Vol.2019-SLDM-189 No.30
Vol.2019-EMB52 No.30
2019/11/15

タスク間とスレッド間の通信を考慮した 可変並列度 Fork-Join タスクのスケジューリング

西川広記[1]　島田佳奈[1]　谷口一徹[2]　冨山宏之[1]

概要：本論文は，均質なマルチコア上における可変な並列度を有する Fork-Join タスクのスケジューリング手法を提案する．可変な並列度を有するタスクは，各タスクが複数のスレッドに分割されて並列に実行されることを許し，そのスレッド数はスケジューリングと同時に決定する．本研究では，DMA を用いたタスク間およびスレッド間の通信に加えて Fork および Join の際に発生するオーバーヘッドを考慮したスケジューリング手法を提案する．

キーワード：タスクスケジューリング，整数計画法，並列タスク，マルチコア

1. はじめに

近年，シングルコアにおける性能向上が鈍化しており，組込みシステムにおけるマルチコア/メニーコア技術のさらなる普及が期待されている．そのなかでも，マルチコアの効率的な利用に向け，マルチコアを用いてタスクの実行順序を決定するタスクスケジューリングの研究が一層進められている．文献[1]ではマルチコアを用いてシングルコア上のみで各タスクが実行されることを想定したスケジューリング手法が提案されている．しかしながら，現実世界のマルチメディア領域などにみられるアプリケーションでは，多くのタスクが並列タスクと呼ばれ，各タスクは複数のスレッドへ分割され並列な実行が許容されたタスクである．そこで，タスクを複数のスレッドへ分割することを想定した並列タスクのスケジューリング問題が近年盛んに研究されている [2-10].

並列タスクを想定したスケジューリング分野の研究領域において，並列タスクは大きく二つの種類に分けられる．一方は，各タスクの実行に用いられる並列度（スレッド数）がスケジューリングの事前に決定されている，固定並列度タスクである [3]．文献[3]では，リストスケジューリングに基づき並列タスクをスケジューリングする方法が提案されている．各タスクはそれぞれ固有の並列度を有しており，複数のコア上で各タスクのスレッドが同時に実行されるような方式を想定したスケジューリング問題を，整数計画法に帰着させて定式化し，全体のスケジュール長を最短化することを目的にしている．他方は，各タスクの実行に用いられる並列度がスケジューリングと同時に決定される，可変な並列度を有するタスクを想定したスケジューリングである [4-11]．文献[4]は，文献[3]とは異なり，可変な並列度を有するタスクを想定している．この研究ではデッドラインを満たしながらハードウェアコストの最小化を目指している．さらに文献[5-9]でも可変並列度タスクに関する研究が行われている．文献[6]はリアルタイムシステム向けの可変並列度タスクのスケジューリングが研究されている．文献[9]では，線形計画法で定式化された可変並列度タスクのスケジューリングに対し，制約プログラミングを用いて定式化されたスケジューリングを比較しており，スケジュール長(メイクスパン)の最短化を目的としている．

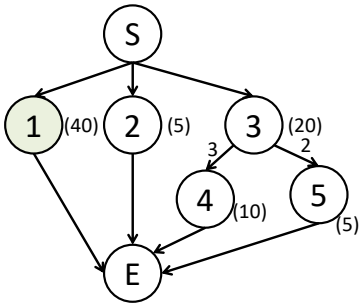
しかしながら，現実のシステムにおいては複数のタスクの間でデータ転送などによる通信が必要な場合がある．例えば，あるタスクを実行したコアから別のタスクを実行するコアに対してデータ転送を行う場合には通信が必要である．また，並列タスクにおいては独立に異なるコアで実行される複数のスレッド間に対してデータの通信が必要な場合がある．こうしたタスク間の通信と，タスク内のスレッド間の通信の両方を考慮したタスクスケジューリング問題は，我々の知る限りでは過去に多くは研究されていない．文献[10]では，タスク間の通信を考慮した並列タスクのスケジューリングを提案しているが，スレッド間の通信については考慮していなかった．加えて，この研究ではタスクにおける全てのスレッドが同時に異なるコア上で実行されることを想定していたが，文献[9]などにみられるように，現実世界の並列タスクの多くは Fork-Join 型であり，分割された複数のスレッドはそれぞれ独立にスケジュールされる．

本稿では，タスク間とスレッド間の通信を考慮した可変並列度 Fork-Join タスクのスケジューリングを提案する．提案するスケジューリング手法は整数計画法に基づいてスケジュール長を最短化することを目的とする．

本稿の構成は以下の通りである．まず第 2 章では，本研究において提案されるタスク間とスレッド間の通信時間を考慮した可変並列度 Fork-Join タスクのスケジューリング問題について，例題を用いながら説明する．そしてそのスケジューリング問題を整数計画法に基づいて定式化する．第 3 章では実験における環境や比較手法について述べ，さらにその結果について述べる．最後の第 4 章では，まとめと今後の課題を述べる．

1 立命館大学
Ritsumeikan University
2 大阪大学
Osaka University

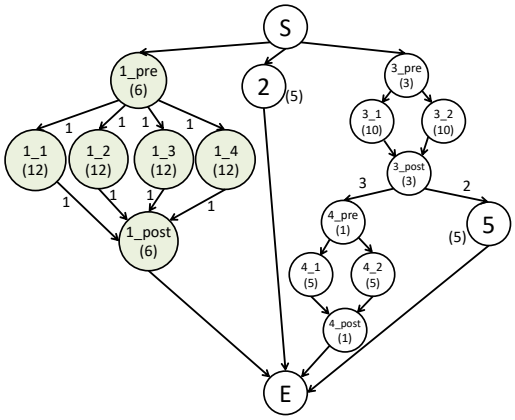
Vol.2019-SLDM-189 No.30
Vol.2019-EMB52 No.30
2019/11/15



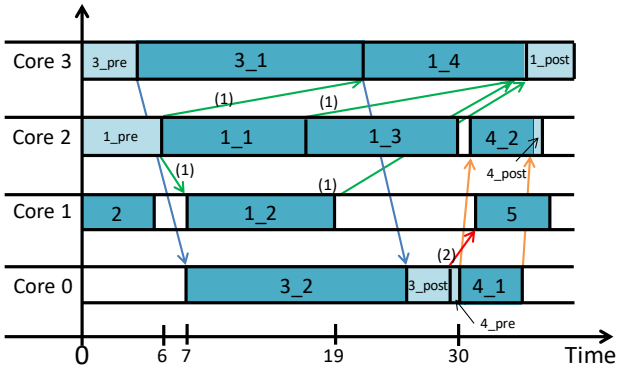
(a) タスクグラフ

#並列度	スレッド実行時間			スレッド間通信	
	前処理	ボディスレッド	後処理	前通信	後通信
1	0	40	0	0	0
2	2	21	2	2	2
3	2	14	4	2	2
4	6	12	6	1	1

(b) タスク 1 におけるスレッド実行時間・スレッド間通信時間



(c) 並列化されたタスクによるタスクグラフ



(d) スケジューリング結果

図 1　タスクスケジューリング問題の例

2. タスク間とスレッド間の通信を考慮した可変並列度 Fork-Join タスクのスケジューリング

本章では，本稿で提案するタスク間とスレッド間の通信を考慮した可変並列度 Fork-Join タスクのスケジューリングを，2.1 節において例題を用いてそのスケジューリングの概要を説明し，また 2.2 節においてスケジュ―リング問題を定式化する．

2.1 スケジューリング例

本節では，本稿で提案するタスク間とスレッドの通信を考慮した可変並列度 Fork-Join タスクのスケジューリングがどのようなものかを例を用いて説明する．図 1 がそのスケジューリング問題の例である．図 1 (a)に示される重み付き有向非循環グラフは一般的なアプリケーションをグラフ化したものであり，本稿ではこれをタスクグラフと呼ぶ．まず，各ノードはアプリケーションにおけるタスクを示している．“S”と“E”でラベル付けされたタスクはそれぞれ開始タスクと終了タスクを示しており，本研究ではこれらをダミータスクと呼び，コアなどの資源を用いた演算処理を要さないタスクとして定義する．そのほかのタスクにおいて，ノード内に数が記されたタスクはタスクグラフにおいてコアなどの資源を用いた演算処理を要するタスクとして定義される．ノード内における数はそれぞれタスクの ID を示している．各タスクの横にある()内における数は，そのタスクがシングルコアで実行されたときの実行時間を示しており，例えばタスク 1 をシングルコアで実行した場合の実行時間は 40 単位時間である．ただし，開始タスクと終了タスクはダミータスクであるため資源を用いた演算処理を要さない．したがって，これらの実行時間は 0 である．次に，タスクグラフにおける各エッジはタスク間の依存関係を示す．各エッジ上の数は二つのタスク間で通信に要する時間を示す．ここで，簡単化のために開始タスクと終了タスクは資源による演算処理を要さないタスクであるため，開始タスクから出ているエッジおよび終了タスクに向かっているエッジにおける通信時間は 0 としても一般性を失わない．

本稿におけるスケジューリング問題において，タスクは可変並列度 Fork-Join タスクを想定しており，このタスクは複数のスレッドに並列化可能である．並列化されたタスクはマルチコア上において独立に，かつ，並列に実行できる．例えば，タスク 1 が可変並列度 Fork-Join タスクであり 4 スレッドで実行される場合について考える．このとき，タスク 1 に対して，タスクをスレッドへ分割するための前処理を行うスレッド，計算を要する 4 つのボディスレッド，

Vol.2019-SLDM-189 No.30
Vol.2019-EMB52 No.30
2019/11/15

それら 4 つのスレッドの計算結果を集約するための後処理を行うスレッドの計 6 つのスレッドが生成される．図 1 (b)はタスク 1 を例とした，各並列度における各スレッドの実行時間とスレッド間における通信時間の表を示す．タスク 1 が 4 並列で実行される場合，前処理の実行時間は 6 単位時間であり，4 つのボディスレッドの実行時間はそれぞれ 12 時間単位であり，後処理の実行時間は 6 単位時間である．また，スレッド間通信は前処理からボディスレッドへ向けて行う通信を行う前通信とボディスレッドから後処理へ向けて行う後通信に分けられ，タスク 1 が 4 並列で実行される場合の通信時間はそれぞれ 1 である．また，この実行時間および通信時間の表は各タスクに対して与えられる．図 1 (c)は図 1 (a)で示したタスクグラフのタスクのうち，タスク 1, 3, 4 が並列化された場合のタスクグラフである．図 1 (d)は図 1 (c)のスケジューリング結果である．ここでタスク 1 に注目すると，タスク 1 の前処理が時刻 0 から時刻 6 にかけてコア 2 で実行されている．1 番目と 3 番目のボディスレッドは，同じコア 2 で実行されている．一方，2 番目と 4 番目のボディスレッドはそれぞれコア 1 とコア 3 で実行されている．これらのスレッドは前処理とは異なるコアで実行されているため，前通信が発生している．そのため 2 番目のボディスレッドは時刻 7 から実行が開始されている．次にタスク 1 の後処理はコア 3 で実行されている．このため，コア 3 以外で実行された 1 番目，2 番目 3 番目のボディスレッドから後処理に向けて通信が発生する．次に，図 1 (d) におけるタスク 3 の後処理に注目する．ここでは，タスク 3 の後処理からタスク 5 に向けて 2 単位時間分の通信が発生している．これは，タスク 3 の後処理とタスク 5 の実行がそれぞれ異なるコア上でマッピングされているためにタスク間の通信が必要となるためである．簡単化のため，本稿では各コアはどこへ向けても通信が可能であるように十分に接続されているものとし，かつ，通信の競合が発生しないような通信路を想定しているが，その場合もスケジューリング問題の一般性は失われない．ただ，本研究を拡張させて通信路の制約や通信の競合などの共有リソースに対する制約を容易に付与することができる．

2.2 整数計画法に基づくスケジューリング問題の定式化

本節では，均質マルチコアにおける可変並列度 Fork-Join タスクのスケジューリングを整数計画法に基づき定式化する．本研究で開発したスケジューリング手法では，各タスクの並列度の決定，マルチコア上へのマッピング，タスクスケジューリングを同時に行う．本稿ではこのスケジューリング手法は整数計画法に基づいて定式化されるが，この定式化にいくつかの数式を新たに書き足して線形計画法として定式化することも容易に行うことができる．

$par_{i,k}$ はタスク i が k 個のボディスレッドに分割される場合に 1 になる 0-1 決定変数である．すなわち k コアで実行されるときに 1 の値をとり，それ以外の場合には 0 をとる．

$$\forall i, \quad \sum_k par_{i,k} = 1 \tag{1}$$

$Time\_pre_{i,k}$，$Time\_body_{i,k}$，$Time\_post_{i,k}$，はそれぞれ，タスクが k 個のボディスレッドに分割された場合における，前処理，ボディや後処理のスレッドの実行時間である．また，これらの値は与条件として与えられる．したがって，前処理，ボディ，後処理のスレッドの実行時間を表す $time\_pre_i$，$time\_post_i$，$time\_body_i$ は次の通りになる．

$$\forall i, \quad time\_pre_i = \sum_k Time\_pre_{i,k} \times par_{i,k} \tag{2}$$

$$\forall i, \quad time\_post_i = \sum_k Time\_post_{i,k} \times par_{i,k} \tag{3}$$

$$\forall i,j, \quad time\_body_{i,j} = \sum_k Time\_body_{i,j,k} \times par_{i,k} \tag{4}$$

$Comm\_intra\_pre_{i,k}$ および $Comm\_intra\_post_{i,k}$ は，タスク i が k コアで実行されるとき，それぞれ前処理からボディに対する通信時間とボディから後処理に対する通信時間を表す．$Comm\_intra\_pre_{i,k}$ および $Comm\_intra\_post_{i,k}$ はいずれも与条件として与えられる．$pre\_intra_{i,j}$ は，前処理とボディとの間に通信が必要な場合に 1 の値をとる 0-1 決定変数である．同様に $post\_intra_{i,j}$ は，ボディと後処理との間に通信が必要な場合に 1 の値をとる 0-1 決定変数である．

$$\forall i,j, \ comm\_intra\_pre_{i,j} = \sum_k (Comm\_intra\_pre_{i,k} \times par_{i,k}) \times pre\_intra_{i,j} \tag{5}$$

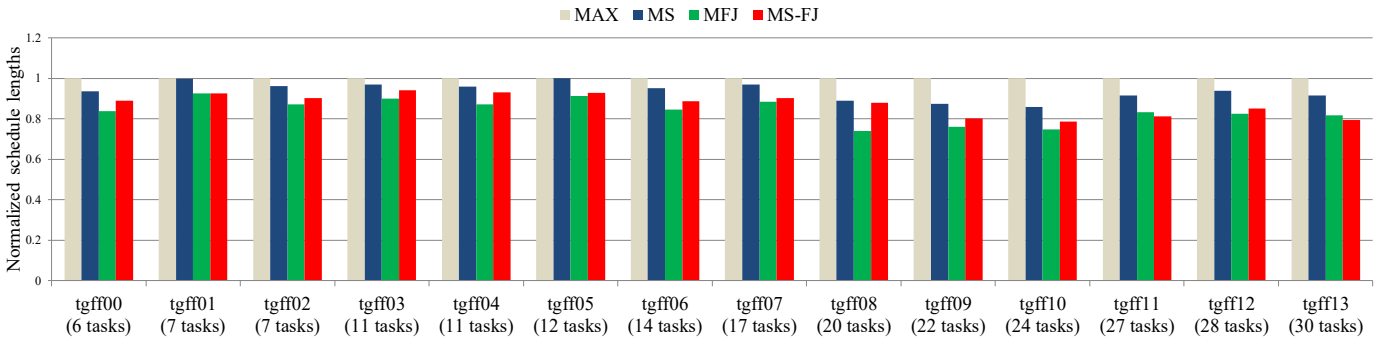
$$\forall i,j, \ comm\_intra\_post_{i,j} = \sum_k \left(Comm\_intra\_post_{i,k} \times par_{i,k}\right) \times post\_intra_{i,j} \tag{6}$$

$pre\_intra_{i,j}$ はタスク i の前処理および j 番目のボディスレッドが異なるコアにマップされるときに 1 の値をとる．同様に $post\_intra_{i,j}$ はタスク i の j 番目のボディスレッドおよび後処理が異なるコアにマップされるときに 1 の値をとる．$map\_pre_{i,k}$ および $map\_post_{i,k}$ はタスク i の前処理と後処理がそれぞれ k 番目のコアにマップされた場合に 1 の値をとる 0-1 決定変数である．$map\_body_{i,j,k}$ は，タスク i の j 番目のボディスレッドが k 番目のコアにマップされたときに 1 の値をとる 0-1 決定変数である．

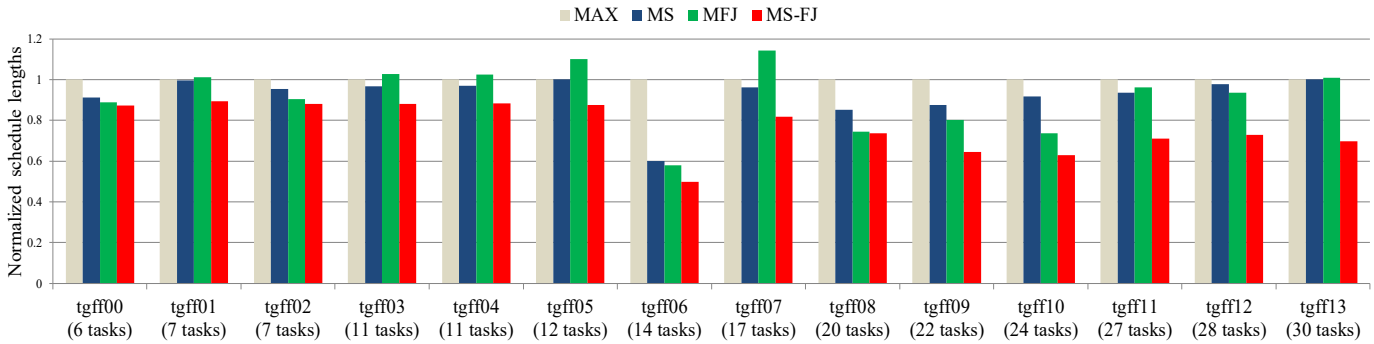
$$\forall i,j, \quad pre\_intra_{i,j} = \begin{cases} 0 & \text{if } map\_pre_{i,k} = map\_body_{i,j,k} \text{ for any } k \\ 1 & \text{otherwise} \end{cases} \tag{7}$$

$$\forall i,j, \quad post\_intra_{i,j} = \begin{cases} 0 & \text{if } map\_body_{i,j,k} = map\_post_{i,k} \text{ for any } k \\ 1 & \text{otherwise} \end{cases} \tag{8}$$

$start\_pre_i$ および $finish\_pre_i$ はそれぞれ，タスク i の前処理における開始時刻および終了時刻を示す．同様に，



(a) スケジューリング結果 (4 コア)



(b) スケジューリング結果 (8 コア)

図 2 タスク間とスレッド間通信を考慮した MFJ タスクのスケジューリング結果

$start\_body_{i,j}$ および $finish\_body_{i,j}$，$start\_post_i$ および $finish_post_i$はそれぞれ，ボディスレッドの開始時刻と終了時刻，後処理の開始時刻と終了時刻を示す．

$$\forall i,j, \quad finish\_pre_i = start\_pre_i + time\_pre_i \tag{9}$$

$$\forall i,j, \quad finish\_body_{i,j} = start\_body_{i,j} + time\_body_i \tag{10}$$

$$\forall i,j, \quad finish\_post_i = start\_post_i + time\_post_i \tag{11}$$

その際に，スレッド間の通信を考慮すると以下の制約式が必要となる．

$$\forall i,j, \quad finish\_pre_i + comm\_intra\_pre_{i,j} \le start\_body_{i,j} \tag{12}$$

$$\forall i,j, \quad finish\_body_{i,j} + comm\_intra\_post_{i,j} \le start\_post_i \tag{13}$$

複数のボディスレッドはマルチコア上で独立に実行することができる．それら複数のボディスレッドは並列，に異なるコア上で実行されるか，あるいは，同じコア上で実行される．後者の場合，同一時刻において複数のスレッドを同一のコア上で実行することが出来ない．この資源制約は次のように表される．

$$\forall i, j1, j2, k\ (j1 \neq j2), \tag{14}$$

$$map\_body_{i,j1,k} = map\_body_{i,j2,k} \rightarrow \left(finish\_body_{i,j1} \le start\_body_{i,j2}\right) \vee \left(finish\_body_{i,j2} \le start\_body_{i,j1}\right)$$

ここまでスレッド間における資源制約および順序制約に注目してきた．タスク間についても同様に資源制約と順序制約を満たす必要がある．しかし，本論文では割愛する．

本研究におけるスケジューリング問題の目的は，スケジュール長 （メイクスパン）の最短化である．すなわち，本問題における目的関数は次の通りになる．

$$\text{Minimize:} \quad \max_i\{finish\_post_i\} \tag{15}$$

本スケジューリング問題は整数計画問題として定式化される．上式のいくつかは線形式ではないが，これらは容易に線形化できる．したがって，本定式化は汎用の ILP ソルバー上で解くことができる．

3. 実験

3.1 実験手法

本節では，本研究で提案するタスク間・スレッド間の通信を考慮した可変並列度 Fork-Join タスクのスケジューリング手法と実験に用いたベンチマーク及び比較手法について説明する．ベンチマークには TGFF [11]を基にして生成した 14 個のタスクグラフを用いた．ターゲットシステム

は 4 コアおよび 8 コアの 2 種類を想定している．求解ソルバーには ILOG CP Optimizer 12.8 を使用している．本研究で開発したスケジューリング問題は非常に複雑であり，解空間が広いため実用的な時間内に最適解を求めることは困難である．そのため，本研究では求解時間を 5 時間として既定し，その時点における許容解をその手法における解として評価に用いる．なお，実験環境は Core i9 7980XE (2.6GHz)プロセッサを使用しており，主記憶は 128GB である．

本研究では，比較手法として以下の 4 手法を用いる．

Max: 各タスクは全てのコア上で実行される．言い換えると，それぞれのタスクは N コアのターゲットシステムにおいて N 個のボディスレッドに分割される．そしてそれらのタスクは逐次的に実行される．

MS: 文献[10]で提案されている可変並列度タスクのスケジューリング手法である．この手法は本稿の手法とは異なり，タスクにおける全てのボディスレッドは同時に開始されることを想定している．

MFJ: 本稿で提案している，可変並列度 Fork-Join 型タスクのスケジューリング手法である．

MS-FJ: 2 段階のヒューリスティックな可変並列度 Fork-Join タスクのスケジューリング手法である．始めの 1 時間で MS 手法を用いて各タスクの並列度を決定し，残りの 4 時間でスレッドのスケジューリングを行う手法である．

3.2 実験結果

図 2 (a)，(b)は実験結果を示す．横軸はタスクグラフを示し，括弧内の数は各タスクグラフに内包されるタスク数を表している．縦軸はタスクグラフのスケジュール長を表し，結果は全て Max 手法によって正規化した．

図 2 (a)は 4 コアを用いたときのスケジューリング結果である．MS 手法，MFJ 手法や MS-FJ 手法はそれぞれ Max 手法に比べると，平均で 6.2%，16.0%，12.6% スケジュール長を短くすることに成功した．ほとんどの場合で，MFJ 手法が 4 つの中では最も良い結果を得た．

図 2 (b)は 8 コアを用いたときのスケジューリング結果である．MS 手法，MFJ 手法，MS-FJ 手法はそれぞれ平均で 7.7%，8.1%，23.2%スケジュール長の縮小に成功した．4 コアの場合と異なる点として，MS-FJ 手法が 4 つの中で最も良い結果を得ていることが挙げられる．これは，8 コアを用いた場合には解空間が巨大化し，計算量が膨大になるために 5 時間という実行時間内に MFJ 手法は良いスケジュール長を得ることが困難であるからである．また，いくつかのケースでは MFJ 手法は MS 手法よりも悪い結果を示したが，その一方で，MFJ 手法のヒューリスティックな手法である MS-FJ 手法は MFJ 手法よりも効率的に解空間の探索を行うことができた．したがって MS-FJ 手法が最も良いスケジュール長を得ることができたと考えられる．

4. おわりに

本稿では，タスク間とスレッド間の通信を考慮した均質なマルチコアによる可変並列度 Fork-Join タスクのスケジューリング手法を提案した．提案手法は，各タスクが複数のスレッドに分割されて並列に実行されることを許し，並列度の決定，マルチコア上へのマッピング，タスクスケジューリングを同時に行う．実験において，小規模なタスクグラフにおいては本研究で提案した手法が有用性を示し，タスクグラフの規模が大きくなるにつれ，提案手法をさらに発展させたヒューリスティック手法が有用であることが示された．今後はさらに現実的なアプリケーションを用いたタスクグラフに対して，より高速なヒューリスティック手法の提案を計画している．

謝辞 本研究は一部，キオクシア株式会社（旧社名 東芝メモリ株式会社）の支援による．

参考文献

[1] M. Drozdowski, “Scheduling multiprocessor tasks: An overview,” *European Journal of Operational Research*, 1996.
[2] P. F. Dutot, G. Mounie, and D. Trystram, “Scheduling parallel tasks approximation algorithms,“ *Handbook of Scheduling: Algorithms, Models and Performance Analysis*, 2004.
[3] Y. Liu, L. Meng, I. Taniguchi and H. Tomiyama, “Novel list scheduling strategies for data parallelism task graphs,” *International Journal on Networking and Computing*, 2014.
[4] H. Yang and S. Ha, “ILP based data parallel multi-task mapping/scheduling technique for MPSoC," *International SoC Design Conference*, 2008.
[5] H. Yang and S. Ha, “Pipelined data parallel task mapping/scheduling technique for MPSoC,” *Design Automation and Test in Europe (DATE)*, pp.69-74, 2009.
[6] J. Sun, N. Guan, Y. Wang, Q. Deng, P. Zeng and W. Yi, “Feasibility of fork-join real-time task graph models: Hardness and algorithms," *ACM Transactions on Embedded Computing Systems (TECS)*, 2016.
[7] K. Lakshmanan, S. Kato, and R. Rajkumar, “Scheduling parallel real-time tasks on multi-core processors.” *IEEE Real-Time Systems Symposium*, 2010.
[8] C. Chen and C. Chu, “A 3.42-Approximation algorithm for scheduling malleable tasks under precedence constraints,” *IEEE Transactions on Parallel and Distributed Systems*, 2013.
[9] H. Nishikawa, K.Shimada, I. Taniguchi, and H.Tomiyama, “Scheduling of malleable fork-join tasks with constraint programming” in *Proc of International Synposium on Computing and Networking (CANDAR)*, 2018.
[10] K. Shimada, I. Taniguchi, and H. Tomiyama, “Communication-aware scheduling for malleable tasks,” in *Proc. of International Conference on Platcon Technology and Service*, 2019.
[11] R. P. Dick, D. L. Rhodes, and W. Wolf, “TGFF: task graph for free,“ *International Workshop on Hardware/Software Codesign*, 1998.