Vol.2013-OS-127 No.11
Vol.2013-EMB-31 No.11
2013/12/3

単方向1:1高速同期機構を用いた組込み制御並列化

中村 陸[1,2]　荒川 文男[1]　枝廣 正人[1]

概要：組込み制御領域において，マルチ・メニーコア化とこれを活用するための制御ソフトウェアの並列化が進められている．しかし期待される処理性能のボトルネックとして，タスクスケジューリングやプロセッサ間通信のオーバヘッドが存在する．1 コア 1 タスクに静的割付けし，単方向 1:1 高速同期機構による通信を用いることで，モータ制御モデルを用いた性能評価において，メニーコア向け高機能 OS 利用と比較して通信時間を 25 分の 1 とし，実行サイクルを逐次実行よりも 25 %程度低減することができた．

キーワード：メニーコア，マルチコア，プロセッサ間通信，並列化

1. はじめに

近年，組込み制御領域において電力や発熱の問題からシングルプロセッサの性能向上が困難になってきており，マルチ・メニーコアプロセッサが主流となりつつある．しかし，マルチ・メニーコア化しただけではシステムの性能向上は期待できず，ソフトウェアを並列化し，マルチ・メニーコアに対応させる必要がある．

ソフトウェアを並列化すると複数のタスクが生成されるため，タスクスケジューリングが必要となる．また，異なるプロセッサで実行されるタスク間でのデータ共有や同期のためプロセッサ間通信を行う必要がある．これらは並列化によるオーバヘッドとなる．また、リアルタイム性の厳しい組込み制御におけるタスクスケジューリングは，優先度設計など設計開発コストへのインパクトも大きい．

自動車のエンジン制御やモータ制御などでよく用いられているフィードバック制御系のアルゴリズムは逐次性が強くソースコードレベルでの並列化は必ずしも容易ではない．このようなシステムの並列化手法として，MATLAB/Simulink[1] のモデルベース開発に着目した並列化手法が提案されている [2]．この手法では，MATLAB/Simulink モデルと逐次処理 C コードから，マルチコア向けに記述された並列処理 C コードに並列化する．これは CSP 理論 [3] に基づき記述されたモデルとなっている．

組込み制御をこの手法により並列化を行うと，タスクの粒度が小さく，また，図 1 のタスクグラフのように各タスクが少数タスクとのみ通信するという特徴がある．そのため，通信のオーバヘッドが大きい場合，並列化による性能向上を得難いという問題がある．

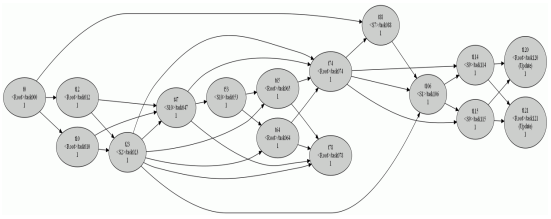

図 1　組込み制御のタスクグラフの例

そこで本報告では，同一コア内でのタスクスケジューリングを不要とするために，1 コア 1 タスクに静的割付けし，1:1 高速同期機構によるプロセッサ間通信を用いることで並列化のオーバヘッドを削減する手法を提案する．

2. タスク実行モデルとプロセッサ間通信高速化

1 つのコアに対し実行するタスクを 1 つだけ静的に割付け，タスクの生成・終了はシステムの起動・終了時のみ行う．各コアは起動時にタスクを割付けられ，周期的にタスクを実行する．これにより同一コア内でのタスクスケジューリングが不要となる．

そして，1 つのタスク間通信に対し，1 つの共有メモリ領域と 1 つのフラグを割り当てる．通信は単方向とし，1 つの共有メモリ領域に対し，書込むタスクは 1 つのみとする．共有メモリは隣接プロセッサ間共有レジスタやローカルメモリに配置することを想定する．タスク間通信は以下のように行う．

書込みタスク: 指定されたメモリアドレスにデータを書

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

込み，フラグを 1 とする．

読込みタスク: フラグをチェックし，1 ならばデータを読込み，フラグを 0 とする．0 ならばウェイト（ループ）する．

共有メモリに書き込むタスクを 1 つのみすることで，複雑な同期処理や調停処理が不要となり通信のオーバヘッドを低減する．

2.1 実装手法

本稿では，フィードバック制御を行うモデルを対象とし，MATLAB/Simulink[1] モデルを仮定する．MATLAB/Simulink で記述した制御モデルを Real-Time Workshop Embedded Coder[4] により，逐次 C コードに変換する．この逐次 C コードをモデルベース並列化ツール [2] を用いて並列化する．これにより，MATLAB/Simulink の 1 つのブロックを 1 つのタスクとする並列化が行われる．

提案手法では共有データをタスク間通信毎に異なるメモリ領域に配置する．しかし，MATLAB/Simulink では複数のタスク間通信で使用する共有データをまとめて，1 つの構造体として定義する場合があり，そのような場合には，共有データを個別の非構造体の変数に変更する．

モデルベース並列化ツールで並列化されたタスクは main_task と run_task の 2 つの関数から構成される．run_task には，実際の制御処理が記述される．main_task は，各タスクのエントリ関数で，最初にタスクの初期化を行う．その後以下の処理を繰り返す．

(1) 他タスクからメッセージを受信し，run_task で使用する共有データを同期する．
(2) run_task を呼び出す．
(3) run_task 内で更新された共有データを他のタスクへ送信する．

この構成では，run_task の終盤でしか使用しない共有データであっても run_task を実行する前に用意されている必要がある．そこで，共有データの同期タイミングを run_task 内の実際に共有データを使用する直前とすることで，待ち時間を低減する．

3. 評価実験

提案手法によるオーバヘッドの削減効果を確認するため，通信にメニーコア向け高機能 OS API(以下 OS API と記す) を用いるソフトと提案手法を用いるソフトを実装し，比較を行った．

実験はまず，通信のみを行う通信評価ソフトを実装し，通信サイクルの測定を行った．次に，MATLAB/Simulink で記述された実際のモータ制御モデルを実装し，タスクを個別に実行し通信サイクルの測定を行った．最後に，モータ制御モデルの全てのタスクを並列実行し，処理サイクルの測定を行った．

表 1 線形近似により算出した通信サイクルの変化量（レイテンシ:0）

パラメータ	変更単位	パラメータの変更単位あたりの通信サイクル変化量 [clock]	
		提案手法	OS API
送信データ長	4byte	2.591	2.489
送信回数	1 回	11.62	1089
受信データ長	4byte	4.738	4.768
受信回数	1 回	16.14	623.0

3.1 評価環境

ルネサスエレクトロニクスのマルチコアシミュレータを用いて実験を行った．マルチコアシミュレータは 4 個のハードウェアスレッドを実行可能な PE を 4 個搭載しており，合計 16 個のハードウェアスレッドを実行できる．ここでは，ハードウェアスレッドをコアとして扱う．また，マルチコアシミュレータでは，複数のメモリ領域を定義し，メモリ領域ごとにレイテンシを設定可能である．各コア間にメモリ領域とそのレイテンシを設定し，通信に用いた．

3.2 通信評価ソフトによる通信サイクル測定

2 つのコア間で通信を行う通信評価ソフトを実行し，通信サイクルを測定した．ここでは，通信サイクルは送信処理に必要なサイクル数（以下，送信サイクルと記す）と受信処理に必要なサイクル数（以下，受信サイクルと記す）の合計とする．

通信評価ソフトを実行する 2 つのコアは，異なる PE で実行されるハードウェアスレッドとした．また，通信評価ソフトを実行する以外のコアは全て停止（HALT）し，通信評価ソフトを実施するコアは通信以外の処理は行わない．

実験のパラメータとして，通信データを配置するメモリのレイテンシと通信データ長，通信回数を設定した．メモリのレイテンシは 0 サイクルと 11 サイクルの 2 種類の設定を用いた．通信データ長と通信回数はどちらか一方のみを変化させた．設定値は組込み制御で典型的な値として以下とした．通信データ長を固定とする場合，通信データ長は 32byte とし，通信回数を 1 回〜8 回の範囲で変化させた．通信回数を固定する場合，通信回数は 1 回とし，通信データ長を 8byte〜128byte の範囲で変化させた．

3.2.1 実験結果

実験結果を図 2〜図 8 に示す．実験結果を最小二乗法により線形近似し，その傾きから通信回数 1 回あたりの通信サイクルと通信データ長 4byte あたりの通信サイクルの変化量を求めた．メモリのレイテンシが 0 の場合の通信サイクルを表 1 に，メモリのレイテンシが 11 の場合の通信サイクルを表 2 に示す．

メモリのレイテンシが 0 の場合の通信のデータ長を l，回数を n とすると送信サイクル S と受信サイクル R は次の式により求まる．

$$S = (2.591) * l/4 + (11.62) * n + C_S \quad (1)$$

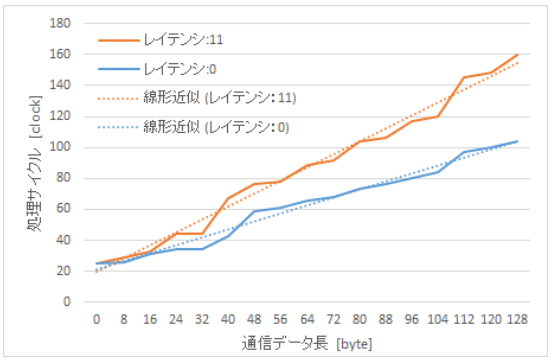

図 2 通信データ長と送信サイクル（提案手法）

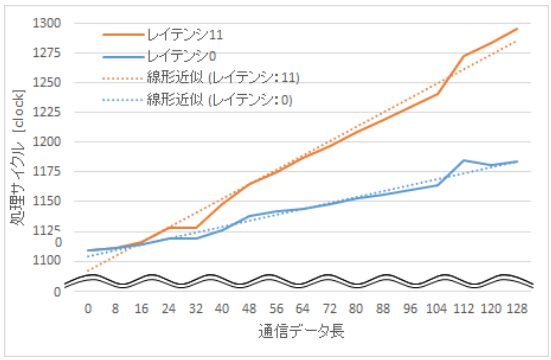

図 3 通信データ長と送信サイクル（OS API）

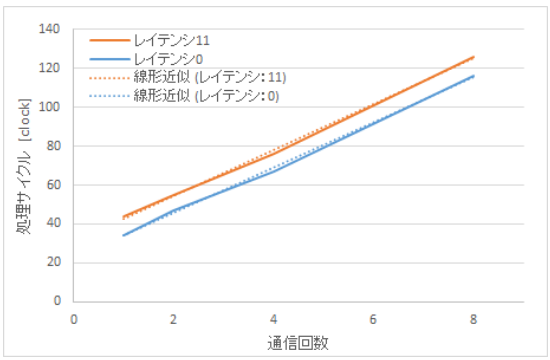

図 4 通信回数と送信サイクル（提案手法）

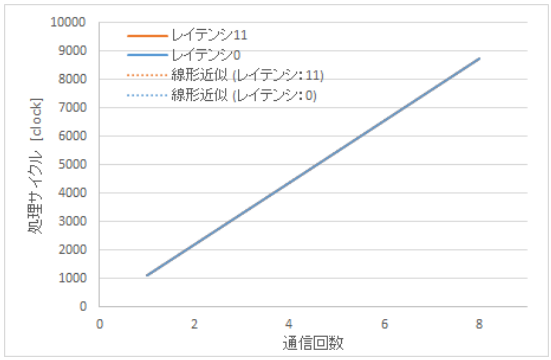

図 5 通信回数と受信サイクル（OS API）

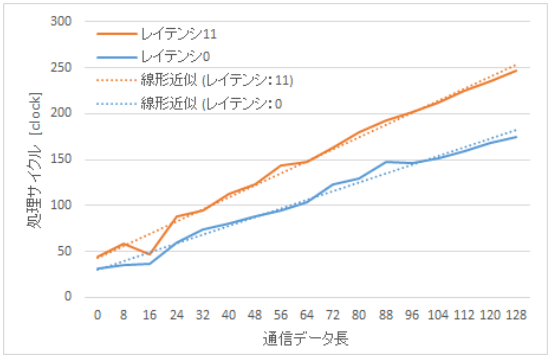

図 6 通信データ長と送信サイクル（提案手法）

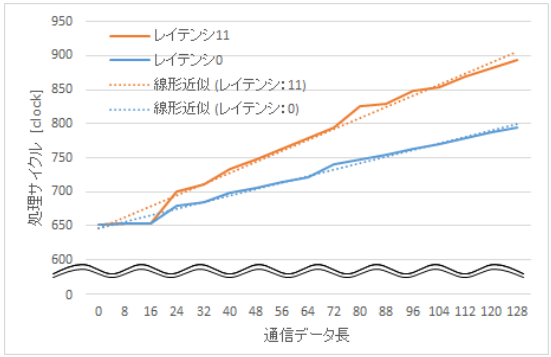

図 7 通信データ長と受信サイクル（OS API）

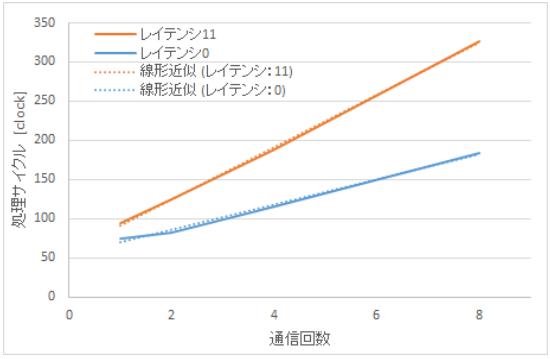

図 8 通信回数と送信サイクル（提案手法）

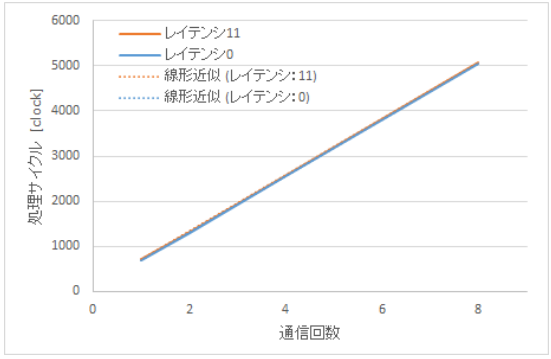

図 9 通信回数と受信サイクル（OS API）

表 2　線形近似により算出した通信サイクルの変化量（レイテンシ:11）

パラメータ	変更単位	パラメータの変更単位あたりの通信サイクル変化量 [clock] 提案手法	OS API
送信データ長	4byte	4.221	6.060
送信回数	1 回	11.73	1087
受信データ長	4byte	6.573	8.124
受信回数	1 回	33.38	621.5

$$R = (4.738) * l/4 + (16.14) * n + C_R \quad (2)$$

ここで C_S，C_R は定数で，初期化の処理などにかかるサイクルである．$l = 32$，$n = 1$ の際の測定結果から以下の値とした．

$$C_S = 44 - 2.591 * 32/4 - 11.62 * 1 = 11.65$$

$$C_R = 94 - 4.738 * 32/4 - 16.14 * 1 = 39.96$$

3.2.2　考察

提案手法は OS API よりも，通信回数に対する通信サイクル数が非常に小さく，実際の制御ソフトにおいても高速化が期待できる．

通信回数を変化させる場合，ソフト上では通信バッファへの書込みまたは，読出しの同じ処理を繰り返す回数が変わるのみである．そのため，通信回数に対し，通信サイクルが線形に変化することを期待したが，増加幅にばらつきがあった．これは，メモリへのストア命令やロード命令の実行サイクルがメモリのストアバッファやパイプラインの効果によって変化するためと考えられる．通信評価ソフトでは，通信バッファへの書込み，読出しを連続して行っているが，実際の制御ソフトでは，常に連続して行うとは限らず実アプリに近い頻度で書込み，読出しを行うソフトで測定することでより効果的な結果が得られると考えられる．

3.3　モータ制御モデルの通信サイクル測定

MATLAB/Simulink で作成したモータ制御モデルを実装し，OS API と提案手法でタスク処理サイクルを測定し，比較した．また，3.2 節の結果から求めた通信サイクルと実測値と比較した．

3.3.1　実験内容

MATLAB/Simulink のモータ制御モデルから，以下のソフトを作成した．

逐次ソフト:　MATLAB/Simulink から生成したソフト

並列ソフト:　モデルベース並列化ツールで並列化したソフト

並列ソフト 1:　通信に OS API を適用

並列ソフト 2:　通信に提案手法を適用

並列ソフトは 15 個のタスクで構成される．制御周期を作成する 1 個のタスクを除いた，14 個のタスクをそれぞれ 1 個ずつ実行し測定した．なお，測定対象タスクが受信するデータはそのタスクの実行前に用意しておくものとした．また，各タスクの送信，受信回数とデータ長を表 3 に示す．

表 3　タスクの通信詳細

	送信データ長 [byte]	送信回数	受信データ長 [byte]	受信回数
task1	8	2	8	1
task2	8	2	8	1
task3	32	6	16	2
task4	24	3	24	3
task5	0	2	16	1
task6	8	2	0	2
task7	8	2	0	2
task8	24	3	32	4
task9	0	1	0	4
task10	16	2	8	3
task11	8	2	32	2
task12	8	2	8	2
task13	8	1	8	2
task14	8	1	8	2

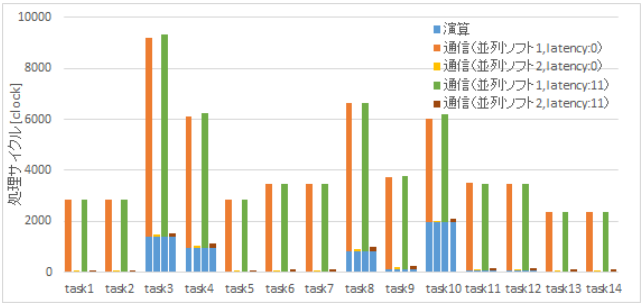

図 10　タスクの演算サイクルと通信サイクル

3.3.2　実験結果

並列ソフト 1 と並列ソフト 2 の実験結果を図 10 に示す．モータ制御モデルから生成・並列化したソフトは，タスクの粒度が小さく並列ソフト 1 ではタスクの大部分（68% 以上）が通信処理となっている．並列ソフト 2 では通信サイクルが並列ソフト 1 の 1%〜4%となり，タスクの処理サイクル（演算サイクルと通信サイクルの合計）も 2%〜34%となった．

モータ制御モデルの通信サイクルの計算値と実測値を図 11（並列ソフト 1），図 12（並列ソフト 2）に示す．並列ソフト 1 では，実測値に対する計算値の誤差が最大で 4% と小さくなったが，並列ソフト 2 では，誤差が最大 48% となった．

3.3.3　考察

通信に OS API を用いた場合，通信サイクルがモータ制御の演算サイクルよりも大きくなった．OS API では，タスクのスケジューリングや複数のタスク間が同時に通信を行った際の調停など行うため，オーバヘッドが大きく，CSP モデルでの並列化には提案方式の方が適しているとい

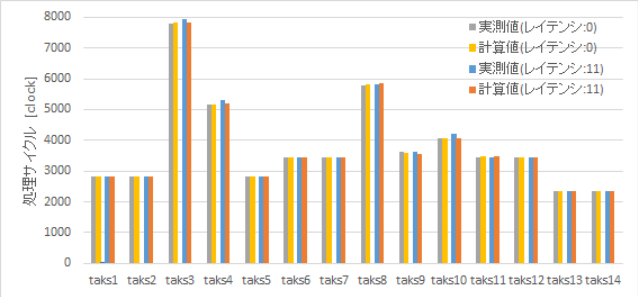

図 11 通信サイクルの計算値と実測値の比較（並列ソフト 1）

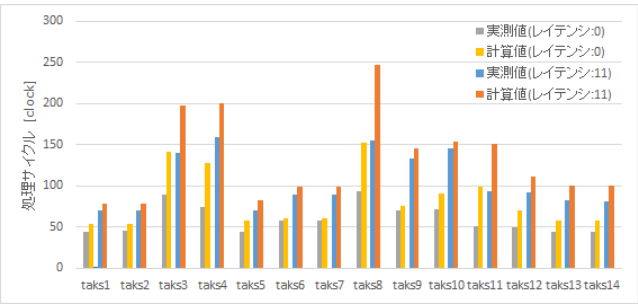

図 12 通信サイクルの計算値と実測値の比較（並列ソフト 2）

える．

通信サイクルの計算値と実測値の誤差の原因は，3.2.2 節で述べたばらつきによるものと考えられる．評価で用いたモータ制御ソフトでは，制御のための演算をする過程で，通信するデータの値が定まると，その時点で通信バッファに書き込む．そして，全ての演算が終わった後に通信を行う．そのため，通信サイクルの実測値の多くが計算値よりも小さくなったと考えられる．

3.4 モータ制御モデルの並列実行評価

モータ制御モデルを並列に実行し，通信オーバヘッドの削減により，フィードバックループの 1 ステップの処理サイクルが減少することを確認した．

3.4.1 実験内容

3.3.1 で作成した逐次ソフト，並列ソフト 1，並列ソフト 2 に加え，下記のソフトを作成し使用した．

並列ソフト 3: 並列化ソフト 2 に対し，同期タイミングを調整したソフト

並列ソフトの 15 個のタスクをマルチコアシミュレータのコアを 15 個用いて，1 コア 1 タスクとなるよう割り当て並列実行し，モータ制御ソフトを 1 ステップあたりの処理サイクルを測定した．

メモリ構成として，階層メモリ構成（図 13）とメッシュ構成（図 14）の 2 つの構成で評価した．階層メモリ構成は，4 コア毎に 1 個の Local RAM と全コアからアクセス可能な Global RAM を配置し，Local RAM へのアクセスレイテンシを 0 サイクルとし，Global RAM へのアクセスレイテンシは 0 サイクルから 11 サイクルまで変化させた．メッシュ構成では，4 コア*4 コアのメッシュ上に並んだ各コアに Global RAM を配置し，Global RAM へのアクセスレイテンシは自コアからのアクセスレイテンシは 0 とし，他コアからのアクセスレイテンシは ((hop 数+1) * (0〜11)) サイクルとした．

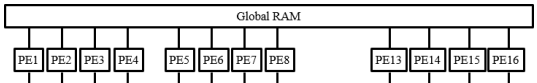

図 13 2 階層メモリ構成

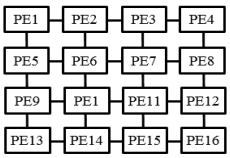

図 14 メッシュ構成

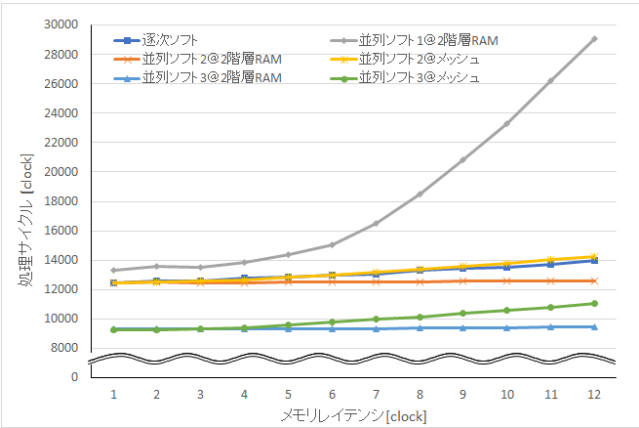

図 15 実験結果

3.4.2 実験結果

実験結果を図 15 に示す．モデルベース並列化ツールを用いて並列化すると，逐次ソフトよりも実行時間が増加したが，提案手法を適用することで，逐次ソフトと同程度の実行サイクルとなった．更に，同期タイミングを調整することで，逐次ソフトよりも 25%処理サイクルを低減した．

3.5 考察

実験では，階層メモリ構成よりもメッシュ構成の実行サイクルが長くなった．今回の実験では，タスクのコア割付けでタスク間の通信を考慮していないため，通信レイテンシが増加したと考えられる．そのため，タスク間通信を解析し，最適なタスク配置をすることでより高速化が可能と考える．

4. おわりに

1 コア 1 タスクに静的割付し，単方向 1:1 高速同期機構を用いたプロセッサ間通信を行うことで並列化のオーバヘッドを削減する手法を提案した．モータ制御モデルを用

いた評価を行い，提案手法を用いて並列化を行うことで，メニーコア向け高機能 OS 利用と比較して通信時間が 25 分の 1 となり，逐次実行よりも 25%実行サイクルを低減できることを確認した．

今後の課題としては，より正確な通信サイクルの計算手法の確立，タスクの最適な割り当てアルゴリズムの検討が必要である．

謝辞 本研究を進めるに当たり，モデルベース並列化ツールをご提供いただきました日本電気株式会社グリーンプラットフォーム研究所 久村孝寛様，マルチコアシミュレータと貴重なご意見をご提供いただきました，ルネサスエレクトロニクス 大槻典正様，城倉梨香様，鈴木均様，西博史様に厚く御礼申し上げます．

参考文献

[1] Simulink - シミュレーションおよびモデルベースデザイン (MBD) - MathWorks, http://www.mathworks.co.jp/products/simulink/, 2013/02/07.
[2] T. Kumura, Y. Nakamura, N. Ishiura, Y. Takeuchi, and M. Imai, ”Model Based Parallelization from the Simulink Models and Their Sequential C Code” in Proc. the Workshop on Synthesis And System Integration of Mixed Information Technologies (SASIMI 2012), R2-8, pp. 186-191 (2012).
[3] C. A. R. Hoare, Communicating Sequential Processes. the United Kingdom, Prentice Hall, 2011.
[4] コード生成 - Embedded Coder - Simulink - MathWorks, http://www.mathworks.co.jp/products/embedded-coder/, 2013/11/08.