Vol.2021-ARC-244 No.18
Vol.2021-SLDM-194 No.18
Vol.2021-EMB-56 No.18
2021/3/25

車載制御モデルの並列設計における ランタイム性能解析と効率化

加藤 聖也[1,a)]　寒河江 翔太[2]　山本 椋太[1]　生沼 正博[1]　キム ジンス[3]　道木 慎二[3]　本田 晋也[4]
枝廣 正人[1]

概要：車載制御をはじめとして，組込みシステムの分野では，モデルベース開発やマルチコアプロセッサに対する需要が高まっている．そのため，モデルベース開発において並列設計を実現するモデルベース並列化が提案されているが，アプリケーションレベルでの並列化のため，実機プラットフォーム上での動作について十分に評価されているとはいえない．本研究では，組込みシステムにおけるモデルベース開発において，モデルレベルで並列化し，自動生成したコードを実機に実装した際の性能解析を行うことで，モデルレベルの並列化に対する課題抽出を実施した．その上で，自動生成コード仕様を改良することにより，実行時間の高速化を実現した．最後に，動作状況を確認した結果から，モデルベース並列化の有用性を示した．

Seiya Kato[1,a)]　Shota Sagae[2]　Ryota Yamamoto[1]　Masahiro Oinuma[1]　Jinsoo Kim[3]
Shinji Doki[3]　Shinya Honda[4]　Masato Edahiro[1]

1. はじめに

近年，ソフトウェアの高機能化が要求され，車載制御をはじめとした組込みシステムの大規模・複雑化が進んでいる．それに対応するため，モデルベース開発およびマルチコアプロセッサによる並列処理が広く用いられている．一般的には，制御設計者がモデルベース開発を用いて設計，逐次コード生成を行い，ソフトウェア開発者や自動並列化ツールが逐次コードをもとに並列化するという流れで行われる．しかし，制御設計と並列化を別々の開発者やツールが行っているため，モデルと実装並列化コードの関係性が希薄になる．これによって，並列化したシステムで時間制約が満たせないなど，期待した性能が得られず，モデルを改良しなければならない場合に，実装からモデルへの手戻りコストに多大なコストが必要となる．

そのような背景のもと，モデルレベルで並列設計を実施し，自動で並列化コードを生成する手法の研究が進められている [1]．しかしながら，現状のモデルレベルでのコード生成では，実機プラットフォーム上での動作や性能について十分な評価がされていないという課題がある．

本研究では，我々が提案しているモデルベース並列化（以降，MBP とよぶ）[1] を例として，自動生成コードの実機実装と性能解析を行い，また，モデルレベルでの並列化に対する課題抽出を行う．

2. 準備

2.1 モデルベース並列化（MBP）

我々が研究開発を進めている MBP の流れについて説明する [2][3]．まず MBP では，Mathworks 社の MATLAB/Simulink[4] で設計されたモデルをもとにブロックレベル構造情報 BLXML を作成する．BLXML には，Simulink モデルの各ブロックの，ブロック情報，対応する実装コード，コア割り当て情報などが XML 形式で記述されている．Simulink モデルの各種情報は MATLAB 言語を用いて取得することができ，ブロック名，ブロックタイプ，入出力関係等を取得して BLXML の骨格を作る．次に，Simulink モデルから Mathworks 社の Embedded Coder[5] を用いて生成した逐次コードから各ブロックの実装コードを抽出する．そして，このコード情報をもとにブロックの処理量を見積もり，BLXML に格納する．処理量の見積にはターゲットハードウェアにおける命令や通信のレイテンシ等が記述されたハードウェア抽象化記述 SHIM[6] を用いる．見

[1] 名古屋大学大学院情報学研究科
[2] 名古屋大学情報学部
[3] 名古屋大学大学院工学研究科
[4] 南山大学理工学部
[a)] seiya_k@ertl.jp

積もった処理量情報は自動並列化時に活用される．

BLXML にブロック情報を格納した後，この情報をもとに並列化を行う．並列化のためには処理をコアに分ける必要がある．このとき，各コアの処理時間の均一性（負荷分散），およびコア間通信によるオーバヘッド最小化を意識したコア割り当てが必要である．これには，各コアの処理量を見積もることが必要であり，BLXML に格納した各ブロック見積処理量を用いる．

MBP では最適なコア割り当て算出をグラフ分割問題として考える手法を提案している [7]．ここでは，モデルのブロックを頂点，信号線を辺とするグラフと捉え，頂点のコストをブロックの処理量とすることで，負荷分散問題をコストが均一となるグラフ分割を求めるという問題として考える．さらに，グラフ分割した際に切断された辺でコア間通信のコストが発生すると考えることで，負荷分散に加えて通信コストを考慮したコア割り当て導出が可能となる．

以上のようにして求めたコア割り当て情報は BLXML に格納され，並列化コード生成に使用する．BLXML にはブロックの接続関係，コア割り当て情報，実装コードが保存されているため，コア割り当て情報をもとに関数やタスクをコアごとに作成し，データ依存関係順に実装コードを記述し，タスク間通信を適切に挿入することで並列化コードを作ることができる．並列化コードとしては eSOL 社のマルチ・メニーコアに対応したリアルタイム OS である eMCOS[8] 向けのコードと，車載システムの標準仕様である AUTOSAR CP に準拠した OS 向けのコードが生成できる [9]．また，コード生成には様々なオプションがあり，その 1 つとして memcpy の展開がある．memcpy はコア間通信にて配列データを送受信する場合に用いられるデータをまとめて代入する関数である．しかし，コンパイラなどによって memcpy にかかる時間は異なる．そのため並列化コード生成の段階で memcpy を使うかどうか選択できる．

2.2 TOPPERS/ATK2-SC1-MC

車載システムは，標準仕様である AUTOSAR CP に準拠した RTOS の上に実装して動かす．前項でも述べた通り，MBP でも AUTOSAR OS 向けの並列化コードを自動生成できる．本研究ではこのコードを AUTOSAR OS 仕様準拠の TOPPERS/ATK2-SC1-MC[10] 上に実装し，並列制御性能や実行時間の評価を行う．

TOPPERS/ATK（Automotive Kernel）は，TOPPERS プロジェクトで公開している自動車制御用リアルタイム OS の総称である [10]．OS 共通部からターゲット依存部を分離設計しているため，ターゲット依存部の書き換えのみで，容易に移植が可能となっている．その中でも本研究で使用している ATK2-SC1-MC はマルチコア対応していることが特徴である．

ここで，TOPPERS が提供している動作状況可視化ツール TLV(TraceLogVisualizer)[11] について説明する．TLV は各種リアルタイム OS やエミュレータ等が出力するトレースログを可視化し，トレースログの解析を容易にするためのツールである．タスクの状態変化を可視化できるが，トレースログの出力にオーバヘッドが発生する．本研究では，より正確な動作状況を見るために，以下の方法を取った．ロジックアナライザを用いて，マイコンボード上の制御に使用していないピンに電流を流す指示文と，電流を止める指示文を，動作状況を見たい箇所の前後に挿入することで見たい箇所だけ ON とし，波形を記録する．タスクの実行部分を可視化したいため，プレタスクフックにて電流を流す命令，ポストタスクフックにて電流を止める命令を記述した．ロジックアナライザの記録を csv ファイルに出力し，csv ファイルのデータをトレースログ形式に変換する．そのログをもとに TLV にて動作状況を可視化する．

3. 性能解析と効率化

本章では，車載制御を想定し，MBP で生成した自動並列化コードを，TOPPERS/ATK2-SC1-MC を搭載した実機上で動作させた際に実施した，性能解析および効率化について述べる．以下では，ブラシレスモータ制御モデルを題材とし，制御器入力部およびコア間通信に関して実施した効率化提案について述べる．

3.1 ブラシレスモータ制御モデル

図 1 は本研究の対象モデルであり，ベースレート 200μs のシングルレートで，ブラシレスモータをフィードフォワード制御からベクトル制御に遷移しながら制御する Simulink モデルとなっている．制御の遷移は実行後 0.84 秒で実施する．フィードフォワード制御は，実際の状況を取得せずに，システムのモデルに基づいて動作を予測しながら制御入力を決定する制御手法である．ベクトル制御は，回転子として永久磁石を用いた永久磁石同期モータを使用した制御方法である．三相コイルに流す電流（三相電流）をトルクに関係する磁力方向の電流に変換して，回転子の磁界と固定子の磁界の引力および斥力を用いてモータを回転させる．

図 1 の制御モデルではモータプラント部分が取り除いてあり，コード生成後にモータからの入力処理として，実際の電流値に対してアナログ電気信号をデジタル電気信号に変換する AD 変換を行う処理，モータへの出力処理として，PWM（Pulse Width Modulation:パルス幅変調）による出力を行う処理を手動で追加している．PWM は，直流電源の ON/OFF（比率をデューティ比と呼ぶ）を変更することによって任意の平均電圧（電流）を発生させる方式である．ただし，これらの処理は Embedded Coder が生成したコードには存在せず，コード生成後に追加しているため，MBP の見積には反映されていない．

Vol.2021-ARC-244 No.18
Vol.2021-SLDM-194 No.18
Vol.2021-EMB-56 No.18
2021/3/25

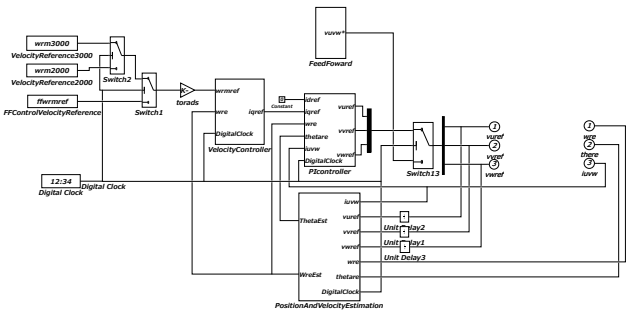

図 1　ブラシレスモータ制御モデル

3.2　実装環境

マイコンボードとして北斗電子社製 HSBRH850F1H176 を用いている．このボードには RH850/F1H マイコンが搭載されている．これは実際に車載マイコンとして用いられている Renesas Electronics 社 [12] の RH850/F1x シリーズのハイエンド向けデュアルコアマイコンである．

ターゲットとしてブラシレスモータを用いている．これも実際に車載モータとして用いられているものであり，実際の車載環境を想定している．

マイコンボードおよびブラシレスモータはブラシレスモータドライバボード（北斗電子社製 BLM_EV6[13]）に接続されている．マイコンボードで制御計算された出力をもとにブラシレスモータドライバボードが適切な電流をモータへ送り，回す仕組みとなっている．

マイコンボードへのプログラムの書き込みには Renesas 純正統合開発環境 CS+および E1 エミュレータ [12] を用いる．コンパイラには CS+の RH850 ファミリ向けコンパイラ CC-RH を使用する．

3.3　制御器入力部の性能解析と効率化

ベクトル制御は，モータに流れている電流値をもとにモータ回転速度を制御するための電流値を計算する．そのため，モータに流れている電流に AD 変換を行い，電流値を検出する必要がある．AD 変換を行う処理は MBP でコード生成した後に手動で追加している．コード上の追加箇所は図 1 右側にある入力ポートの 1 つである iuvw という処理（コード生成時にはコメントが存在するだけで実質的な処理はない）を行う箇所で，タスクの先頭に位置する．実際に追加している記述は AD 変換開始文と AD 変換完了を待つ while ループとなっている．

3.3.1　問題検出の背景と解析方法

初めに MBP で AUTOSAR OS 向けに生成した逐次コードを実装した．しかし，実行後 1 秒程度でモータが停止した．開始直後は動作していること，制御遷移が開始 0.84 秒

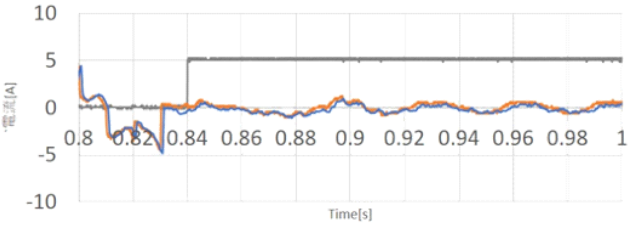

図 2　OS レス手動生成コード実行時の電流実測値と検出値の様子

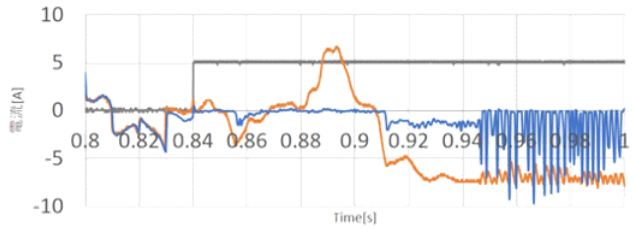

図 3　MBP 自動生成コード実行時の電流実測値と検出値の様子

で実施されることから，制御遷移後に問題があると考えられる．そこで，ベクトル制御を行うために必要な電流検出ができているか確認するため，電流検出値とオシロスコープで計測した電流実測値を比較した．図 2 はベクトル制御の動作が確認できた OS レス手動生成逐次コードによる比較結果で，図 3 は MBP 自動生成コードによる比較結果である．MBP 自動生成コードでは制御遷移後に電流検出ができていないことがわかった．なお，フィードフォワード制御時も電流検出自体は行っているが，動作状況を無視して出力をし続けるため動作できていたと考えられる．

ここで，電流検出の仕組みについて述べる．図 4 はモータに流れる三相電流を再現したものである．U 相 V 相 W 相の三相の電流の関係性を図下部の相電流の計算の式によって表している．各相のインバータの上アームに電流が流れているときは対応する下アームに電流は流れず，その逆も成り立っている．電流検出はインバータの下アームに電流が流れているときに検出可能となる．今回のモデルでは，ハードウェアタイマを用いて出力タイミングを定めており，制御周期の始めは下アームに電流が流れているように設計されている．

次に，図 5 はオシロスコープにて各相の上アームに電流が流れているときに ON となる 3 本の波形と電流検出を開始するタイミングで ON になる波形を表したものである．上アームが ON になっているときに電流検出をしている．つまり，下アームに電流が流れていないときに電流検出するため上手く検出できていない．本来ならば上アームが OFF の区間のちょうど真ん中で電流検出が開始するはずである．

3.3.2　効率化提案

OS レス手動生成逐次コードのときは AD 変換をタスクの先頭で開始していても検出タイミングは間に合っていた．

Vol.2021-ARC-244 No.18
Vol.2021-SLDM-194 No.18
Vol.2021-EMB-56 No.18
2021/3/25

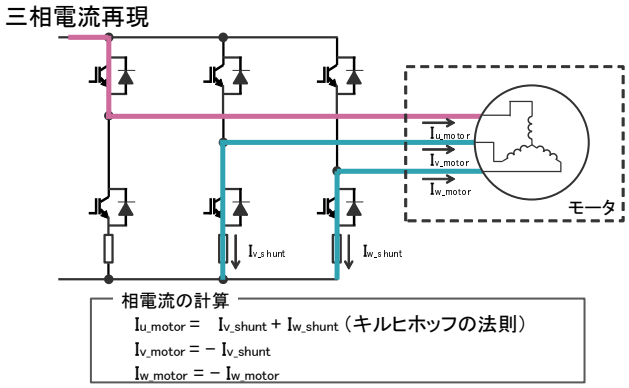

図 4　モータに流れる電流の仕組み

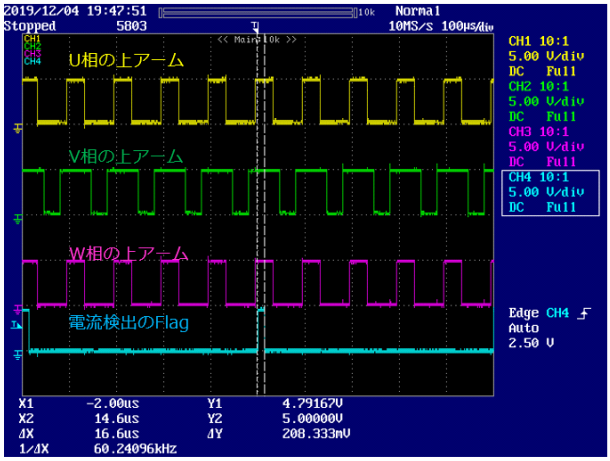

図 5　電流検出タイミングの様子

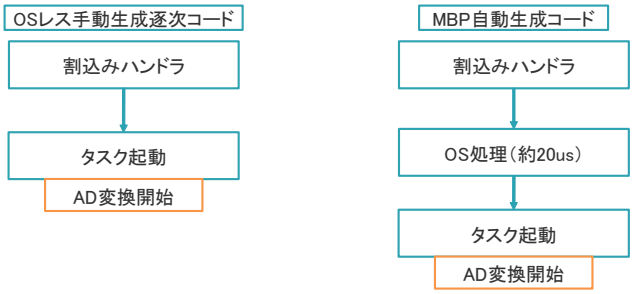

図 6　変更前の AD 変換タイミング

そのため，図 6 のように MBP 自動生成コードでも AD 変換はタスクの先頭で開始するように記述を追加していた．しかし，OS レス手動生成逐次コードの時点で，割込みハンドラから直接タスクを呼ぶのに数 μs かかりタイミングは少し遅れていた．そこに MBP 自動生成コードでは割込みハンドラとタスク起動の間に OS 処理が入り込むため，電流検出が間に合っていないと考えられる．

その OS 処理では，割込みハンドラにて IncrementCounter を用いてアラームによりタスクを起動させていた．また，割込みハンドラの起動にソフトウェアタイマを使用していた．そこで，OS レス手動生成逐次コードのように，MBP

図 7　AD 変換タイミング変更の様子

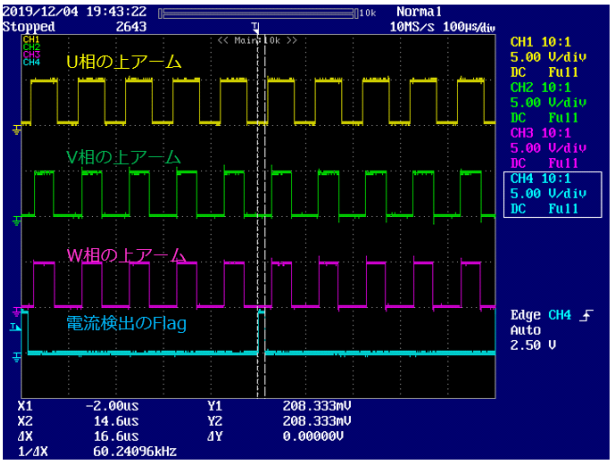

図 8　変更後の電流検出タイミングの様子

自動生成コードでもソフトウェアタイマやアラームは用いず，ハードウェアタイマで割込みハンドラを起動するよう変更した．しかし，割込みハンドラで直接タスクを起動するにも OS 機能を使用するため，まだ検出タイミングは間に合っていなかった．そこで，図 7 のように AD 変換をタスク先頭から割込みハンドラ直後で開始するよう変更した．AD 変換完了を待つ while ループは，そのままタスクの先頭で実行させている．

3.3.3　結果

変更後再び電流検出のタイミングを確認したところ，図 8 のように検出タイミングを間に合わせることに成功し，制御遷移後もモータの回転を確認できた．

3.4　コア間通信部の性能解析と効率化

TOPPERS/ATK2-SC1-MC ではコア間通信を OS 機能であるイベントを用いて実現している．イベントを用いてタスクの実行状況を OS が管理することで，異なる優先度タスクが存在する場合，高優先度タスクが待ち状態のときに低優先度タスクを実行でき，実行性能を向上させることができる．

3.4.1　問題検出の背景と解析方法

前節の効率化後においても，並列実装すると制御遷移後にモータが停止した．実行時間を計測したところ，並列実

表 1　各イベント機能の処理時間

SetEvent	WaitEvent	ClearEvent
約 3.3μs	約 7.5μs	約 2.2μs

行時間は 268μs であり，制御周期は 200μs であるため，デッドラインミスが発生していることがわかった．また，逐次実行時間を計測すると 65μs であったので並列実装に起因するオーバヘッドが影響していると考えられる．例えば，通信毎に発生する OS のイベント（SetEvent，WaitEvent，ClearEvent）の処理にかかる時間を，表 1 に示す．なお，実行時間は，プレタスクフックとポストタスクフックを使用して，実行部の時間をロジックアナライザで計測することによって求めた．

3.4.2　効率化提案

効率化した内容は，コア間通信等の OS 処理，memcpy 処理，メモリマッピングである．以下に効率化方法を示す．

- 通信機構の改良
  コア間通信にイベントを用いる利点は，タスク切替を OS が管理することで同一コアの複数のタスクを効率良く実行できることにある．しかし，今回のようなシングルレートモデルの場合，各コアに 1 つのタスクしか存在しないため OS が管理する必要がない．そこで，通信方式をポーリング方式で実行できるように変更した．ポーリング方式は，通信相手からデータが届くのを定期的に確認しながら待ち続ける方式である．イベント機能を用いないので，通信毎にかかっていたオーバヘッドを削減できる．一方，マルチレートモデルにおいてもポーリング方式を採用することは可能であるが，すべてのタスクでポーリング方式を採用すると，実行中のタスクの優先度以下のタスクは実行中のタスクの処理が終わった後にしか実行できない．他のモデル（アプリケーション）と一緒に実行する場合などを考えたとき，必ずしもポーリング方式を採用するのが最善とは限らない．よって，イベント方式とポーリング方式の両方を上手く取り入れる必要があるため，動作状況などから判断して，通信方式を各通信ごとに選択できるような自動生成コード仕様にした．
- 計測方法の改良
  実行時間の計測に，プレタスクフックとポストタスクフックを使用した方法で実施しているが，プレタスクフックやポストタスクフックを使用することでもオーバヘッドが発生するため，自動生成コード上のタスクの先頭に電流を流す命令，最後に電流を止める命令を記述するように変更した．
- memcpy 処理
  memcpy はコア間通信にて配列データを送受信する場合に用いられるデータをまとめて代入する関数である．しかし，今回使用しているコンパイラではこの memcpy が 1byte 毎しかコピーせず，24byte コピーするのに約 2μs かかっていた．そのため，memcpy を使わず配列を展開してそれぞれ直接値をコピーするようにした．
- メモリマッピング
  マイコンには用途によって様々なメモリが用意されており，用途に合ったデータ配置をすることで高い性能を得ることができる．しかし，MBP が自動生成する並列化コードは，ハードウェアに依存してしまうメモリマッピング最適化には対応できていない．現状のメモリ配置を調べると，すべてのローカルデータが 1 つのローカル RAM に配置されていた．それにより，片方のコアのデータアクセスに時間がかかっていた．そのため，メモリ配置を指定する指示節を加え，配置の最適化を行った．

3.4.3　結果

実行時間は 51μs まで短縮し，約 81%の実行時間の短縮を実現し，制御遷移後もモータの回転を確認できた．

4.　評価

4.1　評価方法

図 1 のモデルに対して，逐次実行，並列実行（ポーリング方式），並列実行（イベント方式）で実装したときの制御性能と実行時間についての評価と，動作状況の比較を行う．

制御性能評価では，Simulink のシミュレーションとの制御結果の比較を行う．シミュレーションと実機で同じ回転速度指令値を与えたときの回転速度をそれぞれ取得し，比較・評価する．

実行時間評価では，MBP で見積もった実行時間や効率化前の結果と比較・評価する．

動作状況比較では，MBP で見積もった動作状況を可視化したガントチャートと実機上での動作状況を可視化した TLV を比較する．

4.2　制御性能評価

モータの回転速度指令値に対して速度推定ができているか確認する．図 9 は，制御遷移後や速度指令値を変更して，推定速度が指令値に収束していく様子をグラフ化したものである．すべてのパターンが，Simulink でシミュレーションした結果と同等の性能を示した．

4.3　実行時間評価

各実装方法に対して，MBP による見積時間と，実機上での実行時間をまとめた．効率化により実行時間は短縮できたが，逐次実行より遅い結果となった．また，通信方式をポーリング方式をイベント方式にするだけで，実行時間が約 3 倍となった．なお，イベント方式の見積時間は，コア割り当てを決定した後に，通信にかかる時間をイベント

Vol.2021-ARC-244 No.18
Vol.2021-SLDM-194 No.18
Vol.2021-EMB-56 No.18
2021/3/25

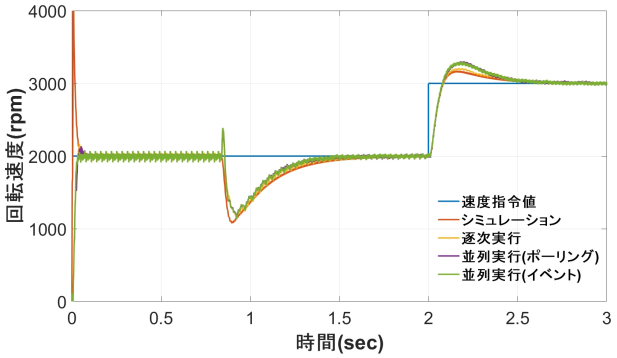

図 9　制御性能結果

表 2　シングルレートモデルの見積時間と実機上での実行時間

実装方法	見積時間	実機上での実行時間
逐次実行	51μs	49μs
並列実行効率化前	39μs	268μs
並列実行ポーリング方式	39μs	51μs
並列実行イベント方式	59μs	161μs

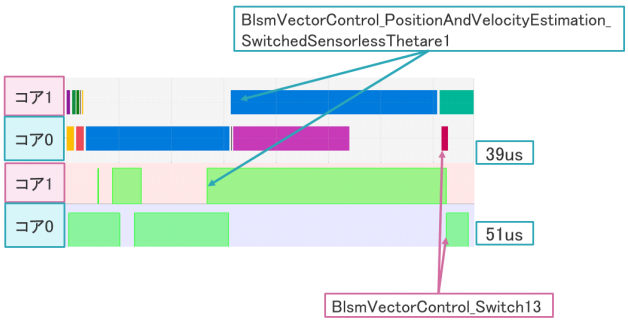

図 10　ポーリング方式による動作状況比較結果

を考慮して大きく見積もった結果である．

4.4　動作状況比較

この比較は MBP のコア割り当ての見積をもとに設計にフィードバック可能であるかどうかを示すために行う．図 10 は，通信方式にポーリング方式を採用したときの並列実行の動作状況を比較した結果で，図 11 は通信方式にイベント方式を採用したときの並列実行の動作状況を比較した結果である．図の上半分は，MBP で見積もった動作状況を可視化したガントチャートで，下半分が，実機上の動作状況を可視化した TLV である．横軸は時間軸となっており，縮尺はガントチャートと TLV で異なっている．MBP のガントチャートは待ちが入る度に色が変わる仕様となっているが，今回は色分けについては考慮せず，実行部分の比較のみ行う．図 10 は，動作状況が近いように見え，長い時間待たされているブロックが見積と実機で同じであった．図 11 は，動作状況は近くなかったが，長い時間待たされているブロックがポーリング方式のときと同じであった．

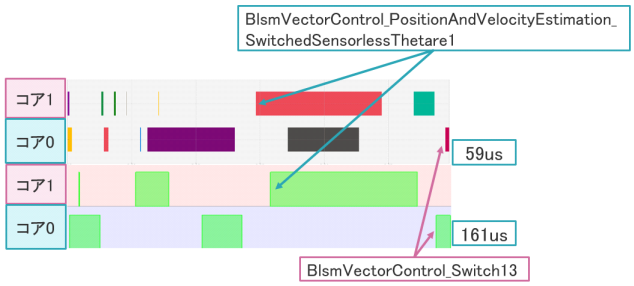

図 11　イベント方式による動作状況比較結果

4.5　考察

制御性能評価にて，どのパターンもシミュレーションと同等の性能を示していたことから，MBP によって並列化コードを自動生成しても制御性能は低下しないと考えられる．

実行時間評価では，今回のモデルを並列実装することによる性能向上はほとんど得られないことがわかった．この原因として，モデルの逐次性が高いことが考えられる．しかしこれは，動作状況の見積結果をもとに設計にフィードバックし，モデルを見直すことで逐次実行より高速なモデルを設計することが可能であると考える．

イベントの処理時間が，制御周期の短いモータ制御系にとって大きなオーバヘッドとなっているため，今回はポーリング方式を採用した方が良い結果だった．しかし，例えば，1 つのコアに異なる優先度の複数のタスクが割り当てられ，かつ，高優先度タスクの実行中に大きな待ち時間が発生する場合には，その待ち時間に低優先度タスクを実行することで実行効率が向上する．このように，状況に応じて通信方式を使い分ける必要があると考える．

並列実装において，見積時間と実機上での実行時間に差があるが，これは見積時には想定できていない OS 処理時間が存在するためであると考えられる．また，入出力部の処理がガントチャートに反映されていないことも考えられる．各ブロックの処理量や，実機上での OS 等の性能を正確にフィードバックすることで，より精度の高いコア割り当てや，実機上での動作状況や実行時間に近い結果を見積もることができると考える．

動作状況を比較した結果から，ポーリング方式の場合ガントチャートと TLV の見た目は近いことがわかった．ただし，入出力部の処理がガントチャートに反映されていないことによる見積誤差が生じていた．入出力部の処理もガントチャートに反映することができれば，かなり近い結果となったと考えられる．

また，ポーリング方式とイベント方式で動作状況は一見異なってはいるが，待ち状態になる箇所（ブロック）が同じであることが多かった．特に，ボトルネックとなっている部分は近いことがわかった．以上から，設計者はポーリング方式時の見積結果，つまり実機の性能を考慮しない方法

で，MBP の見積結果をもとにモデル設計にフィードバックしてよいと考える．しかし，フィードバック後の性能解析を実施していないため，見積結果をもとに設計を見直したモデルを再度評価して性能向上が確認することで，MBP の有用性を示すことができると思われる．

5. おわりに

本研究では，モデルベース開発における，モデルレベル並列化ツールである MBP による自動生成コードの実機実装と性能解析を行った．実装時に気を付けることとして，電流検出の開始タイミングのような厳しい時間制約のある処理は，実装するときに対応しなければならない．また，コア間通信にて発生する OS 処理によるオーバヘッド等の効率化や，メモリマッピングの最適化なども実装時に行う必要がある．

性能解析結果として，MBP による制御性能低下は見られないことがわかった．また，並列実装による性能向上は実現できなかったが，これはモデルの逐次性が高いことに起因し，MBP による動作状況の見積結果をもとにモデル側にフィードバックすることで改善される可能性がある．さらに MBP によって動作状況を見積もったガントチャートと実機上の動作状況を可視化した TLV を比較することによって，実機の性能を考慮しない上位レベルでのフィードバックの可能性を示した．

今後の課題として，モデル上にはないが後から追加する処理を含めたコア割り当てをすることでより実機上の動作状況に近い見積ができ，正確なコア割り当てを実現することがあげられる．実際に見積結果をフィードバックしたモデルに対して再度実装し実行時間等を評価することで，さらなる MBP の有用性を示す．

参考文献

[1] 山口滉平他: Simulink モデルからのブロックレベル並列化, 情報処理学会組込みシステムシンポジウム 2015 論文集, pp.123–124, 2015.
[2] 池田良裕他: 制御モデルに内在する遅延を用いた並列化, 研究報告組込みシステム（EMB）, Vol.2018-EMB-47, No.24, 2018.
[3] 竹松慎弥: モデルベース並列化の応用に関する研究, 修士論文, 名古屋大学, 2018.
[4] MathWorks: MATAB/Simulink, http://www.mathworks.co.jp, 2019/1/30.
[5] MathWorks: Embedded Coder, https://jp.mathworks.com/products/embedded-coder.html, 2021/2/14.
[6] Software-Hardware Interface for Multi-Many-Core(SHIM), https://www.multicore-association.org/workgroup/shim.php, 2019/1/29.
[7] 鍾兆前他: モデルベース開発におけるマルチ・メニーコア向け自動並列化, 研究報告組込みシステム（EMB）, Vol.2017-EMB-44, No.47, 2017.
[8] eSOL eMBP(model based parallelizer), https://www.esol.co.jp/embedded/mbp.html, 2019/1/30.
[9] 中野友貴他: モデルベース並列化（mbp）におけるマルチレートモデルの車載 rtos 向けランタイムとコード生成, 研究報告組込みシステム（EMB）, Vol.2017-EMB-44, No.4, 2017.
[10] TOPPERS/ATK2, https://www.toppers.jp/atk2.html, 2021/2/14.
[11] TLV, https://www.toppers.jp/tlv.html, 2021/2/14.
[12] Renesas Electronics, https://www.renesas.com/jp/ja, 2021/2/14.
[13] 株式会社北斗電子: ブラシレスモータドライバボード https://www.hokutodenshi.co.jp/7/brushless_motor.htm#driver, 2021/2/15.