



マルチコア・並列化ソフトウェア開発支援環境 eMBPの紹介

Embedded-Multi-Core-Summit 2021 @ 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、並列化について

■eMBPの概要

- eMBPの特徴、外観
- 利用シナリオ、GUI、結果レポート
- 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

■開発中/開発検討している新機能

- 現在開発中/検討している機能の説明

■まとめ

■参考資料

■ 背景

eMBP製品・技術の背景

- 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
- 自動並列化とは
- 並列化されたコードの特徴

背景

■ マルチコア(HW)の普及

既に様々な分野で使われている

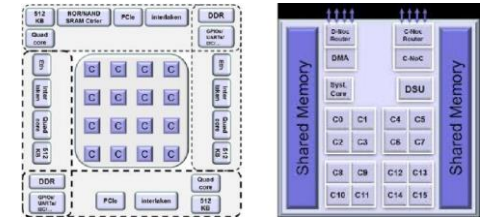
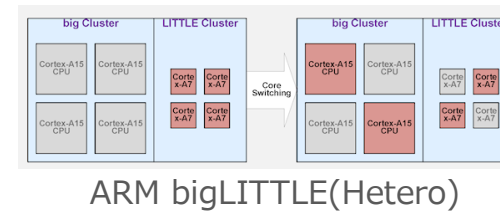
- PC,SmartPhone,Tabletでは2~8コアが普通
- HPC分野
- 組み込み機器でも大規模計算の要求が高まる
- 車載システム(自動運転)がけん引

■ 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(コア配置の階層)を持つ。

■ マルチコアを有効利用できるソフトウェアの開発が必要



- 実行環境・開発環境の対応
 - OS、開発ツール(設計、検証)
- 設計技術、実装技術が未成熟
 - プログラミング技術の確立(デザインパターン、言語、教育)



Karlay MPPA(Cluster)

マルチコア・ソフトウェア開発の課題と対策技術

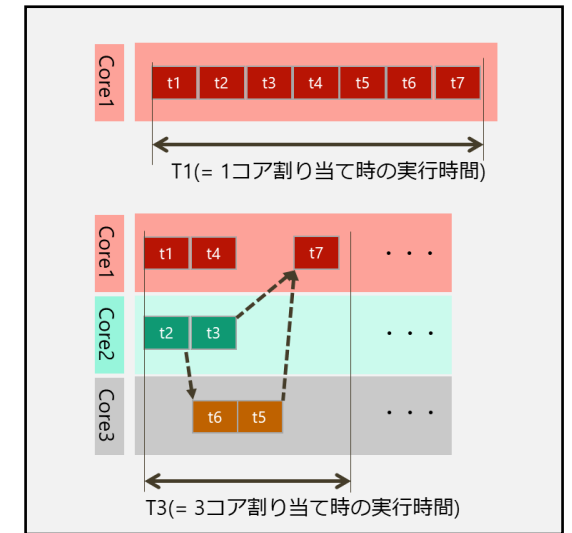
マルチコアの活用のためには、ソフトウェアを「並列化」する必要がある

並列化に伴う課題	対策技術の例
<p>■ Mapping</p> <ul style="list-style-type: none">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 <p>[複数のSWC] - [複数のコア]</p>	<p>■ 実行環境技術：</p> <ul style="list-style-type: none">マルチ・メニーコアOS動的なマッピング <p>■ 並列化支援技術：</p> <ul style="list-style-type: none">自動並列化技術(Parallelizer)並列構造解析技術 <p> </p>
<p>■ 並行プログラムの設計・実装</p> <ul style="list-style-type: none">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。	<p>■ 並列設計/プログラミング技術</p> <ul style="list-style-type: none">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
<p>■ 並行プログラムの動作保証</p> <ul style="list-style-type: none">並行に動作するプログラムの挙動は複雑。<ul style="list-style-type: none">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。	<p>■ 検証技術 (静的/動的)</p> <ul style="list-style-type: none">静的コード解析技術、形式検証(モデル検査)技術<ul style="list-style-type: none">デッドロック検出データ破壊可能性検出

並列化されたコードの特徴

【特徴1】 並行動作可能な処理単位(スレッド)をもつ

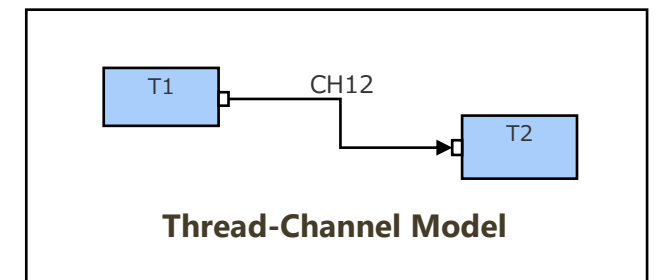
- これらの処理単位が物理的に別々な処理装置（コア等）に割り当てられることで、トータルの処理速度が向上する。
- 並列動作の表現
 - 非並列処理表現を持たない言語（C言語等）
 - スレッドライブラリによるスレッドの生成API(Posix-Thread etc.)
 - 言語仕様として並行処理表現を持つ言語
 - Go, Occam, Erlang, etc.



並列化による処理性能向上

【特徴2】 スレッド間通信の機構がある

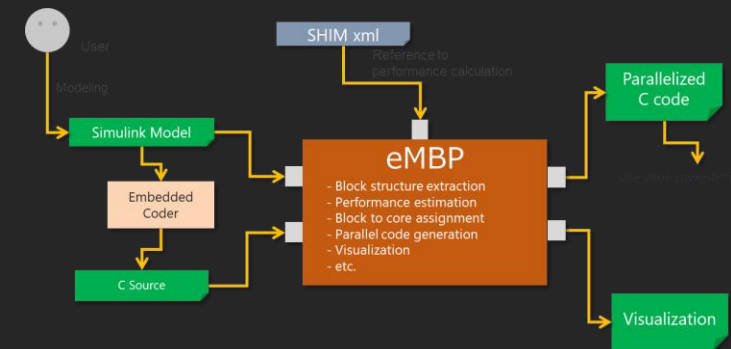
- これにより、「単なる並行動作」でなく「**協調計算**」を実現
- 共有メモリを利用するもの
- 通信チャネルを利用するもの



スレッド間通信の例

■ eMBPの概要

eMBP ソフトウェア紹介



MBP = Model Based Parallelizer

- 機能と特徴
- eMBPの外観
- 利用シナリオ&ツール操作
- ソフトウェア構成
- 基本機能一覧

eMBPの機能と特徴

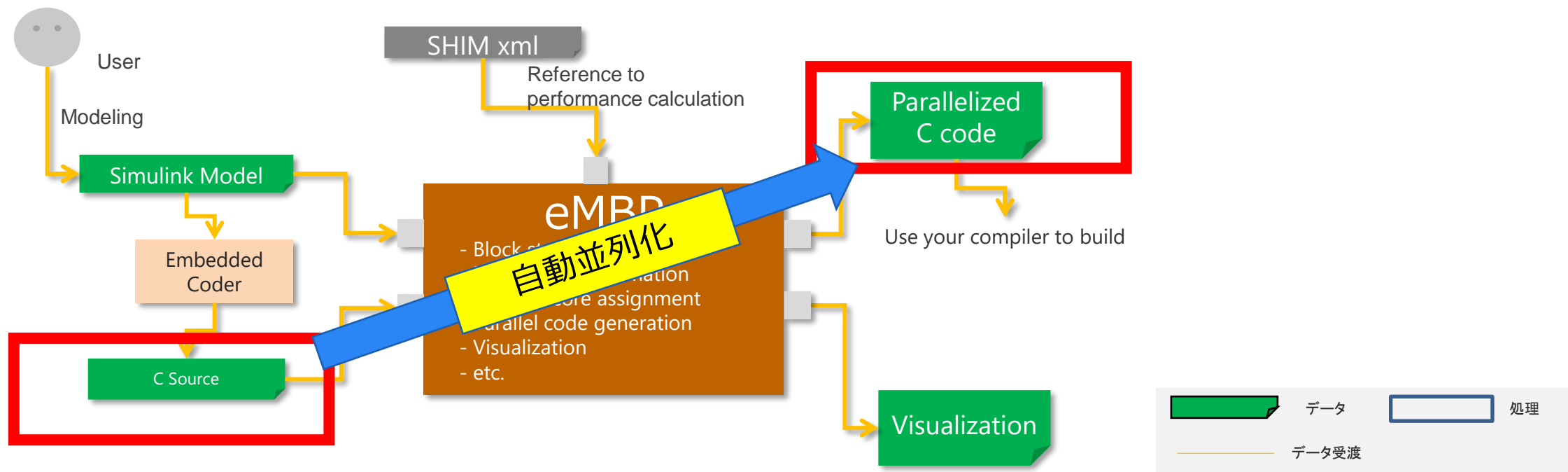
- Matlab/Simulinkで設計された制御モデルから生成されるCソースコードを**並列化**。
- モデルの構造を頼りに並列化を行う（**設計者の意図を反映できる**）。
- ブロック毎の実行性能の見積りにハードウェア構造記述**SHIM (※1)**を採用。
- コア割り当ては、「**階層クラスタリング(※2)**」アルゴリズムを利用
- 並列化コード生成は**eMBPのOSAL層**のAPIを使ったコードを生成するため、ターゲット非依存。
- **PILS(※3)システムと連携**し、モデルベース開発による動作検証を支援。

※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

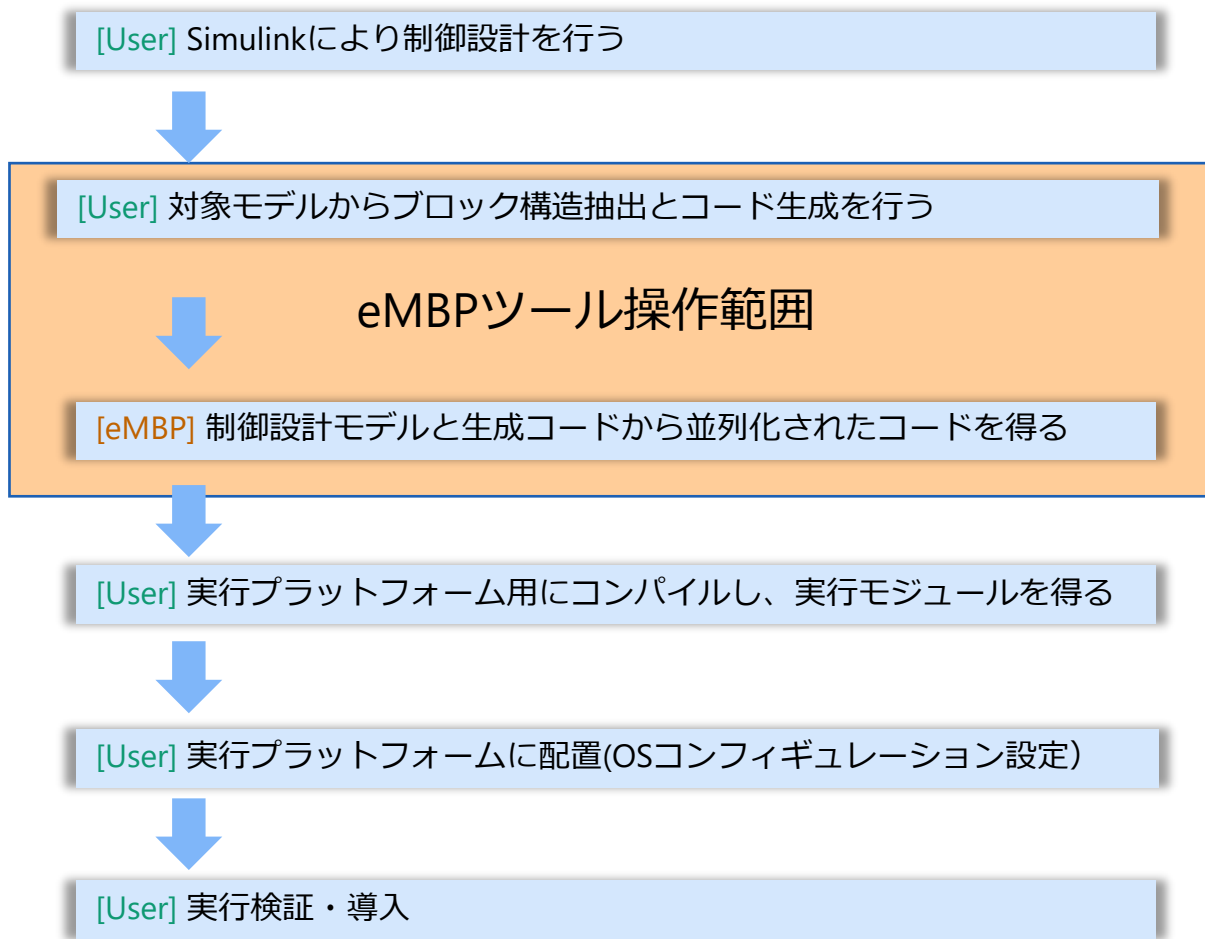
基本機能

- 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

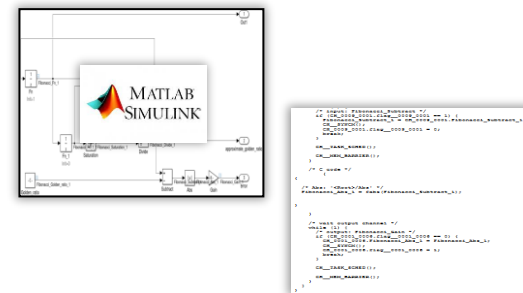
主な出力

- 並列化されたコード
- 可視化データ

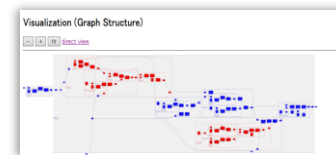
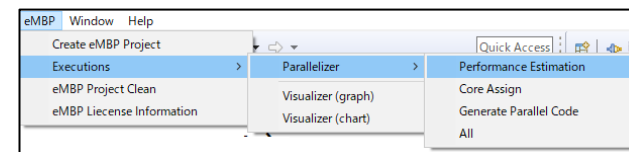
eMBP利用シナリオ&ツール操作



Matlab/SimulinkおGUIまたは
コマンドによる操作



eMBP GUIまたは
コマンドによる操作



Visualization

Performance Estimation Results

ID	Node Name	Block Type	Performance (User)	Performance (System)	Performance (Total)
1	hw00-00-000	Input	0.0000	0.0000	0.0000
2	hw00-00-001	Input	0.0000	0.0000	0.0000
3	hw00-00-002	Input	0.0000	0.0000	0.0000
4	hw00-00-003	Input	0.0000	0.0000	0.0000
5	hw00-00-004	Input	0.0000	0.0000	0.0000
6	hw00-00-005	Input	0.0000	0.0000	0.0000
7	hw00-00-006	Input	0.0000	0.0000	0.0000
8	hw00-00-007	Input	0.0000	0.0000	0.0000
9	hw00-00-008	Input	0.0000	0.0000	0.0000
10	hw00-00-009	Input	0.0000	0.0000	0.0000
11	hw00-00-010	Input	0.0000	0.0000	0.0000
12	hw00-00-011	Input	0.0000	0.0000	0.0000
13	hw00-00-012	Input	0.0000	0.0000	0.0000
14	hw00-00-013	Input	0.0000	0.0000	0.0000
15	hw00-00-014	Input	0.0000	0.0000	0.0000
16	hw00-00-015	Input	0.0000	0.0000	0.0000
17	hw00-00-016	Input	0.0000	0.0000	0.0000
18	hw00-00-017	Input	0.0000	0.0000	0.0000
19	hw00-00-018	Input	0.0000	0.0000	0.0000
20	hw00-00-019	Input	0.0000	0.0000	0.0000
21	hw00-00-020	Input	0.0000	0.0000	0.0000
22	hw00-00-021	Input	0.0000	0.0000	0.0000
23	hw00-00-022	Input	0.0000	0.0000	0.0000
24	hw00-00-023	Input	0.0000	0.0000	0.0000
25	hw00-00-024	Input	0.0000	0.0000	0.0000
26	hw00-00-025	Input	0.0000	0.0000	0.0000
27	hw00-00-026	Input	0.0000	0.0000	0.0000
28	hw00-00-027	Input	0.0000	0.0000	0.0000
29	hw00-00-028	Input	0.0000	0.0000	0.0000
30	hw00-00-029	Input	0.0000	0.0000	0.0000
31	hw00-00-030	Input	0.0000	0.0000	0.0000
32	hw00-00-031	Input	0.0000	0.0000	0.0000
33	hw00-00-032	Input	0.0000	0.0000	0.0000
34	hw00-00-033	Input	0.0000	0.0000	0.0000
35	hw00-00-034	Input	0.0000	0.0000	0.0000
36	hw00-00-035	Input	0.0000	0.0000	0.0000
37	hw00-00-036	Input	0.0000	0.0000	0.0000
38	hw00-00-037	Input	0.0000	0.0000	0.0000
39	hw00-00-038	Input	0.0000	0.0000	0.0000
40	hw00-00-039	Input	0.0000	0.0000	0.0000
41	hw00-00-040	Input	0.0000	0.0000	0.0000
42	hw00-00-041	Input	0.0000	0.0000	0.0000
43	hw00-00-042	Input	0.0000	0.0000	0.0000
44	hw00-00-043	Input	0.0000	0.0000	0.0000
45	hw00-00-044	Input	0.0000	0.0000	0.0000
46	hw00-00-045	Input	0.0000	0.0000	0.0000
47	hw00-00-046	Input	0.0000	0.0000	0.0000
48	hw00-00-047	Input	0.0000	0.0000	0.0000
49	hw00-00-048	Input	0.0000	0.0000	0.0000
50	hw00-00-049	Input	0.0000	0.0000	0.0000
51	hw00-00-050	Input	0.0000	0.0000	0.0000
52	hw00-00-051	Input	0.0000	0.0000	0.0000
53	hw00-00-052	Input	0.0000	0.0000	0.0000
54	hw00-00-053	Input	0.0000	0.0000	0.0000
55	hw00-00-054	Input	0.0000	0.0000	0.0000
56	hw00-00-055	Input	0.0000	0.0000	0.0000
57	hw00-00-056	Input	0.0000	0.0000	0.0000
58	hw00-00-057	Input	0.0000	0.0000	0.0000
59	hw00-00-058	Input	0.0000	0.0000	0.0000
60	hw00-00-059	Input	0.0000	0.0000	0.0000
61	hw00-00-060	Input	0.0000	0.0000	0.0000
62	hw00-00-061	Input	0.0000	0.0000	0.0000
63	hw00-00-062	Input	0.0000	0.0000	0.0000
64	hw00-00-063	Input	0.0000	0.0000	0.0000
65	hw00-00-064	Input	0.0000	0.0000	0.0000
66	hw00-00-065	Input	0.0000	0.0000	0.0000
67	hw00-00-066	Input	0.0000	0.0000	0.0000
68	hw00-00-067	Input	0.0000	0.0000	0.0000
69	hw00-00-068	Input	0.0000	0.0000	0.0000
70	hw00-00-069	Input	0.0000	0.0000	0.0000
71	hw00-00-070	Input	0.0000	0.0000	0.0000
72	hw00-00-071	Input	0.0000	0.0000	0.0000
73	hw00-00-072	Input	0.0000	0.0000	0.0000
74	hw00-00-073	Input	0.0000	0.0000	0.0000
75	hw00-00-074	Input	0.0000	0.0000	0.0000
76	hw00-00-075	Input	0.0000	0.0000	0.0000
77	hw00-00-076	Input	0.0000	0.0000	0.0000
78	hw00-00-077	Input	0.0000	0.0000	0.0000
79	hw00-00-078	Input	0.0000	0.0000	0.0000
80	hw00-00-079	Input	0.0000	0.0000	0.0000
81	hw00-00-080	Input	0.0000	0.0000	0.0000
82	hw00-00-081	Input	0.0000	0.0000	0.0000
83	hw00-00-082	Input	0.0000	0.0000	0.0000
84	hw00-00-083	Input	0.0000	0.0000	0.0000
85	hw00-00-084	Input	0.0000	0.0000	0.0000
86	hw00-00-085	Input	0.0000	0.0000	0.0000
87	hw00-00-086	Input	0.0000	0.0000	0.0000
88	hw00-00-087	Input	0.0000	0.0000	0.0000
89	hw00-00-088	Input	0.0000	0.0000	0.0000
90	hw00-00-089	Input	0.0000	0.0000	0.0000
91	hw00-00-090	Input	0.0000	0.0000	0.0000
92	hw00-00-091	Input	0.0000	0.0000	0.0000
93	hw00-00-092	Input	0.0000	0.0000	0.0000
94	hw00-00-093	Input	0.0000	0.0000	0.0000
95	hw00-00-094	Input	0.0000	0.0000	0.0000
96	hw00-00-095	Input	0.0000	0.0000	0.0000
97	hw00-00-096	Input	0.0000	0.0000	0.0000
98	hw00-00-097	Input	0.0000	0.0000	0.0000
99	hw00-00-098	Input	0.0000	0.0000	0.0000
100	hw00-00-099	Input	0.0000	0.0000	0.0000
101	hw00-00-100	Input	0.0000	0.0000	0.0000

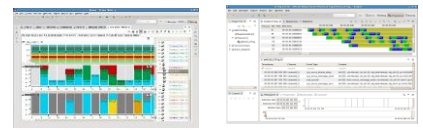
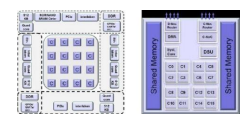
Report

SHIM Info

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

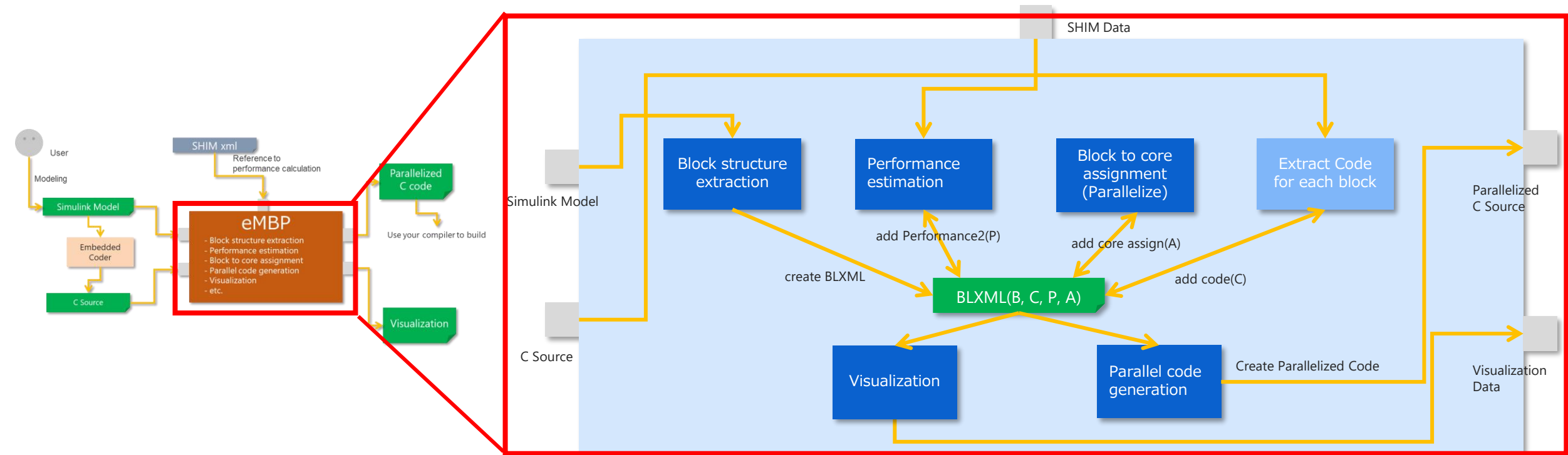
Model Info

Model Name	hw2011.vi
Number of Blocks	112
Number of Nodes	171
Block Type	1.1.1 TDF F R A TF





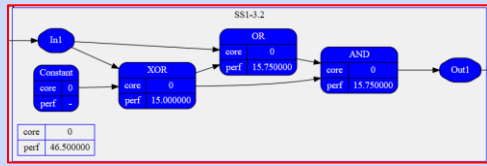
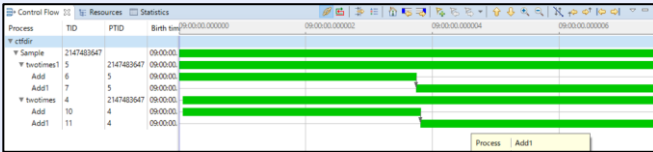
ソフトウェア構成

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



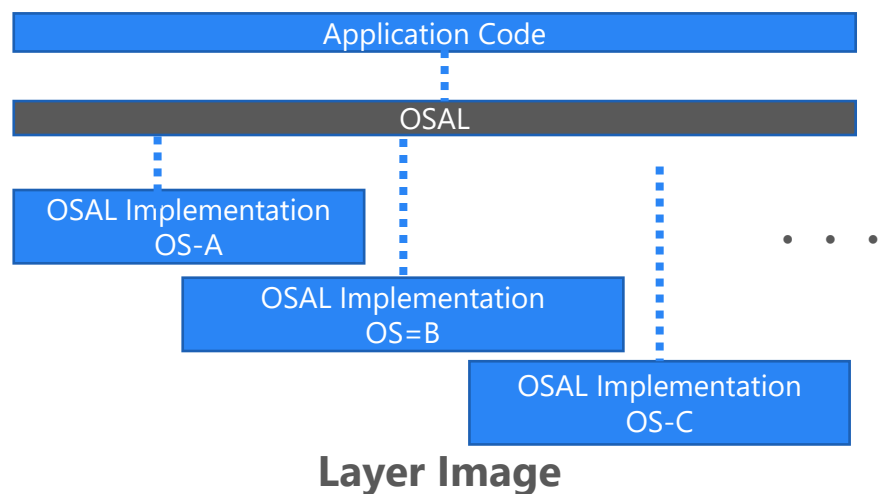
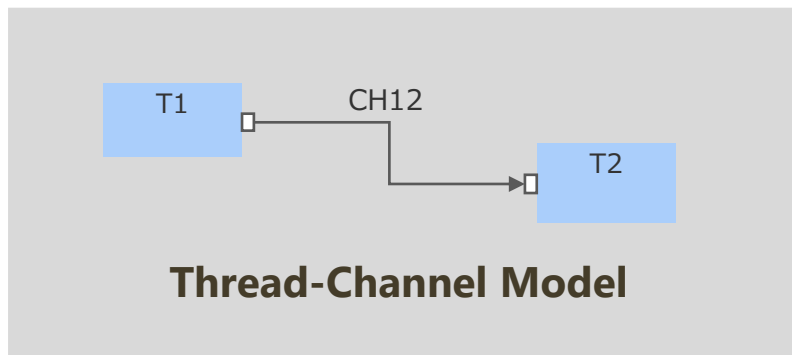
BLXMLの各段階	状態
BLXML(B)	Simulinkモデルから ブロック構造 抽出した初期状態
BLXML(B,C)	ブロックに対応するCの ソースコード片 が付加された状態
BLXML(B,C,P)	ブロック毎に見積もられた 性能情報 が付加された状態
BLXML(B,C,P,A)	ブロック毎の コア割り当て情報 が付加された状態

eMBP基本機能

機能	INPUT	OUTPUT	説明																								
ブロック構造抽出	SimulinkModel	BLXML(B), C-Code	Simulinkブロック構造を抽出したグラフ、C-CodeはE.C生成 																								
ブロック性能見積もり	BLXML(B),C-Code, SHIM	BLXML(B,C,P)	ブロック毎のコード片をSHIMデータを使って見積もった結果  <table border="1" data-bbox="1770 564 2051 685"> <caption>各ブロックの性能情報 (BLXML内)</caption> <thead> <tr> <th>No.</th> <th>Block Name</th> <th>Block Type</th> <th>Performance (cycle)</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>Block1</td> <td>Add</td> <td>xxxxxxxx</td> </tr> <tr> <td>1</td> <td>Block2</td> <td>Xor</td> <td>xxxxxxxx</td> </tr> <tr> <td>2</td> <td>Block3</td> <td>Subsystem</td> <td>xxxxxxxx</td> </tr> <tr> <td>3</td> <td>Block4</td> <td>Add</td> <td>xxxxxxxx</td> </tr> <tr> <td>4</td> <td>Block5</td> <td>Input</td> <td>xxxxxxxx</td> </tr> </tbody> </table>	No.	Block Name	Block Type	Performance (cycle)	0	Block1	Add	xxxxxxxx	1	Block2	Xor	xxxxxxxx	2	Block3	Subsystem	xxxxxxxx	3	Block4	Add	xxxxxxxx	4	Block5	Input	xxxxxxxx
No.	Block Name	Block Type	Performance (cycle)																								
0	Block1	Add	xxxxxxxx																								
1	Block2	Xor	xxxxxxxx																								
2	Block3	Subsystem	xxxxxxxx																								
3	Block4	Add	xxxxxxxx																								
4	Block5	Input	xxxxxxxx																								
コア割り当て	BLXML(B,C,P)	BLXML(B,C,P,A)	「階層クラスタリング」(名大)アルゴリズムによる割り当て																								
並列化コード生成	BLXML(B,C,P,A)	Parallelized-C-Code	OSAL(OS-Abstraction Layer)のAPIをCallするコードの生成																								
可視化(割り当て結果)	可視化用情報	コア割り当て結果グラフ	Graphvizベースのグラフ配置画像をSVGで出力し、Webブラウザで表示 																								
可視化(スケジュール)	可視化用情報	スケジュール・ガントチャート	CTF情報を出力し、外部ツールTraceCompassで表示 																								

[コード生成補足]eMBP OSAL Concept

スレッド生成、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。



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_rcv()	Receive Data using channel

} Thread関連

} 通信関連

eMBP OSAL-APIs

■ 開発中/開発検討している新機能

開発中・検討中機能

- 今後開発を検討している機能
- SHIMを使った性能見積り精度の向上
- 形式検証による不具合検出
- Software Mapping Toolへの進化

開発中/開発検討している新機能

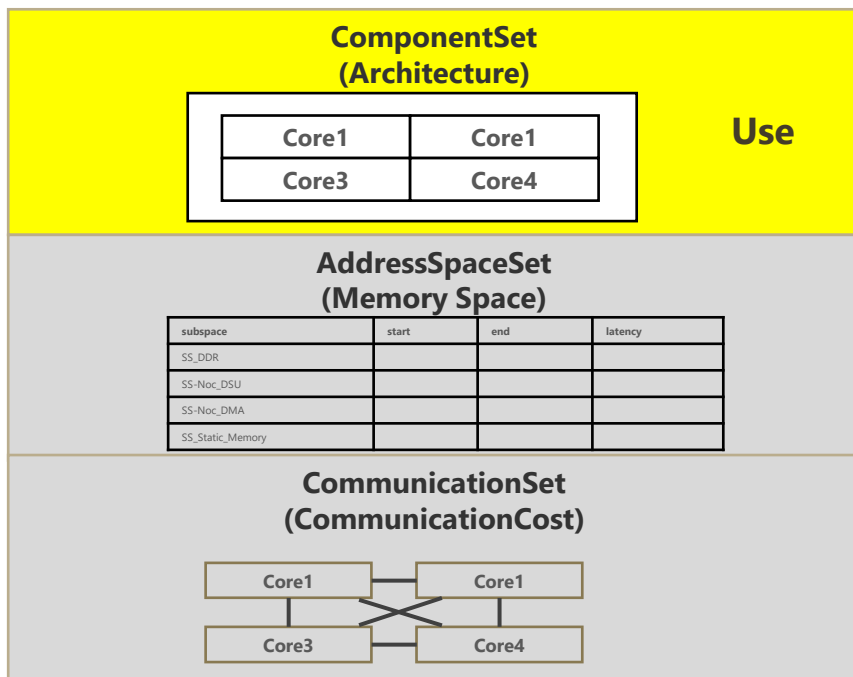
- SHIMを使った性能見積もり精度の向上(開発中)
- 生成コードのPOSIX API対応(開発中)
- ヘテロジニアスマルチコア対応(開発中)
- 形式検証による不具合検出(検討中)
 - ・ 「並列化に伴う課題/並行プログラムの動作保証」への対応
- 他ツールとの連携(検討中)
 - ・ 同様の領域を扱うツール群とのツールチェーン
- ParallelizerからSoftwareMappingToolへの進化(検討中)
- その他、継続的な機能改善開発(継続中)
 - ・ 対応要素(Simulinkブロック)を増やす
 - ・ 継続的なUI改善
 - ・ より効果的な可視化表現 [色付き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

SHIMを使った性能見積もり精度の向上

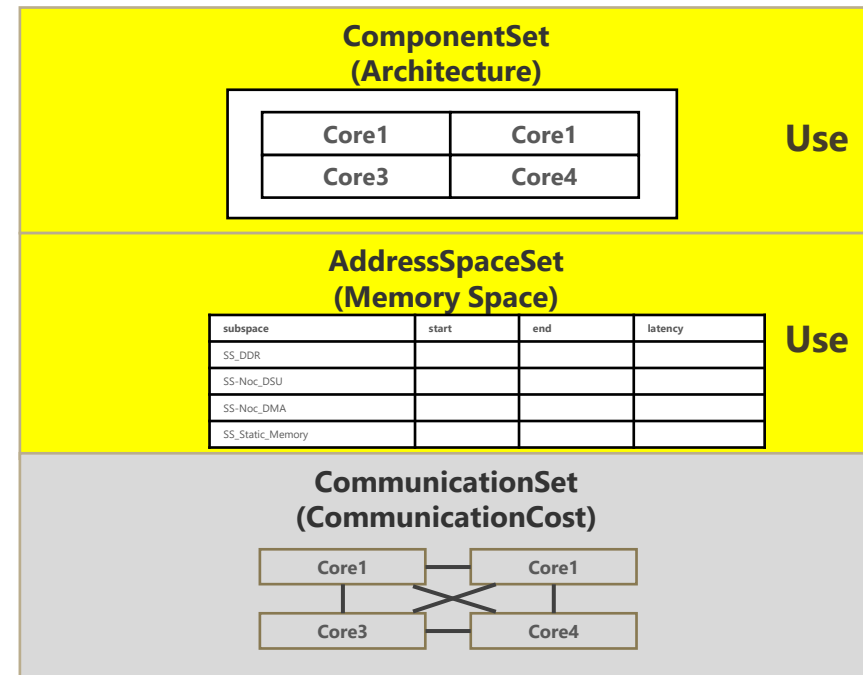
SHIMの情報の有効活用

- これまではSHIM要素のグループ「ComponentSet」のみ使用
- 精度向上にあたって、**メモリアクセス性能の考慮**をするため「AddressSpaceSet」の利用
- メモリアクセス性能を計算する際には**Cache効果**等も考慮する

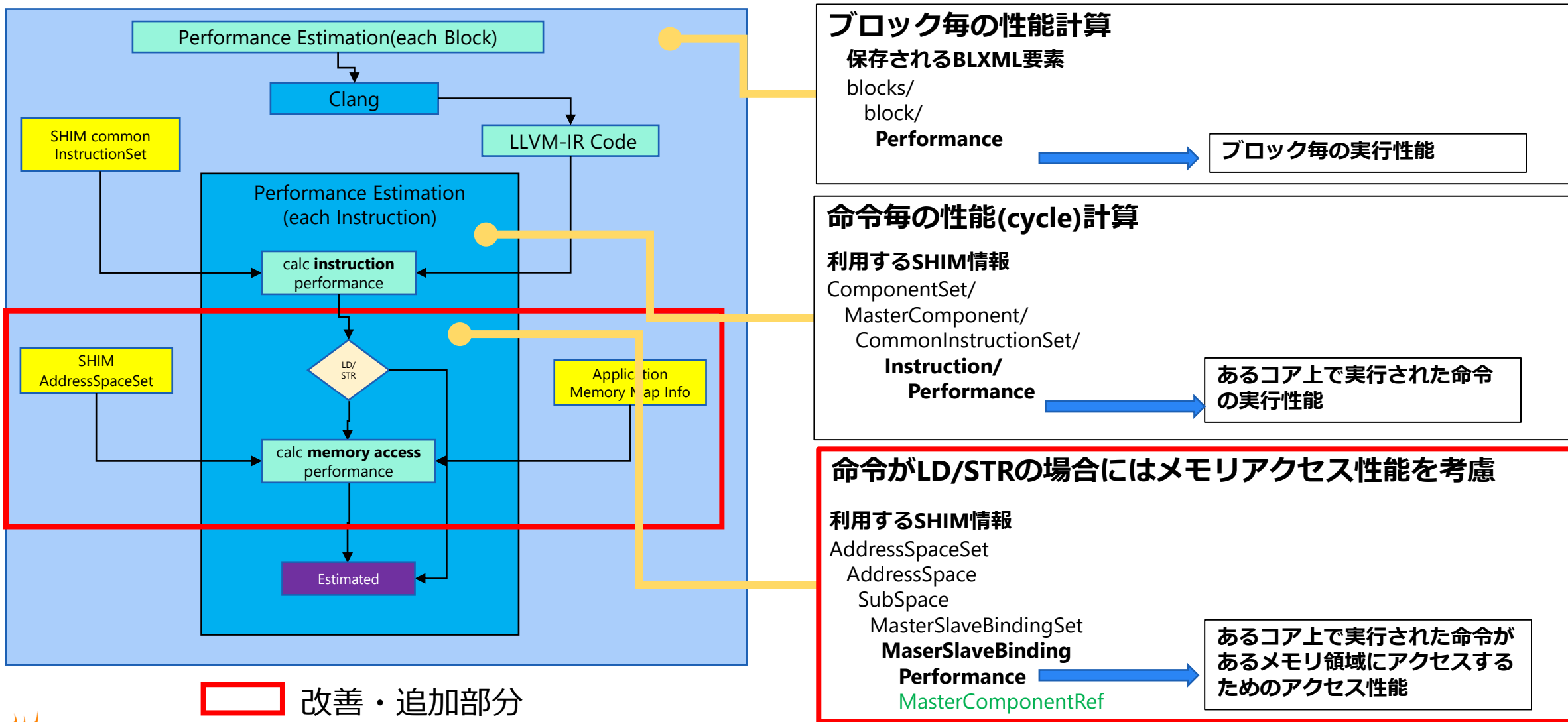
Current



Improved

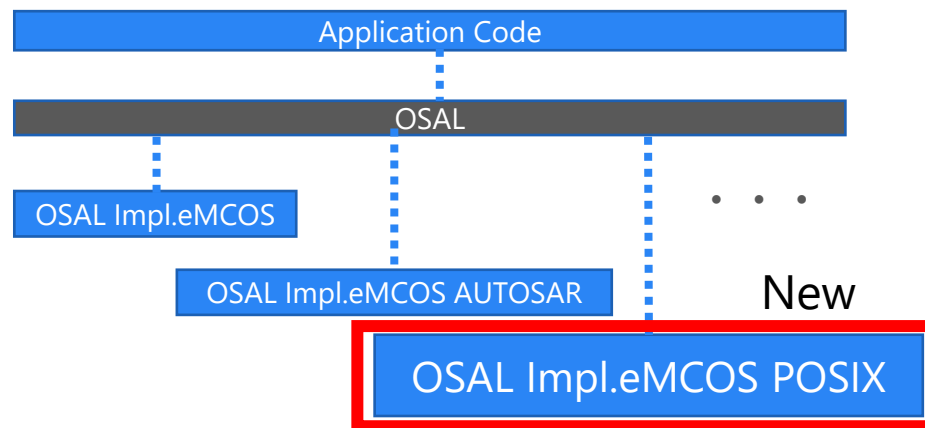


SHIMによる性能見積もり精度の向上 (詳細)



eMBP POSIX OSAL Implementation

C言語/Unixでの標準API仕様 POSIXで定義されているThreadAPI(POSIX-Thread)でのeMBP OSAL実装。これによって「POSIX準拠」のOS環境ではどこでも動作させることが可能になる (※)



※ただし、mbp_thread_create()の実装時に必要な、特定のコアへの固定割り当て(Core-Affinity)指定に関しては、OS依存のAPIを利用した実装(pthread_*_np())になる。

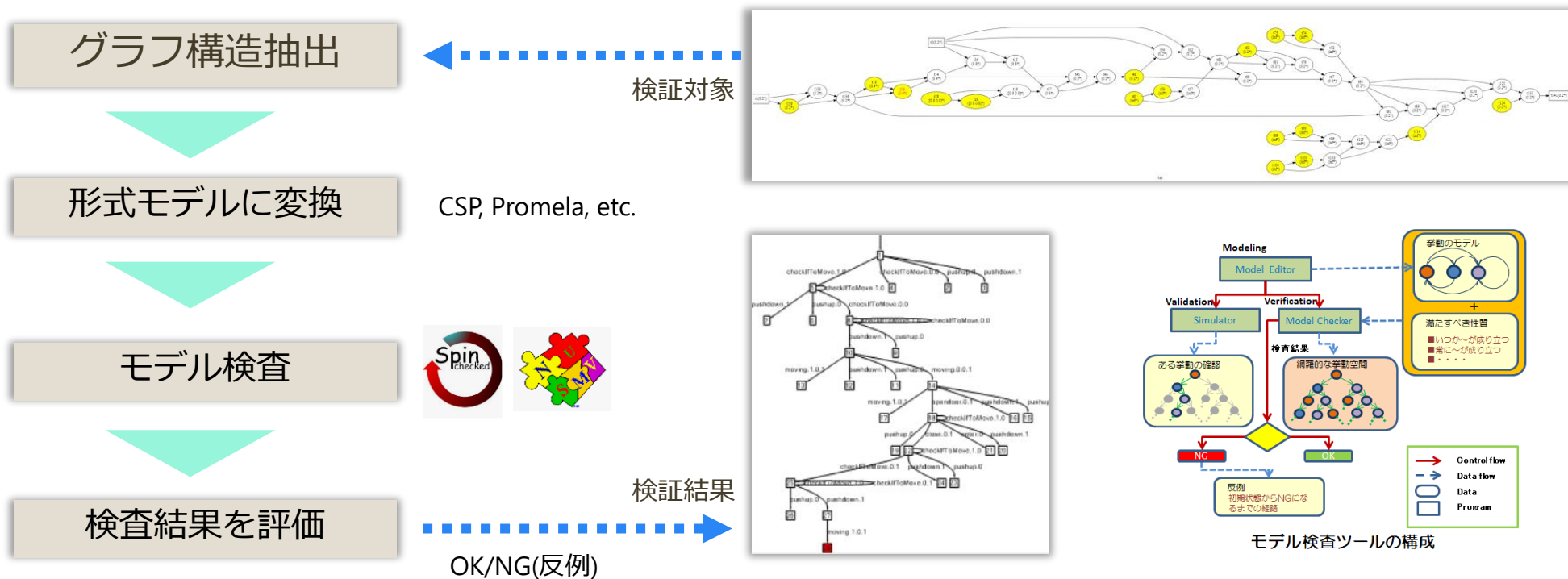
```
Linux -> pthread_setaffinity_np()
eMCOS -> pthread_attr_setlcid_np()
```

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



ParallelizerからSoftwareMappingToolへの進化(※)

■ 現状のeMBP (Parallelizer)

Simulinkモデル(Cコード)

→ M個のスレッド化されたコード : N個のコア

割り当て問題を解き、並列化ソフトウェアを生成。

■ 一般化 (Software Mapping)

ソフトウェアモデル

→ M個のソフトウェアモジュール : N個のハードウェアユニット

マッピング問題を解き、分散・協調ソフトウェアを生成

■ 例

● ソフトウェアモデルの例

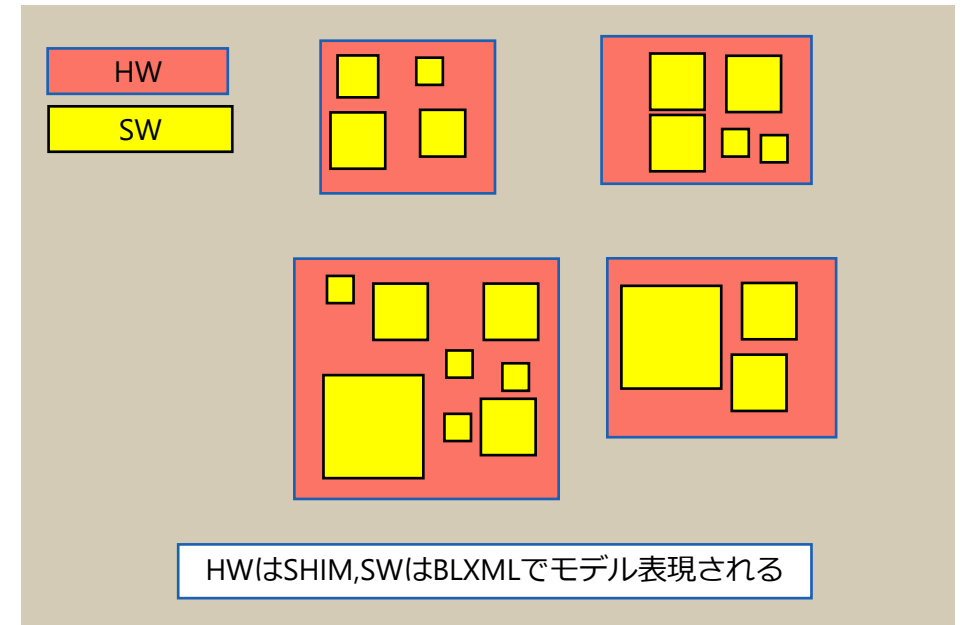
SysMLモデル、Amaltheaモデル、AUTOSARコンポーネント、他

● ソフトウェアモジュールの例

Linuxプロセス、車載・自動運転モジュール、その他高性能計算モジュール

● ハードウェアユニットの例

FPGUボード、GPU、ネットワークに分散されたノード、他



※ NEDO委託研究「高効率・高速処理を可能とするAIチップ・次世代コンピューティングの技術開発／革新的AIエッジコンピューティング技術の開発／スケーラブルなエッジHPCを実現するOS統合型プラットフォームの研究開発」実施項目として検討

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ェ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

- 機能と特徴
- eMBPの外観
- 利用シナリオ・ツール操作概要
- ソフトウェアの構成
- 基本機能の説明

■ 開発中/開発検討している新機能

- SHIMによる見積もり精度の向上
- POSIX OSAL実装
- 形式検証による不具合検出
- Software Mapping Toolへの進化

■ 参考資料

補足情報

- 参考サイト
- 自動並列化ツール例
- SHIM(一般、eMBP利用部分)
- デザインパターン
- モデル検査技術
- モデルベース開発
- トレースフォーマットCTF

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>

(参考資料1) 自動並列化ツールの例

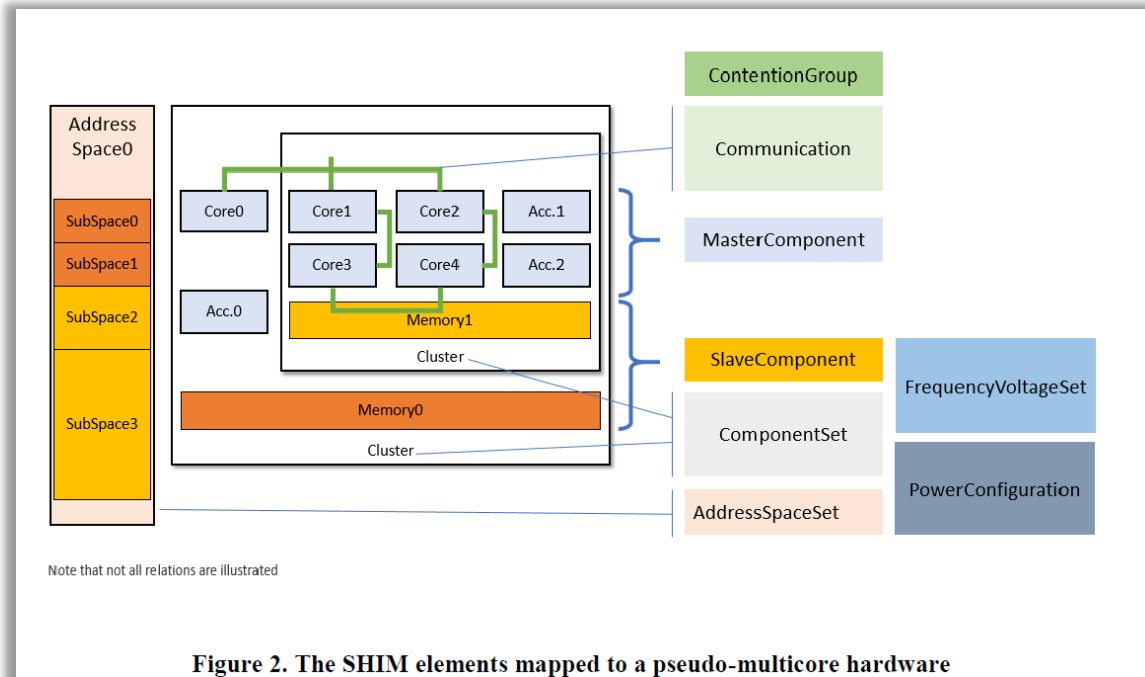
- OSCAR 自動並列化コンパイラ(Waseda Univ.)
 - マルチ・グレイン並列化
 - 入力はC言語プログラム
 - OSCAR APIによる並列構造表現
- モデルベース並列化(Nagoya Univ.) --- eMBPのベース
 - 「2重階層クラスタリング」
 - 入力はSimulinkモデル(と、モデルから生成されるコード)
 - BLXMLによるブロック構造表現

(参考資料2)SHIM

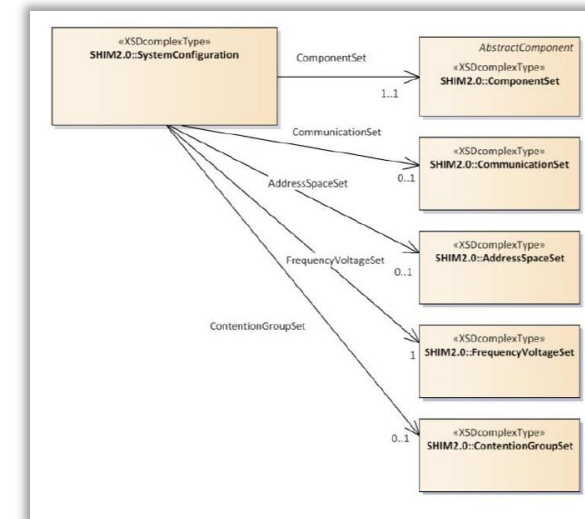
■Software-Hardware Interface for MULTI-MANY core

- 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

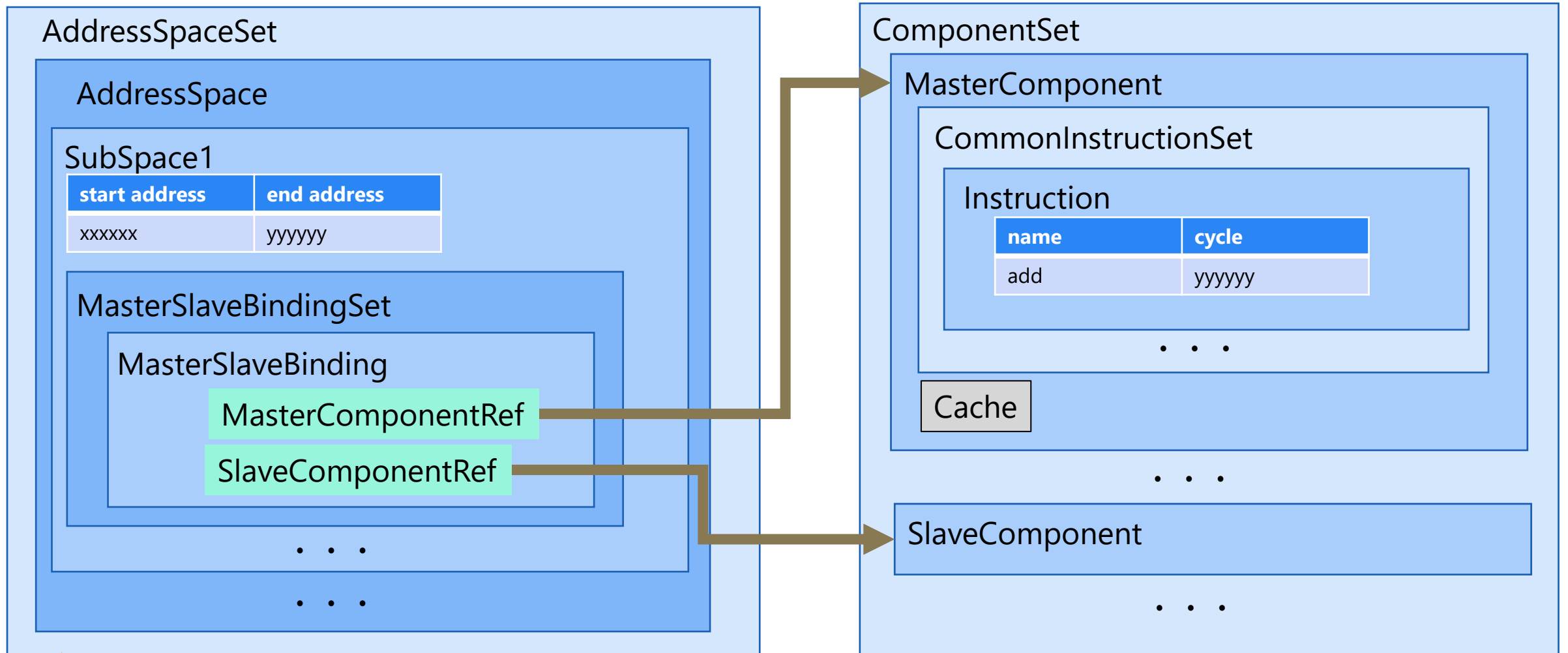


SHIM概念図



SHIM Schema TopLevel

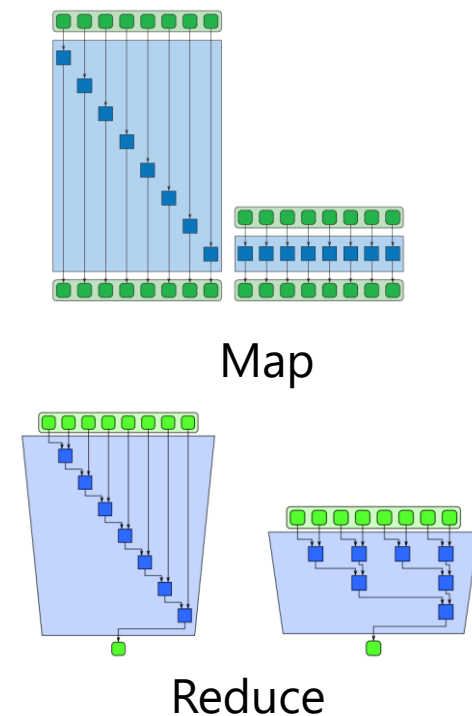
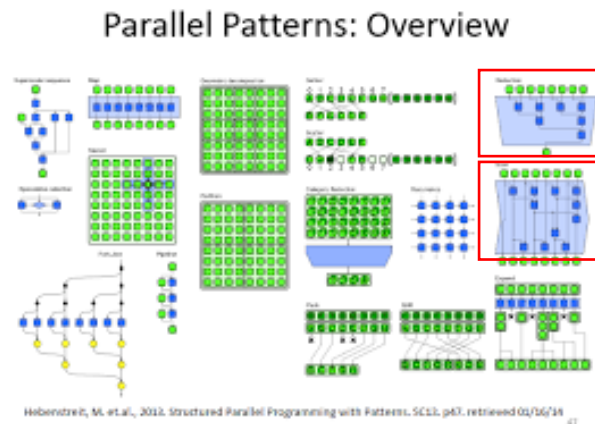
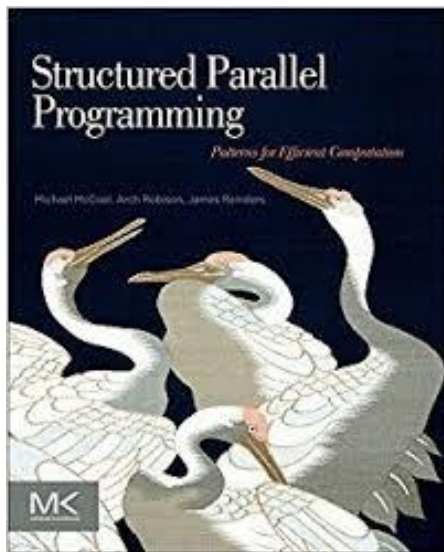
(参考資料3)eMBPが利用するSHIM要素概略図



(参考資料4) 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

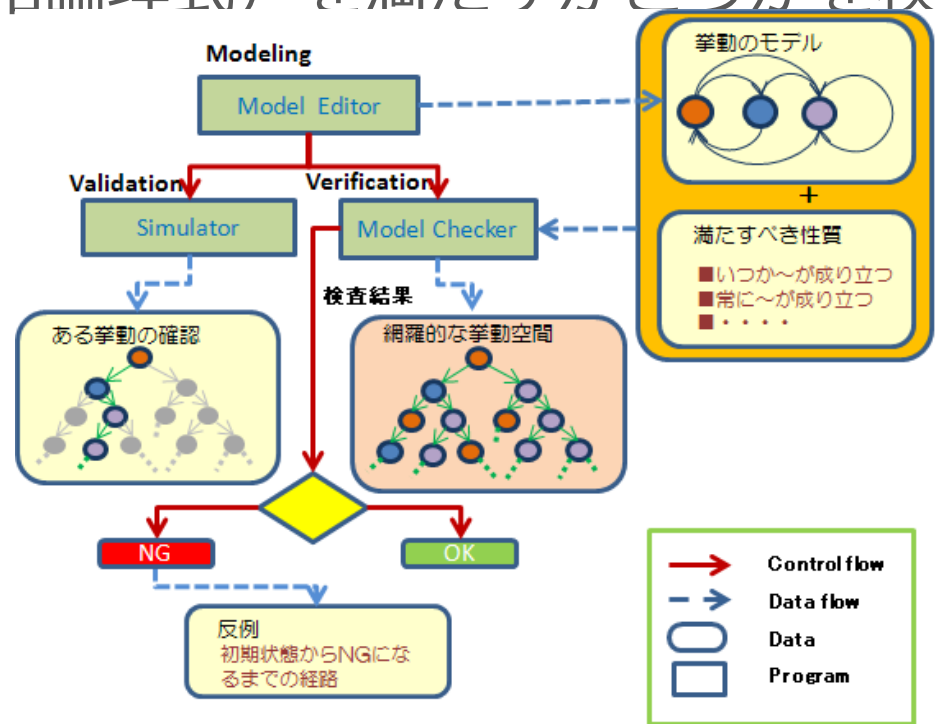


(参考資料5)モデル検査技術

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けられる
- 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/

(参考資料6)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。



(参考資料7) トレースフォーマットCTF

■ Common Trace Format

- <https://www.efficos.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

EfficiOS Operating System Efficiency Services and Consulting
EfficiOS specialises in open source operating systems and performance analysis tools research and development.

HOME SERVICES PROJECTS PUBLICATIONS CONTACT US ABOUT EFFICIOS

Common Trace Format (CTF)

The common trace format specifies a trace format based on the requirements of the industry (through collaboration with the [Multicore Association](#)) and the Linux community. We propose a subset of CTF for use with Linux as a reference.

BabelTrace is a trace conversion library between CTF and other trace formats, which serves as a reference implementation of the Linux trace format.

- The CTF documents are available in this git tree: [Common Trace Format \(CTF\) git tree](#)
 - [Common Trace Format Requirements](#)
 - [Common Trace Format Specification](#)
- The BabelTrace trace conversion source code (and CTF reference implementation) is available in this git tree: [BabelTrace git tree](#)

Copyright © 2016, EfficiOS Inc.

Tracing TraceCompass with JUL

Progress Report Meeting:
Socle Polytechnique de Montréal
December 12, 2016

Genevieve Basile
Research Associate
Dorsal Laboratory
École Polytechnique de Montréal

TRACE COMPASS

Legend

- UNKNOWN
- WAIT_BLOCKED
- WAIT_FOR_CPU
- USERMODE
- SYSCALL
- INTERRUPTED

Process States

Selection: 184309810380-184309810400

Window Span: 000.000 040 672

184309810380-184309810400

ust/uid/1000/64-bit kernel

Challenge With Passion

