情報処理学会論文誌 Vol.59 No.2 735–747 (Feb. 2018)

組込み制御システムに対するマルチコア向けモデルレベル自動並列化手法

鍾 兆前[1,a)] 枝廣 正人[1]

受付日 2017年5月19日, 採録日 2017年11月7日

概要：近年，車載制御システムなど組込み制御システムの大規模化・複雑化にともない，MATLAB/Simulink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．一方，性能向上のため組込み制御システム向けプロセッサのマルチコア化が進んでいる．しかし，実際にモデルベース開発を用いて設計される制御ソフトウェアがマルチ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ても重要である．本論文では，上記目的を達成するためのモデルレベル自動並列化手法，特にグラフ分割手法である2重階層クラスタリング法と，混合整数線形計画法を組み合わせたコア割当て手法を提案する．提案手法は，大規模問題の局所最適解を避けるために2重階層クラスタリング法を用いる．そして，上位階層の少数クラスタに対して混合整数線形計画の定式化を用い，制御モデルから生成したクラスタの処理量を分散させながら，コア間の通信コストを最小化するコア割当てを求める．ランダムクラスタグラフおよび実際の車載制御評価モデルを用いて，既存手法との比較評価を行い，有効性を確認した．

キーワード：モデルベース開発，自動並列化，マルチコア，組込み制御

Model-based Parallelizer for Embedded Control Systems on Multicore Processors

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

Received: May 19, 2017, Accepted: November 7, 2017

Abstract: In recent years, since the embedded control systems such as vehicle control systems are becoming larger and more complex, model-based development (MBD) with platforms such as MATLAB/Simulink is becoming increasingly common. Meanwhile, more and more multicore processors are used on embedded systems for better performance. However, in order to improve the performance of these control softwares designed with MBD on these multicores, proper parallelization and core assignment based on the features of control design are very important. This paper presents a model-level automatic parallelization approach, in particular, a core assignment method using a combination of the double hierarchical clustering method, which is a graph partitioning technique, and mixed integer linear programming (MIP) to achieve the above objects in model-based development on multicores. Our method uses the double hierarchical clustering method to avoid local optimum to partition large control models. For tens of higher level clusters, an MIP formulation is utilized to find the core assignment solution that balance the loads of these clusters generated from control models and minimize the communication cost across different cores on the processors. Also, we evaluated the proposed approach with existing methods on randomly generated cluster graphs and an automotive control evaluation model to address its effectiveness.

Keywords: model-based development, parallelization, multicore, embedded control systems

1. はじめに

1.1 背景

近年，消費電力や発生する熱などの問題により，プロ

[1] 名古屋大学大学院情報科学研究科
Graduate School of Information Science, Nagoya University, Nagoya, Aichi 464–8603, Japan
[a)] zhaoqian@ertl.jp

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が期待されている．それに応えて，数十，数百コアのマルチ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・複雑化にともない，MATLAB/Simulink [13] 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，簡潔で分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る．

このとき制御・実装設計特有の特徴も考慮する必要がある．制御設計をブロック線図で行い，並列化を行う際に必要となる前提，現在の並列化の課題などを以下にまとめておく．まず，現状の組込みシステムにおける制御モデルは，個々の処理単位の計算量が小さく，逐次的な依存関係が多数存在するために，並列性が高くないものが多いが，これがマルチ・メニーコアプロセッサに依存せざるをえない将来の高性能化に向け，深刻な課題となっている．そのため，今後，より複雑で高性能な制御を実現するためには，制御モデルから抽出された並列性を制御設計にフィードバックし，並列性を高めるアルゴリズムを考案するなどが必要と考えられている．たとえば，自動車エンジン制御のセンシング情報の中には，水温のように制御周期あたりの変化が小さいものがあるが，実装からの情報により効果の定量化ができれば，センシング周期をずらすことによるパイプライン並列化などを検討することが可能である．

ブロック線図には制御観点での処理の開始点，終了点，順序関係がありうる．これによって線図上はループであっても，実際には実行順序が規定できる．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り当て，あるいは近接させるなどの配慮が必要である．同じ種類（同じ周期，機能モジュール）のブロック集合は大小様々であり，大きいものでは数万ブロック，小さいものでは数ブロックのものがあり，ばらつきが大きい．1 つの制御モデル内における種類数は数十といわれている [9]．

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．上述したような，ループ構造を持つ線図をグラフ構造と見なし，グラフ分割手法によって分割した場合，処理の前半と後半に分割されるような結果になることがあるため，単純なグラフ分割では効率の良い並列化にはならないことが知られている．

1.2 論文概要

本論文では，完全結合で主記憶アクセスに衝突がないホモジニアス・マルチコアプロセッサに向け，制御モデルを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てを用いる．その中で用いられる最適化手法の 1 つとして，混合整数線形計画法（Mixed-integer linear programming，以下 MIP）を用いるコア割当て手法を提案する．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てを，クラスタの処理量を分散させながら，コア間の通信コスト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制御評価モデルを用いた実験において，従来手法から得た割当て結果と比較評価し，提案手法の有効性を確認する．なお，提案手法は MATLAB/Simulink のほか，Scilab [18] などのプラットフォームを用いたモデルベース開発にも適用可能であるが，本論文では MATLAB/Simulink を対象としている．

本論文の構成は以下のとおりである．まず，2 章では従来研究と本研究の位置づけについて述べる．3 章において提案手法である 2 重階層クラスタリング法を提案し，その中で用いられる MIP を用いた 2 次コア割当てについて 4 章で提案する．5 章で評価を行い，6 章に結論と今後の課題について述べる．

2. 従来研究と本研究の位置づけ

2.1 コードレベル並列化とモデルレベル並列化

モデルベース開発において制御モデルから並列 C コードを生成する手法については，大きく分けてコードレベルで並列化を行う方法とモデルレベルで並列化を行う方法があり，一長一短がある．

コードレベルで並列化を行う方法は，たとえば，MATLAB/Simulink モデルに対し Mathworks 社のコード生成ツール [12] を用いて逐次実行可能な C 言語コードを生成し，自動並列化コンパイラを用いて並列化する方法であり，OSCAR コンパイラを用いた方法 [20] が知られている．この方法は，Simulink モデルのブロックレベルよりも細粒度で並列化が可能であるため，並列性能を高くできる一方，生成コードにおいて制御観点の情報が失われてしまうため，制御の特性を生かした並列化が難しくなる．また，並列化結果を制御設計者にフィードバックし，制御視点での評価を行うことも難しくなる．

これに対し，モデルレベルで並列化を考える方法 [3], [10], [21] は，Simulink ブロックレベルの並列性を抽出し，制御の特性を生かした並列化を行い，制御設計者にフィードバックすることにより，今後のマルチコアを前提とした制御設計を容易にすることを特徴としてい

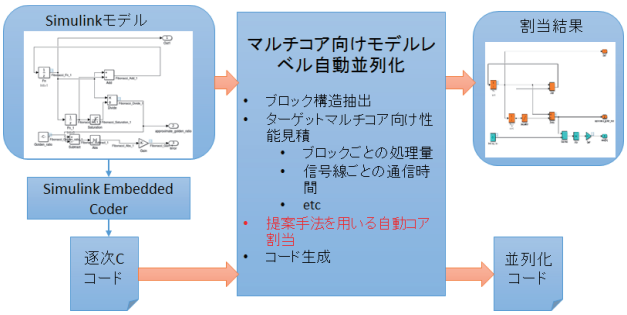

図 1 マルチコア向けモデルレベル自動並列化フロー
Fig. 1 Overview of automatic parallelization flow on model-level in model-based development for multicores.

る．図 1 は我々が提案するモデルレベル並列化環境である [21]．図 1 右上の割当て結果は，コア割当て結果を色分け表示したものであるが，制御設計者はこの図により並列実行を把握でき，制御モデルの改善を図ることができる．

制御設計においては，毎分数千回転するエンジンの回転角に対する水温のように，データ授受のタイミングを変更しても制御の安定性に影響しない場合がある．このとき，モデル内のタイミング変更により並列性を向上できる場合があるが [19]，このようなモデルレベルでの並列性向上制御設計には，コードレベル並列化よりモデルレベル並列化の方が向いている．

本論文では，モデルレベル並列化環境 [21] における自動コア割当てアルゴリズム，2 重階層クラスタリング法を提案する．次節では，既存手法 [3], [10] の概要と提案手法の特徴について述べる．

2.2 モデルレベルでの割当て手法

モデルレベルでブロックのコア割当てを考える手法としては，Simulink モデルをグラフ構造と見なし，コアに最適割当てする方法が一般的である．割当て手法は大きく動的割当てと静的割当てに分類される．

動的割当て手法は，グラフ構造を分割し，複数のタスクまたはスレッドにするが，コアへの割当てを事前に決めることなく，OS などの動的タスクスケジューラに任せる方法である．既存手法 [10] は動的割当てを用いている．この手法ではまず，Simulink モデルの階層をフラットに展開する．そして遅延素子部分でフィードバックループを切断することにより，処理に対応する Simulink ブロックをノード，処理順を有向エッジとする，ループのないグラフ構造を作成する．その後，ノードに対応する処理をタスクとし，処理順を守りながら，SMP OS（Symmetric Multi-Processor OS）によってタスクスケジューリングさせることにより並列化する．この方法で用いられているような動的手法では，粒度の粗い潤沢な並列性を内在するメディア処理アプリケーションなどにおいては性能の高い並列化が可能であるが，タスク処理量が少なく，タスク間依存関係の強い組込み制御アプリケーションにおいては高い並列性が望めない．

これに対し，静的割当てはあらかじめタスクのコア割当てを決めておく方法である．動的割当てと比べ，タスクスケジューラが起動されないため，オーバヘッドが小さく，組込み制御アプリケーションに向いている．既存手法 [3] は静的割当てを用いている．まず，制御設計の特徴によりクラスタ化し，生成されたクラスタを 3 種類のタスクスケジューリング法：CPP（クリティカルパス優先法），ESS（最早開始時間優先法），EDP（最短デッドライン優先法）によりスケジューリングする．評価実験により，CPP 法が優位であることを示している．リアルタイムスケジューリング理論においてよく知られているように，マルチコア向けスケジューリングは NP 完全問題であり [23]，大規模問題に対して実用的な時間内で最適解を求めることは難しい．

また，クラスタ化の際，実際の制御モデルにおいては，クラスタの処理量の偏りが大きい場合がある．たとえば，主たる周期，主たる機能のクラスタは処理量が大きく複数コアに割り当てる必要があるのに対して，処理量がきわめて小さく CPU 負荷がほとんどないクラスタも存在する，といったモデルが典型的な例である．このような場合，処理量の大きいクラスタを分割するとともに，周期や機能，CPU 負荷率を考慮してコア割当てを決めていく必要があるが，既存手法 [3] のようにマルチコアスケジューリングにより最適化することは容易ではない．

これに対し本論文では，処理量が大きく複数コアにまたがるクラスタに対してクリティカルパス優先法を併用したグラフ分割手法を適用し，単一コアを専有しない処理量の小さい複数クラスタの最適コア割当てに対しては混合整数線形計画法を用いることにより，上記課題を解決する．

2.3 静的割当てアルゴリズム

グラフ分割手法，クリティカルパス優先法，および混合整数線形計画法は，モデルレベル静的割当てのようにグラフ構造を分割し複数資源に配置していく NP 完全問題に対し従来から用いられている手法であり，LSI 設計やマルチコア向けタスク割当て問題に対して，1) グラフ分割手法 [1], [2], [6], [7]，2) クリティカルパス優先法 [8]，3) 混合整数線形計画法（MIP）[22] などの研究が知られている．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．

これに対し，本論文では，処理量の大きいクラスタ分割のため，高速手法であるグラフ分割手法を用いる．この方法は，コア間の処理量を均等にしつつ，コア間通信を最小化する．ところが，たとえば 2 コアに分割した際，処理の

前半と後半に分割されるなど，並列性のない分割結果になる場合がある．我々はクリティカルパス優先法を併用し，クリティカルパスがコア間で分割されにくくすることにより，問題を回避している．

また，制御の特性により種類分けした際，クラスタの数は数十になることが経験的に知られており [9]，その規模であれば現在の PC 程度の計算機能力があれば，混合整数線形計画（MIP）によって十分に解けるため，提案手法では厳密解法である MIP を用いている．

3. 提案する手法

本論文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．本章においては，2 重階層クラスタリング法の概要である並列化の流れを説明したうえでアルゴリズム全体の提案を行う．提案手法の中で最も重要な部分である，2 次コア割当てに対する MIP を用いた定式化については次章において詳しく説明する．

提案する手法はモデルレベルでの並列化を行うため，制御設計者に分かりやすく，制御視点での改善を容易にする．また，潤沢な並列性を持たない組込み制御システムを対象としているため，静的割当てにより，スケジューリングのオーバヘッドを少なくする．制御・実装設計制約を考慮し，かつ数万ブロック規模の Simulink モデルに対するコア割当てを実行するため，様々な制約条件を入れた厳密解を求めることができる MIP と，高速なグラフ分割を可能にする階層クラスタリング法を組み合わせた方法を提案する．これにより，制御設計の制約を守り，コア間通信を最小化することにより並列性能を向上させるコア割当てを，数万ブロック規模の大規模モデルでも実用的な時間内に並列化できるアルゴリズムが得られ，実際の組込み制御システムのマルチコア実装において効率良い並列化が実現できると考えられる．

3.1 自動並列化の流れ

提案手法では，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，それぞれの最適化手法の特徴を利用し，段階的にコア割当てを行う．

クラスタリングの過程では，まず，ユーザ指定や制御の特性上，同一コアに割り当てることが望ましいブロックを 1 次クラスタとしてクラスタリングを行う．その後，同一種類（たとえば，同一周期かつ同一機能モジュール）の集合をクラスタ化し，数十個の 2 次クラスタを形成する．

段階的コア割当てにおいては，まず，数十個の 2 次クラスタのコア割当て（2 次コア割当て）に対し，厳密解法である MIP を用いる．同一制御周期を同一コアに優先して割り当てるなどの様々な設計条件を MIP の制約条件として入れることを可能にすることにより，制御・実装設計の特徴をふまえた大域的なコア割当てを求める．個々の 2 次クラスタ内のコア割当て（1 次コア割当て）に対しては，高速なグラフ分割手法である階層クラスタリング法を用いる．実際の製品などで用いられる大規模データでは 1 つの 2 次クラスタ内に数万個の 1 次クラスタが含まれる場合があるが，1 次クラスタは，同一種類であること，同一コアに割り当てることが望ましいブロックがクラスタリングされていることから，複雑な設計条件が課されることはないと想定され，高速グラフ分割手法の適用が可能である．このような階層的なクラスタリング，段階的なコア割当てにより，提案する 2 重階層クラスタリング法は，制御・実装設計の特徴をふまえたコア割当てを高速に実現する．

Algorithm 2重階層クラスタリング法
Input: Simulinkモデル
Output: 各Simulinkブロックに対するコア割当
Step 1: 0次クラスタリング
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化し，0次クラスタを生成
Step 2: 1次クラスタリング
設計として同一コア割当が望ましい0次クラスタをクラスタ化し，1次クラスタを生成
Step 3: 2次クラスタリング
同一種類（例えば，同一周期かつ同一機能モジュール）の1次クラスタをクラスタ化し，2次クラスタを生成
Step 4: 2次コア割当
2次クラスタを混合整数線形計画法によりコア割当．複数コアに割り当てられる場合もある
Step 5: 1次コア割当
複数コアに割り当てられた2次クラスタのそれぞれに対し，1次クラスタに展開し，階層クラスタリング法によりコア割当
Step 6: 0次最適化
依存関係，計算順序を考慮しながらブロックレベルで局所的なコア割当改良

図 2 2 重階層クラスタリングアルゴリズム
Fig. 2 Double hierarchical clustering algorithm.

3.2 2 重階層クラスタリング法

図 2 に提案する 2 重階層クラスタリングアルゴリズムを示し，図 3 に手法の全体像を図示する．

Step 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，コア割当て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．

Step 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．たとえ

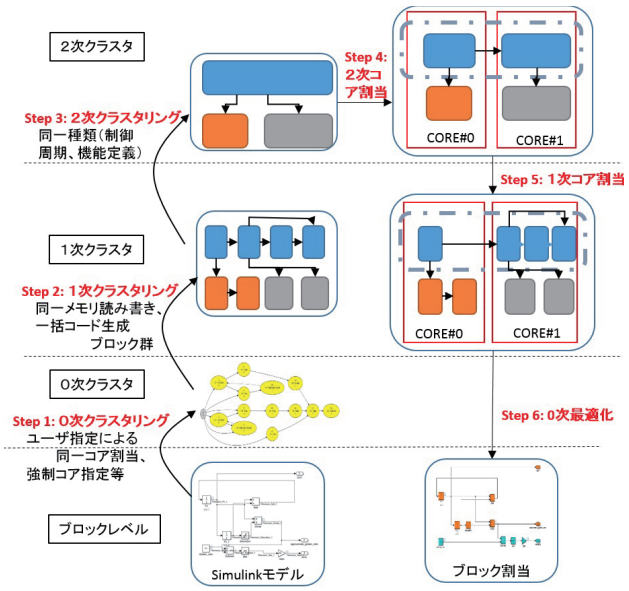

図 3 2 重階層クラスタリング法
Fig. 3 Double hierarchical clustering.

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，ユーザが割当てコア指定している 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．

Step 3：2 次クラスタリング

1 次クラスタのうち，同一種類（たとえば，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，ユーザが割当てコア指定している 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．

2 次クラスタのコア割当てに厳密解法を用いるため，適切なクラスタ数になるよう調整する．現実の問題において，同一周期かつ同一機能モジュールの 2 次クラスタ数は数十程度であるといわれている [9]．

Step 4：2 次コア割当て

2 次クラスタをコア割当てする．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，たとえばコア 0 に 20%，コア 1 に 80%のように決める．ユーザが割当てコア指定している強制割当てに対応し，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てなどをユーザが選択できるようにする．大域的割当てを決めるきわめて重要なフェーズであり，次章において MIP 定式化を用いたアルゴリズムを提案する．

Step 5：1 次コア割当て

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 次コア割当てで決められた割合（上記の例の場合コア 0 に 20%，コア 1 に 80%）に割り当てる．1 次コア割当てには階層クラスタリング法 [1], [6] 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，前章に記載したグラフ分割手法の問題を回避するため，クリティカルパス優先法を併用している．

まず，制御特有の始終点や順序を考慮してクリティカルパス解析を行う．そして，複数コアにまたがると並列性能が落ちるようなパスに対しては，パス上のエッジの重みを大きくし，グラフ分割手法を適用する．これにより，クリティカルパスがグラフ分割においてカットされにくくなる．

Step 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．

なお，1 次コア割当てにおいては，従来手法である階層クラスタリング法を用いている．高速性が重要であるが，Intel 社 Xeon(R) CPU E5-2695v2 @ 2.40 GHz（キャッシュサイズ 30,720 KB，主記憶サイズ 32 GB）を用いた PC 上での実験において，10 K ブロックのクラスタで約 10 秒，24 K ブロックのクラスタで約 18 秒であった．

4. 混合整数線形計画の定式化を用いた 2 次コア割当て

提案する 2 次コア割当てでは，混合整数線形計画（MIP）の定式化を用い，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ながら，コア間の通信回数と通信時間を最小化する 2 次クラスタのコア割当てを求める．さらに，ユーザの指定に従い，コアへの強制割当てを行う，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，などの条件を制約条件として入れる．これによって，設計の制約条件を厳密に守りながら，モデルから生成された制御アプリケーションの実行時間を削減，並列度を向上させるような大域的なコア割当てを求めることができる．

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，割当て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 $clst = (clst\_cpu\_util, clst\_rate, clst\_atomic)$ と定義する．$clst\_cpu\_util$，$clst\_rate$，$clst\_atomic$ はそれぞれクラスタの見積り処理量，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 $clst\_cpu\_util$ を $clst\_cpu\_util_i$ 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．$conn\_weight$ は見積り通信時間とする．$conn\_rate$ 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．

割当て候補となるコアを $core = (core\_cpu\_util)$ 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，ユーザの指定コア数に応じて 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，割当て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，完全結合で主記憶アクセスに衝突がないホモジニアス・マルチコアプロセッサを対象と想定しており，割当て候補となる対称型のコアはそれぞれ同じ仕様である．ゆえに，すべてのクラスタの処理量 $clst\_cpu\_util$ の総和を $total\_clst\_cpu\_util$ とするとき，均等な割当てを求めるため，$max\_core\_cpu\_util = total\_clst\_cpu\_util/c * 1.05$ としている．また，すべてのコアを利用できるため，$min\_core\_cpu\_util = 1$ 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，$cave\_cpu\_util = total\_clst\_cpu\_util/c$ とするとき，$clst\_cpu\_util_i \geq cave\_cpu\_util$ なるクラスタ $clst_i$ は分割する必要があると判断する．

そのような条件を満たすクラスタに対し，本手法では前処理として，処理量が $cave\_cpu\_util$ である 1 つ以上のサブクラスタと，処理量が $cave\_cpu\_util$ 未満である 1 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 によるコア割当て対象とする．

すなわち，$clst\_cpu\_util_i \geq cave\_cpu\_util$ なるクラスタ $clst_i$ に対して分割を行い，MIP に使用する $clst\_cpu\_util_i$ を $clst\_cpu\_util_i \% cave\_cpu\_util$ に再定義する．また $cavedclst = \lfloor clst\_cpu\_util_i / cave\_cpu\_util \rfloor$ 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当てを行い，それらのコアの $core\_cpu\_util$ を $cave\_cpu\_util$ とする．そして $c = c - cavedclst$ と再定義し，すでに割り当て済みのコアを $CORE$ から削除する．

なお，前処理により 2 次クラスタが分割されることになるが，2 次コア割当てでは各コアに割り当てられる 2 次クラスタの処理量を計算しているのみであり，実際の 2 次クラスタ分割は 1 次コア割当てにおいて階層クラスタリング法を用いて最適化される．

4.3 MIP 定式化の目的関数と制約条件

提案する MIP の定式化においては，クラスタグラフにおいて異なるコアに割り当てられるクラスタ間の通信エッジの重み合計を最小化することで，通信コストを最小化する．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，クラスタ $conn\_s\_clst_j$ とクラスタ $conn\_t\_clst_j$ が同じコアに割り当てられる場合，すなわち，

$$\sum_{p \in CORE} x_{conn\_t\_clst_j, p} * p == \sum_{p \in CORE} x_{conn\_s\_clst_j, p} * p$$

を満たすとき 0，そうでない場合 1.

$y_j == 1$ のとき，通信エッジ j が**コア間でカットされる**といい，すべての $y_j == 1$ なる通信エッジ j の重みの合計をクラスタグラフ分割の**カット値**とよぶ．

この**カット値**はコア割当て結果においてクラスタ間の通信時間の合計も表すため，コア割当ての良し悪しを評価するための指標である**通信コスト**とする．

このとき MIP の目的関数を以下のように定式化する．

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

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．

MIP の定式化において，k_1 を増やし，$conn\_rate_j = 1$ に

該当する通信エッジ j の重みを増加させると，そのエッジがカットされると通信コストが増加することになるため，そのエッジがカットされにくくなる．それにより同じ周期を持つクラスタが同じコアに割り当てられやすくなる．同様に，k_2 を増やす場合にも同じく，同じ機能モジュールに属するクラスタが同じコアに割り当てられやすくなる．

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．

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

- クラスタ i をコア p に強制割当てする（ユーザ指定がある場合のみ）．

$$x_{i,p} = 1 \tag{3}$$

- 通信エッジのカットは以下のように計算される．

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

- 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．

$$\forall p \in CORE : core\_cpu\_util_p = \sum_{i \in CLST} x_{i,p} * clst\_cpu\_util_i \tag{5}$$

- すべてのコアにおいて下限を満たす．

$$\forall p \in CORE : core\_cpu\_util_p \geq min\_core\_cpu\_util \tag{6}$$

- すべてのコアにおいて上限を満たす．

$$\forall p \in CORE : core\_cpu\_util_p \leq max\_core\_cpu\_util \tag{7}$$

しかし，$clst\_cpu\_util_i \geq max\_core\_cpu\_util/2$ なる 2 次クラスタの個数が c 以上となると，提案した MIP の定式化に対し解が存在しないことは注意すべき点である．

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．評価実験は，2 次コア割当ての評価と並列化アルゴリズム全体の評価を実施した．

2 次コア割当てに関しては，評価の基準として分散度と通信コスト，およびツール実行時間を用いた．ここで，分散度とはコアごとの負荷バランスの良さを示す指標であり，$total\_clst\_cpu\_util/\max_p\{core\_cpu\_util_p\}$ で計算する．通信コストは，MIP 定式化において定義したものと同様であり，コア間でカットされる通信エッジの重みの総和である．

並列性能向上のためには，コアごとの負荷バランスの良さ，コア間通信オーバヘッドの少なさ，データ依存などに起因する待ち時間の少なさが重要な要素である．このうち依存関係は，クラスタリングによって抽象化されるため評価が難しいが，2 次クラスタは制御の性質（制御周期，機能モジュール）を用いてクラスタ化しているため，2 次クラスタ間では複雑な依存関係を有しないと想定できる．そのため，2 次コア割当て評価には分散度と通信コスト，およびツール実行時間を用いる．

並列化アルゴリズム全体の評価においては，並列性能向上と分散度について評価する．

実験には Intel(R) Xeon(R) CPU E5-2695v2 @ 2.40 GHz（キャッシュサイズ 30,720 KB，主記憶サイズ 32 GB）を搭載した PC を使用した．

評価モデルとして，実際の組込み制御モデルのほか，組込み制御モデルを模擬したランダムグラフ，特殊構造を持つグラフを用いた．

評価モデルとしては，実際の組込み制御システムから取得したモデルを多種，多数準備して評価すべきであるが，実際の制御システムから多種類の例題を入手することが容易ではなく，制御工学的に意味がある多種類の Simulink モデルを人工的に自動生成することも簡単ではないため，ランダムグラフによる実験を行った．ランダムグラフは 2 次クラスタグラフを模擬したものであり，2 次コア割当ての性能評価にのみ用いられる．車載モデルを模擬したランダムグラフ（以下では**ランダムクラスタグラフ**とよぶ）は，実際の車載制御モデルから，グラフのノード数とエッジ数の割合，ノードの次数，ノードの重み，エッジの重みなどのパラメータを測定し，その値から大きく外れないように乱数を発生させ，所望のノード数のグラフを生成している．

また，グラフ分割手法は，重み付き和などの目的関数を用い，発見的手法によって最適化するため，人間にとっては簡単な例題でも最適解が得られない場合がある．そのため，最適解が分かっているモデルを評価することも重要であり，特殊構造を持つグラフも評価モデルに加えた．特殊構造グラフも，制御設計としては意味がなく，2 次コア割当てアルゴリズムの特性を評価する目的で用いた．

並列化アルゴリズム全体の評価に対しては，実際の車載制御アルゴリズム研究から取得したモデル予測制御モデルを用いた．

5.1 実験対象

以下では，実験対象とするランダムクラスタグラフ，特殊構造グラフ，モデル予測制御モデルについて説明する．

5.1.1 ランダムクラスタグラフと特殊構造グラフ

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

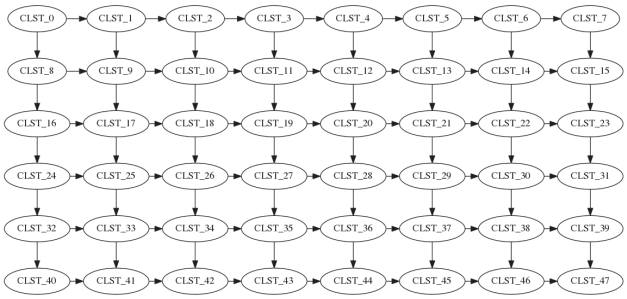

図 4 6 行*8 列の Mesh グラフ例
Fig. 4 6*8 mesh graph.

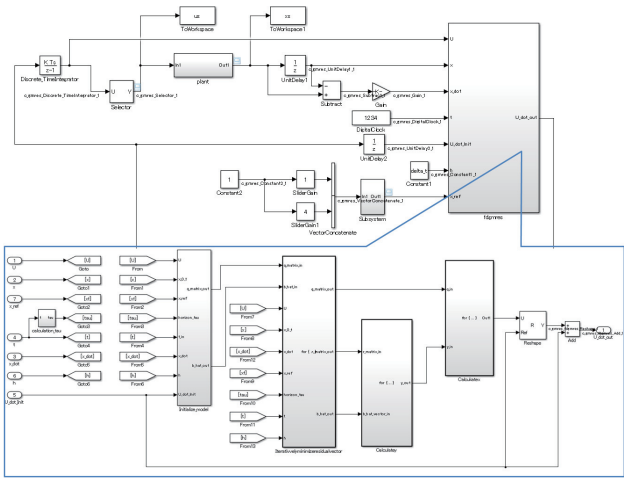

図 5 モデル予測制御モデル
Fig. 5 Model predictive control model.

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，2 次クラスタの接続関係などを乱数を用いて生成した 2 次クラスタグラフである．確率によって生成する各種指標の値の範囲は，車載制御向けモデルを参考に調整している．

また，特殊な形状を持つグラフ構造におけるアルゴリズム性能を評価するため，二分木構造と Mesh トポロジにより生成された 2 次クラスタグラフを用いた．二分木構造の 2 次クラスタグラフは，上記ランダムクラスタグラフと同様の実験結果を得たため，本論文では Mesh トポロジの結果のみを述べる．

Mesh トポロジは対称的な構造であり，直感的な分割との比較による性能評価を目的としている．また，特殊な構造の場合のアルゴリズムの得意不得意も評価の目的である．本論文では，縦 M 行*横 N 列の 2 次 Mesh グラフを実験対象としており，各ノードにおいて縦横方向の隣接ノードにエッジを持っている．**図 4** は 6 行*8 列の 2 次メッシュグラフである．

5.1.2 モデル予測制御モデル

モデル予測制御モデルを**図 5** に示す．このモデルは実際の車載モータ制御研究から抽出されたものであり，モデル予測制御アルゴリズムには C/GMRES 法を用いている．GMRES 法（generalized minimal residual method）[17] は，連立一次方程式の数値解を求めるための反復法の一種であり，C/GMRES（Continuation/GMRES）とは連続変形法を利用し演算量を激減する GMRES 法であり，連続した変形を行うため並列性は高くない．

モデル予測制御モデルにおいて，Simulink ブロック数は 2,147，0 次・1 次クラスタリングを実行させ，2 次クラスタグラフの 2 次クラスタ数は 29 であり，通信エッジ数は 57 であった．

5.2 比較対象

比較対象としてグラフ分割法 khmetis [7] とクリティカルパス優先法 [8] を用いた．

5.4 節および 5.5 節の 2 次コア割当て評価は，2 次クラスタに対するコア割当て部分のみの評価であるが，2 次コア割当てに khmetis を用いた手法と，提案手法である MIP を用いた手法を比較評価する．2 次コア割当てにおいてはクリティカルパスがクラスタ内に隠されてしまっていて公平な評価とならないため，2 次コア割当てに対しては khmetis とのみ比較評価を行った．

5.6 節の並列化全体での評価としては，提案する 2 重階層クラスタリング法と，単階層で khmetis を用いた方法，および単階層でクリティカルパス優先法を用いた方法を比較評価する．ここで単階層とは，提案手法のような階層化を行わず，Simulink ブロックの階層でコア割当てを行う方法を意味する．特に khmetis 法は階層クラスタリング法を応用したグラフ分割手法であるため，提案手法における 1 次割当てのみの方法とも考えることができ，提案手法において 2 次クラスタリングを行い，厳密手法である MIP 法を用いた効果を確認できる．

khmetis は，マルチレベル k-way 分割手法を用いる hMETIS [6] の派生プログラムである．hMETIS は，Karypis 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良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当て [10] や論理回路分割問題 [5] などに利用されている．

khmetis のパラメータである UBfactor は分割結果のバランスを制約する．たとえば 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総和が平均値の 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分かった．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．

表 1 *gap tolerance* によるソルバ時間減少
Table 1 Solver time reduction due to gap tolerance.

クラスタ数	gap = 0 の平均ソルバ時間	gap = 0.1 の平均ソルバ時間	時間減少%
20	5.63	4.48	20.43%
30	45.19	35.67	21.07%
40	73.95	32.71	55.77%
50	210.32	167.19	20.50%

クリティカルパス優先法 [8] は各処理単位（本論文の場合にはブロック）のコア割当てに対する決定木探索になるが，決定順をクリティカルパスを考慮して決めるところが特長である．ブロック数に対して指数関数的に木が大きくなるため，ブロックが多い場合，決定木を根から葉まですべてたどることができず，途中で適切に打ち切りながら探索を進めることになる．このときにクリティカルパスを考慮した順序とすることにより，不適切な解を出しにくいという特徴がある．そのうえで，様々な高速木構造探索アルゴリズムを適用可能である．

5.3 MIP ソルバ

本研究においては，MIP 問題を解くための多数のソルバから，GLPK（GNU linear programming kit）[11] を用いた．GLPK は商用ソフトウェアと比べると性能は高いとはいえないが，オープンソースソフトウェアという利点がある [14]．

なお，GLPK の実行パラメータである *gap tolerance* とは，MIP のソルバにおいてまだ検証していない解の範囲を表す．5.1.1 項のランダムクラスタグラフを用いた実験を実行した結果を**表 1** に示す．実行結果から，*gap tolerance* が 0.1 である場合は 0 である場合と同じ割当て結果を得られたうえ，ソルバ時間を大幅に改善できることが分かった．この結果より，本研究ではパラメータ *gap tolerance* を 0.1 と設定した．つまり，検証範囲が 10%に落ちると GLPK ソルバの実行を中止させ，割当て結果を出力する．

また，GLPK のソルバ実行時間の上限を 18,000 秒（5 時間）とした．

5.4 ランダムクラスタグラフを用いた 2 次コア割当て評価

5.4.1 割当て性能

本実験では，コア数を 4 とし，それぞれのパラメータごとに指定した 2 次クラスタ数のランダムクラスタグラフを 1,000 回ずつ発生させ，対象アルゴリズムを実行させ，各種評価値の平均もしくは結果の割合を算出した．**表 2** は，ランダムクラスタグラフに対するコア割当ての結果である．

ここで，表 2 の「MIP ソルバ時間（s）」と「khmetis ソルバ時間（s）」とは，MIP を用いた提案手法と khmetis を用いた従来手法がコア割当てを得た実行時間（秒）の平均値である．「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 倍を超える割当て結果を得た比率であり，分散度の指標となっている．「MIP が優る%」とは，提案手法の通信コストが 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．ここでは khmetis 解が得られない場合を除いている．なお，ここの「MIP が優る%」がすべて 100%でない原因は，提案手法のソルバ時間が khmetis のソルバ時間より長いことであり，提案手法から得たコア割当ての通信コストが khmetis から得たコア割当てより大きい例はなかった．

表 2 ランダムクラスタグラフに対するコア割当ての結果
Table 2 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	1,699.19	5.53	0%	100%
90	5,283.67	6.46	0%	100%
100	6,694.79	7.57	0%	100%

ランダムクラスタグラフを用いた実験結果から，提案手法により，多くの場合で良い割当て結果が得られることが分かった．通信コストの最小化によって，2 次クラスタグラフに対応するアプリケーションの実行時間も削減されると期待できる．しかし，選定した GLPK の性能制限があるため，2 次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．そのため，2 次クラスタ数の調整が重要である．もしも 2 次クラスタ数を増加させ，2 次コア割当て最適化機能を強める場合には，ユーザが高性能な商用ソフトウェアを導入することが望ましい．

5.4.2 機能・周期ペナルティの有効性

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 定式化の k_1 と k_2 を増やし，前記のランダムクラスタグラフを用いて 4 コア割当てを行い，得られた 2 次コア割当て結果において，$conn\_rate == 1$ や $conn\_atomic == 1$ となる，機能・周期属性のみのカット値と通信コストの比率を比較した．

図 6 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．また，**図 7** 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 の増加にともない，$conn\_rate == 1$ や $conn\_atomic == 1$ となる通信エッ

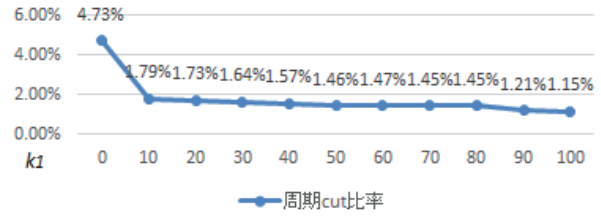

図 6 ランダムクラスタグラフに対するカット値と k_1 の関係

Fig. 6 Cut and k_1 in cluster-level core assignment on randomly generated cluster graphs.

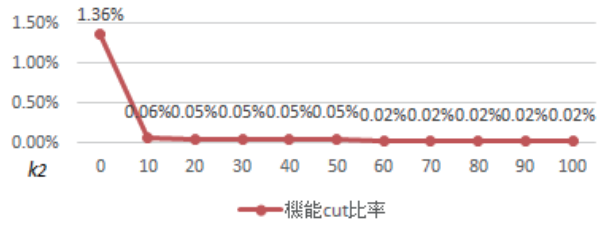

図 7 ランダムクラスタグラフに対するカット値と k_2 の関係

Fig. 7 Cut and k_2 in cluster-level core assignment on randomly generated cluster graphs.

ジがコア間でカットされにくくなり，同じ制御周期や同じ機能モジュール所属のクラスタを同じコアに割り当てられやすくなることが分かる．

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分かった．目的関数（式 (1)）より，見積り通信時間 *conn_weight* のばらつきに対して k_1 と k_2 の影響が変化することが明らかである．今後，入力したモデルを分析し，見積り通信時間の解析を通して適切な k_1 と k_2 をユーザに提示できることが期待される．

5.4.3 通信エッジの見積り通信時間の影響

本実験では，コア数を 4 とし，2 次クラスタの見積り処理量 *clst_cpu_util* を 1 と固定し，通信エッジの見積り通信時間 *conn_weight* を 1 に固定，または 1 から 10，1 から 100 の範囲にランダムに発生させ，各指定した 2 次クラスタ数のランダムクラスタグラフを生成し，評価を行った．**表 3**，**表 4** と，**表 5** は，それぞれ $conn\_weight = 1$，$1 \leq conn\_weight \leq 10$ と，$1 \leq conn\_weight \leq 100$ なる場合の実行結果であり，各項目は 5.4.1 項と同じである．

実行結果から，入力となる 2 次クラスタグラフの各通信エッジにおいて，見積り通信時間 *conn_weight* の差分が大きいほど，提案手法のソルバ実行時間が減少する傾向があると分かった．提案手法の MIP 定式化はコア間でカットされる通信エッジの見積り通信時間 *conn_weight* の合計の最小化を目標としており，GLPK ソルバが *branch and bound* 法を用いて MIP 定式化を解くため，*conn_weight* の差分が小さいと候補解が多くなり，最適化時間も長くなる．このような場合には，*gap tolerance* を調整して打ち切りを早めるなどの対策が必要である．

表 3 $conn\_weight = 1$ のランダムクラスタグラフに対するコア割当ての結果

Table 3 Cluster-level core assignment on randomly generated cluster graphs ($conn\_weight = 1$).

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	0.29	3.30	11%	100%
20	208.25	0.84	0%	76%
30	6,273.93	1.49	0%	49%
40	14,127.90	2.05	0%	73%
50	18,000.00	2.63	0%	81%

表 4 $1 \leq conn\_weight \leq 10$ のランダムクラスタグラフに対するコア割当ての結果

Table 4 Cluster-level core assignment on randomly generated cluster graphs ($1 \leq conn\_weight \leq 10$).

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	0.24	3.05	40%	85%
20	15.85	0.73	0%	90%
30	150.09	10.37	24%	97%
40	4,185.08	2.12	0%	85%
50	16,317.00	3.14	0%	83%

表 5 $1 \leq conn\_weight \leq 100$ のランダムクラスタグラフに対するコア割当ての結果

Table 5 Cluster-level core assignment on randomly generated cluster graphs ($1 \leq conn\_weight \leq 100$).

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	0.14	3.20	11%	100%
20	9.73	0.60	0%	100%
30	191.83	6.21	0%	96%
40	3,162.80	2.05	0%	84%
50	13,744.50	2.92	0%	91%

5.5 Mesh トポロジに対する 2 次クラスタ割当て評価

本実験評価では，様々な大きさのメッシュ，ノードやエッジのパラメータについて 2 次クラスタグラフを生成し，各 Mesh グラフに対し提案手法と khmetis を用いた手法を 100 回実行させ平均を算出した．

本論文では，典型的な例として，2 次クラスタの見積り実行時間を 1 と設定し，通信エッジの通信時間を 1 と設定した 6 行*8 列 Mesh グラフ（図 4）に対する，4 コア割当てと，6 コア割当ての結果を用いて議論する．

図 8 は 6 行*8 列 Mesh グラフに対する 4 コア割当ての結果の図示であり，最適解が算出できていることが分かる．

表 6 は，6 行*8 列 Mesh グラフに対する 4 コア割当てと 6 コア割当ての結果を示す．4 コア割当ての結果では，khmetis は高速であるが，最大カット値 16，最小カット値 14 となり，つねに最適解が得られるわけではなかった．一

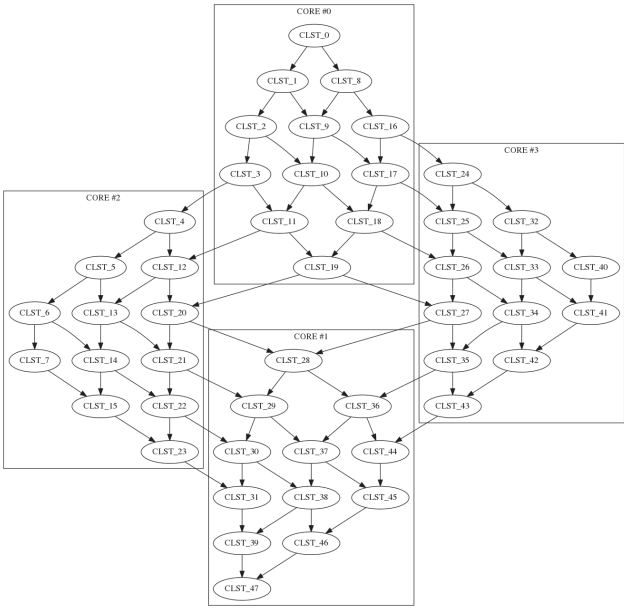

図 8 6 行*8 列 Mesh グラフ（*conn_weight* = 1）に対する提案手法の 4 コア割当て結果の図示

Fig. 8 Proposed cluster-level core assignment on 6*8 mesh graph (*conn_weight* = 1) for 4 cores.

表 6 6 行*8 列 Mesh グラフに対するコア割当ての結果

Table 6 Cluster-level core assignment results on 6*8 mesh graph.

手法	コア数	平均カット値	平均ソルバ時間（s）
MIP	4	14	236.1
khmetis	4	14.4	1.9
MIP	6	25	18,000
khmetis	6	22.3	2.8

方，MIP の方は必ず最適解が得られるが，ソルバ時間は長かった．

6 行*8 列 Mesh グラフに対する 6 コア割当ての場合には，コア数が大きくなるにともない，MIP 定式化の制約条件数と変数の個数もかなり増加し，また各通信エッジの見積り通信時間に差分がないため，GLPK ソルバは設定した実行時間上限（18,000 秒）に達し中止された．そして，中止されたとき，GLPK ソルバの性能制限により未検証の候補解の範囲を表す *gap* はまだ 50%程度であり，6 コアの 6*8Mesh グラフの MIP 定式化が十分解かれていないことが分かった．**図 9** は 6 行*8 列 Mesh グラフに対する 6 コア割当ての十分解かれていない結果の図示であり，提案手法が出力した割当て結果が最適解ではないため，khmetis より劣る可能性がある．

最適に近い解が多数あるような場合，厳密解法である MIP はソルバ時間が長くなる．このような場合に対しては，*gap tolerance* の調整，商用ソルバの適用などの解決策が必要となる．

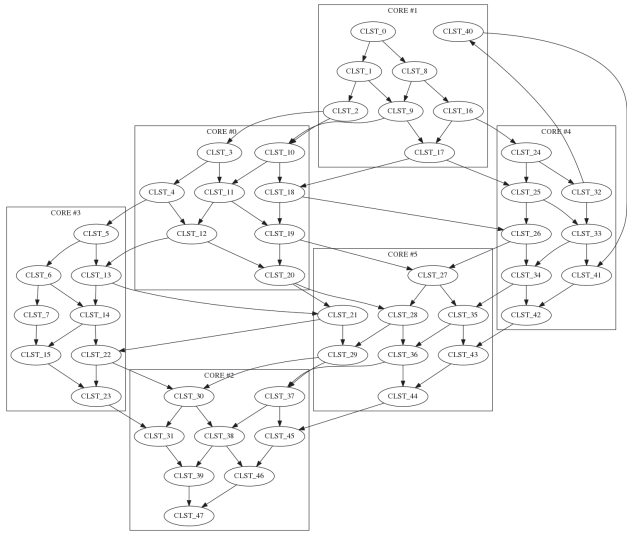

図 9 6 行*8 列 Mesh グラフ（*conn_weight* = 1）に対する提案手法の 6 コア割当て結果の図示

Fig. 9 Proposed cluster-level core assignment on 6*8 mesh graph (*conn_weight* = 1) for 6 cores.

5.6 モデル予測制御モデルを用いる並列性評価

実際の制御モデルに対する並列化性能を評価するため，モデル予測制御モデルに対し，コア数を 2 コアから 4 コアまで変化させ，提案手法，単階層のグラフ分割法 khmetis [7]，単階層のクリティカルパス優先法 [8] を用いてコア割当てを実施した．単階層のグラフ分割法は，提案手法において 1 次クラスタレベルでグラフ分割手法によりコア割当てを行う手法と同等であり，2 次クラスタに対して MIP を用いて 2 次コア割当てを実施する提案手法の効果の評価としても考えることができる．

なお，khmetis では内部で乱数を利用しているため，それぞれの実験を 10 回ずつ実施し，結果を平均している．クリティカルパス優先法は，最早実行時間を陽に目的関数とした手法であるが，ブロック数が多い場合探索木構造が大きくなり，探索段数に対して指数関数的に処理時間が増大するため，他の手法と同程度の処理時間になるよう探索段数を制限している．

評価指標として割当て結果の並列性能向上と分散度を用いた．それぞれの結果を**図 10** と**図 11** に示す．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て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，コア間通信オーバヘッドは，組込み制御向けマイコンにおいて共有メモリを用いた軽量通信を用いることを想定し，15 サイクルとしている [16]．分散度は，上記逐次実行時間と各コアに割り当てられたブロック処理量の総和 *core_cpu_util* の最大値との比である．

並列性能向上結果から提案手法の有効性を示すことがで

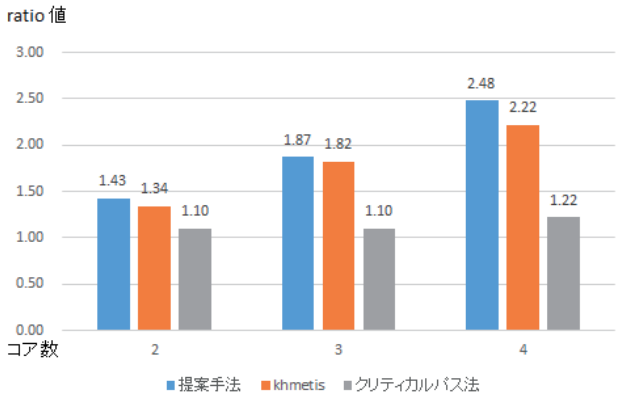

図 10 モデル予測制御モデルに対する並列性能向上

Fig. 10 Parallel speed-up ratio on model predictive control model.

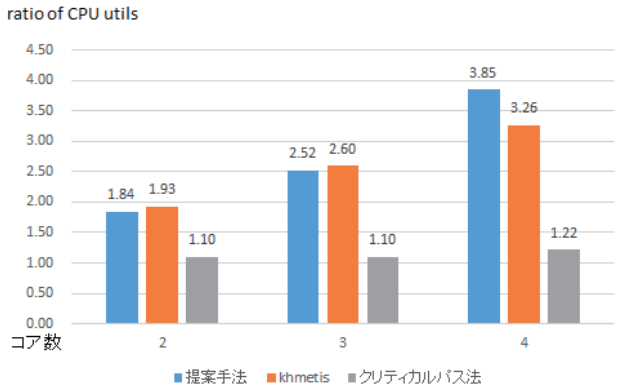

図 11 モデル予測制御モデルに対する分散度

Fig. 11 Load balancing on model predictive control model.

きた．提案手法とグラフ分割法（khmetis）との比較においては，分散度はグラフ分割手法の方が高いが，並列性能向上は提案手法の方が高くなる場合がある．1 次クラスタレベルでコア割当てするのみでなく，2 次クラスタ化を行い厳密解法を用いる提案手法の有効性を示すことができたと考えている．

クリティカルパス優先法に関しては，探索段数を制限したことにより，フィードバックループ構造を発見できず，効率的な並列化を実現できなかったことが原因として考えられる．探索段は個々の Simulink ブロックに対応することになり，少なくともフィードバックループの最初と最後は同一ブロックであり，同一コアに配置されるため，これを考慮した並列化が重要となる．しかしながらフィードバックループ内のブロック数が多くなると，探索段数が増加するため，探索段数の制限により考慮が難しくなる．組込み制御のモデルレベル並列化ではフィードバック構造が多くなるため，提案手法の方が有利であると考えられる．

6. おわりに

本論文は，MATLAB/Simulink のようなツールを用いて設計されるモデルベース開発における制御モデルをマルチコア向けに並列化するため，2 重階層クラスタリング法，特に混合整数線形計画法（MIP）を用いるコア割当て手法を提案した．

本手法では，制御設計の特性を利用したクラスタ化により，最上位では数十個のクラスタとし，各クラスタのコア割当てを厳密解法である MIP により求める．個々のクラスタは多数のブロックから構成される可能性があるが，高速なグラフ最適化手法である，階層クラスタリング法によって局所最適解にできるだけ陥らないようにしつつ，コア割当てを行う．この 2 重のクラスタ構造，グラフ最適化手法と MIP との組合せにより，組込み制御システムのマルチコア向けモデルレベル並列化を効率良く実現できる．なお，制御設計の特性により，ブロック数が数万，数十万になったとしても最上位クラスタ数は数十のレベルでとどめることができるといわれている [9]．各クラスタ内の 1 次コア割当て処理は，10 K ブロックで 10 秒，24 K ブロックで 18 秒の結果を得ている．

評価実験の結果，従来手法と比べ，提案手法はクラスタレベルのコア間通信コストを減少させることができることにより，性能向上が期待できると分かった．また，MIP 定式化のパラメー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．パラメー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ータを自動的に算出することが今後の課題である．

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りも今後の課題の 1 つである．これについては，The Multicore Association のハードウェア抽象化記述である SHIM [15] の利用が考えられる．

また，提案手法では前処理としてクラスタ化を行う．クラスタサイズのばらつきが大きいとき，MIP の定式化において解が存在しない可能性があるが，前処理によるクラスタ分割を定式化の一部にするか，解が必ず存在するようなクラスタリング手法を開発することも課題である．

最後に，MIP ソルバとして今回はオープンソースソフトウェアである GLPK を利用した．2 次クラスタ数を調整することにより，GLPK で十分な性能を達成できると考えている．しかしながら，2 次クラスタ数を増加させ，2 次クラスタレベルでの解の性能を向上させる手法も考えられる．このような場合，大規模問題に対してはソルバ実行時間が課題になる可能性があるが，この課題に対しては，CPLEX [4] などの商用ソフトウェアを導入することで解決できると考えている．

謝辞 本研究は科学研究費補助金 16H02800 の助成を受けている．いつも議論いただく共同研究企業の皆様，組込

みマルチコアコンソーシアムの皆様に深く感謝します．

参考文献

[1] 油谷 創ほか：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] Edahiro, M. et al.: New placement and global routing algorithms for standard cell layouts, *DAC1990*, pp.642–645 (1990).

[3] Hottger, R. 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] IBM: *IBM ILOG CPLEX V12. 1: User's manual for CPLEX*, International Business Machines Corporation (2009).

[5] 上土井陽子ほか：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システム LSI 設計技術（SLDM），1998-SLDM-089, pp.113–120 (1998).

[6] Karypis, G. et al.: hMETIS 1.5: A hypergraph partitioning package, Technical Report, Department of Computer Science, University of Minnesota (1998) available from ⟨http://www.cs.umn.edu/metis⟩.

[7] Karypis, G. et al.: Multilevel k-way partitioning scheme for irregular graphs, *Journal of Parallel & Distributed Computing*, Vol.48, No.1, pp.96–129 (1998).

[8] Kasahara, H. et al.: Practical multiprocessor scheduling algorithms for efficient parallel processing, *IEEE Trans. Comput.*, Vol.11, pp.1023–1029 (1984).

[9] 組込みマルチコアコンソーシアムにおける議論による (2015).

[10] Kumura, T. et al.: Model based parallelization from the Simulink models and their sequential C code, *SASIMI 2012*, pp.186–191 (2012).

[11] Makhorin, A. et al.: GLPK (GNU linear programming kit) (2008).

[12] Mathworks, available from ⟨https://jp.mathworks.com/products/embedded-coder.html⟩

[13] Mathworks, available from ⟨https://jp.mathworks.com/products/simulink.html⟩

[14] Meindl, B. et al.: Analysis of commercial and free and open source solvers for linear optimization problems, *Eurostat and Statistics Netherlands within the Project ESSnet on Common Tools and Harmonised Methodology for SDC in the ESS* (2012).

[15] The Multicore Association, available from ⟨http://www.multicore-association.org/workgroup/shim.php⟩

[16] Nakamura, R. et al.: Simple one-to-one architecture for parallel execution of embedded control systems, *CPSNA'14*, pp.25–30 (2014).

[17] Saad, Y. et al.: GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems, *SIAM Journal on Scientific and Statistical Computing*, Vol.7, No.3, pp.856–869 (1986).

[18] Scilab, available from ⟨http://www.scilab.org/⟩

[19] Suzuki, Y. et al.: Parallel design of feedback control systems utilizing dead time for embedded multicore processors, *IEICE Trans. Electronics*, Vol.E99-C, No.4, pp.491–502 (2016).

[20] 梅田 弾ほか：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).

[21] 山口滉平ほか：Simulink モデルからのブロックレベル並列化，組込みシステムシンポジウム 2015 論文集，pp.123–124 (2015).

[22] Ying, Y. et al.: An ILP formulation for task mapping and scheduling on multi-core architectures, *DATE09*, pp.33–38 (2009).

[23] Zhuravlev, S.: Survey of Scheduling Techniques for Addressing Shared Resources in Multicore Processors, *ACM Computing Surveys*, Vol.45, Issue 1, Article No.4 (2012).

鍾 兆前 （学生会員）

2012 年大連理工大学ソフトウェア学院卒業，2016 年名古屋大学大学院情報科学研究科情報システム学専攻博士課程前期課程修了．現在，名古屋大学大学院情報科学研究科情報システム学専攻博士課程後期課程在学．

枝廣 正人 （正会員）

名古屋大学大学院情報学研究科情報システム学専攻教授．1985 年東京大学大学院工学系研究科計数工学専門課程修士課程修了．同年 NEC 入社．1993 年プリンストン大学修士，1999 年同大学 Ph.D.（Computer Science）．グラフ・ネットワークアルゴリズム，マルチ・メニーコア向けソフトウェアの研究に従事．IEEE，電子情報通信学会，日本 OR 学会各会員．本会シニア会員．