モデルベース並列化ツールを用いたマルチコアシステム開発フローの提案

生沼 正博[1,3,a)]　山本 椋太[1]　竹内 成樹[3]　権藤 正樹[4]　本田 晋也[2]　近藤 真己[5]　枝廣 正人[1]

概要：近年，マルチコアシステム開発の需要が高まっており，また，モデルベース開発（MBD）による開発の効率化が求められている．我々の研究グループでは，モデルベース並列化（MBP）環境の研究開発を進めており，成果物として並列化支援のためのツール（MBP ツール）がある．しかし，MBP のための開発フローが定義されていない．そこで，本稿では，MBD におけるマルチコア開発の問題点を整理し，その問題点を解決するための，MBP ツールを用いた開発フローを示す．評価として，提案する開発フローに基づいて，MBP ツールを用いた並列化と手作業による並列化を試みた．その結果，提案する開発フローに対し，MBP ツールは開発を支援できることが分かり，手作業による並列化に比べ工数は約 10 倍良い結果となった．

Multicore System Development Flow with Model Based Parallelization Tools

Masahiro Oinuma[1,3,a)]　Ryota Yamamoto[1]　Shigeki Takeuchi[3]　Masaki Gondo[4]　Shinya Honda[2]
Masaki Kondou[5]　Masato Edahiro[1]

Abstract: In recent years, the demand for multi-core system development is increasing, and the efficiency of development by model-based development is required. We are advancing research on model-based parallelization (MBP), and develop the tools for supporting parallelization (MBP tools). However, the development flow for MBP has not been defined. Therefore, in this paper, we summarize the current problems in MBD and show the development flow with MBP tools for improving these problems. Based on the development flow, we attempted parallelization using MBP tools and manual parallelization for a small model. The result shows that the MBP tool can support the development of the proposed development flow, and the workload of the proposed tools was about ten times better than manual parallelization.

1. はじめに

現在，車載システムの開発規模は高機能化に伴って増加し続けている [19]．たとえば，エンジン制御においては，低燃費化や排気ガスの低減のために機能が増加し，ソフトウェア規模が 150 万行にも及ぶ [19]．これらの大規模化に伴って演算コストは増加しており，この演算コストはマルチコアプラットフォーム上で動作させることによって高速化し，リアルタイム性を担保することが期待されている．そのため，現在，マルチコアによってシングルコアよりも高い性能を実現することが可能なハードウェアプラットフォームによる研究・開発が進められている．

従来の車載システム開発では，シングルコア上で車載システム開発を行っており，V 字モデルに準ずることができていた．これに倣い，企業においてマルチコアに対応した

[1] 名古屋大学
Nagoya University, Nagoya, Aichi 464–8603, Japan
[2] 南山大学
Nanzan University, Nagoya, Aichi 466–8673, Japan
[3] ガイオ・テクノロジー株式会社
GAIO TECHNOLOGY Co., Ltd., Shinagawa, Tokyo 140–0002, Japan
[4] イーソル株式会社
eSOL Co., Ltd., Nakano, Tokyo 164–8721, Japan
[5] NEC ソリューションイノベータ株式会社
NEC Solution Innovators, Ltd., Kawasaki, Kanagawa 213–8511, Japan
[a)] oinuma@ertl.jp

並列化システムの開発を行う場合，シングルコア上での開発と同様に V 字モデルに沿った開発が行われている．しかし，マルチコアシステムを開発する場合，V 字モデルでは手戻りが生じやすく，効率的に開発を進めることが難しくなる．手戻り要因の例として，マルチコア開発による性能見積の困難さがある．マルチコアをターゲットとする目的は高速化であり，要求される性能を満たしている必要がある．要求されている性能を満たしていない場合には，手戻りが発生する．

我々の研究グループでは，モデルベース開発 (MBD) による並列化システム開発支援ツールである MBP（Model Based Parallelization）ツールの開発を進めてきた．しかし，MBP ツールの開発フローは整理されておらず，MBP ツールを用いたマルチコアシステムの開発が有効かどうか，実証されていない．そこで，本稿では，想定する開発フローについて述べ，開発フローを実現するために MBP ツールを用いることで，効率化することを実験によって明らかにする．

2. 車載システム開発における MBD

車載システム開発は OEM(自動車メーカ) と Supplier(自動車部品メーカ) の協業により開発が行われ，V 字モデル **図 1** に則って進められる [10]，[12]，[13]．**図 2** に車載システム開発における MBD の V 字モデルを示す．

OEM では，システム仕様に基づいて要求仕様，制御モデルおよびテスト計画・実装モデルテスト・統合テスト・システムテストを設計する [10]，[12]．制御モデルの設計には MATLAB/Simulink[9] が用いられ，Simulink モデルを設計する．モデル設計・モデルテストが完了すると，要求仕様，Simulink モデルおよびテスト計画・実装モデルテストを Supplier にリリースする．

Supplier では，これらをもとに ECU へ実装するための開発を行う．リリースされた要求仕様・Simulink モデルから ECU へ実装するための設計を行う．コード生成後，テストを行い，OEM へ納入する [10]，[12]．

OEM では，納入されたソフトウェアに対し統合テスト・システム統合を行いリリースする．

これまでの MBD はシングルコア上で開発が行われてきた．しかし近年，シングルコアの性能向上の限界が近いため，マルチコアの導入が必要とされている．梅田らは，MATLAB/Simulink で設計された Simulink モデルから逐次 C コードを生成し，OSCAR 自動並列化コンパイラで並列化を行い，マルチコア上で動作検証を行った [21]．小川らは，パワートレインマルチコアプログラミングフレームワークを提案し，パワートレインアプリケーションを並列化し，マルチコア上で動作検証を行った [19]．

一方で，マルチコア開発ではシングルコア開発では生じなかった課題が発生する．**図 3** に OEM と Supplier における Simulink モデルとマルチコア配置の例を示す．

OEM では，Simulink モデルを設計するにあたり，シミュレーションベースでの数理モデル設計を行う．このため，新規設計，機能修正や機能追加を行った Simulink モデルが性能を満たしてマルチコア上に配置されるかどうかの判断が困難となる．

Supplier では，ターゲットプラットフォーム上での性能・並列動作未考慮による性能未達時に，生成した逐次 C コードを修正を行わず，Simulink モデルを変更することが原則となっている．これは Simulink モデルと逐次 C コードの一貫性を保つためである．さらにシングルコアにおける逐次実行では生じなかった問題も生じる．並列化によるタスク実行順序の変化・同時動作，通信やタスクスケジューリングによるオーバヘッドの問題が生じる．また，もともと論理的に起こり得たが，物理的には逐次実行であったため偶然起こらなかった問題が生じる．これらの問題が図 2 の統合テストで発見されると，OEM のモデル設計・モデル

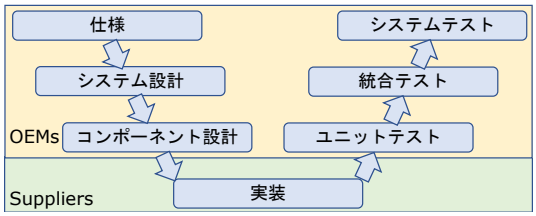


図 1 V 字モデル [12]
Fig. 1 V-Process

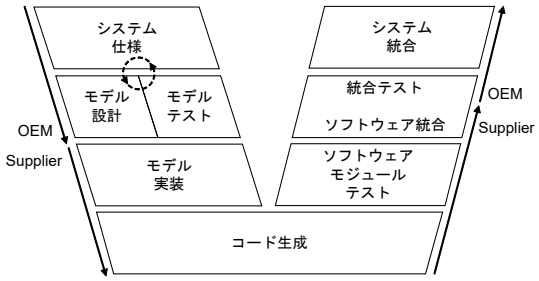


図 2 車載システム開発における MBD の V 字モデル [10]
Fig. 2 MBD V-Process on In-vehicle system Development

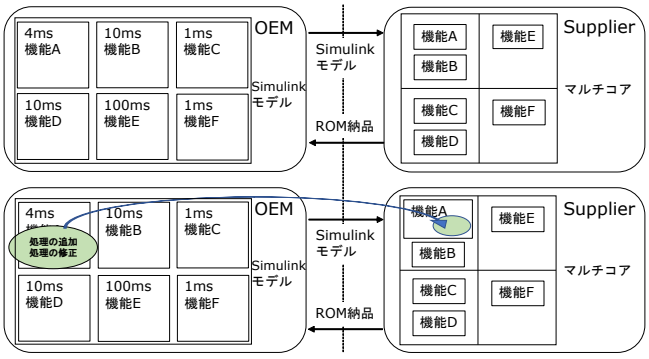


図 3 OEM-Supplier の Simulink モデルとマルチコア配置の例
Fig. 3 Example of simulink model and multicore asign

テストまで手戻りが発生する．以上のことから，マルチコア開発における MBD の課題について述べる．

課題 1　OEM では Simulink モデルが性能を満たしてマルチコア上に配置されるかどうか判断が困難である．

課題 2　Supplier では仕様としてリリースされた Simulink モデルが性能未達かどうかの判断が困難である．

課題 3　OEM では統合テストの段階まで性能の評価が困難である．

課題 4　OEM では統合テストの段階まで並列化特有の問題が生じないか，評価が困難である．

これらの課題を解決するべく，我々は，モデルベース並列化をする手法やツールの研究開発を進めてきた．山口らは Simulink モデルをモデルレベル並列化する提案を行った [28]．そして竹松らがクロスレイヤ設計を提案することでモデルレベルと実装レベルの協調設計が可能となった [18]．また，油谷，鍾らがマルチコア・メニーコアアーキテクチャへのタスクマッピングを提案し，効率的にマルチコアに機能を配置することが可能になった [20]，[22]．

しかし，OEM と Supplier の協業による車載システム開発を行うにあたり，その開発フローが整理されておらず，これらの課題を解決することが困難であった．そこで我々はこれらの課題を解決するために，図 2 をもとに，MBD における並列化システム開発のための開発ステージを定義し，それぞれの開発ステージに対する開発プロセスの提案を行う．これにより，マルチコア対応を段階的に進めながら，並列化システムの開発プロセスに沿った開発が可能となる．

3. MBP 開発フロー

我々が提案する開発ステージと開発プロセスを**図 4** に示す．図 2 を量産開発として見ると，量産開発の前に先行開発を行う．また，マルチコア開発では先行開発を行うための前準備が必要であると考えている．これは Simulink モデルの処理性能見積を行うことで，マルチコアに配置するための指標を得るためである．マルチコア開発における先行開発の前準備，先行開発，量産開発の 3 段階を開発ステージとし，それぞれをマルチコア移行準備開発ステージ，先行開発ステージ，量産開発ステージとしている．マルチコアを対象とした並列処理のアプリケーション開発では，以下の要件が必要であると考える．

- 開発の早い段階で性能見積ができること．
- 性能見積にもとづいた並列化性能設計が可能なこと．
- 並列処理のアプリケーションが並列化特有の問題を生じることなく動作すること．

そこで図 2 におけるシステム仕様とモデル設計・モデルテスト間のイテレーションをもとにし，マルチコア移行準備開発ステージおよび先行開発ステージにおいて，これらの要件を考慮した開発プロセスを行う．量産開発ステージでは，**課題 1** から**課題 4** を解決した状態で並列実装を行い，製品リリースとなる．以降，各開発ステージに分けて説明する．

3.1 マルチコア移行準備開発ステージと開発プロセス

マルチコア移行準備開発ステージでは，Simulink モデルを設計し，シングルコアでどの程度処理性能を有するのか調査を行う．調査の結果得られた処理性能は，マルチコアに配置するときの指標となる．

表 1 に本開発ステージにおけるモデルベース並列化環境の開発プロセスを示す．移行準備開発ステージでは，使用する CPU や回路基板が存在しない状態で開発を行うことが多い．従って，処理性能の測定には，命令セットシミュレータなどの実機レス環境を使用する．精度誤差の許容範囲を定め，使用予定の CPU 命令セット処理時間およびメモリレイテンシを設定する．設定した値を元に設計した Simulink モデルの処理性能を見積もる．

3.2 先行開発ステージと開発プロセス

先行開発ステージでは設計した Simulink モデルをマルチコアに配置し，並列性能検証を行う．並列性能検証の目的は，量産開発へ並列時の課題を持ち越さないことである．

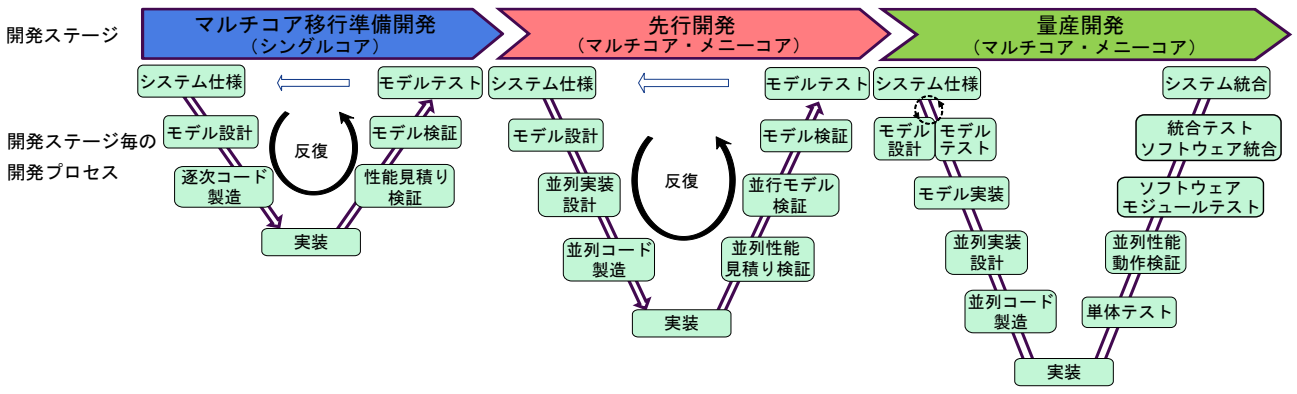


図 4　提案する開発ステージと開発プロセス
Fig. 4　state and process

表 1 マルチコア移行準備開発ステージにおけるモデルベース並列化環境の開発プロセス

開発ステージ	開発プロセス	ツール	説明
マルチコア移行準備開発 (シングルコア)	システム仕様	システム仕様記述ツール	制御の仕様を作成する.
	モデル設計	**並列化対応モデル生成**[28]・MATLAB/Simulink[9]	Simulink モデルを作成し，並列化対応モデル生成機能により 並列化対応モデルを作成する.
	逐次コード製造	Embedded Coder[9]	並列化対応モデルを Embedded Coder に入力し，逐次 C コードを生成する.
	実装	**サイクル計測**[5][14][27][24]・コンパイラ・ISS	サイクル計測用ソースをコンパイルし，ISS でサイクル計測を行う．その結果を SHIM に反映する.
	性能見積り検証	**LLVM-IR 性能見積り**[5][14][27][24]・clang[2]	逐次 C コードを clang でコンパイルし，LLVM-IR と SHIM を用いて性能を見積もる.
	モデル検証	**モデルブロック検証・モデルブロック 逐次 C コード検証**[23]・B2B テストツール	Simulink モデルと並列化対応モデルの一致性検証を行う.
	モデルテスト	モデルテストツール	モデルのテストを行う.

ISS…Instruction Set Simulator，LLVM-IR[8]…Low Level Virtual Machine-Intermediate Representation，SHIM…Software-Hardware Interface for Multi-many-core
B2B…Back to Back
下線太字はモデルベース並列化ツールを表す。

表 2 先行開発ステージにおけるモデルベース並列化環境の開発プロセス

開発ステージ	開発プロセス	ツール	説明
先行開発 (マルチコア・メニーコア)	システム仕様	システム仕様記述ツール	制御の仕様を作成する.
	モデル設計	**BL 構造取得**[18][28]・MATLAB/Simulink[9]・Embedded Coder[9]	機能を Simulink モデルに追加し，BL 構造取得機能により BLXML および並列化対応モデルを生成する．並列化対応モデルを Embedded Coder に入力し，逐次コードを作成する.
	並列実装設計	**ブロック処理コード付加**[18][28]	BLXML に逐次コードを付加する.
		抽出コード処理量見積り[5][18][28]	BLXML と SHIM を用いて付加した逐次コードの処理量を見積もる.
		BL タスクマッピング[18][20][22][28]	自動でコア割り当て情報を BLXML に付加する.
	並列コード製造	**並列 C コード生成**[18][28]	BLXML を用いて並列 C コードを生成する.
	実装	コンパイラ	並列 C コードをコンパイルし，実行バイナリを生成する.
	並列性能見積り検証	**並列性能実機検証ツール**[25]・CDT[1]	実行バイナリの並列動作検証を行い，並列性能結果を生成する.
	並行モデル検証	**CSP 並行モデル生成**[16][17][26]・PAT/FDR[11][4]	CSP 並行モデル生成機能により BLXML から CSP ファイルを生成し，並行動作検証を行う.
	モデル検証	**モデルブロック検証・モデルブロック 逐次 C コード検証**[23]・B2B テストツール	Simulink モデルと並列化対応モデルの一致性検証を行う.
	モデルテスト	モデルテストツール	モデルのテストを行う.

BL…Block Level，BLXML…Block Level Extensible Markup Language，SHIM…Software-Hardware Interface for Multi-many-core，CDT…C/C++ Development Toolkit
CSP…Communicating sequential processes，PAT…Process Analysis Toolkit，FDR…Failures-Divergences Refinement，B2B…Back to Back
下線太字はモデルベース並列化ツールを表す。

表 2 に本開発ステージにおけるモデルベース並列化環境の開発プロセスを示す．先行開発ステージでは，使用する CPU や回路基板が存在しない状況から存在する状況まで，さまざまな状況が存在している．従ってマルチコア移行準備開発ステージで準備した性能見積をもとに，並列実装設計のプロセスで Simulink モデルをマルチコアへ自動的に配置する．そして，並列性能見積り検証のプロセスでは，マルチコアへ割り当てた並列処理が設計通りの性能か検証する．並行モデル検証のプロセスでは，デッドロック，ライブロックや実行順序逆転が発生していないことを検証する．このモデル検証のプロセスを，上流工程において実施することも考えられる．しかし先行開発ステージでは，必要な性能をみたしていることに重点を置いているため，下流工程にて実施している．先行開発ステージにおいてモデル検証を行う理由は，ステージで完結した開発を行うため，およびこの時点でデッドロックといった上述の問題が生じないことを確認する必要があるためである．

3.3 量産開発ステージと開発プロセス

量産開発ステージでは先行開発ステージで設計した Simulink モデルを基に，製品にするための機能を追加し，製品をリリースする．

表 3 に本開発ステージにおけるモデルベース並列化環境の開発プロセスを示す．モデル設計およびモデル実装のプロセスでは，製品にするために必要な非機能・機能要件を設計する．そして先行開発ステージにおいて作成した Simulink モデルに機能を追加する．また，量産開発ステージと先行開発ステージの Simulink モデルを用いて，一致性検証も併せて行う．その後，並列実装設計のプロセスでは先行開発ステージで決定した性能見積をそのまま使用して Simulink モデルをマルチコアに配置する．この結果に対し，デッドロック，ライブロックや実行順序逆転が発生しないことを検証する．実装はコード生成機能を利用することで完了し，単体テストのプロセスを実施する．並列性能動作検証のプロセスでは追加した機能が並列性能を低下させないことを検証する．ソフトウェアモジュールテスト，ソフトウェア統合，統合テストおよびシステム統合のプロセスを経て製品リリースとなる．

4. 評価実験

4.1 実験内容

実験の目的は，MBP ツールを用いるマルチコア開発と

表 3 量産開発ステージにおけるモデルベース並列化環境の開発プロセス

開発ステージ	開発プロセス	ツール	説明
量産開発 (マルチコア・メニーコア)	システム仕様	システム仕様記述ツール	制御の仕様を作成する.
	モデル設計 モデルテスト	MATLAB Simulink[9]	ECU としての機能を Simulink モデルに追加する.
	モデル実装	**BL 構造取得**[18][28]・MATLAB/Simulink[9]・Embedded Coder[9]	ECU に実装するために Simulink モデルを修正し，BL 構造取得機能により BLXML および並列化対応モデルを生成する．並列化対応モデルを Embedded Coder に入力し，逐次コードを作成する.
		モデルブロック検証・モデルブロック逐次 C コード検証[23]・B2B テストツール	Simulink モデルと並列化対応モデルの一致性検証を行う.
	並列実装設計	**ブロック処理コード付加**[18][28]	BLXML に逐次コード情報をマッピングする.
		抽出コード処理量見積り[5][18][28]	BLXML と SHIM を用いて付加した逐次コードの処理量を見積もる.
		BL タスクマッピング[18][20][22][28]	自動でコア割り当て情報を BLXML に付加する.
		CSP 並行モデル生成[16][17][26]・PAT/FDR[11][4]	CSP 並行モデル生成機能により BLXML から CSP ファイルを生成し，並行動作検証を行う.
	並列コード製造	**並列 C コード生成**[18][28]	BLXML を用いて並列 C コードを生成する.
	実装	コンパイラ	並列 C コードをコンパイルし，実行バイナリを生成する.
	単体テスト	単体テストツール	単体テストを行う.
	並列性能動作検証	**並列性能実機検証ツール**[25]・CDT[1]	実行バイナリの並列動作検証を行い，並列性能結果を生成する.
	ソフトウェア モジュールテスト	ソフトウェアモジュールテストツール	ソフトウェアモジュールのテストを行う.
	統合テスト ソフトウェア統合	統合テストツール	統合テストを行う.
	システム統合	システム統合テストツール	システム統合テストを行う.

ECU…Electronic Control Unit，BL…Block Level，BLXML…Block Level Extensible Markup Language，SHIM…Software-Hardware Interface for Multi-many-core
CDT…C/C++ Development Toolkit，CSP…Communicating sequential processes，PAT…Process Analysis Toolkit，FDR…Failures-Divergences Refinement
B2B…Back to Back
下線太字はモデルベース並列化ツールを表す。

手作業によるマルチコア開発を比較したとき，以下の観点で MBP ツールの立場を考察することである.

観点 1 MBP ツールを用いた作業および手作業で Simulink によるシミュレーションと同様の結果を得ることができるか.

観点 2 MBP ツールを用いた作業および手作業を比較して工数に差が出るか.

観点 3 MBP ツールを用いた作業および手作業を比較して性能見積に差が出るか.

実験では，第 1 著者によって先行開発ステージのモデル検証を除いたプロセス，すなわちモデル設計・並列実装設計・並列コード製造・実装・並列性能見積り検証のプロセス (以下 MBP フローと呼ぶ) を実施する．第 1 著者は実験時，企業において 20 年以上組込みソフトウェアの開発に従事しており，現在まで 3 年程度マルチコア開発も行っている．また，MATLAB/Simulink の使用方法を十分理解しており，MBP ツールの利用方法を熟知している.

実験では，実験用のモデル[*1]を事前に用意した．このモデルは温度と湿度を制御するモデルである．MATLAB/Simulink のツールの 1 つである Embedded Coder を使用して，このモデルから生成した逐次 C コードに対し並列化を行う.

また，測定に用いたハードウェアおよびソフトウェアについて，以下に述べる.

- 使用したターゲット: KALRAY MPPA Bostan[6] (うち，同一クラスタの 2 コアを使用した．)
- クロック数: 500MHz
- コンパイラ: k1-elf-gcc (4.9.4)
- 使用した OS: eMCOS[3]
- MATLAB/Simulink: 2017a

実験の進め方を以下に示す.

手順 1 手作業による並列化
- **手順 1-1** Embedded coder で逐次コードを生成
- **手順 1-2** 逐次コードの理解
- **手順 1-3** 並列化する箇所を同定
- **手順 1-4** 手動で逐次コードを並列コードに修正
- **手順 1-5** ビルドおよび実行

手順 2 MBP ツールによる並列化
- **手順 2-1** Simulink モデルから BLXML を生成
- **手順 2-2** Embedded coder で逐次コードを生成
- **手順 2-3** 逐次 C コードを BLXML に付加
- **手順 2-4** SHIM を用いたブロック粒度の性能見積もり結果を BLXML に付加
- **手順 2-5** ブロックの自動コア割当て
- **手順 2-6** 並列コードの生成
- **手順 2-7** ターゲットに向けた並列コードの書き換え
- **手順 2-8** ビルドおよび実行

手順 1 においては，**手順 1-1** および**手順 1-5** を除くすべての手順で，手作業である．ただし，ソースコードの修正時にはテキストエディタに付属の文字列検索程度の機能を用いた．また，**手順 2** においては，**手順 2-7** を除くすべての手順で自動である．ただし，コマンド入力程度の操作を行っている．**手順 2-7** においては，キャッシュコヒーレン

[*1] `https://jp.mathworks.com/help/stateflow/examples/home-climate-control-using-the-truth-table-block.html`（2020 年 6 月 4 日参照）

シを考慮した記述を追加している．**手順 1** および**手順 2** の最後の手順の後，実行結果の確認を行っている．比較対象は，Simulink のシミュレーション結果である．結果が一致するまでソースコードの書き換えとビルド・実行を繰り返している．

手作業を先に実施した理由は，MBP ツールによる自動コア割当結果を確認することなく，作業者の考えだけを反映させるためである．また，MBP ツールによる実験は，すべて自動化されている．そのため，MBP ツール利用時に手作業で得た知見は反映されないため，MBP ツールによる並列化を後とした．

4.2 実験結果

第 1 著者による実験の結果を**表 4** に示す．この結果から，今回のモデルに対し，MBP ツールは手作業に対して作業時間は約 1/10 に短縮できた．ソースコード行数は MBP ツールが手作業と比べて約 1.83 倍程度となったが，いずれも変更元である逐次 C コードから比べて増加している．**図 5** に見積性能と実機性能結果を示す．見積性能については，SHIM による見積性能において手作業によるコードよりも MBP ツールによる自動生成コードは約 12%の性能向上であった．実機性能については，手作業によるコードよりも MBP ツールによる自動生成コードは約 27%の性能劣化となった．

表 5 および**表 6** に，作業時間の内訳を示す．表 5 では，作業のうち**手順 2-1** から**手順 2-6** をまとめた時間を示している．**手順 2-1** から**手順 2-6** は，すべて自動化された作業であるが，15 分を要している．この理由として，今回の実験では，ツールによって異なる PC 上で作業をしたためであり，ファイル移動にかかった時間が 14 分程度を占めている．すなわち，**手順 2-1** から**手順 2-6** の本質的なツールの実行時間そのものは 1 分未満であり，小さい時間となるため，これらの手順でかかった時間をまとめて示した．また，ビルド回数については，**手順 1** では 2 回以上，**手順 2** では 1 回のみであった．

表 4 実行結果（概要）

	手作業	MBP ツール	備考
作業時間	9.5 時間	1 時間	休憩時間を除く
コード行数	1,206	2,204	逐次 C コードは 959
見積性能	3.79us	3.39us	SHIM による見積性能
実機性能	2.62us	3.57us	-
コア間通信	6 回	10 回	送信または受信一度で 1 回

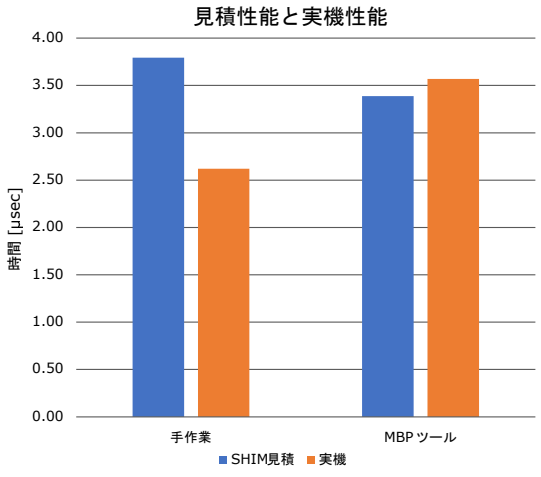


図 5 見積性能と実機性能
Fig. 5 A Result of Performances

表 5 実行結果（**手順 1** の作業時間）

	手順				合計
	1-1	1-2	1-3	1-4	
作業時間（時間）	2.00	4.50	2.00	1.00	9.50

表 6 実行結果（**手順 2** の作業時間）

	手順			合計
	2-1 から 2-6	2-7	2-8	
作業時間（時間）	0.25	0.50	0.25	1.00

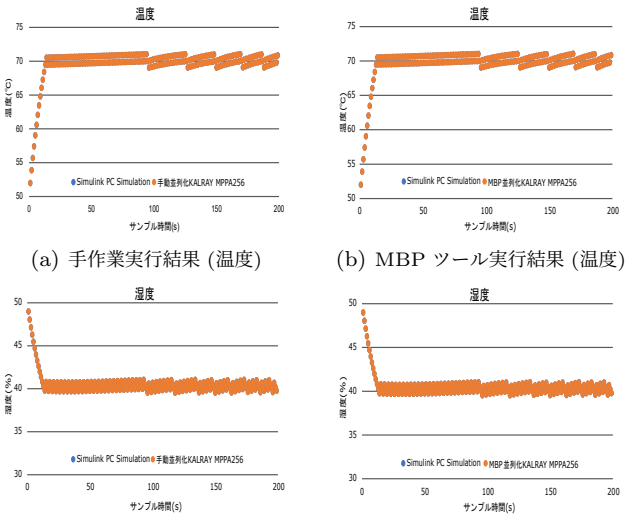
5. 考察

5.1 観点 1: MBP ツールを用いた作業および手作業で Simulink によるシミュレーションと同様の結果を得ることができるか

図 6 に Simulink シミュレーションと実機実行結果を示す．手作業と MBP ツールを用いた実機実行結果は Simulink モデルのシミュレーション結果と一致した．

5.2 観点 2: MBP ツールを用いた作業および手作業を比較して工数に差が出るか

MBP ツールを用いた場合およそ 1 時間で実装を終えた．手作業の場合およそ 9.5 時間で実装を終えた．この結果か



(a) 手作業実行結果 (温度)　(b) MBP ツール実行結果 (温度)
(c) 手作業実行結果 (湿度)　(d) MBP ツール実行結果 (湿度)

図 6 Simulink シミュレーションと実機実行結果
Fig. 6 Simulation and Targetboard execution

ら，MBP ツールを用いた場合，約 10 倍程度短縮できた．

MBP ツールを用いた時の作業時間は 1 時間となっているが，ファイルの移動など本来は必要ない時間を除けば，30 分程度の作業時間となる．この 30 分は，ターゲットシステムのための書き換えとコンパイル時間に用いられており，並列化のための作業時間を大きく減少させることができている．

本稿での対象は，Simulink モデルが 67 ブロック程度，かつターゲットシステムも 2 コアだけであり，Embedded Coder によって生成されたコードも 959 行と，比較的小規模であった．このため手作業では 9.5 時間で実装を終えたが，対象モデルの規模を大きくすることや，ターゲットシステムで利用するコア数を増やすことによって，開発難度が上がると考えている．その結果，手作業では，作業工数が指数関数的に増加することや，そもそも動作させることが困難な状況になると考えられる．

以上から，作業工数は MBP ツールを用いるほうが低減できる．

5.3 観点 3: MBP ツールを用いた作業および手作業を比較して性能見積に差が出るか

SHIM による見積性能において，手作業によるコードよりも MBP ツールによる自動生成コードは約 12%の性能向上であった．しかし，実機性能では，手作業によるコードよりも MBP ツールによる自動生成コードは約 27%の性能劣化となった．解析の結果，性能劣化の原因は，第 1 著者の手作業によるコア間通信タイミング最適化によるものであった．また比較的小規模な Simulink モデルであるため，手作業による高速化についても検討が容易であった．このため手作業によるコードの実機性能が優れる結果になったと考える．

以上から，小規模な Simulink モデルかつ 2 コアの規模が小さいターゲットシステムを想定した場合，MBP ツールを用いた作業のほうがより良い見積性能を得ることができ，手作業のほうがより良い実機性能を得ることができる．

5.4 MBP フローにおける MBP ツール

MBP フローに則って開発する上で，MBP ツールがその支援となるかについて考察する．本稿における実験では，2 つ目のステージである先行開発ステージに適用した．観点 2 の考察から，開発工数の低減に繋げられることがわかった．また，図 4 のうち，並行モデル検証以外のプロセスについて実験を行った．実験を行ってはいないが，並行モデル検証についても MBP ツールに含まれている．以上から，先行開発ステージに対しては，MBP ツールはその支援となっていると考えている．

また，他の開発ステージに対しても検討する．マルチコア移行準備開発ステージでは，逐次コードを対象とするとは言え，性能見積りを実施する必要がある．性能見積りは，MBP ツールが提供しているため，シングルコアの開発においても支援を行うことができると考えている．

量産開発ステージについて，単体テスト，ソフトウェアモジュールテスト，ソフトウェア統合，統合テストおよびシステム統合を除くプロセスに対しては，MBP ツールは支援となる機能を提供している．テストについては，テストの目的によって求められるツールが大きく変わるため，MBP ツールではサポートしない．

以上から，各開発ステージに対して支援となる機能を MBP ツールは持っている．

6. 関連研究

林らは，自動車システム開発に対して，コンカレントフィードバック開発方法を適用し，開発日数の低減を確認した [15]．彼らは，従来，自動車システム開発が要求開発フェーズと製品開発フェーズにわけられて開発が進められていると述べており，本稿における開発ステージの考え方と類似している．また，スパイラルモデルやアジャイル開発を導入することが困難であることについても言及されており，本稿においても同様の主張である．しかし，彼らは，ハードウェアを意識した開発については言及しておらず，Supplier における新規のソフトウェア開発に焦点を当てている．対して，本稿では OEM と Supplier の協業による車載システム開発およびハードウェアアーキテクチャについても考慮した開発フローである．また，MBP ツールでは並列化作業に対して形式手法による検証を支援しており，品質保証も考慮している．

Kienberger は，本稿と同様，MATLAB/Simulink によるモデルから並列化コードを生成するツール群の開発を行った [7]．ツール構成は MBP ツールと近いが，MBP ツールには SHIM を用いた性能見積り機能がある．この機能により，実機レスで開発の早期から自動コア割当機能を適用することができ，MBP ツールを用いることでより効率の良いマルチコアシステムの開発を行うことができる．

7. おわりに

本研究では，モデルベース並列化ツールを用いたマルチコアシステム開発フローを提案した．マルチコア開発を効率化するために 3 つの開発ステージおよびそれぞれの開発プロセスを提案し，それを基にモデルベース並列化環境の開発プロセスを提案した．この開発プロセスには，我々が開発を進めてきた MBP ツールを対応付けた．MBP フローを進めるにあたり，MBP ツールを用いることで工数や性能面で，手作業とどのような差を生じるか調査するため，実験を行った．実験では，モデルベース並列化環境の開発プロセスの要となる MBP フローに基づいて，小規模な Simulink モデル・2 コアのターゲットシステムに対する

マルチコア開発を行った．結果として手作業の場合と比較し，短時間で開発を行うことができた．ただし，実機性能については，小規模な対象に対しては手作業の方が良い結果となった．解析の結果，コア間通信タイミング最適化の違いであることがわかった．

今後の課題として，MBP ツールに対しては大規模モデル，より多くのコアに対しては詳細レベルでのコア間通信タイミング最適化を行う必要がある．また，より大規模な Simulink モデルやターゲットシステムに対して MBP フローや MBP ツールを適用し，同様に性能や工数についての評価を行う必要があると考えている．マルチコア移行準備開発ステージ，先行開発ステージ，量産開発ステージにおけるモデルベース並列化環境によるマルチコア開発の効率化について評価を行っていく．

謝辞　本研究の一部は，国立研究開発法人新エネルギー・産業技術総合開発機構（NEDO）の委託業務の結果得られたものです．

参考文献

[1] CDT: Eclipse CDT (C/C++ Development Tooling), available from <https://www.eclipse.org/cdt>（2020 年 7 月 27 日参照）.

[2] Clang: a C language family frontend for LLVM, available from <https://clang.llvm.org>（2020 年 7 月 27 日参照）.

[3] eSOL: メニーコアプロセッサ対応スケーラブルリアルタイム OS，available from <https://www.esol.co.jp/embedded/emcos.html>（2020 年 7 月 22 日参照）.

[4] FDR: FDR4 — The CSP Refinement Checker，available from <https://cocotec.io/fdr/index.html>（2020 年 7 月 27 日参照）.

[5] Gondo, M., Arakawa, F. and Edahiro, M.: Establishing a standard interface between multi-manycore and software tools - SHIM, *2014 IEEE COOL Chips XVII*, pp. 1–3 (2014).

[6] KALRAY: Processors Bostan, available from <https://www.kalrayinc.com/portfolio/processors>（2020 年 7 月 22 日参照）.

[7] Kienberger, J.: Systematic and Methodical Analysis, Validation and Parallelization of Embedded Automotive Software for Multiple-IEU Platforms, PhD Thesis (2019).

[8] LLVM: The LLVM Compiler Infrastructure, available from <https://llvm.org>（2020 年 7 月 27 日参照）.

[9] MathWorks: Simulink, available from <https://jp.mathworks.com/products/simulink.html>（2020 年 6 月 4 日参照）.

[10] Michailidis, A., Spieth, U., Ringler, T., Hedenetz, B. and Kowalewski, S.: Test front loading in early stages of automotive software development based on AUTOSAR, *2010 Design, Automation & Test in Europe Conference & Exhibition (DATE 2010)*, IEEE, pp. 435–440 (2010).

[11] PAT: PAT:Process Analysis The CSP Refinement Checker, available from <https://pat.comp.nus.edu.sg>（2020 年 7 月 27 日参照）.

[12] Rana, R., Staron, M., Berger, C., Hansson, J., Nilsson, M. and Törner, F.: Increasing efficiency of iso 26262 verification and validation by combining fault injection and mutation testing with model based development, *International Conference on Software Engineering and Applications*, Vol. 2, SCITEPRESS, pp. 251–257 (2013).

[13] Santos, M. M. D., Neme, J. H., Franco, F. R., Stevan Jr, S. L., Torres, W., Lugli, A. B., Laganá, A. A. and Justo, J. F.: Rapid control prototyping for automotive software in power windows systems, *Int. J. Innov. Comput. Inf. Control*, Vol. 11, No. 4, pp. 1–10 (2015).

[14] 鳥越　敬，枝廣正人：ハードウェア抽象化記述 SHIM による性能見積のための LLVM-IR 命令実行時間計測手法，情報処理学会研究報告，Vol. 2020-EMB-53, No. 41, pp. 1–8 (2020).

[15] 林　健吾，青山幹雄，古畑慶次：コンカレントフィードバック開発方法の自動車ソフトウェア開発への適用，情報処理学会論文誌，Vol. 59, No. 4, pp. 1175–1191 (2018).

[16] 多門俊哉，磯部祥尚，枝廣正人：モデルベース並列化アルゴリズムの形式化と正当性の証明，情報処理学会研究報告，Vol. 2019-EMB-50, No. 9, pp. 1–8 (2020).

[17] 山本尚平，鈴木悠太，峰田憲一，森　裕司，枝廣正人：モデルベース並列化における CSP モデルを利用した形式検証の適用，情報処理学会研究報告，Vol. 2017-EMB-44, No. 6, pp. 1–6 (2017).

[18] 竹松慎弥，鍾　兆前，井上雅理，横山静香，小島流石，近藤真己，中本幸一，安積卓也，道木慎二，本田晋也，枝廣正人：モデルベース開発におけるクロスレイヤ設計手法のマルチコア上モータ制御実装への適用，組込みシステムシンポジウム 2017 論文集，Vol. 2017, pp. 110–111 (2017).

[19] 小川真彩高，本田晋也，高田広章：車載制御システム向けマルチコアプログラミングフレームワーク，情報処理学会論文誌，Vol. 58, No. 2, pp. 507–520 (2017).

[20] 油谷　創，枝廣正人：階層構造を持つメニーコアアーキテクチャへのタスクマッピング，情報処理学会論文誌，Vol. 56, No. 8, pp. 1568–1581 (2015).

[21] 梅田　弾，金羽木洋平，見神広紀，林　明宏，谷　充弘，森　裕司，木村啓二，笠原博徳：MATLAB/Simulink で設計されたエンジン制御 C コードのマルチコア用自動並列化，情報処理学会論文誌，Vol. 55, No. 8, pp. 1817–1829 (2014).

[22] 鍾　兆前，枝廣正人：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, No. 2, pp. 735–747 (2018).

[23] 福島直人：モデルベース並列化におけるモデル分析手法の検討，名古屋大学工学部電気電子情報工学科 情報工学コース 卒業論文 http://ertl.jp/~muku/pdf/naoto_fk-thesis.pdf (2019).

[24] 佐合　惇，枝廣正人：ハードウェア抽象化記述 SHIM と SHIMulator によるソフトウェア動的性能見積手法，情報処理学会研究報告，Vol. 2019-EMB-50, No. 14, pp. 1–8 (2019).

[25] 後藤隼弐，本田晋也，長尾卓哉，高田広章：トレースログ可視化ツール TraceLogVisualizer (TLV)，コンピュータソフトウェア，Vol. 27, No. 4, pp. 8–23 (2010).

[26] 于　文博，磯部祥尚，枝廣正人：共有メモリ付階層型制御モデルの並列化アルゴリズムの CSP による形式化と FDR による検証，情報処理学会研究報告，Vol. 2020-EMB-53, No. 40, pp. 1–12 (2020).

[27] 西村　裕，中村　陸，荒川文男，枝廣正人：ソフトウェア向けハードウェア性能記述を用いたマルチコアにおける性能見積り，情報処理学会研究報告，Vol. 2014-EMB-32, No. 26, pp. 1–6 (2014).

[28] 山口滉平，竹松慎弥，池田良裕，李　瑞徳，鍾　兆前，近藤真己，枝廣正人：Simulink モデルからのブロックレベル並列化，組込みシステムシンポジウム 2015 論文集，Vol. 2015, pp. 123–124 (2015).