Vol.2019-ARC-240 No.40
Vol.2019-SLDM-191 No.40
Vol.2019-EMB-53 No.40
2019/2/28

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

于 文博[1,a)]　磯部 祥尚[2]　枝廣 正人[1]

概要：制御システムの大規模・複雑化にともない，並列化によって高速処理を可能にするメニーコアによる並列制御が注目されている．しかし，並列動作には実行順序逆転やデッドロックなど，逐次動作にはない特有の問題があり，人手による並列化は容易ではない．そこで，我々は制御システムの逐次的なモデルから，その観測的な振舞いを変えずに並列制御の実行コードを自動生成するモデルベース並列化システム MBP を開発している．本論文では，MBP の単層制御モデル用の並列化アルゴリズムを，共有メモリと階層構造をもつ制御モデルの並列化にも対応できるように拡張する．さらに，その拡張並列化アルゴリズムによって生成される並列制御の振舞いを形式仕様記述言語 CSP で厳密に記述し，その並列化によって実行順序逆転とデッドロックが発生しないことをモデル検査器 FDR で網羅的に検査する方法を与える．

1. はじめに

近年，組込みシステムの制御分野においてモデルベース開発が主流となっている．モデルベース開発では，システムを可読性の良いモデルで表現することによって，大規模な制御システムを効率よく開発できる．特に MathWorks 社の MATLAB/Simulink[1] を用いたモデルベース開発が広く利用されている．Simulink の制御モデルは，**処理ブロック**と**信号線**を組み合わせて制御システムの動作を表す**ブロック線図**であり，処理ブロックは制御システムの構成要素や機能，信号線は処理ブロック間の信号の流れを表す．Simulink では，大規模な制御システムを可読性よくモデル化するために，複数の**サブシステム**から構成される大規模な制御システムを**階層的**に表現する機能や，処理ブロック間でのデータの受渡しを可能にする**共有メモリ**を表現する機能も提供されている．

一方，制御システムの大規模化にともない，シングルコアプロセッサによる逐次制御が性能的に限界に近づいてきている．そこで，並列化によって高速処理を可能にするメニーコアプロセッサによる並列制御が注目を集めている．しかし，シングルコア用に設計された制御モデルをメニーコア上で並列動作させるためには，制御用の処理ブロック間の依存性などを十分に理解して並列化（各処理ブロックのコア割当てとコア間通信の挿入等）を行う必要があり，理解が不十分なまま並列化すれば，実行順序逆転やデッドロックなどの並列動作特有の問題が発生する可能性がある．このような並列動作特有の問題は発生確率や再現性が低いことが多く，テストによる発見が困難である．

我々は，Simulink の制御モデルから，その観測的な振舞いを変えないように，並列制御コードを自動生成する**モデルベース並列化システム** MBP を開発している [2][3]．このモデルベース並列化システム MBP は，Simulink の制御モデルからブロック間の依存関係を抽出し，その依存関係を満たすように処理ブロックのコア割当とコア間通信の挿入を行い，マルチコア用の並列制御モデルを生成する．この MBP による並列化によって振舞いが変わらないことを確認するために，山本等 [4] は，生成される並列制御モデルが元の制御モデルの処理ブロック間の依存関係を満たしていることを網羅的に検査する方法を与えた．また，多門等 [5] は同期通信版の並列化アルゴリズムによって実行順序逆転が起こらないことを証明した．しかし，現在のモデルベース並列化システム MBP では，共有メモリ付階層型制御モデルの並列化には十分に対応していない．

大規模で複雑な制御システムを可読性よくモデル化するために，サブシステム分割による階層化や共有メモリによるデータの受け渡しは有効であり，Simulink もその機能を

[1] 名古屋大学大学院情報学研究科
[2] 産業技術総合研究所
[a)] yuwenbo@ertl.jp

Vol.2019-ARC-240 No.40
Vol.2019-SLDM-191 No.40
Vol.2019-EMB-53 No.40
2019/2/28

提供している．そこで，本論文では，MBP の単層制御モデル用の並列化アルゴリズムを，共有メモリと階層構造をもつ制御モデルの並列化にも対応できるように拡張する．さらに，その拡張並列化アルゴリズムによって生成される並列制御の振舞いを**形式仕様記述言語 CSP**[8] で厳密に記述し，その並列化によって実行順序逆転とデッドロックが発生しないことを，CSP の代表的な**モデル検査器**である **FDR** [6] を用いて網羅的に検査する方法を与える．ここで，実行順序逆転が発生しないとは，元の制御モデルにおいて依存関係のある処理ブロック（共有メモリへのアクセスも含む）の実行順序を変えないことである．依存関係のない処理ブロックの実行順序の逆転は許容することによって，並列化による高速化が可能になる．この拡張並列化アルゴリズムを FDR のモデル記述言語 CSP_M でコード化し，実際に共有メモリ付階層型制御モデルの例に適用して，並列制御モデルを網羅的に検査した実験結果についても報告する．

なお，本論文では，フィードバックループをもたない制御モデルを並列化の対象にしている．ループをもつ場合は 1 周期分を抽出して展開するなど，並列化の前に処理が必要になるが，それについては今後の課題である．

以降, 2 節では共有メモリをもつ階層的な Simulink モデルについて説明し，3 節では並列制御モデルを厳密に記述するための形式仕様記述言語 CSP を紹介する．4 節では，共有メモリ付階層型制御モデルの並列化を可能にするために，単層制御モデルの並列化アルゴリズムを拡張し，その記述を形式的（明確かつ厳密）に与える．5 節では，拡張並列化アルゴリズムによって生成される並列制御モデルの正しさを検証するための仕様記述を与え，その検証方法を説明する．6 節では，拡張並列化アルゴリズムをモデル検査器 FDR の入力言語 CSP_M でコード化し，複数の制御モデルの例から並列制御モデルを生成して，その振舞いを網羅的に検査した実験結果について報告する．7 節でまとめと今後の課題について述べる．

2. 共有メモリ付階層型制御モデル

本節では，本論文の並列化アルゴリズムの入力となる MATLAB/Simulink の共有メモリ付階層型制御モデルについて説明する．最初に 2.1 小節で Simulink の制御モデルを紹介する．次に，2.2 小節で制御モデルの階層構造を構成するサブシステムについて説明し，2.3 小節でブロック間でデータを受け渡すための共有メモリについて説明する．

2.1 Simulink の制御モデル（ブロック線図）

MATLAB/Simulink とは MathWorks 社が開発したソフトウェアである．MATLAB は数値計算言語用のソフトウェアであり，Simulink は MATLAB の拡張ツールのひとつである．図 1 に Simulink で作成した制御モデルの例を示す．図 1 に示すように，Simulink の制御モデルは，**処理ブロック**と**信号線**を組み合わせて制御システムの動作を表す**ブロック線図**であり，処理ブロックは制御システムの構成要素や機能，信号線は処理ブロック間の信号の流れを表す．MATLAB/Simulink では，このように作成した制御モデルの振舞いをシミュレーション等で確認することができる．

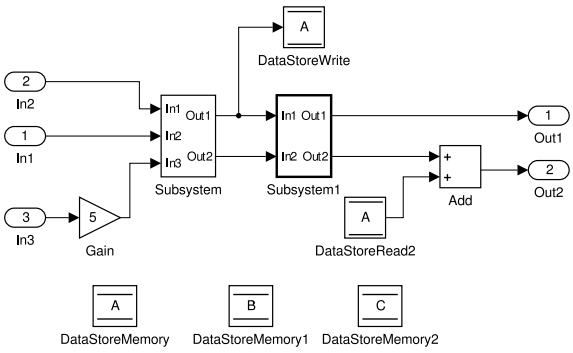

図 1 Simulink の制御モデル（ブロック線図）の例

2.2 制御モデルの階層化（サブシステム）

図 1 中の `Subsystem` と `Subsystem1` に示すように，Simulink の制御モデルでは，一部の振舞いを**サブシステム**としてモジュール化して階層化し，大規模な制御システムを可読性よくモデル化できる．サブシステムには `Atomic` 型と `Virtual` 型の 2 つの型がある．

`Atomic` 型サブシステムでは，全ての入力ポートで受信可能になるまで内部の処理を延期（入力同期）し，内部の処理が完了するまで全ての出力ポートからの送信を延期（出力同期）する．例えば，図 1 の `Atomic` 型サブシステム `Subsystem1` では，2 つの入力ポート `In1` と `In2` の両方で受信可能になるまで `Subsystem1` 内部の処理は開始されない（入出力の同期をとる場合などに利用される）．

一方，`Virtual` 型サブシステムは，可読性を向上させるためだけの記法であり，振舞いには影響を与えない．すなわち，`Virtual` 型サブシステムのブロックをその内部のブロック線図で置換しても振舞いは変わらない．図 1 中の `Subsystem` は `Virtual` 型である．

2.3 共有メモリ

制御モデルに使われる処理ブロックには**基本ブロック**と**メモリブロック**の 2 つの種類がある．基本ブロックとは，信号処理するための通常のブロックであり，例えば，図 1 の `Gain` や `Add` などは基本ブロックである．メモリブロックとは，共有メモリにアクセスするためのブロックであり，書込み用のメモリブロック（例えば，図 1 の `DataStoreWrite`）と読出し用のメモリブロック（例えば，図 1 の `DataStoreRead2`）がある．

共有メモリを使用可能にするためには，名前付きの共有

Vol.2019-ARC-240 No.40
Vol.2019-SLDM-191 No.40
Vol.2019-EMB-53 No.40
2019/2/28

表 1　CSP プロセスの構文

CSP プロセス	名前	
A	プロセス名	($A \in$ PName)
STOP	停止	
SKIP	成功終了	
$a \to P$	プレフィクス	($a \in$ Events)
$P \,\square\, Q$	外部選択	
$P \sqcap Q$	内部選択	
if c then P else Q	条件分岐	($c \in$ Bool)
c & P	ガード	($c \in$ Bool)
P ; Q	逐次合成	
P [| X |] Q	並列合成	($X \subseteq$ Events)
$P \,\|\|\|\, Q$	インターリービング	
$P \setminus X$	隠蔽	($X \subseteq$ Events)

データストアメモリを定義･初期化する必要がある（例えば，図 1 の DataStoreMemory）．共有データストアメモリとは，同じメモリ名を指定するメモリブロックが使用できるメモリ領域である．例えば，図 1 の DataStoreMemory は共有データストアメモリ A を定義･初期化し，DataStoreWrite は Subsystem の Out1 からのデータを共有メモリ A に書き込み，DataStoreRead2 は共有メモリ A からデータを読み込んで Add にそのデータを渡す振舞いを表している．

3. 形式仕様記述言語 CSP

本論文では，並列化アルゴリズムの形式的な記述と検証に**仕様記述言語 CSP**（Communicating Sequential Processes）を用いる．CSP は C.A.R.Hoare によって考案されたプロセス代数であり [8]，並列動作を形式的に記述し，解析するための理論である．本節では，3.1 小節で CSP による並列動作の記法（CSP プロセス），3.2 小節で CSP プロセス間の詳細化関係について簡単に説明する．詳細については，文献 [9] 等を参考にしてほしい．

3.1 CSP プロセス

CSP における動作の最小単位は**イベント**であり，本論文ではイベントの集合を Events で表す．例えば，本研究の並列制御モデルでは，コア間通信やメモリ読出し/書込みなどがイベントである．また，CSP では繰返し動作を表現するために**プロセス名**を用いており，本論文ではプロセス名の集合を PName で表す．

このとき，**CSP プロセス**の集合 Proc は表 1 の式を含む最小の集合である．ここで，表 1 の P, Q はすでに Proc の要素であるとする．また，Bool は論理式の集合である．

以下，表 1 の各プロセスの意味を簡単に説明する．

- 停止 STOP はこれ以上何も実行しないプロセスであり，デッドロックに相当する．これに対し，成功終了 SKIP はプロセスが正常に終了したことを表す．
- プレフィクス ($a \to P$) はイベント a を実行でき，a を実行後はプロセス P のように動作するプロセスである．なお，CSP プロセスは，処理全体の名前ではなく，処理のひとつの状態を表している．例えば，状態 ($a \to P$) ではイベント a を実行でき，実行後は状態 P に遷移する，と言い換えることもできる．
- 外部選択 ($P \,\square\, Q$) はプロセス P または Q のように動作することを表し，その選択はイベントによって外部から制御できる．一方，内部選択 ($P \sqcap Q$) もプロセス P または Q のように動作することを表すが，その選択は内部で自動的に行われるため，外部からは制御できない．
- 条件分岐（if c then P else Q）は条件 c の真偽値による動作の分岐である．ガード (c & P) は条件分岐の特殊な場合（$Q =$ STOP）であり，条件 c が真の場合は P を実行し，偽の場合は停止するプロセスである．
- 逐次合成 ($P;Q$) は，最初にプロセス P を実行し，P が成功終了した後，Q を実行するプロセスである（P が成功終了しない場合，Q は実行されない）．
- 並列合成 (P [| X |] Q) は P と Q が X に含まれるイベントで同期しながら並列に動作するプロセスである．インターリービング ($P \,|||\, Q$) は，並列合成の特殊な場合（$X = \emptyset$）であり，同期することなく P と Q が並列に動作するプロセスである．
- 隠蔽 ($P \setminus X$) は X に含まれるイベントが内部で自動的に実行される（外部からは隠される）以外，P のように動作するプロセスである．

表 1 の他に，3 つ以上の外部選択を簡単に記述するために，次の複製型の略記法も用いる．

$$\square\, x : X \,@\, P(x) = \begin{cases} \texttt{STOP} & (X = \{\}) \\ P(1) \,\square \cdots \square\, P(n) & (X = \{1, ..., n\}) \end{cases}$$

ここで，X はインデックス集合である．同様に，内部選択とインターリービングについても，各々の複製型の略記法 $\sqcap\, x : X \,@\, P(x)$ と $|||\, x : X \,@\, P(x)$ を用いる．

逐次合成については，実行するプロセスの順番を指定する必要があるため，インデックス列 S に対して，次の複製型の略記法を用いる．

$$;\; x : S \,@\, P(x) = \begin{cases} \texttt{SKIP} & (S = \langle\rangle) \\ P(1); \cdots ; P(n) & (S = \langle 1, ..., n \rangle) \end{cases}$$

以上で説明したように，CSP の基本はイベント同期であるが，次の略記法に示すように，チャネル ch へ値 v を送信するイベント $ch!v$ と，チャネル ch から受信した値を変数 x に代入するイベント $ch?x$ を用いて，チャネルを用いたデータの受け渡しも記述することができる．

$$ch!v \to P = ch.v \to P$$
$$ch?x \to P = \;\square\, x : \{v \mid ch.v \in \texttt{Events}\} \,@\, (ch.x \to P)$$

CSP では，送信と受信のイベントが共に実行可能なとき

Vol.2019-ARC-240 No.40
Vol.2019-SLDM-191 No.40
Vol.2019-EMB-53 No.40
2019/2/28

に限り実行できる同期通信を基本とするが，4.5 小節で説明するように，バッファプロセスやメモリプロセスを用いて，非同期通信や共有メモリも表現可能である．

3.2 失敗発散詳細化関係

複数のプロセスの振舞いが相互に影響する並列動作では，プロセス間のタイミングのずれによって，デッドロックなどの並列動作特有の不具合が発生する可能性がある．このような不具合の再現性は低いため，実装後にテストで発見することは困難である．CSP では，並列動作を形式的に（厳密に）記述し，その振舞いを検証することが可能である．

CSP による並列動作の代表的な検証方法は，**失敗発散詳細化**（Failures-Divergences Refinement）と呼ばれる関係の真偽を判定することである．失敗発散詳細化は 2 つの CSP プロセス P, Q の関係であり，Q が P の失敗発散詳細化であることを $P \sqsubseteq_{\mathrm{FD}} Q$ と書く．基本的には，期待する振舞い（例えば，逐次的な仕様）Spec と，実装に近い振舞い（例えば，並列動作）Impl を，各々 CSP プロセスとして記述し，失敗発散詳細化関係 $\mathtt{Spec} \sqsubseteq_{\mathrm{FD}} \mathtt{Impl}$ の真偽を判定する．

失敗発散詳細化関係 $P \sqsubseteq_{\mathrm{FD}} Q$ が成立つとき，次のことが保証される．

- P がデッドロックフリーかつライブロックフリーならば，Q もデッドロックフリーかつライブロックフリーである．
- P が実行可能なトレース（観測されるイベントの列）以外のトレースを Q は実行しない．言い換えれば，許容されるトレースのみを P で表現しておけば，Q は許容されないトレースを実行することはない．すなわち，Q が危険なイベントを実行しないことを保証できる．例えば，並列制御モデルでは実行順序が逆転しないことを保証できる．
- Q があるトレースを実行でき，かつ，その実行後の状態 Q' が安定ならば，P も同じトレースで安定な状態 P' に到達でき，P' で実行可能な任意のイベントを Q' も実行できる．ここで，状態が安定とは，内部選択や隠蔽によって外部からは観測できない状態変化が起こらない状態である．すなわち，Q が実行すべきイベントを実行することを保証できる．例えば，並列制御モデルでは全処理ブロックが実行されることを保証できる．

例えば，次の詳細化関係が成り立つ．

$$(a \to b \to \mathtt{STOP}) \,\Box\, (a \to c \to \mathtt{STOP}) \sqsubseteq_{\mathrm{FD}} (a \to b \to \mathtt{STOP})$$
$$(a \to \mathtt{STOP}) \,\sqcap\, (b \to \mathtt{STOP}) \sqsubseteq_{\mathrm{FD}} (a \to \mathtt{STOP})$$

上記の 1 番目の例は a による非決定的な選択であるため，その選択肢のどちらかを実行可能であれば，失敗発散詳細化関係は成立つ．また，2 番目の例は内部選択（安定でない状態）が使われているため，どちらかの選択肢（安定な状態）を選んで詳細化できる．なお，2 番目の内部選択を外部選択に置き換えると，選択前の状態も安定になるため，右辺も a と b の両方を実行できる必要があり，失敗発散詳細化関係は成り立たなくなる．

$$(a \to \mathtt{STOP}) \,\Box\, (b \to \mathtt{STOP}) \not\sqsubseteq_{\mathrm{FD}} (a \to \mathtt{STOP})$$

失敗発散詳細化関係を利用した簡単な検証例をひとつ紹介する．CSP プロセス P がデッドロックフリーかつライブロックフリーであることは，次の詳細化関係が真であることを示すことによって保証できる．

$$\mathtt{DLfree} \sqsubseteq_{\mathrm{FD}} P$$

ここで，DLfree は，観測できないイベントによる無限ループをもたず（ライブロックフリー），常に少なくとも 1 つのイベントを実行可能であるか，または成功終了する（デッドロックフリー）次のプロセスである．

$$\mathtt{DLfree} = (\,\sqcap\, a : \mathtt{Events} \,@\, a \to \mathtt{DLfree}) \,\sqcap\, \mathtt{SKIP}$$

4. 並列化アルゴリズムの拡張と形式化

本節では，Simulink の共有メモリ付階層型制御モデルから並列制御モデルを生成するための拡張並列化アルゴリズム Algo_Extended_Para_Ctrl について説明する．拡張並列化アルゴリズム Algo_Extended_Para_Ctrl は共有メモリ付階層型制御モデル Ctrl_Model を受け取り，メモリ付並列制御モデル Para_Ctrl（CSP プロセス）を返す関数である．

$$\begin{aligned}\mathtt{Para\_Ctrl} &= \mathtt{Algo\_Extended\_Para\_Ctrl}(\mathtt{Ctrl\_Model})\\ &= \mathtt{Algo\_Para\_Ctrl}(\\ &\quad \mathtt{Algo\_Add\_Deps}(\\ &\quad\quad \mathtt{Algo\_Rem\_OW}(\\ &\quad\quad\quad \mathtt{Algo\_Flatten}(\mathtt{Ctrl\_Model}))),\\ &\quad \mathtt{elm}_4(\mathtt{Ctrl\_Model}))\end{aligned}$$

ここで，Algo_Flatten は制御モデルの階層構造を平坦化するためのアルゴリズム，Algo_Rem_OW は上書きされるブロックを削除するためのアルゴリズム，Algo_Add_Deps は共有メモリによる間接的な依存関係を追加するアルゴリズム，Algo_Para_Ctrl は制御を並列化するためのアルゴリズムであり，$\mathtt{elm}_4$ は Ctrl_Model から 4 番目の要素（コア数）を抽出する関数である．以下の小節で，入力 Ctrl_Model と各アルゴリズムについて順番に説明する．

4.1 拡張並列化アルゴリズムへの入力形式

本小節では，拡張並列化アルゴリズムへの入力として，共有メモリ付階層型制御モデルの形式的な記述について説

Vol.2019-ARC-240 No.40
Vol.2019-SLDM-191 No.40
Vol.2019-EMB-53 No.40
2019/2/28

明する．

4.1.1 共有メモリ付階層型制御モデルの形式化

共有メモリ付階層型制御モデル Ctrl_Model は，単層システムの集合 Systems，ブロック実行列 $\mathtt{Bseq}_0$，コア割当関数 $\mathtt{Asn}_0$，コア数 Cnum の四つ組である．

$$\text{入力 } \mathtt{Ctrl\_Model} = (\mathtt{Systems}, \mathtt{Bseq}_0, \mathtt{Asn}_0, \mathtt{Cnum})$$

単層システムの集合 Systems （本論文では，その要素である単層システムを変数 sys で表す）は，制御モデルを構成する複数の単層システムの集合であり，ひとつの単層システムは制御モデルの 1 層分のブロック線図を表す．その形式化については 4.1.2 小々節で詳しく説明する．

ブロック実行列 $\mathtt{Bseq}_0$ は，MATLAB/Simulink が自動生成する逐次的な C プログラムによって実行される処理ブロックの実行順序（処理ブロックの列）である．

$$\begin{aligned}&\text{ブロック実行列}\\ &\mathtt{Bseq}_0 = \langle b_1, \ldots, b_n \rangle \quad (b_1, \ldots, b_n \in \mathtt{Blks}_0)\end{aligned}$$

ここで，$\mathtt{Blks}_0$ は制御モデルに使われている単層システムの集合 Systems に含まれる全処理ブロックの集合であり，その定義は 4.2.1 小々節で説明する．

コア数 Cnum は並列制御モデルを実行するための並列度（メニーコアのコア数）であり，コア ID の集合 Cnums は $\{0, \ldots, \mathtt{Cnum} - 1\}$ で与えられる．このとき，コア割当関数（$\mathtt{Asn}_0 :: \mathtt{Cnums} \to \mathtt{Blks}$）は，コア $c \in \mathtt{Cnums}$ に割り当てる処理ブロックの集合を指定するための関数であり，次の条件を満たすとする．

- $\forall c, c' \in \mathtt{Cnums}.\ c \neq c' \to \mathtt{Asn}_0(c) \cap \mathtt{Asn}_0(c') = \emptyset$
- $\bigcup_{c \in \mathtt{Cnums}} \mathtt{Asn}_0(c) = \mathtt{Blks}_0$

4.1.2 単層システムの形式化

単層システムは Simulink の制御モデルの 1 層分のブロック線図であり，基本ブロック集合 $bscs$，名前付書込み用メモリブロック集合 mws，名前付読出し用メモリブロック集合 mrs，入力ポート集合 ins，出力ポート集合 $outs$，依存関係 $deps$，単層システムの型 $type$ の七つ組である．

$$\text{単層システム } sys = (bscs, mws, mrs, ins, outs, deps, type)$$

基本ブロック集合 $bscs$ は，この単層システム sys に含まれるすべての基本ブロックの名前（例えば，図 1 の Add 等）の集合である．なお，本論文では，この単層システム sys に含まれる他の単層システム（サブシステム）の入出力ポートの名前（例えば，図 1 の Subsystem_In1, Subsystem1_Out1 等）も基本ブロック集合 $bscs$ に含まれるので注意してほしい．名前付書込み用メモリブロック集合 mws は，書込み用メモリブロックの名前 b_i とメモリの名前 m_i の組の集合 $\{(b_1, m_1), \ldots, (b_n, m_n)\}$ であり，名前付読出し用メモリブロック集合 mrs も同様である．入力ポート集合 ins はこの単層システム sys の入力ポートの名前の集合，出力ポート集合 $outs$ は sys の出力ポートの名前の集合である．ここで，基本ブロック，書込み/読出し用メモリブロック，入力/出力ポートの名前は重複することはなく（各名前集合は互いに素），制御モデル全体で唯一に定まる大域的な名前（ID 番号等）であるとする．

単層システム sys に含まれる各名前集合（基本ブロック集合 $\mathtt{bscs}(sys)$，書込み用メモリブロック集合 $\mathtt{bws}(sys)$，書込み用メモリブロック集合 $\mathtt{brs}(sys)$，入力ポート集合 $\mathtt{pins}(sys)$，出力ポート集合 $\mathtt{pouts}(sys)$，入出力ポート集合 $\mathtt{pios}(sys)$）は次の関数により抽出できる．

$$\begin{aligned}\mathtt{bscs}(sys) &= \mathtt{elm}_1(sys)\\ \mathtt{bws}(sys) &= \{b \mid \exists m.\ (b, m) \in \mathtt{elm}_2(sys)\}\\ \mathtt{brs}(sys) &= \{b \mid \exists m.\ (b, m) \in \mathtt{elm}_3(sys)\}\\ \mathtt{pins}(sys) &= \mathtt{elm}_4(sys)\\ \mathtt{pouts}(sys) &= \mathtt{elm}_5(sys)\\ \mathtt{pios}(sys) &= \mathtt{pins}(sys) \cup \mathtt{pouts}(sys)\end{aligned}$$

ここで，$\mathtt{elm}_n(t)$ は組 t から n 番目（一番左が 1 番目）の要素を抽出するための関数である．処理ブロックは基本ブロックとメモリブロックから構成されるため，単層システム sys に含まれる全ての処理ブロック集合は次の関数により得られる．

$$\mathtt{blks}(sys) = \mathtt{bscs}(sys) \cup \mathtt{bws}(sys) \cup \mathtt{brs}(sys)$$

単層システム sys の 6 番目の要素 $deps$（依存関係）は，この単層システム内のブロック線図のブロック間の信号線（矢印）の集合である．

$$\begin{aligned}\text{依存関係 } deps = \{(b_1, b_1'), \ldots, (b_n, b_n')\}\\ \subseteq\ (\mathtt{blks}(sys) \times \mathtt{blks}(sys))\\ \cup\ (\mathtt{pins}(sys) \times \mathtt{blks}(sys))\\ \cup\ (\mathtt{blks}(sys) \times \mathtt{pouts}(sys))\end{aligned}$$

ここで，$(b, b') \in deps$ は，ブロック b からブロック b' への信号線（b' は b の出力に依存すること）を意味する．単層システム sys に含まれる依存関係 $\mathtt{deps}(sys)$ は次の関数により抽出できる．

$$\mathtt{deps}(sys) = \mathtt{elm}_6(sys)$$

単層システムの型 $type$ は 2.2 小節で説明した Virtual 型または Atomic 型である．

$$\text{単層システムの型 } type \in \{\mathtt{Atomic}, \mathtt{Virtual}\}$$

以降，次に定義するように，Atomic 型のサブシステムの集合（Systems の部分集合）を AtmSystems と書く．

$$\mathtt{AtmSystems} = \{sys \in \mathtt{Systems} \mid \mathtt{elm}_7(sys) = \mathtt{Atomic}\}$$

Vol.2019-ARC-240 No.40
Vol.2019-SLDM-191 No.40
Vol.2019-EMB-53 No.40
2019/2/28

4.2 平坦化アルゴリズム

本小節では，入出力ポートをバイパスして，階層型の制御モデルを単層の制御モデルに変換するためのアルゴリズム Algo_Flatten について説明する．アルゴリズム Algo_Flatten は，4.1 小節で説明したメモリ付階層型制御モデル Ctrl_Model を受け取り，メモリ付単層型制御モデル $\mathtt{Flat\_Model}_0$ を返す関数である．

$$\mathtt{Flat\_Model}_0 = \mathtt{Algo\_Flatten}(\mathtt{Ctrl\_Model})$$

ここで，$\mathtt{Flat\_Model}_0$ は，処理ブロック集合 $\mathtt{Blks}_0$，名前付書込み用メモリブロック集合 $\mathtt{MWs}_0$，名前付読出し用メモリブロック集合 $\mathtt{MRs}_0$，依存関係 $\mathtt{Deps}_0$，ブロック実行列 $\mathtt{Bseq}_0$，コア割当関数 $\mathtt{Asn}_0$ の六つ組である．

メモリ付単層制御モデル

$$\mathtt{Flat\_Model}_0 = (\mathtt{Blks}_0, \mathtt{MWs}_0, \mathtt{MRs}_0, \mathtt{Deps}_0, \mathtt{Bseq}_0, \mathtt{Asn}_0)$$

ここで，$\mathtt{Bseq}_0$ と $\mathtt{Asn}_0$ は Ctrl_Model のブロック実行列 $\mathtt{Bseq}_0$ とコア割当関数 $\mathtt{Asn}_0$ と同じである．以下，それ以外の $\mathtt{Flat\_Model}_0$ の各要素の導出方法について説明する．

4.2.1 処理ブロック集合と名前付メモリブロック集合

メモリ付単層型制御モデル $\mathtt{Flat\_Model}_0$ の処理ブロック集合 $\mathtt{Blks}_0$ は単層システム集合 Systems（制御モデル全体）に含まれる全ての処理ブロック集合である．すなわち，全ての単層システムの基本ブロック集合とメモリブロック集合の和集合であるが，各単層システムの基本ブロック集合には，その内部の他の単層システム（サブシステム）の入出力ポートが含まれるため，平坦化後は入出力ポートを削除する必要がある．すなわち，$\mathtt{Blks}_0$ は次のように求められる．

$$\mathtt{Blks}_0 = (\mathtt{Bscs}_0 \cup \mathtt{BWs}_0 \cup \mathtt{BRs}_0) - \mathtt{PIOs}_0$$

ここで，$\mathtt{Bscs}_0$，$\mathtt{BWs}_0$，$\mathtt{BRs}_0$，$\mathtt{PIOs}_0$ は，各々，全ての基本ブロック集合，書込み用メモリブロック集合，読出し用メモリブロック集合，入出力ポート集合であり，これらは次のように求められる．

$$\begin{aligned}\mathtt{Bscs}_0 &= \textstyle\bigcup_{sys \in \mathtt{Systems}} \mathtt{bscs}(sys)\\ \mathtt{BWs}_0 &= \textstyle\bigcup_{sys \in \mathtt{Systems}} \mathtt{bws}(sys)\\ \mathtt{BRs}_0 &= \textstyle\bigcup_{sys \in \mathtt{Systems}} \mathtt{brs}(sys)\\ \mathtt{PIOs}_0 &= \textstyle\bigcup_{sys \in \mathtt{Systems}} \mathtt{pios}(sys)\end{aligned}$$

右辺の各単層システムに関する集合（$\mathtt{bscs}(sys)$ 等）は 4.1.2 小々節で定義済みである．

名前付書込み用と名前付読出し用のメモリブロック集合 $\mathtt{MWs}_0$ と $\mathtt{MRs}_0$ は単層システム集合 Systems 内の名前付書込み用と名前付読出し用のメモリブロック集合の和集合である．

$$\begin{aligned}\mathtt{MWs}_0 &= \textstyle\bigcup_{sys \in \mathtt{Systems}} \mathtt{elm}_2(sys)\\ \mathtt{MRs}_0 &= \textstyle\bigcup_{sys \in \mathtt{Systems}} \mathtt{elm}_3(sys)\end{aligned}$$

Atomic 型のサブシステムでは入出力を同期する必要があるため，同期すべき入出力ポート集合の集合も次のように定義しておく．

$$\begin{aligned}\mathtt{AtmIns}_0 &= \textstyle\bigcup_{sys \in \mathtt{AtmSystems}} \mathtt{elm}_4(sys)\\ \mathtt{AtmOuts}_0 &= \textstyle\bigcup_{sys \in \mathtt{AtmSystems}} \mathtt{elm}_5(sys)\\ \mathtt{AtmIOs}_0 &= \mathtt{AtmIns}_0 \cup \mathtt{AtmOuts}_0\end{aligned}$$

4.2.2 依存関係

メモリ付単層型制御モデル $\mathtt{Flat\_Model}_0$ の依存関係 $\mathtt{Deps}_0$ については，依存関係の和集合をとるだけなく，(1) 2.2 小節で説明したように Atomic 型の単層システムの入出力を同期するための依存関係の追加，(2) 複数の単層システムを接続する入出力ポートをバイパスする依存関係の追加，(3) 入出力ポートに関する依存関係の削除を行う必要がある．

まず，依存関係の和集合をとる．

$$\mathtt{UDeps}_0 = \textstyle\bigcup_{sys \in \mathtt{Systems}} \mathtt{deps}(sys)$$

次に，Atomic 型の単層システムの入出力を同期させるために追加する依存関係 $\mathtt{AtmDeps}_0$ を定義する．

$$\begin{aligned}\mathtt{AtmDeps}_0 = \quad & \{(b_1, b) \mid \exists ins \in \mathtt{AtmIns}_0.\ b_1 \in ins,\\ & \qquad (\exists b_2 \in ins.\ (b_2, b) \in \mathtt{UDeps}_0)\}\\ \cup\ & \{(b, b_1) \mid \exists outs \in \mathtt{AtmOuts}_0.\ b_1 \in outs,\\ & \qquad (\exists b_2 \in outs.\ (b, b_2) \in \mathtt{UDeps}_0)\}\end{aligned}$$

さらに，上記の $\mathtt{UDeps}_0 \cup \mathtt{AtmDeps}_0$ に対して入出力ポートをバイパスする依存関係を追加する．

$$\mathtt{TCDeps}_0 = \mathtt{trns\_cls}(\mathtt{UDeps}_0 \cup \mathtt{AtmDeps}_0, \mathtt{PIOs}_0)$$

ここで，$\mathtt{PIOs}_0$ は全ての入出力ポートの集合（4.2.1 小々節で定義済）であり，$\mathtt{trns\_cls}(r, S)$ は，次のように定義される条件付（仲介する値 y が S に含まれること）の推移閉包である．

$$\begin{aligned}\mathtt{trns\_cls}(r, S) &= \textstyle\bigcup_{n \geq 0} \mathtt{tc}_{(n)},\\ \mathtt{tc}_{(0)} &= r,\\ \mathtt{tc}_{(n+1)} &= \{(x, z) \mid \exists y \in S.\ (x, y), (y, z) \in \mathtt{tc}_{(n)}\}\end{aligned}$$

最後に，上記の $\mathtt{TCDeps}_0$ から入出力ポートに関係する依存関係を削除して，メモリ付単層型制御モデル $\mathtt{Flat\_Model}_0$ の依存関係 $\mathtt{Deps}_0$ を得る．

$$\mathtt{Deps}_0 = \mathtt{TCDeps}_0 - \mathtt{mkdeps}(\mathtt{Blks}_0 \cup \mathtt{PIOs}_0,\ \mathtt{PIOs}_0)$$

ここで，$\mathtt{mkdeps}(B, B')$ は 2 つのブロック集合 B, B' に関係する依存関係を生成する関数である．

$$\mathtt{mkdeps}(B, B') = \{(b, b'), (b', b) \mid b \in B, b' \in B'\}$$

Vol.2019-ARC-240 No.40
Vol.2019-SLDM-191 No.40
Vol.2019-EMB-53 No.40
2019/2/28

4.3 上書きブロック削除アルゴリズム

本小節では，メモリ付単層型制御モデル $\mathtt{Flat\_Model}_0$ において，上書きされる無効な書込み用ブロックを削除するためのアルゴリズム Algo_Rem_OW について説明する．アルゴリズム Algo_Rem_OW は，4.2 小節で説明したメモリ付単層型制御モデル $\mathtt{Flat\_Model}_0$ を受け取り，上書き削除済みのメモリ付単層型制御モデル $\mathtt{Flat\_Model}_1$ を返す関数である．

$$\mathtt{Flat\_Model}_1 = \mathtt{Algo\_Rem\_OW}(\mathtt{Flat\_Model}_0)$$

ここで，$\mathtt{Flat\_Model}_1$ は，処理ブロック集合 $\mathtt{Blks}_1$，名前付書込み用メモリブロック集合 $\mathtt{MWs}_1$，名前付読出し用メモリブロック集合 $\mathtt{MRs}_1$，依存関係 $\mathtt{Deps}_1$，ブロック実行列 $\mathtt{Bseq}_1$，コア割当関数 $\mathtt{Asn}_1$ の六つ組である．

上書き削除済メモリ付単層制御モデル
$$\mathtt{Flat\_Model}_1 = (\mathtt{Blks}_1, \mathtt{MWs}_1, \mathtt{MRs}_1, \mathtt{Deps}_1, \mathtt{Bseq}_1, \mathtt{Asn}_1)$$

以下，$\mathtt{Flat\_Model}_1$ の各要素の導出方法について説明する．なお，本小節での $\mathtt{Blks}_0, \mathtt{MWs}_0, \mathtt{MRs}_0, \mathtt{Deps}_0, \mathtt{Bseq}_0, \mathtt{Asn}_0$ は $\mathtt{Flat\_Model}_0$ の要素であるとする．

4.3.1 上書きされる書込み用メモリブロックの集合

上書きされる書込み用メモリブロックの集合 OW_Blks は次のように求められる．

$$\mathtt{OW\_Blks} = \{b \mid \exists b'.\ \mathtt{chk\_MRW}(b, b', \mathtt{MWs}_0, \mathtt{MWs}_0, \mathtt{MRs}_0)\}$$

ここで，$\mathtt{chk\_MRW}(b, b', \mathtt{MWs}_0, \mathtt{MWs}_0, \mathtt{MRs}_0)$ は，次に定義するように，ブロック b がメモリ m に書き込み後，b' が書き込むまでにメモリ m を読み出すブロックがないときに真になる次の関数である（汎用性をもたせるため $\mathtt{MWs}_0, \mathtt{MWs}_0, \mathtt{MRs}_0$ を仮引数 M_1, M_2, M で受けている）．

$$\begin{aligned}&\mathtt{chk\_MRW}(b_1, b_2, M_1, M_2, M)\\ &= (\exists i_1, i_2.\ i_1 < i_2 \wedge \mathtt{Bseq}[i_1] = b_1 \wedge \mathtt{Bseq}[i_2] = b_2\\ &\quad \wedge (\exists m.\ (b_1, m) \in M_1 \wedge (b_2, m) \in M_2\\ &\qquad \wedge \{b \mid \exists i.\ i_1 < i < i_2, \mathtt{Bseq}[i] = b, (b, m) \in M\} = \emptyset))\end{aligned}$$

ここで，$\mathtt{Bseq}[i]$ は i 番目に実行される処理ブロックを表す．

4.3.2 上書きされる書込み用メモリブロックの削除

4.3.1 小々節の集合 OW_Blks を用いて，$\mathtt{Flat\_Model}_0$ の各要素から上書きされる書込み用メモリブロックを削除して，$\mathtt{Flat\_Model}_1$ の各要素を次のように求める．

$$\begin{aligned}\mathtt{Blks}_1 &= \mathtt{Blks}_0 - \mathtt{OW\_Blks}\\ \mathtt{MWs}_1 &= \{(b, m) \in \mathtt{MWs}_0 \mid b \notin \mathtt{OW\_Blks}\}\\ \mathtt{MRs}_1 &= \mathtt{MRs}_0\\ \mathtt{Deps}_1 &= \mathtt{Deps}_0 - \mathtt{mkdeps}(\mathtt{Blks}_0, \mathtt{OW\_Blks})\\ \mathtt{Bseq}_1 &= \langle b \in \mathtt{Bseq}_0 \mid b \notin \mathtt{OW\_Blks}\rangle\\ \mathtt{Asn}_1(c) &= \mathtt{Asn}_0(c) - \mathtt{OW\_Blks}\end{aligned}$$

4.4 メモリ依存関係追加アルゴリズム

Simulink の制御モデルのブロック線図のブロック間の信号線（矢印）は直接のデータの受渡しを表しているが，共有メモリアクセスによるブロック間の依存関係は明記されていない．そのため，同じメモリにアクセスするメモリブロックの順序を並列化後も維持できるように，共有メモリ経由の間接的な依存関係を追加する必要がある．

本小節では，共有メモリ経由の間接的な依存関係を追加するアルゴリズム Algo_Add_Deps について説明する．アルゴリズム Algo_Add_Deps は，4.3 小節で説明した上書き削除済みのメモリ付単層型制御モデル $\mathtt{Flat\_Model}_1$ を受け取り，間接的な依存関係を追加したメモリ付単層型制御モデル Flat_Model を返す関数である．

$$\mathtt{Flat\_Model} = \mathtt{Algo\_Add\_Deps}(\mathtt{Flat\_Model}_1)$$

ここで，Flat_Model は次に示す六つ組であるが，依存関係 Deps 以外は $\mathtt{Flat\_Model}_1$ と同じである（例えば，$\mathtt{Blks} = \mathtt{Blks}_1$ である）．

メモリ依存関係追加済メモリ付単層制御モデル
$$\mathtt{Flat\_Model} = (\mathtt{Blks}, \mathtt{MWs}, \mathtt{MRs}, \mathtt{Deps}, \mathtt{Bseq}, \mathtt{Asn})$$

共有メモリ経由の間接的な依存関係は，$\mathtt{Flat\_Model}_1$ の $\mathtt{MWs}_1, \mathtt{MRs}_1, \mathtt{Deps}_1, \mathtt{Bseq}_1$ から次のように求められる．

$$\begin{aligned}\mathtt{Deps} &= \mathtt{Deps}_1 \cup (\mathtt{MDeps} - \mathtt{FDeps})\\ \mathtt{MDeps} &= \{(b, b') \mid \mathtt{chk\_MRW}(b, b', \mathtt{MWs}_1, \mathtt{MRs}_1, \mathtt{MWs}_1)\\ &\qquad \vee \mathtt{chk\_MRW}(b, b', \mathtt{MRs}_1, \mathtt{MWs}_1, \mathtt{MWs}_1)\})\\ \mathtt{FDeps} &= \mathtt{trns\_cls}(\mathtt{Deps}_1, \mathtt{Blks})\end{aligned}$$

ここで，chk_MRW は 4.3.1 小々節で定義した関数であり，追加されるブロック間の依存関係 $(b, b') \in \mathtt{MDeps}$ は，b と b' の間に別の書き込みはなく，b が書き込んだ後に b' が読み出すか，または b が読み出した後に b' が書き込む条件を満たすブロック b, b' の組である．b と b' の間に別の書き込みがない条件によって，余分な依存関係の追加を回避している．さらに，既存の依存関係 $\mathtt{Deps}_1$ から間接的に導かれる依存関係 FDeps は追加する必要がないため，追加分から FDeps を削除している．

4.5 並列化アルゴリズム（単層用）

4.4 小節までに，並列化に必要なブロック間の依存関係などの情報をメモリ付単層型制御モデル Flat_Model として形式化した．本小節では，ブロック間の依存関係を保存しながら，処理ブロックを複数のコアで並列に実行可能にするアルゴリズム Algo_Para_Ctrl について説明する．アルゴリズム Algo_Para_Ctrl は，4.4 小節で説明したメモリ付単層型制御モデル Flat_Model とコア数 Cnum を受け取り，メモリ付並列制御モデル Para_Ctrl を返す関数である．

$$\mathtt{Para\_Ctrl} = \mathtt{Algo\_Para\_Ctrl}(\mathtt{Flat\_Model}, \mathtt{Cnum})$$

Vol.2019-ARC-240 No.40
Vol.2019-SLDM-191 No.40
Vol.2019-EMB-53 No.40
2019/2/28

$$
\begin{aligned}
&\mathtt{Para\_Ctrl} = (\mathtt{Para\_MB}\ [|\ \mathtt{MCh}\ |]\ \mathtt{Mem}(\langle\rangle)) \setminus \mathtt{MCh}\\
&\quad \mathtt{Para\_MB} = (\mathtt{Para\_Mng}\ [|\ \mathtt{BCh}\ |]\ \mathtt{Para\_Buffs}) \setminus \mathtt{BCh}\\
&\qquad \mathtt{Para\_Mng} = |||\ c : \mathtt{Cnums}\ @\ \mathtt{Blk\_Mng}(c)\\
&\qquad \mathtt{Para\_Buffs} = |||\ c : \mathtt{Cnums}\ @\ \mathtt{Buffs}(c)\\
&\qquad\quad \mathtt{Buffs}(c) = |||\ (b, b') : \mathtt{ICh}(c)\ @\ \mathtt{Buff}(b, b')\\
&\text{where}\\
&\quad \mathtt{MCh} = \{|\mathtt{rch}, \mathtt{wch}|\}\\
&\quad \mathtt{BCh} = \{|\mathtt{ch}, \mathtt{bch}|\}\\
&\quad \mathtt{ICh}(c) = \{(b, b') \mid (b, b', c) \in \mathtt{ch\_deps}\}\\
&\quad \mathtt{ch\_deps}(b, b', c) = ((b, b') \in \mathtt{Deps} \wedge b \notin \mathtt{Asn}(c) \wedge b' \in \mathtt{Asn}(c))
\end{aligned}
$$

図 2 メモリ付並列制御モデル Para_Ctrl の構造

ここで，Para_Ctrl は処理ブロックの並列実行順序に着目した CSP プロセスである．なお，本小節での Blks, MWs, MRs, Deps, Bseq, Asn は Flat_Model の要素であるとする．

メモリ付並列制御モデル Para_Ctrl では次の 6 種類のイベントを使用している．

- rch.$m.x$：メモリ m からデータ x を読み出すイベント
- wch.$m.x$：メモリ m にデータ x を書き込むイベント
- ch.$b.b'$：ブロック b の処理完了を b' に伝えるイベント
- bch.$b.b'$：ch.$b.b'$ のバッファ用イベント
- blk.b：基本ブロック b の完了イベント
- rd.$b.x$：読出し用ブロック b の x 読出し完了イベント

ここで，rch, wch, ch, bch はプロセス間通信用のイベントであり，blk, rd は各処理ブロックの実行が完了したことを観測するための検証用のイベントである．なお，メモリ付並列制御モデル Para_Ctrl の目的は，各イベントの実行順序（処理ブロックが依存関係に反した順序で実行されないことなど）を検証することであり，各ブロックで処理されるデータはモデル化の対象にはしていない．そのため，共有メモリには，実際のデータ（例えば，実数値）の代わりに，書き込んだメモリブロック自身の名前 b を書き込んでいる．これによって，読出し用メモリブロックが共有メモリから読出したデータを過去に書き込んだ書込み用メモリブロックを確認できるようになり，共有メモリを経由した依存関係の正しさを検証可能になる．

メモリ付並列制御モデル Para_Ctrl の構造を図 2 に示す．メモリ付並列制御モデル Para_Ctrl は処理ブロックの実行を管理するプロセス Para_Mng，コア間通信で受信したイベントを保存するバッファ Para_Buffs，共有メモリを管理するプロセス Mem から構成されている．ブロック実行管理のプロセス Para_Mng は各コア c 用の管理プロセス Blk_Mng(c) の並列合成であり，バッファ Para_Buffs は各コア c 用の Buffs(c) の並列合成である．さらに，Buffs(c) は各 $(b, b') \in \mathtt{ICh}(c)$ 用の容量 1 のバッファ Buff(b, b') の並列合成であり，この Buff(b, b') は処理完了イベント ch.$b.b'$ を 1 個保存するためのバッファである．ここで，ICh(c) は，コア c に割り当てられたブロック実行の前にコア間通信が必要になる依存関係 (b, b') の集合である．

$$
\begin{aligned}
&\mathtt{Blk\_Mng}(c) = \ ;\ b : \mathtt{Bseq\_Core}(c)\ @\ \mathtt{Blk\_Exe}(b)\\
&\quad \mathtt{Blk\_Exe}(b) = \mathtt{Blk\_In}(b);\ \mathtt{Blk\_Core}(b);\ \mathtt{Blk\_Out}(b)\\
&\qquad \mathtt{Blk\_In}(b) = \ ;\ b' : \mathtt{Bseq\_In}(b)\ @\ \mathtt{bch}.b'.b \rightarrow \mathtt{SKIP}\\
&\qquad \mathtt{Blk\_Out}(b) = \ ;\ b' : \mathtt{Bseq\_Out}(b)\ @\ \mathtt{ch}.b.b' \rightarrow \mathtt{SKIP}\\
&\qquad \mathtt{Blk\_Core}(b)\\
&\qquad\quad = \quad (\Box\ m : \mathtt{name}(b, \mathtt{MWs})\ @\ \mathtt{wch}.m!b \rightarrow \mathtt{SKIP})\\
&\qquad\qquad \Box\ (\Box\ m : \mathtt{name}(b, \mathtt{MRs})\ @\ \mathtt{rch}.m?b' \rightarrow \mathtt{rd}.b.b' \rightarrow \mathtt{SKIP})\\
&\qquad\qquad \Box\ (b \in \mathtt{Bscs}\ \&\ \mathtt{blk}.b \rightarrow \mathtt{SKIP})\\
&\mathtt{Mem}(mem) = \mathtt{wch}?m?x \rightarrow \mathtt{Mem}(mem[m := x])\\
&\qquad\qquad \Box\ \mathtt{rch}?m!mem[m] \rightarrow \mathtt{Mem}(mem)\\
&\mathtt{Buff}(b, b') = \mathtt{ch}.b.b' \rightarrow \mathtt{bch}.b.b' \rightarrow \mathtt{Buff}(b, b')\\
&\text{where}\\
&\quad \mathtt{Bseq\_Core}(c) = \langle b \mid b \in \mathtt{Bseq}, b \in \mathtt{Asn}(c)\rangle\\
&\quad \mathtt{Bseq\_In}(b) = \langle b' \in \mathtt{Bseq} \mid (\exists c.\ \mathtt{ch\_deps}(b', b, c))\rangle\\
&\quad \mathtt{Bseq\_Out}(b) = \langle b' \in \mathtt{Bseq} \mid (\exists c.\ \mathtt{ch\_deps}(b, b', c))\rangle\\
&\quad \mathtt{name}(b, B) = \{m \mid (b, m) \in B\}\\
&\quad \mathtt{Bscs} = \mathtt{Blks} - \{b \mid \exists m.\ (b, m) \in \mathtt{MRs} \cup \mathtt{MWs}\}
\end{aligned}
$$

図 3 メモリ付並列制御モデルを構成する逐次プロセスの振舞い

図 2 の未定義のプロセス Blk_Mng(c), Mem, Buff(b, b') は逐次的なプロセスであり，その振舞いを図 3 に示す．以下，各プロセスの振舞いについて説明する．

4.5.1 ブロック実行管理プロセス

コア c の実行管理プロセス Blk_Mng(c) は，コア c に割り当てられた処理ブロック $b \in \mathtt{Asn}(c)$ をブロック実行順序 Bseq にしたがって，順番にプロセス Blk_Exe(b) を実行する．ここで，Blk_Exe(b) は処理ブロック b を実行するためのプロセスであり，次の 3 つのプロセスを順番に実行する．

- Blk_In(b)：処理ブロック b の前に実行すべき処理ブロック b' が実行済みであることを確認するプロセス．ブロックの集合 Bseq_In(b) は，b が依存する他のコアに割り当てられたブロック b' の集合であり，そのようなすべての b' の実行完了イベント bch.$b'.b$ をバッファから受信すると，Blk_In(b) は成功終了する．
- Blk_Core(b)：処理ブロック b を実行するプロセス．処理ブロック b がメモリ m への書込み用メモリブロックの場合は，チャネル wch.m に書込み用ブロック自身の名前 b を送信して共有メモリプロセス mem のメモリ m に b を書き込む．同様に，b がメモリ m からの読出し用メモリブロックの場合は，チャネル rch.m からそのデータを書き込んだブロックの名前 b' を受信して，検査用の読出し完了イベント rd.$b.b'$ を実行する．検証時に，この b' を確認することによって，共有メモリを経由した依存関係の正しさを判定できる．b が基本ブロックの場合は，ブロック b の実行完了イベント blk.b を実行する．
- Blk_Out(b)：処理ブロック b の後に実行すべき処理ブ

Vol.2019-ARC-240 No.40
Vol.2019-SLDM-191 No.40
Vol.2019-EMB-53 No.40
2019/2/28

ロック b' に実行済みであることを伝えるプロセス．$\texttt{Blk\_In}(b)$ に似ているが，方向が逆であり，ブロックの集合 $\texttt{Bseq\_Out}(b)$ は，b に依存する他のコアに割り当てられたブロック b' の集合であり，そのようなすべての b' に対して実行完了イベント $\texttt{ch}.b.b'$ を b' のバッファに送信して，$\texttt{Blk\_Out}(b)$ は成功終了する．

4.5.2 共有メモリプロセス

共有メモリプロセス $\texttt{Mem}(mem)$ は，すべてのコアから読み書き可能なメモリを管理しており，そのデータは局所的な列変数 mem に保存されている．この列変数 mem は，メモリの名前 m とデータ x（m に書き込んだメモリブロックの名前）の組 (m, x) の列であり，その初期状態は，図 2 の $\texttt{Mem}(\langle\rangle)$ に示すように空列である．チャネル wch からメモリ名 m とデータ x を受信すると $mem[m := x]$ によって列変数 mem に (m, x) を追加する（すでに (m, x') が存在する場合は上書きする）．また，メモリ名 m に保存されているデータ $mem[m]$ を常にチャネル rch に送信可能である．

4.5.3 バッファプロセス

バッファプロセス $\texttt{Buff}(b, b')$ は，ブロック b の実行完了を b' に伝えるイベントを 1 個保存するためのバッファであり，ブロック b からのイベント $\texttt{ch}.b.b'$ を，ブロック b' にイベント $\texttt{bch}.b.b'$ として転送する．

5. 並列制御モデルの検証

4 節では，共有メモリ付階層型制御モデル Ctrl_Model から共有メモリ付並列制御モデル Para_Ctrl を生成するアルゴリズムを形式的に定義した．本節では，その並列制御モデル Para_Ctrl によって実行される処理ブロックの順番が，元の制御モデル Ctrl_Model が要求する処理ブロックの依存関係に反しないことの検査方法について説明する．以下，5.1 小節にて，検証に使用する仕様を定義し，5.2 小節にて，その仕様を用いた検証方法を説明する．

5.1 仕様記述

並列化前の逐次的な制御モデル Ctrl_Model の振舞いを表す形式仕様記述 Spec_Ctrl を図 4 に示す．仕様 Spec_Ctrl は $\texttt{Spec\_Chk}(\emptyset)$ であり，$\texttt{Spec\_Chk}(B)$ は実行済みの処理ブロックの集合 B を更新しながら，次に実行可能なイベントを指定する仕様（CSP プロセス）である（初期状態での集合 B は空集合である）．図 4 中の各集合（$\texttt{Blks}_0$，$\texttt{PIOs}_0$，$\texttt{BRs}_0$，$\texttt{MWs}_0$，$\texttt{MRs}_0$，$\texttt{AtmIOs}_0$）は並列化アルゴリズムで処理する前の集合（4.2.1 小々節で定義済），$\texttt{UDeps}_0$ は全ての依存関係（4.2.2 小々節で定義済），$\texttt{Bseq}_0$ は入力時のブロックの実行列である．以下，図 4 の関数 $\texttt{enable}(B)$ と $\texttt{req}(b)$ について説明する．

- $\texttt{enable}(B)$：実行済みの処理ブロックの集合が B のとき，次に実行可能な処理ブロック（入出力ポートを含む）の集合を返す関数．処理ブロック b が実行可能であるためには，そのブロックが依存しているブロック $b' \in \texttt{src}(b)$ が実行済である必要がある．ここで，コア内では並列化前の実行順序 $\texttt{Bseq}_0$ に従うように，全依存関係 $\texttt{UDeps}_0$ に Bseq_Deps を加えている．さらに，b' が Atomic 型サブシステムの入力ポートである場合は，そのサブシステムの他の入力ポートも入力済でなければならないため（出力ポートも同様），$\texttt{atm}(\texttt{src}(b))$ も実行済みであることを要求している．
- $\texttt{req}(b)$：並列化前の逐次的な実行で，読出し用メモリブロック b が読み出すデータを書き込んだメモリブロック b' の集合を返す関数．これは，並列化後も書込み用メモリブロック b' と読出し用メモリブロック b の共有メモリを経由した間接的な依存関係が保存されることを要求する．処理ブロック b が読出し用メモリブロックでないときは空集合を返す．

$$
\begin{aligned}
&\texttt{Spec\_Ctrl} = \texttt{Spec\_Chk}(\emptyset)\\
&\texttt{Spec\_Chk}(B)\\
&\ = \texttt{if}\ (B = \texttt{PBlks}_0)\ \texttt{then SKIP}\\
&\quad \texttt{else}\ (\Box\ b : \texttt{enable}(B)\ @\\
&\qquad \texttt{if}\ (b \in \texttt{Bscs}_0 - \texttt{PIOs}_0)\ \texttt{then blk}.b \to \texttt{Spec\_Chk}(B \cup \{b\})\\
&\qquad \texttt{else if}\ (b \in \texttt{BRs}_0)\\
&\qquad\quad \texttt{then}\ (\Box\ b' : \texttt{req}(b)\ @\ \texttt{rd}.b.b' \to \texttt{Spec\_Chk}(B \cup \{b\})\\
&\qquad\quad \texttt{else Spec\_Chk}(B \cup \{b\})\\
&\text{where}\\
&\ \texttt{PBlks}_0 = \texttt{Blks}_0 \cup \texttt{PIOs}_0\\
&\ \texttt{enable}(B) = \{b \mid b \in \texttt{PBlks}_0 - B, \texttt{src}(b) \cup \texttt{atm}(\texttt{src}(b)) \subseteq B\}\\
&\ \ \texttt{src}(b) = \{b' \mid (b', b) \in \texttt{UDeps}_0 \cup \texttt{Bseq\_Deps}_0\}\\
&\ \ \ \texttt{Bseq\_Deps}_0 = \{(b', b) \mid \exists c \in \texttt{Cnums}.\ \exists i.\\
&\qquad\qquad\qquad\qquad \texttt{Bseq\_Core}_0(c)[i] = b',\\
&\qquad\qquad\qquad\qquad \texttt{Bseq\_Core}_0(c)[i+1] = b\}\\
&\qquad \texttt{Bseq\_Core}_0(c) = \langle b \mid b \in \texttt{Bseq}_0, b \in \texttt{Asn}(c)\rangle\\
&\ \ \texttt{atm}(B) = \{b' \mid \exists B' \in \texttt{AtmIOs}_0.\ B \cap B' \neq \emptyset, b' \in B'\}\\
&\ \texttt{req}(b) = \{b' \mid \texttt{chk\_MRW}(b', b, \texttt{MWs}_0, \texttt{MRs}_0, \texttt{MWs}_0)\}
\end{aligned}
$$

図 4 逐次的な制御モデルから生成される仕様

5.2 検証方法

並列制御モデル Para_Ctrl がデッドロックフリーかつライブロックフリーであり，元の制御モデルと同じ順番（依存関係のない処理ブロックの実行順序逆転は除く）で全てのブロックを実行することを保証するためには，次の 2 つの失敗発散詳細化関係が成り立つことを示せば良い．

$$\texttt{DLfree} \sqsubseteq_{\texttt{FD}} \texttt{Spec\_Ctrl} \qquad (*1)$$

$$(\texttt{Spec\_Ctrl}; \texttt{STOP}) \sqsubseteq_{\texttt{FD}} \texttt{Para\_Ctrl} \qquad (*2)$$

ここで，DLfree は 3 節で説明したデッドロックフリーかつライブロックフリーである抽象的なプロセスであり，1 番目の失敗発散詳細化関係 $(*1)$ が真ならば，Spec_Ctrl もデッ

Vol.2019-ARC-240 No.40
Vol.2019-SLDM-191 No.40
Vol.2019-EMB-53 No.40
2019/2/28

ドロックフリーかつライブロックフリーになる．この失敗発散詳細化が偽になるのは，制御モデルがフィードバックループ（依存関係のループ）をもつ場合である．本論文では，フィードバックループをもたない制御モデルを並列化の対象にしているため，ループをもつ場合は 1 周期分を抽出して展開するなど，並列化の前に処理が必要になる．

上記の 2 番目の詳細化関係 (∗2) は，仕様 Spec_Ctrl が許可するブロックの（元の制御モデルの依存関係を満たす）実行順序で並列制御モデル Para_Ctrl が全ての処理ブロックを実行した場合に真になる．並列制御モデル Para_Ctrl のブロック実行管理プロセスは 1 周期分の処理を実行後に成功終了するが，バッファやメモリプロセスは停止しないため，全体としては 1 周期分で停止することになる（成功終了ではない）．そのため，仕様 Spec_Ctrl の後に STOP をつけて，Spec_Ctrl 実行後に停止させている．

6. 拡張並列化アルゴリズムの検証用実装

本節では，4 節の拡張並列化アルゴリズムをモデル検査器 FDR[6] の入力言語 CSP_M でコード化し，Simulink の制御モデルから並列制御モデルを生成して，その振舞いを網羅的に検査できることを示す．FDR は CSP の代表的なモデル検査器であり，3 節で説明した失敗発散詳細化関係を自動的に検査できるツールである（ツール名 FDR は Failures-Divergences Refinement の略である）．最初に，6.1 小節で拡張並列化アルゴリズムの CSP_M コードについて説明し，6.2 小節では Simulink の制御モデルから拡張並列化アルゴリズムへの入力形式を生成するために開発したツール XML2CSP について説明する．最後に，6.3 小節で，複数の制御モデルの例に対して拡張並列化アルゴリズムを適用し，モデル検査した実験結果について報告する．

6.1 拡張並列化アルゴリズムの CSP_M コード

モデル検査器 FDR のモデル記述言語（入力）は CSP_M（Machine-readable CSP）であり，CSP_M は CSP の構文を含む関数型言語である．CSP プロセスに加えて，集合や論理式も記述できるため 4 節のアルゴリズムを，ほぼそのまま CSP_M 言語でコード化できる．

4.2 小節で説明した平坦化アルゴリズム Algo_Flatten の CSP_M コードを図 5 に示す．和集合（union）等，FDR 特有の構文もあるが，Algo_Flatten を，ほぼそのまま CSP_M で記述できている．なお，条件付きの推移閉包 trans_cls などの汎用の関数は他のアルゴリズムからも参照できるように大域的に定義しており，図 5 では省略している．

4.5 小節で説明した処理ブロックの並列化アルゴリズム Algo_Para_Ctrl の CSP_M コード（一部）を図 6 に示す．並列制御モデル Para_Ctrl の CSP プロセスの構造（図 2）と振舞い（図 3）も，ほぼそのまま CSP_M で記述できている．その他のアルゴリズム，DLfree プロセス，仕様も同様に CSP_M で記述できる．

```
Algo_Flatten(Ctrl_Model) = let
 (Systems,Bseq0,Asn0,Cnum) = Ctrl_Model

 Blks0 = diff(union(union(Bscs0,BWs0),BRs0),PIOs0)
  Bscs0 = {b | sys <- Systems, b <- bscs(sys)}
  BWs0 = {b | sys <- Systems, b <- bws(sys)}
  BRs0 = {b | sys <- Systems, b <- brs(sys)}
 PIOs0 = {b | sys <- Systems, b <- pios(sys)}
 MWs0 = {bm | sys <- Systems, bm <- mws(sys)}
 MRs0 = {bm | sys <- Systems, bm <- mrs(sys)}

 AtmSystems = {sys | sys <- Systems,
                     type(sys) == Atomic}
 AtmIns0  = {ins(sys)  | sys <- AtmSystems}
 AtmOuts0 = {outs(sys) | sys <- AtmSystems}

 UDeps0 = {dep | sys <- Systems, dep <- deps(sys)}
 AtmDeps0
  = union(
   {(b1,b) | ins  <- AtmIns0,  b1 <- ins,
             b2 <- ins, (b3,b) <- UDeps0, b2 == b3},
   {(b,b1) | outs  <- AtmOuts0,b1 <- outs,
             b2 <- outs, (b,b3) <- UDeps0, b2 == b3})

 TCDeps0 = trns_cls(union(UDeps0,AtmDeps0),PIOs0)
 Deps0 = diff(TCDeps0,
              mkdeps(union(Blks0,PIOs0),PIOs0))

within (Blks0,MWs0,MRs0,Deps0,Bseq0,Asn0)
```

図 5 平坦化アルゴリズムの CSP_M コード

次に，並列制御モデルが正しく動作することを検証するために，5.2 節で説明した詳細化関係の真偽を判定するため図 7 の assert 文を CSP_M コード中に挿入する．FDR は assert 文の真偽を自動的に判定し，結果を表示する機能がある．図 8 に FDR によって失敗発散詳細化関係を検査した結果を表示の例を示す（6.3 小節の例 1 を検査した結果）．右側の 2 つの詳細化関係の下に表示されている Finished:Passed が，その詳細化関係の判定結果であり，成立つことを表している．左側には，その判定で探索した状態数や遷移数，その時間等の情報が表示されている．

6.2 拡張並列化アルゴリズムへの入力生成

6.1 小節の CSP_M コードのアルゴリズムへの入力は 4.1 小節で説明した入力形式である．モデルベース並列化システム MBP は，Simulink の制御モデルからブロック間の依存関係を抽出し，XML 形式のファイルを生成する機能をもつ．本研究では，この XML 形式のファイルを 4.1 小節で説明した入力形式に変換するツール XML2CSP

Vol.2019-ARC-240 No.40
Vol.2019-SLDM-191 No.40
Vol.2019-EMB-53 No.40
2019/2/28

```
Algo_Para_Ctrl(Flat_Model,Cnum) = let
 (Blks,MWs,MRs,Deps,Bseq,Asn) = Flat_Model

 Para_Ctrl = (Para_MB [|MCh|] Mem(<>)) \ MCh
  Para_MB = (Para_Mng [|BCh|] Para_Buffs) \ BCh
   Para_Mng = ||| c:Cnums @ Blk_Mng(c)
   Para_Buffs = ||| c:Cnums @ Buffs(c)
    Buffs(c) = ||| (b,b'):ICh(c) @ Buff(b,b')

 Blk_Mng(c) = ; b:Bseq_Core(c) @ Blk_Exe(b)
  Blk_Exe(b) = Blk_In(b); Blk_Core(b); Blk_Out(b)
   Blk_In(b) = ; b':Bseq_In(b) @ bch.b'.b -> SKIP
   Blk_Out(b) = ; b':Bseq_Out(b) @ ch.b.b'-> SKIP
   Blk_Core(b)
    = ([] m:name(b,MWs) @ wch.m!b -> SKIP)
    [](  [] m:name(b,MRs) @ rch.m?b'-> rd.b.b'-> SKIP)
    [](member(b,Bscs) & blk.b -> SKIP)
 Mem(mem) =  wch?m?x -> Mem(mem_w(mem,m,x))
          [] rch?m!mem_r(mem,m) -> Mem(mem)
 Buff(b,b') = ch.b.b' -> bch.b.b' -> Buff(b,b')

  -- （ICh や Bseq_Core 等、集合や列の定義は省略） --
within Para_Ctrl
```

図 6　並列化アルゴリズム（単層用）の CSP_M コード（一部）

```
assert DLfree [FD= Ctrl_Spec
assert (Ctrl_Spec;STOP) [FD= Para_Ctrl
```

図 7　検証項目（失敗発散詳細化関係）の CSP_M コード

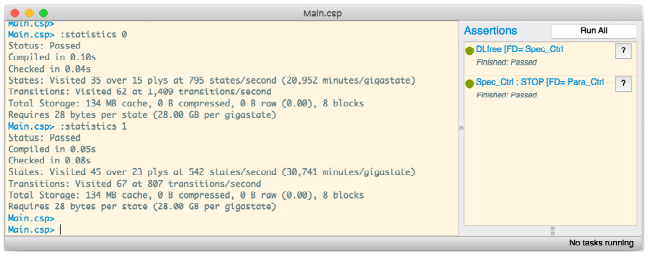

図 8　FDR による失敗発散詳細化関係の検査結果表示の例

を Python 言語で開発した．XML ファイルの読込みには `xml.etree.ElementTree`[7] を用いている．この変換ツール `XML2CSP` によって，Simulink 制御モデルから 4.1 小節の入力形式を自動的に生成し，その詳細化関係を FDR で自動的に判定することが可能である．

6.3　評価実験

6.1 小節の CSP_M コードと 6.2 小節の変換ツール `XML2CSP` を用いて，Simulink の制御モデルから生成した並列制御モデルが，元の制御モデルと同じように振舞うことを検証できる．ただし，モデル検査器は扱える状態数に上限があるため，検査可能な制御モデルのサイズに限界がある．本小節では，異なるサイズの制御モデルに対する検査時間の計測結果を報告する．なお，この実験では，計算機に MacBook Pro（OS：macOS Catalina 10.15.2, CPU：2.3 GHz Dual-Core Intel Core i5, メモリ：8 GB 2133 MHz LPDDR3），モデル変換 `XML2CSP` に Python3.7.1，モデル検査に FDR4.2.3 を使用した．

表 2 に，本実験に使用した制御モデルのブロック数，サブシステム数（層数），書込み/読出しブロック数と，XML ファイルから CSP_M の入力形式への `XML2CSP` による変換時間を示す．表 2 の例 1 の制御モデルのブロック線図を図 9，ブロック数に対する変換時間のグラフを図 10 に示す．モデル変換 `XML2CSP` は，XML ファイルから依存関係等の情報抽出と整形であり，図 10 に示すように，変換時間はブロック数に比例して増加している．

表 3 と表 4 に，各々，図 7 の 1 行目と 2 行目の詳細化関係の判定に使用された状態数，遷移数，前処理時間，検査時間を示す（詳細化関係の判定結果は全て真である）．ここで，前処理時間（コンパイル時間）は 6.1 小節の CSP_M コードから状態遷移系を生成するために要した時間，検査時間は詳細化関係の真偽を判定するために状態遷移系の探索に要した時間であり，状態数と遷移数は，その探索した状

表 2　実験に使用した並列制御モデルと変換時間

例	ブロック数	単層システム数	書込数	読出数	変換時間 [秒]
1	15	3	2	2	0.00080
2	41	10	6	6	0.00191
3	80	19	12	12	0.00422
4	119	28	18	18	0.00720
5	158	37	24	24	0.01169

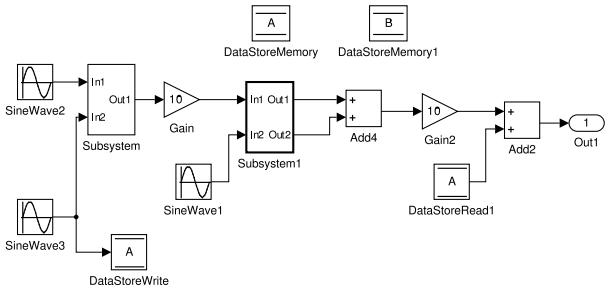

図 9　実験に使用した制御モデル（例 1）のブロック線図

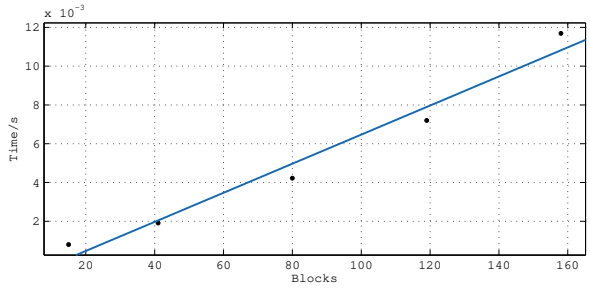

図 10　制御モデルのブロック数に対する変換時間

Vol.2019-ARC-240 No.40
Vol.2019-SLDM-191 No.40
Vol.2019-EMB-53 No.40
2019/2/28

表 3　詳細化関係 1 の判定実験結果

例	前処理時間 [秒]	検査時間 [秒]	状態数	遷移数
1	0.13	0.07	35	62
2	7.94	0.06	363	1486
3	70.92	0.27	306	911
4	441.35	1.38	311	767
5	1675.12	6.33	575	1661

表 4　詳細化関係 2 の判定実験結果

例	前処理時間 [秒]	検査時間 [秒]	状態数	遷移数
1	0.14	0.14	45	67
2	7.83	0.17	301	535
3	15.02	0.19	318	515
4	80.23	0.22	356	534
5	323.07	0.21	422	623

態と遷移の数である．表 3 の状態数と遷移数は Spec_Ctrl の状態数と遷移数と同じであり，表 4 の状態数と遷移数は Para_Ctrl の状態数と遷移数と同じである．

詳細化関係 2（Spec_Ctrl; STOP $\sqsubseteq_{\mathrm{FD}}$ Para_Ctrl）の判定に必要な時間については，表 4 の結果が示すように，検査時間よりも前処理時間に多くの時間が費やされている．この前処理時間には，4 節の拡張並列化アルゴリズムによって Simulink の制御モデルから並列制御モデル（CSP プロセス）を生成する時間と，その並列制御モデルから状態遷移系を生成する時間が含まれる．ブロック数に対するその前処理時間のグラフを図 11 に示す．この前処理時間の時間はブロック数と共に指数関数的に増加しており，並列化アルゴリズムの現在の CSP_M コードでは，数百個程度のブロックで状態爆発を起こし（計算機のリソースを使い果たし），判定不可能になる．

本研究では，拡張並列化アルゴリズムのミスを発見することを主な目的として，モデル検査器を使用している．そのため，4 節の拡張並列化アルゴリズムの数式を可能なかぎりそのままコード化するという方針で，CSP_M コードを作成した．今後，より大きな制御モデルを並列化するためには，効率よく変換するように拡張並列化アルゴリズムを実装する必要がある．その実装方法については今後の課題である．

詳細化関係 1（DLfree $\sqsubseteq_{\mathrm{FD}}$ Spec_Ctrl）の判定に必要な時間についても，前処理時間に多くの時間が費やされているが，この詳細化関係 1 は，次の 3 条件が成立つ任意の入力制御モデル Ctrl_Model について成立つと考えている．

- 制御モデル Ctrl_Model はループをもたない．
- 逐次的なブロック実行列 Bseq_0 は全依存関係を満たす．
- 共有メモリでは書込みのないデータの読出しはない．

すなわち，仕様 Spec_Ctrl のデッドロックフリー性とライブロックフリー性は入力の Ctrl_Model ごとに判定する必要はなくなる．その証明については今後の課題であるが，仕様 Spec_Ctrl の振る舞いは逐次的であり，並列制御モデル Para_Ctrl のデッドロックフリー性とライブロックフリー性を直接証明するよりも比較的簡単である．

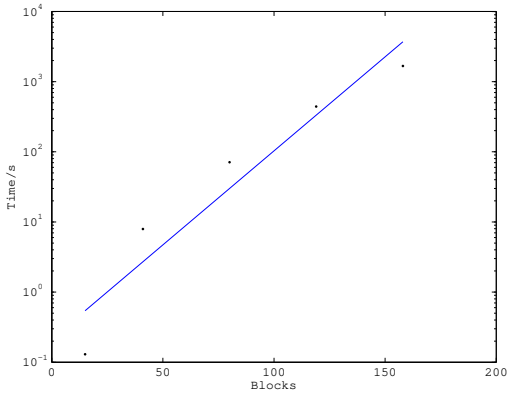

図 11　ブロック数に対する前処理時間（秒）

7. おわりに

我々は現在のモデルベース並列化システム MBP の単層制御モデル用の並列化アルゴリズムを，共有メモリと階層構造をもつ制御モデルの並列化にも対応できるように拡張し，その記述を形式的に厳密に与えた．その拡張並列化アルゴリズムを，CSP のモデル検査器 FDR の入力言語 CSP_M でコード化し，様々なサイズの Simulink の制御モデルの例に適用して，その並列化アルゴリズムで生成した並列制御モデルが正しく動作することを網羅的に検査できることを示した．モデル検査を使う利点は，発生確率の低い（再現性の低い）並列動作の不具合も確実に検出できることにある．

実験結果は，検査時間よりも前処理時間（並列化と状態遷移系生成）に多くの時間を費やすことを示しており，拡張並列化アルゴリズムの効率的な実装は今後の課題である．

参考文献

[1] MathWorks Makers of MATLAB and Simulink. (http://www.mathworks.co.jp)
[2] 鍾兆前, 枝廣正人: 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, ETNET2017, 電子情報通信学会技術研究報告, Vol.2017-EMB-44, No.47, pp.273-278, 2017.
[3] 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, 情報処理学会 組込みシステムシンポジウム（ESS）, pp.123–124, 2015.
[4] 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, ETNET2017, 電子情報通信学会技術研究報告, Vol.2017-EMB-44, No.6, pp.33-38, 2017.
[5] 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,ETNET2019, 電子情報通信学会技術研究報告, Vol.2019-EMB-50,No.9, pp.1-8,2019.
[6] FDR4.(http://www.cs.ox.ac.uk/projects/fdr/)
[7] xml.etree.ElementTree.(http://docs.python.org/3/library/xml.etree.elementtree.html)
[8] C.A.R.Hoare: Communicating Sequential Processes, Prentice Hall (1985).
[9] 磯部祥尚：並行システムの検証と実装 – 形式手法 CSP に基づく高信頼並行システム開発入門，近代科学社, 2012.