情報処理学会論文誌 Vol.57 No.2 657–668 (Feb. 2016)

推薦論文

マルチコア計算機のための 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

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

受付日 2015年5月5日, 採録日 2015年11月5日

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．マルチコアプロセッサの処理性能を向上させるため，ターボブーストおよびハイパースレッディングと呼ばれる処理高速化およ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とき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，1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おらず，多数のタスクを1つのプロセッサに集中させる傾向がある．本研究では，ターボブーストおよ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，より計算機資源を有効に活用するための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およ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．

キーワード：マルチコアプロセッサ，スケジューリング，同時マルチスレッディング，ターボブースト，ハイパースレッディング

Task Scheduling Algorithm Considering Turbo Boost and Hyper-Threading

Yosuke Wakisaka[1] Naoki Shibata[1,a)] Junji Kitamichi[2] Keiichi Yasumoto[1] Minoru Ito[1]

Received: May 5, 2015, Accepted: November 5, 2015

Abstract: Computing systems with multi-core processors are becoming popular. Accordingly, various task scheduling methods are utilized for making the computation more efficient. Technologies called Turbo Boost and Hyper-Threading are utilized in some of the latest multi-core processors. Turbo Boost is a technique for increasing the clock speed of some processor cores within the thermal specification when other cores are inactive. Hyper-Threading is a technology that enables the physical resources of one physical processor to be shared between two or more logical cores to improve the overall throughput. The proposed algorithm minimizes the total computation time taking account of dynamic changes of the processing speed by the two technologies, in addition to the network contention among the processors. We constructed a clock speed model with which the changes of processing speed with Turbo Boost and Hyper-threading can be estimated for various usage patterns. We then constructed a new scheduling algorithm that minimizes the total execution time of a task graph considering network contention and the two technologies. We evaluated the proposed algorithm by simulations and experiments with a real multi-processor system. We confirmed that the proposed algorithm produced a schedule that reduces the total execution time by 36% in the real system compared to conventional methods which are straightforward extensions of an existing method.

Keywords: multi-core processor, scheduling, simultaneous multithreading, Turbo Boost, Hyper-Threading

[1] 奈良先端科学技術大学院大学
Nara Institute of Science and Technology, Ikoma, Nara 630–0192, Japan
[2] 会津大学
The University of Aizu, Aizuwakamatsu, Fukushima 965–8580, Japan

[a)] n-sibata@is.naist.jp
本論文の内容は2014年5月の第159回マルチメディア通信と分散処理研究発表会で報告され，同研究会主査により情報処理学会論文誌ジャーナルへの掲載が推薦された論文である．

© 2016 Information Processing Society of Japan

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 1 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．ターボブーストおよびハイパースレッディングはインテル製のプロセッサに実装されている技術であるが，同様の技術は IBM POWER8 や Oracle SPARC T5 等でも実装されている [3], [4]．また，データセンタや，近年のスーパコンピュータは，PC やワークステーション向けの汎用プロセッサを多数組み合わせて構成されることが多く，このような大型の施設や，あるいは比較的小型のコンピュータクラスターにおいて計算効率を高めるために，ターボブーストやハイパースレッディングをはじめとした，最新の技術を効率的に利用できることが望ましい．このような，ターボブーストおよびハイパースレッディングを搭載した多数の汎用マルチコアプロセッサから構成される環境において，システムに既存のタスクスケジューリング手法（たとえば，古典的なリストスケジューリングにネットワークコンテンションのモデルを取り入れた Sinnen らの手法 [12]）を単純に適用した場合，マルチコアプロセッサどうしを結ぶネットワークを介した通信より，同一ダイ上のプロセッサコア間での通信が高速であるため，その通信性能を積極的に利用して計算時間を短縮するために，同じマルチコアプロセッサにタスクの割当てを集中させる傾向がある．しかし，ターボブーストは，全コアの総 TDP（熱設計電力）をある一定値以下に抑えるための仕組みであるため，同じダイ上の複数のプロセッサが同時に使用される状況では，各コアの動作クロックが低く抑えられてしまい，スケジューリング時に想定したとおりの性能が発揮できない．そのため，計算資源を最大限有効に利用できず，一部のコアがアイドル状態のまま他のコアでの処理が完了するのを待ってしまう問題が生じる．この場合，各タスクを異なったダイ上のプロセッサで実行した方が計算時間を短縮できる場合がある．

本論文では，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およ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，並列処理可能なタスク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 [5], [14]．提案手法が対象とするタスクの例として，大規模行列演算等があげられる．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およ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．

以降 2 章では，関連研究について述べ，提案手法の位置づけを明確にする．3 章では，ターボブーストおよびハイパースレッディングについて述べ，本研究で行った動作周波数のモデル化について述べる．4 章では，タスクスケジューリングおよび本研究での諸定義について述べ，問題の定式化を行う．5 章では作成した動作周波数モデルを用いてターボブーストおよ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，タスクの処理時間が最小となるタスクスケジューリングアルゴリズムについて述べる．6 章では既存のネットワークコンテンションを考慮した手法をターボブーストおよびハイパースレッディングに対応させるため拡張した手法と提案手法をシミュレーションと実機実験を通して比較評価し，考察を行う．最後に 7 章で結論を述べる．

2. 関連研究

Anderson 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当てを行う協調スケジューリングを提案している [6]．この手法では共有 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．

Song らは，共有メモリ型と分散メモリ型の両マルチコアシステム上で，コレスキー分解等の線形代数アルゴリズムを実行するための動的タスクスケジューリング手法を提案している [7]．この手法は優れたスケーラビリティを有しているが，対象が限定されており，そのほかのアルゴリズムに直接適用することはできない．

著者らの所属する研究グループでは，文献 [11] において，複数のマルチコアプロセッサからなる環境において，単一計算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た．この手法では，単一の計算ノードがマルチコアプロセッサで構成されると仮定しており，最大でコアと同数のタスクが単一ノードで実行される．また，タスクノードが次のタスクノードへデータを転送する際に，他のタスクノード間の転送によってネットワーク

© 2016 Information Processing Society of Japan

が利用されているとき，帯域の制限により同時に転送できないと仮定している．また，各タスクのチェックポイントが，タスク実行前に各計算ノードのメインメモリに保存されると仮定して，単一の計算ノードが停止故障し，直後にリブートした際に，チェックポイントから計算をやり直すのに必要なリカバリータイムを加味したうえで，処理時間を削減するようにタスクの割当てを行うことで，単一故障が発生した場合での全体の処理時間を最小化する．提案した手法に関してシミュレーションおよび実機実験を通して有効性を確認した．

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およ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およ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．

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 [10]．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 [12]．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．

Kwok らは，ヘテロジニアスなプロセッサ環境におけるプロセッサ間の通信コストがマルチプロセッサに比べ高い点に着目し，タスクのスケジュールだけでなく，通信経路も同時にスケジュールするアルゴリズムを提案している [8]．この手法では異なる種類のプロセッサ群に対してタスクを割り当て，必要となる通信トラフィックを考慮しつつ最速で処理完了できるようにスケジュールすることで全体の処理時間を削減できることを示している．

Tang らは，ヘテロジニアスなプロセッサネットワークを想定し，タスクの処理時間が最小となるように，タスクをプロセッサに割り当てるタスクスケジューリング手法を提案している [9]．この手法では，プロセッサの処理性能とネットワークの帯域を割当て時のパラメータとして利用することで，ヘテロジニアスな環境において，タスクの処理時間をより短縮できるプロセッサに割当てを行っている．また，この手法は並列にスケジュールを行えるように設計されており，スケジュールを生成する時間が短縮されていることも示されている．

上記のように，ネットワークの特性を考慮したタスクスケジューリング手法はすでに多数提案されている．しかし，ネットワークの特性だけでなくマルチコアの特性，およびターボブースト・ハイパースレッディングによる動作周波数の動的変更を扱ったモデルは調査した限りでは存在しない．

本研究では，ターボブーストおよ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. ターボブーストおよびハイパースレッディングのモデル化

ターボブーストおよ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，そのときどき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およ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，供給される電流およ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

© 2016 Information Processing Society of Japan

表 1 ターボブースト時の動作周波数
Table 1 Operating frequency with turbo boost.

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

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，80 MB 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（2.5 GHz，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 16.0 GB，Windows7 SP1（64 bit），Java™ SE（1.6.0 21，64 bit）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表される．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．

表 2 ハイパースレッディング時の実効動作周波数
Table 2 Effective operating frequency with hyper-threading.

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

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 1 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．論理プロセッサおよ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（**表 2**）．

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. タスクスケジューリングの定式化

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．

本章では，提案手法を説明するうえ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（**図 1** はその例）．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およ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，タスクリンクすべての集合および開始ノードをそれぞれ V，E および 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 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．たとえばタスクノード 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．

© 2016 Information Processing Society of Japan

表 3 記号表
Table 3 Symbols used in this paper.

記号	意味
G	タスクグラフ
V	タスクノードすべての集合
E	タスクリンクすべての集合
$C_{comp}(n)$	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	プロセッサノードすべての集合
R	プロセッサリンクすべての集合
$freq(s)$	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 をプロセッサノード p_i で実行する際の処理完了時刻
$t_{ef}(e_{ij})$	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かかる時間
$proc(n_i)$	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

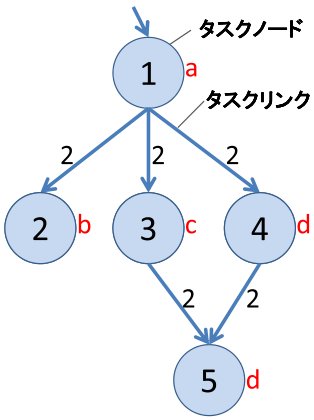


図 1 タスクグラフの例
Fig. 1 Example task graph.

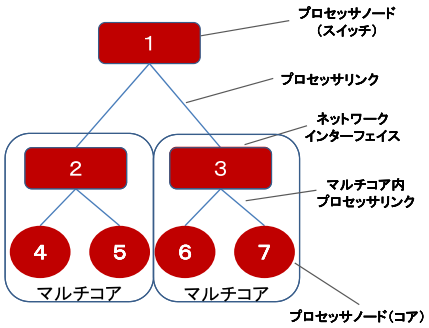


図 2 プロセッサグラフの例
Fig. 2 Example processor graph.

プロセッサグラフは，ネットワークトポロジを表現したものであり，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ースであり，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ことができる．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 のときの実効動作周波数を返す関数を $freq(s)$ とする．このとき，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かか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，リンクの帯域はすべて同じとする．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 [12] 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 1 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すべて完了し，すべ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 [12] の

© 2016 Information Processing Society of Japan

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 ら [12] 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とき，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 [12] 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 困難に属する組合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．本論文では，本問題を効率的に解くために Sinnen ら [12] 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，ターボブーストおよ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およ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およ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および，それにともなう処理開始時間等の調節を行うこと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すべてに対して行い，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．

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

また，本手法では，タスクノードの処理時間の推定をすべ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，そのなかからスケジュールに採用するプロセッサノードへの割当てパターンを選択する必要がある．選択の基準として，各パターンごとの仮割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，すべてのパターンの中で最良と思われる割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．

すべ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 Algorithm 1 に示す [10], [12]．リストスケジューリングでは，先行制約およ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でき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は，通信帯域やスイッチでの遅延，転送するデータの競合等を考慮しつつ，各手法の目的を達成するためにタスクノードの割当てを行う．提案手法のベースとなる Sinnen らの手法（Algorithm 2）では，ネットワークの輻輳のモデルとして，(1) タスクノード間の複数のデータ転送は同時に行えない，(2) 複数のリンクを経由してデータを転送する場合，下流のリンクでのデータ転送の開始時間は上流のデータ転送の開始時間より早くできない，(3) データ転送の上下問わず同一リンク上で複数の転送は同時に行えない，この 3 つの制約の下に，最も早くタスクを処理完了できるプ

© 2016 Information Processing Society of Japan

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 $R = [L_1, L_2, \ldots, L_l]$ 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

ロセッサノードを探すことで，ネットワークデータ転送の遅延および競合を考慮している．Algorithm 1 の 3 行目では，このようにしてタスクノード n が最も早く処理完了できるプロセッサノードを選択する．また，Algorithm 2 では，同様にネットワークによるデータ転送の遅延および競合を考慮している．

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およ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，ターボブーストおよ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当てを採用し，後続のタスクノードの割当てを行う．この一連の動作をすべてのタスクノードに対して行うことでターボブーストおよ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にできるスケジュール結果を作成する．

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 からすべてのタスクノード 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 $R = [L_1, L_2, \ldots, L_l]$ 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 の処理時間およ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 Algorithm 4 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およ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

提案アルゴリズムを Algorithm 3 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（Algorithm 3 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当てを行う．そして，仮ス

© 2016 Information Processing Society of Japan

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 $R = [L_1, L_2, \ldots, L_l]$ 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およ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

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当てとターボブーストおよ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当てパターンの処理完了時刻を比較し，そのなかで最良のものを該当タスクノードの割当て結果として採用する（12〜16 行目）．これをすべてのタスクノードが割り当てられるまで繰り返す（2〜18 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当ておよ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 Algorithm 4 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（Algorithm 4 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 へ順に割り当てていく（5〜11 行目）．すべ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およ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．

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：ターボブーストおよ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** すべ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: すべ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

$adjusterTBHT(S', G, N)$ は，生成したスケジュール結果の処理開始時刻，処理時間および処理完了時刻とそれにともなう通信時間をターボブーストおよ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ストおよ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（Algorithm 5 の 5 行目）．その後，すべてのダイで時刻 ts 以降最初に使用状況が変化する時刻 te を求める（Algorithm 5 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および関連する通信時間を調節し，時刻 ts を更新する（Algorithm 5 の 7〜8 行目）．これらの処理をすべ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

処理時間の短縮率およ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，著者らで作成した Java プログラムによる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，ターボブーストおよ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の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

© 2016 Information Processing Society of Japan

動作周波数モデルを組み込んだ SinnenPhysical と Sinnen-Logical 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およ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 [13] 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および比較手法 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，PC 4 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 (2.5 GHz, 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：16.0 GB，OS：Windows7 SP1 (64 bit)，Java™ SE (1.6.0 21，64 bit)，Gigabit LAN subsystem using the Intel® 82579V Gigabit Ethernet Controller である．この計算機上で Java™ SE Runtime Environment (build 1.6.0_21b07, 64 bit) 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 420 Mbps を使用した．

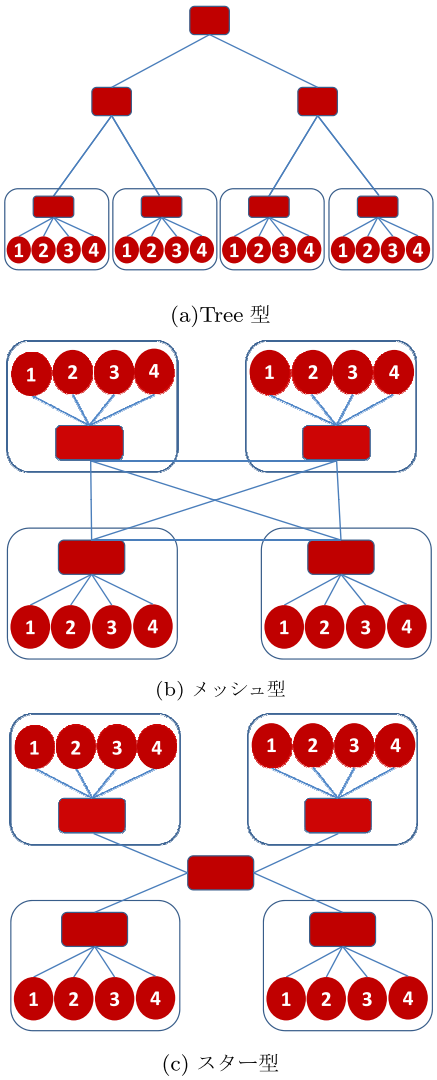


図 3 実験で用いるプロセッサグラフ
Fig. 3 Processor graphs used in evaluation.

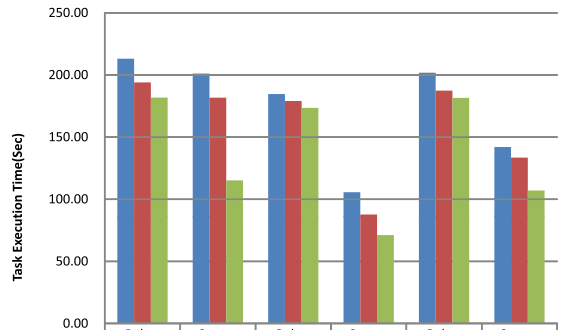
6.2 性能評価実験

提案手法と比較手法 2 種類に対してシミュレーションおよ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および想定する 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の生成を行い，各組合せごと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，提案手法および比較手法 2 種でシミュレーションにより生

© 2016 Information Processing Society of Japan

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かかった時間の計測および比較を行った．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，タスクノードの割当てに関してもスケジュール結果どお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ことでOSのタスクの割当てに関するポリシ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合せを表している．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



	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
Fig. 4 Simulated execution time of each task graph.

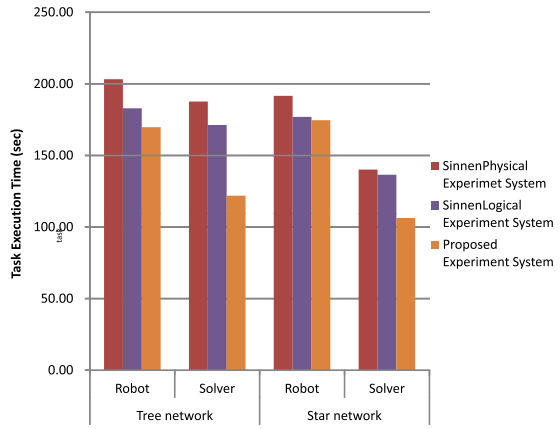


図 5 タスクグラフの実行時間：実機実験
Fig. 5 Measured execution time on real devices.

が低いことも分かった．提案手法ではターボブーストおよ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の割当てが行われる．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，ターボブーストおよ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およ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 型およびスター型の方が提案手法による改善が大きいことが分かった．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およ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うえ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，ターボブーストおよ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すべ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かかっていた．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%の差であることが分かった．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

© 2016 Information Processing Society of Japan

ボブーストおよ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，ターボブーストおよ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 (2008), available from ⟨http://download.intel.com/design/processor/applnots/320354.pdf⟩ (accessed 2015-08-30).

[2] Marr, D.T., Binns, F., Hill, D.L., Hinton, G., Koufaty, D.A., Miller, J.A. and Upton, M.: Hyper-Threading Technology Architecture and Microarchitecture, *Intel Technology Journal*, Vol.6, No.1, pp.4–15 (2002).

[3] Stuecheli, J.: POWER8 processor, *Hot Chips*, Vol.25 (2013).

[4] Feehrer, J., Jairath, S., Loewenstein, P., Sivaramakrishnan, R., Smentek, D., Turullols, S. and Vahidsafa, A.: The oracle sparc t5 16-core processor scales to eight sockets, *IEEE Micro*, Vol.33, No.2, pp.48–57 (2013).

[5] Wakisaka, Y., Shibata, N., Yasumoto, K., Ito, M. and Kitamichi, J.: Task Scheduling Algorithm for Multicore Processor Systems with Turbo Boost and Hyper-Threading, *The 2014 International Conference on Parallel and Distributed Processing Techniques and Applications* (*PDPTA '14*), pp.229–235 (2014).

[6] Anderson, J.H. and Calandrino, J.M.: Parallel task scheduling on multicore platforms, *ACM Special Interest Group on Embedded Systems* (*SIGBED*) Vol.3, No.1, pp.1–6 (2006).

[7] Song, F., YarKhan, A. and Dongarra, J.: Dynamic task scheduling for linear algebra algorithms on distributed-memory multicore systems, Technical Report, The University of Tennessee, Knoxville CS Technical Report 638 (2009).

[8] Kwok, Y.-K. and Ahmad, I.: Link contention-constrained scheduling and mapping of tasks and messages to a network of heterogeneous processors, *Cluster Computing*, Vol.3, pp.113–124 (Sep. 2000).

[9] Tang, X., Li, K. and Padua, D.: Communication contention in APN list scheduling algorithm, *Science in China Series F: Information Sciences*, Vol.52, pp.59–69 (Jan. 2009).

[10] De Micheli, G.: Synthesis and Optimization of Digital Circuits, p.207, McGraw-Hill Higher Education (1994).

[11] Gotoda, S., Ito, M. and Shibata, N.: 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).

[12] Sinnen, O. and Sousa, L.: Communication contention in task scheduling, *IEEE Trans. Parallel and Distributed Systems*, Vol.16, No.6, pp.503–515 (2005).

[13] Tobita, T. and Kasahara, H.: A standard task graph set for fair evaluation of multiprocessor scheduling algorithms, *Journal of Scheduling*, Vol.5, No.5, pp.379–394 (2002).

[14] 脇坂洋祐，柴田直樹，北道淳司，安本慶一，伊藤　実：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，研究報告マルチメディア通信と分散処理，Vol.2014-DPS-159, No.13, pp.1–9 (2014).

推薦文

当該研究報告では，近年広く使われているプロセッサに搭載されている，動作クロック等を自動的に可変させる技術について取り扱っている．この技術が用いられた場合の実行速度の変化をモデル化し，実行速度の変化を考慮した効率的なスケジュールを算出する手法を提案し，実機を使用した評価を行っている．有用性が高く，また実現可能性が高い提案である．以上より，本研究会からの推薦に値する．

（マルチメディア通信と分散処理研究会主査　重野　寛）

脇坂 洋祐

2013 年 3 月奈良先端科学技術大学院大学情報科学研究科博士前期課程修了．現在，日本オラクル株式会社に勤務．

柴田 直樹　（正会員）

1996 年大阪大学基礎工学部中退，1998 年基礎工学研究科博士前期課程修了，2001 年基礎工学研究科博士後期課程修了．2001 年奈良先端科学技術大学院大学情報科学研究科助手．2004 年 4 月滋賀大学経済学部情報管理学科准教授．2012 年 9 月より奈良先端科学技術大学院大学情報科学研究科准教授．ITS，モバイルコンピューティング，並列分散システム等の研究に従事．ACM，IEEE 各会員．

北道 淳司　（正会員）

1988 年大阪大学基礎工学部情報工学科卒業．1991 年同大学大学院博士後期課程中退．同年大阪大学基礎工学部情報科学科助手．1999 年同大学大学院基礎工学研究科講師．2000 年会津大学コンピュータ理工学部准教授．2013 年より同大学同学部教授．工学博士．ハードウェアおよび組込みシステムの形式的設計および検証，回路設計教育に関する研究に従事．

© 2016 Information Processing Society of Japan

安本 慶一 （正会員）

1991 年大阪大学基礎工学部情報工学科卒業．1995 年同大学大学院博士後期課程退学後，滋賀大学経済学部助手．2002 年奈良先端科学技術大学院大学情報科学研究科助教授，2011 年より同研究科教授．博士（工学）．モバイルコンピューティング，ユビキタスコンピューティングに関する研究に従事．電子情報通信学会，ACM，IEEE 各会員．

伊藤 実 （正会員）

1977 年大阪大学基礎工学部卒業，1979 年同大学大学院基礎工学研究科博士前期課程修了．1979 年大阪大学基礎工学部助手．1986 年大阪大学基礎工学部講師．1989 年大阪大学基礎工学部助教授．1993 年より奈良先端科学技術大学院大学情報科学研究科教授．現在に至る．工学博士．データベース理論，効率的なアルゴリズム開発等の研究に従事．ACM，IEEE，電子情報通信学会各会員．

© 2016 Information Processing Society of Japan