モデルベース並列化ツールeMBPの紹介

Embedded-Multi-Core-Summit 2020 @ Online

Hiroshi Fujimoto
Expert, eSOL Co.,LTD

本日の内容

■背景
- マルチコアの普及、ソフトウェア開発課題-解決技術、 MBD等

■eMBPの概要
- ソフトウェアの構成、特徴

■eMBPの各機能紹介
- ブロック抽出、SHIM性能見積もり、コア割り当て、コード生成、可視化

■ツール操作概要
- 利用シナリオ、GUI、結果レポート

■今後の展開
- 現在開発検討している機能の一部についての説明

■まとめ

■参考URL

eMBP製品・技術の背景

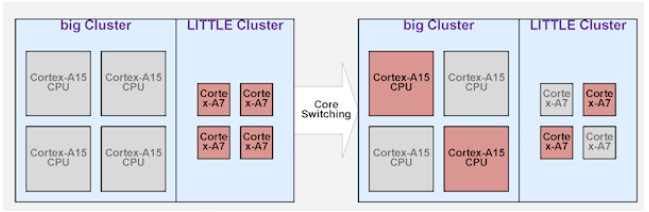
■背景

■マルチコアの普及
■マルチコア・ソフトウェア開発の課題と対策技術
■自動並列化ツール
■モデルベース開発
■（参考）eSOLマルチコア製品

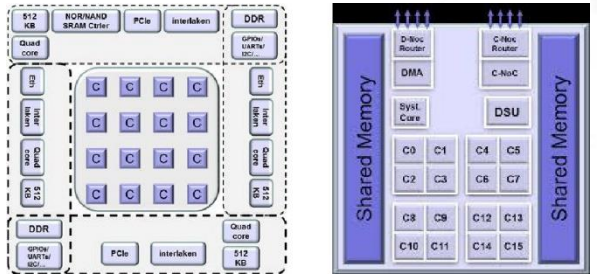
マルチコア(HW)の普及

■既に様々な分野で使われている
- PC,SmartPhone,Tabletでは2～8コアが普通
  - クロックを上げずに平均性能を上げるため。
- HPC分野
  - 数値計算の高速化。流体・構造計算などで、高性能な計算機が必要。
- 組み込み機器でも大規模計算の要求が高まる
  - 信号処理・画像認識のリアルタイム処理
  - AI関連の処理(Deep Learinig)
- 車載システム(自動運転)がけん引
  - 位置推定、衝突回避

■ チップも進化している
- ARM, Intelの「マルチコア」だけでなく、 KALRAY などの「メニーコアチップ」も進化している。クラスタ構造を持つ。



ARM bigLITTLE(Hetero)



Karlay MPPA(Cluster)

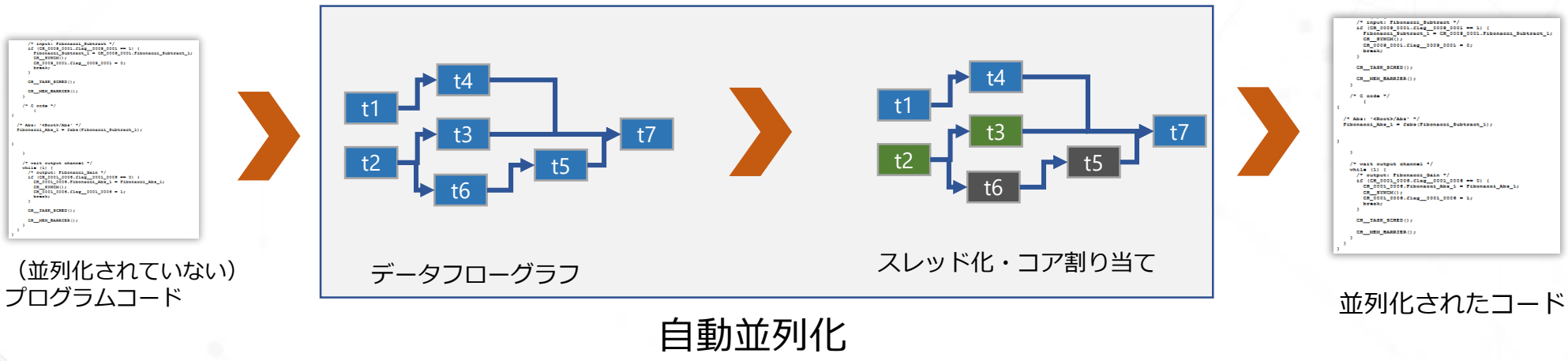
マルチコア・ソフトウェア開発の課題と対策技術

マルチコアの活用のためには、ソフトウエアを**「並列化」**する必要がある

並列化に伴う課題	対策技術(確立しているわけではない)
■Mapping ・HWの並列・分散構造とSWの並行構造のマッピングが必要。 ・HW要素・SW要素の組み合わせをみつける [複数のSWC] – [複数のコア]	**■実行環境技術：** ・マルチ・メニーコアOS ・動的なマッピング **■並列化支援技術：** ・自動並列化技術(Parallelizer) ・並列構造解析技術
■並行プログラムの設計・実装 ・並列構造をもつアーキテクチャを設計・実装する手段が必要。	**■並列設計/プログラミング技術** ・設計技術(並列処理デザインパターン) ・ライブラリ、フレームワーク、言語
■並行プログラムの動作保証 ・並行に動作するプログラムの挙動は複雑。 ・挙動の非決定性由来 ・ソフトウェアの妥当性・安全性を保障する技術が必要。	**■検証技術（静的/動的)** ・静的コード解析技術、 ・形式検証(モデル検査)技術 ・デッドロック検出 ・データ破壊可能性検出

自動並列化ツール

- OSCAR 自動並列化コンパイラ(Waseda Univ.)
  - マルチ・グレイン並列化
  - 入力はC言語プログラム
  - OSCAR APIによる並列構造表現
- モデルベース並列化(Nagoya Univ.)　--- eMBPのベース
  - 「2重階層クラスタリング」
  - 入力はSimulinkモデル
  - BLXMLによるブロック構造表現



（並列化されていない）
プログラムコード

データフローグラフ

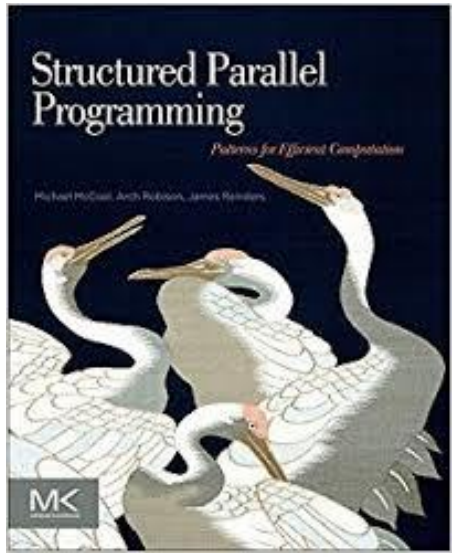
スレッド化・コア割り当て

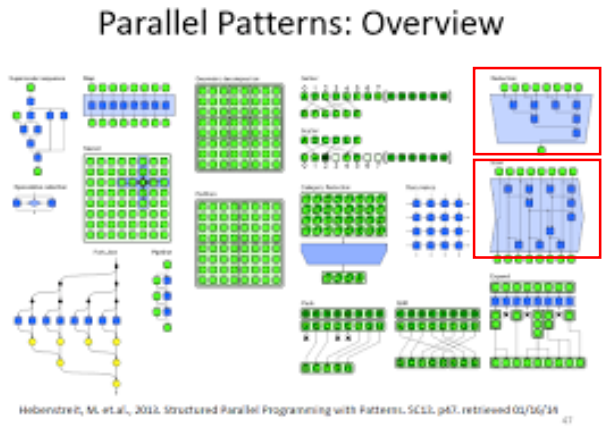
並列化されたコード

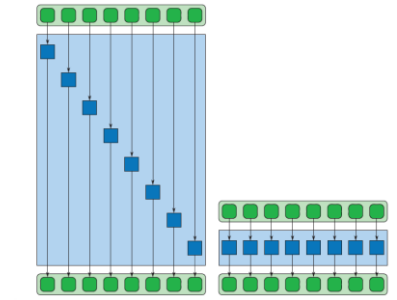
自動並列化

(参考) 並列処理のためのデザインパターン(文献)

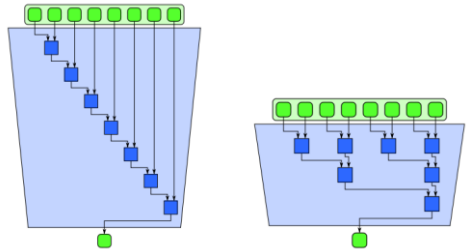
- Structured Parallel Programming (～ Patterns for Efficient Computation～)
  - Michael McCool, Arch D. Robinson, James Reinders
  - 並列プログラムを構成するためのパターンを紹介し、後半にはそれらを使った並列アルゴリズムの解説を行っている
  - GoogleのMapReduceなどもパターンの一部







Map

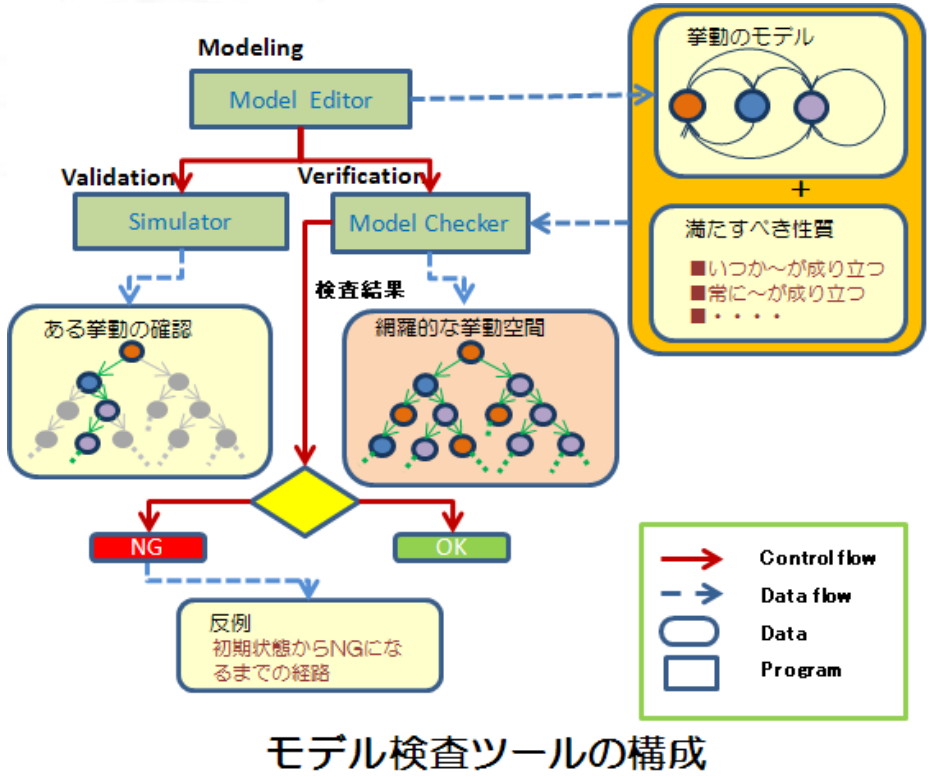


Reduce

(参考)モデル検査技術

ソフトウェアの挙動を状態遷移を元にモデル化し、挙動空間を**網羅的**に探索する事で、システムが与えられた検証式(時相論理式）を満たすかどうかを検査できる。

■ 特徴
- モデルは形式言語でかかれる
- シミュレーション機能を使った妥当性確認も可能
- 検証がNGの場合には、「反例」と呼ばれるシーケンスが得られる。
- 検証式は時相論理式が用いられる
- モデル検査ツールの例
  SPIN,NuSMV,PAT,UPPAALL他



モデル検査ツールの構成

参考URL:https://www.infoq.com/jp/articles/PAT_20111117/

モデル・ベース開発

■ Model-Based Development/Design
- Modelを起点としたソフトウェア開発手法。モデルを作成し、シミュレーションなどで妥当性を確認した上で、コードを作る（可能ならモデルからコードを自動生成）。
- 設計レベルの妥当性検証を可能にする。
- シミュレーション手法： HILS, SILS, MILS, PILS, …

■ Modelの種類
- Simulinkモデル(車載制御系DefactStandard)、SysML、Modelica(FMI/FMU)

■ 検証技術
- モデルから生成されたコードの実行とシミュレーションが同等かどうか(B2B Test,等)
- Formal Verificationとも相性が良い。

(参考) eSOLマルチコア製品

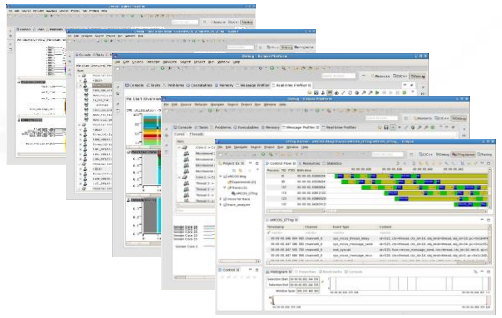
eSOLはマルチコア対応の組み込みOSとそれを活用するための開発環境を製品として提供

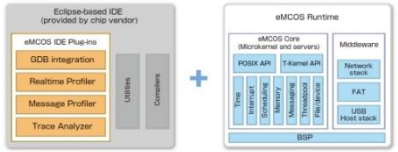
Development Environment

eMBP

eMCOS SDK

eBinder

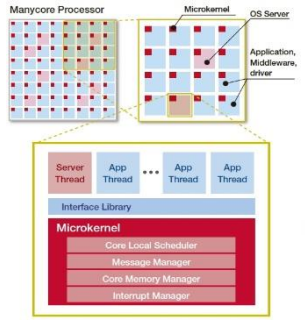


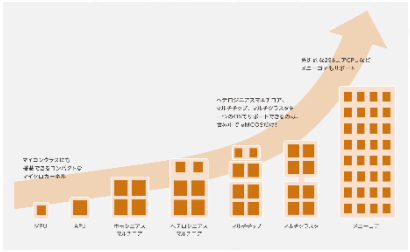


Operating Systems

eMCOS

eMCOS POSIX

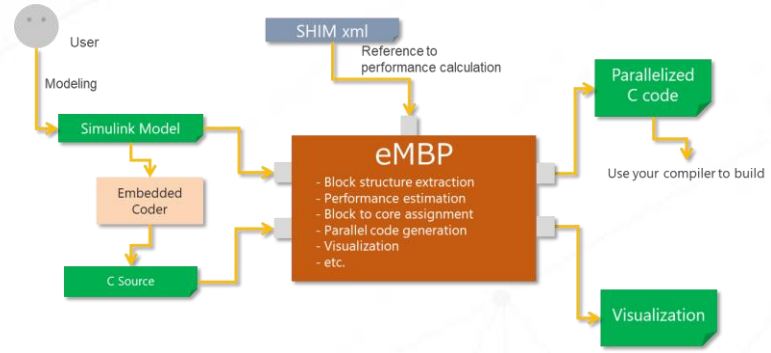




https://www.esol.co.jp/embedded/multicore_manycore.html

eMBP ソフトウェア紹介

■eMBPの概要



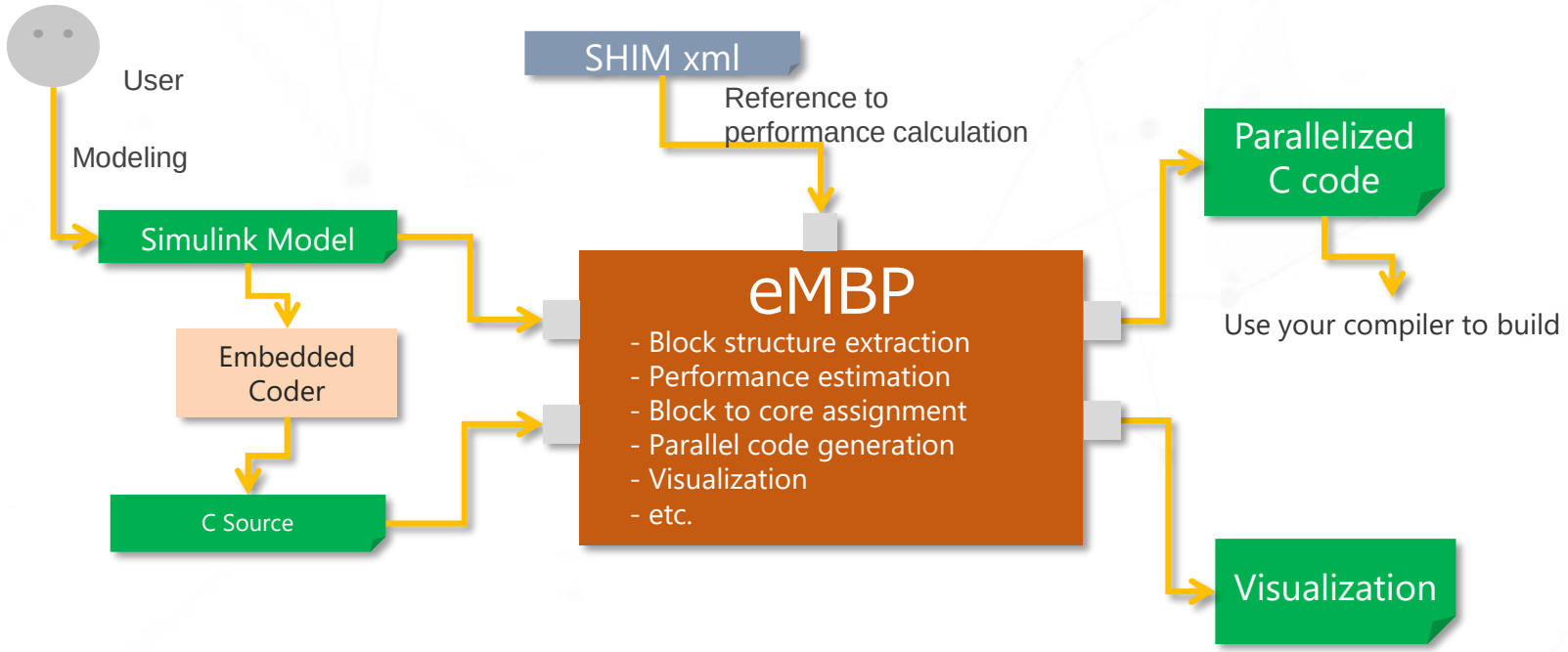
MBP = Model Based Parallelizer

■eMBPの外観

■ソフトウェア構成

■機能と特徴

eMBPの外観



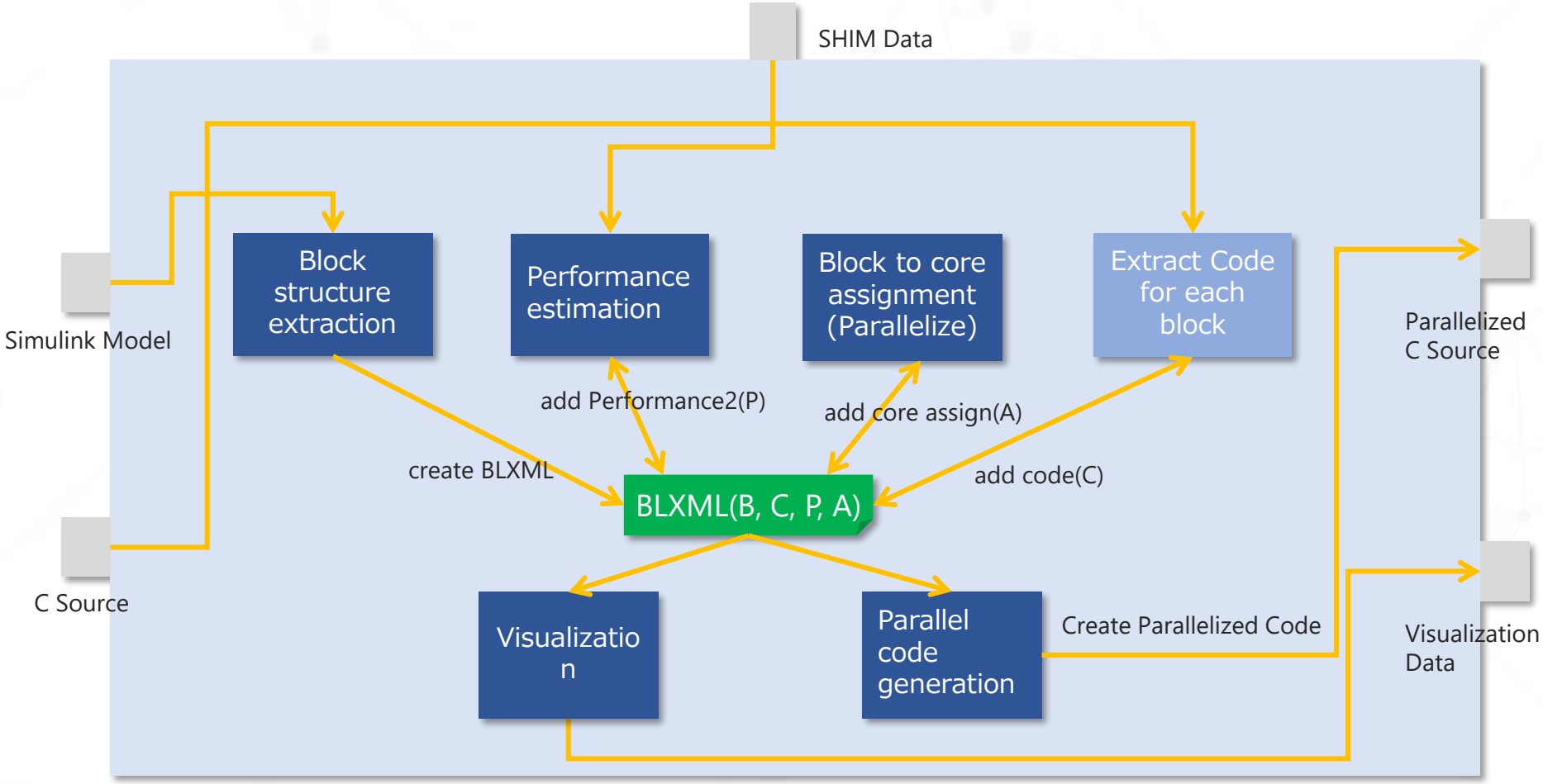
主な入力
- ■Simulinkモデル
- ■ECによる生成コード
- ■SHIM

基本機能
- ■結果の可視ブロック構造抽出
- ■ブロック性能見積もり
- ■コア割り当て
- ■並列化コード生成化
- ■可視化

主な出力
- ■並列化されたコード
- ■可視化データ

ソフトウェア構成

各機能が共通のブロック構造データBLXMLを参照・追記して連携



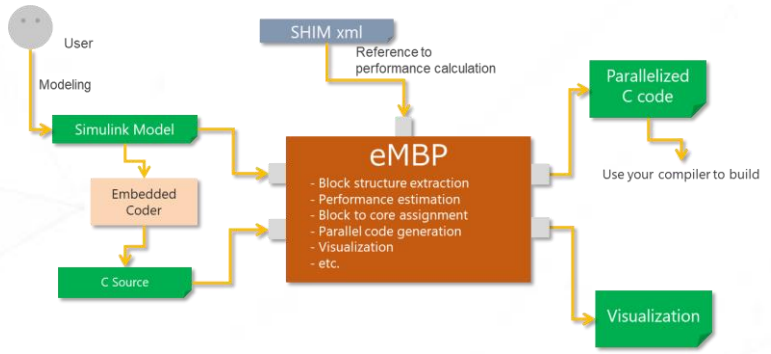
機能と特徴

- Matlab/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**。
- モデルの構造を頼りに並列化を行うため、**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ハードウェア構造記述**SHIM (※1)を採用**。
- コア割り当ては、「**階層クラスタリング(※2)**」アルゴリズムを利用
- 並列化コード生成はeMBPのOSAL層のAPIを使ったコードを生成するため、ターゲット非依存。
- **PILS(※3)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 (IEEE)
※2 名古屋大研究成果
※3 Processor In the Loop Simulation

eMBP ソフトウェア紹介

■eMBPの各機能紹介

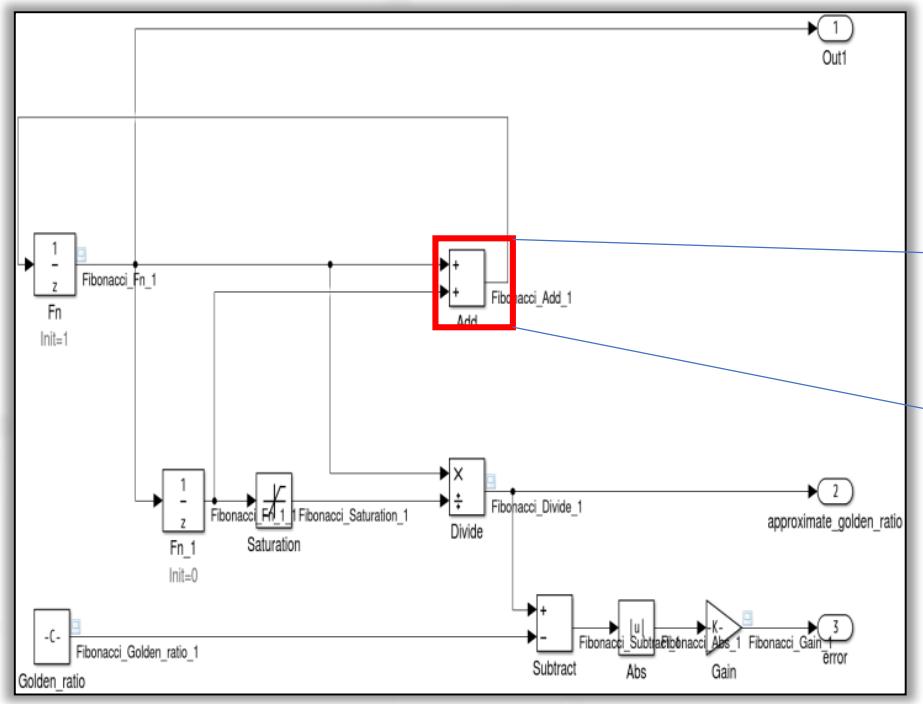


MBP = Model Based Parallelizer

■eMBPの外観
■ソフトウェア構成
■ブロック構造抽出
■ブロック性能見積もり
■コア割り当て
■並列コード生成
■可視化

ブロック構造抽出

Simulinkのブロック構造を解析し、ブロック、サブシステム構造、ブロック間の接続情報を抽出し、内部利用のXML(**BLXML**)形式に変換。



Simulink Model

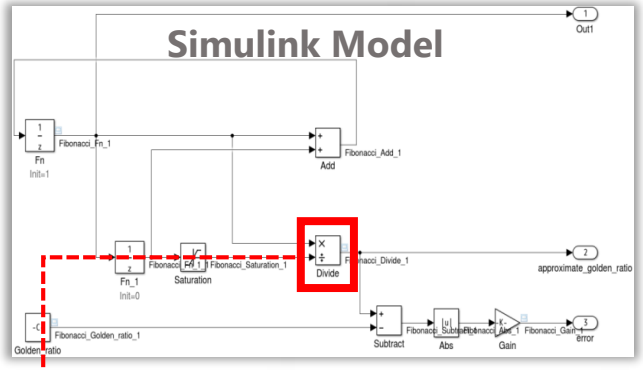
extract

```
<?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"/>
```

eMBP Internal Data(XML)

ブロック性能見積り

コア割り当て時に利用するブロック毎の性能情報を見積もる。
見積もりには、Simulinkモデルから生成されたCコードに対応するアセンブラの命令毎の処理量を、SHIMの情報を参照する事によって計算。



SHIM
CommonInstructionSet

Instruction Name	Performance(cycle)
add	10
sub	10
div	40
mul	20
call	5

generate by E.C.

reference

Generated Code

Block対応生成コード

compile

Block対応LLVM命令列

```
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
}
```

Σ

各ブロックの性能情報 (BLXML内)

No.	Block Name	Block Type	Performance (cycle)
0	Block1	Add	xxxxxxxx
1	Block2	Xor	xxxxxxxx
2	Blcok3	Subsystem	xxxxxxxx
3	Block4	Add	xxxxxxxx
4	Block5	Inport	xxxxxxxx

(参考)SHIM

- Software-Hardware Interface for MULTI-MANY core
  - ソフトウェア視点でモデル化されたハードウェア記述(XML形式)
- MCAにて仕様策定・標準化され、現在はIEEE標準として認可済み

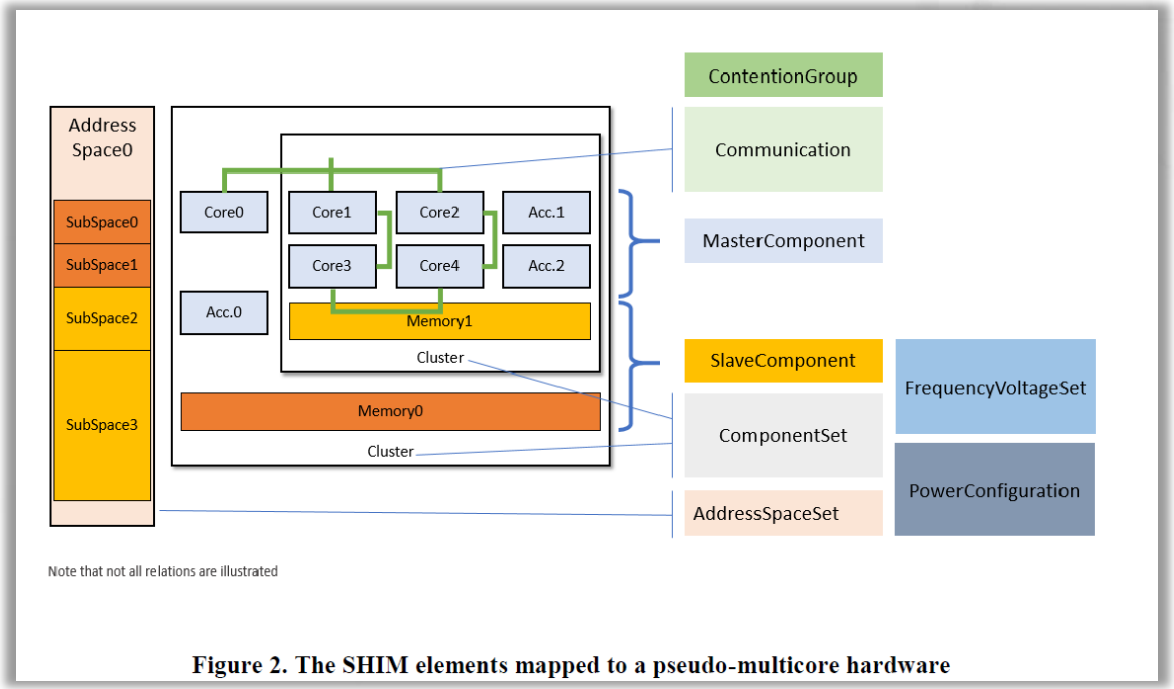
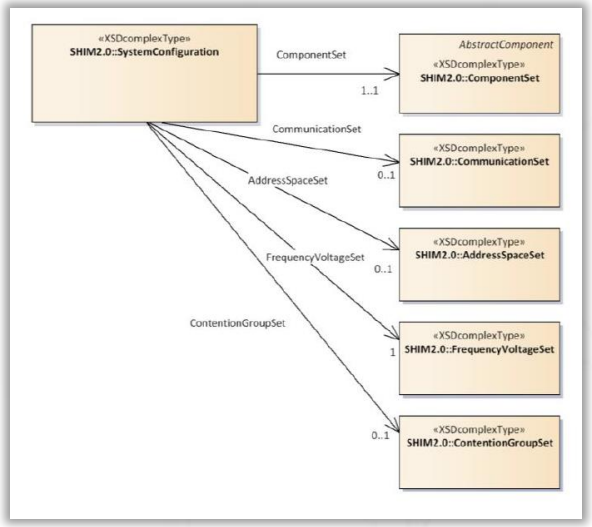


Figure 2. The SHIM elements mapped to a pseudo-multicore hardware

SHIM概念図

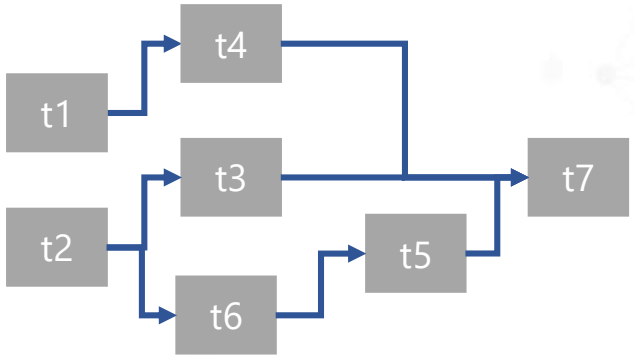


SHIM Schema TopLevel

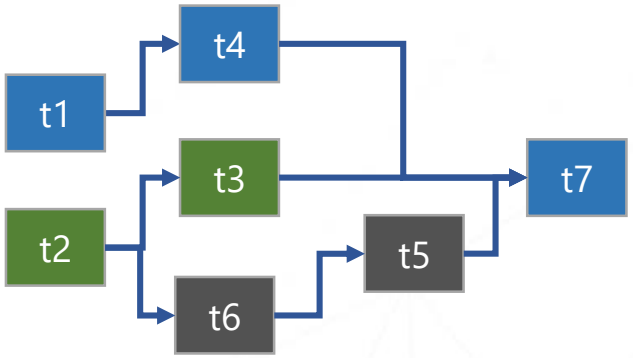
コア割り当て

ブロックへ のCore割り当て(Mapping)を行う。コア割り当てには「2重階層クラスタアルゴリズム」(名大研究成果）を利用。

SW(BLXML)



Mapping



割り当て結果(BLXML)

HW(SHIM)

※ 割り当てアルゴリズム

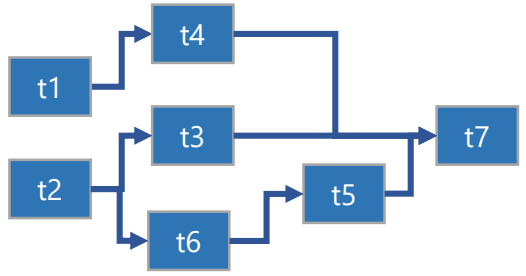
- Min-Cut戦略によるロードバランス(ノード・エッジの重み付き)
- 接続関係、ATOMIC属性指定など。
- クリティカルパス上のブロックは同じコア

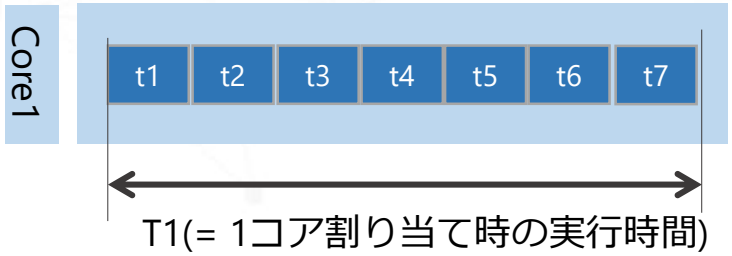
補)コア割り当てによる処理性能向上

eSOL MBP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。

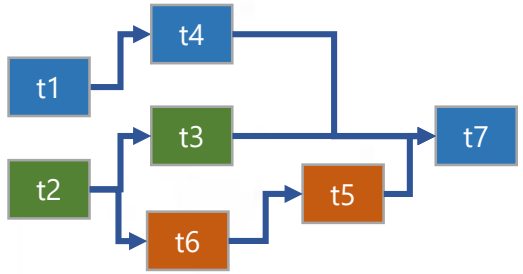
性能向上率は、１コア割り当てした場合（並列化無し）の計算時間との比較によって得る。

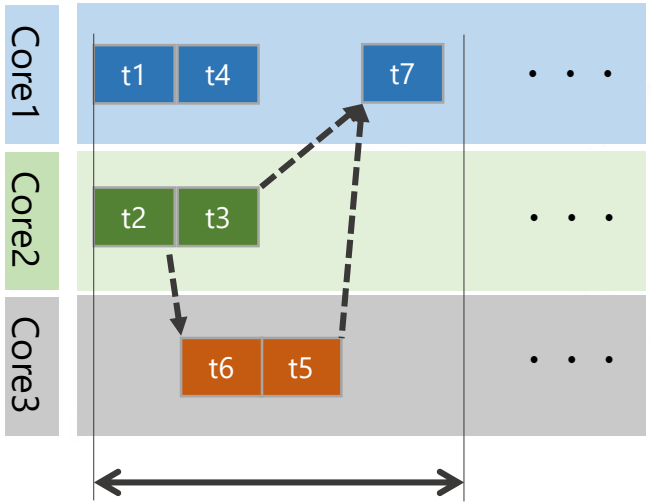
1-core





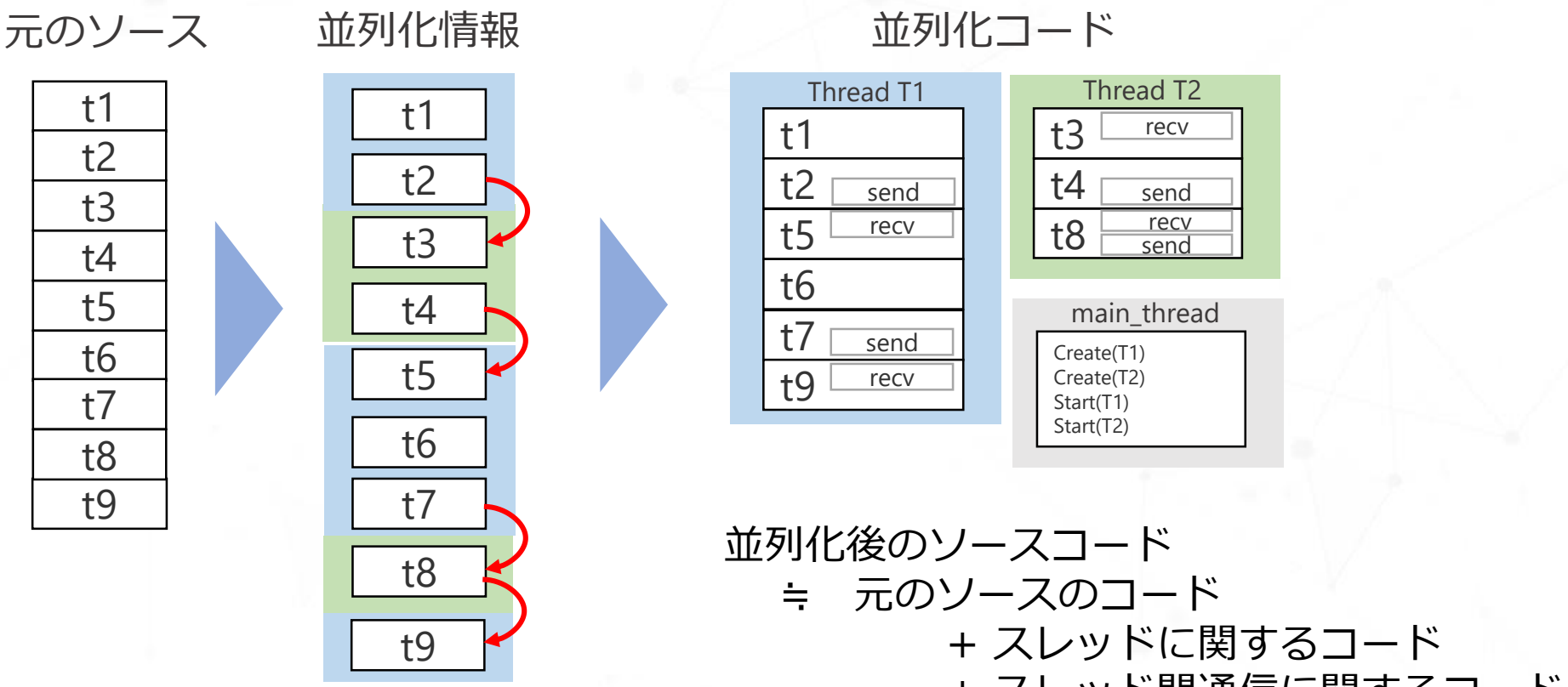
3-core





並列コード生成

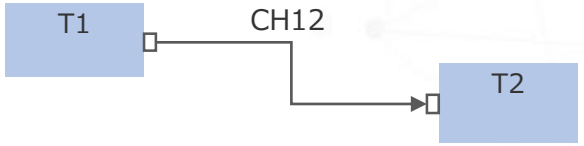
ブロック/サブシステム間の依存関係とコア割り当ての情報を元に並列コードを生成する。
その際、スレッドに関するコード、スレッド間通信に関するコードが加えられる。



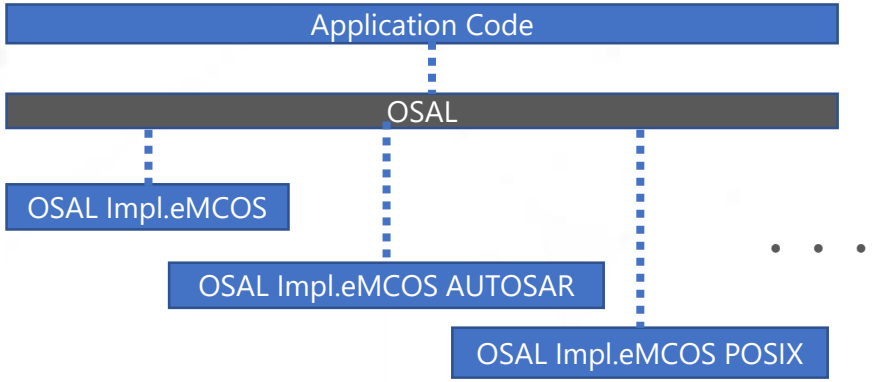
並列化後のソースコード
≒　元のソースのコード
+ スレッドに関するコード
+ スレッド間通信に関するコード

補）OSAL Concept

生成されるスレッド、通信に関するコードは、はOSAL(OS Abstraction Layer)として定義されているAPIを使ったターゲット非依存のものとして生成される。
OSAL実装をターゲット毎に作成する事で、さまざまなターゲットOSに対応可能。



Thread-Channel Model



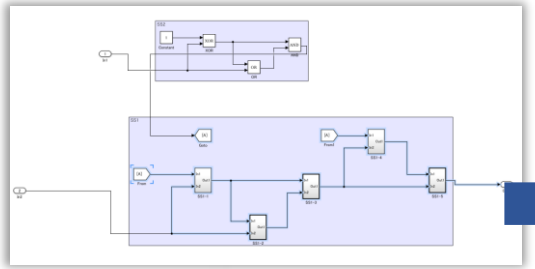
Layer Image

OSAL API	Description
mbp_thread_create()	Create Thread Object
mbp_thread_start()	Start Thread
mbp_channel_create()	Create Channel Object
mbp_channel_send()	Send Data using channel
mbp_channel_recv()	Receive Data using channel

OSAL-APIs

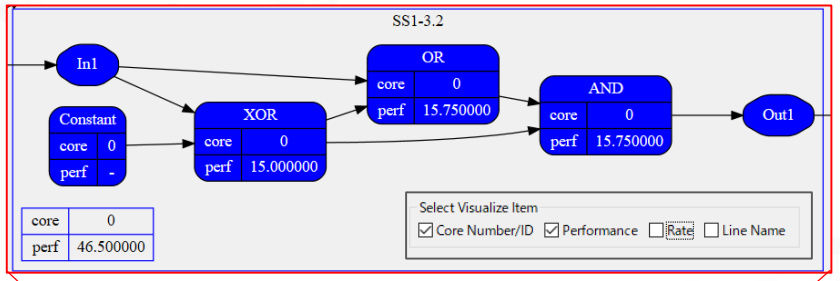
可視化①：ブロック構造

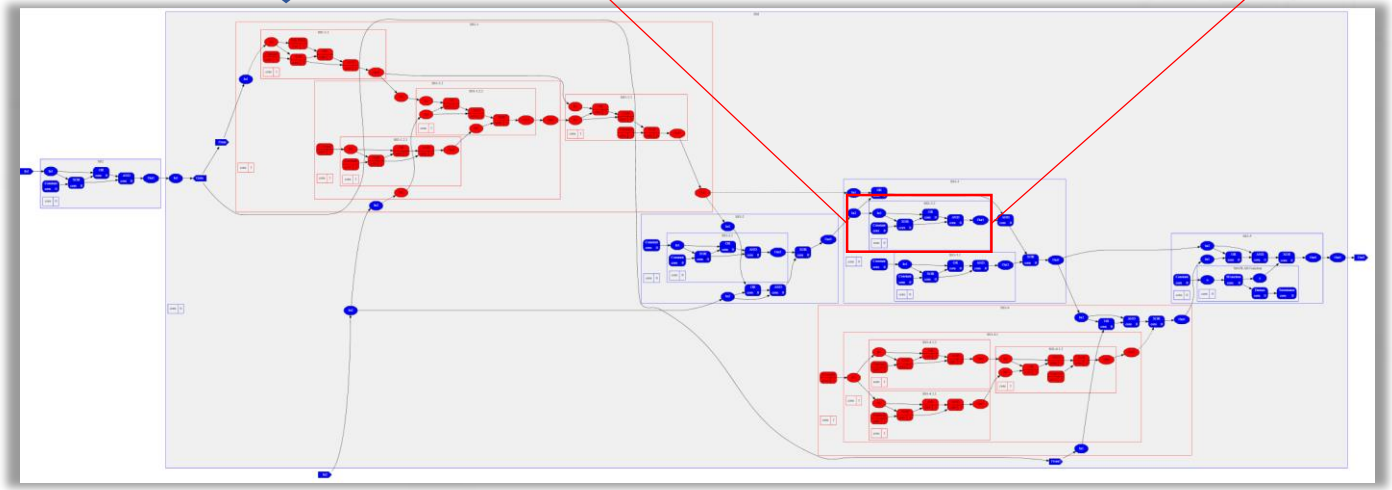
抽出したブロック構造（グラフ構造）を表示
・ブロック毎の情報(コア、SHIM見積もりした性能、レート)
・クリティカルパス表示



元のSimulinkモデル

ブロック抽出
性能見積もり
コア割り当て

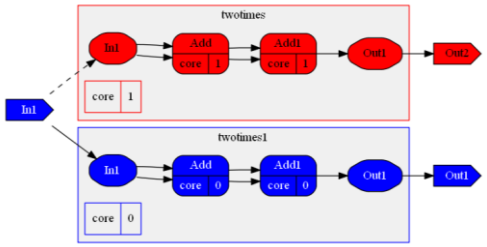




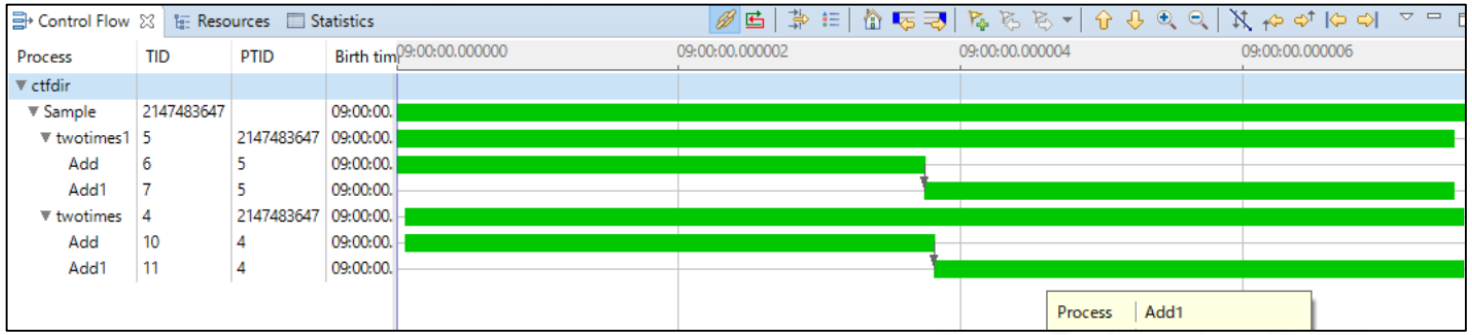
コア割り当て結果(2Core)の可視化結果

可視化②：スケジュール情報

コア割り当て結果としての想定スケジュール（タイミングチャート）を表示
スケジュール情報はCTF(※1)として出力し、TraceCompass(※2)のControlFlowView(※)を利用して表示



eMBP GettingStarted モデル

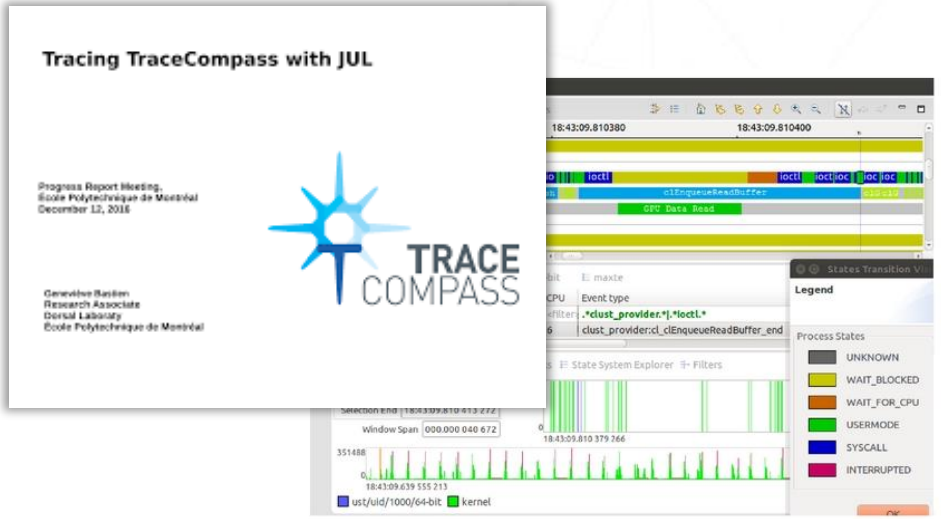


スケジュール情報

※1 CTF = Common Trace Format
※2 本来はLinuxなどのOSのログを可視化するツール。

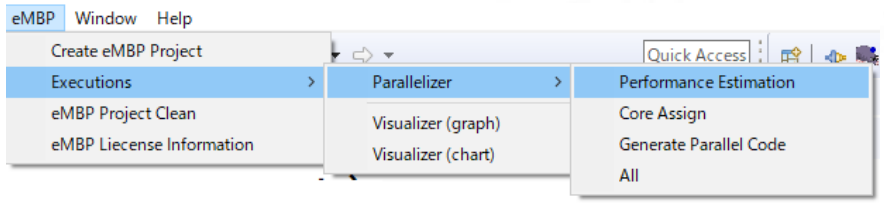
(参考)トレースフォーマットCTF

- Common Trace Format
  - https://www.efficios.com/ctf
- 対応ツール
  - TraceCompass
    - Eclipse-plug-in
- 対象ファイルフォーマット
  - トレース要素定義ファイル(DSL) meta
    - イベントを定義
  - 対象ファイル
    - 計測データ



ツールの操作イメージ

■ツール操作概要



■利用シナリオ

■GUIによる操作

■機能と特徴

eMBP利用シナリオ

[User] Simulinkにより制御設計を行う

↓

[User] Embedded Coderによって対象モデルからコード生成

↓

[MBP] 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

↓

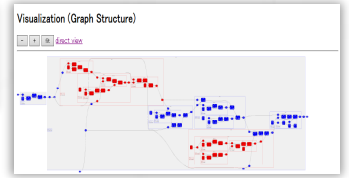
[User] 実行プラットフォーム用にコンパイルし、実行モジュールを得る

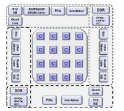
↓

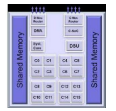
[User] 実行プラットフォームに配置(OSコンフィギュレーション設定)

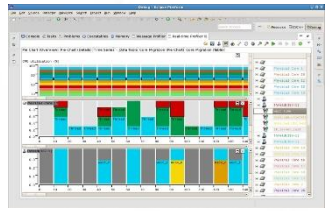
↓

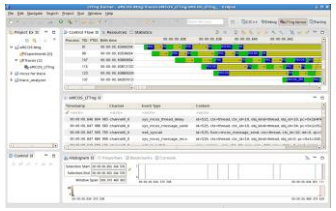
[User] 実行検証・導入





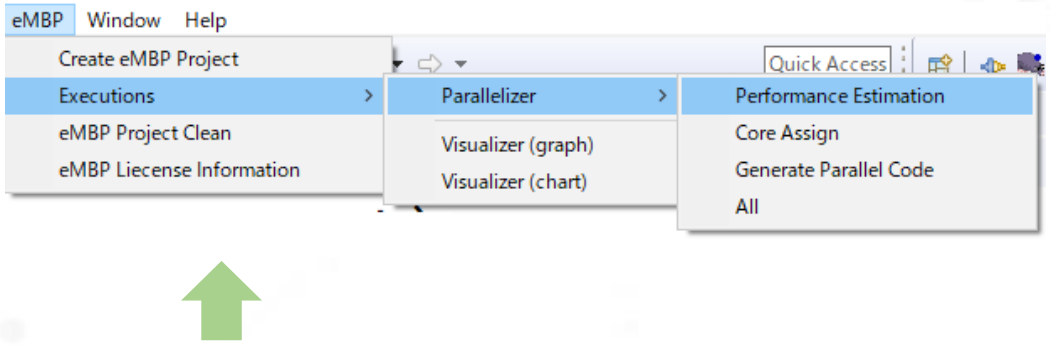






GUIによる操作

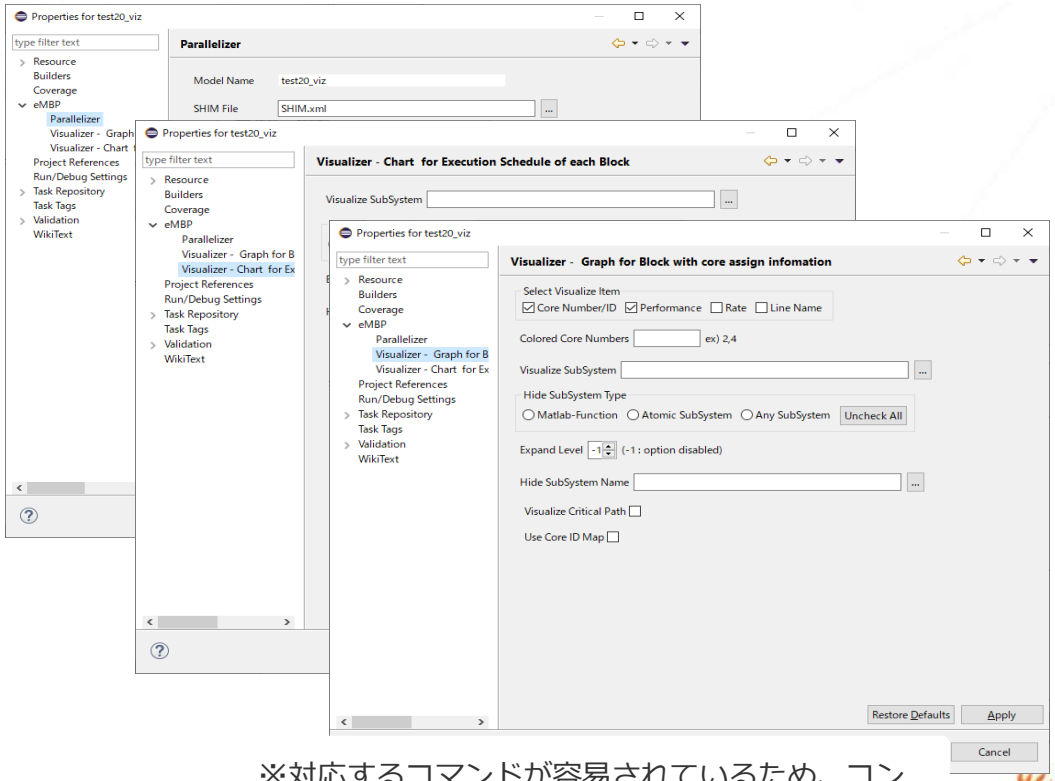
eMBP操作は基本的にGUIを通して行う。GUI項目としては、コマンド（機能）の実行のための**メニュー**と、コマンドオプションを指定するための**設定画面**がある。
なお、対応する機能はコンソールからのコマンド実行でも可能。



■コマンド実行メニュー
・性能見積もり
・コア割り当て
・可視化

■Parallelizer設定画面
・割り当てコア数
・コア間通信コスト指定
・コアIDマップデータ指定
・SHIMファイル指定
・他

■可視化設定画面
・ブロック表示情報指定
・クリティカルパス表示指定



※対応するコマンドが容易されているため、コンソールからのコマンド実行も可能

実行結果のレポート

コマンドの実行終了時にレポートが表示される

SHIM Info

SHIM Version	1.0
SHIM File Name	SHIM.xml
MasterComponent Name	sample

Performance Estimation Results

No.	Block Name	Block Type	Performance (Task)	Performance (Update)	Performance (Total)
0	test20_viz/In1	Inport	0.000000	0.000000	0.000000
1	test20_viz/In2	Inport	0.000000	0.000000	0.000000
2	test20_viz/Out1	Outport	0.000000	0.000000	0.000000
3	test20_viz/SS1	SubSystem	0.000000	0.000000	0.000000
4	test20_viz/SS1/From	From	0.000000	0.000000	0.000000
5	test20_viz/SS1/From1	From	0.000000	0.000000	0.000000
6	test20_viz/SS1/Goto	Goto	0.000000	0.000000	0.000000
7	test20_viz/SS1/In1	Inport	0.000000	0.000000	0.000000
8	test20_viz/SS1/In2	Inport	0.000000	0.000000	0.000000
9	test20_viz/SS1/Out1	Outport	0.000000	0.000000	0.000000
10	test20_viz/SS1/SS1-1	SubSystem	0.000000	0.000000	0.000000
11	test20_viz/SS1/SS1-1/In1	Inport	0.000000	0.000000	0.000000
12	test20_viz/SS1/SS1-1/In2	Inport	0.000000	0.000000	0.000000
13	test20_viz/SS1/SS1-1/Out1	Outport	0.000000	0.000000	0.000000
14	test20_viz/SS1/SS1-1/SS1-1.1	SubSystem	6.000000	0.000000	6.000000

■SHIMによるブロック性能見積もり結果

・SHIM情報
・ブロック毎の性能

Model Info

Model Name	test20_viz
Number of Edges	177
Number of Nodes	141
Rate Type	MULTIPLE RATE

CoreAssign Results

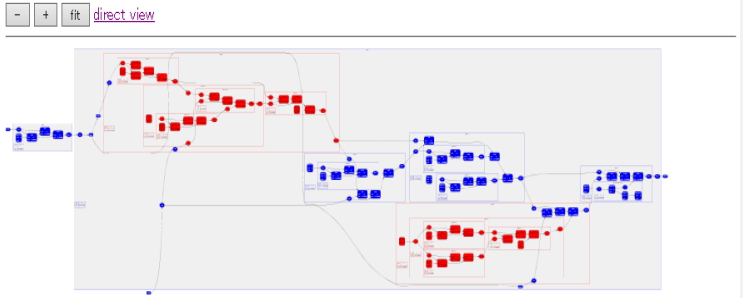
Total MaxDelay(2 core)	3449
Total SumDelay(1 core)	3500
Ratio (SumDelay/MaxDelay)	1.01449

Communication Overhead (cycle)

from core i to core j (for row i, column j)

Core No.	0	1
0	0	15
1	15	0

Visualization (Graph Structure)



■コア割り当て結果

・1コア割り当てにたいしての性能向上率
・コア間通信情報
・コア割り当て結果（グラフ）

29

今後の活動・開発予定

■今後の展開

■今後開発を検討している機能
■ヘテロジニアスマルチコア対応
■SHIMを使った性能見積もり精度の向上
■形式検証による不具合検出

今後開発を検討している機能

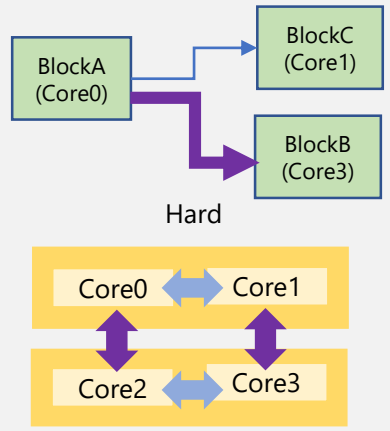
- ヘテロジニアスマルチコア対応
- SHIMを使った性能見積もり精度の向上
- 形式検証による不具合検出
  - 「並列化に伴う課題/並行プログラムの動作保証」への対応
- 他ツールとの連携
  - 同様の領域を扱うツール群とのツールチェーン
- ParallelizerからSoftwareMappingToolへの進化
- その他、継続的な機能改善開発
  - 対応要素(Simulinkブロック）を増やす
  - 継続的なUI改善
  - より効果的な可視化表現

色付きのものについて説明

ヘテロジニアスマルチコア対応

コア毎の処理性能が異なるチップに対応する。
段階的に対応。Step1, Step2までは開発済み。

☑



Cluster内、Cluster間
で通信コストが異なる

Step1:
コア間通信コストが異なる場合の考慮

☑

core1

instruction	perf
add	5
sub	5
mul	10
div	20

Clock=X

core2

instruction	perf
add	10
sub	10
mul	20
div	40

Clock=2X

Step2:
Multi-Frequency対応
（各コアは同種。クロックのみ異なる）

☐

core1

instruction	perf
add	5
sub	5
mul	10
div	20

core2

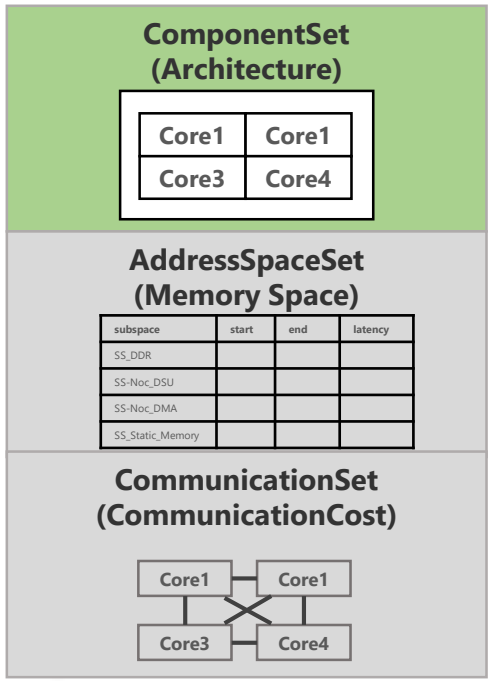
instruction	perf
add	3
sub	4
mul	15
div	30

Step3:
異なるコア種類対応
（各コアの各命令毎の性能も異なる）

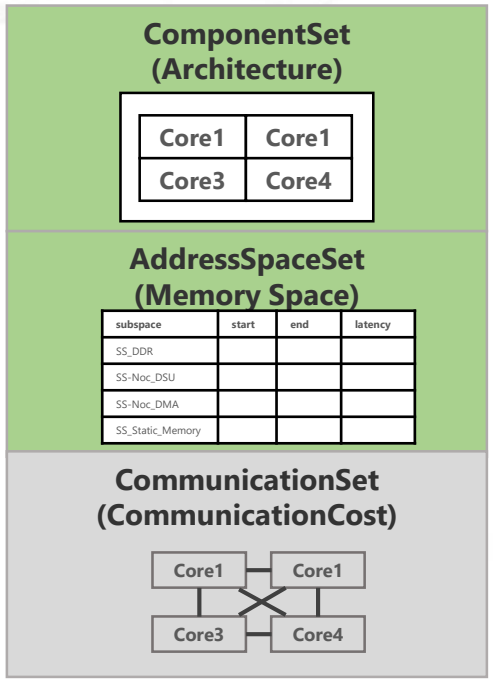
SHIMを使った性能見積もり精度の向上

SHIMの情報の有効活用活用

- 現在はSHIM要素のグループ「ComponentSet」のみ使用
- 今後はメモリアクセス性能の考慮をするため「AddressSpaceSet」も利用予定



Current Ver.



Feature Ver.

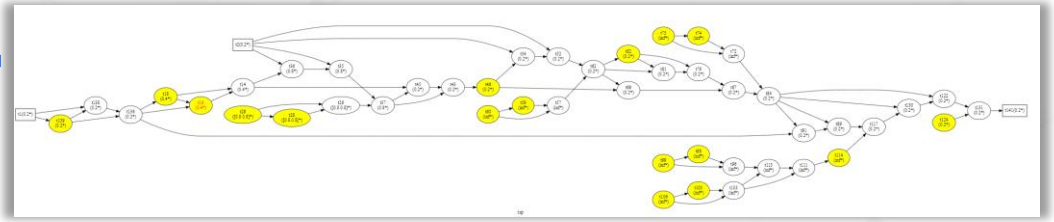
形式検証による不具合検出

モデル-コア割り当て構造を解析し、並列性に起因するバグの早期発見
- デッドロック検出
- デッドライン検出

→コード生成のための内部的なグラフ構造を利用した形式検証等の利用を検討

グラフ構造抽出

検証対象



形式モデルに変換

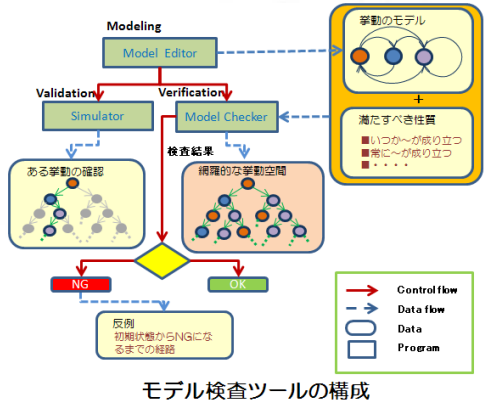
CSP, Promela, etc.

モデル検査

検証結果

検査結果を評価

OK/NG(反例)



モデル検査ツールの構成

まとめ

以下の説明を行いました

■背景
- マルチコアの普及
- マルチコアソフトウエア開発の課題と対応技術
- 自動並列化研究
- モデルベース開発

■eMBPの概要
- eMBPの外観
- ソフトウェア構成
- 機能と特徴

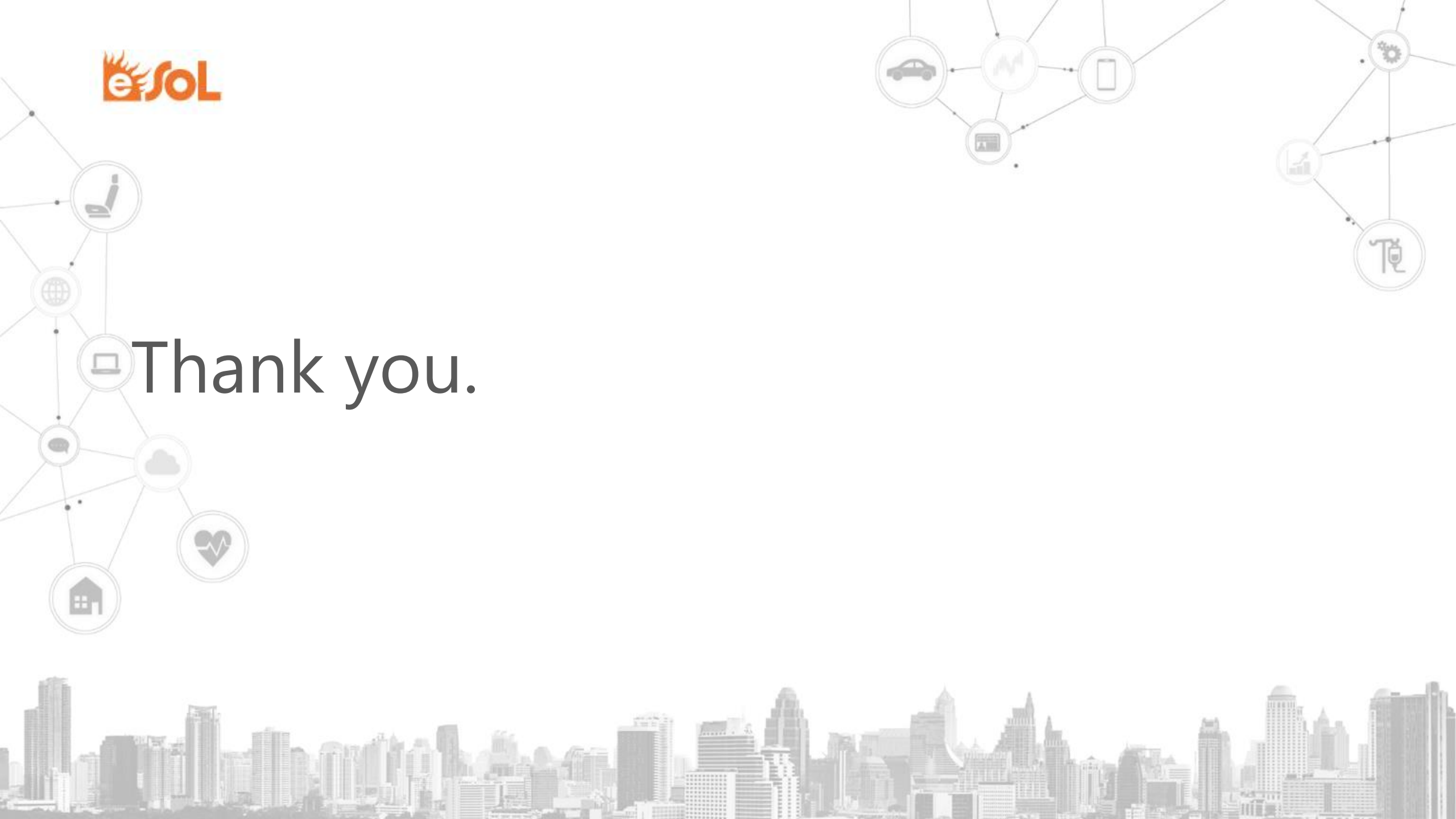
■各機能の説明
- ブロック抽出
- 性能見積もり
- コア割り当て
- 並列化コード生成
- 可視化

■今後の展開
- ヘテロジニアスマルチコア対応
- SHIMによる見積もり精度の向上
- 形式検証による不具合検出

■まとめ

参考サイト

- eSOL eMBP
  - https://www.esol.co.jp/embedded/mbp.html
- 名大PDSL
  - https://www.pdsl.jp/
- 組み込みマルチコアコンソーシアム
  - http://www.embeddedmulticore.org/
- SHIM(IEEE)
  - https://standards.ieee.org/standard/2804-2019.html
- OpenSHIM(GitHub)
  - https://github.com/openshim/shim
- KARLAY
  - https://www.kalrayinc.com/
- Embedded Target for RH850 Multicore
  - https://www.renesas.com/jp/ja/products/software-tools/tools/model-base-development/embedded-target-for-rh850-multicore.html



Thank you.