Vol.2017-ARC-225 No.45
Vol.2017-SLDM-179 No.45
Vol.2017-EMB-44 No.45
2017/3/10

モデルベース開発におけるデータ並列化に関する検討

竹松 慎弥[1,a)]　枝廣 正人[1]

概要：近年，大規模・複雑化が進む組込みシステムの開発に対して，モデルベース開発が普及している．また，性能向上やリアルタイム性向上のために並列処理の適用が検討されることが多く，特に制御分野においてモデルベース並列化の実現が求められている．一般的に制御分野のシステムにおいてはデータ並列性が低く，タスク並列やパイプライン並列のほうが有効であることが多い．しかし，例えば自動運転システムなどでは，データ並列性が高い画像処理などをシステムに組み込むこともあり，データ並列化も必要となってきている．そこで，本研究ではモデルベース開発におけるデータ並列化の実現を目指す．制御システムのモデルベース開発でよく用いられる MATLAB/Simulink において，データ並列性の高いモデルは ForIterator ブロックや Unbuffer ブロックを用いて作られる．このようなモデルからいくつかのマルチ・メニーコアアーキテクチャに向けた自動コード生成手法を検討した．

A study on Data Parallelization in Model Based Parallelizer

Shinya Takematsu[1,a)]　Masato Edahiro[1]

1. はじめに

近年，大規模・複雑化が進む組込みシステムの開発に対して，モデルベース開発が普及している．モデルベース開発はシステムをモデルと呼ばれる抽象的な表現を用いて書き表し，仕様検討や動作検証およびコード自動生成を行うことで開発コストや開発時間を削減する開発手法である．また，性能向上やリアルタイム性向上のためにマルチ・メニーコアを用いた並列処理の適用が検討されることが多くなっている．こういった背景から並列動作を考慮しながらモデルベース開発を行うモデルベース並列化の実現が求められている．

特に制御システムにおいては MATLAB/Simulink[1]（以下，Simulink）によるモデルベース開発が盛んに行われており，Simulink モデルを対象としたモデルベース並列化手法が提案されている [2]．一般的に制御システムにおいてはデータ並列性が低く，タスク並列やパイプライン並列のほうが有効であることが多いとされており，タスク並列やパイプライン並列の実現手法が提案されている．一方で，自動運転がトレンドとなっている昨今，データ並列性が高い画像処理などをシステムに組み込むこともあり，データ並列化も重要となってきている．

そこで，本研究では Simulink モデルを対象としたモデルベース開発におけるデータ並列化の実現を目指す．

2. MATLAB/Simulink モデル

制御システムにおいては数理モデル化にブロック線図が用いられてきた歴史がある．そのため，制御システム開発ではブロック線図でモデルを表現できる MathWorks 社が提供する MATLAB/Simulink と呼ばれるモデルベース開発ツールが用いられることが多い．このツールにはモデルのシミュレーション機能や自動コード生成機能が搭載されており，バグの早期発見や実装コストの大幅な削減を実現し，システム開発を効率的に行うことができる．

Simulink モデルでは主に以下の 2 つの要素を用いてシステムを表現する．

- ブロック
  あるまとまったひとつの処理を表す．必要とする入出力データに応じた入出力端子がある．
- 信号線

[1] 名古屋大学大学院情報科学研究科
[a)] ts1413@ertl.jp

Vol.2017-ARC-225 No.45
Vol.2017-SLDM-179 No.45
Vol.2017-EMB-44 No.45
2017/3/10

ブロック間のデータ入出力関係を表す．スカラーやベクトル，配列データといった様々な次元のデータを伝播できる他，幾つかの信号線をまとめたバス信号を形成できる．

ブロックには四則演算や定数出力，シミュレーション結果表示のためのスコープなど Simulink ライブラリとして幾つかの基本的なブロックが提供されている．これらのブロックを信号線でつなぎ合わせることによって様々なシステムを表現できる．また，幾つかのブロックをひとまとめにして 1 つのブロックと見なすこともできる．そうしてできたブロックは小さなシステムと捉えることができ，サブシステムと呼ばれる．大規模なモデルの場合，モジュール単位でサブシステム化することにより，視覚的に機能分割が分かりやすくなったり，再利用しやすくなるといった効果がある．また，サブシステムをさらに小さなサブシステムで構成し，システムの階層構造を形成することが可能である．

3. 従来の並列化手法

並列化を行う上で最も重要なことは負荷分散とコア間通信コストを削減することである．負荷分散とは処理を複数のコアに分割した際に処理量が均一になっているかどうかであり，並列実行時の最大性能に大きく影響を与える．一方，コア間通信は一般的にオーバヘッドが大きく，負荷がうまく分散できていたとしても，コア間通信が頻繁に起こると並列性能が上がらない場合やむしろ逐次実行時よりも遅くなってしまう場合がある．よって，負荷分散とコア間通信量のバランスが非常に重要である．

Simulink モデルを対象としたモデルベース並列化において，最適な負荷分散とコア間通信量で処理を分割する方法が提案されている [3]．この手法では，ブロック線図からブロックをノード，信号線をエッジとしたデータフローグラフを作成する．すると，ブロックのコア割当はデータフローグラフのグラフ分割として表現することができる．またこのとき，各ノードをブロックの処理量によって重み付けをしておくことで，ノード重みの均一度合いが負荷分散に相当し，分割時のカットエッジ数がコア間通信量に対応することになる．よって，ノードの重みができる限り均一になり，かつエッジのカット数が最小となるようなグラフ分割を導き出すことで，最適なコア割当てを導き出すことができる．

このグラフ分割を用いた方法であれば，データの依存関係のない異なる処理を行うブロック群を別々のコアに割当てることでタスク並列化を実現でき，データ依存関係のあるブロックも段階別に異なるコアに割当てることでパイプライン並列化を実現できる．一方，複数のデータに対して同じ処理を同時に実行するデータ並列では処理を細かく分割するのではなく，同じ処理を複製したい．これはグラフ分割では表現できず，データ並列化が本質的に難しいという現状にある．

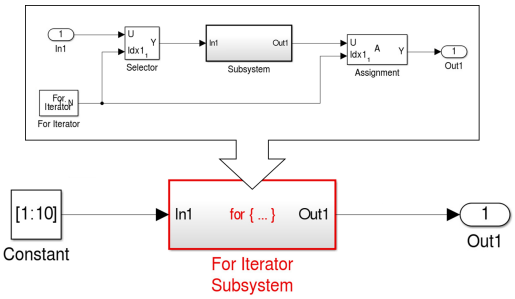

図 1 ForIterator モデル
Fig. 1 ForIterator model

4. 提案手法

データフローグラフからデータ並列性を抽出し，並列化を行うことは困難である．一方，データ並列性を生み出すことができるブロックはそもそも多くない．例えば，ForIterator ブロックおよび Unbuffer ブロックが挙げられる．そこで，本研究ではデータ並列性の高いモデルは ForIterator ブロックおよび Unbuffer ブロックを用いて記述されるものと仮定し，これらのブロックからデータ並列を利用した並列化を実現する方法を提案する．

4.1 ForIterator

ForIterator ブロックはサブシステム（特別に ForIteratorSubsystem と呼ばれる）に置かれ，サブシステム内の他のブロックの処理を指定した回数だけ繰り返し実行するというブロックである．つまり，コードにおける for ループを実現するブロックである．このブロックを用いることによって配列の各要素に対して繰り返し同じ処理をさせることができるなど，データ並列性の高いモデルを作ることができる．

本研究では ForIterator ブロックを用いて配列の各要素にアクセスして処理を行う図 1 のようなモデル (以降，ForIterator モデルと呼ぶ) を仮定してデータ並列化方法を考える．まず，ForIterator モデルの動作について解説する．Constant ブロックから配列データが ForIteratorSubsystem ブロックに送られる．ForIteratorSubsystem 内では Selector ブロックによって ForIterator ブロックが示す現在の反復回数に対応した 1 つの要素が配列から取り出される．取り出された 1 要素に対して Subsystem で定義した処理を行い，Assignment ブロックによって再び配列に格納される．これらの 1 要素に対する処理が，ForIterator ブロックによって配列データの要素数だけ繰り返し実行され，配列の全ての要素に対して Subsystem の処理が実行される．

ForIterator モデルに対してコード生成を行うと，繰り返

Vol.2017-ARC-225 No.45
Vol.2017-SLDM-179 No.45
Vol.2017-EMB-44 No.45
2017/3/10

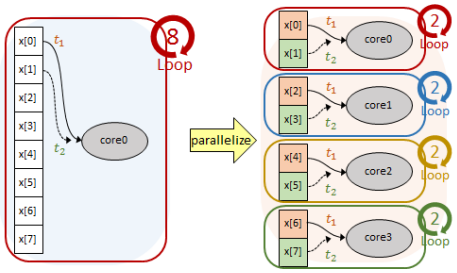

図 2　ForIterator ブロックのデータ並列化
Fig. 2　Data parallelization for ForIterator block

し実行される Subsystem の処理が for ループで包括されたコードとなる．この for ループの反復回数を分け，複数のコアに分配することで，データ並列が実現できる（図 2）．このような for ループの分割には OpenMP[4] を用いることが有効である．OpenMP は同じ処理を行うスレッドを複製し，各スレッドにイテレーションを分配する，という処理を自動で行い，並列化する．OpenMP による for ループの自動並列化は for 文の直前にディレクティブを挿入するだけで実現できるため，既存の自動並列化コード生成ツールを大きく変更することなくデータ並列の実現が可能である．また，指示文を追加することでチャンクサイズやスケジューリングアルゴリズムの変更が可能であり，モデルに合わせたループスケジューリングが容易に実現できる．

一方，OpenMP は共有メモリ型マシンに適した並列プログラミング基盤であり，分散メモリ型マシンなどに対しては効果的な並列性能が得られないことが危惧される．

4.2　Unbuffer

Unbuffer ブロックは配列データを入力として受け取り，その配列の要素を一つずつ順番に送り出すブロックである．そのため，Unbuffer ブロックでも配列の各要素に繰り返し同じ処理をさせるような高データ並列モデルを作ることができる．

本研究においては図 3 に示すような配列の全ての要素に同じ処理を適用するモデル (以降，Unbuffer モデルと呼ぶ) を仮定してデータ並列化方法を考える．Unbuffer モデルの動作を説明する．まず，Constant ブロックが配列データを出力する．Unbuffer ブロックがその配列データを受け取ると，要素毎に分解し，1 要素ずつ以降のブロックに送る．よって，Subsystem ブロックには次から次へと 1 要素のデータが送られ，次々に Subsystem の処理が適用されていく．最終的には Buffer ブロックが 1 つずつ処理済のデータを受け取り，分解された要素を再度ひとつの配列にまとめ，出力する．

Unbuffer ブロックをデータ並列化するには配列から複数の要素を取り出し，別々のコアに同時に送り出せばよい（図 4）．これを実現するには以下の 3 つを実現する必要がある．

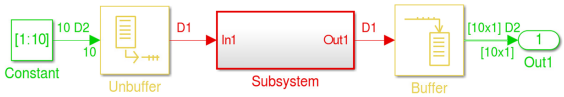

図 3　Unbuffer モデル
Fig. 3　Unbuffer model

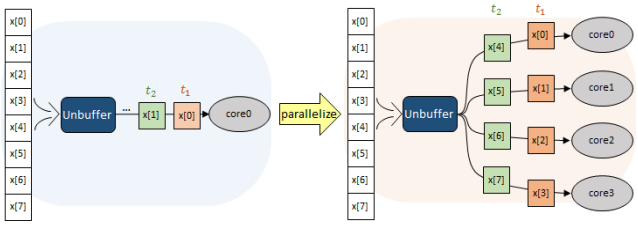

図 4　Unbuffer ブロックのデータ並列化
Fig. 4　Data parallelization for Unbuffer block

(1) Subsystem の処理を実現できるコアを複製すること
(2) 複製したコアに適切にデータを分配すること
(3) 処理が完了したデータを Buffer が順番どおりに受け取ること

まず，同じ処理を実行するコアの複製方法について述べる．既存のモデルベース並列化 [3] が生成する並列コードでは，あるコアが担当する全てのブロックの処理が一つの関数にまとめられる．そして，スレッド生成時にスレッドが実行する処理としてその関数を指定することで，そのコアに割当てられたブロックのみを処理するスレッドが生成される．よって，同じ処理を行うスレッドを複製するには同じ関数を指定したスレッド生成を複数回行えばよい．(1) を実現するためには，Subsystem の処理のみがまとめられた関数がなければならない．これは Subsystem ブロックの入出力信号線に対応するエッジでグラフ分割することによって実現できる．

次に，データの分配方法について述べる．Unbuffer-Subsystem 間でグラフ分割したため，データの送信はコア間通信を用いて行われる．このとき，Subsystem 処理を実現するスレッドへのデータ送信はスレッド ID を用いることによって任意に切り替えることができる．また，各スレッドに処理させるデータ集合はモデルによって静的に決定したほうが良い場合と動的に決定したほうが良い場合がある．例えば，データによらず処理量が一定ならばデータとコアの対応を静的に決めてしまっても問題は無く，むしろスケジューリングオーバヘッドがない分高速になる．一方，データによって処理量が大きく異なる場合はその都度処理が完了したコアに次のデータが割当てられたほうが無駄がなく，性能が良くなる．ForIterator の並列化に用いる OpenMP では主に static,dynamic,guided の 3 種類のスケジューリングアルゴリズムが用意されている．また，一度に割当てる反復回数（データサイズ）を示すチャンクサイズも変更できるようになっている．Unbuffer モデルにお

いても同様の機能が提供されるべきであり，これらのスケジューリングアルゴリズムの実装を考える必要がある．

まず，static は静的にデータ割当を決定するスケジューリング方式である．送ろうとしているデータを把握しておけば，そのデータを処理すべきスレッドが静的に決まるので，そのスレッド ID を指定してデータを送信すればよい．次に dynamic は動的にデータ割当先を決定するスケジューリング方式である．この実装にはデータ受信可否を示すフラグをスレッド毎に用意すればよい．データ受信側はフラグを立てることで受信待ちを知らせ，送信側はフラグをポーリングして，処理が完了しているスレッドを探し，フラグが立っているスレッドがあればそのスレッドにデータを送信すればよい．そして，guided は動的なスケジューリング方式であるが，始めは大きいチャンクサイズで割当てていき，徐々にチャンクサイズを小さくする方法である．dynamic と同様にフラグを設け，チャンクサイズをまとめて送り出す仕組みを作れば，チャンクサイズを動的に変更していくことで実現できる．チャンクサイズ分まとめてデータ送信を行うにはチャンクサイズ分の送信データを保持できるバッファを用意する．そして，Unbuffer ブロックの処理をチャンクサイズ回実行してバッファに格納する．後は，バッファのデータをまとめて送ればよい．以上のように，スケジューリングの実装と任意のスレッドへの送信を実現し，(2) を実現する．

最後に処理したデータを Buffer が順番どおりに受け取ることを考える．static スケジューリングを用いていれば，次のデータを処理しているスレッドが分かるため，そのスレッドからのデータの送信を待てばよい．一方，dynamic や guided はデータを処理しているスレッドが分からない．そのため，単純にデータを受け取ると順番がバラバラになってしまう．例えば，画像処理システムの場合，正しく画像処理されていても配列の順番がバラバラになっていると出力される画像が全く別のものになってしまう．よって，正しい順番でデータを受け取る必要がある．これを実現する方法としてはデータと共に時間やデータの番号などの情報を送る方法が考えられる．これにより，Buffer 側は共に送られてきた番号情報を基に適切な配列の位置にデータを格納でき，送られてくる順番に関係なく適切な順序の配列を生成できる．ただし，Buffer ブロックより Assignment ブロックに近い動きになってしまうため，元のモデルとは異なった動きになる．

以上のように処理の複製およびデータの分配を行うことによってデータ並列化を実現する．この手法はコア間通信を多用する手法であるため，並列化にコア間通信を必要とする分散メモリ型マシンに向いていると言える．一方で，コア間通信は並列性能に大きな影響を与えることがあり，コア間通信が遅いアーキテクチャへの適用には向かないことが予想される．

4.3 相互変換

ForIterator モデルと Unbuffer モデルはどちらも配列の各要素に対して Subsystem の処理を適用させるという機能を提供するモデルである．そのため，この 2 つのモデルは相互変換することができる．変換方法は，変換元のモデルの Subsystem ブロックを取り出し，もう一方の Subsystem ブロックを置き換えればよい．ForIterator モデルが共有メモリ型アーキテクチャに有効で，Unbuffer モデルが分散メモリ型アーキテクチャに有効であった．モデルの相互変換が容易であるため，ターゲットアーキテクチャの構成に応じて実装を切り替えることにより，ターゲットに適したデータ並列化を行うことができる．

5. まとめ・今後の展望

ForIterator ブロックおよび Unbuffer ブロックに対するデータ並列実現方法を提案した．ForIterator ブロックのデータ並列化は OpenMP を用いて for ループを分割することで実現できる．また，Unbuffer ブロックに対するデータ並列化はデータ処理部を複製し，コア間通信によってデータを分配することで実現できる．今後この方法の並列性能について評価し，実現可能性を検討する．

また，画像データに対する空間フィルタリング処理などがデータ並列化としてニーズが高い．一方で，このような処理は 1 つのピクセルデータだけでなく周辺のピクセルを参照する必要があり，データ依存関係を考える必要がある．本提案手法では現段階でデータ依存関係は考慮しておらず，周辺ピクセルの参照を行うような処理には対応していないため，そのようなデータ依存関係を保ちながら並列化を実現する方法を考える必要がある．

さらに，画像処理では GPU が用いられることが多い．そのうえ，多次元かつ大規模なデータに対してはスーパーコンピュータを用いて並列計算する例もある．よって，GPU やスーパーコンピュータなどをターゲットとしたデータ並列コード生成も要求される．既存のモデルベース並列化ではこれらのターゲット向けの並列コードを生成することができないため，本研究の実用性を高めるためには，これらのターゲットに適した並列コード生成に対応する必要がある．

参考文献

[1] MathWorks,Simulink, https://jp.mathworks.com/products/simulink.html, 2017.
[2] 梅田弾他: モデルベース開発向け画像処理ソフトウェアの並列化フレームワーク, 2015-EMB-38, No.4, 研究報告組込みシステム（EMB）, 2015.
[3] 山口滉平，竹松慎弥他: Simulink モデルからのブロックレベル並列化，ESS2015, 21, 2015.
[4] OpenMP，https://www.openmp.org/，2017.