情報処理学会論文誌 Vol.56 No.8 1568–1581 (Aug. 2015)

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

油谷 創[1] 枝廣 正人[1,a)]

受付日 2014年11月21日, 採録日 2015年5月9日

概要：本論文では，階層型メニーコアアーキテクチャに対し，階層構造を考慮したタスクマッピング手法を提案し，既存手法との比較を行う．提案手法は，LSI 配置向けに提案された階層クラスタリング法をもとにしているが，タスクマッピング問題に適用するために，タスク間通信量や，アーキテクチャ内の配線の使用頻度の偏り等を考慮している．また，提案手法では段階的にクラスタリングとデクラスタリングを繰り返す．これにより，アルゴリズム内で形を変えていくタスクグラフに柔軟に対応できる．提案手法を従来手法と比較した結果，通信コストを平均で 36%～14%削減した．また，アプリケーション完了時間では，従来手法を平均で 32%～8%削減し，通信コストの最小化によってアプリケーション自体の実行時間が削減できることを示した．

キーワード：マルチコア，メニーコア，タスクマッピング，階層構造，クラスタリング

Task Mapping Method for Hierarchical Many-core Processor Architectures

Sou Aburadani[1] Masato Edahiro[1,a)]

Received: November 21, 2014, Accepted: May 9, 2015

Abstract: Task mapping method is proposed for many-core processor architectures with hierarchical structure. Our method is based on a hierarchical clustering algorithm proposed for LSI placement, and tries to minimize latency of inter-task communication and to balance usage of routing networks among processors. Furthermore, the proposed method repeats clustering and declustering, so that the algorithm dynamically adapts task graphs which change shapes during task mapping. Compared with existing algorithms, our method achieves communication cost reduction of 14%–36% on average, which improves application execution time of 8%–32% on average.

Keywords: multicore, many-core, task mapping, hierarchical structure, clustering

1. はじめに

近年，半導体技術の進展により 1 つの LSI 上に複数のプロセッサコアが搭載されたマルチコアや，数十，数百のプロセッサコアが搭載されたメニーコアが広く使われている．マルチコア，メニーコア化の流れが起こった理由としては，シングルコアの性能向上が限界に到達したこと，処理性能の向上のために周波数を上げることで消費電力量，発熱量の増加という問題が発生したことがあげられる．メニーコアは，単体プロセッサの動作周波数向上による方法と比べ，低消費電力で性能向上を図れるため，将来の主流になるといわれている [18]．汎用用途向けのプロセッサについてもメニーコアの時代になるといわれており，Intel 社 [19]，Tilera 社 [20] 等が発表している．携帯電話や自動車等のプロセッサは組込みプロセッサと呼ばれ，汎用用途向けのプロセッサと比較すると，電力や発熱量等の制約が大きくなる [16]．したがって，今後組込みプロセッサもメニーコアの時代になるといわれている．

[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 464–8601, Japan
[a)] eda@ertl.jp

しかし，実際にメニーコアの性能を享受するためには，ソフトウェアに含まれる 1 つの機能単位であるタスクをプロセッサコアに割りつけていく必要がある．これをタスクマッピングと呼ぶ．特に組込みシステムは，製品が一度市場に出ると数年から数十年使用されるため，処理時間を長くかけてでも，良いマッピング結果を求めることが重要である．現在発表されている汎用用途向けメニーコアはタイル型アーキテクチャ [20] が多く，これに対するタスクマッピング手法が数多く提案されている [1], [7], [9], [12], [14].

従来手法には，大きく分けて貪欲法 [1], [9] とグループ化法 [7], [12], [14] がある．貪欲法は優先度に応じてタスクを 1 つずつ配置していく方法であり，高速であるが，一般的にアーキテクチャ全体のマッピング結果を考慮してマッピングを行うということをしないため，良いマッピング結果を得にくい．グループ化法はタスクをグループ化し，グループの位置関係を調整することで，貪欲法よりも良いマッピング結果を得やすいものとなる．しかしながら，グループ化法も局所最適解に陥りやすいという問題がある．

このような局所最適解に陥りやすい組合せ最適化問題に対し，LSI 回路設計において，回路を LSI 上に小面積，高性能に配置する手法として，階層クラスタリング法が提案されている [5].

この手法は，階層的にクラスタリングを行い，階層を崩しながら配置することにより，局所最適解から逃れやすいという特徴を持つ．階層クラスタリング法は，LSI 上で通信を行う構成要素どうしを近くに配置するというものであり，タスクマッピングと基本は同じであるが，タスク配置に適用するためにはタスクの実行順序，タスク間の通信量等を考慮する必要がある．

また，さらにプロセッサコアが増えたときには階層構造化し階層型メニーコアになると考えられており，階層構造を考慮し，プロセッサコアにマッピングしていく必要がある．

本論文ではそのような制約に沿い，階層クラスタリング法をタスクマッピング向けに拡張したマッピング手法を提案し，貪欲法，グループ化法，元の階層クラスタリング法との比較実験を行った．タスク生成ツールを用いた 16 から 1,024 タスクのデータ，および実際のモータ制御データから取得したタスクグラフを用いて評価した．生成グラフにおいては，通信コストを 36%～14%削減し，アプリケーション完了時間では，32%～8%の削減値を示した．モータ制御においては，通信コストを平均で 15%，アプリケーション間完了時間を平均で 8%改善し，階層を有効利用する解を生成した．

本論文の構成は以下のとおりである．まず，2 章でメニーコアアーキテクチャ，さらには階層型メニーコアアーキテクチャについて説明し，3 章においてタスクマッピングについて述べる．4 章では，提案手法について説明し，5 章で評価，6 章で結論と今後の課題について述べる．

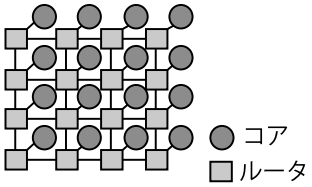

図 1 メッシュトポロジ
Fig. 1 Mesh topology.

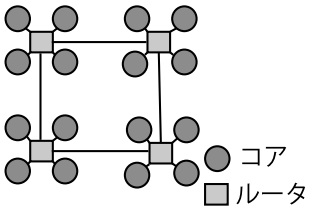

図 2 集中メッシュトポロジ
Fig. 2 Concentrated mesh topology.

2. メニーコアアーキテクチャとタスクグラフ

2.1 トポロジ

並列計算機の構成要素を接続するネットワークの形をトポロジと呼ぶ．理想的なトポロジは各コアがすべてのコアとつながり，それぞれのコアがすべてのコアに対し，直接通信できる完全結合であるが，大規模なシステムでは配線コスト，面積面から困難である．性能面と実装面のトレードオフの議論から多数のトポロジが提案されている [2], [11], [13], [15].

トポロジは隣接デバイスに接続するリンク数である次数，リンクの太さであるチャネル幅，経由するルータの数であるホップ数で特徴づけられる．次数，チャネル幅が大きければ消費電力が大きくなり，通信遅延が小さくなる．逆に次数，チャネル幅が小さければ消費電力は小さくなるが，通信遅延は大きくなるというトレードオフが存在する．しかし，ホップ数に関しては，一般的に小さければ小さいほど良い．現在最も広く使用されているトポロジはメッシュである [11], [13], [15]（**図 1**）.

しかし，メッシュトポロジは大規模なネットワークにおいて性能，電力効率が落ちてしまうという欠点がある．それを防ぐために提案されたものが集中メッシュである [2]（**図 2**）.

また，このようにトポロジ中に階層構造を持つものを階層型メニーコアアーキテクチャと呼ぶ．階層型メニーコアアーキテクチャには様々な種類があるが，本論文では典型的なトポロジとして集中メッシュについて取り上げる[*1]．集中メッシュは 1 つのルータに複数のコアを接続することでスケーラビリティを高めたものである[*2]．集中メッシュ

[*1] 提案手法は集中メッシュに限定されるものではない．
[*2] ここでいうスケーラビリティとは，大規模なシステムになっても

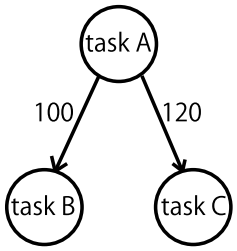

図 3 タスクグラフの性質
Fig. 3 Dependency in task graph.

において 1 つのルータに接続するコアの数を集中度 c と呼ぶ．図 2 では $c = 4$ である．集中メッシュでは，その少ないルータ数のためにルータ間リンクのチャネル幅を大きく設計しており，ルータ間通信の消費電力だけを見れば，メッシュよりも大きい．しかし，ホップ数削減による消費電力削減値の大きさがこの消費電力の増加量を上回っており，結果として，メッシュに対し，大幅に通信レイテンシ，消費電力を削減することを可能にしている [2]．加えて c 個の隣接コアと，低レイテンシ，低消費電力のローカル通信を行うことができ，データ通信の局所性を利用することによってさらなる性能向上，消費電力削減を可能としている．

2.2 タスクグラフ

タスクとは，プログラムを機能単位で分割したものである．つまり，プログラムはタスクの集合であるといえる．また，タスク間には**図 3** にあるように，A の実行が完了してから B と C が実行可能になるというような先行制約が存在する場合が多い．このようなタスクの特徴をふまえ，タスクグラフは各タスクをノード，先行制約をエッジで表現する有向グラフで表される．また，タスク間に通信がある場合はエッジに通信量を重みとして付加するものとする．

3. タスクマッピング

3.1 概要

タスクマッピングとは，タスクグラフの各タスクを与えられたトポロジに，後述する通信コスト，またはアプリケーション完了時間を最小にすることを目的として割り当てることである．タスクマッピング問題は NP 困難問題として知られており，仮に線形計画法によって解こうとすると膨大な時間がかかることが示されている [14]．そのため，発見的手法を用いてタスクマッピング問題を解く手法が多く提案されている．4 章において本論文で提案するマッピング手法について紹介するが，その前に 3 章では本論文で実験結果の指標としたコスト関数等の定義，そして，タスクマッピングの従来手法と，LSI に対する構成要素のマッピング手法である，階層クラスタリング法について説明する．

性能，電力効率が落ちないという意味である．

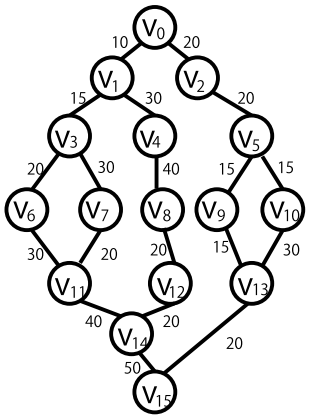

図 4 16 タスクからなるタスクグラフ
Fig. 4 Task graph with 16 tasks.

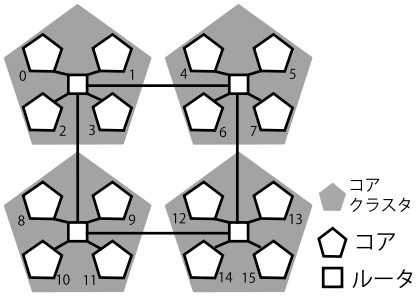

図 5 集中メッシュトポロジ
Fig. 5 Concentrated mesh topology graph with 16 cores.

3.2 通信コスト

マッピング結果の良し悪しを表すための指標として通信コストがある．通信コストはコスト関数によって計算される．コスト関数を用いるためにはいくつかの定義が必要となるため，以下でまとめて定義を行う．

3.2.1 定義

本論文ではタスクグラフを $G(V, E)$ と定義する．ここでそれぞれのノード $v_i \in V$ はタスクを表し，$e_{ij} \in E$ は 2 つのタスク v_i と v_j 間の制約関係を表す．また，v_i と v_j の間のデータ通信量は重み w_{ij} によって表される．**図 4** に，16 タスクからなるタスクグラフを示す．図 4 には 16 個のタスクがあるが，このタスクをマッピングするためには少なくとも 16 個のコアが必要であるとする．もしタスク数がコア数を超えている場合には，事前に通信コストや負荷分散等を考慮してタスクをマージしておく必要がある．

集中メッシュトポロジグラフを $M(P, L)$ で定義する．ここで $p_i \in P$ はトポロジのコアを表し，$l_{ij} \in L$ は p_i と p_j 間のリンクを表す．**図 5** は 16 個のコアを持ち，集中度 4 の集中メッシュである．

同一のルータに接続される 4 つのコアをコアクラスタと呼ぶことにする．同一コアクラスタ内の通信は高速，低消費電力で行うことができ，逆に他コアクラスタとの通信は低速，高消費電力となってしまう．

すべてのタスクを異なるコアに割り当てることから，マッ

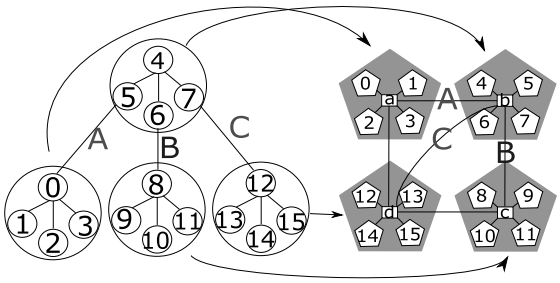

図 6 通信コストの算出
Fig. 6 Communication cost.

ピング関数 f: $V \to P$ において以下が成り立つ

$$\forall v_i \in V,\ \exists p_k \in P \quad f(v_i) = p_k \tag{1}$$

$$\forall v_i, v_j \in V,\ v_i \neq v_j \Rightarrow f(v_i) \neq f(v_j) \tag{2}$$

またマッピング結果の通信コストを CommCost で表す．

$$CommCost = \sum_{\forall e_{ij} \in E} N(f(v_i), f(v_j)) \times w_{ij} \tag{3}$$

ここで (3) 式の N はマッピングされたコア $f(v_i)$，$f(v_j)$ が属するコアクラスタ間のマンハッタン距離を表す．$CommCost$ を削減するためには直感的には通信量を最小限に抑え，通信を行うタスクをできる限りマンハッタン距離の小さいコアクラスタ間にマッピングすることが必要になる．ローカル通信は低レイテンシを実現する通信である一方，グローバル通信は大きい配線遅延を生む．本論文ではローカル通信の通信量を大きく，逆にグローバル通信の通信量を小さくすることを目的とし，簡単化のため，集中メッシュにおけるローカル通信のコストは考慮せず，グローバル通信のコストのみがコスト関数に影響を与えるとする．つまり，各通信の重みをアルファベットとして，マッピング結果が**図 6** のようであった場合，C の通信ではルータ b からルータ a を通り，ルータ d に到達するため，2 段階の通信が必要になり，このマッピングの通信コストは A + B + 2C となる．

3.3 アプリケーション完了時間

アプリケーション完了時間は，クラスタ内通信時間，クラスタ間通信時間にタスクの実行時間を足すことによって求められるマッピング後のタスクグラフの最長経路である．クラスタ間通信とクラスタ内通信を差別化するため，クラスタ間通信には定数 k をかけるものとする．**図 7** のタスクグラフが，簡略化された集中メッシュにマッピングされる場合について考える．図 7 右のようにマッピングされた場合，最長経路はタスク 1，2，4 かタスク 1，3 であるが，ここでは 1，2，4 が最長経路であると仮定する．すると，アプリケーション完了時間は，1 の実行時間 + 2 の実行時間 + 4 の実行時間 + A × k + B × 1 となる．

ここまでタスクグラフ，タスクマッピング，通信コスト，

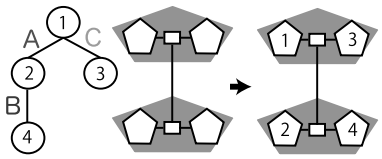

図 7 アプリケーション完了時間
Fig. 7 Application execution time.

アプリケーション完了時間について定義した．次節では既存マッピング手法について説明する．

3.4 従来のマッピング手法

ここでは，従来の代表的なタスクマッピング手法を紹介する．従来手法は，大きく分けて 2 種類ある．一方は貪欲法であり，優先度順にタスクを適切なコアに 1 つずつ割り当てる．他方はグループ化に基づいた方法で，多くの通信を行うタスクのグループと，トポロジ内で隣接するコアのグループを作成し，タスクグループを適切なコアグループに割り当てるものである．

3.4.1 貪欲法

代表的な方法として，NN Embed 法 [9] と，Topo-LB 法 [1] がある．

NN Embed 法は，枝の重みが大きいタスクのペアを選択し，両方のタスクがともに割当て済みでないときは一方をランダムにコアに配置，他方をそのコアから最も近い位置に配置するという手法である．

Topo-LB 法では，マッピングするタスク，コアの選出指標として，見積もり関数 f_{est}，$gain$ が用いられる．ここで，f_{est} は未割当てのタスク，コアの情報を入力にとり，あるタスク t があるコア p に割り当てられた際のコストを算出するものである．コスト $f_{est}(t,p)$ の値が大きい場合，タスク t をコア p にマッピングすると通信性能が悪化する．$gain(t)$ は，マッピングされた際に通信コストの悪化を招きやすいタスク t ほど大きい値を持つものである．タスクグラフの中から，$gain(t)$ が最大になるタスクを選出し，そのタスクを $f_{est}(t,p)$ の値を最も小さくするコア p に割り当てるという処理を繰り返すことによってマッピングを決定する．

3.4.2 グループ化法

グループ化法の代表的な手法として，Cluster-Based ILP 法 [14]，RMATT 法 [7]，MOPT 法 [12] がある．

Cluster-Based ILP 法は，グラフカット手法である Kernighan-Lin 法 [8]（以下 KL 法）によってタスクのグループ化を行い，対応するタスクグループとコアグループにおいて線形計画法が適用され，各タスクのコアへの配置を決定する．

タスクグラフとトポロジグラフが与えられると，タスクグラフは KL 法を用いて，トポロジグラフは均等に分割される．このとき，タスクグラフの分割面にはダミータスク

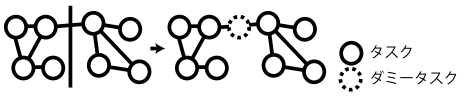

図 8 タスクグラフの分割
Fig. 8 Partition of task graph.

図 9 トポロジグラフの分割
Fig. 9 Partition of topology graph.

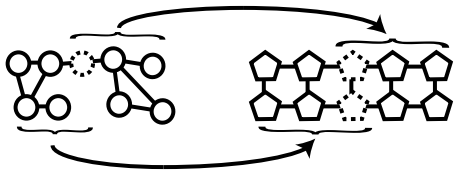

図 10 線形計画法によるマッピング
Fig. 10 Mapping with linear programming.

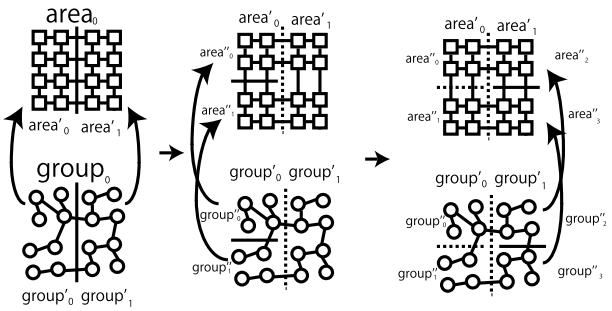

図 11 RMATT 法によるマッピング
Fig. 11 Mapping by RMATT method.

が，トポロジグラフの分割面にはダミーコアが挿入される（**図 8**，**図 9**）．各グラフの分割が終わると，**図 10** のように，各サブタスクグラフが，対応するサブトポロジグラフに線形計画法を用いてマッピングされる．このときダミータスクはダミーコアにマッピングされるものとする．このダミータスクとダミーコアによって，隣接するサブトポロジグラフとの通信を考慮したマッピング結果を得られることが特徴である．

RMATT 法 [7] はタスクグラフをグラフカットツールである METIS [6] によってグループ化し，対応するタスクグループとコアグループにおいて焼き鈍し法が適用され，各タスクのコアへの配置を決定する．**図 11** は 16 個のタスクを 16 コアメッシュトポロジに配置する処理の様子である．

タスクグラフ（$group_0$）とトポロジグラフ（$area_0$）が与えられると，タスクグラフは METIS を用いて，トポロジグラフは均等に分割される．METIS を用いた分割のあとには，焼き鈍し法によってより良い分割解，そしてマッピング解が探索される．この焼き鈍し法による探索は，以後，METIS による分割が行われるたびに実行されるものである．分割後の $group'_0$ は $area'_0$ に，$group'_1$ は $area'_1$ にマッピングされる．次に分割後の $group'_0$ が METIS によって，$area'_0$ が均等に分割され，同様にマッピングが行われる．最後に $group'_1$，$area'_1$ が分割され，$area''_2$ と $area''_3$，$group''_2$ と $group''_3$ の 2 通りの組合せにおいて，通信コストの良いグループの配置が選ばれ，マッピングが行われる．

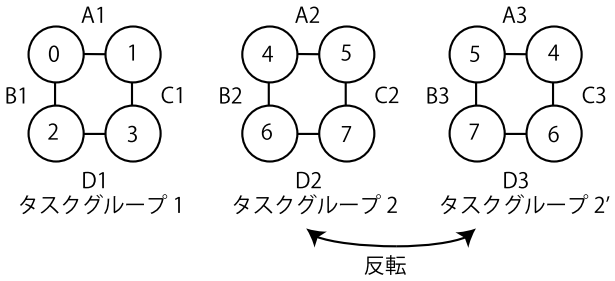

図 12 MOPT 法によるマッピング
Fig. 12 Mapping by MOPT method.

MOPT 法 [12] は，通信量の大きいタスクのペアを選択し，そのペアに対するすべての結合パターンの中で最良の結合を行うという処理を繰り返すことでグループ化，各タスクのコアへの配置を決定する手法である．

「すべての結合パターン」について述べる．ここでは，2×2 の 4 個のタスク含むタスクグループを結合する例を考える．**図 12** に 2 つのタスクグループ，タスクグループ 1 とタスクグループ 2 を示す．丸はタスクを表し，丸中の数字はタスクの ID を示す．タスクグループ 2′ はタスクグループ 2 の左右を反転させたものである．それぞれのタスクグループの辺には図 12 のようにアルファベットと数字の組合せによる ID が付けられる．タスクグループ 1 と 2 を回転して結合する辺の組合せは，「A1–A2，A1–B2，A1–C2，A1–D2, B1–A2, B1–B2, B1–C2, B1–D2, C1–A2, C1–B2, C1–C2，C1–D2，D1–A2，D1–B2，D1–C2，D1–D2」の 16 種類となる．しかし，このままでは反転を含めたすべての組合せが含まれないため，タスクグループ 2 を反転させたグループ 2′ を考える．このときの組合せは「A1–A3，A1–B3，A1–C3，A1–D3，B1–A3，B1–B3，B1–C3，B1–D3，C1–A3，C1–B3，C1–C3，C1–D3，D1–A3，D1–B3，D1–C3，D1–D3」の 16 種類である．タスクグループ 1 と 2 の結合の組合せは列挙したこれらの 32 種類となる．

3.5 階層クラスタリング法

3.5.1 概要

階層クラスタリング法は，枝廣らが提案した，LSI に対し高速で性能の良いレイアウト（構成要素のマッピング）を行うマッピング手法である [5]．LSI 向けのレイアウト手法には従来手法としてペア交換による Min-cut 法がある．この方法は，Cutline（構成要素を表すグラフを分割する線）を横切るエッジの数を最小とするようにノードを 1 つ

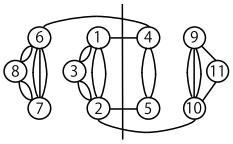

図 13 局所最適解
Fig. 13 Local optimum before clustering.

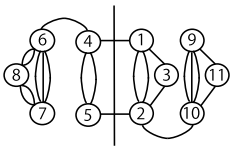

図 14 最適解
Fig. 14 Global optimum before clustering.

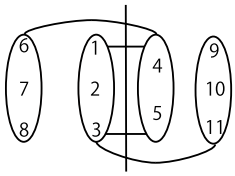

図 15 局所最適解
Fig. 15 Local optimum after clustering.

ずつ取り出して交換を行うという手法である．Cut-line をエッジが横切るということは，対応する配線の通信コストが高くなってしまうことを意味する．

図 13 のような場合，いったん解を悪くしてからでないと最適解（**図 14**）にたどり着くことができないが，1 つずつノードを交換するという手法の性質上，結果を悪くする交換は採用されず，これ以上の交換は行われない．このようにペア交換による Min-cut 法は高速であるが局所最適解に陥りやすいという欠点がある．

一方，階層クラスタリング法では，接続する要素をクラスタリングし，要素間に階層構造を構築してからマッピングを行うため，局所最適解に陥りにくいという特徴を持つ．以下が階層クラスタリング法の処理の流れである．

1. 接続度により，各クラスタがほぼ同じ大きさになるようにクラスタを形成．
2. 最終的にクラスタの数が 2 個になるまで階層的にクラスタを形成．
3. クラスタを崩しながら各階層でペア交換による Min-cut 法を行う．

この，構成要素をクラスタリングしてから交換を行うという特徴により，大きいクラスタを交換してから小さいクラスタが交換され，局所最適解に陥りにくいのである．**図 15** の状態でクラスタ単位での交換が行われ，最適解（**図 16**）にたどりつくことができている．

3.5.2 従来手法に対する優位性

ここではアルゴリズム動作の観点から，従来手法に対する階層クラスタリング法の本質的な優位性について述べる．貪欲法による手法では，通信量が大きいタスクどうしを近隣にマッピングすることを目指すが，階層クラスタリングを含めたグループ化を行う手法では，通信量が大きいタスクどうしを近隣にマッピングするためにグループ化を行い，さらにグループ間の位置関係を調整する．このグループの位置関係の調整により，グループ化を行う手法は貪欲法を用いる手法よりも優れた解を得ることができる．

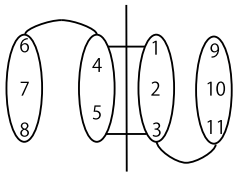

図 16 最適解
Fig. 16 Global optimum after clustering.

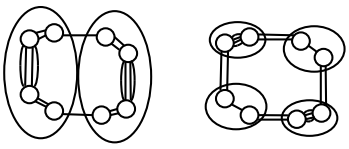

図 17 分割を用いる手法の問題点
Fig. 17 Problem in Partition-based method.

次に階層クラスタリング法と，グループ化を行う従来手法との比較を行う．グループ化を行う手法では，いったんグループ化されると，それを組み替えない．ところが，ターゲットアーキテクチャをすべて考慮してグループ化することは難しい．これに対して階層クラスタリング法は，配置を行いながらグループを崩して解を良くするため，より良い配置が得られる．本論文で提案する手法は，クラスタリングについても繰り返すことにより，さらなる解の向上を図っている．

グループ化する方法は，大きく分けて分割による方法とクラスタリングによる方法がある．Cluster-Based ILP 法と RMATT 法は分割による方法に属し，MOPT 法はクラスタリングによる方法に属する．以下では，それぞれに対して具体例を用いて比較する．

分割による方法は，min-cut を再帰的に用いることが一般的であるが，2 次元以上の配置問題として考えた場合，最初の分割が良すぎると，後段の分割に影響を及ぼすという問題がある．たとえば**図 17** において 8 タスクを 4 クラスタに配置する問題について考える．2 分割の最適解はグループ間接続が 2 本の左の分割であるが，はじめに最適な分割をしてしまうと配置問題としての最適解（図 17 右）は得られない．階層クラスタリング法では，たとえ最初に左のようにグループ化されたとしても，分割していく段階で最適解に到達する．

グループ化にクラスタリングを用いる方法は，はじめの段階のクラスタリングが，後段に影響を及ぼすという問題がある．**図 18** のようなタスクグラフの場合，クラスタリ

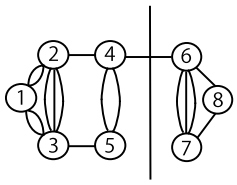

図 18 対象タスクグラフ
Fig. 18 Example.

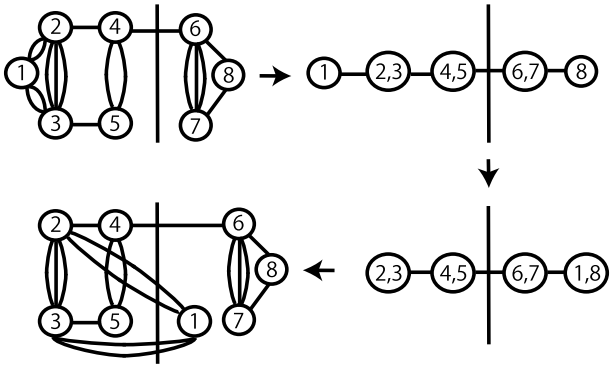

図 19 クラスタリングを用いる手法のマッピング
Fig. 19 Mapping by clustering method.

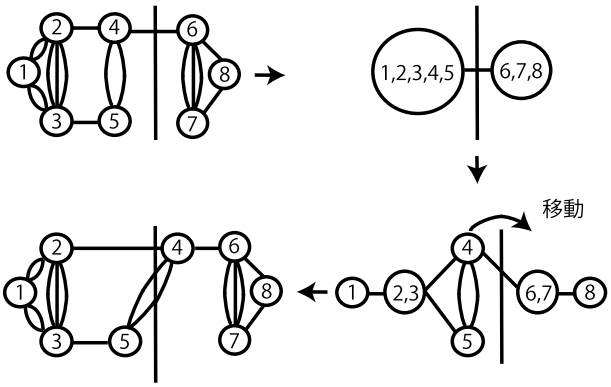

図 20 階層クラスタリング法によるマッピング
Fig. 20 Mapping by hierarchical clustering method.

ングを用いる方法では図 **19** の矢印のように結合が進み，両端のタスクどうしが結合され，最終的に分割線を横切る通信の数は 5 となる．階層クラスタリング法では図 **20** のようにタスクの結合が進んでいき，最終的に 2 つのタスクグループとなるが，その後順に結合を解いていき，各階層でタスクグループの交換を行い，最も良い分割を探索する．この結合の解除とタスクグループの交換はすべての結合が解除されるまで続く．その結果，図 20 のように分割線を横切る通信の数を 3 とする分割を得ることができる．

4. 提案手法

4.1 概要

LSI レイアウト向け配置問題とタスク配置問題は，接続する要素を近くに配置するという意味で基本は同じである．そこでこの階層クラスタリング法をメニーコア，さらには階層型メニーコア向けに発展させることで，高性能な配置を導くことができる発見的手法とすることが本研究の貢献である．

提案手法は 3 つの部分からなる．これは問題の分割が，問題の簡単化，性能向上につながるためである．

クラスタリング：タスクグラフをクラスタ化し，階層構造を構成
分割：クラスタを均一に分割
配置：分割結果の位置関係を決定

タスクマッピングの既存手法では「配置」のみ，もしくは「分割」と「配置」によってマッピング問題を解いているが，提案手法では前述の階層クラスタリング法の特徴を反映し，クラスタリングステージを追加した．階層クラスタリング法 [5] との違いは，クラスタリング，分割，配置を何度も繰り返す点である．分割，配置が行われるとタスクグラフは 2 つに分割され，各サブタスクグラフを 1 つのまったく新しいタスクグラフと見なして処理は進む．逐一クラスタリングを行うことによって，アルゴリズム内で形を変えていくタスクグラフに柔軟に対応することができるのである．

4.2 クラスタリング

階層クラスタリング法では，構成要素を接続するエッジ数に関してクラスタリングが行われていた．階層クラスタリング法をタスクマッピング手法に拡張するため，エッジ数ではなく，タスク間の通信量に関してクラスタリングを行うものとした．通信量の多いエッジを持つ 2 つのタスクを次々に結合していく．結合が進むたびにタスクの数は 1 つずつ減り，最終的には 2 つのクラスタとなる．クラスタリングを行う目的としては，分割において局所最適解に陥りにくい，通信量の大きいエッジは早期にクラスタリングされるため，大きな通信が同一コアクラスタ内にマッピングされやすい等の利点がある．

4.3 分割

分割のステージではタスククラスタ内に分割線を挿入し，分割線を横切る通信を考慮してタスククラスタを分割する．この分割では，通信コストの削減を狙うほか，全体的なコアクラスタ間の通信を減らし，通信の分散を狙う目的もある．

分割ではグラフカット手法である KL 法に変更を加えたものを使用する．本論文内では均等分割 KL 法と呼ぶものとする．均等分割 KL 法内では分割線を挟んだタスククラスタを 1 つずつ交換することにより分割を決定する．アルゴリズム中で使用されている分割コストは，クラスタ間通信が，分割線をまたぐ場合には，またがない場合の 2 倍のコストとするが（図 **21**），未分割のクラスタについては分割線をまたがない場合として扱う（図 **22** の b）．分割は以

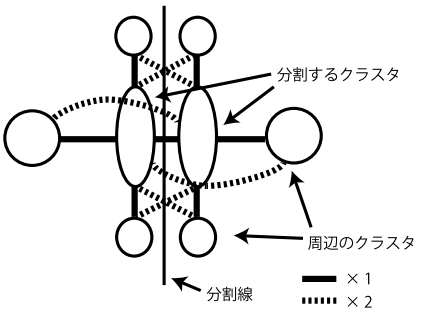

図 21 分割時に導入するモデル
Fig. 21 Graph model at partitioning.

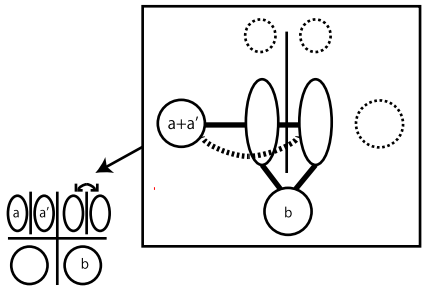

図 22 モデルの適用例 1
Fig. 22 Model example 1.

下のアルゴリズムに沿って計算が行われる．

1. モデルの各リンク（実線，破線）を通る通信量を求める．存在しない場合は 0 とする．
2. 破線のリンクは通信量を 2 倍にする．
3. すべての通信量を足し，さらに分割されたクラスタが内包するタスク数の差に，十分大きな数をかけたものを加算する．

クラスタ間のタスク数の差に十分大きな数をかけるのは，タスク数を同数に分割することを分割決定の絶対条件にするためである．以下が均等分割 KL 法のアルゴリズムである．

1. 初期分割により，クラスタを集合 S，T に分割する．このときの分割コストを記録．
2. 集合 S を集合 X，集合 T を集合 Y にコピー．
3. 集合 X と集合 Y がともに空でない限り，$a \to X$ と $b \to Y$ から交換した場合に最もコストが良くなる (a, b) を選んで交換を行う．a または b が空の場合もある．
4. その際のコストを記録し，交換の対象から a，b を取り除く．X と Y のすべてのタスクが交換対象でなくなるまで 3，4 を繰り返す．
5. 3，4 のループ中で最も良かったコストが，ループ前よりも良かった場合，S，T の集合を X，Y で更新し 2 に戻る．
6. コストがループ前よりも良くならなかった場合，集合 S，T の内包するタスクの数が同数であれば分割の解として集合 S，T を返す．同数でなかった場合にはクラスタを一段階崩し，1 からやりなおす．

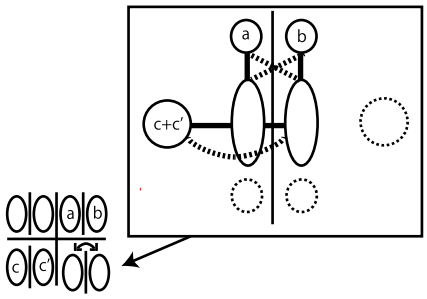

図 23 モデルの適用例 2
Fig. 23 Model example 2.

分割コストの定義により，周辺のクラスタと行われる通信を考慮してクラスタ分割を行うことができ，これが通信コストの削減につながる．

4.4 配置

配置のステージでは分割されたクラスタの位置関係を決定する．配置のステージで実現するのは通信コストの最小化と通信の分散の 2 点である．配置を決定する際にも分割時と同様のモデルを導入する．下に 2 つモデルの適用例を示す（図 22，**図 23**）．

配置は以下のアルゴリズムに沿って配置を決定する．

1. モデルの各リンク（実線，破線）を通る通信量を求める．存在しない場合は 0 とする．
2. 破線のリンクは通信量を 2 倍にする．
3. 分割されたクラスタの配置パターン 2 通りのうち，総通信量が少ない方を採用する．
4. 1〜3 をクラスタ分割が行われるたびに実行し，配置を決定する．

4.5 全体のアルゴリズム

クラスタリング，分割，配置を繰り返し，最終的なマッピングを決定する．アーキテクチャの形に応じて分割，配置のパラメタを調整することによって，今回対象としている集中メッシュトポロジのみだけでなく，様々なアーキテクチャに対応できる．以下が提案手法のアルゴリズムである．

1. （クラスタリング）1 つのタスクにつき 1 回結合できるものとして，通信の重みが大きいタスクを，結合できる箇所が存在する限り，順に結合．
2. （クラスタリング）1 つのクラスタを 1 つのタスクと見なす．1，2 を繰り返し，最終的に 2 つのクラスタにする．
3. （分割）均等分割 KL 法を用いて，内包タスク数が同数になるようにクラスタを分割．
4. （配置）分割後のクラスタを前述のクラスタ配置法によって配置．
5. 配置されたクラスタをデクラスタリングし，タスクグ

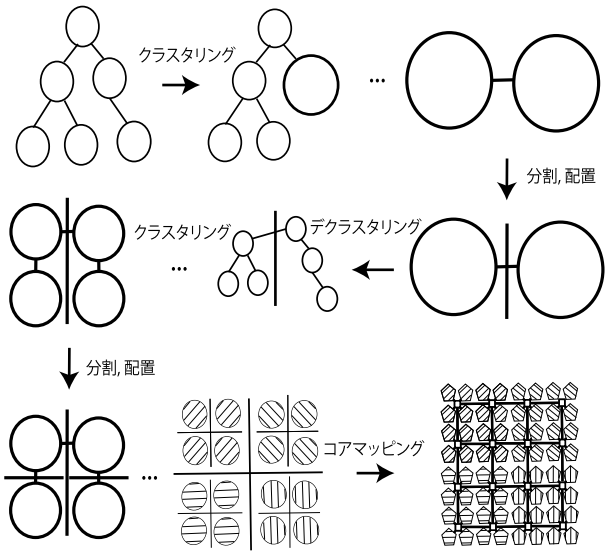

図 24 アルゴリズム全体のイメージ
Fig. 24 Flow of proposed algorithm.

ラフとする．

6. 分割されたそれぞれのクラスタにおいて，1 クラスタずつ 1〜5 を繰り返す．十分に分割され，配置が完了したら終了．
7. 集中メッシュにマッピング．

アルゴリズムは**図 24** のように進行する．図の簡単化のため，図 24 のタスクグラフに含まれるタスクの数を 6 としている．

4.6 タスクのマージ法

コア数よりもタスク数が多い場合，複数のタスクが 1 つのコア上で実行される．同一コア上で実行されるタスクを決定し，複数のタスクをまとめることをマージと呼ぶ．マージを考える際には，通信コストのほかにコア間の負荷分散を考慮する必要がある．ここでいう負荷とは同一コア上で実行されるタスクの総実行時間である．この負荷にコア間で偏りがあると，いくつかのコアの利用効率が下がり，実行時間の増加につながる．前述のクラスタリング法は通信コストのみを考慮しているが，そのほかにも，負荷分散のみ，または通信コストと負荷分散の両方を考慮した方法も考えられる．

4.6.1 マージ法 1：通信コストのみを考慮

このマージ法は前述したタスクのクラスタリング法をそのまま用いるものである．タスク数＝コア数となるまでクラスタリング法を適用する．

4.6.2 マージ法 2：負荷分散のみを考慮

負荷分散のみを考慮したマージ法では，**図 25** のように各タスクを実行時間が大きい順にソートし，上位タスクに下位タスクをマージすることによって負荷の分散を達成するものである．

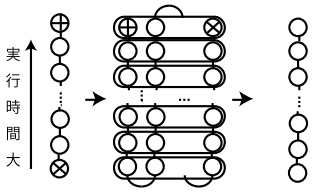

図 25 負荷の分散のみを考慮したマージ法
Fig. 25 Merge method considering load balance.

4.6.3 マージ法 3：通信コストと負荷分散の両方を考慮

このマージ法では，以下のアルゴリズムに従ってタスクのマージを行う．

1. 各タスクの実行時間の平均値をとり，平均値以上の実行時間を持つタスクを H，平均値以下のタスクを L とラベル付けする．
2. 最も通信量が大きかった 2 つのタスクが H と L の組合せであれば結合する．H と L でなかった場合は次に通信量が多かった組で同様に繰り返す．すべての結合候補で H と L の組合せがなかった場合には条件を L と L の結合として再走査．それでもなければ H と H に進む．
3. クラスタをタスクと見なす．
4. コア数とタスク数が同数となるまで繰返し．

上記のアルゴリズムにおいて，2 番目の結合候補が L と L になっている理由は，H と H で結合を行うと，そのクラスタの実行時間が大きくなりすぎるためである．

5. 実験

5.1 実験手法

貪欲法，グループ化法，階層クラスタリング法と，提案手法の比較，そして前述の 3 種のマージ法の比較を行う．貪欲法からは NN Embed 法，Topo-LB 法を選択し，グループ化法からは Cluster-Based ILP 法を選んだ．この 3 手法を選んだ理由であるが，まず NN Embed 法は，最も簡潔で Greedy なアルゴリズムであるためである．Topo-LB 法に関しては，NN Embed 法よりも評価関数が複雑であり，その差異を 1 つの指標とするために選択した．最後に，前述したグループ化手法の中で唯一線形計画法を用いて最適解を得ている Cluster-Based ILP 法を選んだ．ここで，これら 3 つの既存手法は階層を持つアーキテクチャを想定していないため，集中メッシュトポロジに対してのマッピング解を求められるよう，既存手法 3 種の一部を拡張した．

NN Embed 法：アーキテクチャ情報において，クラスタ内のコア間距離を 0 と設定

Topo-LB 法：アーキテクチャ情報において，クラスタ内のコア間距離を 0 と設定

Cluster-Based-ILP 法：ダミーコア挿入数変更（ダミーコアクラスタを挿入するようにする）アーキテクチャ情報において，クラスタ内のコア間距離を 0 と設定

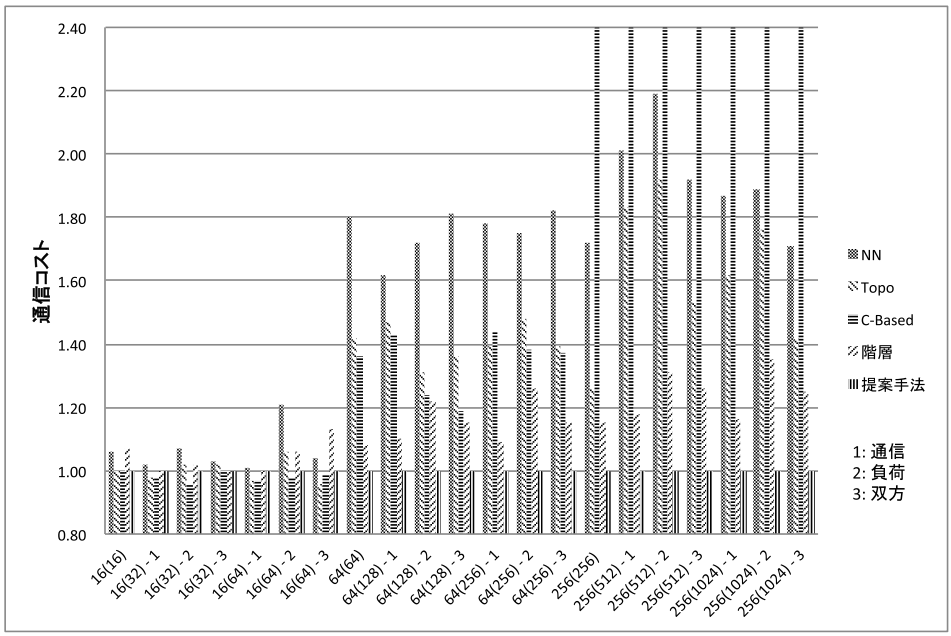

図 26 通信コスト
Fig. 26 Communication cost.

元の階層クラスタリング法を改良したものでは，クラスタリングのログを利用して内包タスク数に差を許してクラスタを分割し，分割後のクラスタを NN Embed 法で配置している．こちらもクラスタリング，分割，配置を繰返し行うことによってアルゴリズム内で形を変えていくタスクグラフに柔軟に対応できるようになっている．グラフ中では便宜上「階層」と表記する．

タスクグラフはグラフ生成ツールを用いたものとセンサレスモータ制御から生成されたタスクモデルを用いた．グラフ作成ツールは TGFF [3] を用い，16 タスクから 1,024 タスクのタスクグラフをランダムに各 10 種類作成した．各マッピング手法，各マージ手法の組合せによって集中メッシュトポロジへのマッピングを決定し，通信コスト，負荷の分散，実行時間，アプリケーション完了時間によって評価を行う．なお，Cluster-Based ILP 法内の線形計画法部分は，論文に習い，商用線形計画法ソルバである Xpress-MP [17] を用いた．

モータ制御については，センサレスモータ制御アルゴリズム [10] より生成したタスクグラフを用いた．

実験には以下の PC を使用した．

OS：Windows7 Professional 64 bit CPU：Intel Xeon W5590 3.33 GHz（2 ソケット）メモリ：24 GB

5.2 実験結果（通信コスト）

通信コストでの比較は図 **26** のようになった．このグラフは提案手法の通信コストを 1 とした場合の各手法の通信コストのグラフである．グラフの x 軸の値は「コア数（マージ前のタスク数）– マージ法の種類」である．マージ法の種類は 1 が通信コストのみを考慮したもの，2 が負荷分散のみを考慮したもの，3 が通信コストと負荷分散の両方を考慮したものである．なお，今回は実行時間に関して 8 時間の制限時間を課しており，実行時間が 8 時間以上となったものに関しては結果なしとしている．グラフでは C-Based ILP 法の 256 コアへのマッピングが結果なしとなっており，見やすさのためにグラフの値を最大値としている．

提案手法と NN Embed 法を比較した場合，NN Embed 法では，マッピング決定時に多くのランダム要素が入り，アーキテクチャ全体の情報を考慮していないのに対し，提案手法にはランダム性がなく，局所的なアーキテクチャ情報だけでなく，全体的なアーキテクチャの情報を考慮しているため，大きな差が生まれている．

Topo-LB 法は一番最初に配置されるコアの位置をもとに処理が進むが，その初期配置が Topo-LB 法の評価関数の仕様上，毎回のトポロジの中心付近になる．最適マッピング解を考える場合，最初に選ばれたタスクが配置されるべき場所が中心付近でなく，端に近い部分であった場合，その時点で最適解に到達することが不可能となるという欠点がある．一方，提案手法では 1 つのタスクをあるコアにマッピングし，そのコアを中心として処理が進むという性質はなく，全体のマッピング結果が決定されて初めてコアへのマッピングが行われる．つまり，各タスクはそれぞれふさわしいコアにマッピングされるのである．しかし，16 コアへのマッピングでは，提案手法を Topo-LB 法が上回っている点が見られる．これは，提案手法内で使用している KL 法では，分割決定の評価関数が通信コストよりも同数

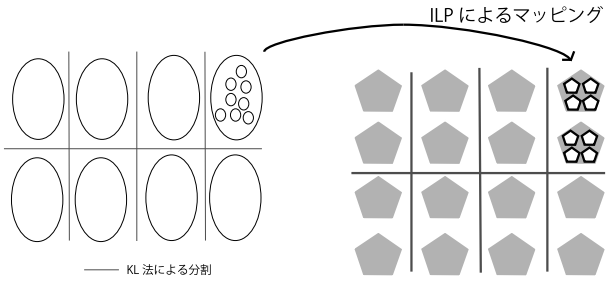

図 27 C-Base ILP によるマッピング
Fig. 27 Mapping by C-based ILP method.

での分割ということに重きを置いており，このことによるコストの悪化が，前述した Topo-LB 法の問題点によるコストの悪化を上回ってしまったためであると考えられる．

Cluster-Based ILP 法に関しては，線形計画法の特性上，一度に広い範囲のマッピング問題を解くことが不可能である．つまり，マッピング問題の対象が 12 タスクを超えると実用的な時間でマッピング問題を解けなくなる．そのため，今回の実験では，線形計画法で解く範囲を 8 タスク → 8 コアのマッピングとした．そのため，マッピング解の大部分がグラフカット手法である KL 法によって決定されており，その点がコストの悪化につながったと考えられる（**図 27**）．また，Cluster-Based ILP 法内で用いられている KL 法には，グループを崩しながらタスクの交換を行うという機能はないため，局所最適解に陥りやすいこともコストの悪化の一因になっていると考えられる．KL 法の介入が少ない 16 コアへのマッピングでは優れた通信コストを見せてはいるものの，線形計画法を用いてマッピング問題を解く手法は，大規模なアーキテクチャへのマッピングには不向きであると考えられる．

元の階層クラスタリングを改良したものでは，内包タスク数に差を許して分割を行い，分割後にクラスタ間でタスクを移動するというクラスタリング部の差，そして配置法として NN Embed 法を用いているという配置部の差がこの通信コストの差につながっている．

5.3 実験結果（マージ手法）

各マージ法と，提案マッピング手法の組合せによる負荷分散のグラフを**図 28** に示す．このグラフの x 軸の値は「コア数（マージ前のタスク数）」であり，y 軸は分散値である．この分散値が小さいほど付加の偏りが少ないといえる．

マージの種類と通信コストの関係のグラフを**図 29** に示す．このグラフでは，1（通信コストのみを考慮したマージ法）の通信コストを 1 とした場合の各マージ法の通信コストのグラフである．

2 つのグラフより，通信コストのみ，負荷分散のみ，通信コストと負荷分散の両方を考慮したマージ法は，それぞれ想定どおりの結果となっていることが分かる．

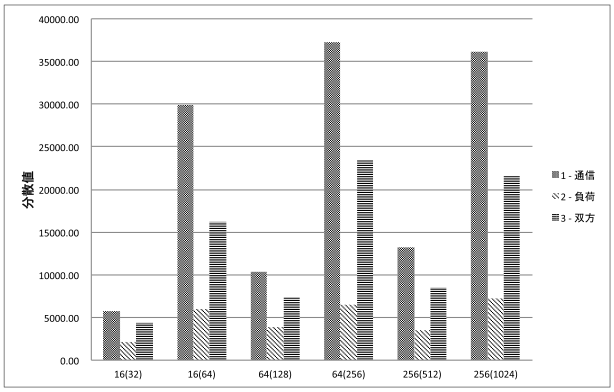

図 28 負荷分散
Fig. 28 Load balance.

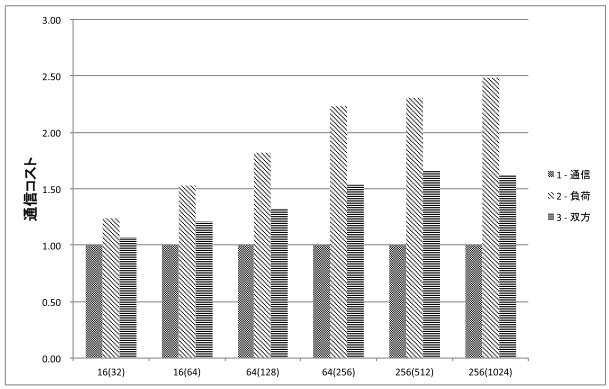

図 29 通信コスト
Fig. 29 Communication cost.

5.4 実験結果（実行時間）

各マッピング手法とマージ法の組合せによる実行時間のグラフを**図 30**，**図 31**，**図 32** に示す．図 30 の 64 コアへのマッピング部分を拡大したものが図 31，16 コアへのマッピング部分を拡大したものが図 32 である．図 30 の C-Based ILP による 256 コアへのマッピング部分は，実行に 8 時間以上かかったため値なしとしてグラフの値を最大値としている．NN Embed 法と Topo-LB 法のオーダはタスクグラフのエッジの数を $|E|$ とした場合，$O(|E|)$ であり，対象アーキテクチャのコア数を $|v|$ とした場合，KL 法 $(O(|v|^3))$ がコア数オーダで呼び出される提案手法と階層クラスタリング法は $O(|v|^4)$ となっている．この計算量については，アルゴリズム実装上の工夫で下げられると考えており，それは今後の課題である．

しかし，組込みアプリケーションにおいては，組込みプロセッサ上で同じアプリケーションが数年から数十年も作動し続けるということが起こる．そういった場合，優れたマッピング結果によって実行時間が低減できるのであれば，数時間程度の計算時間は許容範囲であると考えられる．元の階層クラスタリングを改良したものでは，タスク数の差を許してクラスタを分割するという性質上，通信コストは悪くなるが，実行時間が小さいという性質を示した．

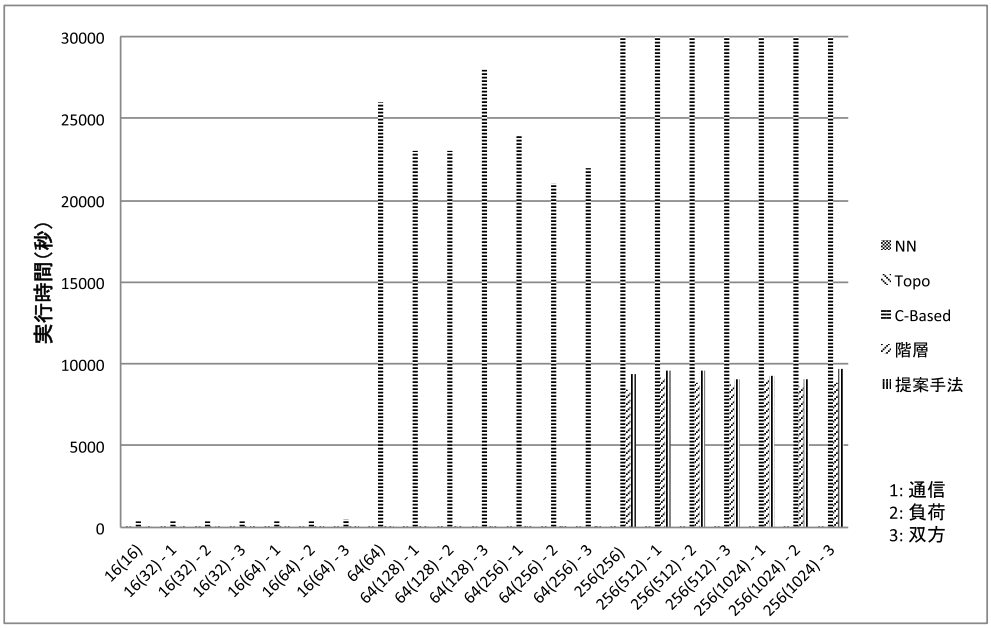

図 30 実行時間
Fig. 30 Computation time for mapping.

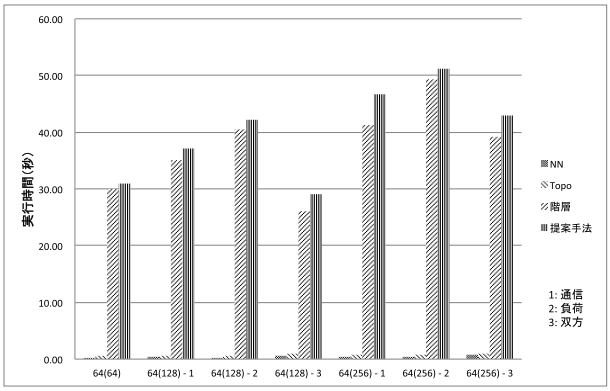

図 31 実行時間（64 コア）
Fig. 31 Computation time for mapping on 64 cores.

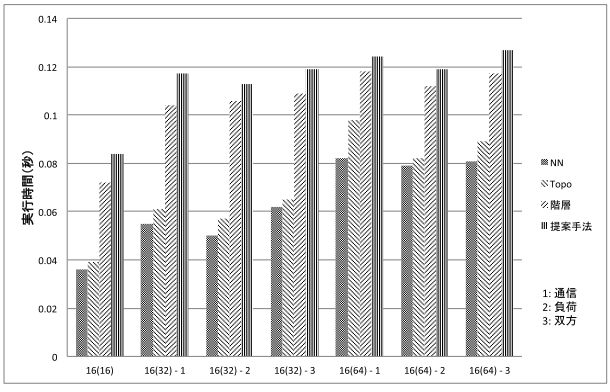

図 32 実行時間（16 コア）
Fig. 32 Computation time for mapping on 16 cores.

5.5 実験結果（アプリケーション完了時間）

アプリケーション完了時間での比較は図 **33** のようになった．なお，クラスタ間通信に掛ける定数は 10 とした．このグラフは提案手法の完了時間を 1 とした場合の各手法の完了時間のグラフである．グラフの x 軸の値は「コア数（マージ前のタスク数）– マージ法の種類」である．マージ法の種類は 1 が通信コストのみを考慮したもの，2 が負荷分散のみを考慮したもの，3 が通信コストと負荷分散の両方を考慮したものである．グラフでは C-Based ILP 法の 256 コアへのマッピングが結果なしとなっており，見やすさのためにグラフの値を最大値としている．

通信コストの比較と同様の傾向が見られることが分かり，通信コストの最小化によってアプリケーション自体の実行時間が削減できるということを示している．ただ，タスクグラフのクリティカルパスが通信量の小さいエッジを多く含む場合，アプリケーション完了時間が削減しにくいという欠点があり，クリティカルパスとなりうるタスク群を事前に調査し，補正をかけてマッピングする等の解決策が必要であると考える．

5.6 実験結果（モータ制御）

センサレスモータ制御から生成した 110 タスクのタスクグラフを用いて，通信コスト，アプリケーション完了時間，実行時間を指標として評価を行った．110 タスクを，通信コストのみを考慮したマージ方法によって 64 タスクにマージし，64 コア集中メッシュトポロジにマッピングした．図 **34** にその結果を示す．

通信コストとアプリケーション完了時間の比較では，提案手法の通信コストを 1 とした場合の各手法の割合を示しており，実行時間は各手法の実行時間（秒）である．ランダム生成したタスクグラフにおける結果と同様の傾向を示しているが，提案手法と他手法との差が全体的に縮まって

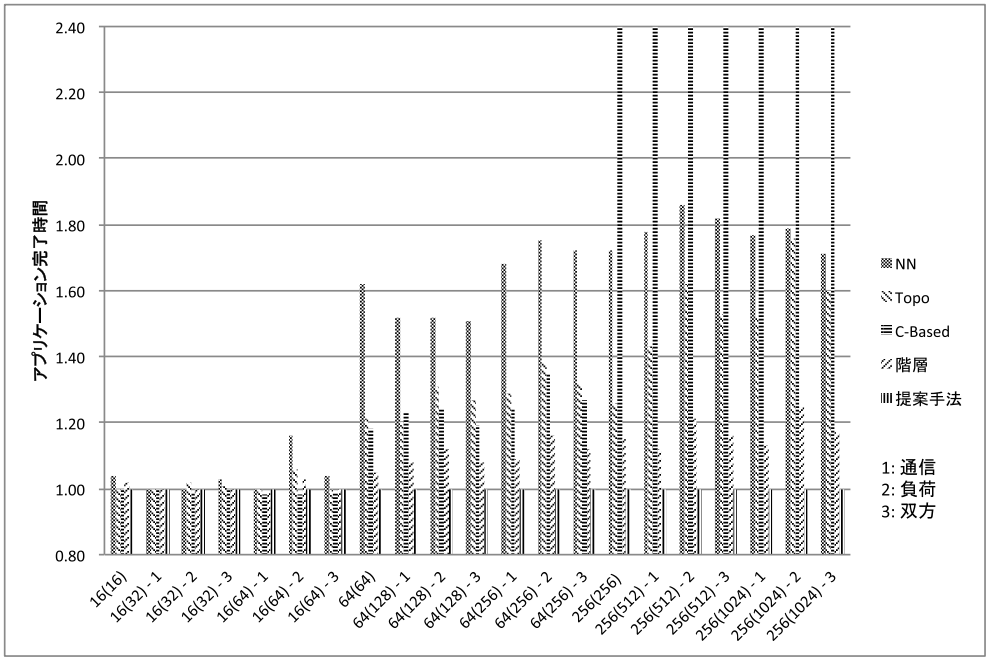

図 33 アプリケーション完了時間
Fig. 33 Application execution time.

通信コスト (提案手法を 1 に正規化)				
NN	Topo	C-Based	階層	提案手法
1.31	1.26	1.10	1.05	1

アプリケーション完了時間 (提案手法を 1 に正規化)				
NN	Topo	C-Based	階層	提案手法
1.21	1.16	1	1	1

実行時間 (秒)				
NN	Topo	C-Based	階層	提案手法
0.16	0.31	260000	39.02	42.24

図 34 モータ制御モデルにおける比較
Fig. 34 Comparison with motor control model.

いる．これは，センサレスモータ制御のタスクグラフの通信量の重みが小さく，差異が出にくいことが原因だと考えられる．およそ 40 秒の処理時間で，通信コストにおいて平均 15%，アプリケーション完了時間において平均 8%の削減値は，組込みタスクマッピングとしては優れているといえる．

6. 結論

本論文では，階層構造を持つアーキテクチャへのマッピング手法を提案し，比較実験を行った．

提案手法を，貪欲法を用いた手法や，グループ化法を用いた手法と，グラフ生成ツールを用いて作成したタスクグラフを用いて比較した結果，通信コストを平均で 36%〜14%削減した．

マージ手法の比較においては，提案マージ手法は通信コストを重視し負荷分散も考慮したマッピングであることが分かり，狙いどおりの結果を出すことができた．負荷分散についてさらに厳しい制約をマージ時に課せば，今回の結果とは別の，負荷分散に重きを置き，通信コストも考慮したマージ法ができるはずである．

実行時間に関しては，提案マッピング手法は大規模なアーキテクチャに対して通信コストの良いマッピングを出力するが，マッピングのための処理時間が長いという問題がある．階層クラスタリング法 [5] のアルゴリズムでは，数十万要素の LSI 配置に利用されており，実装上の工夫により，計算量を削減することは可能であると考えている．たとえば高速にマッピングを行うことができ，ある程度良いマッピング解が得られる Topo-LB 法を初期配置に用い，そこから提案手法によって配置を調整するといったことが考えられる．

アプリケーション完了時間の比較では，通信コストの最小化によってアプリケーション自体の実行時間が削減できるということを示した．

本論文内では，集中メッシュへのマッピングのみを行ったが，提案手法は階層を持たないメッシュやトーラス等のアーキテクチャへの適用も可能である．今後も多くのアーキテクチャは階層を持たないことが予想され，本手法により，階層を持たないアーキテクチャでも効率良いタスクマッピングを実現することが課題である．

謝辞 本研究は，科学研究費助成事業 24500058 の支援をいただいた．査読者の方々には適切なコメントをいただき，論文が大幅に改善した．深く感謝したい．

参考文献

[1] Agarwal, T. et al.: Topology-aware task mapping for reducing communication contention on large parallel machines, *20th International Parallel and Distributed Processing Symposium, IPDPS 2006*, IEEE (2006).

[2] Balfour, J. and Dally, W.J.: Design tradeoffs for tiled CMP on-chip networks, *Proc. 20th Annual International Conference on Supercomputing*, ACM (2006).

[3] Dick, R.P., Rhodes, D.L. and Wolf, W.: TGFF: Task graphs for free, *Proc. 6th International Workshop on Hardware/Software Codesign*, IEEE Computer Society (1998).

[4] de Dinechin, B.D. et al.: A Distributed Run-Time Environment for the Kalray MPPA-256 Integrated Manycore Processor, *Procedia Computer Science*, Vol.18, pp.1654–1663 (2013).

[5] Edahiro, M. and Yoshimura, T.: New placement and global routing algorithms for standard cell layouts, *Proc. 27th ACM/IEEE Design Automation Conference*, ACM (1991).

[6] Karypis, G. and Kumar, V.: A fast and high quality multilevel scheme for partitioning irregular graphs, *SIAM Journal on Scientic Computing*, Vol.20, No.1, pp.359–392 (1998).

[7] 今出広明ほか：大規模計算環境のためのランク配置最適化手法 RMATT，先進的計算基盤システムシンポジウム SACSIS 2011 論文集，pp.340–347 (2011).

[8] Kernighan, B.W. and Lin, S.: An efficient heuristic procedure for partitioning graphs, *Bell System Technical Journal*, Vol.49, No.2, pp.291–307 (1970).

[9] Lo, V.M. et al.: OREGAMI: Tools for mapping parallel computations to parallel architectures, *International Journal of Parallel Programming*, Vol.20, No.3, pp.237–270 (1991).

[10] Akimatsu, R. and Doki, S.: Improve torque response using the inverter overmodulation range in position sensorless control system of PMSM, *IECON 2013-39th Annual Conference of the IEEE Digital Object Identier*, Industrial Electronics Society (2013).

[11] Sankaralingam, K. et al.: Distributed microarchitectural protocols in the TRIPS prototype processor, *39th Annual IEEE/ACM International Symposium on Microarchitecture* (*MICRO-39*), IEEE (2006).

[12] 佐野伸太郎ほか：メッシュ/トーラス接続型スーパーコンピュータに適した高性能タスク配置手法，電子情報通信学会論文誌，Vol.J96-D, No.2, pp.269–279 (2013).

[13] Taylor, M.B. et al.: Evaluation of the Raw microprocessor: An exposed-wire-delay architecture for ILP and streams, *Proc. 31st Annual International Symposium on Computer Architecture*, IEEE (2004).

[14] Tosun, S.: Cluster-based application mapping method for Network-on-Chip, *Advances in Engineering Software*, Vol.42, No.10, pp.868–874 (2011).

[15] Vangal, S. et al.: An 80-tile 1.28 TFLOPS network-on-chip in 65 nm CMOS, *IEEE International Solid-State Circuits Conference Digest of Technical Papers* (*ISSCC 2007*), IEEE (2007).

[16] Wayne Wolf 著，安浦寛人，中西恒夫，北須賀輝明，久住憲嗣，室山真徳，田頭茂明訳：組み込みシステム設計の基礎，日経 BP 社 (2009).

[17] Dash Optimization: Xpress-mp (2001).

[18] Intel ニュースルーム，入手先 ⟨http://newsroom.intel.com/community/ja_jp/blog/2011/09/15/⟩.

[19] Intel® XeonPhi™ コプロセッサ 5110P，入手先 ⟨http://www.intel.co.jp/⟩.

[20] TheTILEPro™ processor, available from ⟨http://www.tilera.com/solutions/cloud computing⟩.

油谷 創

H&L Works 所属．2013 年名古屋大学工学部電気電子・情報工学科卒業，2015 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．

枝廣 正人 （正会員）

名古屋大学大学院情報科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攻修士課程修了．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．