Vol.2012-HPC-136 No.23
2012/10/4

ターボブースト・ハイパースレッディングを考慮したマルチコアプロセッサ向けタスクスケジューリング

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

概要：本稿では，近年のマルチコアプロセッサに実装されているターボブースト及びハイパースレッディングに着目し，実験結果を元に作成した動作周波数モデルについて述べる．また，そのモデルに基づき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．提案アルゴリズムは，スケジューリングの際にタスク間の依存関係を満たした組み合わせ順列を生成する．生成した順列に対して割り当てられるプロセッサコアの使用状況を考慮し，動作周波数の決定と処理時間の算出を行う．これらのタスクノードは仮スケジュールされ，その結果に対してタスクグラフ全体の処理時間を推定し，処理時間が最も早くなるであろう結果を実際のスケジュール結果に採用する．提案手法と，既存のネットワークコンテンションのみを考慮し，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いない手法を比較し，提案アルゴリズムは全体の処理時間を最大 16%短縮できることを評価シミュレーションにより確認した．

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

Task Scheduling for Multi-core Processors Systems Considering Turbo Boost and Hyper Threading

Yosuke WAKISAKA[1,a)]　Naoki SHIBATA[1]　Junji KITAMICHI[2]　Keiichi YASUMOTO[1]
Minoru ITO[1]

Abstract: In this paper, we design an operating frequency model with Turbo Boost and Hyper-Threading Technology used in recent multi-core processors from actual measurements and propose a task scheduling algorithm considering a network contention and a dynamic frequency shift based on the designed model. When scheduling the task graph to processors, firstly, the proposed algorithm generates the combinational permutation that satisfies the dependence among the tasks. Then, the number of tasks is determined as the input parameter. In regard to the generated permutations, the proposed algorithm calculates the processing time of a task node under the defined operating frequency considering the processor usage. Then these tasks are scheduled temporarily. This algorithm estimates the processing time of the entire task graph for the temporary scheduling result and selects the best of temporary scheduling results in which the estimated processing time is the shortest. We compared the proposed algorithm and the existing task scheduling algorithms which consider the network contention and do not consider the dynamic shift of the operating frequency with Turbo Boost and Hyper-Threading Technology, the proposed algorithm reduced the total processing time by 16%.

Keywords: Task Scheduling, Multi-core Processor, Turbo Boost, Hyper Threading

[1] 奈良先端科学技術大学院大学 情報科学研究科
Graduate School of Information Science, Nara Institute of Science and Technology

[2] 会津大学大学院コンピュータ理工学研究科
Graduate School of Computer Science and Engineering, The University of Aizu

[a)] yosuke-w@is.naist.jp

1. はじめに

一つのダイ上に複数のプロセッサコアを搭載し，各プロセッサコアが独立してスレッドを実行することで，スレッドレベルの並列性を利用したマルチコアプロセッサが主流となっている．マルチコアプロセッサの動作周波数は，同時に全てのプロセッサコアで向上させると発熱が問題になる．この問題を回避するため最近の Intel 社と AMD 社製のマルチコアプロセッサにはターボ・ブースト・テクノロジー及びターボコア・テクノロジー（以下これら両方をターボブースト）[1] と呼ばれる処理高速化技術が採用されている．ターボブーストは，仕様の範囲内で一部のプロセッサコアの動作周波数を向上させ，より高速な処理を可能にする．また，Intel 社製のプロセッサにはハイパー・スレッディング・テクノロジー（以下ハイパースレッディング）[2] と呼ばれる並列処理技術も採用されており，一つの物理プロセッサコアで複数のスレッドを同時に実行することで，ハードウェア資源の効率的な使用を可能にする．近年，データセンタを始め，様々な分野で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いる．より高性能な環境をより低コストで提供するため，これら最新の技術を効果的に利用できることが望ましい．

マルチコアプロセッサを含む並列システムに一連のタスクを割り当てる処理としてタスクスケジューリング手法 [3] がある．多くのスケジューリング手法ではネットワークでの転送遅延を考慮しているが，近年のマルチコアプロセッサに採用されている並列化技術及び処理高速化技術を考慮して，スケジューリング結果にプロセッサコアの動作周波数の動的設定を組み込んだタスクスケジューリングは提案されていない．既存のシングルコアプロセッサ向けのタスクスケジューリング手法 [4]-[9] を単純にマルチコアプロセッサを含むシステムに適用した場合，ネットワーク通信よりプロセッサコア間の通信が高速であるため，1 つのマルチコアプロセッサにタスクの割り当てが集中し，計算機資源を有効に利用できないという問題がある．

本稿では，ネットワークの制約やマルチコアプロセッサ向けの高速化及び並列処理技術による動作周波数の動的変更を考慮し，タスク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し，評価する．提案するタスクスケジューリングにおいては閉路を持たない有向グラフ（Directed Acyclic Graph：DAG）によりタスクの依存関係が指定されたとき，複数のプロセッサコアをもつ各ダイにタスクを割り当てる．提案するタスクスケジューリングアルゴリズムは，Sinnen[5] ら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とし，計算機環境をマルチコアプロセッサに対応させ，各プロセッサコア（以下ではプロセッサと呼ぶ）でのタスクノードの処理に関しては上記のターボブースト・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する．提案手法による処理時間の短縮を確認するため並列度と依存関係の強さの異なる 2 つのタスクグラフを用いてシミュレーションによる比較を行った．シミュレーションでは比較手法として既存手法をターボブースト・ハイパースレッディングによる動作周波数の変化に適応させるために拡張した 2 種類の手法を用いた．比較を行った結果，提案手法は比較手法に比べ最大 16%処理時間を短縮することを確かめた．

2. 関連研究

タスクスケジューリング手法には，既存手法が数多く存在する．マルチコアプロセッサを含むシステムにおいては，計算機間でのデータ転送やネットワークの遅延等の考慮やマルチコアプロセッサの特性の考慮が必要であり，様々な環境を想定したタスクスケジューリング手法が提案されている [6], [7], [8], [9]．また，古典的な手法ではネットワークコンテンションが考慮されていないものが多く，それらの手法ではネットワークの利用による遅延のない現実的でない環境を想定している．

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，ネットワークの状況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状況を考慮したスケジューリングを行い，スケジュールの正確性や，処理効率の向上に関する議論が行われている．このモデルでは，シングルコアプロセッサを対象としており，マルチコアプロセッサを想定していない．

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タスクの実行時間がある程度大きくなるような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 [10]．この手法は優れたスケーラビリティを有しているが，特定のアルゴリズムを実現するタスクスケジューリングに限定されている．また，ネットワーク上でのデータ転送にかかる遅延や，プロセッサの動作周波数変更に関しては考慮していない．

Ahmad らは，ゲーム理論を用いたマルチコアプロセッサ向けのタスクスケジューリング手法を提案している [11]．この手法ではヘテロジニアス，ホモジニアスの両方のマルチコアプロセッサを搭載するアーキテクチャに対応したスケジューリングが扱われており，スケジューリングの目標はタスクの処理時間の最小化である．問題を協力ゲームとして定式化している．このモデルでは，割り当てに関してはマルチコアプロセッサの並列実行の特性を考慮していない．

著者らの所属する研究グループでは，文献 [12] 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及び，マルチコアプロセッサの単一停止故障を考慮したタスクのスケジューリングを行うことで故障時のリカバリータイムを最小している．この手法では，動作周波数の動的変更を考慮したタスクの割り当ては行われていない．

以上のように，既存手法ではネットワークの遅延及びマルチコアプロセッサの特性の両方を考慮しておらず，マルチコアプロセッサを含む環境では効率的な割り当てが困難になる．本稿では，ターボブーストやハイパースレッディングによるダイの動作周波数の変更を考慮したスケジューリングにより，データ通信による遅延を加味しても他のダイで実行した方が処理時間を短縮できる可能性に着目したスケジューリング手法を提案する．

3. ターボブースト・ハイパースレッディングのモデル化

本章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説明し，これらを提案手法で用いるためのモデルについて述べる．

3.1 ターボブースト

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技術である．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，同時にアクティブとなっているプロセッサ数に対し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 [1].

3.2 ハイパースレッディング

ハイパースレッディングは，物理プロセッサ内のレジスタやパイプライン回路の空き時間を有効利用するため，1物理プロセッサを複数の論理プロセッサに見せ，1物理プロセッサで複数の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し，スレッドレベルの並列性向上によって1クロックあたりの処理命令数を向上させる技術である [2]．また，ハードウェア資源を効率的に利用できるため，パフォーマンスあたりの消費電力を抑えることもできる．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，各スレッドの実行速度が低下する．本稿では，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.

3.3 両技術のモデル化

これらの技術は，ダイの使用率，供給される電力及び電流，ダイの温度など複数の要因を総合的に考慮し [1], [2], ダイ全体の設計上の仕様を超えない範囲で動作周波数を動的に変更する．本研究では簡単のため，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．本稿では，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する．

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周波数を測定し．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 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) を用いて行った．

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す．実験により，ターボブースト時の動作周波数はプロセッサの状態 (メモリアクセスの比率) に関わらず全物理プロセッサの使用状況のみに依存していることが判った．また，本稿においては，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．その結果を表2に示す.

4. 問題定義

タスクスケジューリングの入力は，後述するタスクグラフとプロセッサグラフである．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．目的関数はタスクグラフの処理完了時刻の最小化である．

4.1 諸定義

本節では，提案手法を説明する上で必要となる用語を定

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] , [3 , 1] , [3 , 1]	3.1
[4 , 1] , [1 , 1] , [1 , 1] , [1 , 1]	3.7
[4 , 1] , [4 , 1] , [4 , 1] , [4 , 1]	3.1

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

義する．また，以降で使用する記号を表 3 にまとめる．

表 3: 記号表

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
$w(n_i, p_i, s)$	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間
$t_f(n_i, p_i)$	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間
$t_{ef}(e_{ij})$	通信 e_{ij} の完了時刻
$t_{dr}(n_j, p_i)$	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
$succ(n_i)$	タスクノード n_i の子ノード
$bl(n_i)$	タスクノード n_i の実行開始時間　先行タスクの実行完了時間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

タスクグラフ

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．タスクノード及びタスクリンク全ての集合をそれぞれ V, E，開始ノードを 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を表す．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．

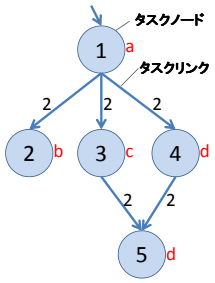


図 1: タスクグラフ

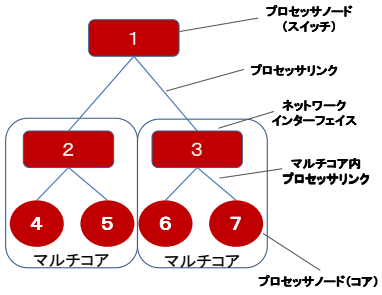


図 2: プロセッサグラフ

プロセッサグラフ

プロセッサグラフは本研究で扱う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および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出来ないものとする．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．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．

4.2 問題の仮定

本稿では，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．

本稿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) 同じプロセッサに割り当てられたタスクノードの処理はどちらかが終了するまで開始できない．(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間を先行のデータ転送の開始時間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．

4.3 問題の定式化

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 内の最遅処理完了時刻を表す．

5. 提案手法

本研究では 4 章で定義した問題を解くための基本方針を示し，ネットワークコンテンション，マルチコアプロセッサ，ターボブースト及び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アルゴリズムを示す．

5.1 提案手法の概要

本稿で扱う問題は NP 困難に属する組み合わせ最適化問題であり [5]，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算出することは困難である．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．

提案アルゴリズムでは，与えられたタスクグラフにおいて先行するタスクノードを空いているプロセッサノードの中から，そのタスクノードが最も早く実行完了するように割り当てる．しかし，その後のスケジュールによっては，割り当てたプロセッサノードが属するダイの使用率が高くなり，動作周波数が低下するため，最小時間で実行完了できない可能性がある．そのため，提案アルゴリズムでは，先行する一定数のタスクノードの順列とそのタスクノードの割り当て結果を比較して，最善のものを選ぶ方法を採用する．この手順を，以降，“先読み ”と呼ぶ．しかし，先読みされるタスク数が多くなると，タスクノードの割り当て回数が指数的に多くなり，実用的な時間でスケジューリングすることが困難となる．そこで，提案手法では先読みを行うタスク数をパラメータとして，実行時に指定するものとする．

5.2 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

提案アルゴリズムを Algorithm1 に示す．まず，定格動作周波数での全タスクノードの実行時間をもとにした実行開始時間 bl を用いてタスクノードをソートし (Algorithm1 の 1 行目)，その順にタスクノードの割り当てを行う．先読みするタスク数分のタスクノード群からなるタスクノード間の依存関係を満たした全ての順列を生成し，各順列ごとに仮割り当てを行う．その際に，ハイパースレッディングおよびターボブーストによる動作周波数の動的変更を考慮する (3 行目-8 行目)．全ての順列に対する割り当てのうち，最良のものを実際のタスクノードの割り当てとして採用する (10 行目-17 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リング結果である．

以下では，このアルゴリズムにて用いる，(1) 先読みによる割り当て順の集合 $Combi$ の生成．(2) 順列 $Combi$ 内のタスクノード n_i を割り当てるプロセッサノード p_i の選択．(3) 組み合わせ順列 $Combi$ を用いてスケジューリングを行った場合の全体の処理完了時刻 $ftime$ の推定の詳細を述べる．

(1) 先読みによる割り当て順の集合 $Combi$ の生成

先読みによる割り当て順序を生成するため，リスト L 中の処理すべきタスクノードへのポインタ $taskI$ から先読みの数 $nlookahead$ 分のタスクノード (ない場合は残りの全タスクノード) の全ての組み合わせ順列を求め，その後，タスク間の先行制約を満たすもののみを求める．求めた順列の集合を $Combi$ とする．

(2) 順列 $Combi$ を基にしたスケジューリング

Sinnen らの手法 (Algorithm2) では，先行制約及びタスクノードの実行開始時間の大きさによってソートされたリス

Algorithm 1 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アルゴリズム

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 $N = (P, R, freq)$ および先読みの数 $nlookahead$
$taskI$：リスト L の先頭
$FLOAT_{MAX}$：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: **while** $taskI != NULL$ **do**
3: $taskI$ から先読みによる割り当て順の集合 $Combi$ を得る．
4: 変数 $latestTime = FLOAT_{MAX}$．
5: **for** $c_i \in Combi$ **do**
6: **for** $n_i \in c_i$ **do**
7: 順列 $Combi$ を基にタスクノード n_i をスケジュールする．
8: 新たにタスクノードを割り当てた際に影響されるタスクノードの処理時間及び動作周波数の調節する．
9: **end for**
10: 組み合わせ順列 $Combi$ を用いてスケジュールを行った場合の全体の処理完了時刻 $ftime$ を推定する．
11: **if** $ftime < latestTime$ **then**
12: スケジュールの採用候補を入れ替える．
13: $latestTime = ftime$.
14: **end if**
15: **end for**
16: $taskI$ を $nlookahead$ 分進める．
17: スケジュールの候補を適用する．
18: **end while**
19: 最終的なスケジュール結果を返す．

ト L(Algorithm2 の 1 行目) に対して，findProcessor 関数を通してネットワークコンテンションを考慮した上で，タスクノードの実行を最も早く完了出来るプロセッサを選択する (3 行目)．その後，割り当てたプロセッサへの通信をスケジュールし (4 行目-9 行目)，後続のタスクノードの割り当てに移行する．

提案アルゴリズムでの findProcessor 関数は，さらに，プロセッサノードのターボブースト及びハイパースレッディング時の動作周波数の変更を考慮して，最も早く処理が完了するプロセッサノードを選択する．

(3) 順列 $Combi$ の処理完了時刻 $ftime$ の推定

このサブルーチンでは，スケジュール結果として採用する順列を選択するために，各順列のスケジュール結果に対してタスクグラフの処理完了時刻を推定する．スケジュールの途中結果を初期値として，残りの未割り当てタスクノードを Sinnen らの手法 (Algorithm2) を用いて仮スケジューリングを行う．仮スケジューリング後のタスクグラフの処理時間をその順列の処理完了時刻として扱う．

提案手法による処理時間短縮の例を示す．すでに 2 つのタスクノードが先行してスケジュールされている状況から，スケジュールを行うものとする．後続タスクが図 3 のタスクグラフの時，提案手法と動作周波数の変更を考慮し

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 $G = (\mathbf{V}, \mathbf{E}, w, c)$，プロセッサグラフ $H = (\mathbf{P}, \mathbf{R})$

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: **for** each $n_j \in L$ **do**
3: $findProcessor(\mathbf{P}, n_j)$ を通し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．
4: **for** $pred(n_j)$ 中の n_i **do**
5: **if** $proc(n_i) \neq p$ **then**
6: $proc(n_i)$ から p 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: n_j を p にスケジュール
11: **end for**
12: **return** スケジュール

ない手法でのスケジューリング結果をそれぞれ図 4，5 に示す．動作周波数の変更を考慮しない手法は，空いているプロセッサにタスクをスケジュールするため，タスクノードが比較的一つのプロセッサに集中していることが判る．一方，提案手法は，依存関係があるタスクノードが同じプロセッサに集まるが，一部のプロセッサにタスクが集中して割り当てられていないことが判る．そのため，提案手法の方がターボブースト及びハイパースレッディングをより効率的に利用し，処理時間を短縮することができる．

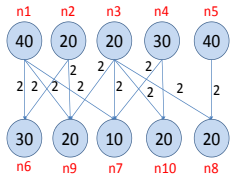


図 3: アルゴリズム実行例で用いるタスクグラフ

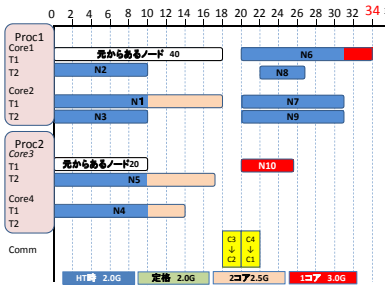


図 4: 図 3 のタスクグラフに対する動作周波数の変更を考慮しない手法によるスケジュール結果

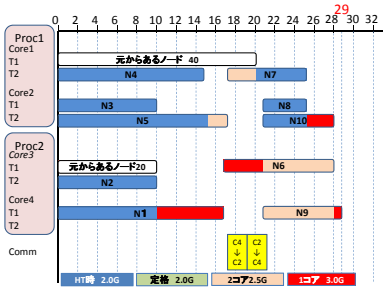


図 5: 図 3 のタスクグラフに対する提案手法によるスケジュール結果

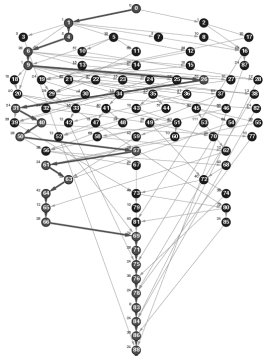
6. 評価

本章では，提案するアルゴリズムとマルチコア環境に適応させるために拡張した 2 つの既存手法をシミュレーションによりスケジュールし，結果の比較を行う．シミュレーション環境は，**3.3** 節でのプロセッサモデル作成の実験環境と同じである．

6.1 比較シミュレーション

比較手法として，Sinnen らのネットワークコンテンションを考慮した手法 [5] を基にし，**3.3** 節において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モデルを用いて，マルチコア環境へ適応させるため拡張した 2 つの手法を用いた．それぞれ，物理プロセッサのみに割り当てを行う手法と論理プロセッサに割り当てを行う手法とした．2 つの比較手法は，プロセッサを選択する際に，提案した動作周波数モデルを用いてタスクの処理時間を計算し，タスクを割り当てるプロセッサノードを決定する．タスクグラフとして Standard Task Graph Set[13], [14] の Robot Control と Sparse Matrix Solver を用いた．これらのタスクグラフを図 6 に示す．各タスクグラフのパラメータはノード数が 90 と 98，エッジの数が 135 と 67，各タスク間の通信量は 10kB であり，処理量は用意されているものを用いる．これらのタスクグラフでは，メモリアクセスとコンピュテーションの比が考慮されてないので，実効動作周波数モデルは 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する．

本シミュレーションでは，図 7 に示す 4 つのクアッドコアプロセッサと複数のネットワークスイッチから成る 3 つの異なるプロセッサグラフを用いた．(a) は Tree 型，(b) は各プロセッサを相互接続したメッシュ型，(c) は全てのプロセッサを 1 つのスイッチを用いて接続したスター型のネットワークトポロジである．プロセッサグラフにおいて，ダイ間のリンクの帯域幅は $10kB/ms$ とする．提案手法及び比較手法に対して，各ネットワークトポロジ a,b,c と，Robot Control 及び Sparse Matrix Solver の組み合わせに対してスケジューリングを行い，スケジュール結果からタスクグラフの処理完了時刻を求める．



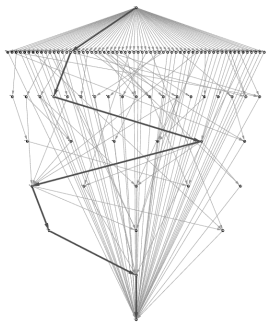
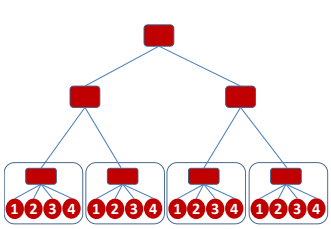
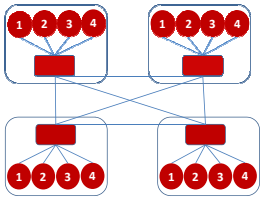


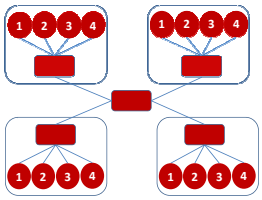
図 6: シミュレーションで用いるタスクグラフ．左：Robot Control 右：Sparse Matrix Solver



(a)Tree 型



(b) メッシュ型



(c) スター型

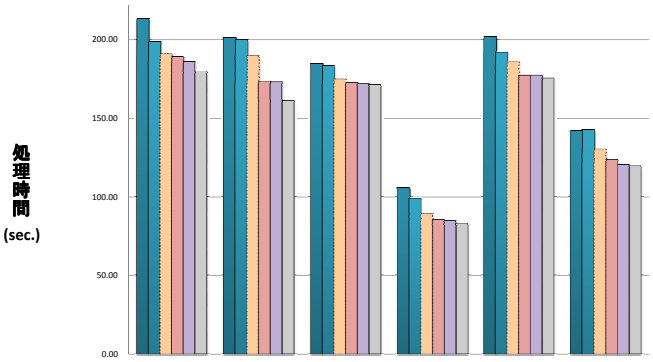
図 7: シミュレーションで用いるプロセッサグラフ

6.2 結果と考察

シミュレーションの結果を図 8 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と提案手法及び比較手法の組み合わせを表す．提案手法は比較手法よりタスクグラフの処理時間を短縮しており，最大で 16%短縮できたことが判る．

依存関係の弱いタスクグラフ (Sparse Matrix Solver) に対して提案手法を適応した場合と比べ依存関係の強いタスクグラフ (Robot Control) では処理時間の短縮効率が低いことが判った．提案手法では各ダイにある程度以上のタスクをノードを割り当てないようにしてタスクノードの割り当てを行うが，依存関係が強いタスクノードを異なるダイに割り当ててしまった場合，ターボブーストの動作周波数の向上による処理時間短縮よりもタスクノード間でのデータ転送に要する遅延時間が大きくなる．これにより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，提案手法による処理時間の短縮は，ネットワークを介したデータ転送による遅延が少ないトポロジ b に比べ，ある程度データ転送による遅延があるトポロジ a 及びトポロジ c の方が提案手法による改善が大きいことが判った．これは比較手法が動作周波数の変更を動的に考慮して

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るプロセッサの動作周波数を動的に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．提案手法によるスケジュールに必要な時間を表 4 に示す．比較手法での Robot Control と Sparse Matrix Solver のスケジュール時間がそれぞれ約 3 秒および 5 秒程度であるのに対し，提案手法では先読みを行い，細かい時間単位で動作周波数の変更を考慮しているため，時間を必要とする．しかし，スケジューリングとタスクの実行からなる全体の処理時間は短縮できている．



	Robot トポロジa	Sparse トポロジa	Robot トポロジb	Sparse トポロジb	Robot トポロジc	Sparse トポロジc
SinnenPhysicalCore	213.10	201.16	184.62	105.66	201.84	142.04
SinnenLogicalCore	198.30	199.68	183.34	99.02	191.76	142.60
提案手法 先読み1	190.71	189.54	174.57	89.10	185.79	130.02
提案手法 先読み2	188.89	173.36	172.57	85.34	177.19	123.76
提案手法 先読み3	185.72	172.82	171.77	84.68	177.17	120.50
提案手法 先読み4	179.87	161.24	171.05	83.12	175.23	119.44

図 8: スケジューリングによるタスクの処理時間

表 4: 提案手法によるスケジューリング実行時間 (sec.)

先読みの数	1	2	3	4
Robot Control	3.9	4.7	6.3	11.4
Sparse Matrix Solver	6.2	7.5	10.8	17.2

7. まとめ

ネットワークコンテンションを考慮し，マルチコアプロセッサを搭載した計算機が主となる分散システム環境でのタスクスケジューリング問題に対して，処理時間を最小化する問題を定式化した．加え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モデル化を行い，両技術による動作周波数の動的変更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した．提案手法を評価するため，既存手法をマルチコアプロセッサ環境へ適応させるために拡張した手法と提案手法で 2 種類の並列度及び依存関係の強さが異なるタスクグラフを用いて比較シミュレーションを行った結果，提案手法は，既存手法のスケジュール結果と比べタスクグラフの処理時間を最大 16%短縮することが出来た．

今後の課題として，実機を用いた環境を構築し，実機実験による提案手法の評価を行う．また，タスクノードの先読みにおいて，タスクノードの全ての順列を列挙しているので，先読みの増大により計算量が膨大になる．そのため，冗長な組み合わせの削除など先読みの候補を絞り込む手法の実装及び並列化による処理速度の向上が挙げられる．

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] Y. Kwok and I. Ahmad: “Static Scheduling Algorithms for Allocating Directed Task Graphs to Multiprocessors,” *ACM Computing Surveys, Vol. 31, No. 4, pp. 406-471, 1999.*

[4] 須田 礼仁: “ヘテロ並列計算環境のためのタスクスケジューリング手法のサーベイ,” 情報処理学会論文誌. コンピューティングシステム, *Vol. 47, No. SIG18, pp. 92-114, 2006.*

[5] O. Sinnen and LA. Sousa: “Communication Contention in Task Scheduling,” *IEEE Transactions on Parallel and Distributed Systems, Vol. 16, No. 6, pp. 503-515, 2005.*

[6] N.M. Amato and P. An: “Task Scheduling and Parallel Mesh-Sweeps in Transport Computations,” *Technical Report, TR00-009, Department of Computer Science, Texas A&M University, 2000.*

[7] 尾高 輝，甲斐 宗徳: “通信遅延を考慮したタスクスケジューリングのためのタスク粒度解析,” 成蹊大学理工学研究報告, *Vol. 44, pp. 17-23, 2007.*

[8] O. Sinnen, A. To and M. Kaur: “Contention-Aware Scheduling with Task Duplication,” *Journal of Parallel and Distributed Computing, Vol. 71, Issue 1, pp. 77-86, 2011.*

[9] O. Sinnen, LA. Sousa and FE. Sandnes: “Toward a realistic task scheduling model,” *IEEE Transactions on Parallel and Distributed Systems, Vol. 17, Issue 3, pp. 263-275, 2006.*

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

[11] I. Ahmad, S. Ranka and S. Khan: “Using game theory for scheduling tasks on multi-core processors for simultaneous optimization of performance and energy,” *Workshop on NSF Next Generation Software Program in conjunction with the International Parallel and Distributed Processing Symposium (IPDPS), pp.1-6, 2008.*

[12] 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.*

[13] T. Tobita and H. Kasahara: “A standard task graph set for fair evaluation of multi-processor scheduling algorithms,” *Journal of Scheduling, Vol. 5, pp. 379-394, 2002.*

[14] “Standard Task Graph Set”Home Page (Online) 入手先 ⟨http://www.kasahara.elec.waseda.ac.jp/schedule/⟩.