Vol.2014-SLDM-165 No.25
Vol.2014-EMB-32 No.25
2014/3/16

単方向1:1高速同期機構を用いたFPGA実装と評価

溝口 裕哉[1]　中村 陸[1,2]　安藤 友樹[1]　荒川 文男[1]　枝廣 正人[1]

概要：近年，組込み制御分野でマルチ・メニーコアの使用が主流となりつつある．複数のコアで制御プログラムを並列に動作させる場合, コア間通信のオーバーヘッドが大きな問題である．そこで，我々は単方向1:1 高速同期機構とよぶ高速な通信メカニズムを提案し，その改善を図っている．本稿では，提案機構向けに新たに考案したハードウェア支援手法と FPGA による評価結果を報告する．モーター制御アプリケーションを用いた評価では，提案手法はマルチコア RTOS の通信 API と比べ，通信時間が約 12 分の 1 に短縮された．

キーワード：メニーコア，マルチプロセッサ，プロセッサ間通信，並列化，FPGA

1. はじめに

近年，制御分野においてモデル予測制御などの新しい高度な制御理論が提案されている．しかし，モデルをリアルタイムに計算する方式などにおいて，現在の制御プロセッサでは計算能力が不足するため，大幅な性能向上が望まれている．そして，制御システムに対する電力・温度・コストなどの様々な制約条件の下での性能向上には，マルチ・メニーコアの活用が有望である．複数のコアで制御プログラムを並列動作させるためには，ソフトウェアをマルチコア・メニーコアのために並列処理可能な形に分割する必要がある．分割の選択肢は多様であり，プログラム解析なしでは効果的な分割は困難である．プログラムを解析せずに分割すると，データの依存関係が崩れ正しい結果が得られない場合や，クリティカルパスが長くなり並列度が下がる可能性がある．そこでデータ依存性や機能性（モジュール性）による分割が求められる．さらに，分割したプログラムを実行するコア間において演算したデータや同期信号の送受信も必要である．こうした並列動作特有の処理は逐次動作では不要であるため並列化オーバーヘッドと呼ばれ，プログラム分割の粒度が細かいと非常に大きな問題となる．

我々は，単方向 1：1 高速同期機構を用いて通信オーバーヘッド問題の解決を図る研究を進めている．過去の研究において，この単方向 1：1 高速同期機構により通信オーバーヘッドの削減が可能であることを示した [1]．しかし，この研究はシミュレータによる検証のみで実機による検証はされていない．そこで，我々は FPGA を用いて実機検証を進めた．

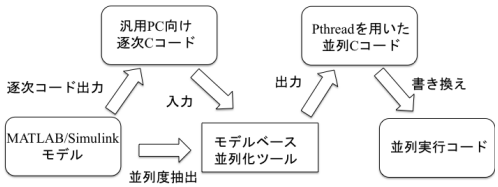


図 1　モデルから実行コード生成までのフロー

検証では，非対称型マルチコア（AMP）と対称型マルチコア（SMP）に対して単方向 1:1 高速同期機構の評価を行う．2 種のマルチプロセッサにおいて，通信を 1 つの集中共有メモリで行う方式と，複数の分散共有メモリで行う方式を評価した．また，従来の通信関数（OS が提供する排他制御機構を用いて通信）との比較評価も行った．

2. 準備

2.1 並列化プログラム

本研究では，現在モデルベース開発においてデファクトスタンダードになっている MATLAB/Simulink[2] を用いてシステムコンポーネントを生成し，コンポーネント間を結ぶブロック線図を作成することにより制御モデルが記述されていることを前提とする．本研究で使用するモデルから並列化コード出力までのフローを図 1 に示す．

まず，MATLAB/Simulink を用いて，与えられた制御モデルを汎用 PC 向け逐次コードに変換する．しかし，この段階では変換したプログラムは並列化されていない．これ

[1] 名古屋大学
Nagoya University
[2] 萩原電気株式会社
Hagiwara Electric Co.,Ltd

ⓒ 2014 Information Processing Society of Japan

Vol.2014-SLDM-165 No.25
Vol.2014-EMB-32 No.25
2014/3/16

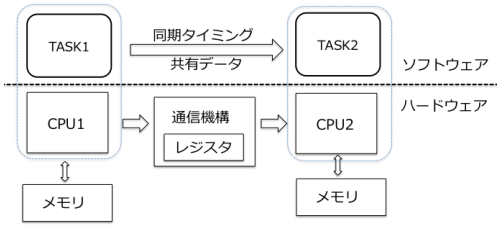


図 2　単方向 1:1 高速通信の概念図

に対して，モデルベース並列化ツール [4] を用いて逐次プログラムを並列実行可能なプログラム群に分割し，分割した各プログラムをタスクと呼ぶ．その際に使用されるデータの依存関係はモデル記述から抽出する．また，並列化ツールは，汎用 PC などで動作可能な POSIX API（pthread）を活用したプログラムを出力する．そのため，本研究では実験対象のシステムで駆動出来るように POSIX API などの環境依存部を手動で書き換えた．

MATLAB/Simulink を用いて生成された制御プログラムは，制御対象をコントロールするための出力値をセンサー情報等をもとに繰り返し計算する．その繰り返し計算される処理中にフィードバックが存在する可能性がある．フィードバックが分割された 2 つのプログラムの間に存在すると双方向のデータの通信が必要であるため，並列化ツールはフィードバックを切るようにプログラムを更に分割する．これにより並列化ツールを適用したプログラムのフローは単方向となる．

2.2　単方向 1:1 高速同期機構

単方向 1:1 高速通信のもとになる共有レジスタを使った近接プロセッサ間高速通信は，1990 年代後半から知られており，MUSCAT[5]，Multiscalar[6] など多くの研究がある．これらのアプローチは，メモリアクセスの衝突が問題であるならば，その衝突が起こらないように設計することである．複数のコアが通信のために単一の共有メモリに接続しようとするとアクセス競合は必然の問題となる．したがって通信を行う機構をコア間に置くということでアクセスの分散を図る．また，コア間に共有する資源が豊富にあると通信以外の目的によるメモリアクセスが衝突する可能性がある．その問題を排除するべく，「メモリを分散配置するメニーコア」が重要となる．

単方向 1:1 高速通信の概念を図 2 に示す．図 2 では CPU1 の TASK1 から CPU2 の TASK2 への通信が行われる．ソフトウェアレベルでは TASK1 の演算終了後，TASK2 に対して同期タイミング（演算可能のタイミング）の通知と共有データの転送を行う．このソフトウェアレベルの通信をハードウェアレベルでは専用の通信機構を用いて行う．単方向 1:1 が前提であるため，通信機構は関係のないコアからのアクセスは許していない．

[1] の研究では，図 2 の通信機構をメモリに置き換えている．そして，同期信号の通知は通信機構のメモリに配置したフラグのポーリング，共有データは同じメモリアドレスにアクセスするように実現する．タスク間通信は以下のように行う．

- 書き込みタスク:指定されたメモリアドレスにデータを書き込み，フラグを 1 とする．
- 読み込みタスク:フラグをチェックし，1 ならばデータを読み込む．フラグが 0 ならばポーリングをしながら待つ．

ポーリング中は常にタスクがメモリにアクセスしているため，同じコアに他のタスクが割り当てられている場合，ディスパッチするタイミングがない．そこで前提条件として，本手法を用いる場合は 1 つのコアにおいて 1 つのタスクのみ割り当てるものとする．この前提条件により，スケジューリングの影響を受けずに制御プログラムを実行できるようになる．

3.　FPGA 上での実現と評価

FPGA を用いて様々なメニーコアアーキテクチャを実装する．本稿では次の 2 種類のアーキテクチャを対象に実験する．そして，実機におけるメモリを用いた単方向 1:1 高速同期機構の評価を行う．搭載する OS は TOPPERS[7] により開発されたシングルプロセッサ向けの ASP（Advanced Standard Profile）カーネルとマルチプロセッサ向けの FMP（Flexible MultiProcessor Profile）カーネルである．

- UMA 型の SMP における評価
  現在のスタンダートなマルチコア構成であり，SMP 型 OS として FMP カーネルを用いる．
- NUMA 型の AMP における評価
  我々が前提としている将来のメニーコア構成であり，AMP 型 OS として複数の ASP カーネルを用いる．

3.1　SMP を用いた FPGA 実装

本項では FMP カーネルの搭載に必要な機能を実装した SMP を FPGA に構成し，制御プログラムにおける実行時間と通信時間の評価を行う．SMP に対応している FMP カーネルは 1 つの実行コードで複数の OS を管理する．本研究では，メニーコア上にタスクを固定的に割り付け，単方向 1:1 高速同期機構を用いるため，タスクの動的移動を行わない FMP カーネルは適している．

3.1.1　実験準備

評価実験で搭載するプログラムはモーター制御モデルから生成する．このモーター制御プログラムを並列化ツールにかけて変換したプログラムのタスクフローグラフを図 3 に示す．実験で使用するプログラムは 15 個のタスクから形成され，1 コア 1 タスク基準で割り当てられる．

ⓒ 2014 Information Processing Society of Japan

Vol.2014-SLDM-165 No.25
Vol.2014-EMB-32 No.25
2014/3/16

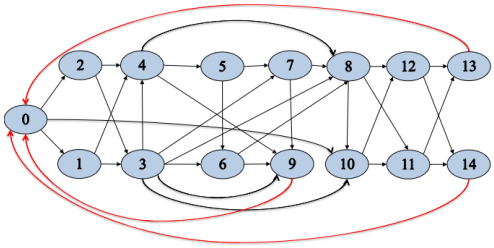


図 3　制御モデルのタスクフローグラフ

表 1　Nios2 コアの設定内容

コアオプション	Nios2/s
ISA	RISC
命令語長	32 bit
命令キャッシュ	4kB
データキャッシュ	なし
動作周波数	100MHz

表 2　SMP 型実験における FPGA の資源使用率

Logic utilization	57 %
Combinational ALUTs	70604/182400(39%)
Dedicated logic registers	32918/182400(18%)
Total registers	33316
total pins	156/888(18%)
Total block memory bits	9556160/14625792(65%)

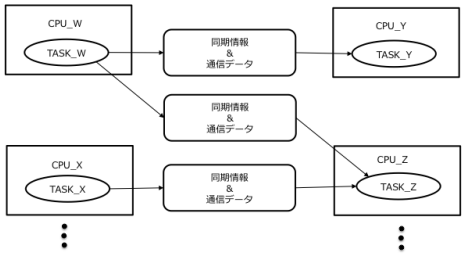


図 4　提案手法 A:通信メモリが分散配置されているケース

並列化ツールによって出力されたタスクには複数の出力を持つタスクがある．そのタスクは次に演算可能になる後続タスクに起動許可を伝えるため，タスクの全処理終了後に複数のタスクに対して同時に通信を行う．しかし，出力データの生成が途中で完了している例もあり，各出力データの生成完了後に同期信号を出力することで，並列度が上がると考えられる．この考えを元に演算関数内の終了通知関数の実行タイミングを手動で変更した．この変更したプログラムを「最適化後」と呼ぶ．

FPGA ボードは様々なメモリ機構を提供している．単方向 1:1 高速同期機構は，容量制限は厳しいものの高速な組込みメモリを駆使して実現する．前以って実装を行う制御プログラムのチャネル数と，それぞれのチャネルがどのコアに接続しているか解析しておき，FPGA の設計ツールによって，組込みメモリで通信メモリを通信チャネルの数だけ生成する．そして，該当チャネルで通信を行うコアが生成した通信メモリにアクセス出来るようにワイヤーでつなげる．同期フラグや通信される変数はリンカスクリプトによって対応する通信メモリに割り当てる．また，比較のために通信メモリを分散配置せずに共有メモリに集約させる構成も実装する．

作成したハードウェアの構成について述べる．CPU 構成として 15 個の Nios2 コアを搭載し，更にそれぞれのコアに対して FMP カーネルに必要なハードウェアを追加した．Nios2 コアの詳細情報を表 1 に示す．

メモリ構成については全てのコアがアクセス可能な領域として DDR2 SDRAM（4GB）と組込みメモリ（750KB）を，通信を行うコアの間に 1KB の組込みメモリを多数配置した．使用した FPGA ボードの DE4 における上記構成の資源使用率を表 2 に示す．

SMP 型であるため，データを置くセクションは全て共有メモリである．変数領域である rwdata セクション，bss セクション，rodata セクションは高速アクセス可能な組込みメモリに配置する．実行されるコード情報がある text セクションはプログラムのサイズによって変動する．したがって，1.5MB しか利用できない組込みメモリでは配置できない可能性がある．このため，テキストセクションは利用可能なメモリ容量の大きな DDR2 SDRAM の領域に配置する．

本実験の重要な構成要素である通信セクションのメモリ配置は，大きく分けて次の 2 パターンを用意した．

(1) 分散配置
通信セクションを各通信チャネルごとに分散配置で作成した共有メモリに割り当てる．各メモリに対してアクセス可能なコアを制限しているため，メモリの入口の競合が起こる可能性は非常に低い．このメモリ配置による方式を本実験では「**提案手法 A**」とする．提案手法 A による割り当て方法を図 4 に示す．専用メモリをそれぞれの通信用に配置し，通信を行うプロセッサだけがアクセスする．

(2) 集中配置
通信セクションを一つの集中配置された共有メモリに割り当てる．共有メモリは全てのコアがアクセス可能である．bss や rwdata などの他のセクションと分離して通信専用の共有メモリとする方式を「**提案手法 B**」，全て同じ共有メモリに配置する方式を「**提案手法 C**」とする．提案手法 B および提案手法 C の割り当て方法を図 5 に示す．全てのコアが一つの共通メモリを介して通信する．

従来手法（並列化ツールが自動生成する通信関数）では OS が管理している資源を用いて通信を行う．そこで，FMP カーネルのセマフォ機構を用いて通信関数を実装する．こ

ⓒ 2014 Information Processing Society of Japan

Vol.2014-SLDM-165 No.25
Vol.2014-EMB-32 No.25
2014/3/16

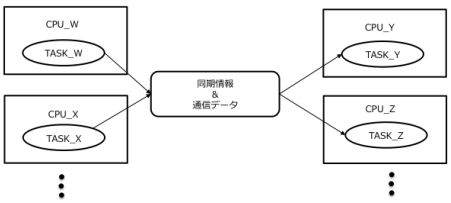


図 5　提案手法 B，C:通信メモリが集中配置されているケース

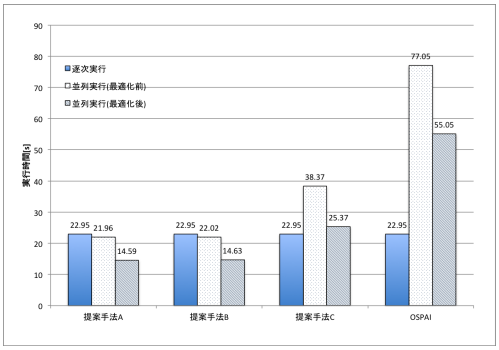


図 6　SMP 型における実行時間

の方法を用いて通信を行うケースを本稿では「**OSAPI**」と表記する．

以上の 4 パターンの通信手法と通信メモリマッピングで検証を行った．検証の様々な条件を表 3 にまとめる．なお，表 3 において，2 種類の共有メモリを共有メモリ A および，共有メモリ B としている．また，分散メモリは各通信チャネル専用に分散配置した共有のメモリを指す．

3.1.2　実験結果

制御モデル 5,000 ステップの実行時間を図 6 に示す．逐次実行では，通信メモリの条件変更が実行に全く影響しないため，全ての条件設定に対し全て同一の実行時間であった．図 6 よりメモリを分散配置する提案手法 A が最も短い実行時間であった．また，通信専用の共有メモリを用いた提案手法 B は提案手法 A とほぼ同じ実行時間であった．通信のポーリングを含む全ての共有アクセスが一つの共有メモリに集中する提案手法 C と OSAPI を使用する従来手法の場合，実行時間が提案手法 A と比べると，最適化前の並列実行時間がそれぞれ 1.75 倍，3.51 倍になった．

次に図 3 におけるタスク 2 からタスク 3 への通信時間を測定した結果を図 7 に示す．通信時間は送信タスクの送信関数開始から受信タスクの受信関数完了までとする．計測結果はタイミング最適化前の並列プログラムの場合である．図 7 より，提案手法 A が最も速いといえる．提案手法 A と比較して，提案手法 B は 1.21 倍，提案手法 C は 1.97 倍，そして，OSAPI は 14.27 倍の通信時間が必要であった．

3.1.3　考察

本実験では SMP 型 OS を搭載した UMA 型マルチコア

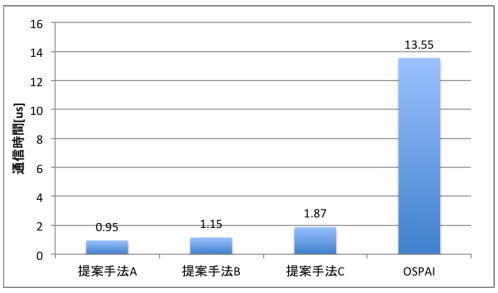


図 7　SMP 型における通信時間

プロセッサにおいて提案手法の有効性を検証した．その結果，同じアーキテクチャでも通信の際にアクセスするメモリの配置条件で，通信時間が大きく変動することがわかった．その中でも必要最低限のコアからアクセスのみが可能な分散メモリを用いるケースおよび通信専用の共有メモリを用いるケースが良い結果であった．今回利用した FPGA では組込みメモリで作成した共有メモリが，アクセス可能なコア数が増えても，コアのローカルメモリ並みに高速であるため両者の差はほとんどなかった．しかし，実チップではアクセスするコア数の多い共有メモリの遅延は大きくなるため，両者の差が拡大すると考えられる．したがって，今回の実験結果はメニーコアにおける分散配置の重要性を示している．

OSAPI については，実行時間，通信時間の両項目で最も時間が長いという実験結果であった．OSAPI ではセマフォを用いて実装を行った．OSAPI はセマフォにアクセスをする際にアクセスするメモリをロックする．問題はロックをされる共有メモリに全てのコアがアクセスすることである．そのため，クリティカスパスのタスクの共有メモリへの読み出し・書き込みのオーバーヘッドが増大し，実行時間と通信時間が遅くなったと考えられる．同様に，提案手法 C のケースは他のコアの通信ポーリングによるアクセスで共有メモリの応答速度が落ち，クリティカルパスの処理に遅れが生じたと考えられる．

3.2　AMP 型を用いた FPGA 実装

3.2.1　実験準備

3.1.1 項で述べた実験準備と基本的に同じであるため，異なる点のみ説明する．15 コアを同時に実行し AMP 型の実装であるため異なる設定を持つ 15 の ASP カーネルが必要となった．それぞれのペリフェラルも異なるため，別々に実行コードを生成する必要がある．

ハードウェア構成では，FMP カーネルに特有な機構を削除した．メモリアーキテクチャについては大幅な変更があり，共有メモリの構成はそのままにして，更にそれぞれのコアがローカルにアクセス出来るローカルメモリを組込みメモリで作成した．構成したアーキテクチャの FPGA の資源使用率を表 4 に示す．

ⓒ 2014 Information Processing Society of Japan

Vol.2014-SLDM-165 No.25
Vol.2014-EMB-32 No.25
2014/3/16

表 3　SMP 型実験における条件のまとめ

パターン名	逐次	提案手法 A	提案手法 B	提案手法 C	OSAPI
通信プログラム	-	メモリポーリング			セマフォ同期
通信が使用するメモリ	-	分散メモリ	共有メモリ B	共有メモリ A	共有メモリ A
データおよびカーネル領域	共有メモリ A				
コア数	1	15	15	15	15
OS	FMP				

表 4　AMP 型実験における FPGA の資源使用率

Logic utilization	31 %
Combinational ALUTs	39460/182400(22%)
Dedicated logic registers	29634/182400(16%)
Total registers	30032
total pins	156/888(18%)
Total block memory bits	7459008/14625792(51%)

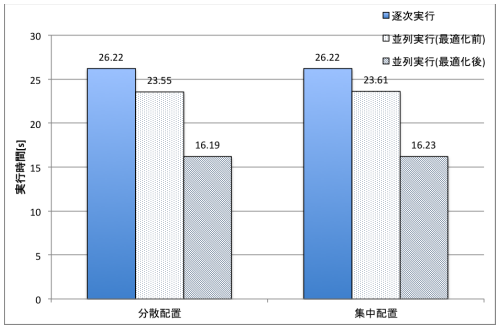


図 8　AMP 型における実行時間

ローカルメモリに変数データを置き，テキストセクションは DDR2 SDRAM に置く．テキストセクションは共有しており，アクセス衝突などが考えられる．しかし，Nios2 プロセッサは 4KB の命令キャッシュを有しているため，通信のためのポーリング命令がキャッシュに載ることが期待され，アクセス競合の起こる確率は減ると推測される．

SMP 型の実験と同様に，通信で使用するセクションは分散配置をしたメモリあるいは全てのコアがアクセス可能なメモリのどちらかに割り当てる．本実験の条件は 2 パターンであるため，SMP 型において提案手法 A と呼ばれていた配置方法は「分散メモリ」，提案手法 B は「集中メモリ」と表記する．

3.2.2　実験結果

モーター制御プログラム 5,000 ステップ実行時間を図 8 に示す．逐次の列は並列化ツールを使用する前の逐次プログラムを 1 つのプロセッサで動作させた場合の実行時間である．並列に関しては，それぞれの通信アーキテクチャに対しての結果が行に対応しており，通信を行うタイミングについて最適化したものを並列（最適化後）の列で示している．分散メモリ，共有メモリともほとんど差がない．

タイミングの最適化をかけていない並列プログラムのタスク 2 からタスク 3 へコア間の通信時間の測定結果を図 9

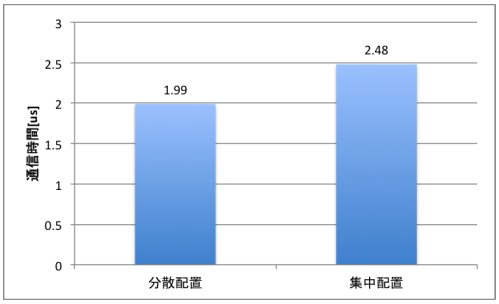


図 9　AMP 型における通信時間

に示す．分散メモリを用いる場合，一回の通信の平均時間が共有メモリに比べ 1.25 倍速くなる．AMP の実験においても，FPGA のメモリの特徴が分散メモリと共有メモリの差が小さい原因であると考えられる．

3.2.3　考察

本実験では通信セクションのメモリマッピングとして，分散メモリと共有メモリについて検証を行った．そして，UMA 型の SMP の場合と同様に分散メモリの方が通信が速いことを確認した．これは NUMA 型の AMP でも共有メモリにアクセス競合が起こるために生じたと考えられる．FPGA を用いたため競合する領域がメモリコンポーネントの入り口だけである．実機においては共有メモリの遅延が大きく，かつバス競合という要因も考えられるため，分散メモリ方式の通信メモリの分散配置による効果が更に大きくなると考えられる．

AMP では SMP の実験のように共有メモリへのアクセス競合による大きな遅延はなかった．したがって，AMP は NUMA を考慮したメモリ配置の設定は難しいが，安定した性能を求めるシステムを構成する際には適するプロセッサ構成であると考えられる．

4.　おわりに

本稿では様々なアーキテクチャにおいて通信方法の実装による影響を検証した．UMA 型の SMP を用いた実験では，単方向 1:1 高速同期機構がプロセッサ間通信のオーバーヘッドを削減できることを示した．また，NUMA 型の AMP を用いた実験でも同様にメモリを分散したほうが 1.25 倍高速に通信できることがわかった．これら 2 種のマルチプロセッサの検証結果より，我々が提案している分散メモリを用いた単方向 1:1 高速同期機構はメニーコアの 1

ⓒ 2014 Information Processing Society of Japan

Vol.2014-SLDM-165 No.25
Vol.2014-EMB-32 No.25
2014/3/16

つの有用な通信手法であると結論付けられる．ただし今回使用した FPGA では組込みメモリアクセスが常に高速であり，分散メモリと共有メモリの差が大きく出なかった．分散メモリを用いた通信機構は，実際のメニーコアプロセッサでは分散メモリ方式の優位性がより大きくなることが期待できる．

また，SMP の実験より，セクションのメモリへの配置が原因でコア同士のメモリアクセス競合が起き易く，全体の実行時間が増大する場合があることがわかった．この結果は，メモリに対してアクセスするコア数を減らすことが重要であることを示している．それと比較して，AMP の場合，各コアが持つローカルメモリに変数データなどを置くためデータのメモリ割り当てにおける衝突を考慮する必要がない．この先，32 コア，64 コアとメニーコアの時代が到来したときに AMP の方が安定した動作が期待でき，コア数の変化によるスケーラビリティが高いと考えられる．したがって，評価結果より「メモリを分散配置するメニーコア」が重要になるという仮設は正しいと言える．

提案手法の次の課題は，メモリ関連の設定自動化と通信機構のハードウェア化である．

参考文献

[1] 中村 陸, 荒川 文男, 枝廣 正人：単方向 1:1 高速同期機構を用いた組込み制御並列化，研究報告システムソフトウェアとオペレーティング・システム（OS），Vol.2013-OS-127, No.11，pp.1-6，(2013-11-26)．
[2] MathWorks:MATLAB/Simulink，MathWorks（オンライン），入手先〈http://www.mathworks.co.jp〉（参照 2014-02-07)．
[3] C.A.R.Hoare：Communicating Sequential Processes. Prentice Hall（1985)．
[4] T. Kumura,Y.Nakamura, N.Ishiura, Y.Takeuchi,and M.Imai：Model Based Parallelization from the Simulink Models and Their Sequential C Code, The Work- shop on Synthesis And System Integration of Mixed In- formation Technologies（SASIMI 2012)，R2-8，pp.186-191 (2012)．
[5] 鳥居 淳，近藤 真己，本村 真人，池野 晃久，小長谷 明彦，西 直樹：オンチップ制御並列プロセッサ MUSCAT の提案，情報処理学会論文誌，Vol.39，No.11，pp.1622-1631 (1998-06-15)．
[6] Gurindar,S. Sohi，Scott , E. Breach and T. N. Vijaykumar：Multiscalar processors，Proceedings of the 22nd annual international symposium on Computer architecture（ISCA '95)，Vol.23，No.2，pp.414-425 (1995-05)．
[7] TOPPERS：TOPPERS プロジェクト／INDEX，TOPPERS（オンライン)，入手先〈http://www.toppers.jp〉(参照 2013-02-07)．

ⓒ 2014 Information Processing Society of Japan