Vol.2014-DPS-159 No.13
Vol.2014-MBL-71 No.13
2014/5/15

ターボブースト・ハイパースレッディングを考慮したタスクスケジューリング手法

脇坂 洋祐[1]　柴田 直樹[1]　北道 淳司[2]　安本 慶一[1]　伊藤 実[1]

概要：マルチコアプロセッサを搭載した計算機が広く利用されるようになり，また様々なタスクスケジューリング手法が利用されている．また，マルチコアプロセッサの処理性能を向上させるため，ターボブースト及びハイパースレッディングと呼ばれる処理高速化及び並列処理技術が開発され，搭載されるようになった．これらはそれぞれ一部のコアが使用されていない時に使用されているコアの動作周波数を向上させる技術と，一つの物理コアを複数の論理プロセッサに見せかける技術である．既存のタスクスケジューリング手法は，これらの最新技術を考慮していない．本研究では，ターボブースト及びハイパースレッディングによる実効動作周波数の動的変更とネットワークコンテンションを考慮したタスクスケジューリングアルゴリズムを提案する．実機実験を通してターボブースト及びハイパースレッディングによる動作周波数モデルを作成し，タスクの処理時間を正確に推定することで両技術を考慮したタスクのスケジュールを行う．シミュレーションと実機実験による評価の結果，提案手法は全体の処理時間をシミュレーションで最大 43%，実機実験では最大で 36% 短縮できることを確認した．

1. 序論

近年，マルチコアプロセッサを搭載した計算機が広く利用されており，一部のプロセッサにはターボブースト [1] と呼ばれる処理高速化技術が採用されている．これは，一部のコアのみが使用されている場合，使用中のコアの動作周波数を向上させる技術である．また，ハイパースレッディング [2] と呼ばれる技術も採用されており，これは一つの物理プロセッサコアで複数のスレッドを並行に実行することでハードウェア資源の効率的な使用を可能にしている．

データセンタの様な分散並列環境では，最新の技術を効率的に利用できることが望ましい．既存のタスクスケジューリング手法を単純にターボブースト及びハイパースレッディングを搭載したシステムに適用した場合，ネットワークを介した通信よりプロセッサコア間での通信が高速であるため，同じマルチコアプロセッサにタスクの割り当てが集中し，ターボブーストによる動作周波数の向上がなされなくなるため，計算資源を有効に利用できないという問題がある．

本稿では，上記の問題を解決するため，ネットワークの制約やターボブースト及びハイパースレッディングによる動作周波数の動的変更を考慮し，全体の処理時間を最小化するようなタスクスケジューリング手法を提案する．提案手法では，閉路を持たない有向グラフ，すなわち非循環有向グラフ（Directed Acyclic Graph : DAG）によりタスクの依存関係を示したタスクグラフと，計算機システムのネットワークトポロジを表すプロセッサグラフを入力とし，複数のコアにタスクを割り当てる．

提案手法を評価するため，シミュレーションと実機実験を行った．生成したスケジュール結果及びそれを用いた実機実験の結果の比較により，提案手法は比較手法に比べシミュレーションで最大 43%，実機実験で最大 36% 処理時間を短縮できることを確かめた．また，シミュレーションと実機上での実行結果を比較することで本研究で作成した動作周波数モデルの妥当性を評価した．その結果，モデルと実際の処理とでの誤差は 7% 以下であることを確認した．

2. 関連研究

リストスケジューリングは古典的なタスクスケジューリング手法であり，タスクを定められた優先度に従って、最も早く完了できるプロセッサに割り当てていくヒューリスティックな処理を行う．

James らは，マルチコアプラットフォーム上でのリアルタイム制約を満たしつつ複数のタスクグループを同時に割り当てを行う協調スケジューリングを提案している [3]．こ

[1] 奈良先端科学技術大学院大学
Nara Institute of Science and Technology
[2] 会津大学
The University of Aizu

の手法では共有 L2 キャッシュのより効率的な利用を促進するようにスケジュールを生成することで，キャッシュミスレートを低下させ，キャッシュミスによる処理時間の増加を最小限にすることを目標としている．

著者らの所属する研究グループでは，文献 [4] において，複数のマルチコアプロセッサからなる環境での単一ノード故障時における回復時間を最小化するタスクスケジューリングを提案している．この手法ではネットワークコンテンションを考慮しつつ，故障時のリカバリータイムを加味した上で処理時間を削減するようにタスクの割り当てを行う事で，単一故障が発生した場合での全体の処理時間を最小化している．提案した手法に関してシミュレーション及び実機実験を通して有効性を確認している．

上記以外にもマルチコアプロセッサを対象としているタスクスケジューリング手法が提案されている．これらの研究はターボブースト及びハイパースレッディングによる動作周波数の動的変更及びネットワークコンテンションを同時に考慮しておらず，その他の研究においても，調査した限り考慮している研究は見つけられなかった．

古典的なタスクスケジューリング手法の多くは，ネットワークコンテンションを考慮していない．そのため，ネットワークを介したデータ転送を行った際の転送遅延や競合が発生しない理想的な通信路として扱われ，タスクの処理に必要な現実の時間が正確に反映されていない．Sinnen らは，ネットワーク帯域やネットワークコンテンションが生じない環境を仮定したリストスケジューリングを拡張し，ネットワークの遅延やコンテンションを考慮したスケジューリング手法の提案をしている [5]．この文献では，実環境におけるネットワークの遅延やコンテンションの発生を考慮しつつ，タスクの処理時間が最小となるようにタスクのスケジュールを行い，同時にタスクの処理に必要なデータ転送に使用する通信経路のスケジュールも行なっている．提案されたスケジューリング手法の評価として，スケジュールの正確性や，タスクグラフ全体の処理時間の削減率に関する議論が行われている．

本研究では，ターボブースト及びハイパースレッディングを搭載したマルチコアプロセッサが主となる環境において，タスクノードを処理するダイの使用状況により動的に変更される動作周波数とネットワークコンテンションによる遅延とのバランスを考慮しながら，全体の処理時間を削減するスケジューリング手法を提案する．

3. ターボブースト及びハイパースレッディングのモデル化

ターボブースト及びハイパースレッディングを搭載したマルチコアプロセッサでは，両技術がプロセッサの処理状況に応じて，動的に動作周波数を変更し，その時々で可能な最高動作周波数で処理を行う．そのため，各プロセッサの使用状況に応じた動作周波数を推定する必要が生じる．両技術がどのように動作周波数を変化させるかは，一部公開されているものの，この情報から動作周波数を完全に求められるわけではない．本研究では，実機上で負荷を評価するためのプログラムを複数回実行し，各使用状況における動作周波数を計測することにより，これらの技術に対する動作周波数モデルを作成した．以下ではターボブースト及びハイパースレッディングの両技術について述べ，両技術に対する動作周波数のモデルについて述べる．

ターボブーストは，ダイの使用状況を監視し，処理状況に応じて動的に動作周波数を変更する．この技術は，ダイ全体の温度，供給される電流及び電力が仕様上設定されている限界と比べ余裕がある場合，アクティブとなっているプロセッサ数に応じて決められた動作周波数の上限まで自動で動作周波数を引き上げることでダイ全体の処理性能を向上させる．本研究ではマルチコアプロセッサを搭載した計算機のターボブーストによる動作周波数をモデル化するため，実機上で負荷プログラムを各コアに割り当て，複数回実行した際のタスクの実行時間を計測し，使用されているコアの数ごとの動作周波数を推定した．

ハイパースレッディングは，1 物理プロセッサを複数の論理プロセッサに見せ，1 物理プロセッサ上で複数の論理プロセッサを実行することで，各論理プロセッサに割り当てられたスレッドを同時に実行する．ハイパースレッディングにより，1 つの物理プロセッサ上で複数スレッドを動作させる場合，単一スレッドの場合に比べ，複数スレッドでリソースを共有しているため，各スレッドの実行速度は低下する．本研究では，この速度低下を動作周波数の低下としてモデル化する．各スレッドの実行速度の低下を加味した動作周波数を**実効動作周波数**と呼ぶ．

本研究では，ダイ上の全論理プロセッサの使用状況のみから，各論理プロセッサの実効動作周波数が一意に定まると仮定してモデルを構築する．両技術による動作周波数の切り替えはタスクの実行中でも瞬時に行えることとし，論理プロセッサの使用状況は，以下の 4 状態のいずれかであると仮定する：(1) アイドル，(2) コンピュテーションヘビー，(3) メモリアクセスヘビー，(4)2 と 3 の中間．

両技術による実効動作周波数の変化モデルを構築するため，80MB の配列を持ち，ランダムに選択された 2 要素のスワップ (メモリアクセス) とキャッシュ内で完結する複数回の反復 (コンピュテーション) を行う負荷プログラムを作成した．作成したプログラムは反復回数を変更でき，2 種類の処理の比を調節することができる．複数の論理プロセッサで負荷プログラムを実行した際の動作時間を測定し．得られた結果から実効動作周波数の算出を行った．

実験環境は Intel Core i7 3770T(2.5GHz，物理 4 プロセッサ，論理 8 プロセッサ，シングルソケット)，メモリ

表 1: ターボブースト時の動作周波数

論理プロセッサの状態のパターン	動作周波数 (GHz)
[2 , 1] , [1 , 1] , [1 , 1] , [1 , 1]	3.7
[2 , 1] , [2 , 1] , [1 , 1] , [1 , 1]	3.5
[2 , 1] , [2 , 1] , [2 , 1] , [1 , 1]	3.3
[2 , 1] , [2 , 1] , [2 , 1] , [2 , 1]	3.1
[3 , 1] , [1 , 1] , [1 , 1] , [1 , 1]	3.7
[3 , 1] , [3 , 1] , [1 , 1] , [1 , 1]	3.5
[3 , 1] , [3 , 1] , [3 , 1] , [1 , 1]	3.3
[3 , 1] , [3 , 1] , [3 , 1] , [3 , 1]	3.1
[4 , 1] , [1 , 1] , [1 , 1] , [1 , 1]	3.7
[4 , 1] , [4 , 1] , [1 , 1] , [1 , 1]	3.5
[4 , 1] , [4 , 1] , [4 , 1] , [1 , 1]	3.3
[4 , 1] , [4 , 1] , [4 , 1] , [4 , 1]	3.1
[1 , 1] , [1 , 1] , [1 , 1] , [1 , 1]	2.5

1 : アイドル，2 : コンピュテーションヘビー，
3 : メモリアクセスヘビー，4 : 2 と 3 の中間

表 2: ハイパースレッディング時の実効動作周波数

コアの状態	実効速度比	実効動作周波数 (GHz)
[2 , 2]	0.84	2.6
[3 , 3]	0.76	2.3
[4 , 4]	0.79	2.5

16.0GB，Windows7 SP1(64bit)，Java(TM) SE (1.6.0 21, 64bit) である．動作周波数の計測は CPU-Z(Ver1.61.3) とインテル・ターボブースト・モニター (Ver2.5) を用いた．

ターボブーストに関する実験結果を表 1 に示す．左欄はプロセッサの使用状況を表す．4 つの括弧で閉じられたものは 1 つの物理プロセッサに対応し，物理プロセッサは 2 つの論理プロセッサの組であらわされる．右欄は，左欄の使用状況に対応する動作周波数を表している．

本研究においては，ハイパースレッディングは物理プロセッサ数を超えて一つのダイにスレッドを割り当てる場合にのみ利用する．よってハイパースレッディング時の各論理プロセッサの実効動作周波数はメモリアクセスの割合にのみ依存すると仮定する．論理プロセッサ及び物理プロセッサに対して同様の負荷プログラムを実行した際の処理時間の比と全物理プロセッサを使用した状態の動作周波数から実効動作周波数を算出した．

本研究で作成する動作周波数モデルは，これらの結果を利用し，タスクノードがスケジュールされているプロセッサの使用状況を入力とし，使用状況に応じたターボブーストまたはハイパースレッディングによる動作周波数を測定した結果から一意に決定する．

4. 問題定義

タスクスケジューリングへの入力は後述するタスクグラフとプロセッサグラフであり，出力は各タスクノードをプロセッサノードに割り当てたスケジュール結果である．本研究では，タスクグラフ全体の処理時間の最小化を目指す．

本節では，提案手法を説明する上で必要となる用語の定義を行う．また，以降で使用する記号の一覧を表 3 に示す．

スケジュールされるプログラムを DAG によって表現し，タスクグラフ G と呼ぶ．タスクグラフのスケジュールとは，各タスクノードの処理開始時刻とプロセッサノードの関連付けである．タスクグラフ内の各頂点をタスクノードと呼び，各タスクノードは，1 つのプロセッサで逐次的に実行するための命令の集合を表す．タスクノード n の処理量を $C_{comp}(n)$ で示す．タスクノード間の有向枝をタスクリンクと呼び，ノード間の依存関係及び必要な通信を表す．タスクリンク e での通信に用いられるデータ量を $C_{comm}(e)$ と表す．タスクノード，タスクリンク全ての集合及び開始ノードをそれぞれ V, E 及び v_{start} とするとき，タスクグラフは $G = (V, E, v_{start}, C_{comp}, C_{comm})$ と表現される．図 1 にタスクグラフの例を示す．図 1 は 5 つのタスクノードと 5 つのタスクリンクからなるタスクグラフである．各タスクノード内の値はそのノードの処理量 C_{comp} を表す．また，各タスクリンクの値はそのノード間で行われるデータ転送のデータ量 C_{comm} を示している．例えばタスクノード c の処理はタスクノード a の処理とデータ転送の完了後に処理を開始できる．また，タスクノード b, c, d は並列に処理することが可能であることを示す．

プロセッサグラフは，ネットワークトポロジを表現した

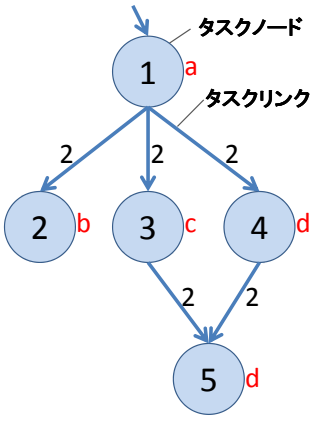


図 1: タスクグラフの例

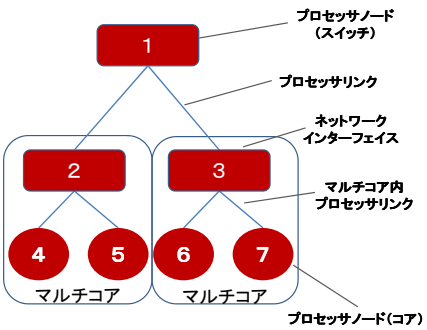


図 2: プロセッサグラフの例

表 3: 記号表

記号	意味
G	タスクグラフ
V	タスクノード全ての集合
E	タスクリンク全ての集合
$C_{comp}(n)$	タスクノード $n \in V$ の計算コスト
$C_{comm}(e)$	タスクノード $e \in E$ の通信コスト
N	プロセッサグラフ
P	プロセッサノード全ての集合
R	プロセッサリンク全ての集合
$freq(s)$	プロセッサの使用状況を s とし，使用率に応じた動作周波数を決定する関数
n_i	i 番目のタスクノード
$w(n_i, p_i, s)$	タスクノード n_i を状態 s のプロセッサノード p_i で処理する際の処理時間
e_{ij}	タスクノード n_i からタスクノード n_j への通信
$c(e_{ij})$	通信 e_{ij} に要する時間
$t_s(n_i, p_i)$	タスクノード n_i をプロセッサノード p_i で実行する際の処理開始時刻
$t_f(n_i, p_i)$	タスクノード n_i がプロセッサノード p_i で実行する最の処理完了時刻
$t_{ef}(e_{ij})$	通信 e_{ij} の完了時刻
$t_{dr}(n_j, p_i)$	親ノードの終了時刻からプロセッサノード p_i でタスクノード n_j の実行を開始するまでに掛る時間
$proc(n_i)$	タスクノード n_i が割り当てられているプロセッサノード
$pred(n_i)$	タスクノード n_i の親ノードの集合
$succ(n_i)$	タスクノード n_i の子ノードの集合
$bl(n_i)$	タスクノード n_i の実行開始時刻　先行タスクの実行完了時刻と通信時間の和の最大として与えられる
$lt(S)$	スケジュールした結果 S 内のタスクノードの最遅処理完了時刻

ものであり，各頂点をプロセッサノード，各辺をプロセッサリンクと呼ぶ．リンクを 2 つ以上持つプロセッサノードはネットワーク上のスイッチあるいはマルチコアプロセッサを搭載した計算機のネットワークインターフェイスであり，それらでタスクノードの処理はできないものとする．リンクを 1 つのみ持つものはマルチコアプロセッサ内のプロセッサを示している．リンクは，通信経路を表し，接続される 2 つのプロセッサ間は双方向に通信を行う事ができる．プロセッサノードの集合を P，プロセッサノード間のリンクの集合を R とする．3 章で述べたターボブースト・ハイパースレッディングのモデルを用いて，各状態において，ダイの使用率が状態 s の時の実効動作周波数を返す関数を $freq(s)$ とする．この時，プロセッサグラフを $N = (P, R, freq)$ と表す．

また，スイッチにおけるデータの転送処理時間はタスクの処理に比べて非常に短いため無視できるものとする．プロセッサグラフの例を図 2 に示す．この図において，角が丸い四角で囲まれた白い領域はデュアルコアプロセッサを示す．図のネットワークグラフは 2 つのデュアルコアプロセッサが 1 つのスイッチで接続されているトポロジを表す．

本研究では，マルチコアプロセッサ内のプロセッサ間でのデータ転送に掛る時間はネットワークを介したデータ転送と比較すると小さく，0 とする．ネットワークを介した通信は方向を問わず同一リンク上で複数の転送は同時に行えないものとし，リンクの帯域は全て同じとする．マルチコアプロセッサの動作周波数モデルは 3 章での実験で得られたものを用いる．

本研究で扱うスケジューリング問題は Sinnen らのモデル [5] をベースにしており，以下の 3 つを制約条件とする．(1) 同じプロセッサノードに割り当てられた 2 つのタスクノードは，片方の処理が終了するまでもう一つの処理を開始できない．(2) 各タスクノードの処理は親ノードの処理が全て完了し，全ての親ノードからのデータ転送が完了するまで開始できない．(3) タスクノードの処理をスケジュールする際，その処理はプロセッサノードの空き時間内に終了しなければならない．これらは，文献 [5] の Condition1〜3 に該当する．

ネットワークリソースは有限であり，ネットワークコンテンションが発生するとする．そのため，ネットワークコンテンション回避のためネットワークに関して，Sinnen[5] らのネットワークコンテンションのモデルをマルチコアプロセッサへ対応させたモデルを用いる．制約条件は以下の 3 つである．(1) タスクノード間のデータ転送は同時に行えない．(2) データ転送の開始時刻を先行のデータ転送の開始時刻より前にすることはできない．(3) データ転送をスケジュールする時，利用するネットワークが使用されていない空き時間内にデータ転送が終了しなければならない．これらは，文献 [5] の Condition4〜6 に該当する．以下では上記の条件をマルチコアプロセッサ向けに拡張する．

本問題への入力としてタスクグラフ $G = (V, E, v_{start}, C_{comp}, C_{comm})$，プロセッサグラフ $N = (P, R, freq)$ を与える．出力は各タスクノードをプロセッサノードへ割り当てたスケジュール S である．本問題の目的関数は式 $minimize\,(lt(S))$ で示す．ただし，$lt(S)$ はスケジュールされた結果 S に対する最遅処理完了時刻を表す．

5. 提案手法

本研究で扱う問題は NP 困難に属する組み合わせ最適化問題であり，タスクグラフのサイズが大きくなると最適解を短時間で求めることは困難である．本稿では，本問題を効率的に解くために Sinnen[5] らのリストスケジューリン

グアルゴリズムを拡張した近似アルゴリズムを提案する．

提案手法では，ターボブースト及びハイパースレッディングによる動作周波数の動的変更とネットワークコンテンションを考慮し，タスクノードの処理時間を推定することで，全体の処理時間を最小とするスケジュールを生成する．ネットワークの遅延及び競合を考慮するために，既存のネットワークコンテンションを考慮した手法をベースに拡張する．割り当て対象とするタスクノードの処理期間中のダイの使用状況を調べるために，並列に処理が行われるタスクノードを判定する必要がある．そのために，後続タスクノードを Sinnen らのネットワークコンテンションを考慮した手法を用いて空いているプロセッサノード群の中から最も早く処理完了できるプロセッサノードへ仮割り当てを行う．その後，生成したスケジュール結果に対して，本研究で作成したターボブースト及びハイパースレッディングによる動作周波数の変更モデルを用い，両技術によるプロセッサノードの利用率に応じた動作周波数の変更による処理時間の変動及び，それに伴う処理開始時間等の調節を行う事でタスクノードの処理時間等を推定する．これをプロセッサノード全てに対して行い，本来割り当てたいタスクノードを各プロセッサノードに割り当てた場合の全体の処理時間を正確に推定していく．

また，本手法では，タスクノードの処理時間の推定を全てのプロセッサノードに対して割り当てた場合を考えており，その中からスケジュールに採用するプロセッサノードへの割り当てパターンを選択する必要がある．選択の基準として，各パターン毎の仮割り当て時の結果から推定した全体の処理完了時刻を利用し，全てのパターンの中で最良と思われる割り当てパターンを採用する．その後，後続のタスクノードのスケジュールへ移行する．

全てのタスクノードに対してこの処理時間推定と割り当てるプロセッサの選択処理を繰り返し行うことで最終的なスケジュールを生成する．

5.1 古典的なリストスケジューリング

提案するタスクスケジューリングアルゴリズムは，古典的なリストスケジューリングアルゴリズムを拡張している．リストスケジューリングアルゴリズムの一例を Algorithm1 に示す．リストスケジューリングでは，先行制約及び問題に依存する優先度 (例：残りのスケジュール長の大きい順) に従ってタスクノードをプロセッサノードに割り当てる．タスクノードが割り当てられるプロセッサノードは，そのタスクノードを最も早く処理完了出来るプロセッサノードである．このアルゴリズムではネットワークコンテンションは考慮されていない．

5.2 ネットワーク特性の考慮

ネットワークの特性を考慮するスケジューリング手法で

Algorithm 1 リストスケジューリング

入力：タスクグラフ $G = (V, E, w, c)$，プロセッサグラフ $H = (P, R)$

1: 先行制約と手法ごとの優先度に基づき $\mathbf{V}$ 中のタスクノード n をソートし，リスト L に代入.
2: リスト L の先頭ノードのポインタを i とする.
3: **for** each リスト $n_i \in L$ **do**
4: n_i が最も早く終了できるプロセッサノード p を見つける.
5: p に n_i をスケジュール.
6: ポインタ i を後続ノードへ移す.
7: **end for**
8: **return** スケジュール

Algorithm 2 ネットワークコンテンションを考慮したスケジューリング

入力：タスクグラフ $G = (V, E, w, c)$，プロセッサグラフ $H = (P, R)$

1: 先行制約と手法ごとの優先度に基づき $\mathbf{V}$ 中のノード n をソートし，リスト L に代入.
2: リスト L の先頭ノードのポインタを i とする.
3: **for** each $n_j \in L$ **do**
4: $findProcessor(\mathbf{P}, n_j)$ を用いて最も早く処理完了できるプロセッサノード p を見つける.
5: **for** $n_i \in pred(n_j)$ **do**
6: **if** $proc(n_i) \neq p$ **then**
7: $proc(n_i)$ から p への通信リンク $R = [L_1, L_2, ..., L_l]$ を決定する.
8: R に e_{ij} をスケジュール
9: **end if**
10: **end for**
11: n_j を p にスケジュール
12: ポインタ i を後続ノードへ移す.
13: **end for**
14: **return** スケジュール

は，通信帯域やスイッチでの遅延，転送するデータの競合などを考慮しつつ，各手法の目的を達成するためにタスクノードの割り当てを行う．提案手法のベースとなる Sinnen らの手法 (Algorithm2) では，タスクノードを割り当てるプロセッサノードを選択する部分においてネットワークコンテンションを考慮している．Algorithm1 の 3 行目では，タスクノード n が最も早く処理完了出来るプロセッサノードを選択する．この処理に加えて，Algorithm2 では，ネットワークによるデータ転送の遅延及び競合を考慮する．

5.3 ターボブースト・ハイパースレッディングによる動作周波数の変更を考慮した提案アルゴリズム

タスクノードの処理時間を正確に推定するためには，実際にタスクノードが処理される際のターボブースト及びハイパースレッディングによる動作周波数を知る必要がある．しかし，リストスケジューリングではタスクノードに割り当てられた優先度に基づいてソートされたリストの先頭か

ら順に割り当てを行うため，タスクノードを割り当てる時点では，後続のタスクノードの割り当てによって割り当てたプロセッサノードの使用率が変化してしまう可能性があり，正確な動作周波数を求めることができない．そこで，プロセッサノードの使用状況を把握するために，提案手法では Sinnen らのネットワークコンテンションを考慮した手法を用いて後続のタスクノードの仮割り当てを行い，割り当てたいタスクノードを処理する動作周波数を暫定的に決定し，タスクノードの処理時間を求める．提案手法では，ターボブースト及びハイパースレッディングの動作周波数変更モデルを使用し，両技術が動作している場合の処理時間の再調整を行う．その後，割り当てたプロセッサノードの中から最も処理完了時刻が早い結果と推定された割り当てを採用し，後続のタスクノードの割り当てを行う．この一連の動作を全てのタスクノードに対して行う事でターボブースト及びハイパースレッディングによる動作周波数の動的変更を考慮したうえで全体の処理時間を最小と出来るスケジュール結果を作成する．

提案アルゴリズムを Algorithm3 に示す．提案手法ではまず，既存のリストスケジューリングと同じように入力として与えられたタスクグラフ内のタスクノードを各タスクノードの $bl(n)$ に従ってソートする (Algorithm3 の 1 行目)．$bl(n)$ は, プロセッサノードの動作周波数をある値に固定して得られた各タスクノードの実行時間と, タスクノード間の通信時間を 0 としたときの, タスクノード n の最も早い実行開始時間である．タスクノードの処理時間を正確に推定するために，各プロセッサノードにネットワークコンテンションを考慮してタスクノード割り当てる (4-10 行目)．その後，各割り当てパターンに対して，処理完了時刻推定タスクスケジューリングを通してタスクノードを処理する際の動作周波数を暫定的に決定するために必要な後続のタスクノードの仮割り当てを行う．そして，仮スケジュールの結果に対して本研究で作成した動作周波数モデルを用いてプロセッサノードの使用状況に応じた動作周波数を暫定的に決定し，全体の処理完了時刻の推定を行う (11 行目)．後続タスクノードの割り当てとターボブースト及びハイパースレッディングによる処理時間の際調節を行うタスクスケジューリングについての詳細は後述する．割り当てようとするタスクノードに対して推定した各割り当てパターンの処理完了時刻を比較し，その中で最良のものを該当タスクノードの割り当て結果として採用する (12-16 行目)．これを全てのタスクノードが割り当てられるまで繰り返す (2 行目-18 行目)．最終的に得られた結果が提案手法のスケジュール結果である．

後続タスクノードの割り当て及び全体の処理完了時刻の推定に用いる処理完了時刻推定タスクスケジューリングを Algorithm4 に示す．このタスクスケジューリングは Sinnen らの手法をベースにしている．入力として，割り当てパターンの途中結果，タスクグラフとプロセッサグラフを受け取り，受け取った結果に対して残りの後続タスクノードの割り当てを行う．割り当てるプロセッサノードの選択には findProcessor 関数を用い，タスクノードを最も早く処理を完了できるプロセッサノード p を決定する (Algorithm4 の 4 行目)．後続タスクノードを選択したプロセッサノード p へ順に割り当てて行く (5-11 行目)．全てのタスクノードを仮スケジューリングした結果に対して，本研究で作成した動作周波数モデルを用いてターボブースト及びハイパースレッディングによる動作周波数の変更を適用し，処理時間の再調整を行う (14 行目)．再調整後のスケジュール結果の処理完了時刻を入力として与えられた割り当てパターンに対する全体の処理完了時刻とする．

$adjusterTBHT(S', G, N)$ は，生成したスケジュール結果の処理開始時刻，処理時間及び処理完了時刻とそれに伴う通信時間をターボブースト及びハイパースレッディングによる動作周波数の変更を考慮して調節する関数である．この関数では，タスクグラフのスケジュール結果を入力として受け取り，各時間でのダイの使用状況からターボブー

Algorithm 3 ターボブースト・ハイパースレッディングを考慮したタスクスケジューリング

入力：タスクグラフ $G = (V, E, v_{start}, C_{comp}, C_{comm})$ とプロセッサグラフ $N = (P, R, freq)$
出力：タスクグラフ G の最終的なスケジュール結果 S
変数 $FLOAT_{MAX}$：浮動小数点データの最大値

1: G から全てのタスクノード n を先行制約を満たした状態で $bl(n)$ の最も大きくなる順でソートした結果をリスト L に代入する．
2: **for** $n_i \in L : n_i$ はリスト L の先頭ノード **do**
3: **for** each $p_i \in P$ **do**
4: **for** $pred(n_j)$ 中の n_i **do**
5: **if** $proc(n_i) \neq p_i$ **then**
6: $proc(n_i)$ から p_i へのリンク $R = [L_1, L_2, ..., L_l]$ を決定する．
7: R に e_{ij} をスケジュール
8: **end if**
9: **end for**
10: $s_1 =$ タスクノード n_i の処理時間及び処理完了時刻を算出し，プロセッサノード p_i にスケジュールする．
11: $s_2 = s_1$ の結果を基に Algorithm4 を用いて後続タスクの仮スケジューリングと処理完了時刻の推定を行う．
12: $ftime =$ スケジュール s_2 の処理完了時刻を代入する．
13: **if** $ftime < latestTime$ **then**
14: 採用するスケジュール S を s_1 に入れ替える．
15: $latestTime = ftime$.
16: **end if**
17: **end for**
18: リスト L の先頭ノードを取り除く．
19: **end for**
20: $adjusterTBHT(S, G, N)$ を通してターボブースト及びハイパースレッディングによる処理時間の調節を行う．
21: **return** スケジュール S

Algorithm 4 処理完了時刻推定タスクスケジューリング

入力：タスクグラフ $G' = (V, E, v_{start}, C_{comp}, C_{comm})$ とプロセッサグラフ $N' = (P, R, freq)$，途中のスケジュール結果 S'，優先度付けされたリスト L
出力：入力タスクグラフ G' に対する仮スケジュール結果

1: スケジュールの初期値を S' とする．
2: $taskI$：リスト L 内の仮割り当て時の開始ノードのポインタとする．
3: **while** $taskI! = NULL$ **do**
4: タスクノード $n_{taskI} \in L$ が最も早く処理完了できるプロセッサノード p を $findProcessor(\mathbf{P}, n_{taskI})$ を通して見つける．
5: **for** $n_i \in pred(n_{taskI})$ **do**
6: **if** $proc(n_i) \neq p$ **then**
7: $proc(n_i)$ から p へのリンク $R = [L_1, L_2, ..., L_l]$ を決定する．
8: R に e_{ij} をスケジュール
9: **end if**
10: **end for**
11: タスクノード n_{taskI} をプロセッサノード p に割り当てる．
12: $taskI$ を 1 進める．
13: **end while**
14: $adjusterTBHT(S', G, N)$ を通してターボブースト及びハイパースレッディングによる処理時間の調節を行う．
15: **return** スケジュール

Algorithm 5 adjusterTBHT 関数

入力：タスクグラフ $G = (V, E, v_{start}, C_{comp}, C_{comm})$ とプロセッサグラフ $N = (P, R, freq)$，スケジュールを終えた結果 S''
出力：ターボブースト及びハイパースレッディングによる処理時間を調節したスケジュール結果
変数 ts，te：時刻を表す変数，初期値は 0

1: **while** TRUE **do**
2: **if** 全てのタスクノードの調節が終了 **then**
3: 処理を終了する．
4: **end if**
5: 時刻 ts での各ダイの使用状況に応じた動作周波数を求める．
6: 全てのダイの中で時刻 ts 以降，最初にダイの使用状況が変化する時刻を探索し te に代入する．
7: 時刻 ts から te までの間に処理が行われるタスクノードに対し，各ダイで求めた動作周波数に基づいて処理時間等を調節する．
8: $ts = te$.
9: **end while**
10: **return** 調節したスケジュール結果

スト及びハイパースレッディングが動作した場合の動作周波数を選択し，その時間中に処理が予定されているタスクノードの処理時間等の調節を行う．まず時刻 ts 時点での使用状況を各ダイごとに把握し，それに対応した動作周波数を本研究で作成した動作周波数モデルを用いて求める (Algorithm5 の 5 行目)．その後，全てのダイで時刻 ts 以降最初に使用状況が変化する時刻 te を求める (Algorithm5 の 6 行目)．そして求めた期間 ts から te の間に処理が予定されているタスクノードの処理時間及び関連する通信時間を調節し，時刻 ts を更新する (Algorithm5 の 7-8 行目)．これらの処理を全てのタスクノードの処理時間を調節し終わるまで繰り返し，調節したスケジュール結果を返す．

6. 評価実験

処理時間の短縮率及び実機上での有効性と作成した動作周波数モデルの妥当性を評価するため，シミュレーションと実機実験による評価を行った．

既存手法は，ターボブースト及びハイパースレッディングによる動作周波数の動的変更を考慮を行っていない．本研究では比較手法として，既存手法に本研究で作成した動作周波数モデルを組み込んだ SinnenPhysical と SinnenLogical の 2 種類を用意し，それらと比較することでスケジューリングアルゴリズムによる処理時間の短縮率を評価する．

SinnenPhysical はネットワークコンテンションを考慮した Sinnen らの提案したスケジューリングアルゴリズムの割り当て対象を物理プロセッサに限定し，ターボブーストによる動作周波数の変更を考慮するため拡張したものである．ターボブーストによる動作周波数の変更を考慮するため，本研究で作成した動作周波数モデルをタスクノードを割り当てるプロセッサノードを決定する関数に組み込み，各タスクノードを割り当てる際にプロセッサノードの使用状況に応じて変化する動作周波数を考慮するように拡張した．**SinnenLogical** は，同様に Sinnen らの提案したスケジューリングアルゴリズムをターボブースト及びハイパースレッディングによる動作周波数の変更を考慮するために拡張したものであり，割り当て対象とするプロセッサが物理プロセッサだけでなく，論理プロセッサにも割り当てを行うことでハイパースレッディングによる動作周波数の変更についても考慮する．

6.1 実験設定

この実験で用いるタスクグラフとして Standard Task Graph Set[6] の Robot Control と Sparse Matrix Solver を用意した．Sparse Matrix Solver のタスクグラフは，タスク数 98，エッジ数 67 であり，Robot Control のタスクグラフは，タスク数 90，エッジ数 135 である．今回の実験で用いる実効動作周波数モデルは，使用する両タスクグラフがメモリアクセスとコンピュテーションの比が考慮していないものであるため，3 章で述べたプロセッサの状態が 4 である場合の結果を利用した．この実験では上記の 2 つのタスクグラフに対して提案手法及び比較手法 2 種でのスケジュール生成および生成したスケジュールの実行を行い，それぞれの場合でタスクグラフ全体の処理完了時刻の比較を行った．

スケジュールした結果を基に，タスクを実行する計算機

システムとして，PC4 台を Gigabit Ethernet により接続した物を使用した．計算機のスペックは，CPU:Intel Core i7 3770T(2.5GHz，物理 4 プロセッサ，論理 8 プロセッサ，シングルソケット)，Memory : 16.0GB，OS:Windows7 SP1(64bit)，Java(TM) SE (1.6.0 21，64bit)，Gigabit LAN subsystem using the IntelR 82579V Gigabit Ethernet Controller である．この計算機上で Java(TM) SE Runtime Environment(build 1.6.0_21b07, 64bit) により作成したプログラムを用いた．計算機間のネットワーク接続は TCP ソケットにより行った．

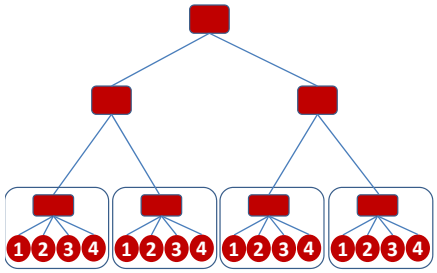
プロセッサグラフとして，シミュレーションでは上記の計算機を想定し，図 3 にあるような 3 つのネットワークトポロジを構築した．構築したトポロジはデータセンタ等の分散並列環境で主に使用されている Tree 型とスター型，フルコネクト型である．ただし，実機実験においては上記の計算機を使用し，完全型のネットワークトポロジを用意することが困難であったため，これを除く 2 種類のプロセッサグラフを構築している．各ネットワークの帯域は上記のシステムで予備実験を行って得た 420Mbps を使用した．

6.2 性能評価実験

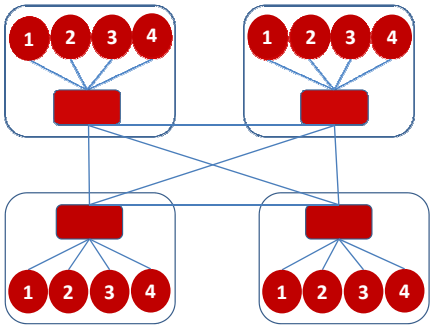
6.2.1 処理時間の短縮率

提案手法と比較手法 2 種類に対してシミュレーション及び実際の計算機上でタスクノードを実行する実機実験を行いシミュレーションと実機実験のそれぞれでどの程度処理時間を短縮できたのか評価した．シミュレーションでは，先に述べた 2 種類のタスクグラフ及び想定する 3 つのプロセッサグラフを組み合わせた 6 パターンに対してスケジュールを生成を行い，各組み合わせ毎に生成したスケジュール結果の処理完了時刻を比較した．実機実験では，提案手法及び比較手法 2 種でシミュレーションにより生成したスケジュール結果を用いて，実際の計算機上にタスクノードを割り当てて実行し，実行開始から実行完了までに掛った時間の計測及び比較を行った．この際，OS からの影響を排除するため，必要最低限なプログラム以外を極力停止させ，タスクノードの割り当てに関してもスケジュール結果通りのプロセッサノードで処理が行われるように各タスクにアフィニティーを設定する事で OS のタスクの割り当てに関するポリシーに依存しないようにした．

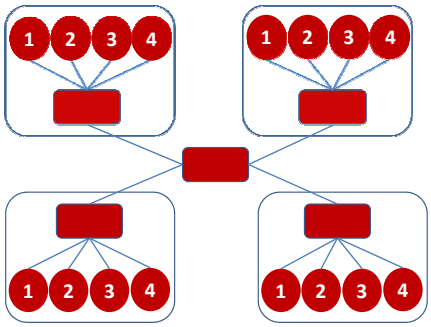
シミュレーションによるスケジュール結果の比較を図 4 に，実機実験の結果の比較を図 5 に示す．縦軸は各タスクグラフの処理時間，横軸は各プロセッサグラフの種類とタスクグラフの組み合わせを表している．これらの結果から，提案手法は比較手法と比べシミュレーションでは最大でおよそ 43% ，実機実験の結果では最大で 36% の処理時間を短縮できることを確認した．図 4 と図 5 において，各プロセッサグラフに対してスケジュールした依存関係の異なる 2 つのタスクグラフの結果を比較すると，どのプロセッサグラフにおいても依存関係の弱いタスクグラフ (Sparse Matrix Solver) と比べ依存関係の強いタスクグラフ (Robot Control) が提案手法による処理時間の短縮効率が低いことも判った．提案手法ではターボブースト及びハイパースレッディングによる動作周波数の動的変更を考慮して，データ転送による通信時間や遅延時間の増加を加味しても処理時間を短縮することができるのであれば使用状況が低いダイにタスクノードが割り当てが行われる．しかし，依存関係が強いタスクグラフでは，依存関係の弱いタスクグラフと比べて親ノードと子ノード間のデータ転送が多く，ターボブースト及びハイパースレッディングによる動作周波数の動的変更による処理時間の短縮を行おうと使用状況の低いダイ上に割り当てようとしても，データ転送による通信時間及び遅延時間が大きくなり，結果的に処理時間が増加してしまうことが多い．そのため，依存関係の強いタスクグラフでは，タスクノードの割り当て方がある程度制限されてしまい，提案手法による効果が得られなくなったからではないかと思われる．加えて，各タスクグラ



(a)Tree 型

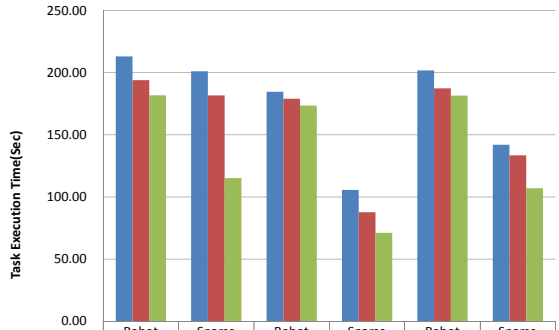


(b) メッシュ型



(c) スター型

図 3: 実験で用いるプロセッサグラフ



	Tree network		Fully connected		Star network	
	Robot	Sparse	Robot	Sparse	Robot	Sparse
SinnenPhysical	213.10	201.16	184.62	105.66	201.84	142.04
SinnenLogical	193.98	181.72	179.02	87.70	187.44	133.50
Proposed	181.80	115.19	173.50	71.10	181.60	107.00

図 4: タスクグラフの処理時間：シミュレーション

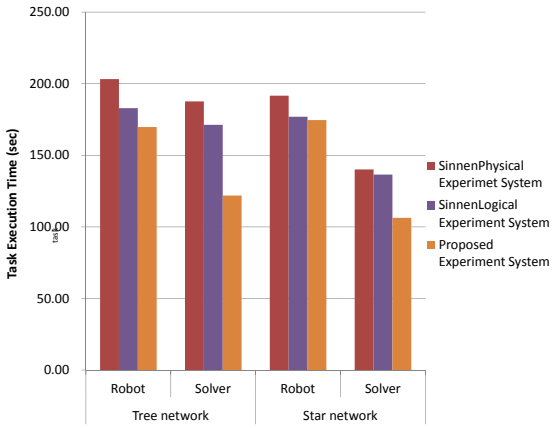


図 5: タスクグラフの実行時間：実機実験

フに対するプロセッサグラフごとのスケジュール結果に着目すると，提案手法による処理時間の短縮率は，ネットワークを介したデータ転送による遅延が少ない完全型に比べ，ある程度データ転送による遅延がある Tree 型及びスター型の方が提案手法による改善が大きいことが判った．これは比較手法が後続のタスクノードを割り当てたことによる動作周波数の変更を考慮していないため，なるべく同じダイもしくは近隣のプロセッサに割り当てようとするのに対し，提案手法はネットワークでの遅延及び割り当てたプロセッサノード上でタスクノードを処理する動作周波数を考慮した上で最も早く処理できるプロセッサに割り当てているためだと考えられる．

予備実験として，ターボブースト及びハイパースレッディングによる動作周波数の変更をまったく考慮しない既存手法との比較実験を行った．その結果，両技術による動作周波数の動的変更を考慮していない手法によるスケジュール結果は今回の実験で行った全ての場合で比較手法として用いた両手法よりも処理時間が多く掛っていた．既存手法と比較手法の SinnenPhysical の結果から，依存関係の強いタスクグラフを用いる場合では，処理時間の差が 14% であり，依存関係の弱いタスクグラフを用いた場合では，3% の差であることがわかった．これは依存関係が弱く多くのタスクノードを並列に処理できるタスクグラフにおいて，SinnenPhysical は物理コアのみに割り当てを行い，ターボブーストによる動作周波数変更の恩恵を受けているが，ハイパースレッディングによる処理時間短縮の恩恵を受けることができないのに対し，既存手法はターボブーストによる動作周波数の変更に加え，ハイパースレッディングを用いて並列処理性を上げてスケジュールができるためだと考えられる．

7. 結論

本論文では，ネットワークコンテンションが発生し，ターボブースト及びハイパースレッディングが搭載されているマルチコアプロセッサ環境を想定したタスクスケジューリングアルゴリズムを提案した．今後の課題としては，ターボブースト及びハイパースレッディングによる動作周波数モデルの妥当性の向上およびアルゴリズムの性能向上，異なるタスクグラフを用いた評価等があげられる．

参考文献

[1] Intel: “Intel Turbo Boost Technology in Intel Core Microarchitecture (Nehalem) Based Processors,” 入手先 ⟨http://download.intel.com/design/processor/applnots/320354.pdf⟩.

[2] DT. Marr, F. Binns, DL. Hill, G. Hinton, DA. Koufaty, JA. Miller and M. Upton: “Hyper-Threading Technology Architecture and Microarchitecture,” *Intel Technology Journal, Vol. 6, No. 1, pp. 4-15, 2002.*

[3] Anderson, J.H., Calandrino, J.M.: “Parallel task scheduling on multicore platforms, ” *ACM Special Interest Group on Embedded Systems(SIGBED) **3**(1), pp. 1–6, 2006.*

[4] S. Gotoda, M. Ito and N. Shibata: “Task scheduling algorithm for multicore processor system for minimizing recovery time in case of single node fault,” *International Symposium on Cluster, Cloud, and Grid Computing (CCGrid), pp. 260-267, 2012.*

[5] Sinnen, O., Sousa, L.: “Communication contention in task scheduling, ” *Parallel and Distributed Systems, IEEE Transactions on **16**(6), pp. 503–515, 2005.*

[6] Tobita, T., Kasahara, H.: “A standard task graph set for fair evaluation of multiprocessor scheduling algorithms, ” *Journal of Scheduling **5**(5), pp. 379–394, 2002.*