モデルベース並列化ツールの実用化

Embedded-Multi-Core-Summit 2017-West @ OSAKA

eMBP

2017-07-13

Hiroshi Fujimoto

Expert, eSOL Co.,LTD

Abstract

- マルチ・メニーコアチップを導入し、それを有効に利用するソフトウェアを開発するためには、OSなどのプラットフォームに加え、その上で動作するアプリケーションの 並列化支援ツール も重要な要素となる。
- 本講演ではeSOLが名古屋大枝廣研究室と共同開発しているMBP（Model Based Parallelization）技術を実用化したeMBPの特徴と提供する機能、利用シナリオ、その効果について今後の展望を交えながら紹介する。

Agenda

- マルチコアソフトウェア支援環境
- eMBP
- 利用シナリオ
- 効果
- 技術説明
- 今後の展望
- 参考

モデルベース並列化ツールの背景

■マルチコアソフトウェア開発環境

マルチコアは既に定着している

- 既にPC,SmartPhoneでは2～4コアが普通
  - クロックを上げずに平均性能を上げるため。
  - ソフトウェアを並列化しないと性能がでない。
- 「メニーコア」へ
  - 16-256,512コア！
  - KALRAY, TILERA, Intel, Adaptiva
- 組み込みシステムでも無視できない
  - スパコン並みの処理量を要求する認識・学習する組み込み機器

問題・ソフトウェア技術

- マルチ・メニーコアを使いこなすソフトウェア技術が必要だが、並列ソフトウェアの設計・検証技術はまだ特殊領域

  → 職人芸、ソリューション、コンサルビジネス

  - 例）
    - 並列プログラミング技術
    - 並列システム検証技術
    - GPGPUプログラミング
- アーキテクチャ依存でかつソフトとハードを切り離し辛い

開発支援技術

- 実行環境技術：
  - マルチ・メニーコアOS
    - スケーラビリティ
    - コア・マイグレーション (動的なタスクーコア割り当て変更）等
- デバッグ環境技術：
  - シミュレータ(HILS, SILS, MILS, PILS)
- 並列化支援技術：
  - Parallelizer
- 検証技術（静的/動的)
  - デッドロック検出、データ破壊可能性検出等
  - 静的コード検証技術、形式検証技術
- 並列設計/プログラミング技術
  - 並列プログラミング言語、並列モデル設計技術

Parallelizerの基本構成

■並列化構造抽出

- 普通に書かれた（並列化されていない）プログラムコードから、データの依存関係（変数代入）解析
  - →データフローグラフ　→　タスクグラフ
- 並列構造を表現する何等かの言語表現を作成

■コア割り当て

- 有限数（現状固定）の計算資源（コア）に同時実行可能な処理単位（タスク/スレッド）を割り当て
  - →組み合わせ問題

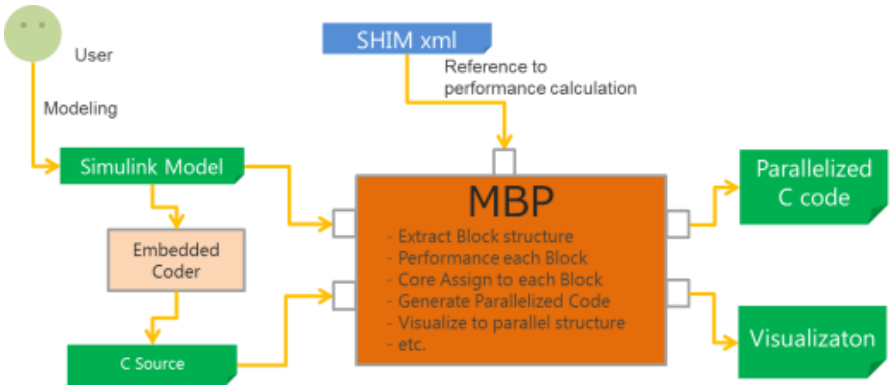
Some Parallelizer's

- OSCAR 自動並列化コンパイラ(Waseda Univ.)
  - 研究・実用化、ソリューション
  - OSCAR APIによる並列構造表現　(OpenMP Like)
- SILEXICA (SLEXICA社)
  - SILEXICA/Parallelizer 製品
  - KPN(Kahn Process Network)を基礎とする並列構造表現
- モデルベース並列化(Nagoya Univ. & eSOL Co.,LTD)
  - 研究・実用化→eSOL(株)による製品化(eMBP)
  - BLXMLによるブロック構造表現
  - CSP(Communicating Sequential Process)として解釈

eMBP ソフトウェア紹介

■ eMBP

eMBP



MBP = Model Based Parallelizer

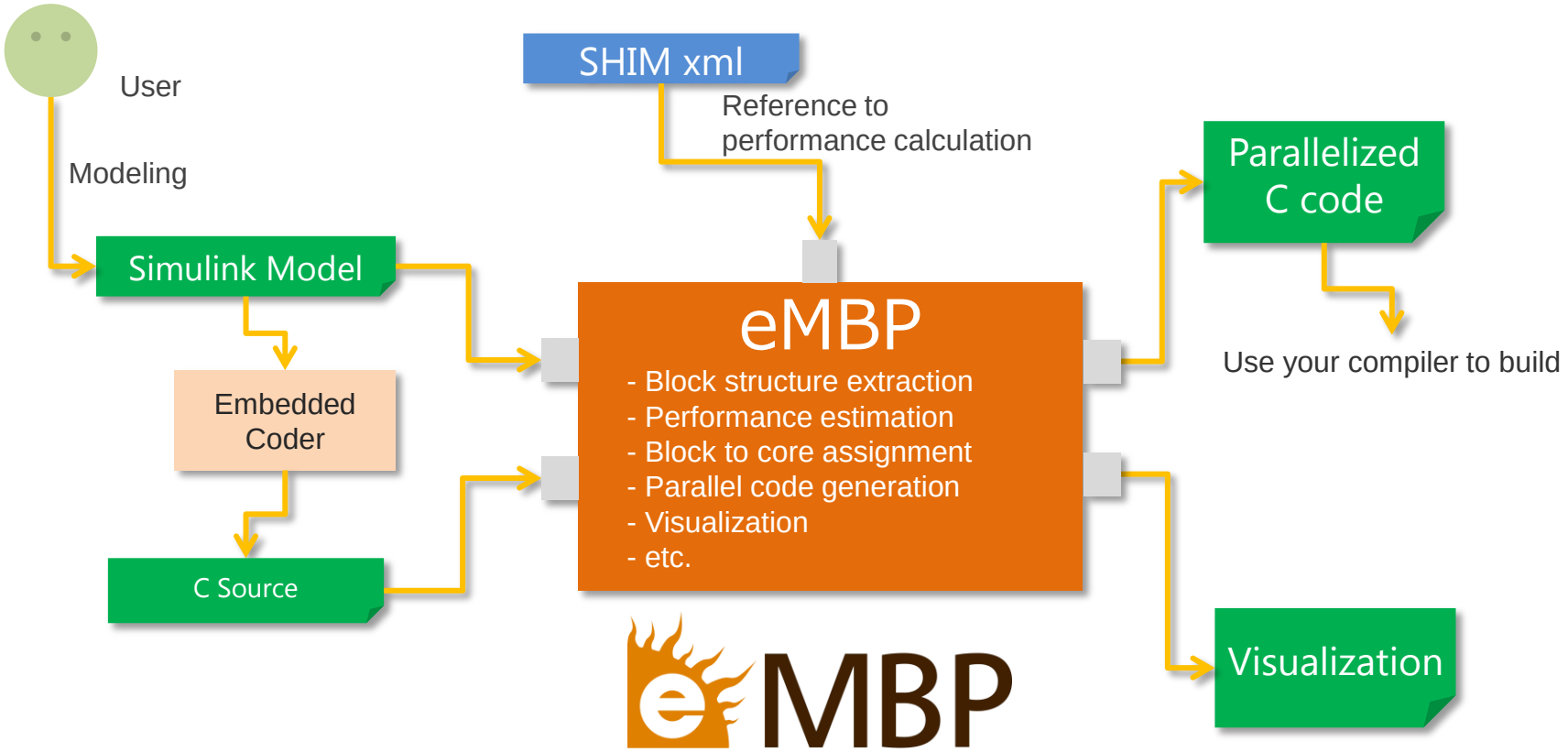
機能と特徴

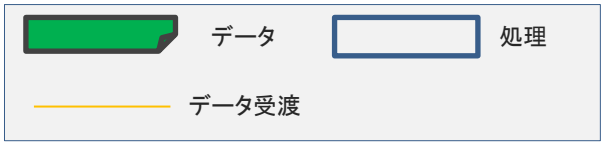
- Simulinkで設計された制御モデルから生成されるCソースコードを並列化
- Simulinkのブロック線図に対応した並列化を行うため、設計者の意図が反映される。
- ブロック毎の実行性能の見積りに標準化されたハードウェア構造記述SHIM (※1)を利用。
- コア割り当ては、Min-Cut戦略によるロードバランシング、クリティカルパスを考慮した「階層クラスタリング」アルゴリズムを採用
- PILS(※2)システムと連携し、モデルベース開発による動作検証を支援。

※1 Software-Hardware Interface for MULTI-MANY core

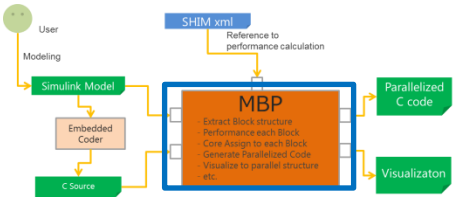
※2 Processor In the Loop Simulation

ツール構成 (外部)

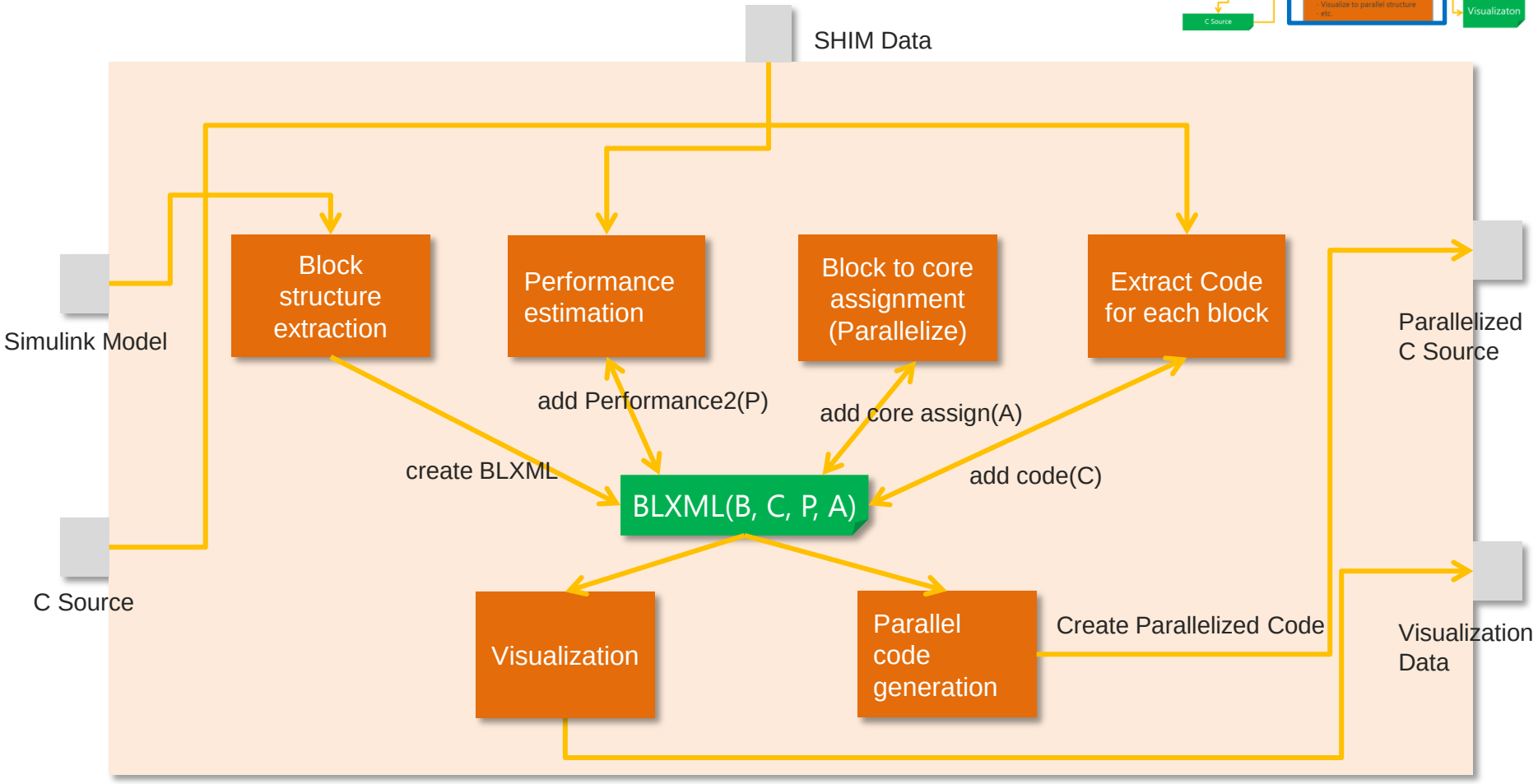

User
Modeling
SHIM xml
Reference to
performance calculation
Simulink Model
Embedded
Coder
C Source
eMBP
- Block structure extraction
- Performance estimation
- Block to core assignment
- Parallel code generation
- Visualization
- etc.
eMBP
Parallelized
C code
Use your compiler to build
Visualization


データ
処理
データ受渡

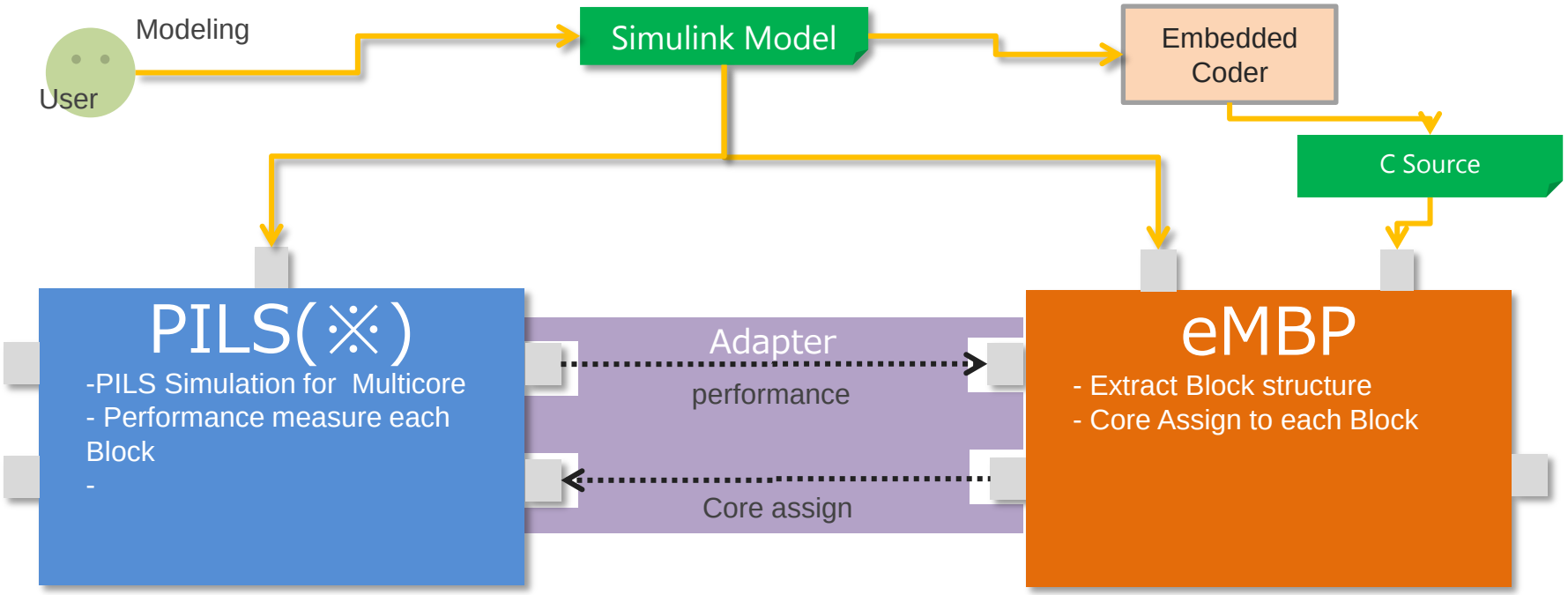
ツール構成（内部）



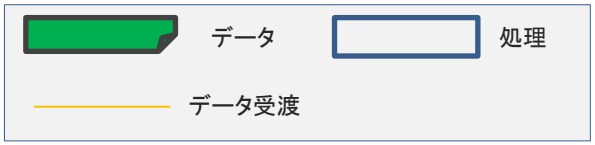
各機能が共通のブロック構造フォーマットデータを参照・追記して連携



PILS連携 システム構成



※Processor In the Loop Simulation



処理機構上の特徴

- C言語ソースのみからの並列化に比べ、データフロー解析が簡略化される
  - ブロック図は既にデータフローを表現している
  - 並列構造を設計者がコントロール可能
- 無駄な同期・通信のコードが組み込まれない
  - コアをまたいで割り当てられたブロック同士の情報受け渡しのみその様なコードが生成される
  - それ以外は元のまま
- 同期・排他が問題になる様な部分は同じコアへ
- サブシステムによる階層的モデルに対応

eMBPとしての利用シナリオ＋PILS連携時のシナリオ

■利用シナリオ

コードを並列化し、実行(eMBP)

- [User] Simulinkにより制御設計を行いシミュレーション実行によりモデルの妥当性を確認
- [User] Embedded Coderによって対象モデルからコード生成
- [eMBP] 制御設計モデルと生成コードから並列化されたコードを得る
- [User] 実行プラットフォーム用にコンパイルし、実行モジュールを得る
- [User] 実行プラットフォームに配置(OSコンフィギュレーション設定)
- [User] 実行検証・導入

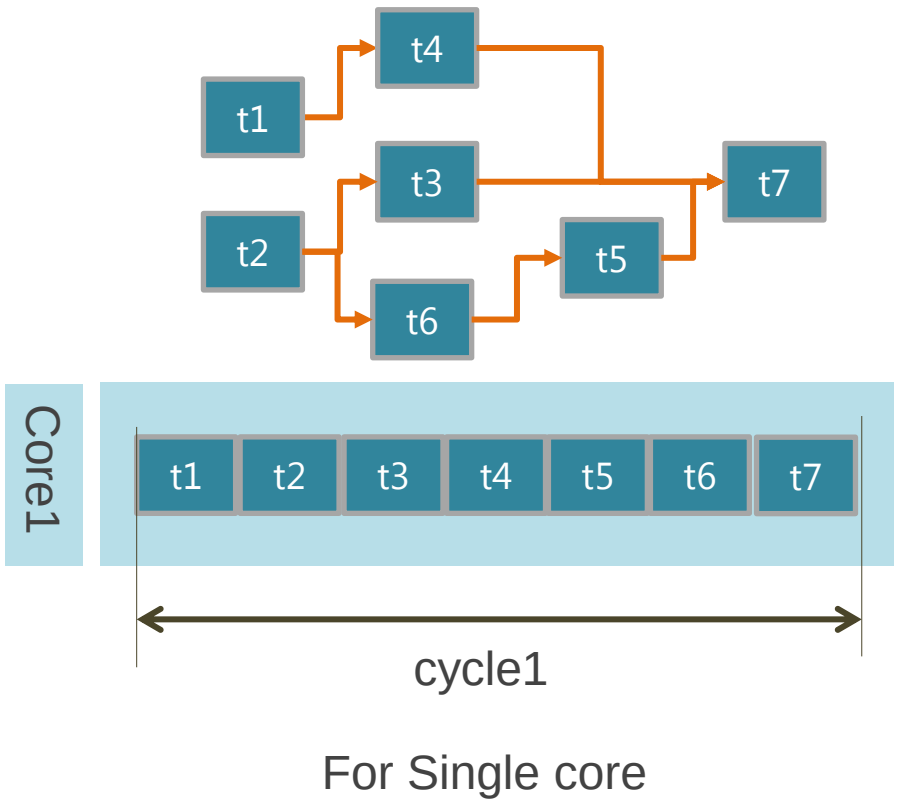
PILSシステムからコア割り当計算依頼(PILS連携)

- [PILS User]Simulinkにより制御設計を行いシミュレーション実行によりモデルの妥当性を確認
- [PILS User]PILS用モデルに変形し１コア割り当てでシミュレーションし、対象サブシステム毎の処理量計測
- [PILS User] 計測した対象サブシステム毎の処理量をeMBPに渡し、コア割り当て計算を依頼
- [eMBP]対象サブシステム毎の処理量をeMBPに渡しコア割り当て計算
- [eMBP]コア割り当て情報をPILSシステムに返す
- [PILS User]コア割り当て情報を元にマルチコア割り当てのPILS用モデルに変形し、シミュレーション
- [PILS User]PILSシミュレーションによる実行検証

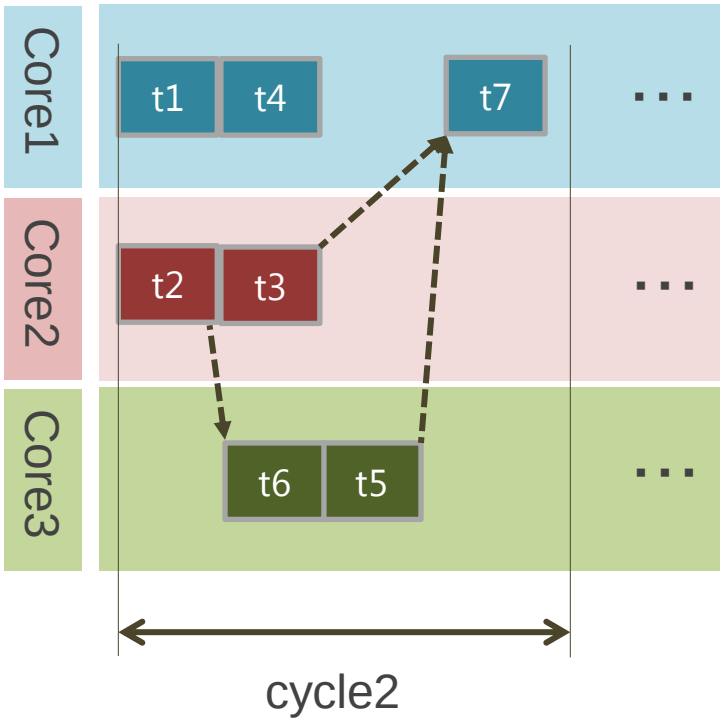
■効果

マルチコア有効利用

- eMBPによってコアを有効利用できるタスクの割り当てを行う事で、トータルの計算時間(周期)の短縮が期待できる。



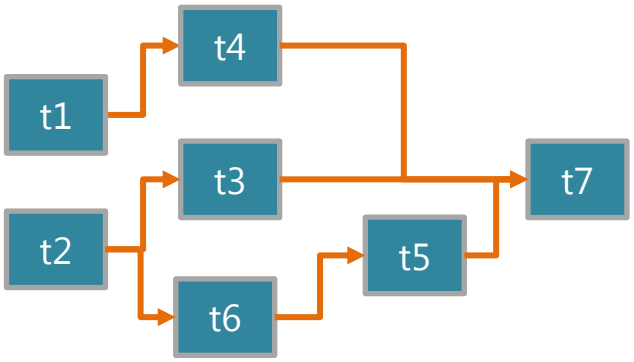
For Single core

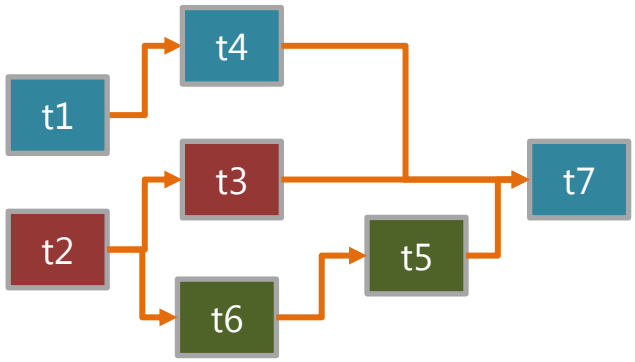


For Multi-core

制御設計の構造≒並列構造

制御設計モデルをベースにしているため(※)生成されたコードの並列構造が理解しやすい。





※ 設計者が並列構造をコントロールできる

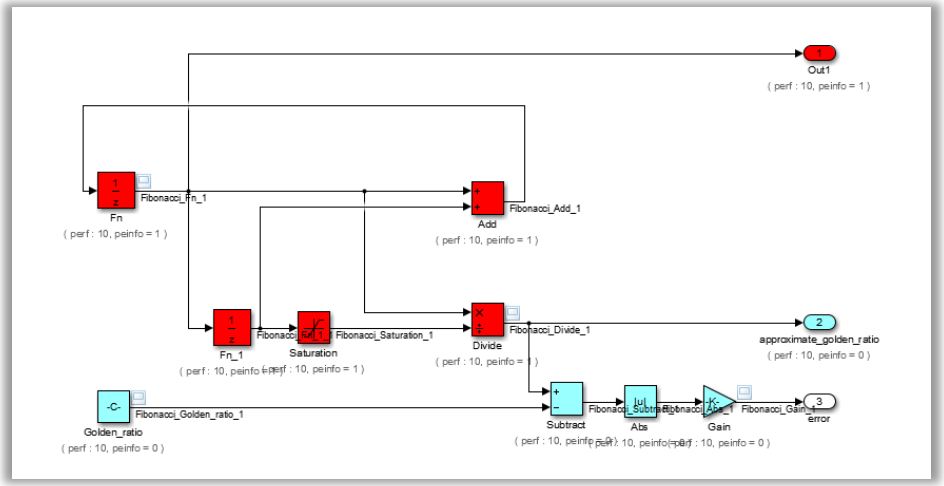
接続関係、ATOMIC属性指定など。

→ 「並列モデル設計指針」につながる

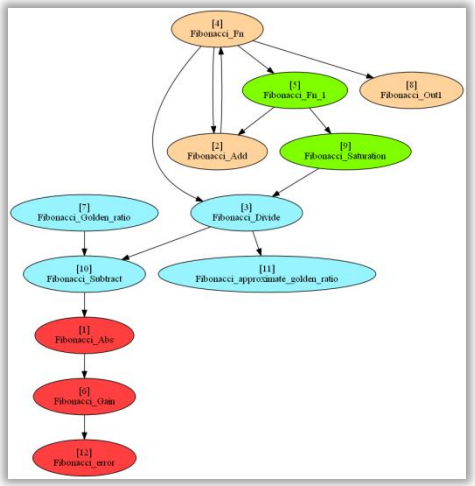
可視化機能による結果の把握

処理ブロック毎のコア割り当て結果を、直観的に把握し、定性的な評価が可能になる。

Simulinkブロックの色属性による可視化

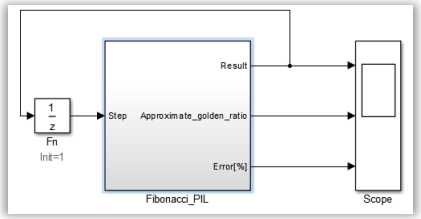


グラフ記述データによる可視化(Graphviz)

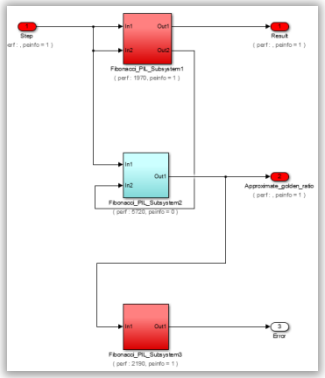


階層構造の全体把握

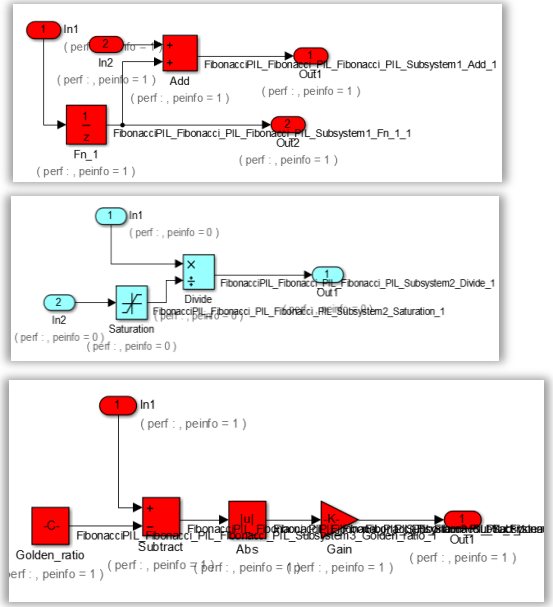
SubSystemによる階層構造を利用したモデルではブロックレベル（最下層）のブロック全体の割り当て状況が把握しずらい→展開図での可視化が効果的



Top Level



Second Level

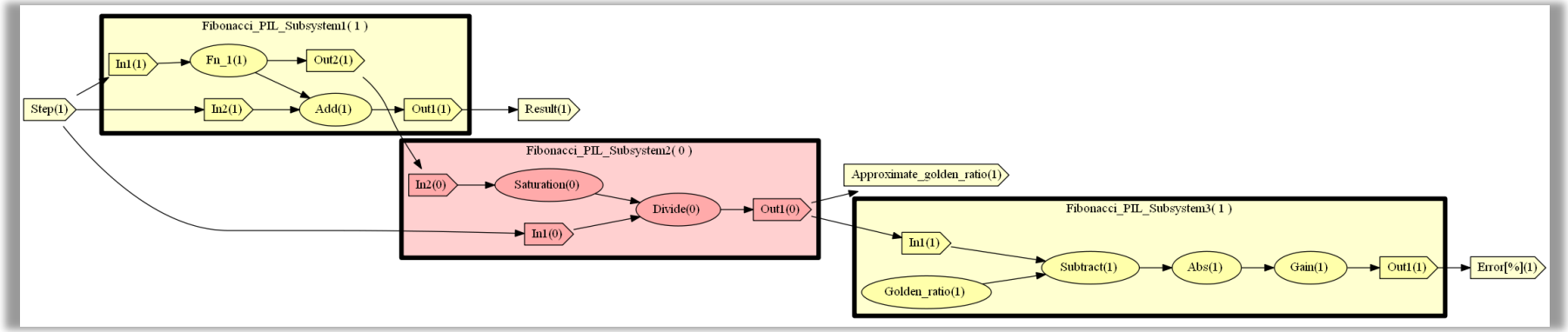


Third Level

全階層を一度に表示

２階層、ブロック数30弱

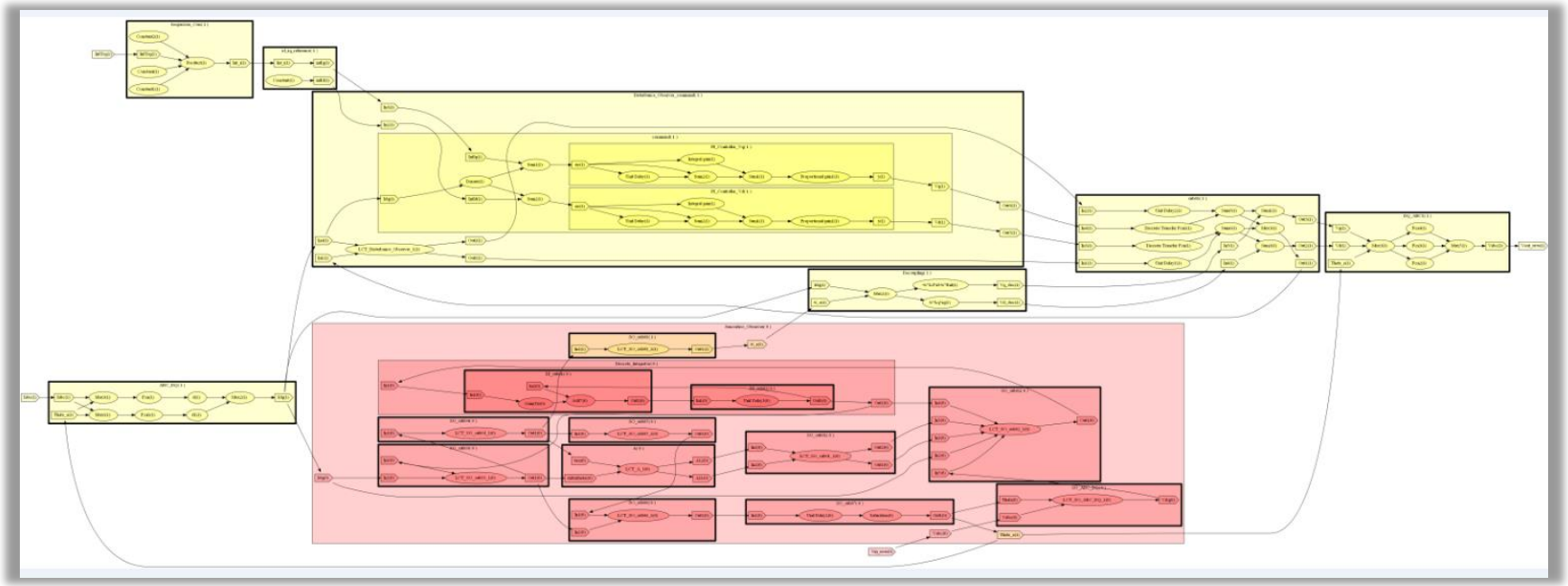
※SubSystemは矩形で表示



補足：PILS連携用のモデルでは、通常のサブシステムと並列化対象のサブシステムを区別する表現が必要。

全階層を一度に表示(2)

4階層・ブロック数200弱

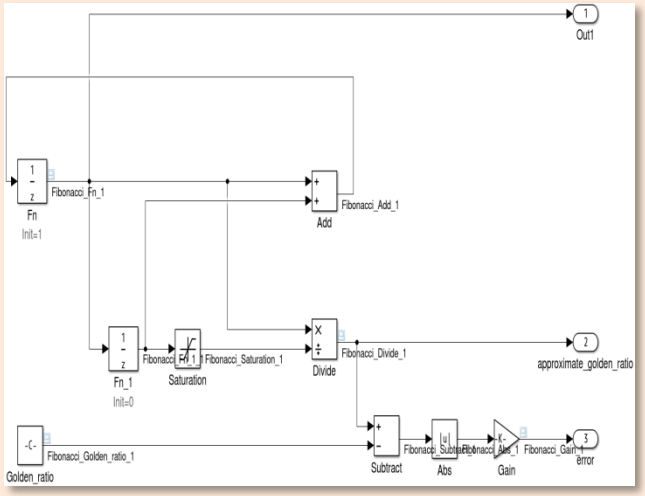


並列化対象のサブシステムが太線

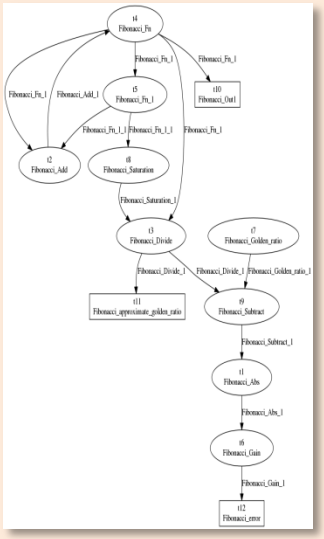
■ 技術説明

Parallelization / eMBP

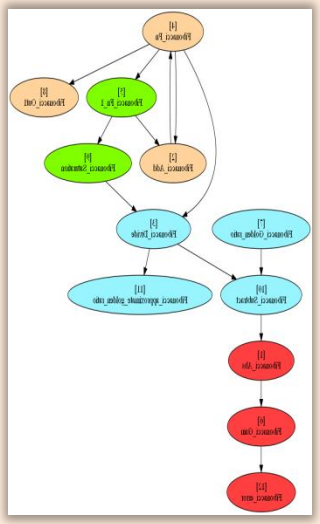
- 並列化 = 並行性抽出 + コア割り当て



制御モデル



DFG

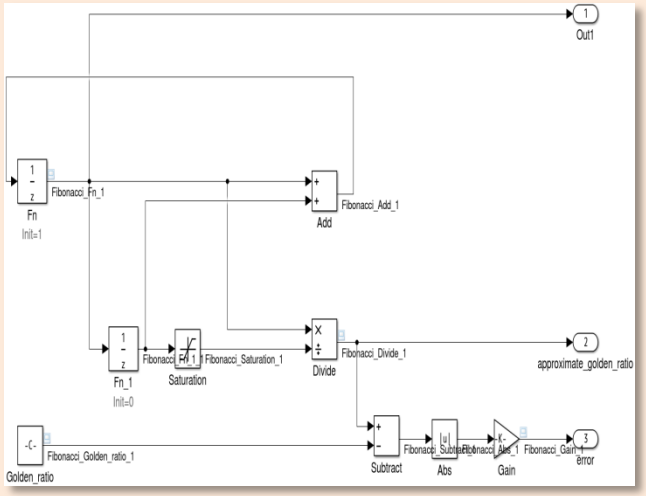


Core Assigned

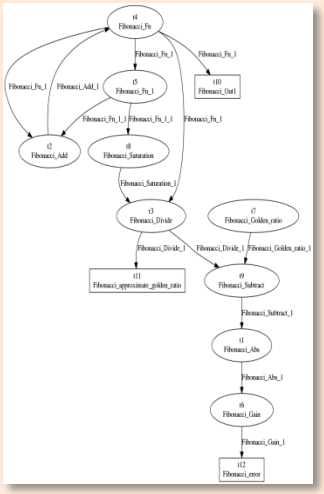
並行性抽出　Extract Concurrency

eMBPではデータフローはブロック図に表現されているため、Simulinkモデルからグラフ構造を抽出するだけ

※コード生成時に必要な意味論としては、通信により協調動作する並列モデルを表現する



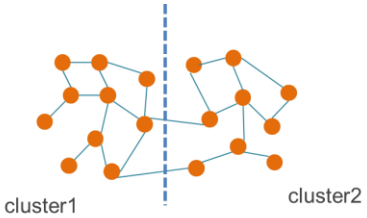
制御モデル

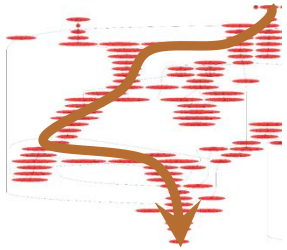


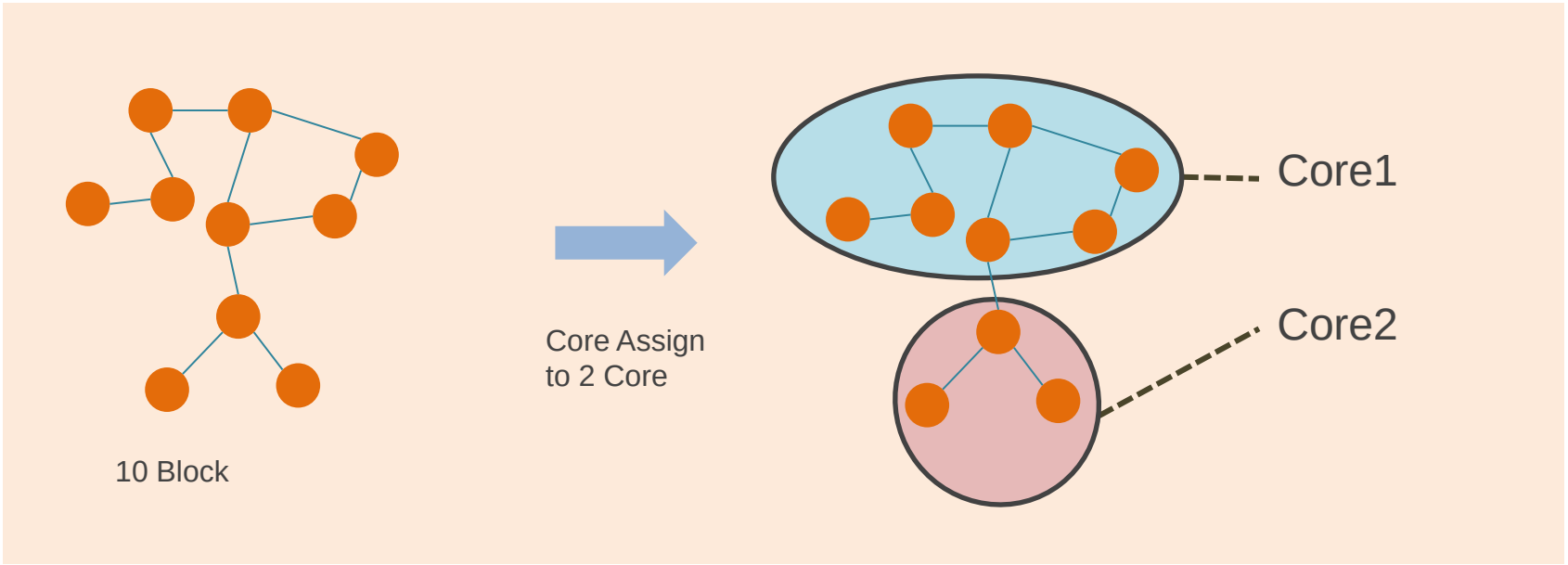
DFG

コア割り当て

- いくつかの条件を満たす組み合わせ問題
  - 負荷分散(Load balancing)
    - できるだけコアが均等に利用されるように
  - 通信オーバヘッド
    - 通信が少ない様に割り当て
  - タイミング要件
    - デッドライン(例： 1 周期内）を守れないのはNG

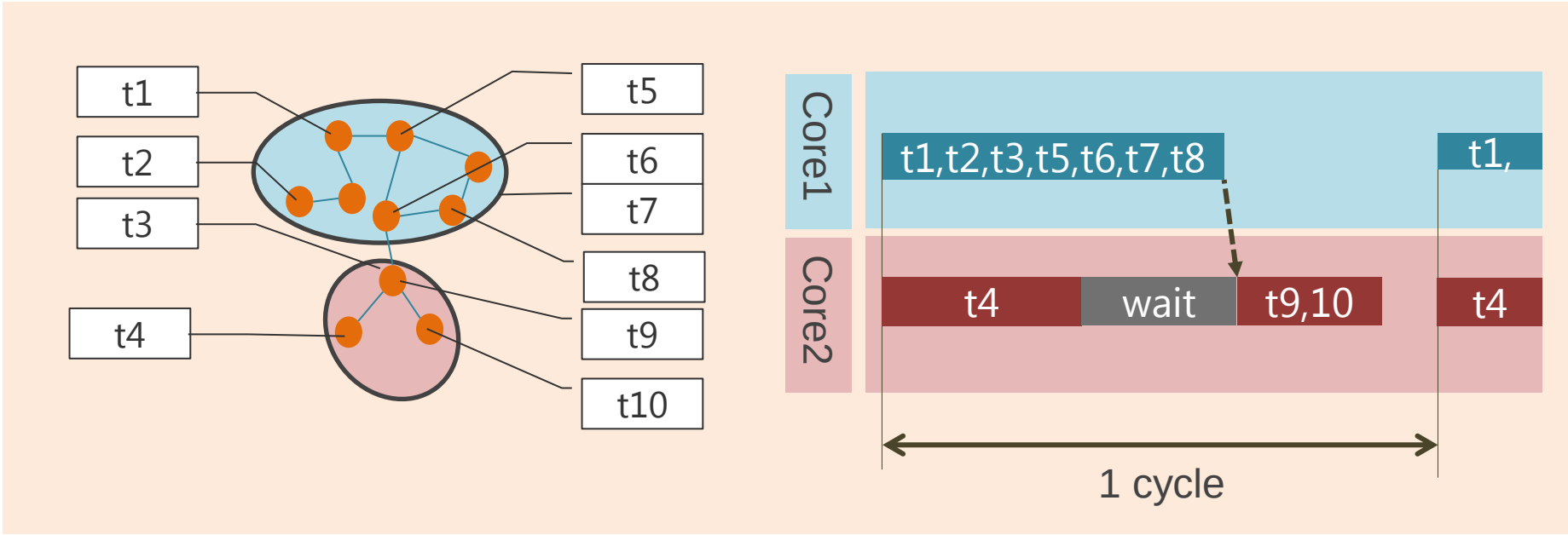






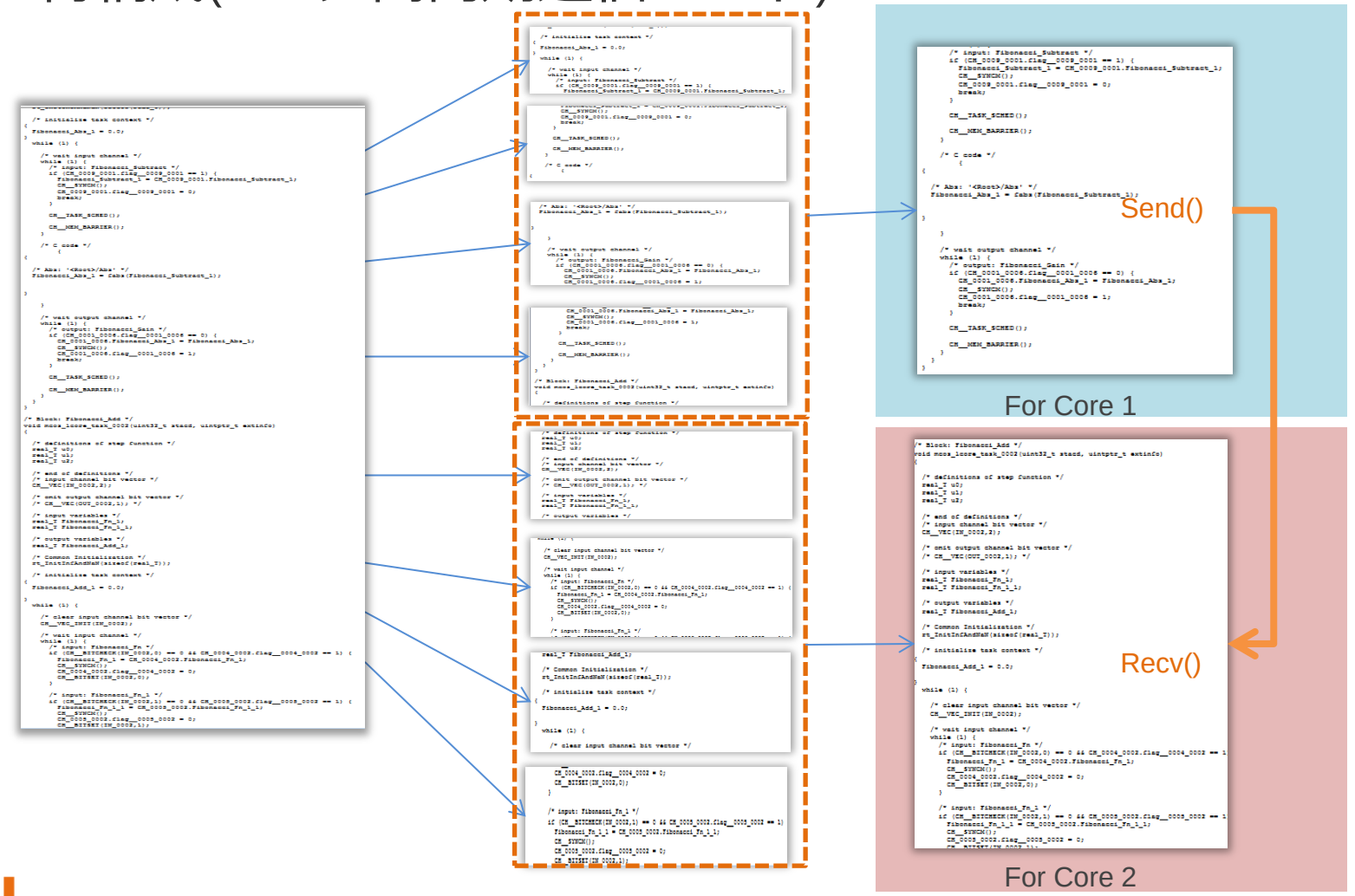
並列コード生成

- ブロック/サブシステム間の依存関係とコア割り当ての情報を元に並列コードを生成（再構成）する。

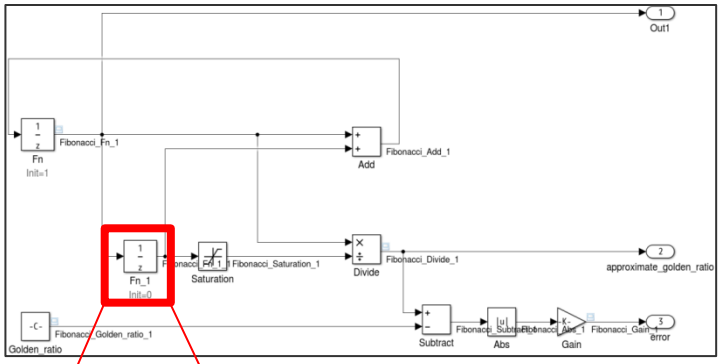


コード再構成（イメージ）

対応ブロック毎に分割されたコード片を割り当てコア毎に再構成(+コア間同期通信コード)



SHIMを利用したブロック性能見積り



Simulink モデル

SHIM

```
…
<CommonInstructionSet>
 <InstructionSet>
    <Instruction name="add">
      <Performance
          best="xxx"
          typical="xxx"
          worst="xxx"
      />
    >
    ….
</CommonInstructionSet>
…
```

```
Block_function()
{
  int x;
  int y;
  int z;
  x = 3;
  y = 4;
  z = x + y;
}
```

①

Block対応生成コード (イメージ)

```
define i32 @block_function() #0 {

  %x = alloca i32, align 4
  %y = alloca i32, align 4
  %z = alloca i32, align 4

  store i32 3, i32* %x, align 4
  store i32 4, i32* %y, align 4

  %1 = load i32* %z, align 4

  %2 = load i32* %x, align 4
  %3 = load i32* %y, align 4

  %4 = add nsw i32 %2, %3

  store i32 %4, i32* %z, align 4

  ret i32 0

}
```

LLVM命令列

Σ

各命令の処理量の総和

②

```
<?xml version="1.0" encoding="UTF-8"?>
- <sm:blocks xsi:schemaLocation="http://example.com/SimulinkModel SimulinkModel.xsd" name="Fibonacci"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:sm="http://example.com/SimulinkModel">
  - <block name="Fibonacci_Abs" rate="-1" blocktype="Abs">
    - <input port="Fibonacci_Abs_1" line="Fibonacci_Subtract_1">
        <connect port="Fibonacci_Subtract_1" block="Fibonacci_Subtract"/>
      </input>
    - <output port="Fibonacci_Abs_1" line="Fibonacci_Abs_1" usename="true">
        <connect port="Fibonacci_Gain_1" block="Fibonacci_Gain"/>
      </output>
  </block>
  - <block name="Fibonacci_Add" rate="-1" blocktype="Sum">
    - <input port="Fibonacci_Add_1" line="Fibonacci_Fn_1">
        <connect port="Fibonacci_Fn_1" block="Fibonacci_Fn"/>
      </input>
    - <input port="Fibonacci_Add_2" line="Fibonacci_Fn_1_1">
        <connect port="Fibonacci_Fn_1_1" block="Fibonacci_Fn_1"/>
      </input>
    - <output port="Fibonacci_Add_1" line="Fibonacci_Add_1" usename="true">
        <connect port="Fibonacci_Fn_1" block="Fibonacci_Fn"/>
      </output>
  </block>
  - <block name="Fibonacci_Divide" rate="-1" blocktype="Product">
    - <input port="Fibonacci_Divide_1" line="Fibonacci_Fn_1">
        <connect port="Fibonacci_Fn_1" block="Fibonacci_Fn"/>
      </input>
    - <input port="Fibonacci_Divide_2" line="Fibonacci_Saturation_1">
        <connect port="Fibonacci_Saturation_1" block="Fibonacci_Saturation"/>
```

BLXMLの該当ブロックの性能情報

今後の活動・開発予定

■今後の展望

eMBP開発・今後

- 対応要素(Simulinkブロック）を増やす
  - Matlab依存部分の継続的なメンテナンスが必要
- UI改善
  - GUI化(現状は主にCUI)
- SHIM情報の活用(Ver 1.0, 2.0)
- より効果的な可視化表現
- 他ツールとの連携
  - 同様の領域を扱うツール群とのツールチェーン
- 検証機能検討
  - 形式検証等、並列性に起因するバグの早期発見
  - 機能安全対応

Thank you.