Vol.2017-ARC-225 No.47
Vol.2017-SLDM-179 No.47
Vol.2017-EMB-44 No.47
2017/3/10

# 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

鍾　兆前[1,a)]　枝廣 正人[1]

**概要**：近年，車載制御システムの大規模化・複雑化に伴い，MATLAB/Simulink などを利用するモデルベース開発が普及しつつある．一方，性能向上のためプロセッサのマルチ・メニーコア化が進んでいる．しかし，実際に Simulink を用いて設計される制御ソフトウェアがマルチ・メニーコアの性能を享受するためには，並列化が必要である．また，制御設計の特徴をふまえたコア割当も重要である．本論文では，上記目的を達成するための自動並列化手法，特に混合整数線形計画法を用いるコア割当手法を提案する．車載モータ制御評価モデルを用いて既存手法と比較評価し，有効性を確認した．

# Model-Based Parallelizer for Multi- / Many-Core Processors

Zhaoqian ZHONG[1,a)]　Masato Edahiro[1]

## 1. はじめに

近年，消費電力や発生する熱等の問題により，プロセッサの動作周波数を上げることが困難になってきている．性能を向上させるためにはプロセッサの数を増やすしかなく，プロセッサのマルチコア化，メニーコア化が期待されている．それに応えて，数十，数百コアのシングルチップ・プロセッサが出現しつつあり，マルチコア・メニーコア製品は広まってきているが，組込みシステムにおいてソフトウェアの並列化など実現には多くの課題が見える．

一方，車載制御システムの大規模化・複雑化に伴い，MATLAB/Simulink[10] などのプラットフォームを利用するモデルベース開発が普及しつつある．Simulink モデルは，簡潔でわかりやすいブロック線図で構成され，モデルから組込み実装向けの逐次ソースコードを自動生成することも可能などの利点がある．Simulink モデルで記述する制御モデルをマルチコア・メニーコアへ実装するため，生成した制御ソフトウェアを分割し並列実行させることが重要である．この問題はグラフ最適化問題に帰着され，多くの最適化手法が提案されているが，一長一短がある．

さらに制御・実装設計特有の特徴も考慮する必要がある．例えば，ブロック図には制御観点での処理の開始点，終了点，順序関係があり得る．また，同じ制御周期，同じ機能モジュール（Simulink では Atomic Subsystem）内のブロックを同じコアに割当，あるいは近接させる等の配慮が必要である．実装観点では，同一メモリ資源への読み書きを同一コアにまとめる，複数ブロックをまとめた一括最適化コード生成への配慮，などが考えられる．

本研究では，MATLAB/Simulink で設計される制御モデルをマルチ・メニーコアプロセッサ向けに自動並列化する手法，**２重階層クラスタリング法**，を提案する．提案手法は，制御設計の特性をふまえた階層的なクラスタリングと，最適化手法の特性を生かした段階的なコア割当を用いる．その中で用いられる最適化手法の一つとして，混合整数線形計画法（Mixed-integer linear programming，以下 MIP）を用いるコア割当手法を提案する．MIP を用いることにより，Simulink モデルから生成したクラスタのコア割当を，クラスタの処理量を分散しながら，コア間の通信を最小化する結果を求め，かつ制御設計特有の性質も考慮する．ランダムクラスタグラフおよび車載モータ制御評価モデルを用いた実験では，従来手法である khmetis[6] から得た割当

[1] 名古屋大学
Furo-cho, Chikusa-ku, Nagoya, 464-8603 Japan
[a)] zhaoqian@ertl.jp

Vol.2017-ARC-225 No.47
Vol.2017-SLDM-179 No.47
Vol.2017-EMB-44 No.47
2017/3/10

結果と比較評価し，提案手法の有効性を確認した．

## 2. 関連研究

MATLAB/Simulink モデルから並列 C コード生成を生成する手法については，大きく分けて自動並列化コンパイラを用いる方法 [11] と，モデルレベルで並列化を考える方法 [3], [8] がある．いずれの方法も制御の特性をふまえつつ制御設計レベル（モデルレベル）で大規模問題を解くことに適用することは難しい．

これに対し我々は，モデルレベルで並列化を行うための環境を提案している [12]．本論文は [12] における自動コア割当機能に関するものである．

モデルレベルでブロックのコア割当を考える際，Simulink モデルのようなグラフ構造をコアに最適割当する必要がある．この問題は NP 困難のクラスに属するグラフ最適化問題に帰着され，様々な最適化手法の適用が試みられている．代表的には，1) グラフ分割手法 [1], [2], [5], [6]，2) クリティカルパス法 [7]，3) 混合整数線形計画法 (MIP)[13] が知られている．1) は高速解法があるが並列実行時間最小化を直接解いていない，2) は並列実行時間最小化を直接解いているが高速解法が知られておらず，また制御設計の特徴を考慮することが簡単ではない，3) は制約を入れた厳密解が求まるが，小規模問題にしか適用できない，といった特徴がある．

## 3. ２重階層クラスタリング法

本章では大規模問題への適用を可能とし，制御・実装設計の特徴をふまえた並列化手法，**２重階層クラスタリング法**を提案する．この方法では，まず制御の特徴をふまえながら階層的にクラスタリングを行い，その後，最適化手法の特徴をふまえながら段階的にコア割当を行う．

**図 1** に提案する手法の全体像を示す．この方法は以下の６段階からなる．

(1) 0 次クラスタリング

Simulink モデル内のブロックのうち，ユーザ指定により同一コアに割り当てられるべきブロック群をクラスタリングする．生成されたクラスタを **0 次クラスタ**とよび，コア割当完了まで分割されない．指定のないブロックは各ブロックが 0 次クラスタとなる．

(2) 1 次クラスタリング

0 次クラスタのうち，設計として同一コアに割り当てられることが望ましい集合をクラスタ化する．例えば同一メモリへの読み書き，一括コード生成により最適化されるブロック群が該当する．生成されたクラスタを **1 次クラスタ**とよぶ．ただし，ユーザが割当コア指定している 0 次クラスタは対象外とする．また，クラスタ化されない 0 次クラスタはそれぞれが 1 次クラスタとなる．

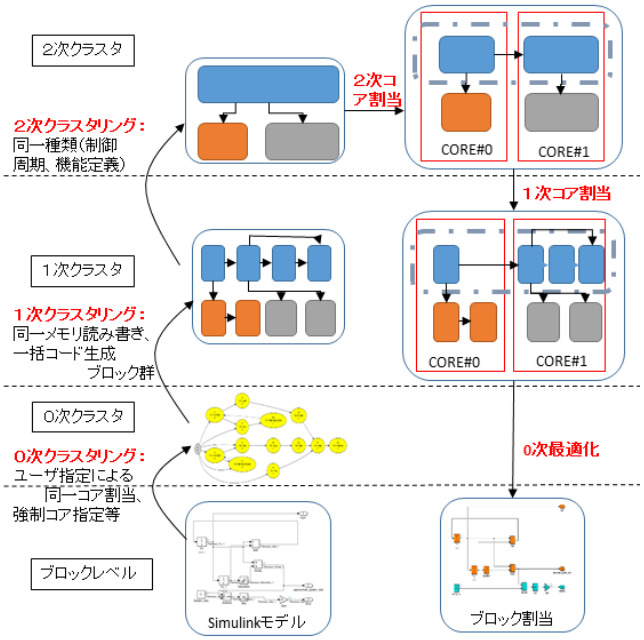


**図 1** ２重階層クラスタリング法
**Fig. 1** Double hierarchical clustering

(3) 2 次クラスタリング

1 次クラスタのうち，同一種類（本論文では同一周期かつ同一機能モジュール）の集合をクラスタ化する．ただし，ユーザが割当コア指定している 1 次クラスタは対象外とする．生成されたクラスタを **2 次クラスタ**とよぶ．クラスタ化されない 1 次クラスタはそれぞれが 2 次クラスタとなる．

(4) 2 次コア割当

2 次クラスタをコア割当する．処理量が単一コアの処理量上限を超えるクラスタは複数コアに割り当てる．複数コアに割り当てる際には，例えばコア 0 に 20%，コア 1 に 80%のように決める．ユーザが割当コア指定している強制割当に対応し，かつ接続度の高いクラスタを近接させる．同一周期や同一機能モジュールの優先近接割当などをユーザが選択できるようにする．大域的割当を決める極めて重要なフェーズであり，次章において MIP を用いたアルゴリズムを提案する．現実問題において 2 次クラスタ数は数十以下であり，MIP で実用的時間内に解くことができる．

(5) 1 次コア割当

各 2 次クラスタを 1 次クラスタに展開する．このとき複数コアに割り当てられた 2 次クラスタは，2 次コア割当で決められた割合（上記の例の場合コア 0 に 20%，コア 1 に 80%）に割り当てる．1 次コア割当には階層クラスタリング法 [1], [5] を用いている．そのため全体の手法を 2 重階層クラスタリング法とよんでいる．その際クリティカルパス法を併用し，前述のグラフ分割手法の問題を回避している．なお，クリティカルパス

解析では，制御特有の始終点や順序を考慮している．

(6) 0 次最適化

すべての 1 次クラスタを 0 次クラスタに展開し，さらにブロックレベルに展開，依存関係を考慮しながら計算順序を最適化する．

## 4. MIP を用いた 2 次コア割当

提案する 2 次コア割当では，混合整数線形計画 (MIP) を用い，クラスタの処理量を各コアに分散させ，かつ接続度の高いクラスタを近接させる．ユーザの指定に従い，コアへの強制割当を行う，あるいは，同じ周期または機能モジュールのクラスタを優先的に同じコアに割り当てる，などの対応を行う．

### 4.1 記法

2 次クラスタリング結果から**クラスタグラフ**を生成する．クラスタグラフの構成要素であるクラスタ，通信エッジと，割当ターゲットとなるコアを以下のように定義する．

クラスタを $clst = (cpu\_util, clst\_rate, clst\_atomic)$ と定義する．$cpu\_util$，$clst\_rate$，$clst\_atomic$ はそれぞれクラスタの見積処理量，周期，機能モジュールである．これを用いて $m$ 個のクラスタの集合を

$CLST = \{clst_i | i \in [0, m-1]\}$

と定義する．なお，$clst_i$ の $cpu\_util$ を $cpu\_util_i$ と書く．以下同様とする．

通信エッジを $conn = (conn\_s\_clst, conn\_t\_clst, conn\_weight, conn\_rate, conn\_atomic)$ と定義する．ここで，$conn\_s\_clst$ と $conn\_t\_clst$ はそれぞれエッジの始点，終点となるクラスタ番号である．$conn\_weight$ は見積通信時間とする．$conn\_rate$ はエッジ周期属性で，エッジ両端のクラスタが同じ周期を持つ場合 1，そうでない場合 0 とする．$conn\_atomic$ はエッジ機能属性で，エッジ両端のクラスタが同じ機能モジュールに属する場合 1，そうでない場合 0 とする．これを用いて $n$ 本の通信エッジの集合を

$CONN = \{conn_j | j \in [0, n-1]\}$

と定義する．

コアを $core = (core\_cpu\_util)$ と定義する．$core\_cpu\_util$ はコアに割り当てられているクラスタの処理量の合計である．これを用いて $c$ 個のコアの集合を

$CORE = \{core_p | p \in [0, c-1]\}$

と定義する．

ここで，処理量を分散するために各コアに設定する，割当クラスタの処理量上限を $max\_core\_cpu\_util$，下限を $min\_core\_cpu\_util$ とする．本論文では，すべてのクラスタの処理量 $cpu\_util$ の総和を $total\_cpu\_util$ とするとき，$max\_core\_cpu\_util = total\_cpu\_util/c * 1.05 + 1$ としている．また，$min\_core\_cpu\_util = 1$ としている．これらの値を，設計者による設定にする，各コアで異なる値を設定するといった拡張は容易である．

### 4.2 前処理によるクラスタ分割

各 2 次クラスタは同じ周期，同じ機能属性のブロックの集合であるため同じコアに割り当てることが望ましいが，本論文では対称型のコアに均等に分散することを目的としているため，$cave\_cpu\_util = total\_cpu\_util/c$ とするとき，$cpu\_util \geq cave\_cpu\_util$ なるクラスタは分割する必要がある．そのようなクラスタに対し，本手法では前処理として，処理量が $cave\_cpu\_util$ である一つ以上のサブクラスタと，処理量が $cave\_cpu\_util$ 未満である一つ以下のサブクラスタに分割し，前者は前処理でコアに割り当て，後者のみを MIP によるコア割当対象とする．

すなわち，$cpu\_util_i \geq cave\_cpu\_util$ なるクラスタ $clst_i$ に対して，$cpu\_util_i = cpu\_util_i \% cave\_cpu\_util$ と $cavedclst = cpu\_util_i / cave\_cpu\_util$ とする．ここで，$cavedclst$ は $clst_i$ を分割したクラスタ数であるが，コア $core_{c-1}$ から降順に空のコアに直接割り当てを行ない，それらのコアの $core\_cpu\_util$ も $cave\_cpu\_util$ にする．そして $c = c - cavedclst$ とし，すでに割り当て済のコアを $CORE$ から削除する．

なお，前処理後の 2 次クラスタはすべて，一つのコアに割り当てられることになることは注意すべき点である．

### 4.3 MIP の目的関数と制約条件

提案する MIP の定式化においては，コア間でカットされる通信エッジの重みを最小化する．これにより通信回数と通信時間を削減し，制御アプリケーション実行時間を削減，並列度を向上させる．ここで，制御周期優先，機能優先などを重みづけできるようにする．

MIP の定式化にあたり，以下の変数を定義する．

$x_{i,p}$：クラスタ $i$ がコア $p$ に割り当てられる場合 1，そうでない場合 0.

$y_j$：通信エッジ $j$ に対し，$conn\_s\_clst \neq conn\_t\_clst$ のとき 1，そうでない場合 0.

$y_j = 1$ のとき，通信エッジ $j$ が**コア間でカットされる**という．

このとき MIP の目的関数を以下のように定式化する．

$$minimize \sum_{j \in CONN} abs(conn\_weight_j) * (k_1 * conn\_rate_j + k_2 * conn\_atomic_j + 1) * y_j \quad (1)$$

なお，$k_1$ と $k_2$ はそれぞれ，ユーザが指定するクラスタの周期または機能モジュールの通信エッジに付けるペナルティである．$k_1$（$k_2$）を増やし，該当する通信エッジの重みを増加すると，そのエッジはカットにくくなるため，同じ周期（機能）を持つクラスタが同じコアに割り当てられ

Vol.2017-ARC-225 No.47
Vol.2017-SLDM-179 No.47
Vol.2017-EMB-44 No.47
2017/3/10

やすくなる．

制約条件は以下となる．

- 各クラスタは 1 つのコアにしか割り当てない．

$$\forall i \in CLST : \sum_{p \in CORE} x_{i,p} = 1 \qquad (2)$$

- クラスタ i をコア p に強制割当する．（ユーザ指定がある場合のみ）

$$x_{i,p} = 1 \qquad (3)$$

- 通信エッジのカットは以下のように計算される．

$$\forall j \in CONN : y_j = ( \sum_{p \in CORE} abs(x_{conn\_t\_clst_i,p} - x_{conn\_s\_clst_j,p}))/2 \qquad (4)$$

- あるコアに割り当てられるクラスタの総 *cpu_util* は以下のように計算される．

$$\forall p \in CORE : core\_cpu\_util_p = \sum_{i \in CLST} x_{i,p} * cpu\_util_i \qquad (5)$$

- すべてのコアにおいて下限を満たす．

$$\forall p \in CORE : core\_cpu\_util_p \geq min\_core\_cpu\_util \qquad (6)$$

- すべてのコアにおいて上限を満たす．

$$\forall p \in CORE : core\_cpu\_util_p \leq max\_core\_cpu\_util \qquad (7)$$

## 5. 評価実験

提案手法の有効性を示すために評価実験を行った．実験では MIP ソルバとして GLPK（GNU linear programming kit）[9] を用いた．パラメタ *gap tolerance* は 0.1 とした．

### 5.1 対象とする Simulink モデル

実験にはランダムクラスタグラフと，車載モータ制御評価モデルを用いた．

ランダムクラスタグラフとは，入力とする Simulink モデルを持たず，クラスタ数を指定することにより，クラスタと信号線の比率や，制御周期，機能モジュール属性のクラスタ数比率などを確率としてランダムに生成した 2 次クラスタグラフである．なお，確率によって生成する各種指標の値の範囲は，後述の車載モータ制御向け評価モデルを参考に調整している．

車載モータ制御評価モデルを**図 2** に示す．モデルには Triggered subsystem，S-function などの複雑な構造が含まれており，Simulink ブロック数は 514，2 次クラスタのクラスタ数は 26 であり，通信エッジ数は 60 であった．

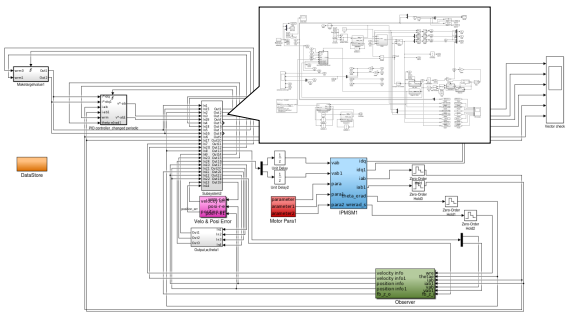

**図 2** 車載モータ制御評価モデル
**Fig. 2** Motor control model

### 5.2 比較対象

比較対象として khmetis[6] を用いた．khmetis は，マルチレベル k-way 分割手法を用いる hMETIS[5] の派生プログラムである．hMETIS は，Karypis 氏らが開発した大規模なハイパーグラフを分割するための効率のよいソフトウェアパッケージであり，マルチコア向けのタスク割り当て [8] や論理回路分割問題 [4] などによく使われている．

khmetis のパラメタである UBfactor は分割結果のバランスを制約する．例えば UBfactor が 5 である場合，分割後のサブグラフそれぞれのノード重みの和が平均値の 1.05 倍を超えないように分割する．今回の実験では，特に指定がない限り UBfactor を 5 としている．しかしながら，評価データにおける実際の実行結果において，UBfactor を 5 に指定しても分割結果の重みが平均値の 1.05 倍を超えてしまうデータがあり，クラスタを分散する目的が果たせない可能性があるとわかった．これは評価データのブロック重みのバランスが想定よりも大きいといった理由が考えられる．分割の偏りが大きい方がカット数を小さくできるため，そのようなデータは実験結果から除外した．

### 5.3 ランダムクラスタグラフを用いた評価

**表 1** は，ランダムクラスタグラフに対するコア割当の結果である．実験では，コア数を 4 とし，グラフを 1000 回ずつ発生させ，平均もしくは割合を算出した．ここで，表 1 の「khmetis 解が得られない%」とは，khmetis を実行して，分割結果の重みが平均値の 1.05 倍を超える割当結果を得た比率である．「MIP が優る%」とは，提案手法のカット値が khmetis のカット値より小さいか，同じである場合には MIP ソルバ時間が短いものの比率である．なお，ここでは khmetis 解が得られない場合を除いている．実験結果から，クラスタ数の増加とともに提案手法のソルバ時間が khmetis と比べかなり大きくなるが，多くの場合で良い割当結果が得られることがわかった．

次に，提案手法において機能・周期ペナルティの有効性を示すため，MIP モデルの $k_1$ と $k_2$ を増やし，前記のラ

Vol.2017-ARC-225 No.47
Vol.2017-SLDM-179 No.47
Vol.2017-EMB-44 No.47
2017/3/10

表 1　ランダムクラスタグラフに対するコア割当の結果
**Table 1**　Cluster-level core assignment on randomly generated cluster graphs

| クラスタ数 | MIP ソルバー時間 (s) | khmetis ソルバー時間 (s) | khmetis 解が得られない% | MIP が優る% |
|---|---|---|---|---|
| 10 | 0.03 | 1.11 | 30% | 99% |
| 20 | 4.48 | 12.25 | 70% | 70% |
| 30 | 35.67 | 16.92 | 48% | 100% |
| 40 | 32.71 | 22.41 | 33% | 90% |
| 50 | 167.19 | 9.29 | 10% | 91% |
| 60 | 572.57 | 3.81 | 0% | 100% |
| 70 | 609.35 | 4.83 | 0% | 93% |
| 80 | 1699.19 | 5.53 | 0% | 100% |
| 90 | 5283.67 | 6.46 | 0% | 100% |
| 100 | 6694.79 | 7.57 | 0% | 100% |

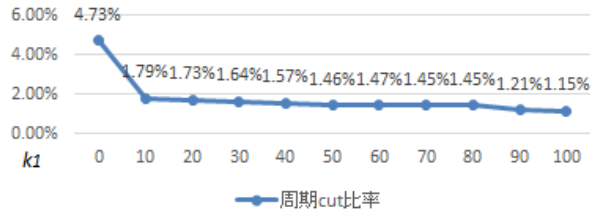


**図 3**　ランダムクラスタグラフに対するカット値と $k_1$ の関係
**Fig. 3**　Cut and $k_1$ in cluster-level core assignment on randomly generated cluster graphs

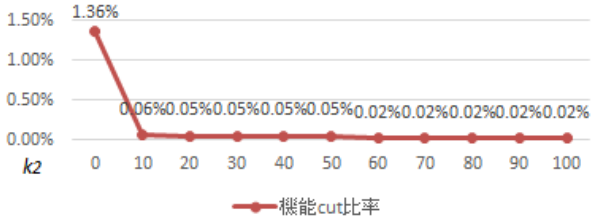


**図 4**　ランダムクラスタグラフに対するカット値と $k_2$ の関係
**Fig. 4**　Cut and $k_2$ in cluster-level core assignment on randomly generated cluster graphs

ンダムクラスタグラフを用いて 4 コア割当を行い，得られた 2 次コア割当結果の機能・周期属性のみのカット値と総カット値の比率を比較した．**図 3** は $k_2$ を 0 に固定し，$k_1$ を増やした場合の実験結果であり，**図 4** は $k_1$ を 0 に固定し，$k_2$ を増やした場合の実験結果である．それぞれの $k_1$ と $k_2$ の増加に伴い，同じ制御周期や同じ機能モジュール所属のクラスタを繋ぐ通信エッジがカットされにくくなることがわかる．しかしながら，$k_1$ と $k_2$ の適切な値はモデル依存であることもわかった．今後，入力したモデルを分析し，適切な $k_1$ と $k_2$ をユーザに提示できることが期待される．

### 5.4　車載モータ制御評価モデルの評価

車載モータ制御評価モデルを，コア数を 2 コアから 8

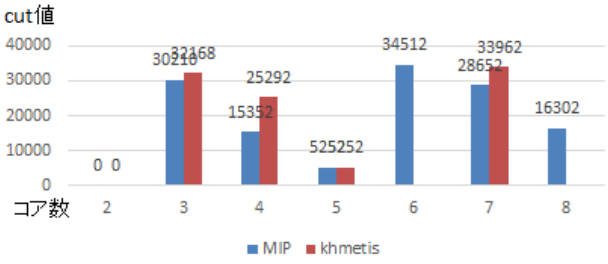


**図 5**　モータ制御モデルに対する 2 次コア割当結果の総カット値
**Fig. 5**　Cut in 2nd-level core assignment on motor control model

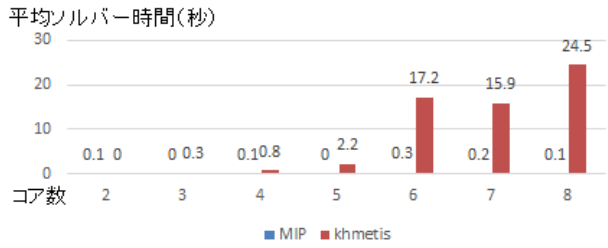


**図 6**　モータ制御モデルに対する 2 次コア割当の平均ソルバ実行時間
**Fig. 6**　Average solver time for 2nd-level core assignment on motor control model

コアまで変化させて割り当てることにより，提案手法と khmetis を比較した．なお，khmetis では内部で乱数を利用しているため，実験はそれぞれ 10 回ずつ実施し，結果を平均している．

#### 5.4.1　2 次コア割当の評価

2 次コア割当の評価として，通信エッジの総カット値と平均ソルバ実行時間を指標とした．結果を**図 5**，**図 6** に示す．ここで総カット値とは，割当結果においてカットされる通信エッジの通信時間の和である．なお，khmetis の実行結果において，コア数が 6 と 8 である場合，UBfactor を 5 から 30 まで試しても，分割の重みが指定基準を満たす解が得られなかったため図 5 から除いている．

総カット値と平均ソルバ時間の比較から，評価モデルにおいては，提案手法が khmetis より高速かつカット数の少ない割当結果を得られることがわかった．

#### 5.4.2　並列化の評価

前述の 2 次コア割当結果に基づき，1 次コア割当，0 次最適化まで実施し，最終的な結果を評価した．指標としては割当結果の並列性能向上と分散度を用いた．なお，並列性能向上は逐次実行時間と並列実行時間の比である．逐次実行時間はブロック処理量の総和（*total_cpu_util*）とし，並列実行時間はコア割当結果と実行順序に基づき，依存関係を考慮しながら制御アプリケーション実行時間を算出したものである．なお，コア間通信は 15 サイクルとしている．分散度は，上記逐次実行時間と *core_cpu_util* の最大値との比である．

**図 7** と**図 8** にその結果を示す．ここでは，2 次コア割当

Vol.2017-ARC-225 No.47
Vol.2017-SLDM-179 No.47
Vol.2017-EMB-44 No.47
2017/3/10

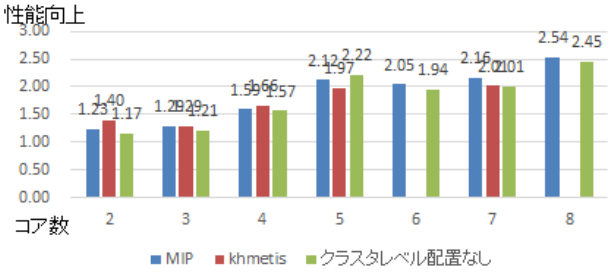


**図 7** モータ制御モデルに対するブロックレベル並列性能向上
**Fig. 7** Parallel speed-up ratio in block-level core assignment on motor control model

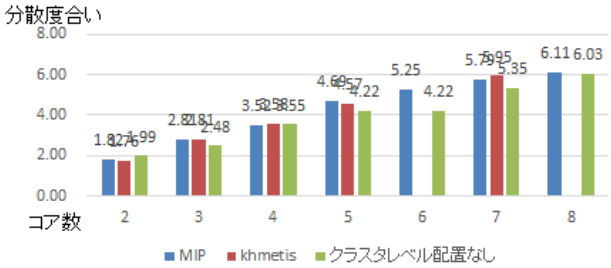


**図 8** モータ制御モデルに対するブロックレベル分散度
**Fig. 8** Load balancing in block-level core assignment on motor control model

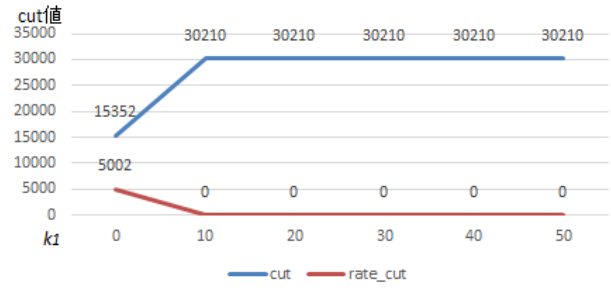


**図 9** モータ制御モデルに対するカット値と $k_1$ の関係
**Fig. 9** Cut and $k_1$ in cluster-level core assignment on motor control model

を単純な best fit 法を用いて実施した結果も比較対象としており，グラフにおいて「クラスタレベル配置なし」としている．結果から 3 つのアプローチにおいて同程度の割当結果が得られたことがわかる．

### 5.4.3 機能・周期ペナルティの有効性

提案手法において機能・周期ペナルティの有効性を示すため，MIP モデルの $k_1$ と $k_2$ を増やし，評価モデルを 4 コアに割り当て，2 次コア割当結果の総カット値と機能・周期属性のみのカット値を比較した．**図 9** は $k_2$ を 0 に固定し，$k_1$ を増やした場合の実験結果である．$k_1$ の増加に伴い，同じ制御周期のクラスタを繋ぐ通信エッジがカットされなくなる様子がわかる．$k_2$ を増やす実験においては，$k_1 = k_2 = 0$ の場合にもカットが存在しなかったため，変化がなかった．

## 6. おわりに

本論文は，MATLAB/Simulink を用いて設計される制御ソフトウェアをマルチ・メニーコア向け並列化するために，自動並列化手法，特に混合整数線形計画法 (MIP) を用いるコア割当手法を提案した．従来手法と比べ，MIP を用いるコア割当手法はコア間の通信時間を減少できることにより，性能向上が期待できるとわかった．また，MIP モデルのパラメタを調整することで，同じ制御周期，同じ機能を持つクラスタを同じコアに割り当てる目的も果たせることも確認できた．しかし，パラメタは入力とするモデルに依存するため，モデルを分析し適切なパラメタを提案することが必要と考えられる．なお，ブロックの実行時間や，通信エッジの通信時間は実際のマルチ・メニーコアアーキテクチャから大きな影響を受けている．そのため，実際のターゲットアーキテクチャを考慮した処理量や通信時間の高精度見積も今後の課題の一つである．

## 参考文献

[1] 油谷他, ”階層構造を持つメニーコアアーキテクチャへのタスクマッピング,” 情報処理学会論文誌, Vol. 56, No. 8, pp.1568-1581, 2015.
[2] M. Edahiro, et al., ”New Placement and Global Routing Algorithms for Standard Cell Layouts,” *DAC1990,* pp.642-645.
[3] R. Hottger, et al., ”Model-Based Automotive Partitioning and Mapping for Embedded Multicore Systems,” *International Journal of Computer, Electrical, Automation, Control and Information Engineering,* Vol.9, No.1, pp.268-274, 2015.
[4] 上土井他, ”大規模論理回路分割に関する一手法,” 情報処理学会研究報告, *SLDM*, 113-120, 1998.
[5] G. Karypis, et al., ”hMETIS 1.5: A hypergraph partitioning package,” Technical report, Department of Computer Science, University of Minnesota, 1998. Available on the WWW at URL http://www.cs.umn.edu/metis, 1998.
[6] G. Karypis, et al., ”Multilevel k-way Partitioning Scheme for Irregular Graphs,” *Journal of Parallel & Distributed Computing*, Vol. 48, No. 1, pp.96-129, 1998.
[7] H. Kasahara et al., ”Practical multiprocessor scheduling algorithms for efficient parallel processing,” *IEEE Trans. Comput.*, Vol. 11, pp.1023-1029, 1984.
[8] 久村他, ”Simulink モデルにもとづいた並列 C コード生成,” *ETNET2011*, pp.303-308.
[9] A. Makhorin, et al., GLPK (GNU linear programming kit). 2008.
[10] Mathworks, https://jp.mathworks.com/products/simulink.html
[11] 梅田他, ”MATLAB/Simulink で設計されたエンジン制御 C コードのマルチコア用自動並列化,” 情報処理学会論文誌, Vol. 55, No. 8, pp.1817-1829, 2014.
[12] 山口他, ”Simulink モデルからのブロックレベル並列化,” *ESS2015*, 21, 2015.
[13] Ying Yi, et al., ”An ILP formulation for task mapping and scheduling on multi-core architectures,” *DATE2009*, 2009.