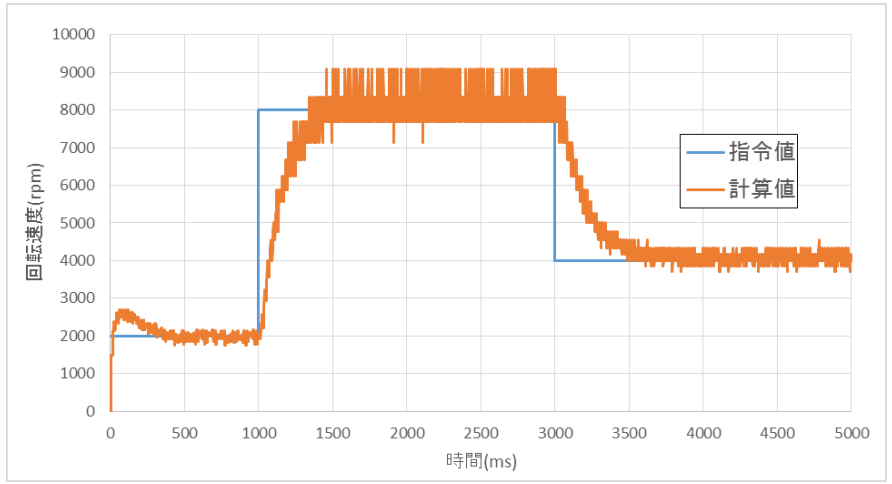
※色分けはコア割当を表す

並列コードの性能評価

- 出力比較



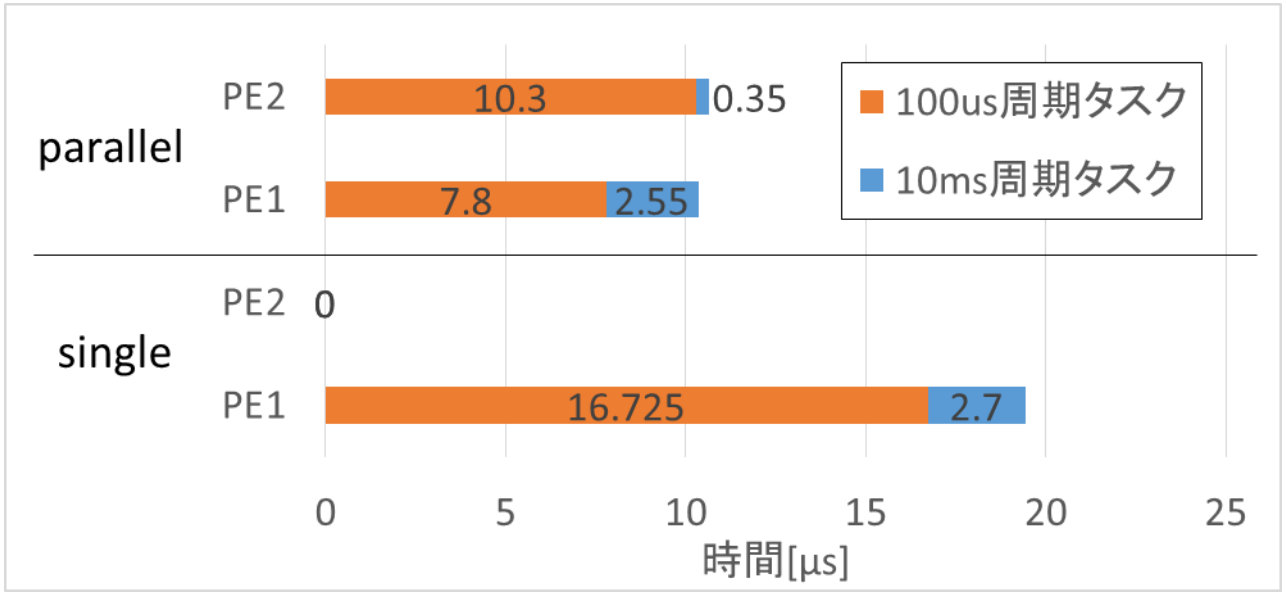
シミュレーション結果



実際の結果

並列性能比較

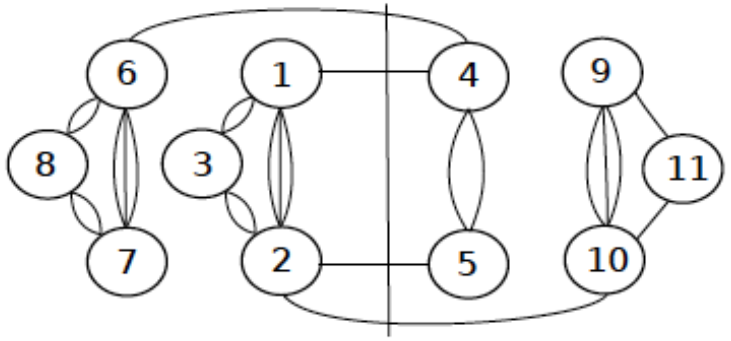
- 実行時間



- 100us周期タスク　1.62倍
- 100us周期タスク + 10ms周期タスク　1.82倍

Min-Cut法

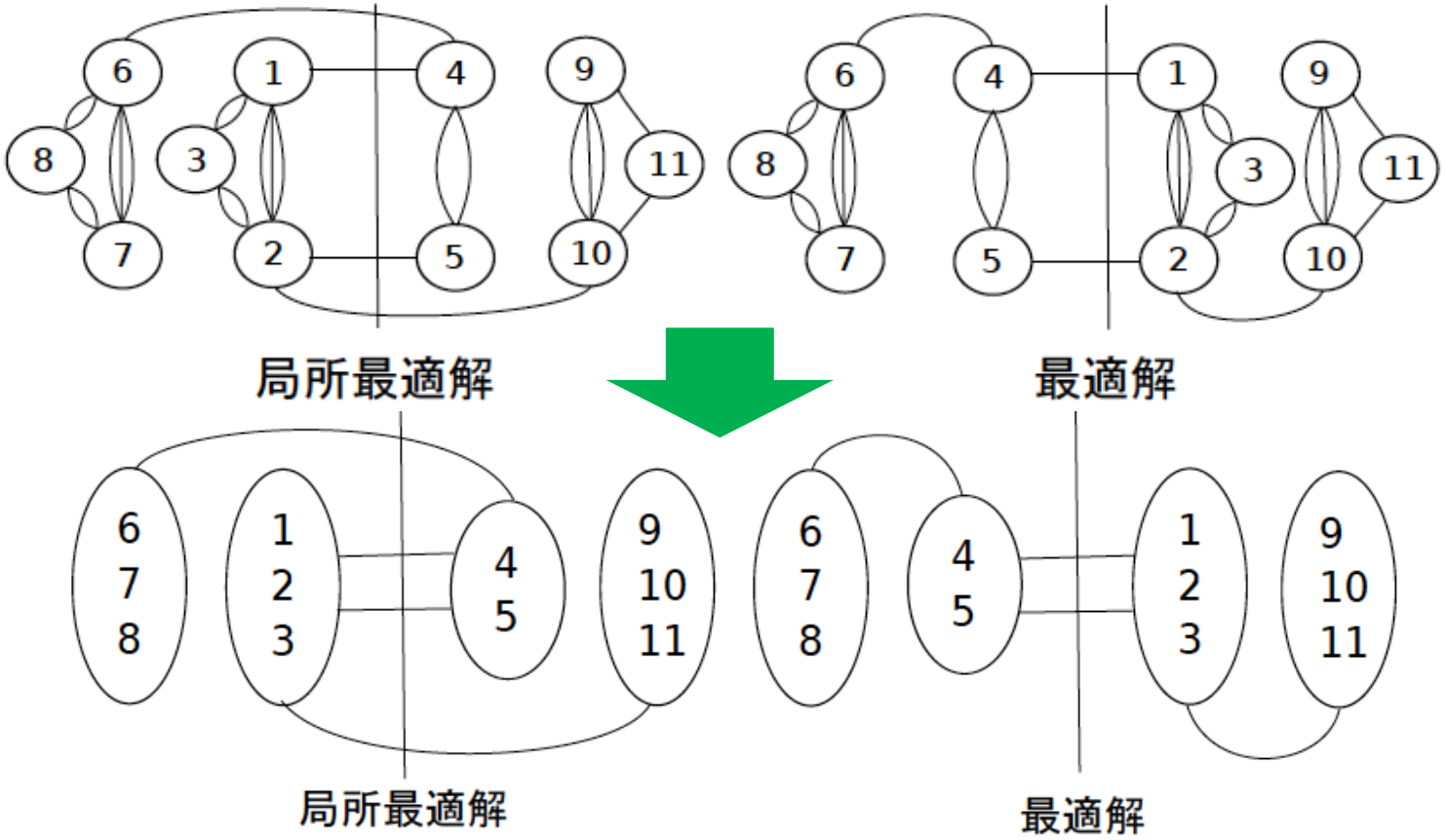
- 負荷を同一にしてコア間通信を最小化
  - 典型的には２段階
    - 初期解・・・問題に応じて良さそうな解を高速に生成
    - 反復・・・両側から選び、解が良くなる方向で交換
  - 局所最適解に陥りやすく、脱出のための工夫がポイント
    - 局所最適解・・・最適解に到達していないのに、どれを交換しても解は良くならない
  - 脱出アルゴリズム
    - Kernighan-Lin法
    - メタヒューリスティック
      - Simulated Annealingなど
    - 階層クラスタリング
    - など数多く提案されている



局所最適解

階層クラスタリング手法(*)

- 階層的にクラスタリングし、崩しながらペア交換アルゴリズムを適用すると局所最適解から脱出しやすい



(*) M. Edahiro and T. Yoshimura: New Placement and Global Routing Algorithms for Standard Cell Layouts. DAC 1990: pp. 642-645

二重階層クラスタリング法

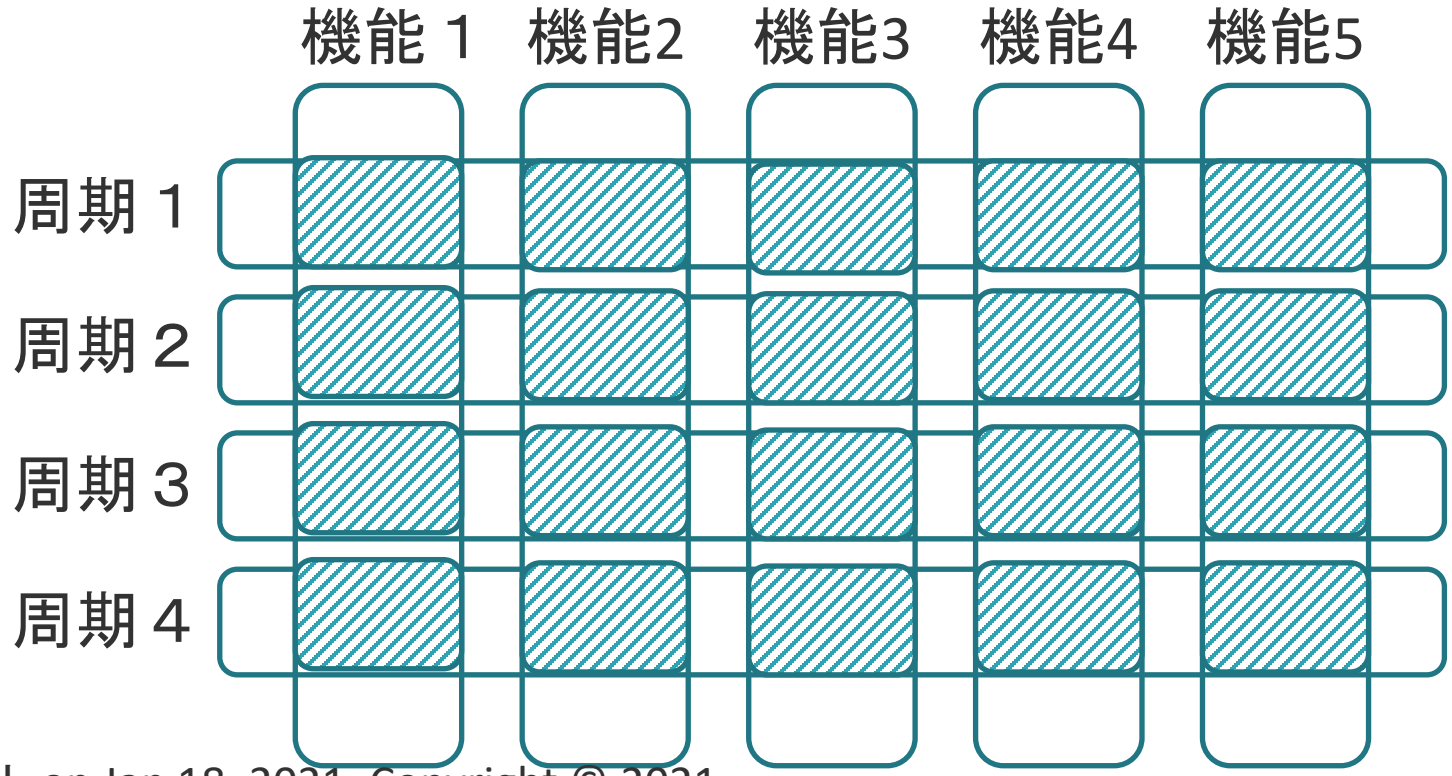
- 設計制約等により同じプロセッサに配置したいグループなどの考慮を入れる
- 二重階層クラスタリング法（全体）
  1. 第ゼロ段階のクラスタリング（ユーザ強制配置）
  2. 第一段階のクラスタリング
  3. 第二段階のクラスタリング
  4. 第二段階クラスタのコア配置（グローバル配置）
  5. 第一段階クラスタのコア配置（ローカル配置）
     - ローカル配置に階層クラスタリング法を適用
  6. 第一段階および第ゼロ段階クラスタリングを展開し、並列化完了

第一段階のクラスタリング

- メモリアクセスクラスタリング
  - 同じ共有メモリ資源に読み書きする処理ブロックをグループ化
  - 検証に有利
- 一括コードブロッククラスタリング
  - 複数ブロックに対し、一括してコード生成されるブロック群のグループ化
- サブシステム関連クラスタリング
  - 特殊ポート付や、標準ライブラリなど分割しない方が良いサブシステムのグループ化

第二段階のクラスタリング

- 例えば（機能 (Atomic Subsystem)×周期）でグループ化
- グローバル配置
  - 個々の第二段階クラスタをコア（群）へ割当
  - クラスタ数が多くないため様々な手法が可能



グローバル配置アルゴリズム

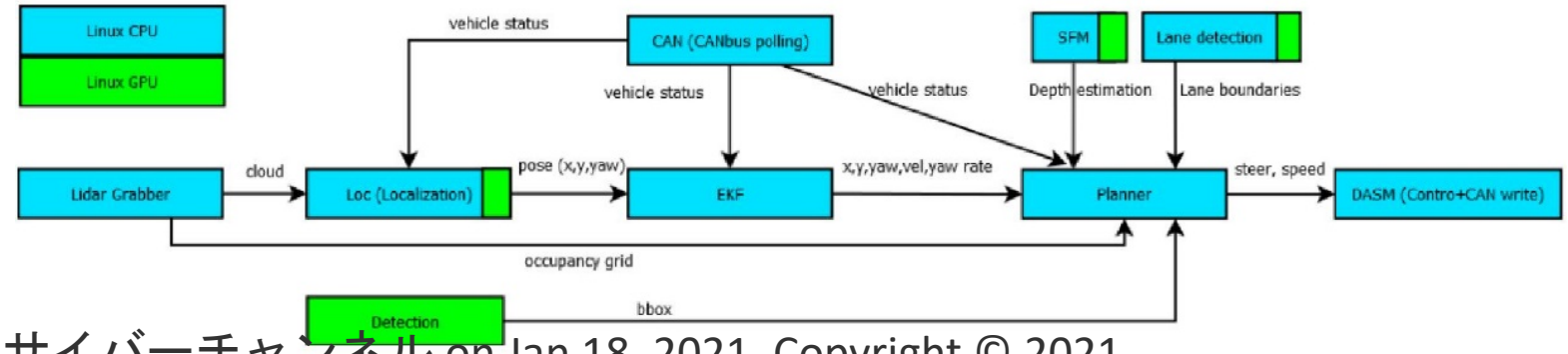
- 混合整数線形計画を利用
  - 鍾，枝廣. "組込み制御システムに対するマルチコア向けモデルレベル自動並列化手法", 情報処理学会論文誌, Vol.59, pp. 735-747, 2018.
  - 性能ヘテロジニアス対応
    - コアに性能倍率属性を対応させる
    - Z. Zhong and M. Edahiro. "Model-Based Parallelizer for Embedded Control Systems on Single-ISA Heterogeneous Multicore Processors," International Journal of Computer & Technology, Vol. 19, pp. 7470-7484, 2019.
  - 一般のヘテロジニアス対応
    - コアにコア種類属性を対応させ、ブロックにコア種類ごとの性能情報を対応させる（コア種類: CPU, GPU, etc.)
    - 一般には非線形問題になるが、パスに注目することにより、混合整数線形計画問題に帰着
    - Z. Zhong and M. Edahiro. “Model-Based Parallelization for Simulink Models on Multicore CPUs and GPUs," International Journal of Computer & Technology, Vol. 20, pp. 1-13, 2020.

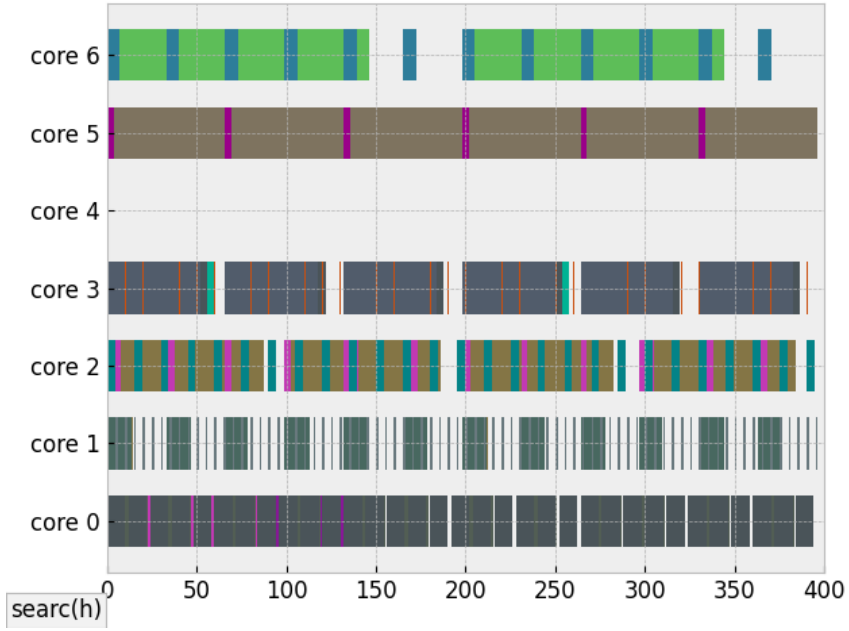
並列化に関するその他の話題

入力拡張：AMALTHEAモデル

- AMALTHEAモデルからBLXMLへの変換
- AMALTHEAモデル
  - 欧州車載向けマルチコアシステム記述のためのモデル
  - ヘテロジニアスシステムも記述可能
- WATERS2019（欧州組込み関連学会）提供の車載制御モデルを用いて実験
  - 自動運転を想定したモデル
  - NVIDIA社 Jetson TX2 (4CPU+2CPU+1GPU)を想定し、各ブロック（下図の四角）に、２種のCPUおよびGPUでの実行性能が与えられている
  - 6コアにて周期制約を満たす割り当てを達成

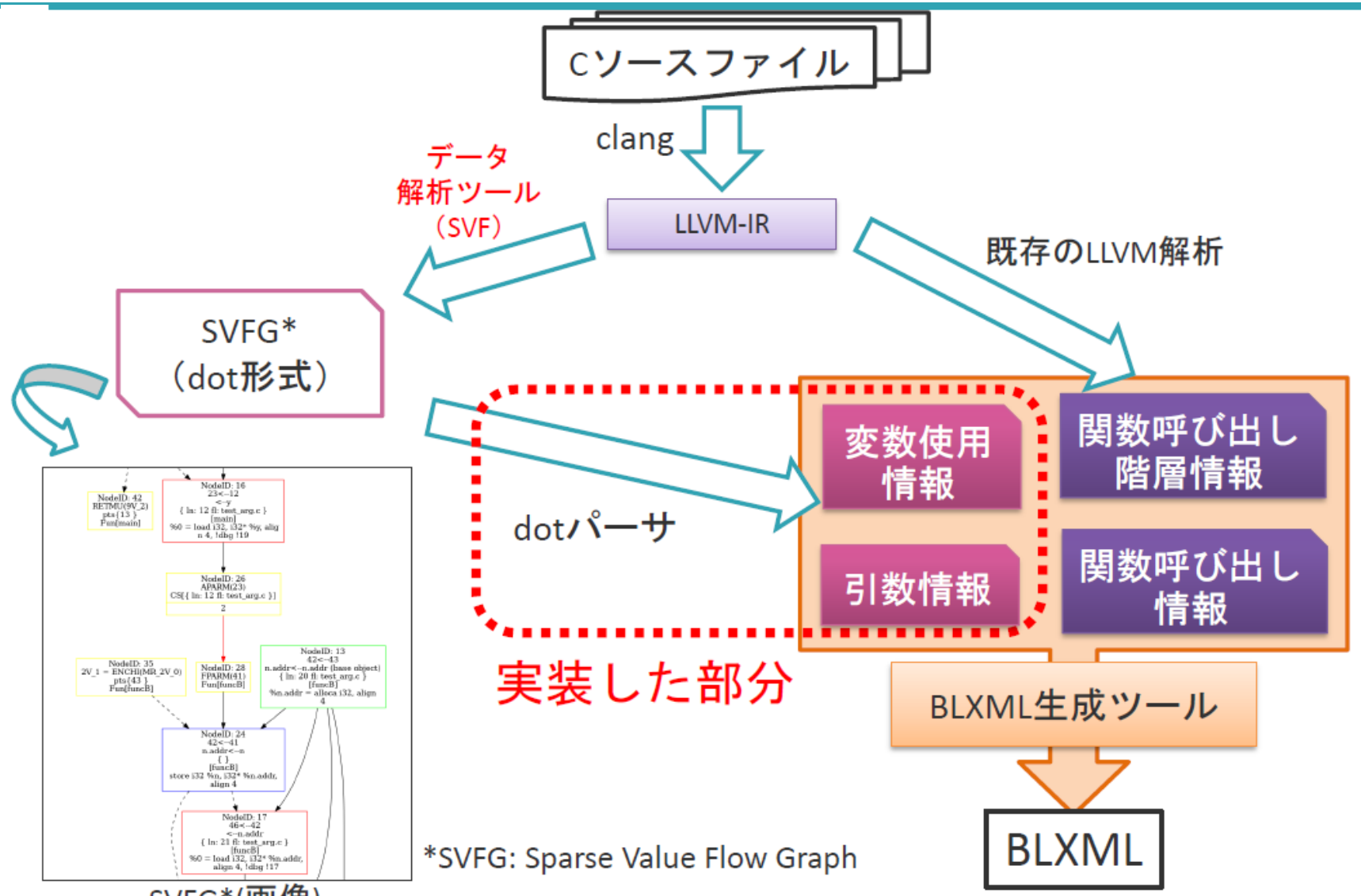
http://www.amalthea-project.org/





入力拡張：Cコード関数単位並列化

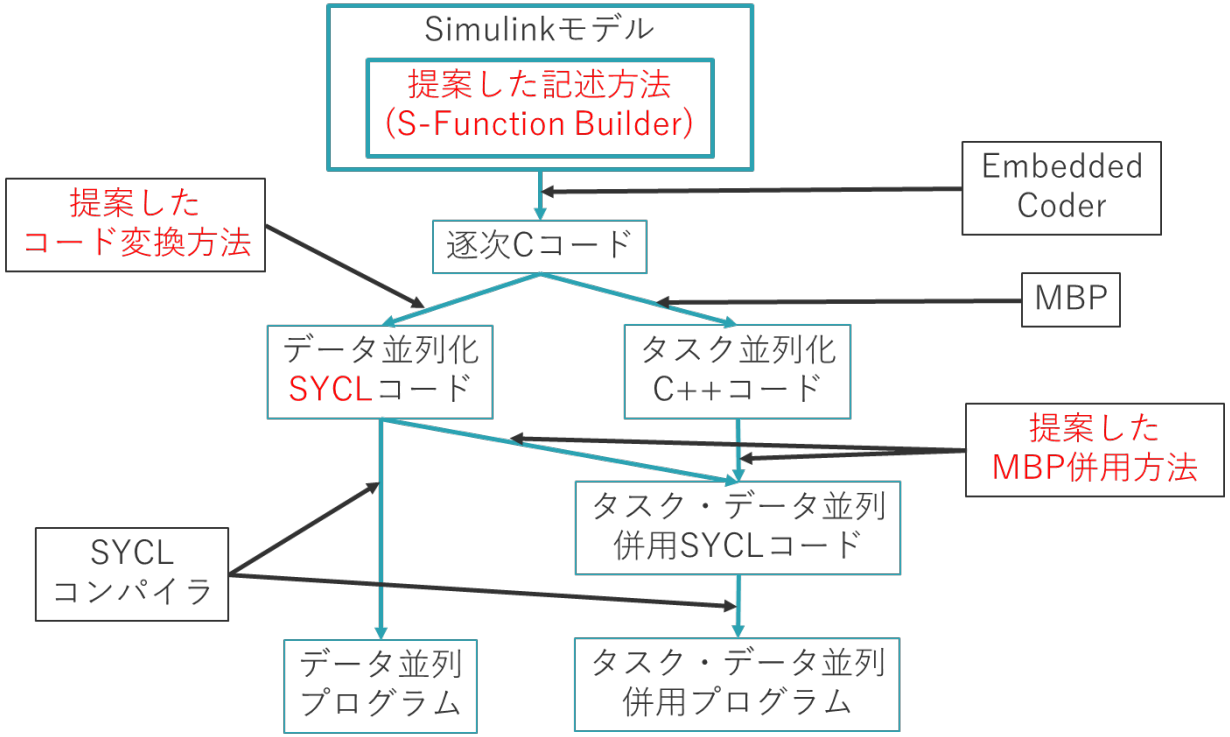
福島[ETNET2021](予定)

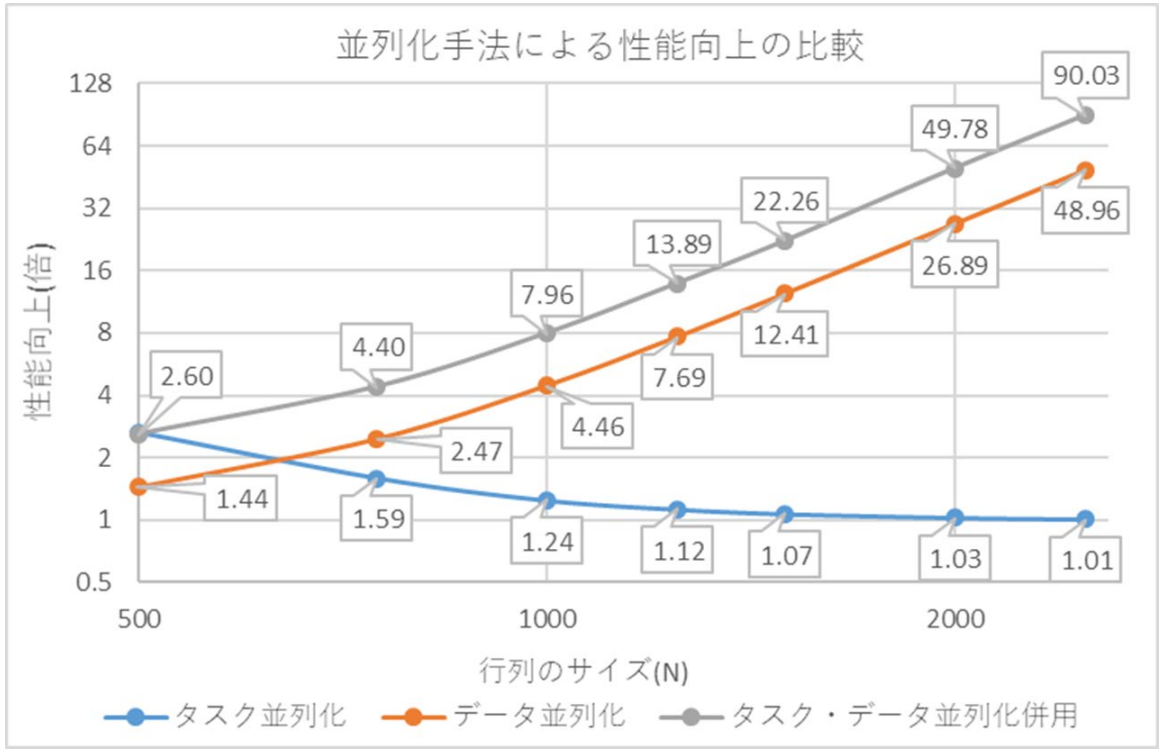


入力拡張：データ並列

徐[ETNET2021]（予定）

- S-Function Builderを用いてforループを記載し、オープンツール（PPCG等）により自動/手動で並列化
- 従来からのMBPによるタスク並列と組み合わせることが可能

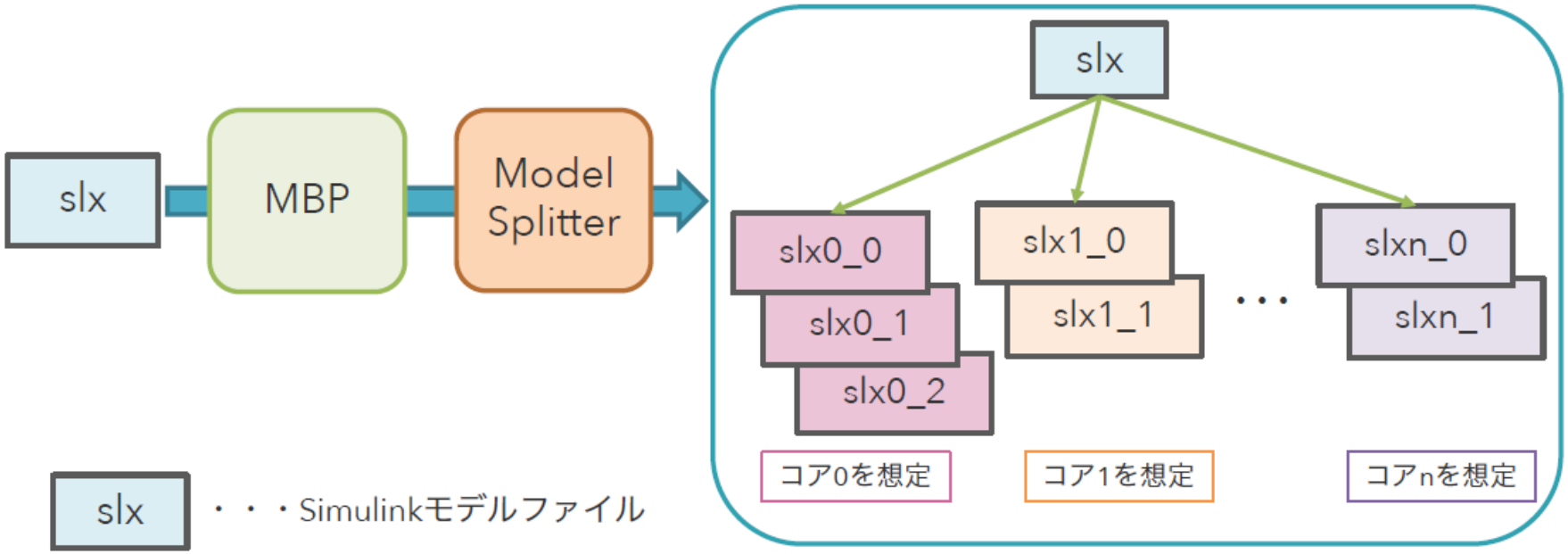




出力拡張：モデル分割

山田[SIGEMB2020]

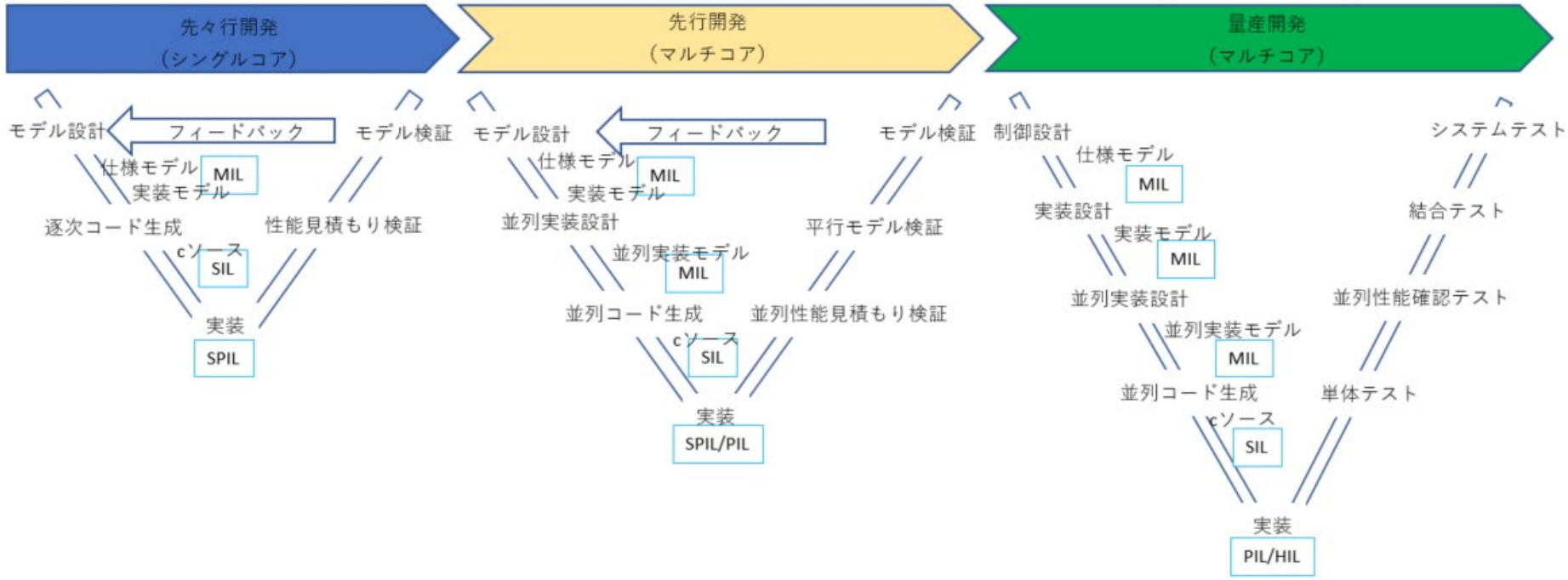
- Simulinkのふるまいを変えずにコア割当に対応した分割モデルを出力



開発ステージ毎のV字プロセス

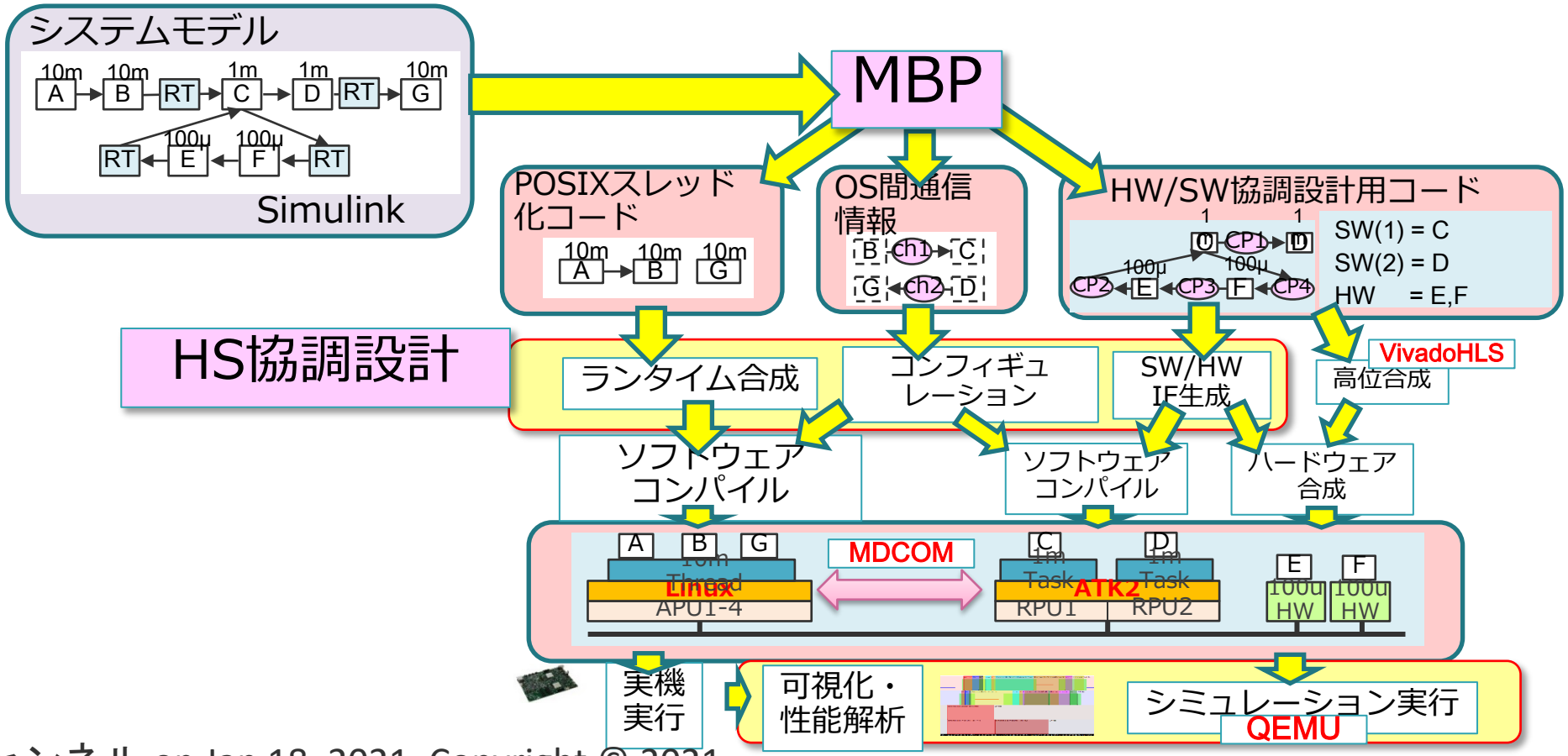
生沼[ガイオサイバーチャネル2020]

- **それぞれの開発ステージでV字プロセスを設定し、ツールをマッピング**



HS-MBP

- ヘテロジニアスプロセッサ向けにコア割当されたBLXMLから RTOS＋Linux＋FPGAで動作するランタイムの生成を実現

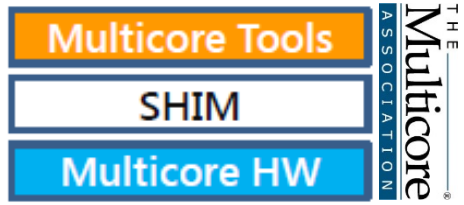


目次

- どうしてモデルレベルでの並列化なのか
- 課題と研究状況
  - 並列化
  - 性能見積
  - 検証
- 組込みマルチコアコンソーシアムについて

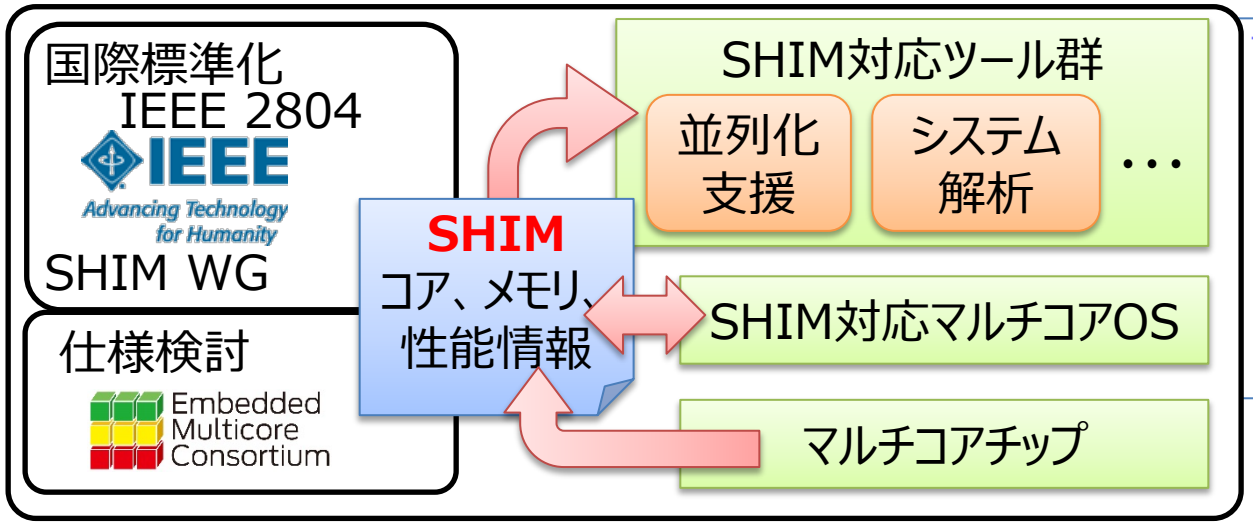
SHIMとは

Software-Hardware Interface for Multi-many-core



- 多様なマルチコアチップを抽象化したXML記述
  - コア種類・数、メモリ配置、アドレスマップ、通信、コア→メモリ性能情報等が、数百ページの説明書を読まずとも、機械的に読める
  - 性能情報の例：コアAからメモリ番地Xにアクセスしたときの(best, typ, worst)レイテンシ（右下図参照）、LLVM-IR**命令ごとのプロセッサ性能情報**
  - ツール群、OS等がSHIM対応することにより、多様なマルチコアチップを共通的に扱えるようにすることが目的

SHIM2.0はIEEE 2804-2019として標準化
Open SHIM Github https://github.com/openshim/shim

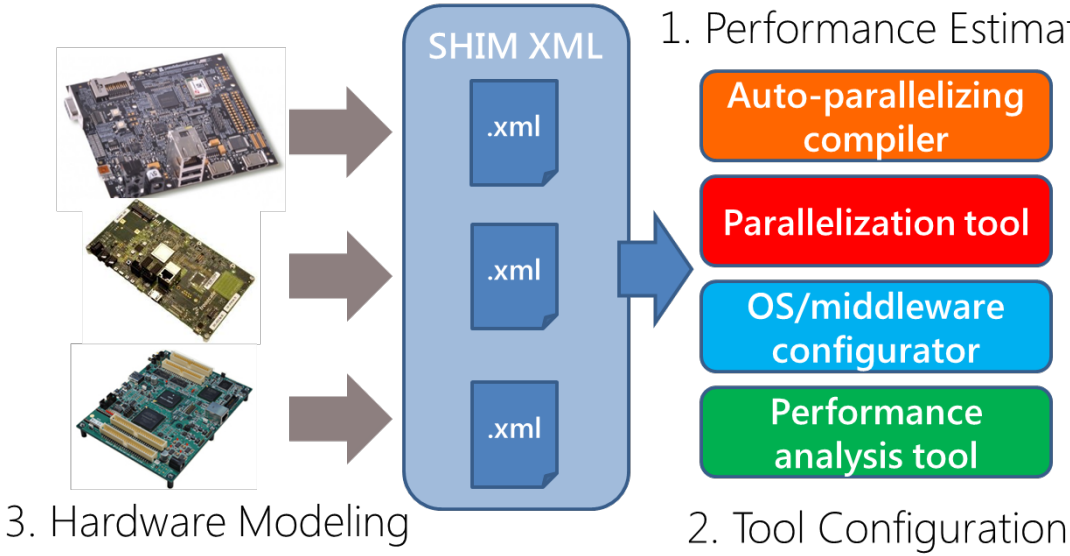



```
<MasterSlaveBinding slaveComponentRef="LRAM_B
  <Accessor masterComponentRef="CPU_BOCOP2">
    <PerformanceSet>
      <Performance>
        <accessTypeRef>Instruction_Fetch</acc
        <Pitch best="1.0" typical="1.0" worst
        <Latency best="1.0" typical="1.0" wor
      </Performance>
      <Performance>
        <accessTypeRef>Load_Algined_Byte</acc
        <Pitch best="1.0" typical="1.0" worst
        <Latency best="1.0" typical="1.0" wor
      </Performance>
```

コア→メモリ性能情報
SHIM記述例

Embedded Multicore Consortium

SHIMのユースケースとメリット



- システム性能見積
- ハードウェアモデリング/複数HWの対応
- ツールコンフィギュレーション

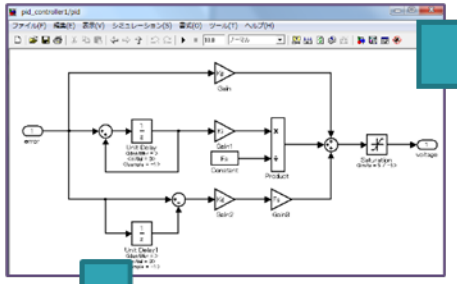
- マルチコアにおけるアプリケーション実行性能見積
- マルチコア選定時のアプリケーション実行性能比較
- 異なるマルチコアへのアプリケーション移植の際の性能見積
- 複数マルチコアをターゲットとしたソフトウェア部品開発
- 特定アプリケーション向けに特化したマルチコアを企画する際の性能評価
- マルチコア向け開発支援を行う各種ツールの開発コスト低減とSHIM対応ツールエコシステム

見積誤差±２０%が目標

Embedded Multicore Consortium

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_Saturati
} else if (pid_controll
{
  pid_contr
} 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_controlle
    } else if (pid_controller1_pid_Sum2_1 &lt;= pid_co
    {
      pid_control
    } else {
      pid_control
    }
    /* End of Sat
  </code>
  <code
    pid_
  </code>
  <performa
  <performanc
  <forward_block=
    <var line="pi
  </forward>
  <backword block
    <var line="pid_controller1_pid_Sum2_1" mode="outpu
  </backword>
</block>
```

ブロックレベル構造XML (BLXML)

④抽出コード処理量見積

```
<ComponentSet name="Board_BO">↓
  <ComponentSet name="Cluster_BOCO">↓
    <ComponentSet name="PE_BOCOP1">↓
      <SlaveComponent name="LRAM_BOCOP1"
      <MasterComponent name="CPU_BOCOP1"
        <CommonInstructionSet name="I
          <Instruction name="ret">
            <Latency best="10.0"  ypical="10.
            <Pitch best="10.0" typical="10.0"
          </Instruction>↓
```

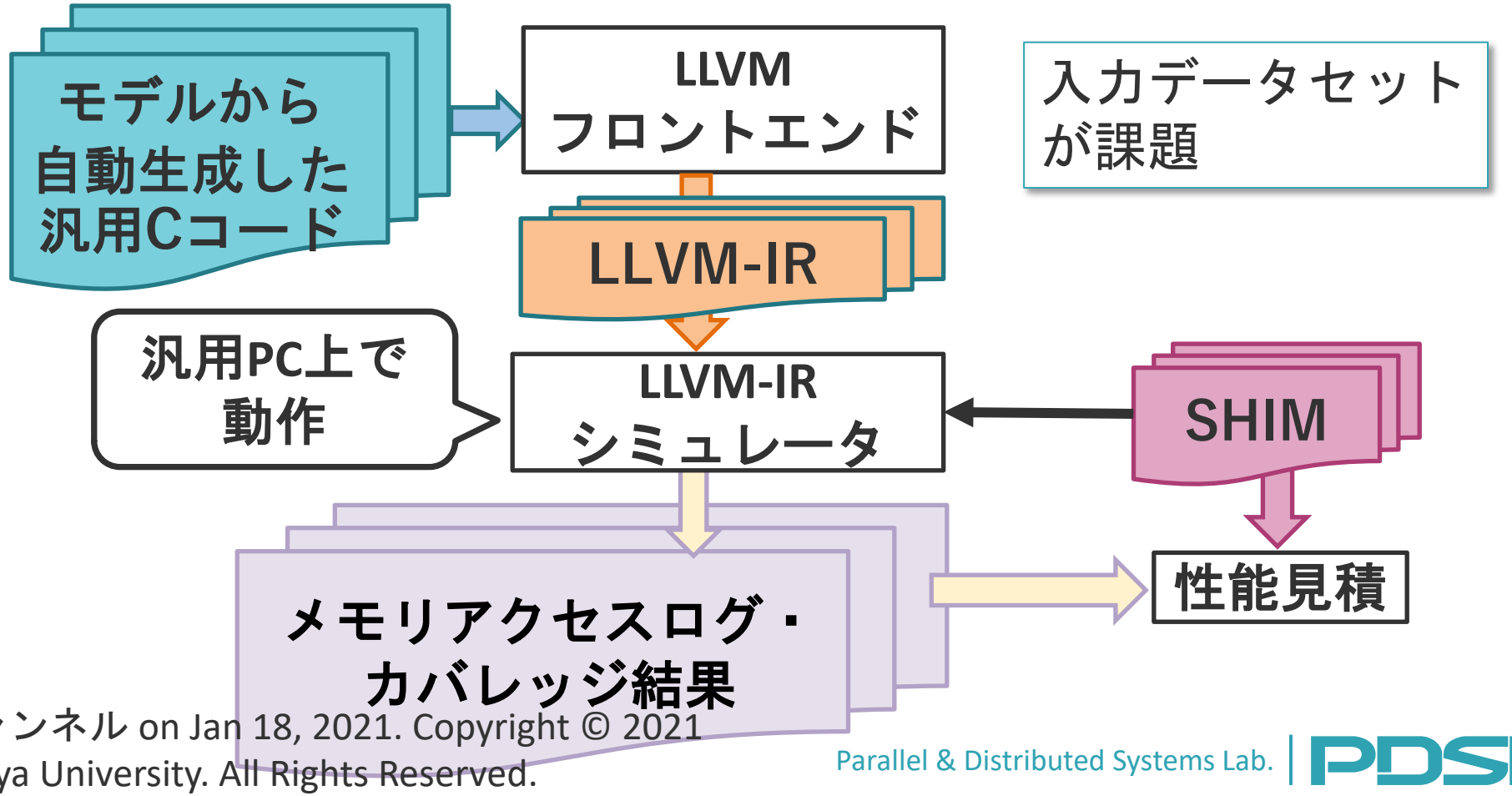
SHIM

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で性能見積を行う

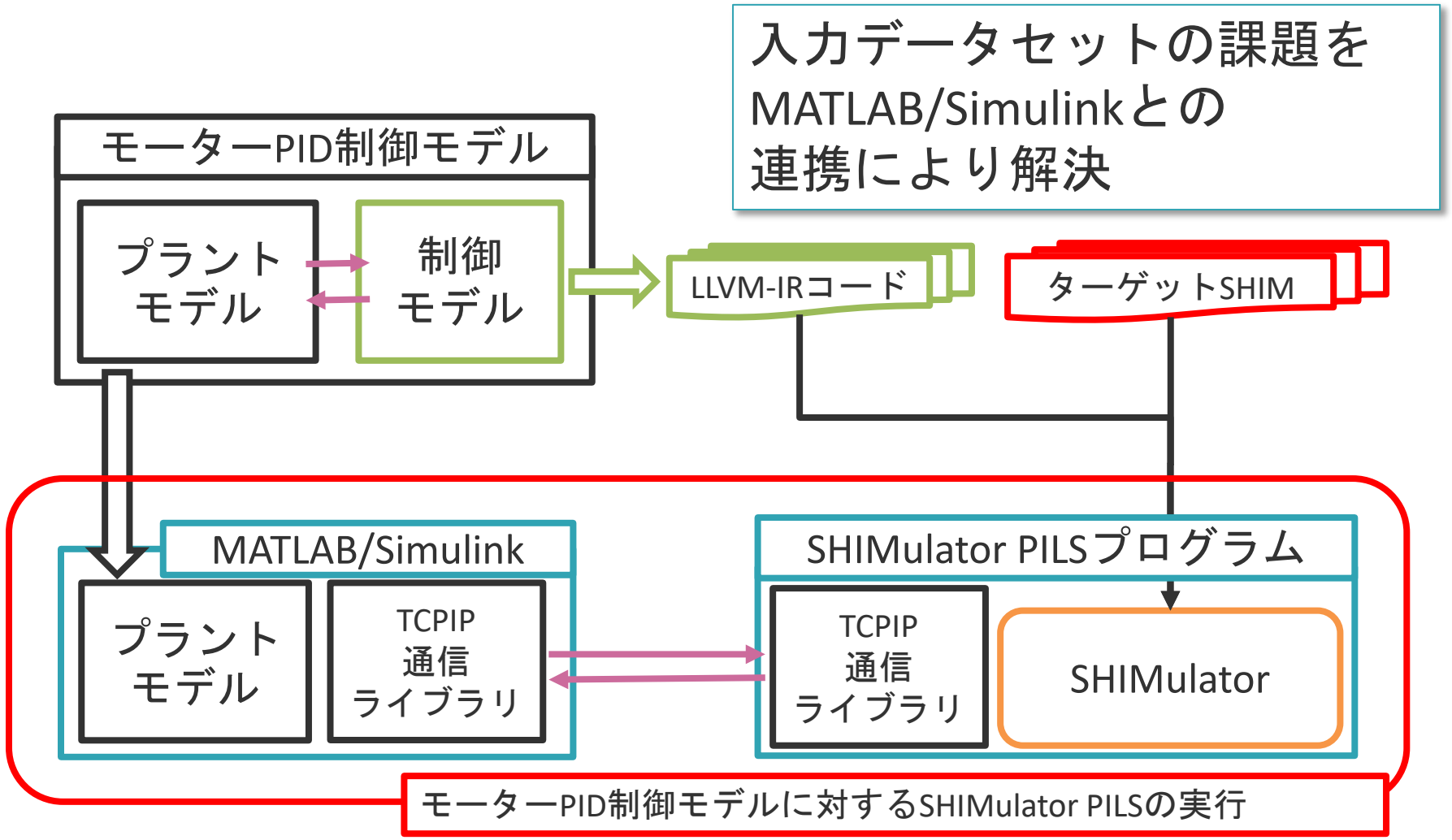
SHIMulator

佐合[ETNET2019]

- 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



SHIMulator PILS



SHIM性能見積の課題

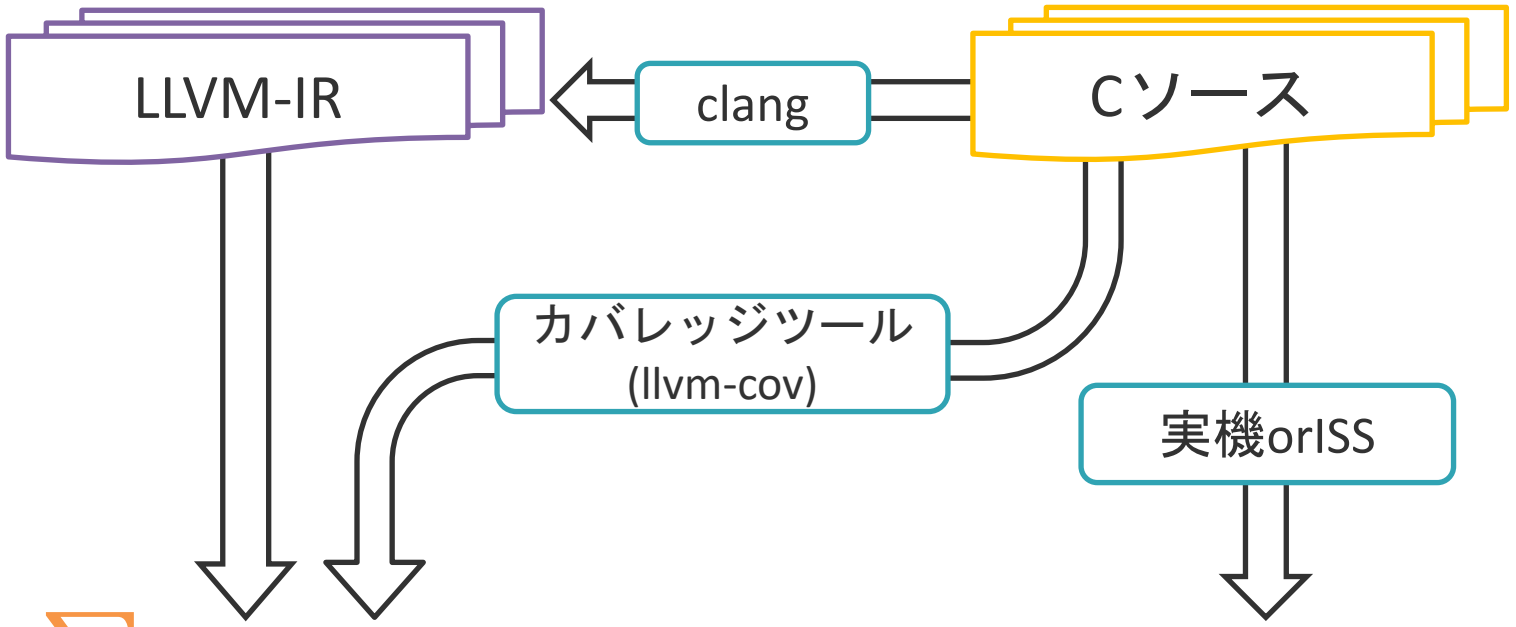
- SHIM性能見積の基本的方法
  - （事前・ターゲットごと）LLVM IR各命令の性能情報を計測→SHIM XMLを作成
  - 対象ソフトウェアをClang (LLVMコンパイラ)で中間言語表現にし、SHIM XMLを用いて見積もる
  - $\sum_{i \in IR}(i$の性能 $\times$ iの出現回数$)$により簡単に求まりそうであるが、以下に示す理由で実際には簡単ではない
- LLVM IRの特徴
  - 様々なアーキテクチャに対し汎用的に対応するため、レジスタ数無限で中間言語表現を生成する
  - 状況に応じ、一種類のLLVM IR命令から様々な種類のターゲットコードが生成される
    - レジスタヒット/ミス、直接ジャンプ、２のべき乗乗除算、命令入れ替え等
- 何をもってLLVM IR命令の性能値とすればよいのか？

当研究室でのSHIM計測・見積の歴史

- 当初は様々なパターンのアセンブラを書いて計測
  - キャッシュミスしないプログラムでSHIMの可能性を検証（西村[ETNET2014]）（この年にSHIM1.0公開）
  - レジスタスピルを考慮することにより性能向上 （溝口[2016]）
  - パイプラインハザードなども考慮することにより性能向上（佐合[ETNET2019]）
- 各IR命令から生成されるアセンブラ列の種類も多過ぎ、アセンブラ記述では多様なISAへの対応が困難であることから方針変更

回帰分析によるSHIM計測

鳥越 [ETNET2020]



$$\sum IR\text{実行回数} \times \text{レイテンシ} = \text{実行サイクル}$$

この式を複数用意して各LLVM-IR命令のレイテンシを変数とする誤差最小問題として解く

利点と課題点

- 利点
  - どのターゲットにも同じサンプルプログラムで計測可能
  - Typicalサイクルが求まるため、見積に用いやすい
- 課題点
  - サンプルプログラムの用意
    - 処理が偏らないようにしながらたくさん用意する必要がある
  - LLVM-IR命令の変数への割り当て
    - 変数をどこまで細分化するか
    - 細分化しすぎるとその分必要サンプル数が増える
    - 大まかすぎると精度が落ちる

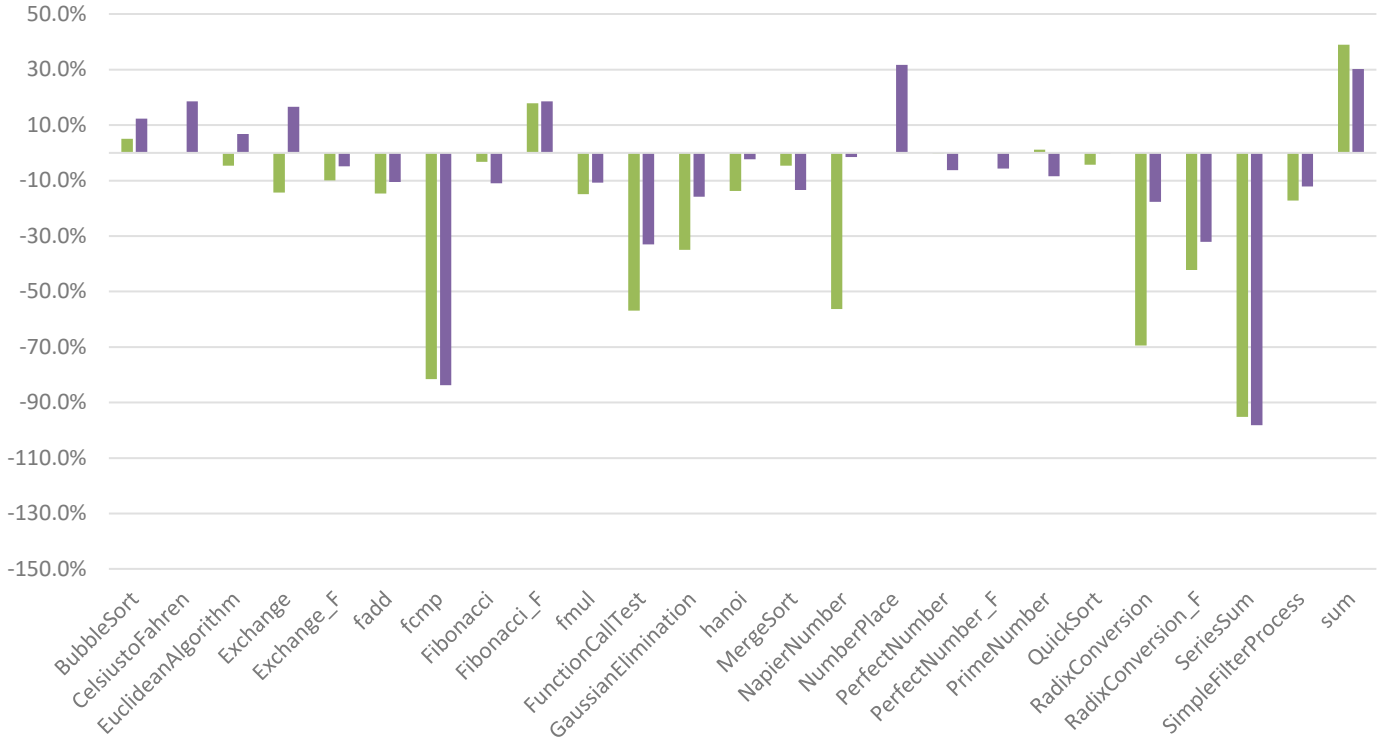
サンプルの準備と変数の設定

- サンプルは２５種類を用意
- 以前の見積で計測したSHIMを参考にするなどして下表のような変数を設定

arithmetic	float	load	store	callret	others
add	fadd	load	store	call	その他の命令
sub	fsub			ret	
sdiv	fdiv				
srem	fmul				
urem	fcmp				
mul					

正規化を行ったSHIM計測

- 各プログラムの誤差（プログラムはサンプルと同一）
  - 計測サンプルと評価サンプルが同じでも誤差大（下図）
  - さらに、各変数にハードウェア・マニュアルから取ってきた値を代入しても誤差大

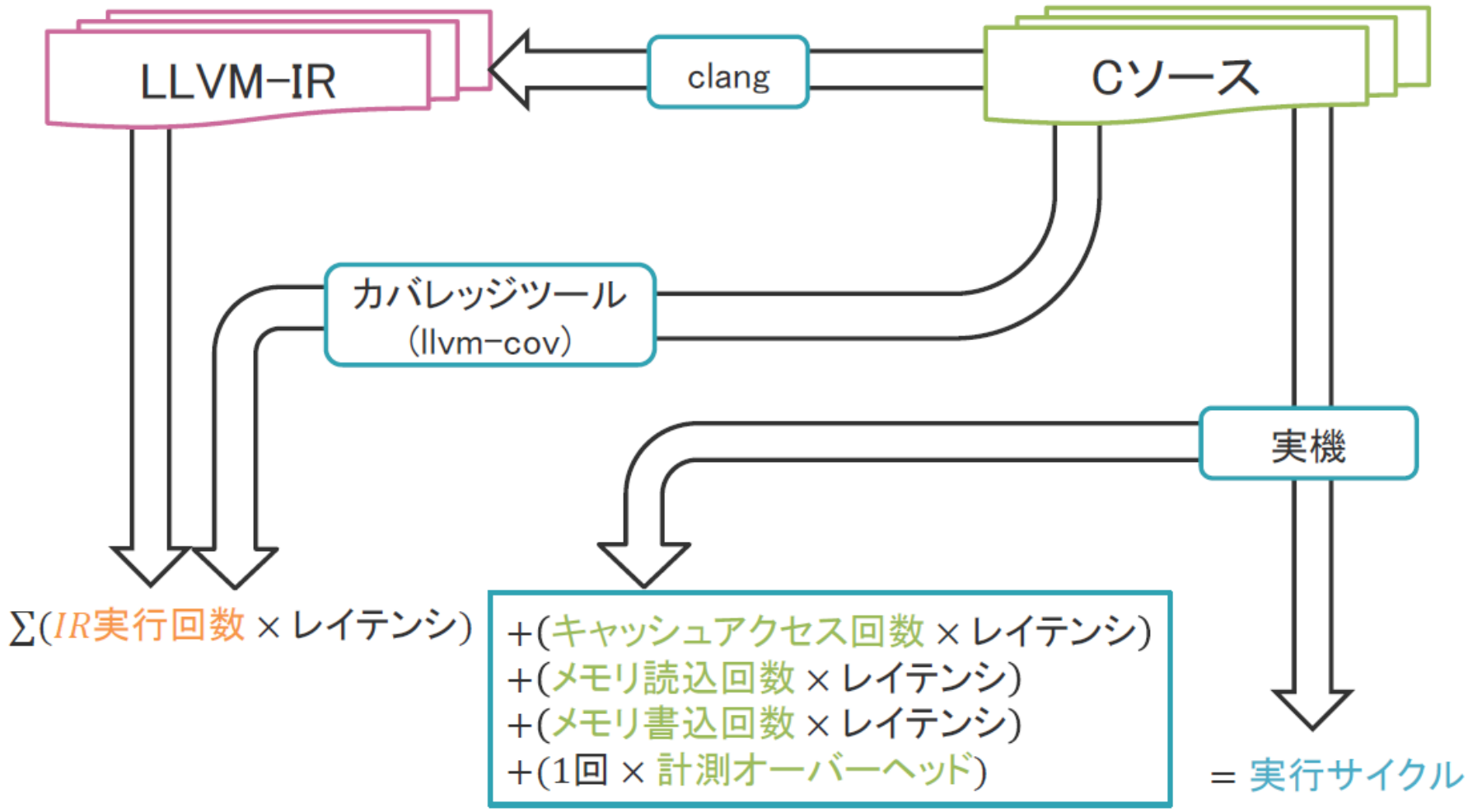


課題と対応の例

- 課題
  - レジスタスピル率、キャッシュミス率がプログラムごとに異なる
  - サイクル数計測オーバヘッド
- 対応
  - 実機計測
    - キャッシュアクセス回数・リフィル回数調査
      - 実際の見積時は外から与えることになるが、今回は計測
    - 実行サイクル数の下限を10000に調整
  - 回帰分析
    - LLVM-IR命令分類変更
    - キャッシュとメモリへのアクセスレイテンシの変数を追加
    - 計測オーバーヘッドの変数を追加
    - 外れ値除外
    - 制約条件追加

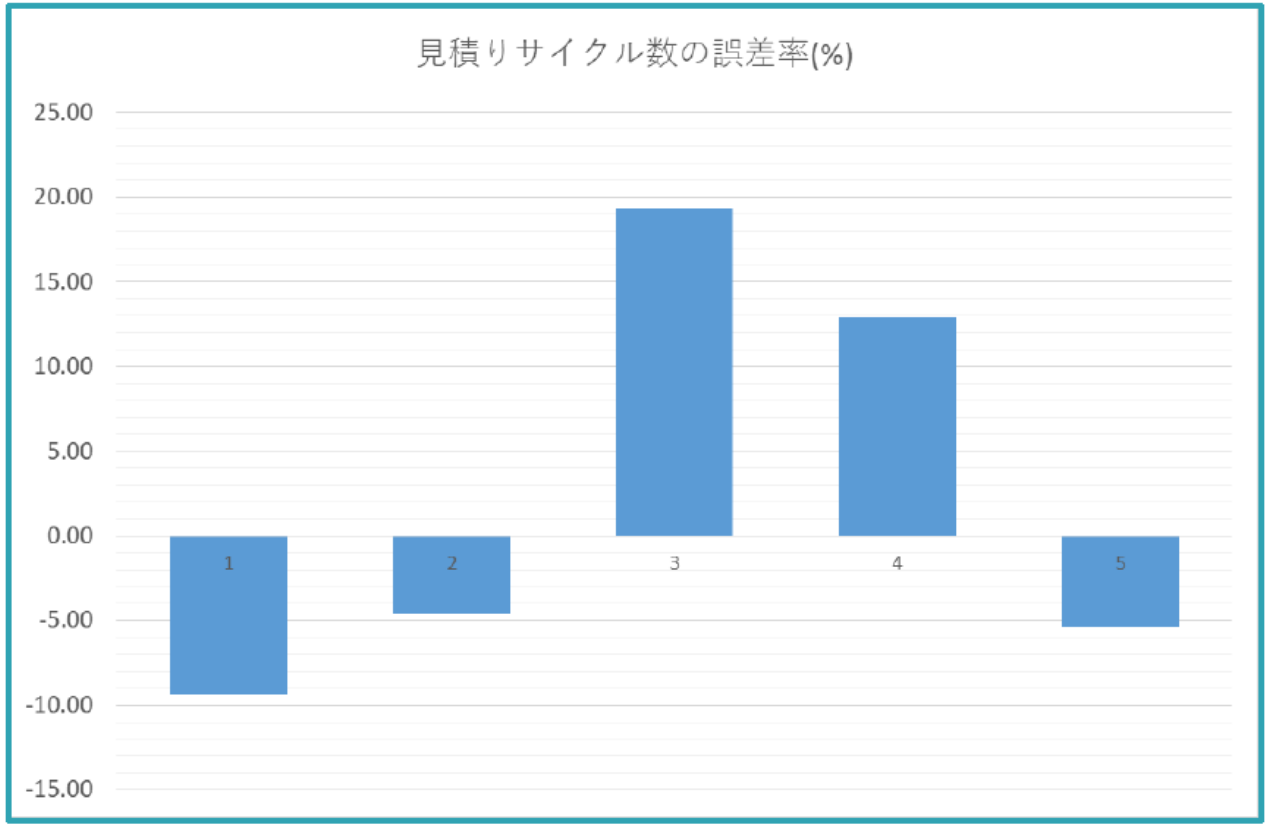
新しい定式化

井ノ川 [ETNET2021]（予定）



評価結果

- サンプルとは異なる評価プログラム（５種）を使用
  - すべてのプログラムで±20%以内を達成



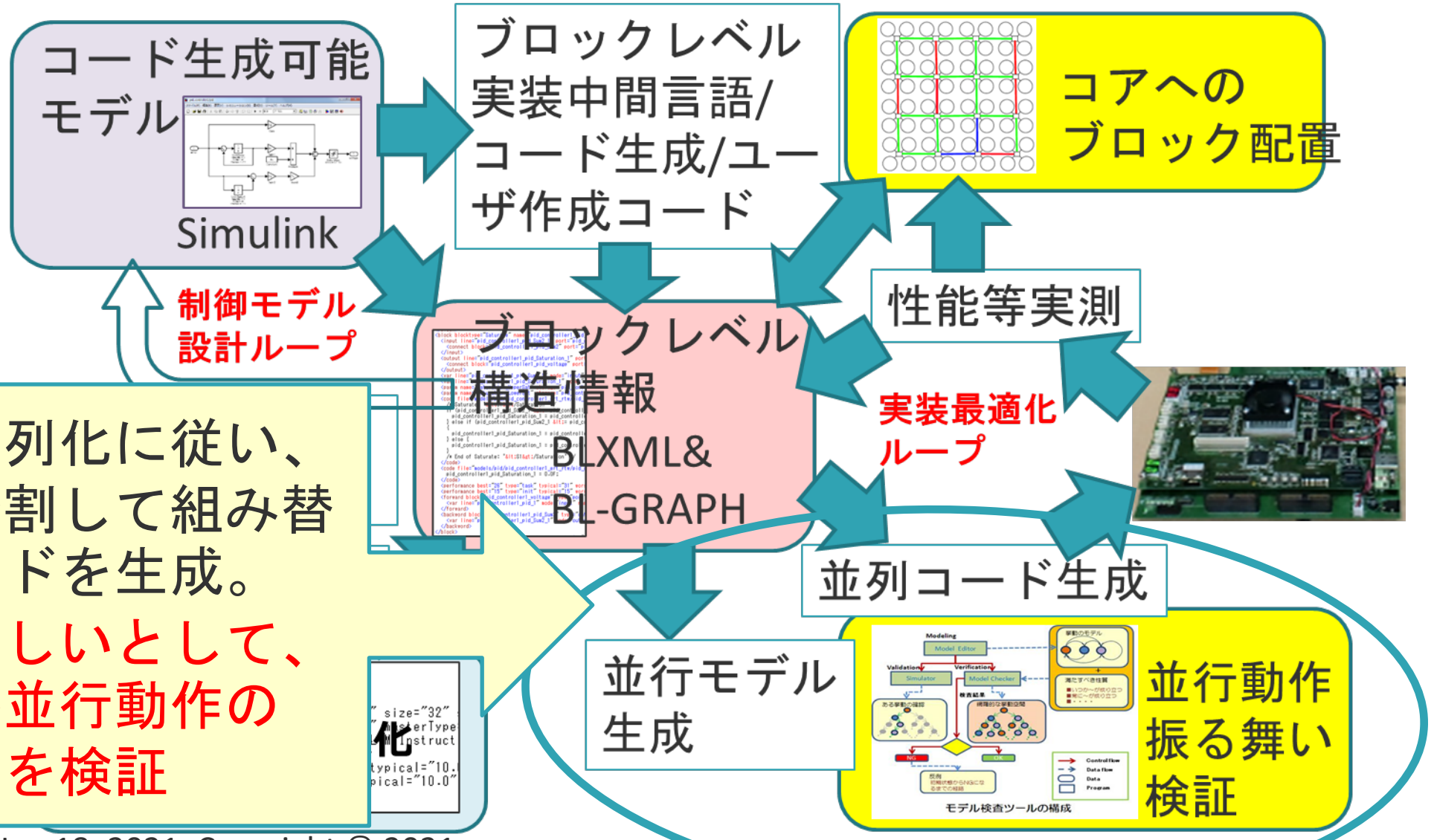
目次

- どうしてモデルレベルでの並列化なのか

- 課題と研究状況
  - 並列化
  - 性能見積
  - 検証

- 組込みマルチコアコンソーシアムについて

モデルレベルでどう検証するのか？

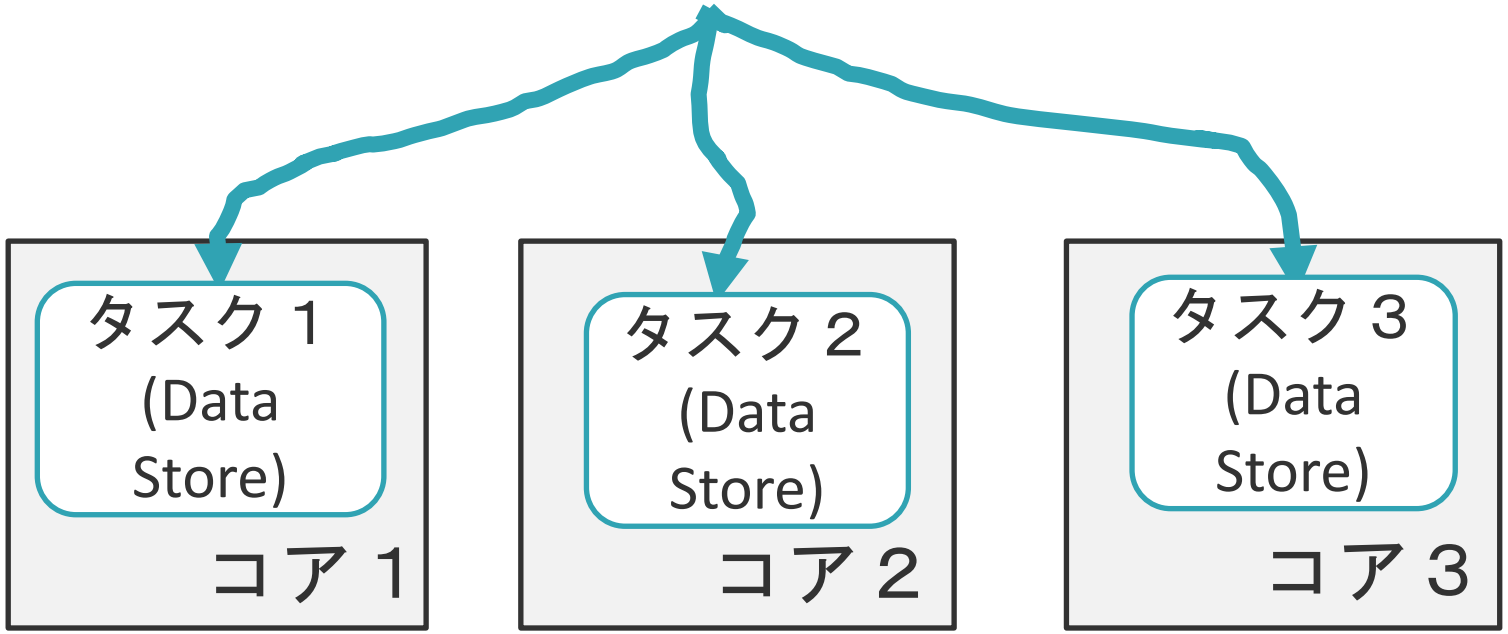


モデルレベル並列化に従い、逐次コードを分割して組み替え、並列コードを生成。

逐次コードは正しいとして、モデルレベル並行動作のふるまいを検証

並列化コードに対する課題

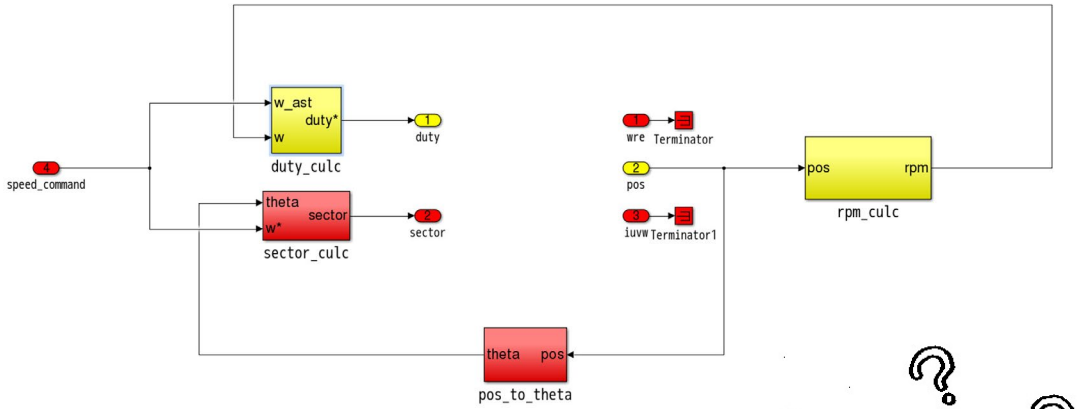
- 並列・並行実行により、タイミングに関する自由度が指数関数的に増大



デッドロック？　読み書き順序逆転？

自動コード生成ツールに対する課題

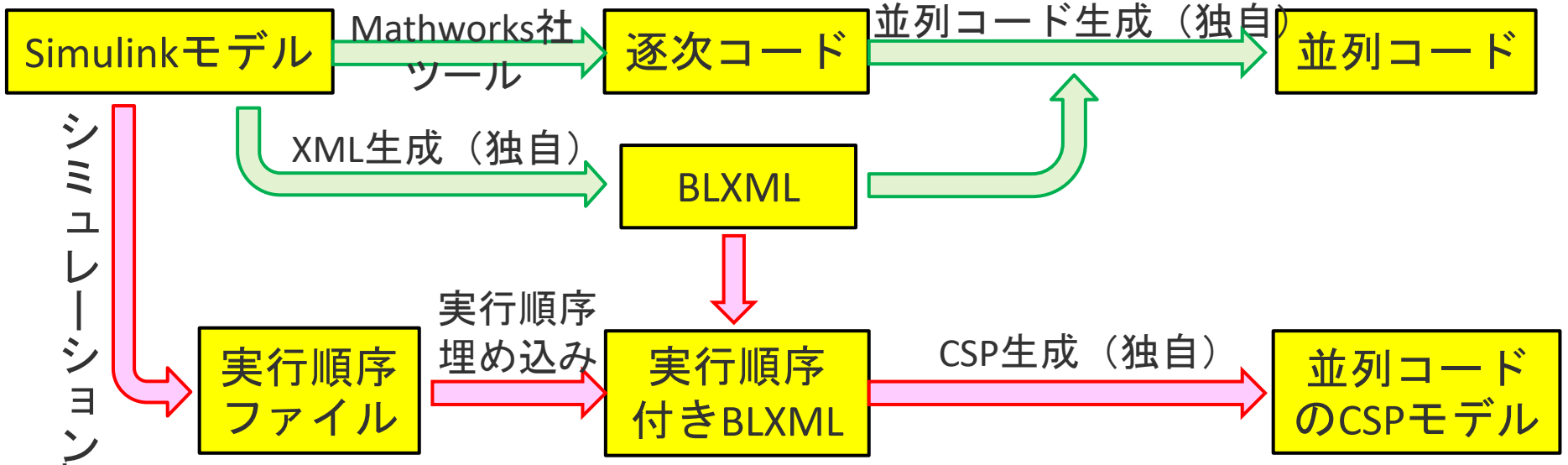
- もとのSimulinkモデルと並列化コードは等価なのか？



```
/*
 *  メインタスク
 *
 *  ユーザコマンドの受信と，コマンドごとの処理実行
 */
TASK(MainTask)
{
        uint8           command;
        uint8           task_no;
        uint32          i;
        CoreIdType      coreid = GetCoreID();

        TickType        val = 0U;
        TickType        eval = 0U;

        syslog(LOG_EMERG, "activate MainTask! @ core%d", coreid);
        /*
         *  タスク番号・コマンドバッファ初期化
         */
        task_no = (uint8) (0);
        for (i = 0U; i < (sizeof(command_tbl) / sizeof(command_tbl[0])); i++) {
                command_tbl[i] = 0U;
        }

        /*
         *  MainCycArm0, MainCycArm1を周期アラームとして設定
         */
        SetRelAlarm(MainCycArm0, TICK_FOR_10MS, TICK_FOR_10MS);
```

a. 並列化が原因で発生する問題の検証

- 生成されたCSPモデルを用いてモデル検査
  - 機能要件
    - デッドロック：deadlock-freeを利用して検証
    - 読み書き順序逆転：LTL（線形時相論理）を利用した順序保証検証
  - 非機能要件
    - デッドラインミス：時間情報付CSPモデルとdeadline制約で検証
      （deadline制約：deadline以内に終了しないとデッドロックする)



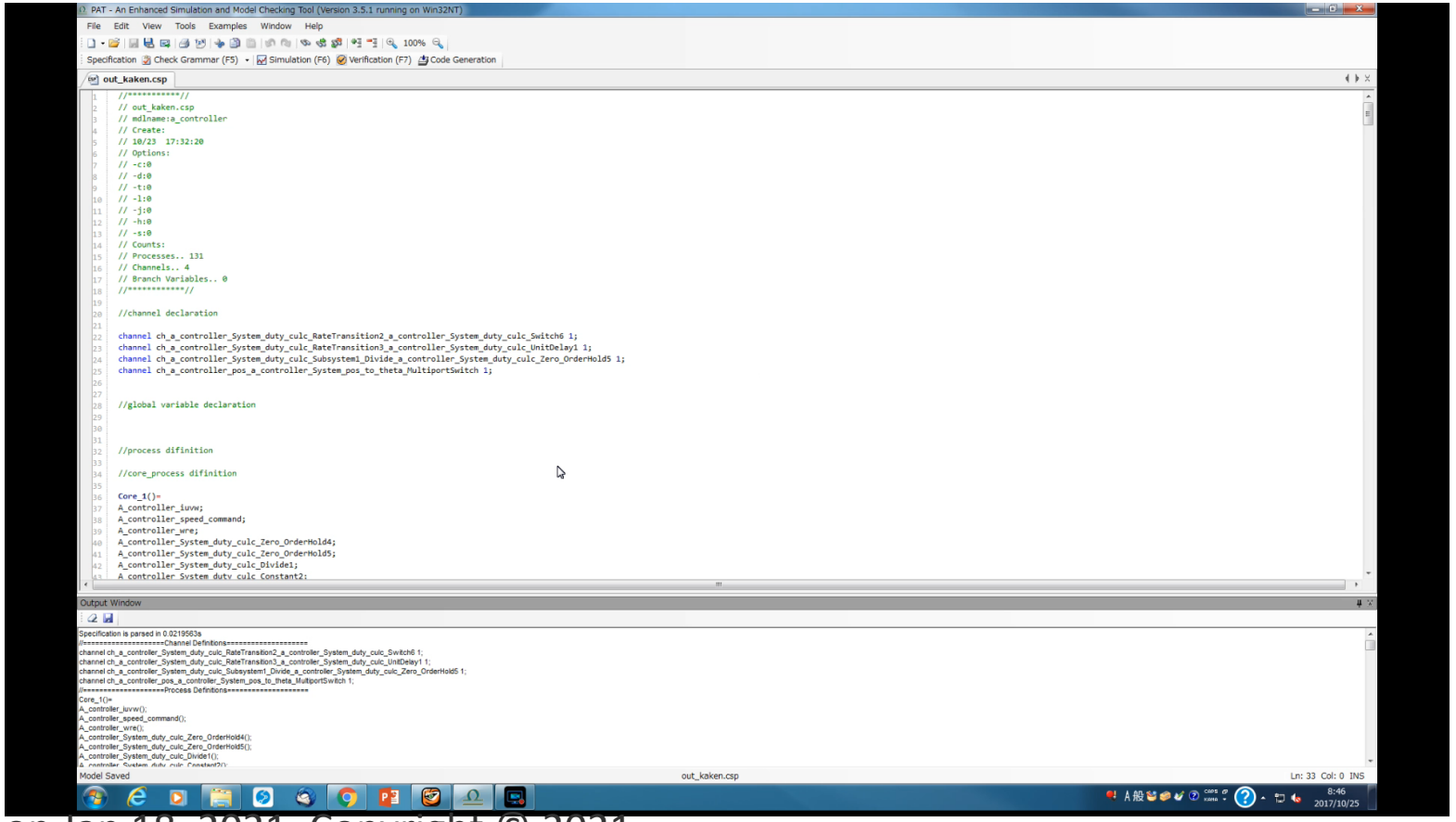
xml2csp

于 [ETNET2020]

- BLXML（コア割当てと実行順序情報を保持）からCSPモデルを自動生成するツール
  - デッドロック、実行順序逆転、デッドラインミスの検証モデルを自動生成が可能
  - 生成されたCSPモデルを検証ツールに通すことでデッドロックなどの検証が可能
  - 検証を行う際の状態爆発の対応策として、AtomicSubsystemによる分割を行った状態で検証を行える階層分割が可能

xml2csp

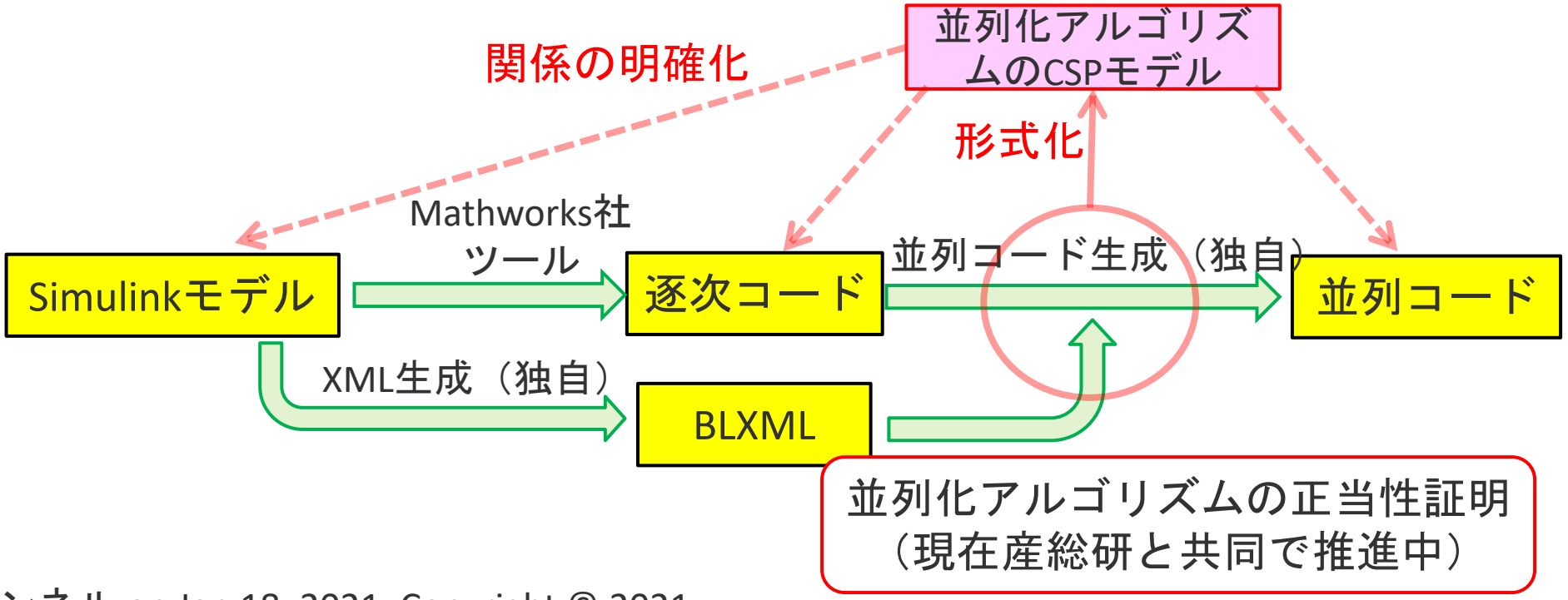
- 生成されたCSPモデルをPATに通すことで
  自動的に検証を行うことが可能

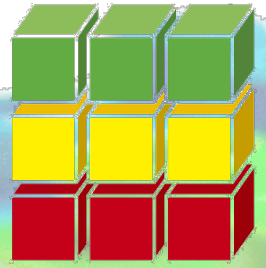


並列化アルゴリズムの正当性の証明

多門 [ETNET2019]

- 並列化アルゴリズム自身をCSPで厳密に記述し、その正当性を証明する
  - 機能要件
    - 並列コードはSimulinkモデルのブロック間依存関係を満たすこと
    - 逐次コードと並列コードの出力が等しくなること





Embedded Multicore Consortium

www.embeddedmulticore.org

組込みマルチコアコンソーシアム

ハードベンダ/ソフトベンダ/メーカを繋ぎマルチコア活用を支援

2020-11

名古屋大学　枝廣　正人

イーソル(株)　権藤　正樹

ガイオテクノロジー(株)　岩井 陽二

Embedded Multicore Consortium

組込みマルチコアの課題

EMC
設立動機

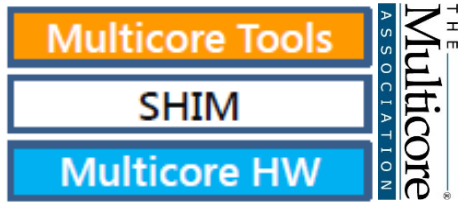
- マルチコアプロセッサはアーキテクチャの自由度が高く、各種ツールやプラットフォーム支援が重要
- 様々な並列化手法、ライブラリ、ツールを組合せるには様々な知見が必要
- システムベンダから半導体ベンダまで、すべての関連技術の協働が必要
- 関連業界で協力・連携し、(1) 活用支援、(2) ビジネス推進、(3)市場の活性化貢献を実現することが必要

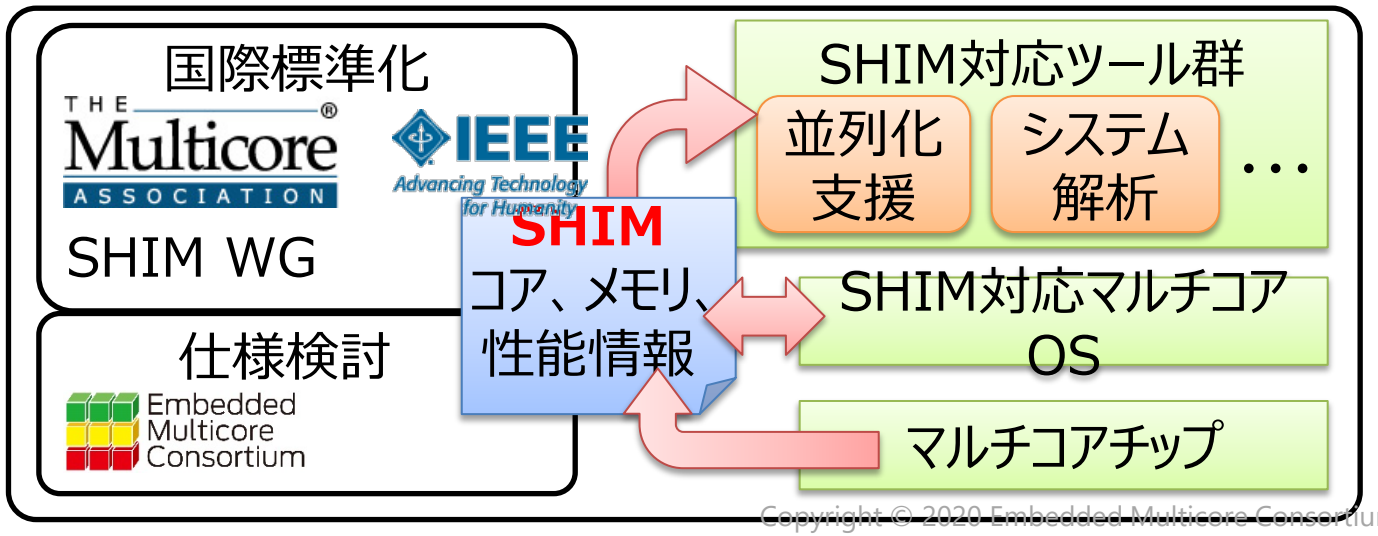
様々なベンダや大学が集まり連携するための場が求められている
→2014年10月組込みマルチコアコンソーシアムを設立

Embedded Multicore Consortium

組込みマルチコアコンソーシアムの取り組み

- SHIM 1.0 の標準化に貢献 (Software-Hardware Interface for Multi-many-core)
  - 多様なマルチコアチップを抽象化したXML記述
  - コア種類・数、メモリ配置、アドレスマップ、通信、コア→メモリ性能情報等が、数百ページの説明書を読まずとも、機械的に読める
  - 性能情報の例：コアAからメモリ番地Xにアクセスしたときの(best, typ, worst)レイテンシ
  - ツール群、OS等がSHIM対応することにより、多様なマルチコアチップを共通的に扱えるようにすることが目的





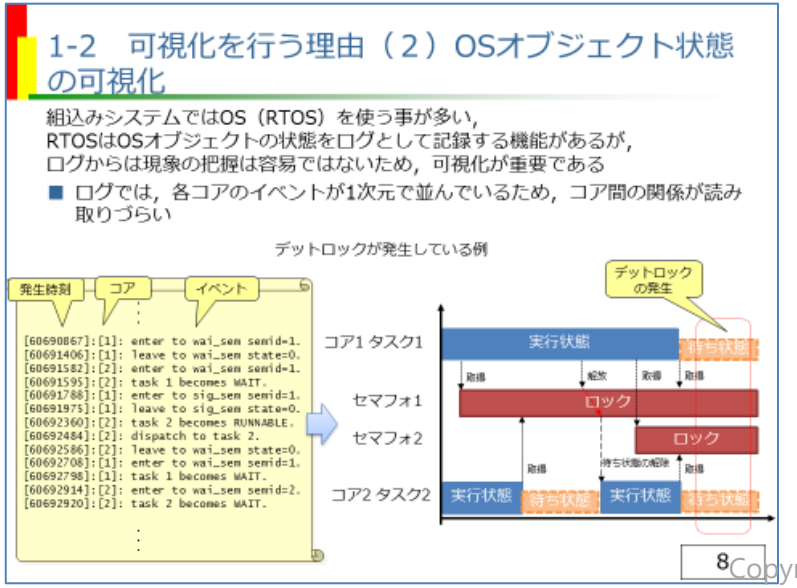

```
<MasterSlaveBinding slaveComponentRef="LRAM_B
  <Accessor masterComponentRef="CPU_BOCOP2">
    <PerformanceSet>
      <Performance>
        <accessTypeRef>Instruction_Fetch</acc
        <Pitch best="1.0" typical="1.0" worst
        <Latency best="1.0" typical="1.0" wor
      </Performance>
      <Performance>
        <accessTypeRef>Load_Algined_Byte</acc
        <Pitch best="1.0" typical="1.0" worst
        <Latency best="1.0" typical="1.0" wor
      </Performance>
```

コア→メモリ性能情報
SHIM記述例

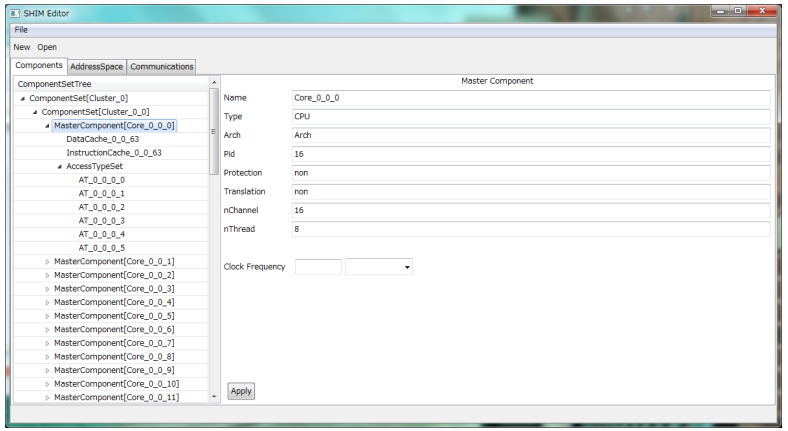
Embedded Multicore Consortium

これまでの会員向け公開成果

- マルチコア技術導入ガイド
  - 主にマルチコア特有の技術に関し、基本的な事項を経験豊かな専門家によってわかりやすく解説
- SHIM利用文書およびサンプルプログラム類
- モデルベース並列化プログラム類（名古屋大版評価バイナリ配布）



←マルチコア技術導入ガイド

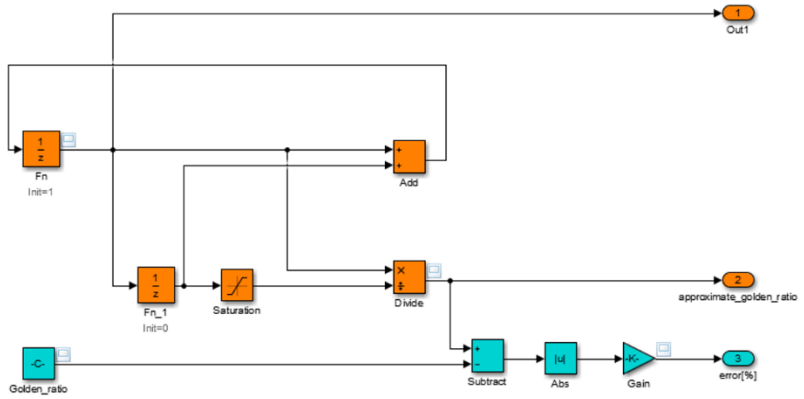


↑SHIM Editor

Embedded Multicore Consortium

これまでの一般向け公開成果

- MCA MPP和訳 (Multicore Programming Practice)
  - マルチコアを利用するための基本知識とベストプラクティス集
  - 2017.3組込みマルチコアコンソーシアム ダウンロードページに公開
  - 2020.11現在 595ダウンロード
- モデルベース並列化サンプル
  - 簡単なサンプルモデルと結果



並列化モデル

データ読み書き間の依存性は計算の部分的な順序を決定する。順序を制限するデータ依存には3つのタイプがあり、真のデータ依存、逆依存、出力依存がある。(図8)

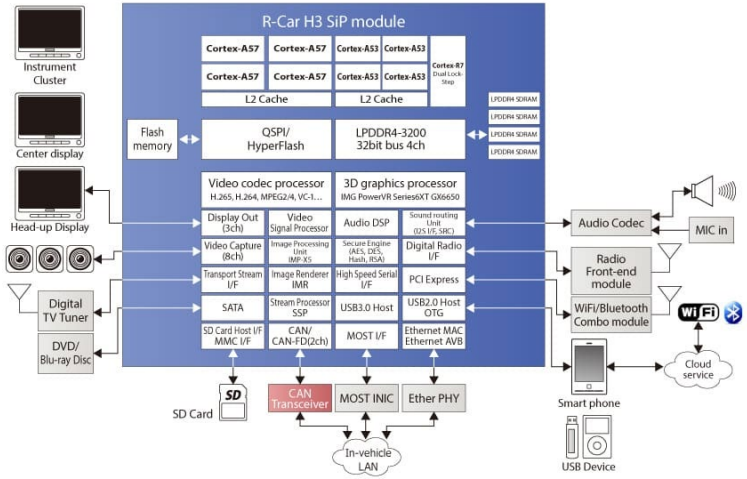
真のデータ依存は、あるデータ値への書き込みが終わるまでは読み込みができないような操作間の順序を示す。これはアルゴリズム内の基本的な依存であるが、このデータ依存性の影響を最小化するようアルゴリズムを改良することもできる場合もある。

MPP

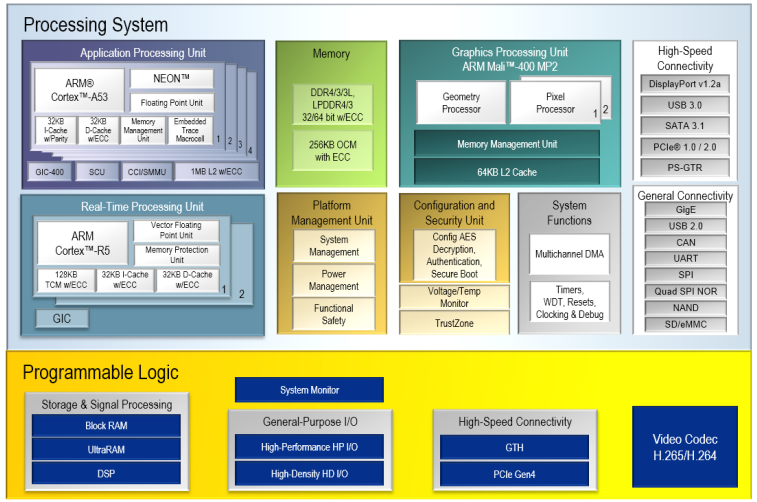
Embedded Multicore Consortium

AI向けに様々なヘテロジニアスSoCが登場

現状



R-Car H3
デュアルコアARM A57 CPU
クアッドコアARM A53 CPU
Video Codec, 3D Graphics,
Audio DSP, etc.
（Xilinx社WWWから転載）



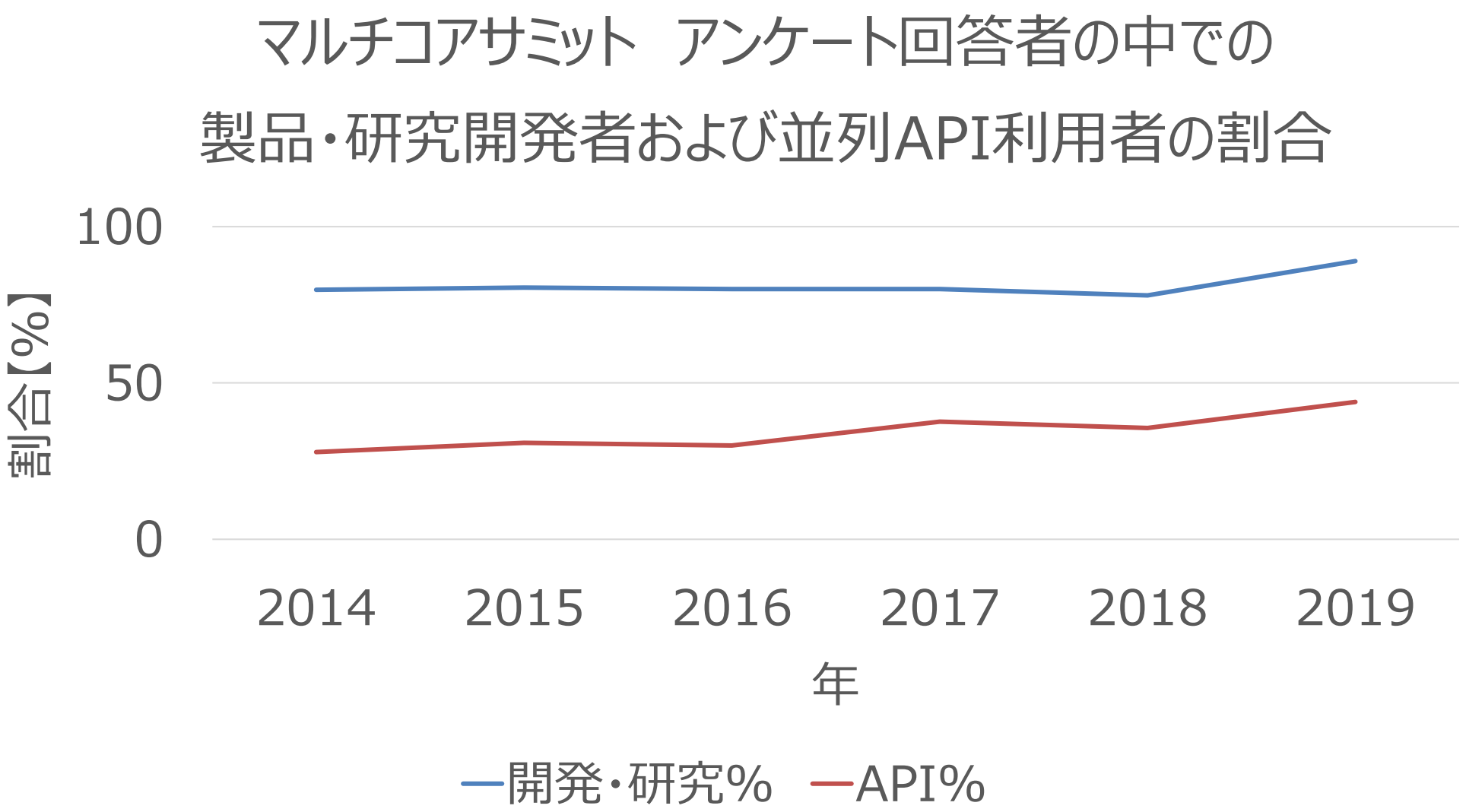
Xilinx Zynq UltraScale+ EV
FPGA
クアッドコアARM A57 CPU
デュアルコアARM R5 CPU
ARM Mali GPU
（Xilinx社WWWから転載）

NVIDIA Jetson Nano
128コアNVIDIA Maxwell GPU
クアッドコアARM A57 CPU
（NVIDIA社WWWから転載）

Embedded Multicore Consortium

しかし、まだまだマルチコア技術者は少ない

現状



Embedded Multicore Consortium

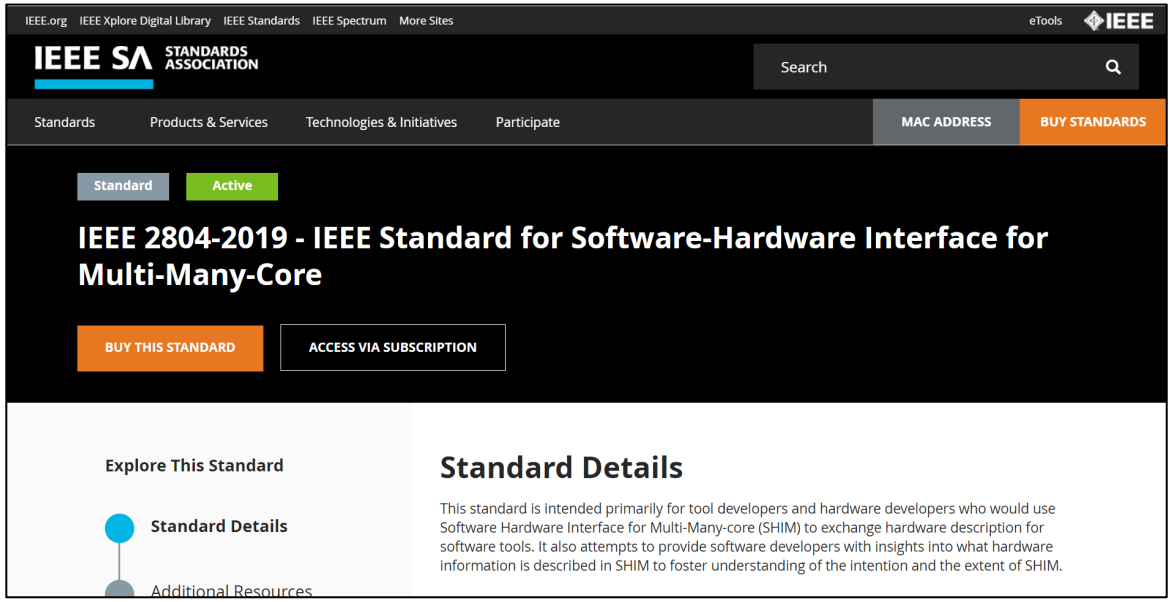
組込みマルチコアコンソーシアム最近の取り組み

- SHIM2.0
  - 2019年IEEE標準化
  - ヘテロジニアス拡張、アーキテクチャ詳細記述、電力記述, etc.
- ヘテロジニアス・アーキテクチャ向けモデルベース並列化
- マルチコア向けプログラミング手法
  - 組込みマルチコアサミット「初心者がマルチコアソフト開発を成功させるポイント」

Embedded Multicore Consortium

SHIM2.0がIEEE標準に！

- SHIM2.0では以下の課題について強化
  - ヘテロジニアス対応 / LLVM-IRでは表しきれない命令
    - ハードウェアが持つ画処理・知能処理関数アクセラレータ等
  - 電力見積
    - DVFS (Dynamic Voltage & Frequency Scaling)
  - 通信競合
    - 特にマルチコアでの見積に重要
  - アーキテクチャの表現強化
    - Out-of-Order, SIMDなど
  - キャッシュ/メモリアーキテクチャの表現強化
  - モジュール化による記述量削減
  - etc.
- IEEE標準として承認



Embedded Multicore Consortium

コンソーシアム活動

- マルチコア向け開発支援ツールのためのハードウェア抽象化記述SHIM標準化と導入支援（SHIM委員会）
  - SHIM (Software-Hardware Interface for Multi-Many-Core)
  - SHIM WG, Multicore Association (Chair: M. Gondo (eSOL))
  - NEDO省エネPJから仕様提案、MCA標準として2015年2月V1.0、2019年1月V2.0、2019年秋IEEE標準に
- リファレンスとしてSHIMを利用したマルチコア向け設計支援ツール群を開発
  - MCAとしても公開するSHIM Editorと性能計測ツールに加え、設計支援ツール群を会員向けに無償公開。所定の期間経過後に一般にも公開する可能性有
  - モデルベース並列化委員会
- 様々な並列化手法の知見共有とガイドラインの検討
  - マルチコア適用委員会
- セミナー開催、技術情報提供、MCAとの連携

Embedded Multicore Consortium

今後のEMC

- SHIM2.0のIEC標準化、SHIM3へ
  - SHIM3ではプラットフォーム（基本ソフトウェア含む）のレイテンシについて検討
- ヘテロジニアス向けMBPをはじめとしたツール類の会員向け公開
- マルチコア初心者が開発を成功させるための方法論
- マルチコアに関する知見のフィードバック
  - マルチコア駆け込み寺
  - アンケートにご記入ください

- 活動にご意見をいただくとともに一緒に検討しましょう！

Embedded Multicore Consortium

メンバーシップ

- 会員（2020年11月現在14団体）
  - アイシン精機、ルネサス エレクトロニクス、NSITEXE、eSOL、ガイオテクノロジー、萩原エレクトロニクス、三菱電機、大阪大学、埼玉大学、名古屋大学、早稲田大学アドバンスドマルチコアプロセッサ研究所、他
  - 相互協力：JASA、MCA(Multicore Association)
- メンバーシップ構成
  - 正会員（入会金なし、年会費20万）準会員、特別会員
  - 詳細は http://www.embeddedmulticore.org/
- （参考）SHIM WG Primary Contributing Members
  - Cavium Networks, CriticalBlue, eSOL, Freescale, Nagoya University, PolyCore Software, Renesas, Texas Instruments, TOPS Systems, Vector Fabrics, and Wind River.

Embedded Multicore Consortium