Vol.2018-ARC-230 No.24
Vol.2018-SLDM-183 No.24
Vol.2018-EMB-47 No.24
2018/3/8

# 制御モデルに内在する遅延を利用した並列化

池田 良裕[1,a)]　鈴木 悠太[2]　峰田 憲一[2]　森 裕司[3]　井上 雅理[4]　道木 慎二[4]　枝廣 正人[1]

**概要**：組込みシステムの分野では，マルチコアプロセッサに対する需要が高まっている．しかし，人手でソフトウェアを分割し，マルチコアプロセッサへ配置することは難しい．また，システム次第では元々並列性が少なく，マルチコアプロセッサで動作させたとしても性能向上が期待できない．本研究ではMATLAB/Simulink を用いて設計された並列性の少ない制御モデルを対象に，並列化時のデータ依存関係を調整することで実行性能を向上させる．さらに調整箇所をモデルに反映することで制御性能面も確認する方法を提案する．最後に提案手法による実行性能向上と制御性能の変化をシミュレーションにより評価し，提案手法の有効性を確認した．

# Parallelization using delay in control model for Model-Based Development

Yoshihiro Ikeda[1,a)]　Yuta Suzuki[2]　Kenichi Mineda[2]　Hiroshi Mori[3]　Masamichi Inoue[4]
Shinji Doki[4]　Masato Edahiro[1]

## 1. はじめに

近年，ソフトウェアの高機能性が要求され，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の 2 種類の技術が広まってきている．モデルベース開発ではシステム全体をブロック線図を用いて「実行可能な仕様書」として記述することで設計から実装までを円滑に行うことが可能である．またシングルコアプロセッサの性能向上限界により実行時間を短縮するためにマルチコアプロセッサを用いることが増えている．そこで 2 つの技術に注目し，システムを記述したモデルからマルチコアプロセッサ上で動作可能な並列化コードを生成する，モデルベース自動並列化技術 (以降，MBP と呼ぶ) への期待が高まっている．しかしながら，MBP を用いたとしても対象のモデル次第では性能向上が難しいケースもある．その一つとして逐次性の高いモデルが挙げられる．モデルをマルチコア上で動作させる際には通信によってデータ依存関係を維持している．そのため，逐次性が高いモデルでは，並列化時にコア間同期のための通信により，待ち時間が多くなり性能向上しない．この問題は将来の制御システムのマルチコア化における重要な課題となっており，本質的な解決が期待されている．

本論文では，待ち時間が少なくなるように，データ依存関係を調整することで，実行性能を向上させる．さらに，データ依存関係の調整結果をデータ更新の時間遅れとしてモデリングすることで，制御性能を確認する手法を提案する．最後に，モデル変更における実行性能面と制御性能面のトレードオフ解消を目的として，MBP を用いた制御－実装協調設計手法についても提案する．

[1] 名古屋大学情報科学研究科
Graduate School of Information Science, Nagoya University
[2] 株式会社デンソー
DENSO CORPORATION
[3] 株式会社エヌエスアイテクス
NSITEXE,Inc.
[4] 名古屋大学工学研究科
Graduate School of Engineering and School of Engineering, Nagoya University
[a)] yoshi12@ertl.jp

## 2. 準備

### 2.1 MATLAB/Simulink[1]

MATLAB/Simulink はモデルベース開発支援ツールの一つであり，車載向けの組込みシステム開発では広く使われている．MATLAB/Simulink を用いたモデルベース

Vol.2018-ARC-230 No.24
Vol.2018-SLDM-183 No.24
Vol.2018-EMB-47 No.24
2018/3/8

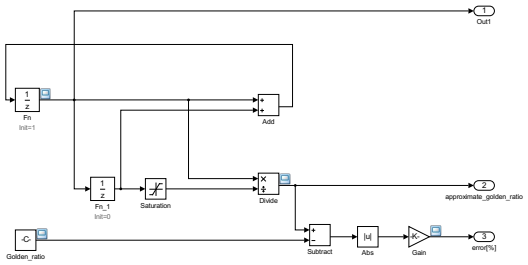



**図 1** Simulink モデル

開発では Simulink モデルと呼ばれるブロック線図によりアプリケーションを記述する．Simulink モデルの例を図 1 に示す．Simulink モデルを作成することで，シミュレーションや自動コード生成を行う事ができる．本研究では MATLAB/Simulink により記述された制御モデルを対象とする．

### 2.2 モデルベース自動並列化技術 [2]

現在，我々が進めているモデルベース自動並列化技術の流れについて説明する．まず，自動コード生成可能な Simulink モデルからブロック情報及びブロック間の接続情報を抽出する．次に，自動生成した逐次コードをブロック毎に対応させ，**BLXML**(Block Level XML) を生成する．BLXML は，ブロックの入力・出力，対応するコード情報を一つのグループとして，ブロック毎に記述される．最後に，ブロック毎の処理量や通信オーバヘッドを考慮してコア割り当てを行い，並列化コードを生成する．ブロック毎の処理量見積もりには SHIM(Software Hardware Interface for Multi-many-core) [5] を用いた．

また，モデルベース自動並列化技術では，BLXML を重み付き有向グラフとして表現した **BLGraph** を利用し，モデルの構造情報を扱いやすくすることで，コア割り当て等の自動化を容易にする機能が組み込まれている [4]．図 1 のモデルから生成された BLGraph を図 2 に示す．BLGraph 内のノードはモデル内のブロックに該当し，エッジはブロック間を繋ぐ信号線に該当する．本論文での解析は BLGraph を用いる．BLGraph 上で解析を行う際には，各ノードに対応したブロックの処理を考慮する必要がある．例えば，Rate Transition や UnitDelay ブロックは前後のブロック間で，1 周期内でのデータの受け渡しが行われないため，BLGraph においてもブロックの振る舞いを考慮し，解析を工夫する必要がある．後述するクリティカルパス探索でも，このような制御の性質に基づいた解析が必要となる．

## 3. 制御モデルの並列化における課題

一般に自動車などの制御システムでは逐次性が高く，並列化時の性能向上が期待できない．永久磁石同期モータ (PMSM:Permanent Magnet Synchronous Motor) の制御

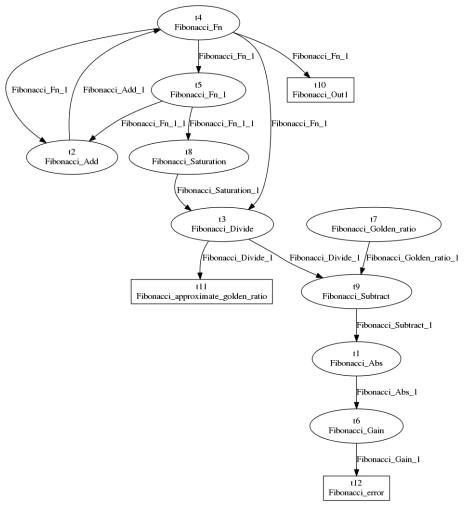



**図 2** BLGraph

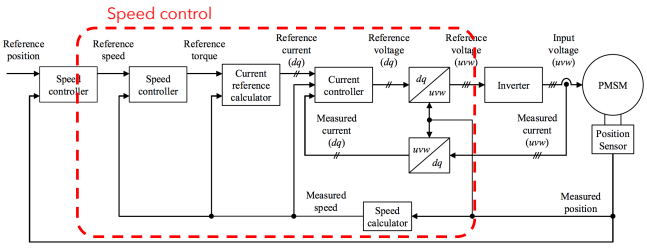



**図 3** PMSM の制御システム

を例に説明する [6]．PMSM の一般的な制御システムは図 3 で表される．モータの速度を指令値として制御を行いたい場合，図 3 の赤枠内の処理が必要となる．

以下に PMSM の制御フローを示す．

(1) 外部からの速度指令値とモータからの角度センサにより取得した速度から速度誤差を算出し，必要トルクを決定 (赤枠内の Speed Controller)
(2) 必要トルクと現在の速度から必要電流を決定 (Current reference calculator)
(3) 必要電流と現在の速度，電流のセンサ値から印加電圧を決定 (Current controller)

他にも制御を単純にするためのベクトル変換や印加電圧からモータを回転させるための波形を生み出すインバータの処理がある．この例からも分かる通り制御システムは逐次性が高く，モデル内ブロックに対するコア割り当て手法では性能向上が期待できない．

本論文では，データ依存関係を調整することでシステムの持つ並列性を向上させる方法を 4 章，5 章で提案し，6 章でモータ制御モデルを用いて評価を行う．

## 4. 時間遅れのモデリング

コア間同期通信の時間遅れのモデリング方法の一つに，UnitDelay ブロックを追加する方法がある．UnitDelay ブロックは入力信号を 1 制御周期遅延させて出力するブロッ

Vol.2018-ARC-230 No.24
Vol.2018-SLDM-183 No.24
Vol.2018-EMB-47 No.24
2018/3/8

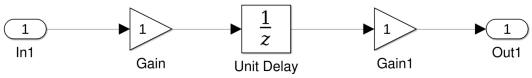



図 4　UnitDelay を含む Simulink モデル

```
1  void sample_step(void)
2  {
3  /* UnitDelay: '<Root>/UnitDelay' */
4  sample_UnitDelay_1 = sample_DW.UnitDelay_DSTATE;
5  /* Gain: '<Root>/Gain1' */
6  sample_Gain1_1 = 1.0 * sample_UnitDelay_1;
7  /* Gain: '<Root>/Gain' incorporates:
8  *  Inport: '<Root>/In1' */
9  sample_Gain_1 = 1.0 * sample_In1_1;
10 /* Update for UnitDelay: '<Root>/UnitDelay' */
11 sample_DW.UnitDelay_DSTATE = sample_Gain_1;
12 }
```

図 5　図 4 の Simulink モデルから生成された逐次 C コード

クである．UnitDelay ブロックの前後は 1 ステップの間にデータのやり取りが行われないため，コア間同期通信を 1 ステップの最後に行うことが可能となる．UnitDelay ブロックを用いたシンプルな Simulink モデル (図 4) と自動生成された逐次 C コード (図 5) を載せる．図 4 の実行順序は以下のフローで示される．括弧内は各処理に対応する図 5 中の行番号を示している．

(1) (4 行目)UnitDelay に格納されていた 1 ステップ前の入力を Gain1 へ送信
(2) (6 行目)Gain1 の処理を行い，結果を Out1 へ送信
(3) (9 行目)Gain が In1 からデータを受信し，処理を行い，結果を UnitDelay に送信
(4) (11 行目)Gain からの入力を UnitDelay に格納

本来，UnitDelay ブロックがなければ Gain の次に Gain1 が実行されるはずである．しかし，図 5 に示される実行順序では Gain1 の後に Gain が実行されている．そのように Gain と Gain1 の間に UnitDelay を追加することで，元々あったデータ依存関係を解消し，並列化時の自由度を向上させることができる．

## 5. 遅延挿入位置の選定方法

### 5.1 実行性能向上が見込まれる調整位置

本節では UnitDelay ブロックの挿入位置の選定方法について提案する．UnitDelay ブロック追加位置次第で，実行性能向上は大きく異なる．Simulink モデルを解析することで，より実行性能向上が期待できる UnitDelay ブロックの挿入位置を選定する方法を提案する．

ここで BLGraph も含まれるデータフローグラフ (DFG) におけるクリティカルパスが並列化後の実行性能向上に影響を与えることを考慮する [7]．DFG ではエッジで結ばれたノード間にはデータ依存関係が存在するため，辿ることができるパス上の処理は逐次的に行われなければならない．クリティカルパスとは DFG 上で辿ることのできる最長経路のことである．そのため，コア数を増やしたとしてもクリティカルパス上の処理が実行性能面でのボトルネックとなる．

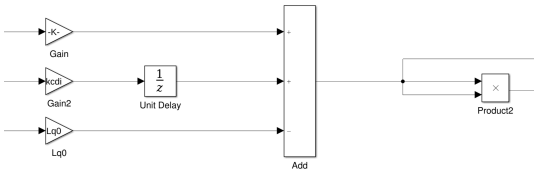



図 6　UnitDelay ブロックを追加してもデータ依存関係が解消しないケース

そこで本手法ではこのクリティカルパスを短くすることに注目する．4 章で述べた通り，UnitDelay ブロックを入れることでその前後のデータ依存関係を緩和することができる．これを利用し，クリティカルパス上に UnitDelay ブロックを挟むことで，前後のデータ依存関係を緩和し，クリティカルパスを短くすることが可能である．

### 5.2 クリティカルパスの探索方法

クリティカルパスの探索には Warshall-Floyd 法 [8] を用いる．Warshall-Floyd 法は多始点-多終点の有向グラフ内のクリティカルパスを探索するアルゴリズムである．本来，エッジに対して重み付けされた有向グラフを対象としているが，ノードに重み付けした有向グラフに向けに変更を加えている．また，探索の際には UnitDelay ブロック等のノードに対応する処理も考慮する必要がある．例えば UnitDelay ブロックは探索の始点ノード，終点ノードの両方として扱っている．

### 5.3 クリティカルパス上の遅延挿入位置の決定

最後にクリティカルパス上において，UnitDelay ブロックを追加する信号線の決定方法について提案する．先述した並列化のボトルネックを解消するだけならば，処理量のみを考慮してパスの中央に当たる信号線に追加すればよい．しかし，他のノードとのデータ依存関係次第では性能向上できないパターンが存在する．例としてはクリティカルパスの中央の信号線が多入力ブロックの入力の一つとなるパターン (図 6) が挙げられる．

図 6 は 3 入力 1 出力ブロックである Add ブロックの入力信号線のうち一つ (中段) のみに UnitDelay ブロックを追加した場合である．UnitDelay ブロックを追加することで後続のブロックをデータ依存関係に関わらず実行できるはずだが，図 6 のケースにおいては上段，下段の入力を待たなければならず，すぐに Add ブロックを実行することができない．このように多入力ブロックの入力信号線のうち，一つのみに UnitDelay ブロックを追加しても意味がな

Vol.2018-ARC-230 No.24
Vol.2018-SLDM-183 No.24
Vol.2018-EMB-47 No.24
2018/3/8

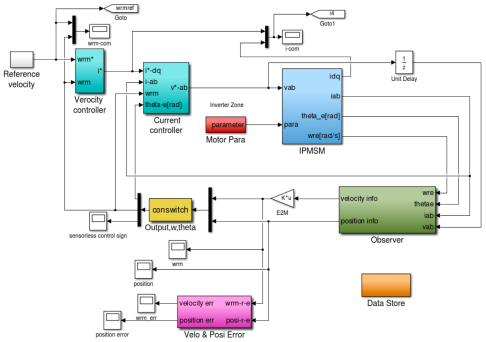



図 7　埋込磁石同期モータ (IPMSM) の制御モデル

いため，クリティカルパス上でブロックの追加位置を動かす必要がある．移動先の候補に挙がるのは Add ブロックの出力信号線である．なぜなら，後続の Product2 ブロックは Add ブロック以外の入力を持たないため，その位置に追加することで実行可能になるからである．Add ブロックの出力信号線のように，多入力ブロックの直後である点，1 出力である点を考慮した UnitDelay ブロックの追加位置の選定が必要となる．

本手法ではクリティカルパス上で処理量の面から中央にあたるブロックの前後± 20%以内にある，上記 2 点を満たすブロックを BLGraph 上で探索し，UnitDelay ブロックの追加位置としている．満たすブロックがなかった場合，分岐・合流の少ないモデルであるため，クリティカルパス上にないブロックとのデータ依存関係を考慮せず，処理量のみから UnitDelay ブロックの追加位置を選定しても実行性能向上が期待できる．

## 6. 性能評価

### 6.1 評価モデル

本評価では，名古屋大学工学研究科道木研究室提供の埋込磁石同期モータ (IPMSM) の制御モデル (図 7) を使用した．このモデルでは左上 (色：シアン) の Velocity Controller と Current Controller ブロックがそれぞれ速度制御器，電流制御器に該当し，右上 (色：水色) の IPMSM ブロックがモータのプラントモデルに該当する．本提案手法は制御器である Velocity Controller と Current Controller に対して適用した．IPMSM 制御では速度制御と電流制御は異なる周期で動作しており，本モデルにおいても速度制御は 1ms 周期，電流制御は 100 $\mu$ s 周期で動作している．その場合，0ms,1ms,2ms といったタイミングでは速度制御と電流制御の両方の処理が行われる事になり，時間内に処理が終了しないデッドラインミスを起こす可能性があるため，実行性能の向上が期待される．

以降，図 7 の Simulink モデルから Velocity Controller と Current Controller ブロックを切り出したものを評価モデルとして扱う．評価モデルには 108 個の Simulink ブロックが含まれており，主に算術演算ブロック (足し算，掛け算，三角関数演算等) と積分ブロックが使われている．

評価モデルに対して提案手法を適用し，電流制御器内部に UnitDelay ブロックを追加した．以降の評価では，UnitDelay ブロックを追加する前と後で比較評価を行う．

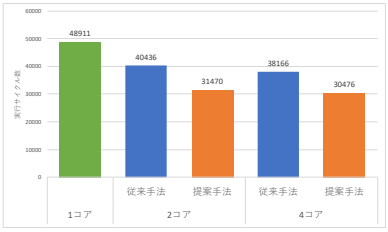



図 8　1ms 動作時の実行サイクル数

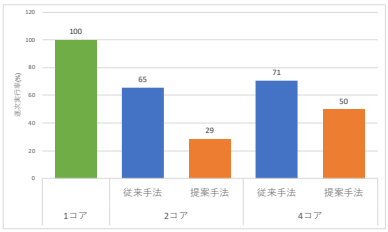



図 9　1ms 動作時の逐次実行率

### 6.2 実行性能評価

本節では実行性能面での評価について述べる．実行性能評価として行ったのは実行サイクル数比較，逐次実行率，スケジューリング図の三点である．

**実行サイクル数**

1ms 動作時の実行サイクル数の評価結果を図 8 に示す．図 8 の従来手法は遅延挿入前の Simulink モデルに対して並列化したもので，提案手法は遅延挿入後の Simulink モデルに対して並列化したものである．並列化時のコア割り当て手法はどちらも同じアルゴリズムを用いている．2 コア動作時，4 コア動作時のどちらにおいても提案手法により，実行サイクル数が減少していることが確認できる．

**逐次実行率**

1ms 動作時の逐次実行率の評価結果を図 9 に示す．UnitDelay ブロックを追加する前と比べ，逐次実行率は減少しているため，並列性は向上している．しかし，提案手法において，2 コアから 4 コアにした際に逐次実行率が高くなっている．このことから更にコア数を増やしても，見合った実行性能向上が期待できないことが想定される．更なる実行性能向上のためには，まず追加する UnitDelay ブロックの数を増やすことが考えられる．そうすることでデータ依存関係に縛られず実行可能なブロックが増えるためである．しかし，制御性能面では安定性と速応性が低下する可能性があるため，実行性能と制御性能のトレードオフを考える必要がある．

**スケジューリング図**

最後に 1ms 実行時のスケジューリングを図 10 に示す．

Vol.2018-ARC-230 No.24
Vol.2018-SLDM-183 No.24
Vol.2018-EMB-47 No.24
2018/3/8

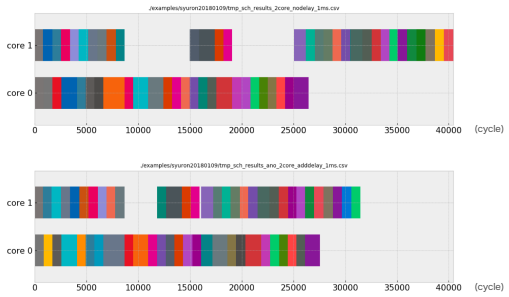



**図 10** 1ms 動作時の遅延追加前 (上) と遅延追加後 (下) のスケジューリング

1ms 動作時では，元々約 8000〜15000 サイクル及び，約 19000〜25000 サイクルに存在したコア 1 のアイドル時間の減少を確認できる．追加した UnitDelay ブロックによる前後のブロックの実行タイミングを確認すると，追加した UnitDelay ブロックの直後にある Product3 ブロックは，追加前では 27049 サイクル，対して追加後では 17244 サイクルから実行されていた．このことから UnitDelay ブロック追加により実行サイクル数が減少したことが分かる．

### 6.3 制御性能評価

本節では制御性能面での評価について述べる．制御性能評価としてステップ応答による安定性・速応性評価と極解析による安定性評価を行った．

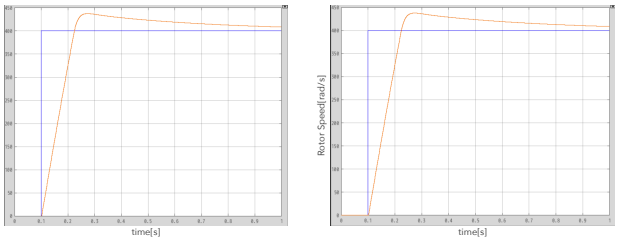
**ステップ応答評価**

ステップ応答は，ある制御システムへの指令値をステップ状に変更した際の出力値応答のことであり，ステップ応答を評価することで安定性 (出力値が収束するか) と速応性 (出力値がどれだけ早く指令値に到達できるか) を確認することができる．UnitDelay ブロック追加前後でのステップ応答を図 11 に示す．本評価では時刻 0.1s に 0rad/s から 400rad/s にステップ状に変化する速度指令値を用いた．図 11 の中において，青線が入力の指令値でオレンジ線が出力値である．

UnitDelay ブロック追加前後で応答時間，整定時間ともに特に差は見られない．よってステップ応答による評価では UnitDelay ブロックの影響はほぼないと考えられる．しかし，入力に様々なケースが考えられる制御システムではより幅広い条件において評価する必要がある．そこで次に極解析による評価を行った．

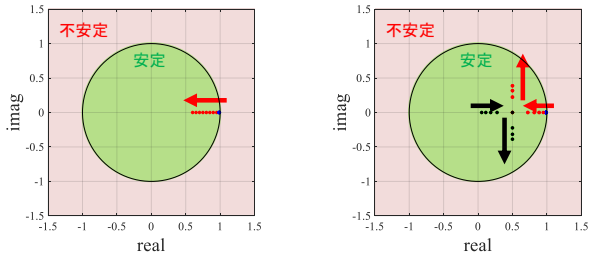
**極解析評価**

極解析は指令値を変化させた際に伝達関数から求められる極の変化を観測することで制御システムの特性を把握する解析手法である．本評価では離散化された伝達関数に対して解析するため，Z 領域で安定性を確認する．解析対象は UnitDelay ブロックを追加した電流制御器で，電流制御器の入力の電流制御帯域 ($\omega_{cc}$，単位:rad/s) と制御周期 ($T_{cc}$，単位:$\mu$ s) を変化させた場合の極の位置を観測する．




(a)UnitDelay ブロック追加前　(b)UnitDelay ブロック追加後

**図 11** UnitDelay ブロック追加前後のステップ応答比較




(a)UnitDelay ブロック追加前　(b)UnitDelay ブロック追加後

**図 12** UnitDelay ブロック追加前後の極解析結果

まず，UnitDelay ブロックを追加する前の電流制御器に対して $T_{cc}$ を 100 $\mu$ s で固定し，$\omega_{cc}$ を 500rad/s から 4000rad/s へ変化させた際の極解析結果について図 12(a) に示す．UnitDelay ブロック追加前では極が 2 つ存在 (図中の青，赤の点) し，青色の極は安定域の右端から動かず，赤色の極が $\omega_{cc}$ を上昇させるにつれて原点方向へ移動していく．この極解析結果から電流制御器が $\omega_{cc}$ を 500rad/s から 4000rad/s の範囲内で安定した制御を行えることが確認できる．

続いて，クリティカルパス上に UnitDelay ブロックを追加した電流制御器に対して，上記と同様に $\omega_{cc}$ を変化させた際の極解析結果について図 12(b) に示す．UnitDelay ブロック追加後では極が 3 つ存在し (図中の青，赤，黒の点)，青の極は UnitDelay ブロック追加前と変わらず，安定域の右端から動かなかった．しかし，赤，黒の極が $\omega_{cc}$ を上昇させるについれて実軸から離れていくように移動した．極が実軸から離れるとオーバーシュート (指令値を実測値が越えてしまう) する制御となる．そのため，UnitDelay ブロックを追加した後の電流制御器では設計とは異なる制御となってしまう．ここで電流制御器に用いられている PI 制御における制御周期について考える．PI 制御では安定制御を行えない際には，制御周期を短くすることで安定する場合がある [9]．そこで，電流制御器の制御周期を変更したモデルに対して極解析を行った．

UnitDelay ブロック追加後の電流制御器に対して $\omega_{cc}$ を 4000rad/s で固定し，$T_{cc}$ を 100 $\mu$ s から 10 $\mu$ s へ変化させた場合の極解析結果を図 13 に示す．UnitDelay ブロックの数は変わらないため，極の数は変化していない．$T_{cc}$ を減少させるにつれて赤，黒の極が実軸に近づいていき，$T_{cc}$ が 60

Vol.2018-ARC-230 No.24
Vol.2018-SLDM-183 No.24
Vol.2018-EMB-47 No.24
2018/3/8

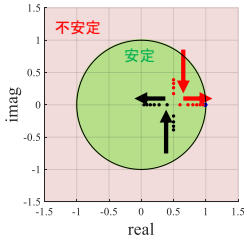



**図 13** UnitDelay ブロック追加後の制御周期変化時の極解析結果

$\mu$ s で実軸上にのるため，設計通りの制御が可能となった．これにより実行性能と制御性能を両立できる UnitDelay ブロックの追加位置が存在することが言えた．しかし，制御要件を満たすために制御周期を 60 $\mu$ s にすることは実行性能に大きな影響を与える．1ms 動作時を例に考えると，2 コア実行時の実行サイクルは 31470 に対して 1 コア実行時の実行サイクル数は 48911 のため，$\frac{31470}{48911} = 0.643$ 倍の時間で実行可能になる．ここで 48911 サイクルを処理するのに 100 $\mu$ s かかるとすると，60 $\mu$ s で処理できるサイクル数は $48911 \times 0.6 = 29346.6 < 31470$ となり，2 コアでの実行性能向上では足りず，デッドラインミスを起こしてしまう．制御要件が 80 $\mu$ s で満たせると仮定すると，39128.8 サイクル以内に実行すればよいため，UnitDelay ブロックを追加することにより実行性能を向上できる．

上述のように，実行性能と制御性能のトレードオフを議論する方法が必要となる．次章では両性能面を考慮するための設計者間の連携方法について検討する．

## 7. 制御-実装協調設計手法

従来のような実装による最適化のみでは，実行性能の向上が限界に来ており，制御設計段階からマルチコア上で動作させることを考慮してモデリングする必要がある．

本研究において，制御性能と実行性能を両立できる時間遅れのモデリングが理想であるが，自動解析により選定するのは難しい．実行性能に関してはクリティカルパス探索により，絞り込むことが可能だが，制御性能において制御要件を決めるのは人であり，判断も人が行わなければならない．そこで本論文では下記のような制御設計者と実装設計者間の時間遅れモデリングの選定フローを提案する．

(1) 制御設計者：制御モデル (Simulink モデル) を作成
(2) 実装設計者：制御モデル解析・時間遅れモデリング
(3) 実装設計者：実行性能評価
(4) 制御設計者：制御性能評価
　制御周期，パラメータの調整で対応可能なら 5 または 6 へ，そうでなければ 1 または 2 へ
(5) 実装設計者：制御周期調整依頼への対応
(6) 制御設計者：制御モデル内パラメータ調整
(7) 制御設計者・実装設計者：マイコンへ実装・実機テスト
手順 4 の段階で 1 または 2 に戻る回数が多くなれば，設計コストが大きくなってしまい両設計者にとって好ましくない．このフローを円滑に進めるための方法として，データ依存関係の調整箇所を複数提示することが挙げられる．実行性能面のみを考慮して最も優れた候補を制御設計者に提案しても，制御性能の面から却下される場合もある．しかし，複数の候補があれば制御設計者が選択可能となり，一つでも手順 5 以降に進める候補があれば手戻りは減ると考えられる．

## 8. おわりに

### 8.1 まとめ

本論文では，逐次性の高いモデルを対象として，データ依存関係を調整することで，実行性能を向上させる手法を提案した．さらに時間遅れモデリング方法の一つである遅延ブロックの追加により，制御性能を確認する手法を提案した．評価では，遅延を追加する前後で約 1.28 倍の並列実行性能向上を確認した．制御性能面では性能低下も見られたが，制御周期を変更することで安定した制御が行えることを示した．

### 8.2 今後の課題

本論文では 100 $\mu$ s と 1ms の 2 つの周期を持つモータ制御モデルに対して評価を行った．その際，両周期タスクが動作するタイミングでクリティカルパスを求め，データ依存関係の調整箇所を決定した．しかし，100 $\mu$ s タスクのみが動作するタイミングでの実行性能の向上は考慮できていない．よって，このような複数の周期を持つモデルに対して効果的なデータ依存関係の調整箇所を検討する必要がある．

### 参考文献

[1] MathWorks: MATLAB/Simulink, http://www.mathworks.co.jp
[2] 山口滉平, 池田良裕, 枝廣正人他: 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(ESS2015)，pp.123-124，2015.
[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 http://www.esol.co.jp/embedded/emcos.html
[4] 山口滉平:モデルベース開発におけるブロックレベル並列化のためのデータ形式，修士論文，名古屋大学，2016.
[5] M. Gondo, F. Arakawa, and M. Edahiro: Establishing a Standard Interface between Multi-Manycore and Software Tools - SHIM，COOL Chips XVII，VI-1，2014.
[6] 松本純:新しい数学モデルを用いた永久磁石同期モータの位置センサレス制御系のロバスト化に関する研究，博士論文，名古屋大学，2014.
[7] 佐藤寿倫:知識ベース 6 群:コンピュータ －基礎理論とハードウェア，電子情報通信学会，2010.
[8] Floyd, Robert W. ”Algorithm 97: shortest path.” Communications of the ACM 5.6 (1962): 345.
[9] 日本工業出版:I-PD 制御における制御周期 $\Delta$t 変更時の制御応答比較，http://www.nikko-pb.co.jp/products/k_data/28P_31P_13.pdf