Simulinkモデルベース自動並列化

- Simulinkブロックレベル並列化
- ブロック内自動並列化は従来から多くの研究あり（例：早大OSCAR）
- 本発表では車載制御を想定

アウトライン

- Simulinkモデルベース開発と並列化
- 提案する並列化設計検証フロー
- 並列化実験結果
- 評価版について

組込み制御でのマルチ・メニーコア実現の必要性

自動車業界のニーズ

演算量の増加とﾘｱﾙﾀｲﾑ処理の両立

－高機能化、複合センサー処理

ECU統合によるコスト削減ニーズ　など

統合制御による安全支援、低燃費化等

＋

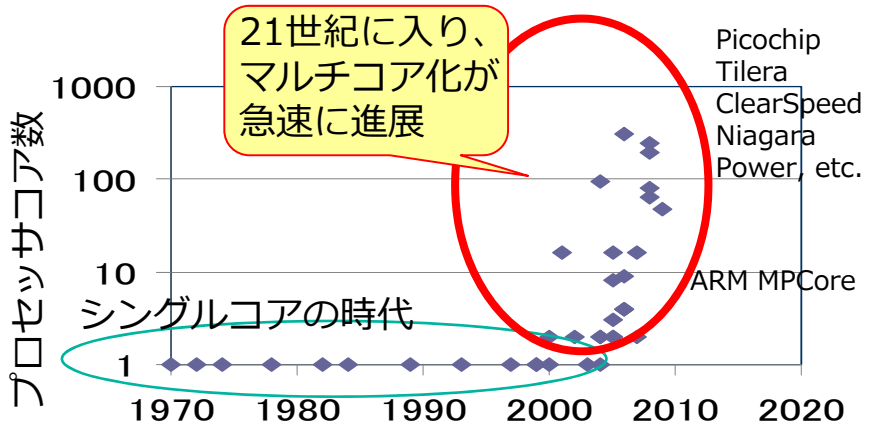
ECU統合によるコスト削減

技術シーズ

CPUはマルチコアが主流へ

－シングルコアでの性能向上の限界

（熱、消費電力の課題など）



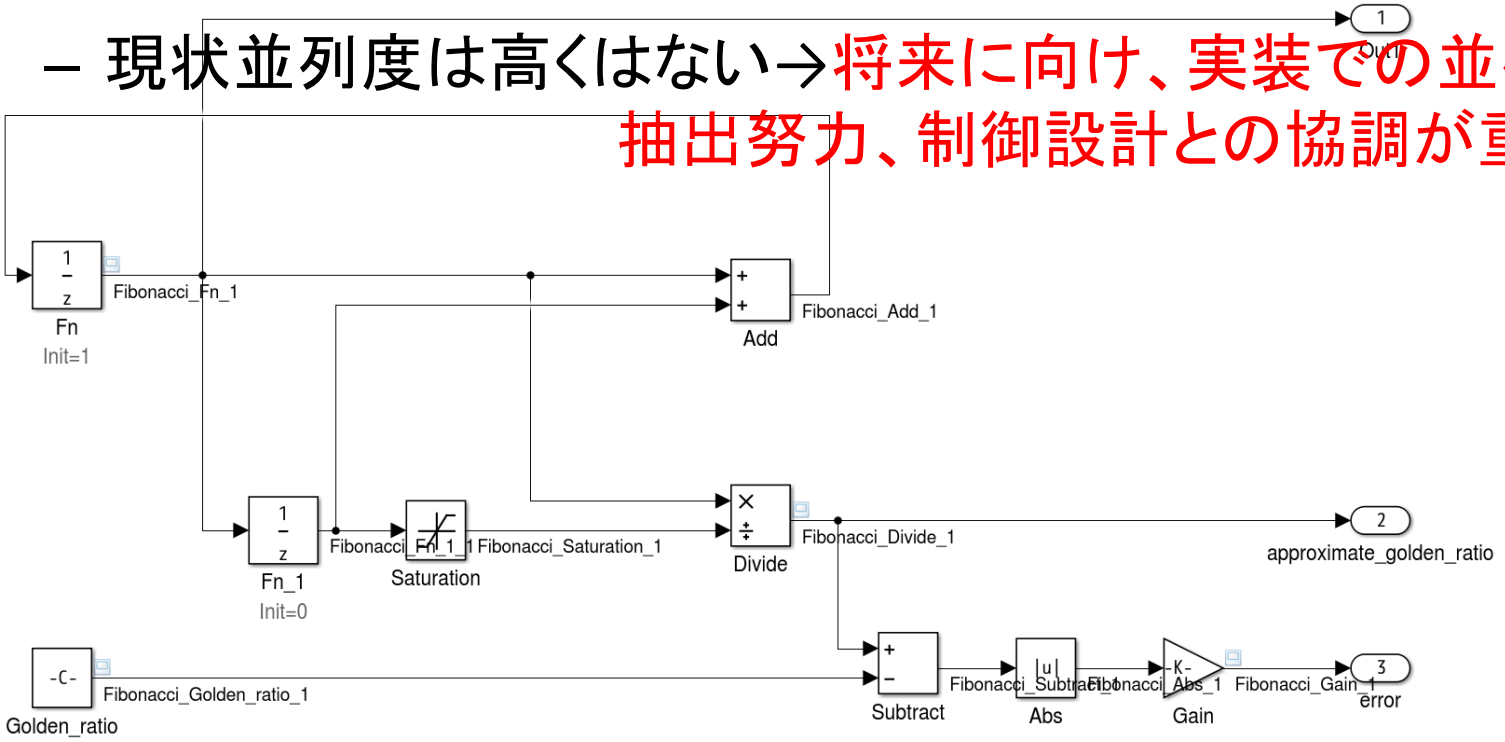
制御系マイコンでも2コア品が登場: ルネサスV850E2/MN4 など

ソフトウェアの並列化が必要不可欠

⇒モデルベース開発からの並列化

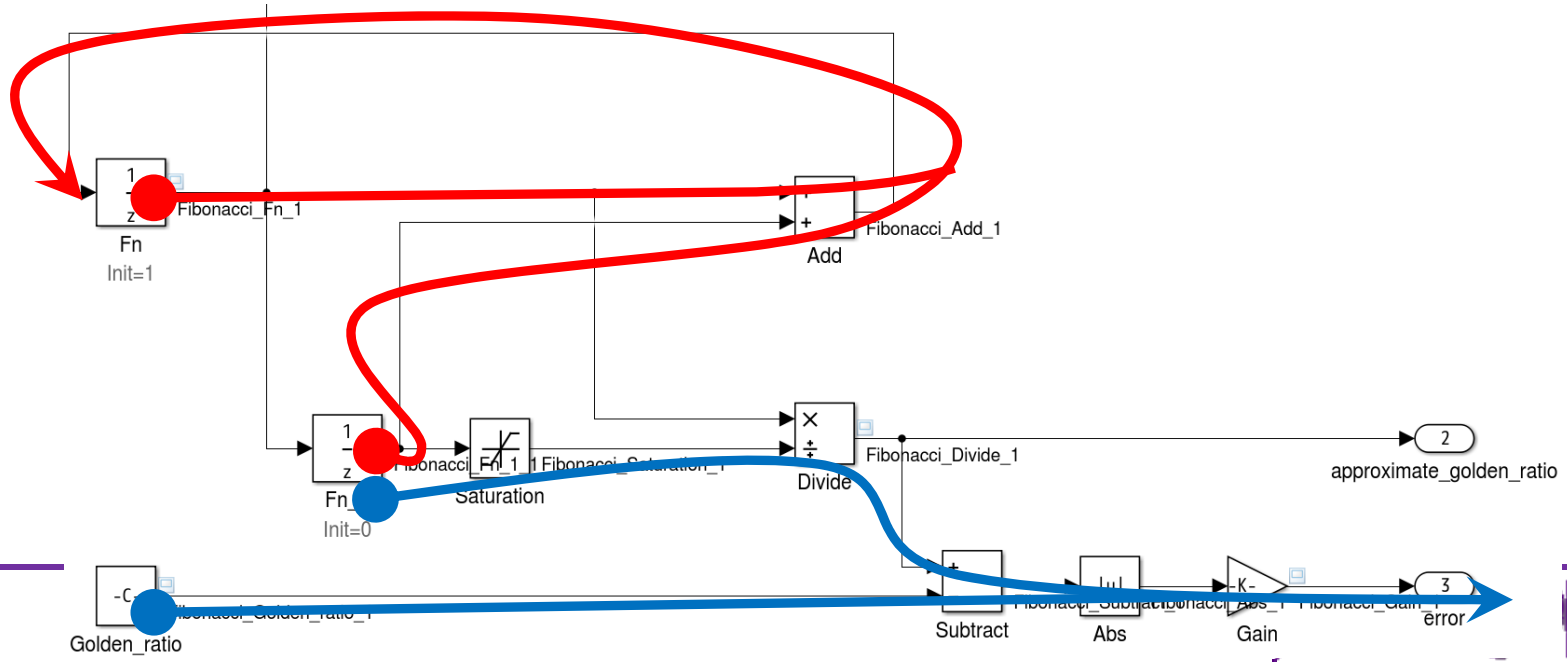
組込み制御のSimulinkモデル

- ブロック線図構造
  - 処理と通信がわかりやすい→並列化が考えやすい
  - 通信は1対1単方向(ただし信号線幅はあり得る)
  - 処理量は様々。最下位層ブロックの処理量は小さい
  - 現状並列度は高くはない→将来に向け、実装での並列性抽出努力、制御設計との協調が重要



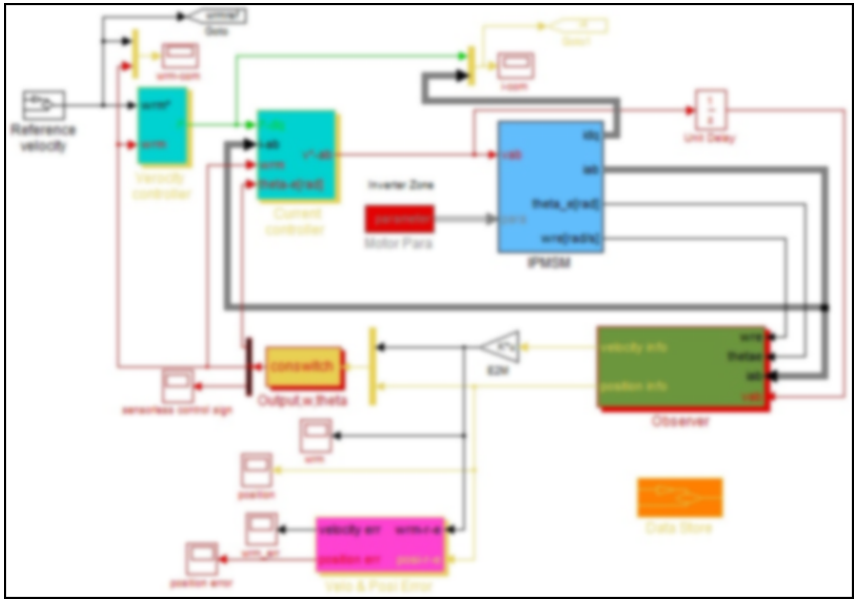
制御特有の課題(1)

- 周期がある
  - 各周期での信号の開始点、終了点がある
    (入力、メモリ、Unit Delay)から(出力、メモリ、Unit Delay)
  - これらが信号のパスとなる
    - 時間的に厳しい所をクリティカルパスという
- フィードバックループがある



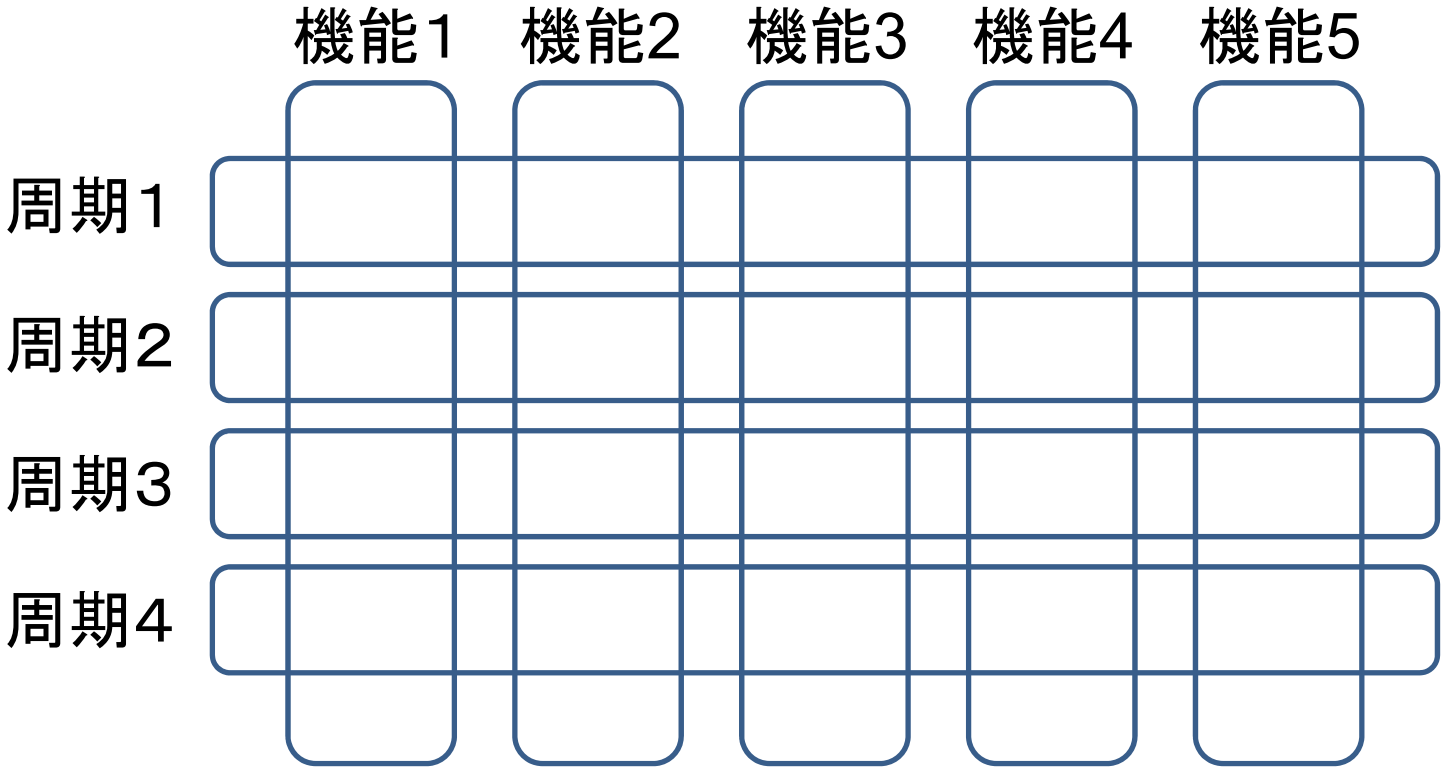
制御特有の課題(2)

- 周期は複数ある
  - 下図において、異なる色は異なる周期
  - 実装上は同じコアに同じ周期をまとめたい
- 制御設計者の意図がある
  - “ATOMIC”サブシステム(「同時」に処理してほしい)
    ⇒並列化において、
    サブシステム内のブロック
    (処理)がばらばらのコアに
    配置され、適当なタイミング
    で処理されたくない
  - 機能単位分割と
    周期単位分割



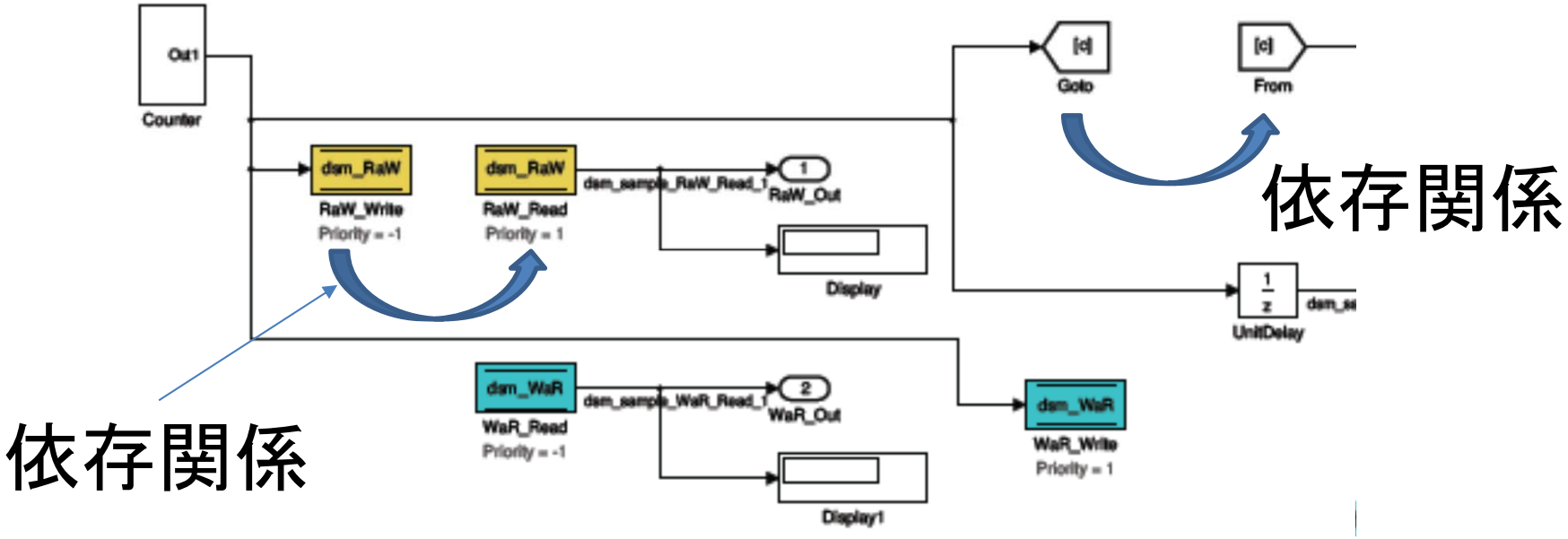
機能分割と周期分割

- 設計の考え方による
- 混在もあり得る



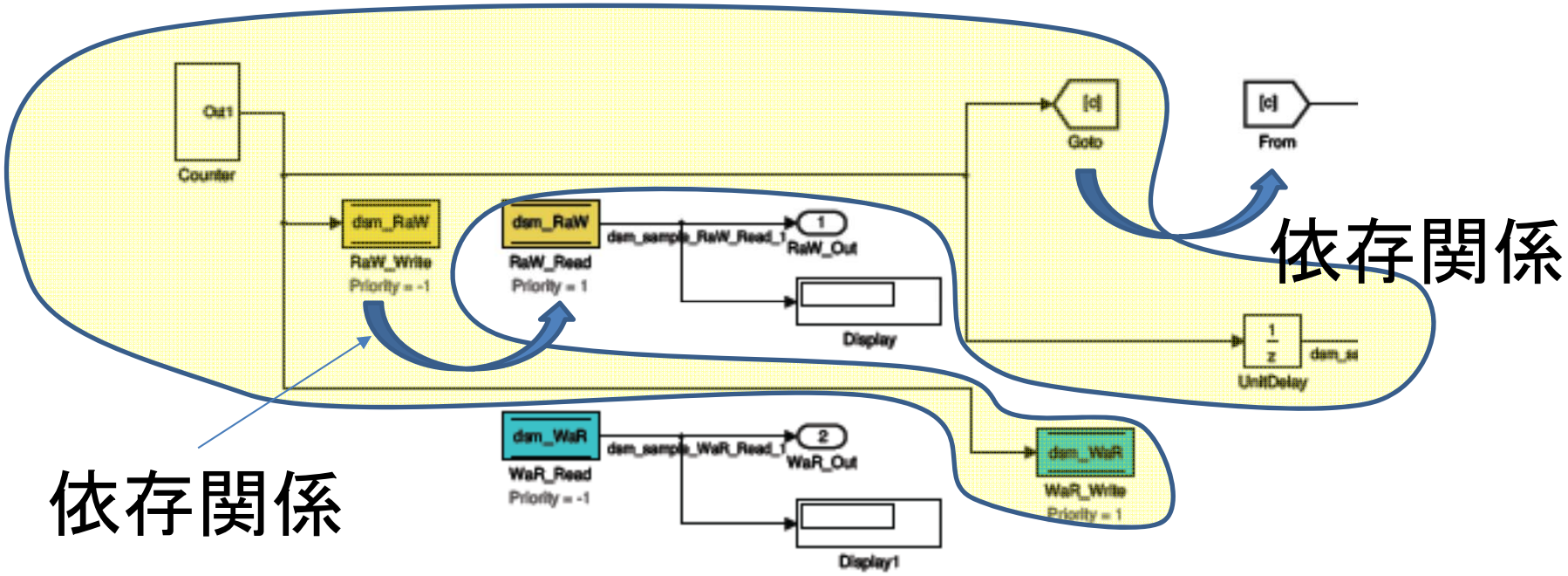
Simulinkの状態、依存関係（1）

- 各周期において計算は、入力または状態から開始し、出力または状態で終了する
- 状態はData Store MemoryやUnit Delayで蓄えられる
- Data Store Memoryは同じ名前の読み書きに依存関係がある
- 状態ではないが、Goto-Fromといった依存関係もある



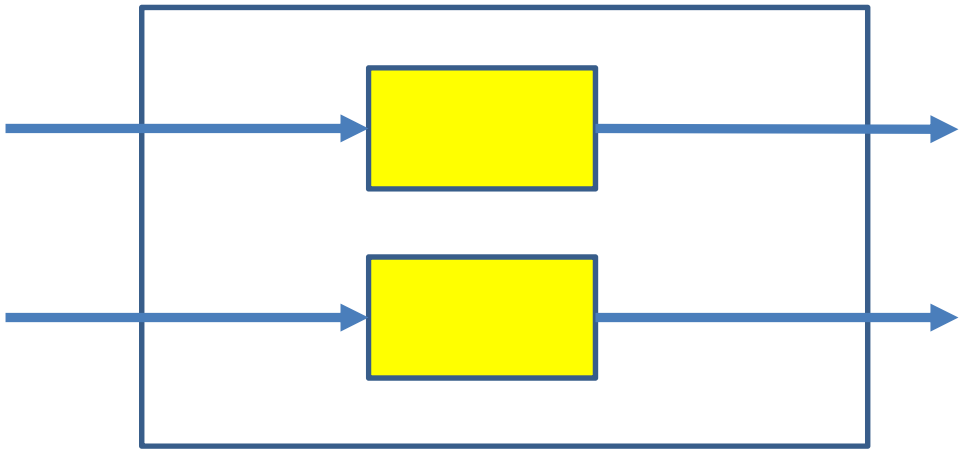
Simulinkの状態、依存関係（2）

- 単純にブロック線図の塊で分割・並列化してもうまくいかない
- また、複数のプロセッサから一つの状態を読み書きすると、不具合原因になりやすいため、一つの状態に対するアクセスは同じプロセッサに固めたい



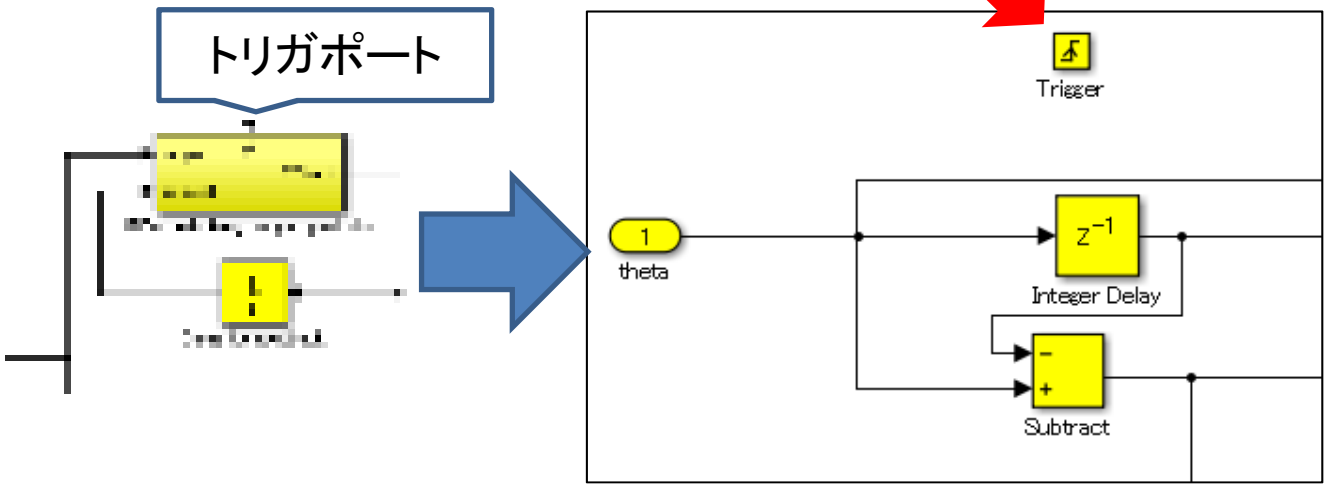
サブシステムについて

- 単に統合しただけのものがある
  - 別々のコアに割り当てても問題ない
- 同時に処理したい場合もある（制御設計者の意図がある）
  - ATOMIC
  - 特殊なポート



特殊ポート(1)

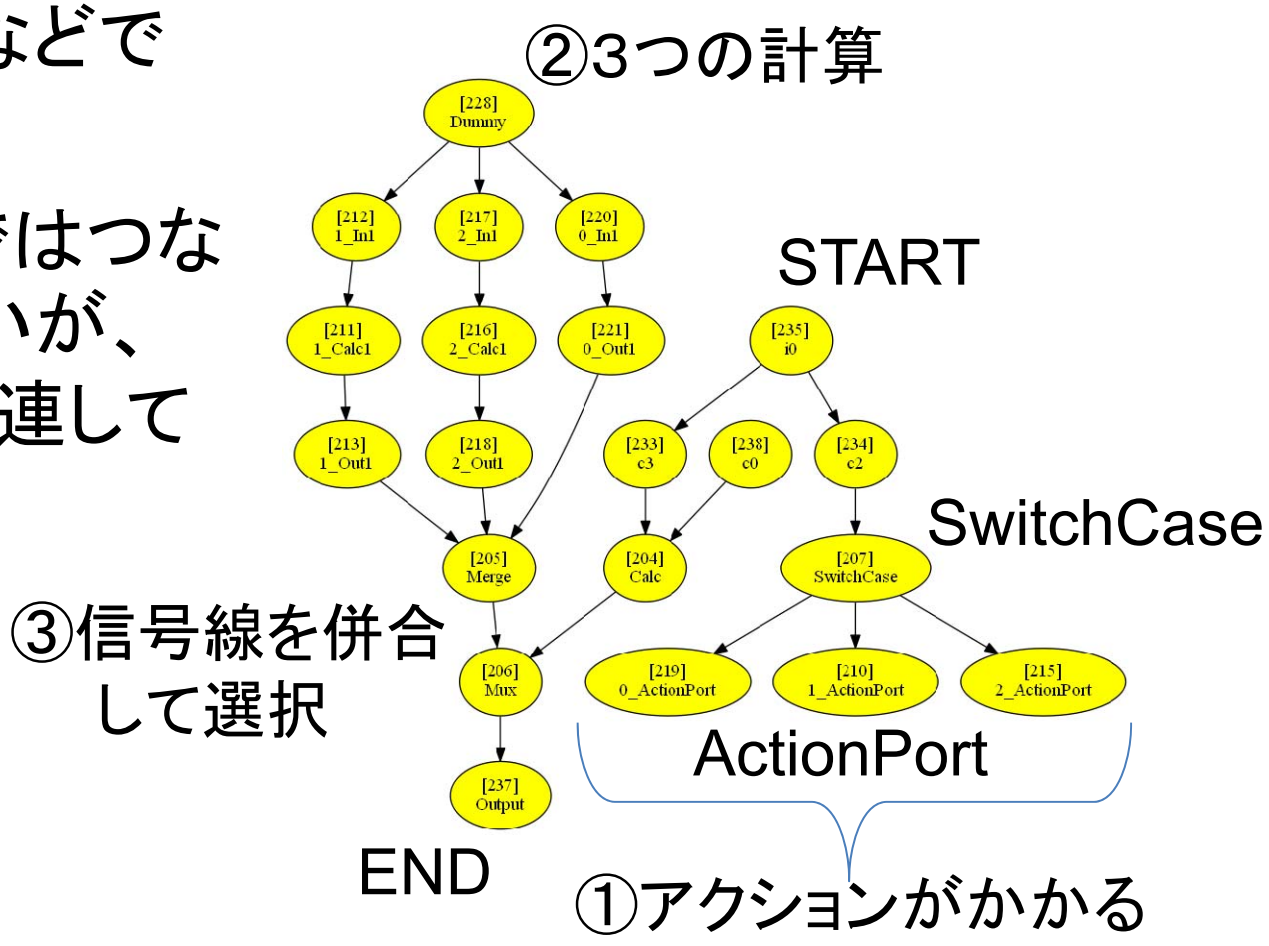
- 特殊なポートがある
  - サブシステムのトリガポートは、サブシステムの内部ではどこにもつながっていない場合がある
  - しかし、トリガがかかると全体が動き出す
  - 並列化で考慮必要



放っておくと、
どのプロセッサに
配置されてしまうか
わからない

特殊ポート(2)

- Action Port
  - SwitchCaseなどで使われる
  - ブロック図ではつながっていないが、実際には関連して動作する



ここまでのまとめ

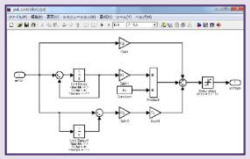
- Simulinkを用いたモデルベース開発では、ブロック線図により設計、処理と1対1単方向通信で考えるため、並列化が考えやすい
  - ただし、並列度は現状高くなく、並列性抽出の努力、制御設計との協調が重要
- 単純にブロック線図を分割すればよいわけではなく、いくつかの留意点がある
  - 制御特有の課題　例：機能と周期
  - 設計者の意図　例：ATOMICサブシステム
  - モデル上の意味　例：Goto-From, 特殊ポート
  - 検証容易性　例：Data Store Memory

アウトライン

- Simulinkモデルベース開発と並列化
- 提案する並列化設計検証フロー
- 並列化実験結果
- 評価版について

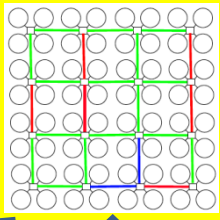
全体の設計検証フロー

コード生成可能
モデル



Simulink

ブロックレベル
実装中間言語/
コード生成/ユー
ザ作成コード



コアへの
ブロック配置

制御モデル
設計ループ

ブロックレベル
構造情報
BLXML&
BL-GRAPH

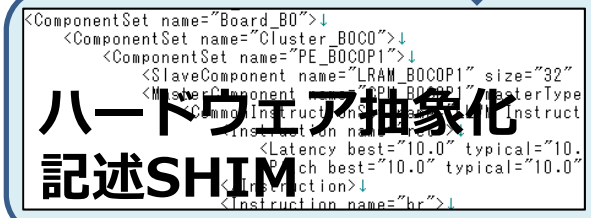
性能等実測

実装最適化
ループ

性能情報等
フィードバック

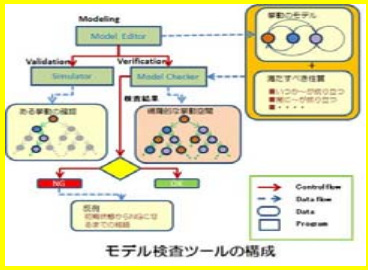
性能見積

並列コード生成



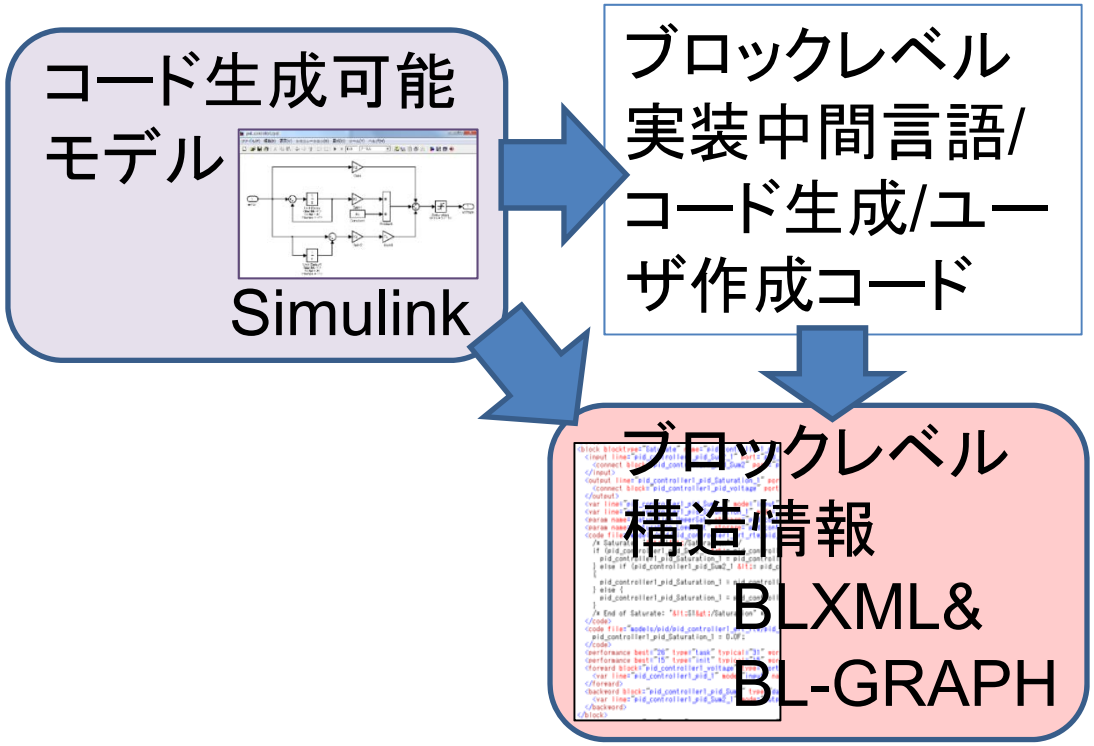
ハードウェア抽象化
記述SHIM

並行モデル
生成



並行動作
振る舞い
検証

ブロックレベル構造入力



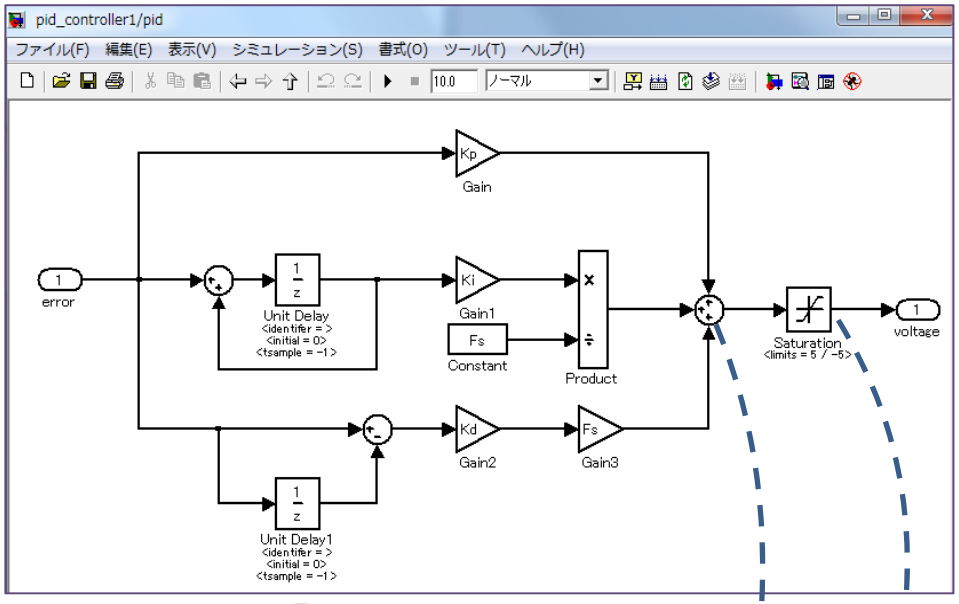
注）名前付けと入力範囲

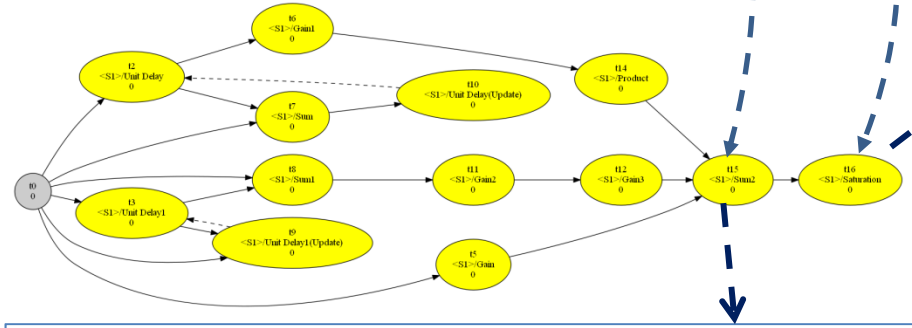
Simulinkモデルと生成コード変数との対応を取るため信号線に名前付け。そのためのツールとして、Qgen、Simulink Library for Java、独自ツールの三通りのフローを開発。

Simulinkは仕様が広いため、正式サポート可能なQgen入力範囲をツールフローの入力範囲としている

Simulinkモデルおよび生成コードを入力し、ブロックレベルの構造情報、設計者指定、周期、ブロックに対応するコードなどの情報を構築

例：Simulinkモデルで記載されたPID制御





```
/* Saturate: '<S1>/Saturation' */
 if (pid_controller1_pid_Sum2_1 >=
     pid_controller1_P.Saturation_UpperSat) {
   pid_controller1_pid_Saturation_1 =
     pid_controller1_P.Saturation_UpperSat;
 } else if (pid_controller1_pid_Sum2_1 <=
     pid_controller1_P.Saturation_LowerSat)
 {
   pid_controller1_pid_Saturation_1 =
     pid_controller1_P.Saturation_LowerSat;
 } else {
   pid_controller1_pid_Saturation_1 =
     pid_controller1_pid_Sum2_1;
 }
```

```
/* Sum: '<S1>/Sum2' */
 pid_controller1_pid_Sum2_1 = (pid_controller1_pid_Gain_1 +
   pid_controller1_pid_Product_1) + pid_controller1_pid_Gain3_1;
```

ブロックレベル処理量情報付記

ブロックレベル
実装中間言語/
コード生成/ユー
ザ作成コード

ブロックレベル
構造情報
BLXML&
BL-GRAPH

性能等実測

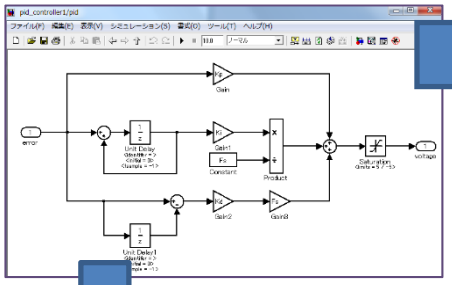
性能見積

ハードウェア抽象化
記述SHIM

ブロック対応コードをSHIMによって
性能見積、もしくは実機等でプロ
ファイリングすることによりブロック
ごとの処理量を付記

Simulinkモデルからの並列構造抽出

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_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_controlle
      pid_controller1_pid_Saturation_1 = pid_controlle
    } 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_ert_rtw/pid_
    pid_controller1_pid_Saturation_1 = 0.0F;
  </code>
  <performance best="26" type="task" typical="31" wors
  <performance best="15" type="init" typical="15" wors
  <forward block="pid_controller1_voltage" type="port"
    <var line="pid_controller1_pid_1" mode="input" na
  </forward>
  <backword block="pid_controller1_pid_Sum2" type="dat
    <var line="pid_controller1_pid_Sum2_1" mode="outpu
  </backword>
</block>
```

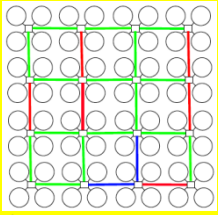
④抽出コード処理量見積

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

SHIM

ブロックレベル構造XML (BLXML)

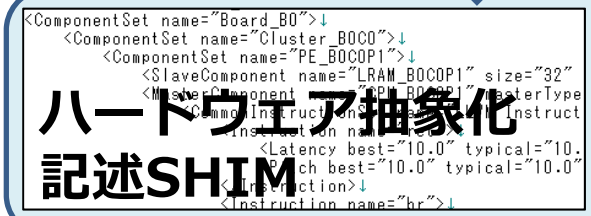
並列化



コアへの
ブロック配置

ブロックレベル
構造情報
BLXML&
BL-GRAPH

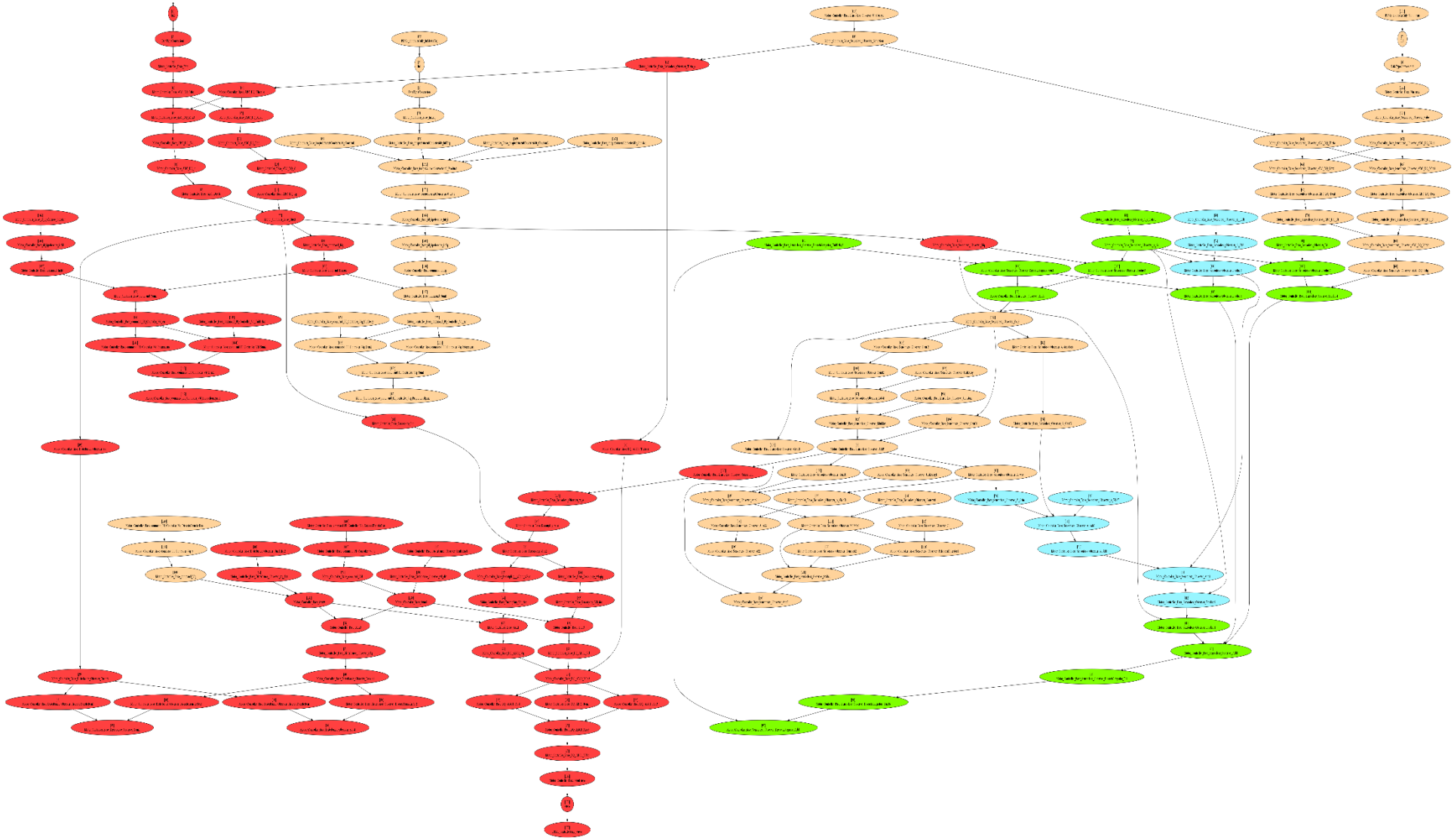
性能見積



ハードウェア抽象化
記述SHIM

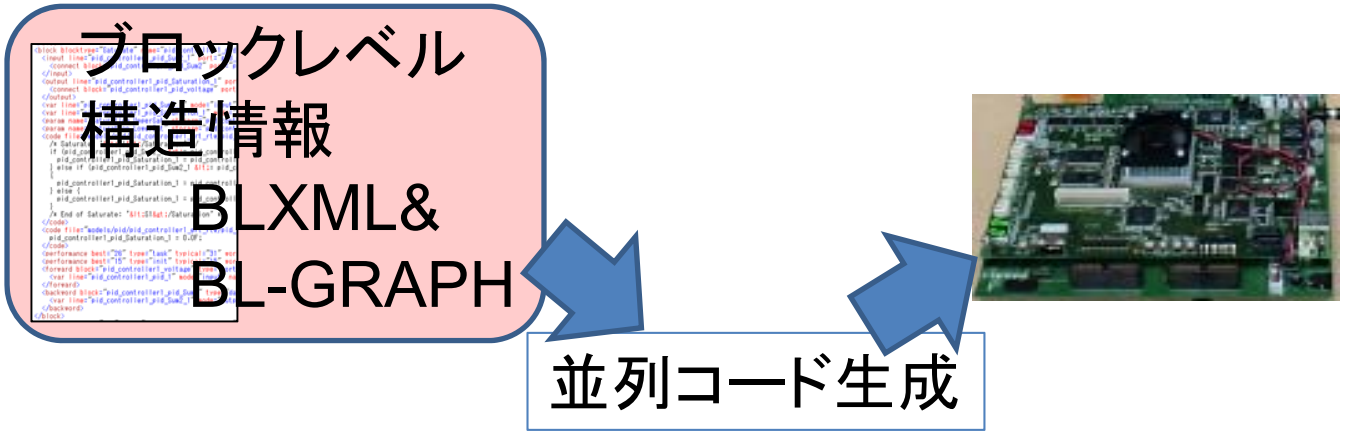
ブロックレベル構造情報、設計者指定、周期、処理量、SHIM記載のプロセッサ間通信性能を利用し、コアへのブロック配置決定

4コア配置の例



並列化コード生成

ブロックのコア配置、ブロックごとの対応コードをもとに並列化コード生成



Cベースではあるが、リターゲッタブル
(現在、pthread, XC (XMOS), SigmaC (KALRAY), eMCOS スレッドに対応）

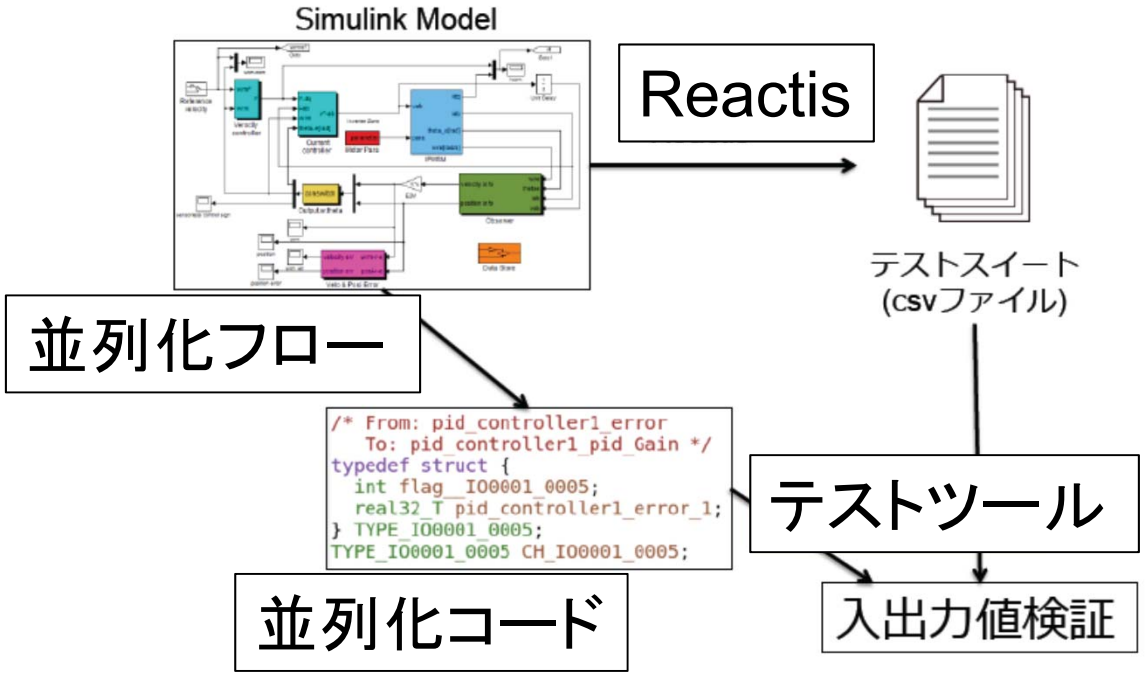
自動コード生成例 for Many Core

```
/* 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
  }

    /* params */
    struct {
      real32_T Saturation_UpperSat;
      real32_T Saturation_LowerSat;
    } pid_controller1_P = {
  5.0F,                                 /* Computed P
                                         * Referenced
                                         */
  -5.0F                                 /* Computed P
                                         * Referenced
                                         */
    };

    /* 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;
  }
}
```

並列化コードの入出力テスト

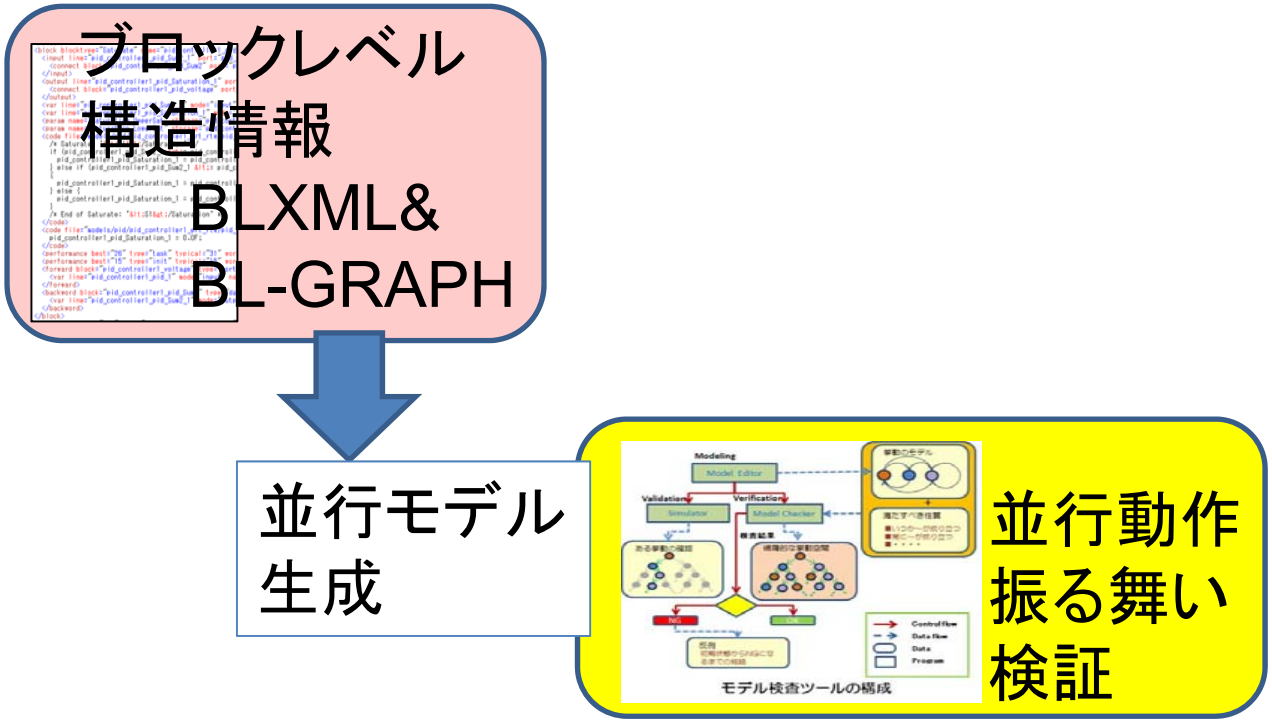
- Simulink対応テストツールReactisを使って自動入出力テストフローを構築
- 基本ブロック(*)に関しテスト中

  (*) Qgenが対応する基本離散ブロック100強種



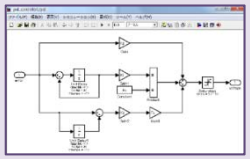
並行動作ふるまい検証

ブロックレベル構造、処理量などから時間付き並行モデル（Timed CSP）を出力し、振る舞いを形式的に検証



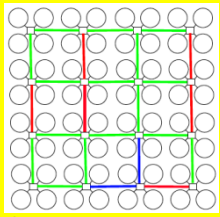
制御モデル設計ループ

コード生成可能
モデル



Simulink

ブロックレベル
実装中間言語/
コード生成/ユー
ザ作成コード



コアへの
ブロック配置

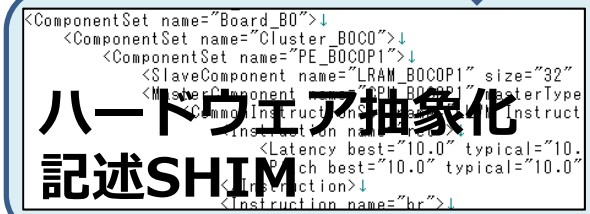
制御モデル
設計ループ

ブロックレベル
構造情報
BLXML&
BL-GRAPH

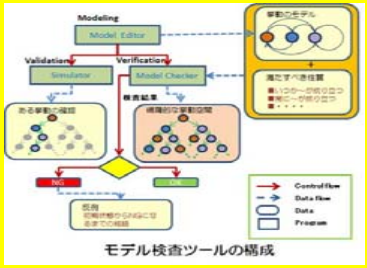
性能情報等
フィードバック

おおよその動作を制御
設計にフィードバック
（注：現在検討中）

性能見積

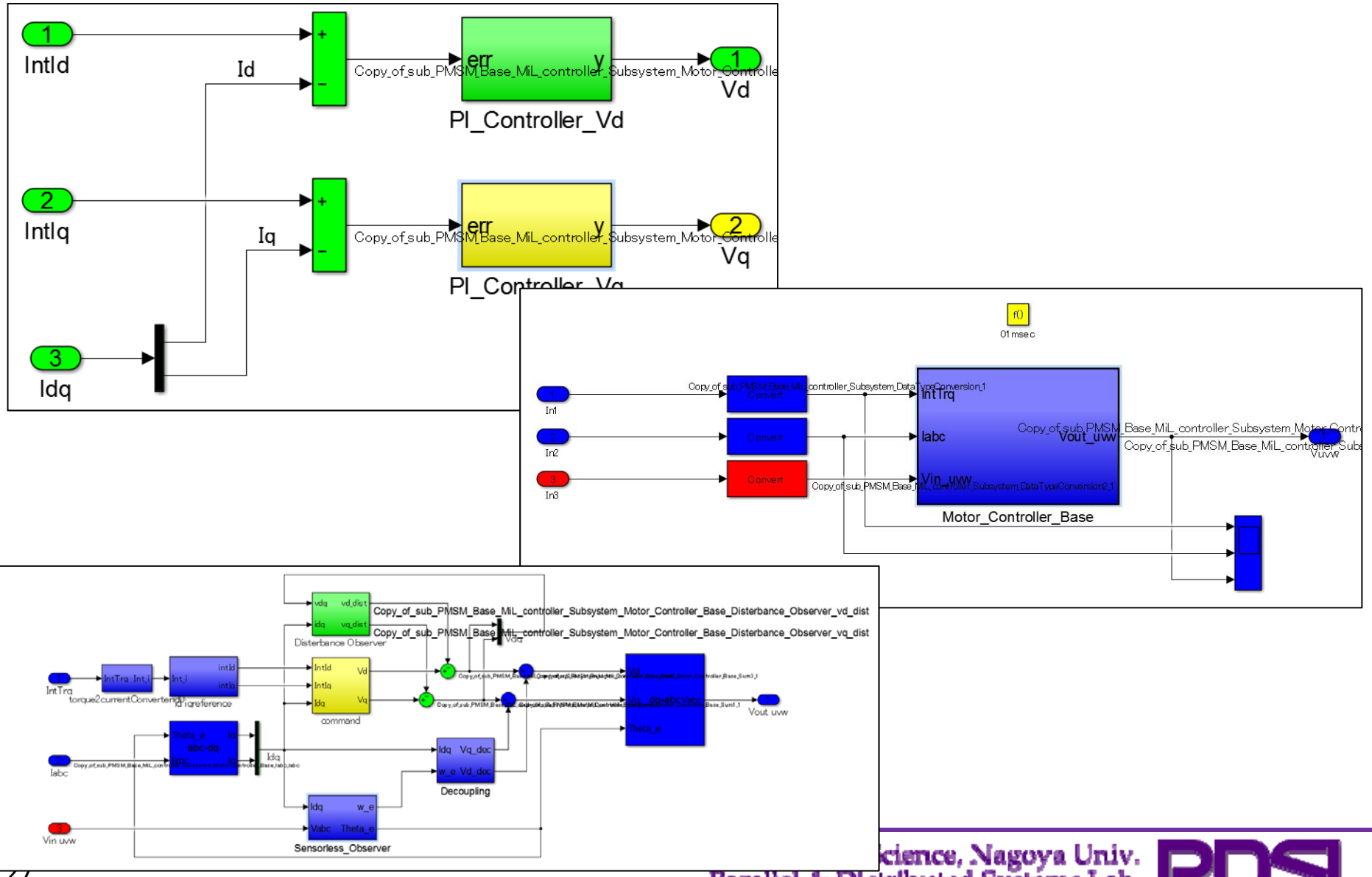


並行モデル
生成

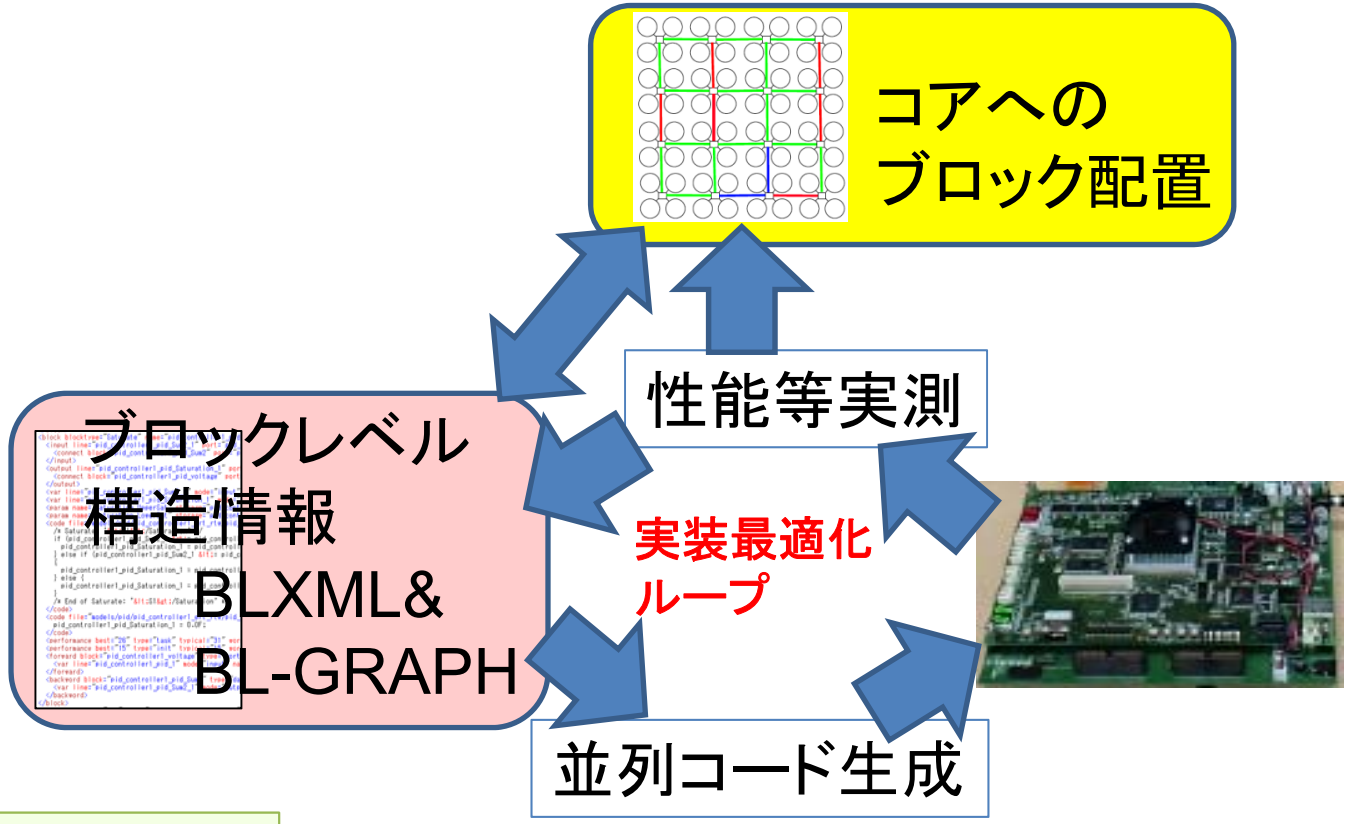


並行動作
振る舞い
検証

コア割当の色分け表示


1
IntId
Id
err
y
Copy_of_sub_PMSM_Base_MiL_controller_Subsystem_Motor_Controlle
1
Vd
PI_Controller_Vd
2
IntIq
Iq
err
y
Copy_of_sub_PMSM_Base_MiL_controller_Subsystem_Motor_Controlle
2
Vq
PI_Controller_Vq
3
Idq
f()
01 msec
In1
In2
In3
Convert
Convert
Convert
IntTrq
Iabc
Vin_uvw
Vout_uvw
Motor_Controller_Base
Copy_of_sub_PMSM_Base_MiL_controller_Subsystem_Motor_Controller_Base_Disterbance_Observer_vd_dist
Copy_of_sub_PMSM_Base_MiL_controller_Subsystem_Motor_Controller_Base_Disterbance_Observer_vq_dist
Disterbance Observer
IntTrq
torque2currentConverter
Id,Iqreference
command
Iabc
abc-dq
Idq
Decoupling
Vin_uvw
Sensorless_Observer
Vout_uvw

実装最適化フロー



実装レベルでコア割当を
見直しながらチューニング
（注：現在検討中）

アウトライン

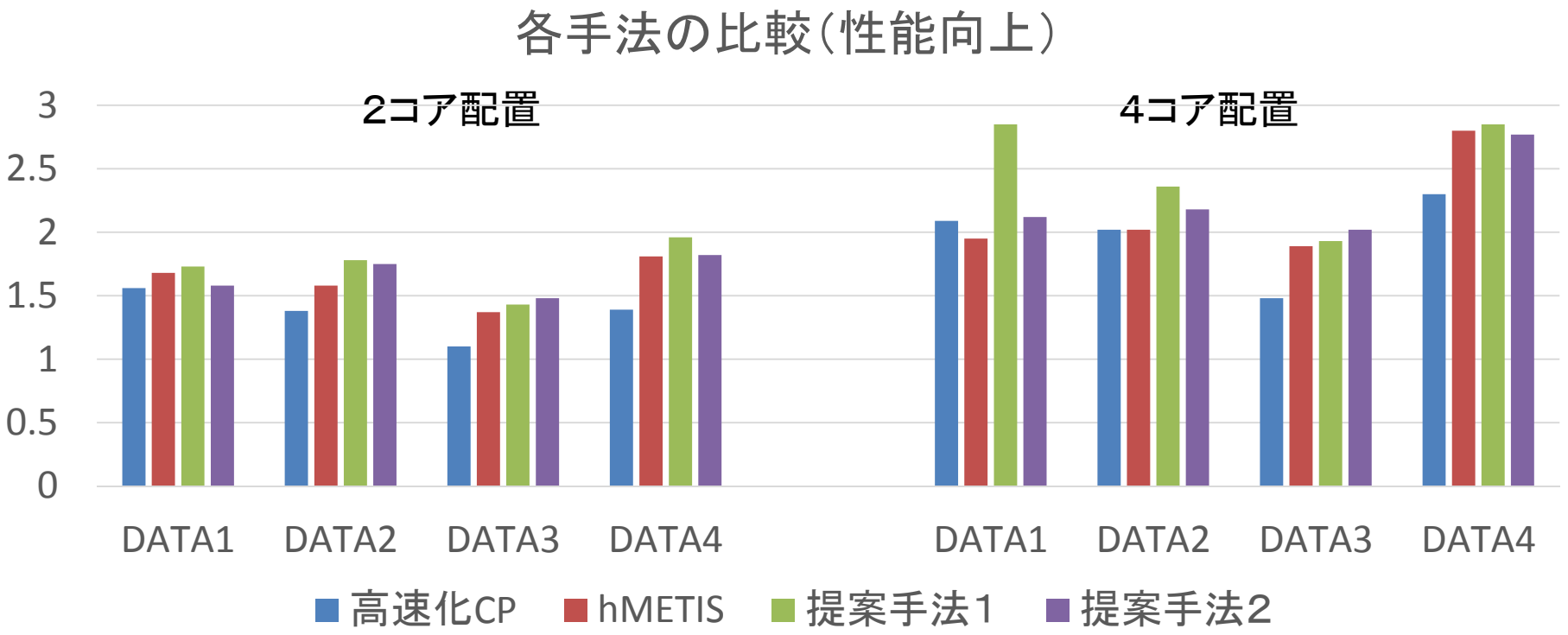
- Simulinkモデルベース開発と並列化
- 提案する並列化設計検証フロー
- 並列化実験結果
- 評価版について

実験結果（1）

- ４種のデータ
  - DATA1 (143ブロック)
  - DATA2 (502ブロック)
  - DATA3 (2158ブロック)
  - DATA4 (177ブロック)
- ２コアと4コアで実験
- ４種のアルゴリズム
  - 高速化クリティカルパス法
  - 階層クラスタリング法 (hMETIS)
  - 提案手法１（クリティカルパス法＋階層クラスタリング法の融合）
  - 提案手法２（提案手法１＋設計制約考慮）

実験結果(2)

- 提案手法により10～30%性能向上、設計制約を考慮すると、多くの場合少し性能が落ちる
- 計算時間は、すべて1秒未満



アウトライン

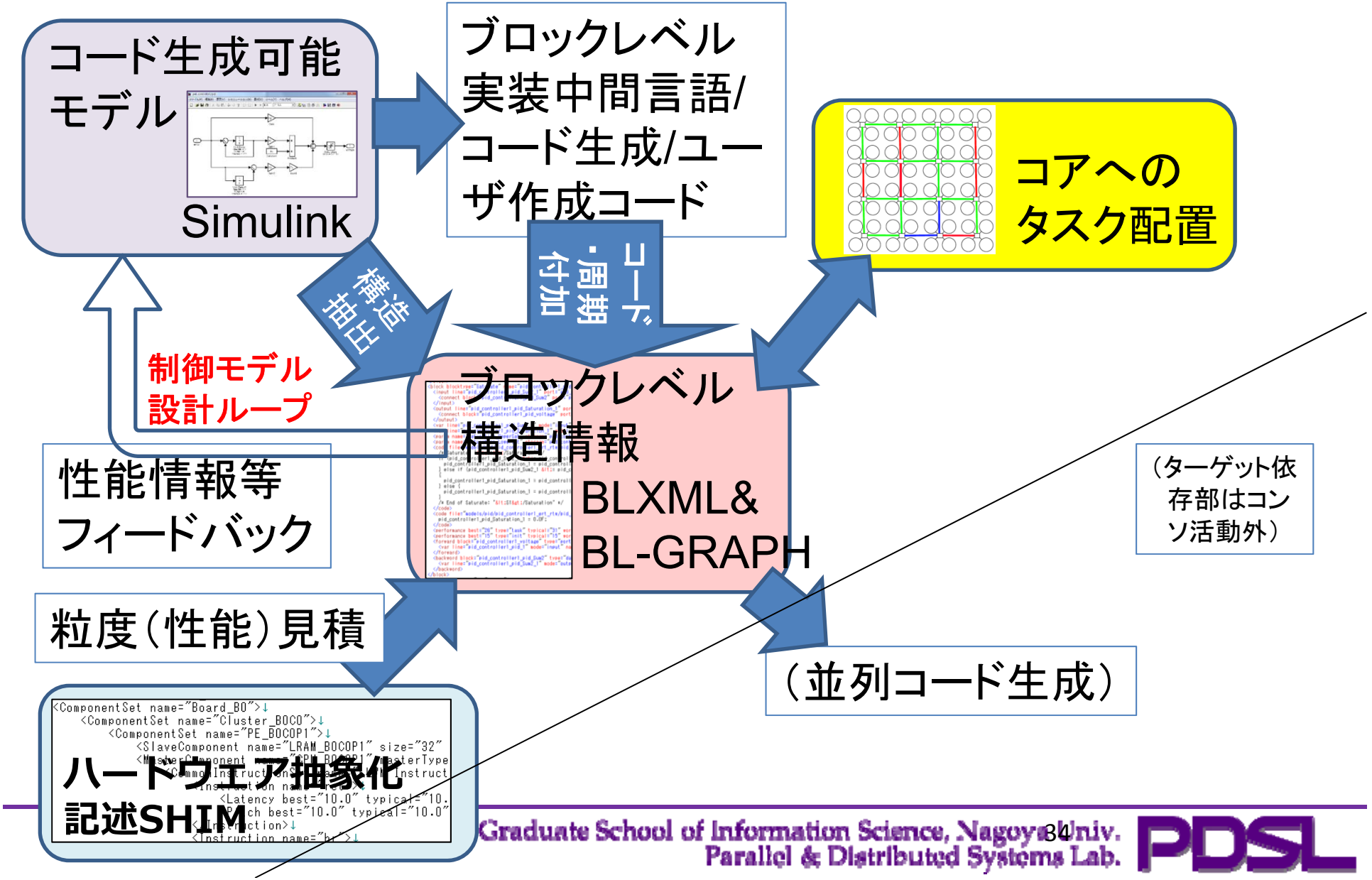
- Simulinkモデルベース開発と並列化
- 提案する並列化設計検証フロー
- 並列化実験結果
- 評価版について
  - 組込みマルチコアコンソーシアム内にて評価版をリリース

評価版バージョン

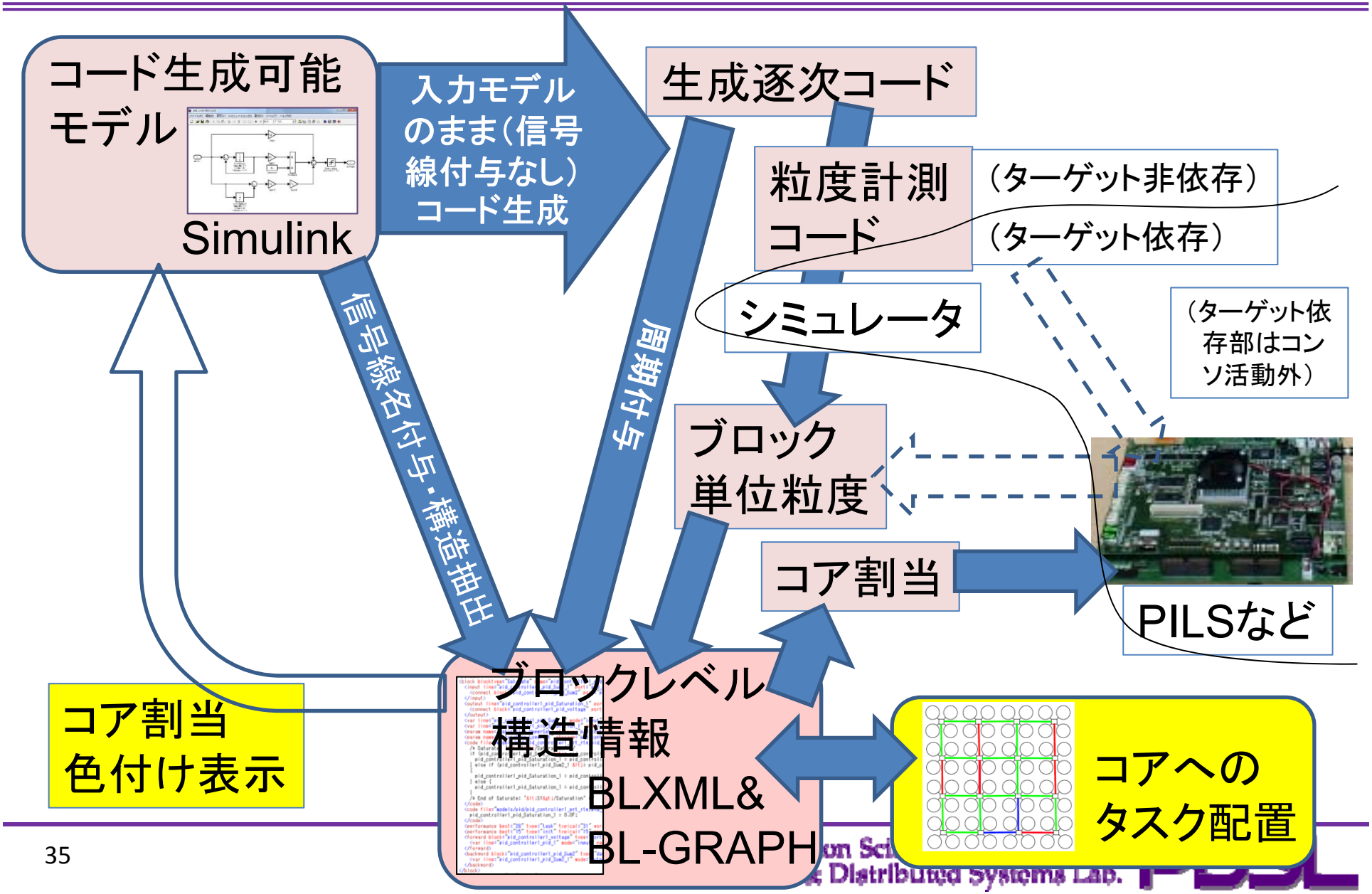
- バージョンは二種類
  - 主バージョン
    - 全体フローからターゲット依存部を除いたもの
  - ブロック割当バージョン
    - 並列化コード生成を想定せず、ブロックのコア割当のみを行う。ターゲット依存部は除く

- 形態
  - MATLABスクリプト、VM上のバイナリ、スクリプト

主バージョン


コード生成可能
モデル
Simulink
ブロックレベル
実装中間言語/
コード生成/ユー
ザ作成コード
コア への
タスク配置
構造抽出
コード・周期付加
制御モデル
設計ループ
ブロックレベル
構造情報
BLXML&
BL-GRAPH
性能情報等
フィードバック
粒度（性能）見積
（ターゲット依存部はコンソ活動外）
（並列コード生成）
ハードウェア抽象化
記述SHIM

ブロック割当バージョン



残課題

- BLXML生成・コード生成
  - 対応外のSimulink仕様（パラメタ）にあたると名前付け～
    コード生成関連でフローが止まる場合がある
- 性能見積
  - SHIMからの性能見積精度が悪い場合がある
    - 特殊演算（sin, cosなど）
    - メモリ周り（キャッシュ、連続/ランダムアクセス）

- 研究プロトタイプである

まとめ

- Simulinkからの並列化手法
  - Simulinkを用いたモデルベース開発の特徴
  - 制御設計の特徴
  - 提案する設計検証フロー
  - 並列化アルゴリズム
  - 評価版