RC-007

タスク割当てアルゴリズムにおける消費電力削減のための DVS 適用タスク選択機構

森　裕一朗[†1]　朝倉宏一[†2]　渡邉豊英[†1]

プロセッサの動作周波数を低下させることにより消費電力を削減する DVS と呼ばれる技術が存在する．しかし現在では，処理速度が要求されない低負荷時の消費電力の削減に対処することが主流となっており，処理速度が要求される高負荷時の消費電力の削減にはほとんど注意が払われていない．また，既存研究では，プロセッサ間における通信コストが考慮されておらず，現実的な環境下における適切な消費電力削減手法を提供していない．本稿では，高負荷時にも DVS を適用し，全体の実行時間を増加させずに消費電力量を削減するための，タスク割当てアルゴリズムを提案する．クリティカルパス上に存在しないタスクに対し，通信コストを考慮して DVS の適用効果が最も大きいタスクから順に DVS を適用するという機構を導入することで，全体の実行時間を増加させずに消費電力を削減することが可能になる．シミュレーションによる評価実験により，プロセッサ数が 4 台の場合に平均 9.0%，プロセッサ数が 8 台の場合に平均 12.8%の消費電力削減率の向上が見られ，提案手法の有効性を確認することができた．

A Selection Algorithm of Tasks for Applying DVS toward a Power-aware Task Scheduling

YUICHIRO MORI,[†1] KOICHI ASAKURA[†2] and TOYOHIDE WATANABE [†1]

Although DVS (Dynamic Voltage Scaling) can reduce power consumption, this method is mainly used at idle time or low load computation time. In existing methods, which propose application of DVS at high load computation time, interprocessor communication cost between tasks is not considered. In this paper, we propose a task scheduling algorithm for reducing power consumption especially for high load computation time. The algorithm takes interprocessor communication cost into consideration. DVS is applied to tasks that are not in the critical path. Thus, we can reduce power consumption without increasing the whole processing time. Experimental results show that our algorithm can reduce about 9.0% and 12.8% of power consumption on 4 and 8 processors respectively on average.

1. はじめに

近年，並列処理の技術進歩が目覚ましく，サーバ計算機ではマルチ・プロセッサ環境，マルチ・コア環境が普遍的になっている．しかし，処理能力の向上と共に，消費電力も著しく増加しており，消費電力の削減は非常に重要な問題である[1]．プロセッサでの消費電力はサーバ計算機の中でも大部分を占めており，さらにプロセッサの使用率と消費電力は密接に関連している[2]．このため，プロセッサの数が多いほど消費電力は増大するので，プロセッサで消費される電力を削減することは，サーバ計算機の消費電力を削減するために非常に重要である．

プロセッサの消費電力を削減する技術として，DVS (Dynamic Voltage Scaling) と呼ばれる技術がある[3]．DVS は，動作電圧と動作周波数の組合せである P-State を動的に切り替えることで，プロセッサの処理速度と消費電力を変更する技術である．現状では，処理速度が要求されないアイドル時や低負荷時に DVS 技術を適用することで消費電力を削減する手法が一般的であり，AMD 社の Cool'in' Quiet[4] や Power Now![5]，Intel 社の SpeedStep[6] などで実現されている．しかし，DVS では処理速度を低下させることで消費電力を減少させるため，処理速度が要求される高負荷時の利用はほとんど考慮されていない．

我々は，タスク実行時にタスク単位で DVS を適用することで，実行時間を増加させずに消費電力を削減するアルゴリズムを提案する．我々のアルゴリズムは，非循環有向グラフの形式で表されるタスクグラフに対

†1 名古屋大学大学院情報科学研究科社会システム情報学専攻
Department of Systems and Social Informatics, Graduate School of Information Science, Nagoya University
†2 大同大学情報学部情報システム学科
Department of Information Systems, School of Informatics, Daido University

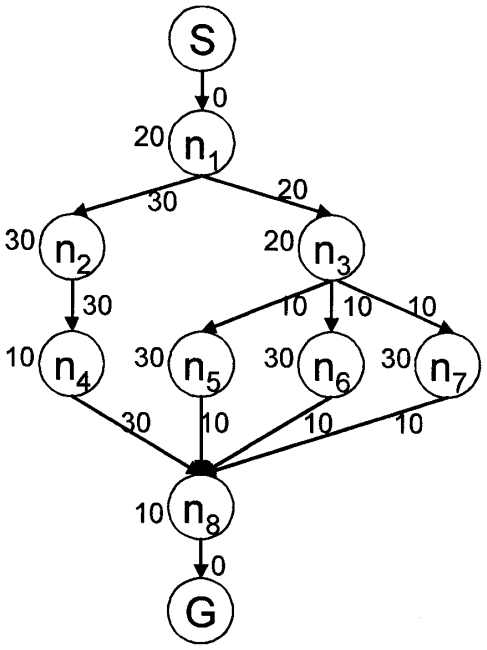


図 1　タスクグラフの例
Fig. 1　An example of a task graph

してタスク・スケジューリングを行い，生成されたスケジュールを入力とし，スケジュール長を増加させずに消費電力を削減することを目的とする．

2. 前　　提

2.1 タスク・スケジューリング

タスク・スケジューリングとは，プログラムから得られるタスクグラフに基づき，各タスクをプロセッサに割り当てる処理を表す．タスクグラフは，図 **1** のような非循環有向グラフで表現される．各タスクをノード，タスク間の通信をエッジで表し，それぞれ処理コスト $comp(n)$，通信コスト $comm(n_i,n_j)$ を持つ．図中の各ノード，エッジ付近の数字はそれぞれのコストを表し，これらのコストは既知であるとする．このタスクグラフを，様々なスケジューリング・アルゴリズム[7]により，各プロセッサに割り当てたスケジューリング結果はスケジュールと呼ばれ，本稿では図 **2** に示すガントチャートで表現する．図 2 の横軸は時間を表し，単位は [unit time] とする．縦に並ぶ矩形は各プロセッサにおけるタスクの実行状況を表している．各タスクは矩形で示され，エッジはプロセッサ間で通信が発生していることを意味している．エッジの起点は通信が開始する時間を示し，終点は通信が終了した時間を示している．

本稿で扱うタスク・スケジューリングの適用対象とする並列システムのモデルは以下の通りである．

(1)　並列システムは与えられたスケジュールの実行のみに着手できる．

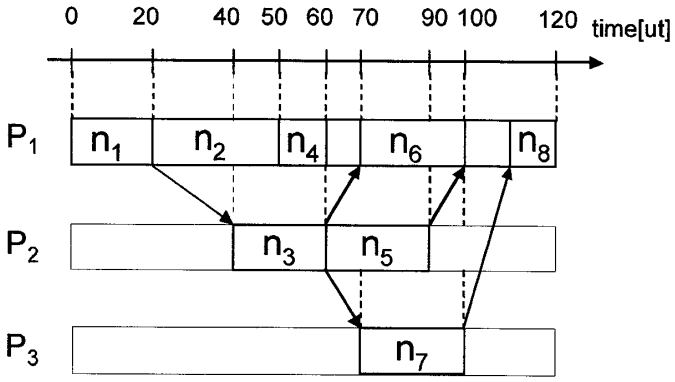


図 2　スケジュールの例
Fig. 2　An example of a schedule

(2)　スケジュールを実行する各プロセッサは，一度に一つのタスクのみ実行可能であり，それらは同質である．

(3)　同じプロセッサ上で実行されるタスク間通信のコストは 0 とする．

(4)　プロセッサ間通信においては，通信用のサブシステムが存在し，プロセッサは通信中や通信準備などの処理[8]に影響されない．

(5)　プロセッサ間通信において，同時に複数の通信が発生する場合，それらは同時に行うことができ，通信の競合[8]は発生しない．

2.2 DVS

DVS とは，プロセッサの動作周波数と動作電圧の組で表される P-State を，プロセッサの動作中に動的に変更可能とする技術である．P-State は各プロセッサに対して固有であり，通常プロセッサは離散的に複数の P-State を持つ．処理速度は動作周波数に比例し，CMOS 回路の動的消費電力は動作周波数と動作電圧の二乗の積に比例するため，これらの P-State 間の遷移により，処理速度と消費電力を変更させることができる．プロセッサの安定動作のために，動作電圧だけを下げることはできないため，動作電圧と動作周波数は同時に変更される．したがって，DVS は，処理速度を下げることで電力消費を抑える技術と言える．このように，処理速度と消費電力の間にはトレードオフの関係がある．異なる状態の P-State を s_1，s_2 とし，各 P-State に対応する動作周波数を f_{s_1}，f_{s_2}，タスクの実行時間を T_{s_1}，T_{s_2} とすると，以下の等式が成立する．

$$\frac{f_{s_1}}{f_{s_2}} = \frac{T_{s_2}}{T_{s_1}}.$$

予備実験として，表 **1** に示す計算機環境下で，各

表 1　計算機の仕様
Table 1　Specification of a computer

parts	specification
Motherboard	Gigabyte GA-K8 VM800M
CPU	AMD Athlon64 3000+
Clock frequency	2.00GHz
Number of P-States	3
RAM	1.00GB
OS	Windows XP Professional SP3

表 2　DVS の適用効果
Table 2　Effects of DVS

P-State	周波数 [MHz]	電圧 [V]	消費電力 [W] アイドル時	高負荷時
1	2000	1.50	87	100
2	1800	1.40	78	87
3	1000	1.10	59	61

P-State において計算機全体の消費電力を測定したところ，表 **2** で示される結果が得られた．本プロセッサは 3 種類の P-State を持っている．アイドル時の項目は何もタスクを実行していない状態の消費電力を示し，高負荷時の項目は高負荷タスクを実行している状態の消費電力を示す*1．DVS を適用し，周波数と電圧を低下させることで低負荷時，高負荷時共に消費電力が削減されていることがわかる．本稿では，DVS を実行中のタスク単位に適用することにより，実行時の消費電力を削減するアルゴリズムを提案する．また，本稿を通して，タスクを実行していない区間には予め DVS が適用され，最大限に消費電力が削減されていると仮定する．

3. 関連研究

DVS を用いて処理速度を考慮しながら消費電力量を削減する研究が複数存在する[9][10][11][12]．Chen らは，図 **3** に示すようなツリー型のタスクグラフをスケジューリングして得られたスケジュールを入力とし，消費電力が削減されたスケジュールを生成するアルゴリズムを提案している[9]．クリティカルパス上に存在しないタスクに注目して DVS を適用することで，全体の実行時間を増加させずに消費電力を削減することが可能になっている．本稿で提案するアルゴリズムは，Chen らの手法を基とし，全体の実行時間を増加させないよう DVS を適用するタスクを選択する機構を備えるよう拡張している．

Chen らの手法をツリー型のタスクグラフだけでなく，非循環有向グラフで表現されるタスクグラフにまで対象を拡張した手法が木村らにより提案されている[10]．木村らの手法では，Chen らと同様，タスクが実行されていない処理待ち時間に着目している．実行終了後に処理待ち時間があるタスクに対し，処理待ち時間内で DVS を適用することにより，消費電力の削減に貢献している．しかし，両者の手法とも，通信コストについては考慮されておらず，プロセッサ間通信に対して通信コストが存在する環境下では適切な DVS 適用法が提供できない．本稿で提案するアルゴリズムでは，通信コストが発生する環境下でも，通信コストを考慮して DVS を適用する．また，4.3 節で述べる DVS 適用タスク選択機構を備えることで，消費電力のさらなる削減に貢献している．

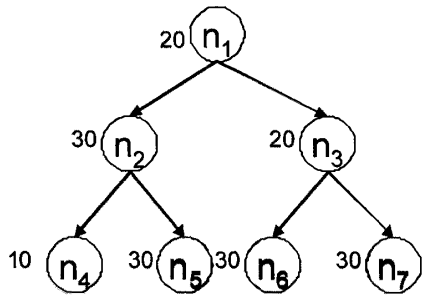
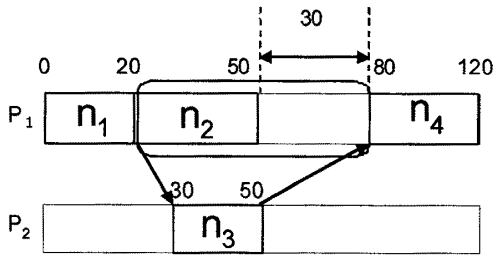


図 3　ツリー型タスクグラフ
Fig. 3　A tree task graph

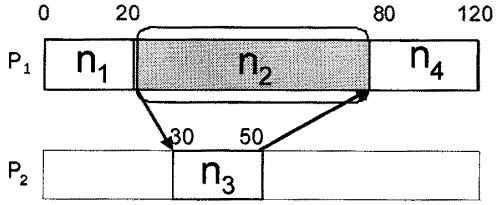
Varatkar らは，プロセッサ間通信において，通信量に応じて電力が消費されるというモデルの下でのタスク・スケジューリング・アルゴリズムを提案している[11]．タスクグラフ全体の実行時間に対して，ある制限時間が決定されたときに，その制限時間を越えないように，プロセッサ間の通信量が削減されるようスケジューリングを行い，DVS を適用することにより消費電力の削減に貢献している．この手法では，通信量の削減をすることが主眼となっており，DVS の適用方法についてはほとんど触れられていない．また，通信量による実行時間の遅延は考慮されておらず，現実的なモデルとは言えない．

堀田らは，電力性能を最適化するために，プログラムを複数の領域に分割し，領域ごとに適切な動作周波数で実行するアルゴリズムを考案している[12]．アプリケーションを予め様々な動作周波数で実行させプロファイル情報を収集することにより，各領域の実行に最適な動作周波数を決定している．この手法では，予め様々な周波数で実行することで各領域の実行時のプロファイル情報を取得しなければならないので，手法の適用に前処理が必要である．

*1 高負荷タスクとして，円周率を計算するプログラムを用いた．実行時の CPU 使用率は 100%であった．



(a) DVS 適用前



(b) DVS 適用後

図 4　DVS 適用例

Fig. 4　An example of applying DVS

4. 消費電力削減手法

本章では，並列プログラムに対し，DVS を用いて消費電力を削減する手法について述べる．非循環有向グラフで表現されるタスクグラフに対し，適当なアルゴリズムによりタスク・スケジューリングを適用して得られたスケジュールを対象とし，スケジュール長を増加させずに消費電力を削減する手法を提案する．

本手法ではタスク間の処理待ち時間に着目する．タスクの実行を遅延させても全体の実行時間を増加させない許容時間（以下，slack-time と呼ぶ）を各タスクごとに算出し，その slack-time 内で，DVS の削減効果が最も高くなるようなプロセッサの P-State を選択することで消費電力を削減する．

簡単なスケジュールに対して DVS を適用した例を図 **4** に示す．図 4 では，表 2 に示すプロセッサの下で，タスク n_2 に対して DVS を適用した例を示している．n_2 の実行終了後に処理待ち時間が存在し，slack-time は 30[ut] である．したがって，DVS が適用可能であり，タスク n_2 実行時の動作周波数を 2000MHz から 1000MHz に低下させる．このため実行時間が二倍に増加するが，slack-time 内で収まっているため，全体の実行時間を増加させない．このとき，適用前後の消費電力量を表 2 に基づいて計算すると，DVS 適用前は 4,770[W·ut], DVS 適用後は 3,660[W·ut] となり，DVS を適用することで消費電力量が削減されていることがわかる．

また，提案手法では，DVS 適用タスク選択機構を導入する．プロセッサの動作周波数と動作電圧の組を示す P-State が離散的であることと，その P-State によって消費電力の削減度合が異なることを利用する．同プロセッサ内で実行されるタスク群に対して，DVS の適用可能候補である，slack-time が存在する隣り合っている連接タスク群を抽出する．連接タスク群についての説明を，図 **5** に示すスケジュールの一部分を用いて示す．図 5 において，タスク n_1, n_2, n_4, n_5, n_6 に slack-time が存在し，タスク n_3, n_7 に存在しないとするとき，n_1, n_2 の組，n_4, n_5, n_6 の組がそれぞれ連接タスク群として抽出される．それぞれの連接タスク群中で DVS を適用するタスクを決定するとき，最も消費電力削減効果が高いタスクから順に DVS を適用することにより，既存手法よりも多くの消費電力の削減を狙う．

図 **6** に，本稿で提案する消費電力削減アルゴリズムを示す．図中の標記のうち，$est(n,p)$ は，タスク n のプロセッサ p 上での最早実行開始時間 (earliest start time) を，$ect(n,p)$ はタスク n のプロセッサ p 上での

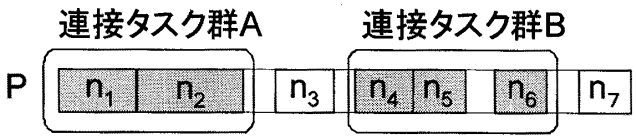


図 **5**　連接タスク例

Fig. 5　An example of connective tasks

Require: schedule: a scheduling result by non-power-aware scheduling algorithm
Ensure: schedule: a scheduling result
1: **begin**
2: **for** each processor p **do**
3: **for** each task n in descending order of $est(n,p)$ **do**
4: **if** $(slacktime(n) > 0)$ **then**
5: add n to $nlist$.
6: $n_{prev} \leftarrow$ a previous task of n.
7: **while** $(slacktime(n_{prev}) > 0)$ **do**
8: add n_{prev} to $nlist$.
9: $n_{prev} \leftarrow$ a previous task of n_{prev}.
10: **end while**
11: **while** ($nlist$!= $\phi \vee$ all tasks of $nlist$ can not be applied DVS) **do**
12: $n_{target} \leftarrow argmax(n$ in $nlist)$ $f(n)$, where $f(n)$ means the amount of power reduction for n.
13: apply DVS to n_{target}.
14: adjust est and compute slack-time for every task.
15: remove n_{target} from $nlist$.
16: **end while**
17: **end if**
18: **end for**
19: **end for**
20: **end**

図 **6**　提案アルゴリズム

Fig. 6　Our proposed algorithm

最早実行終了時間 (earliest completion time) を，それぞれ表している[7]．これらの値はアルゴリズムの入力であるスケジュールから取得される．

我々の提案する消費電力削減アルゴリズムは，タスク・スケジューリング・アルゴリズムによって生成されたスケジュールを入力とし，

(1) DVS 適用候補タスクリスト生成
(2) DVS 適用タスクの決定・適用
(3) 実行開始時間と slack-time の更新

の三つのフェーズを各プロセッサに割り当てられているタスクに対して適用することにより，実行時の消費電力量が削減されたスケジュールを生成する．なお，スケジュールにおける各タスクの所要時間は，タスクグラフにおける各タスクの処理コストとして与えられ，既知とする．

4.1 Slack-time の計算

タスク n に対する slack-time を計算する手続き $slacktime(n)$ の計算方法について説明する．まず，タスク n の後続タスクの処理待ち時間をそれぞれ計算する．タスク n に対する後続タスクとは，スケジュールにおいて，同一プロセッサ中でタスク n の次に実行されるタスクか，タスク n を起点としたプロセッサ間通信における通信先のタスクである．後続タスク n_{s1} が他プロセッサ p_{s1} に割り当てられているとき，処理待ち時間 $wait(n,n_{s1})$ は以下のように定義される．

$$wait(n,n_{s1}) = est(n_{s1},p_s) - ect(n,p) - comm(n,n_{s1}).$$

後続タスク n_{s0} が同一プロセッサに割り当てられているときは，$slacktime(n_{s0})$ を用いて定義される．

$$wait(n,n_{s0}) = est(n_{s0},p) - ect(n,p) + slacktime(n_{s0}).$$

後続タスクの $slacktime(n_{s0})$ も含めることで，連接タスクの slack-time を有効に利用することが可能となる．これらより，タスク n の slack-time は以下のように定義される．

$$slacktime(n) = \min_{n_s \in succ(n)} wait(n,n_s).$$

なお，タスク n に後続タスクが存在しないときは $slacktime(n) = 0$ と定義する．slack-time の計算をガントチャート上に表したものを図 7 に示す．

4.2 DVS 適用候補タスクリスト生成

あるプロセッサで実行されるタスク群に対して，最早実行開始時間が遅いタスクから順に，slack-time を計算する．タスク n の slack-time が正であれば，DVS 適用候補タスクリスト $nlist$ に n を加える (図 6 の 4〜5 行目)．その後，プロセッサ p 上での n の先行タスクである n_{prev} に着目する．$slacktime(n_{prev})$ が正であれば n_{prev} を $nlist$ に加え，n_{prev} の先行タスクに対して slack-time を計算する．これを先行タスクが存在しないか，先行タスクの slack-time が存在しなくなるまで続ける (6〜10 行目)．

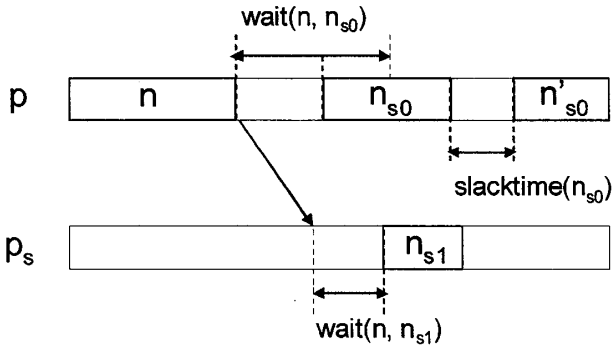


図 7 slack-time の計算方法
Fig. 7 A computation method of slack-time

4.3 DVS 適用タスクの決定・適用

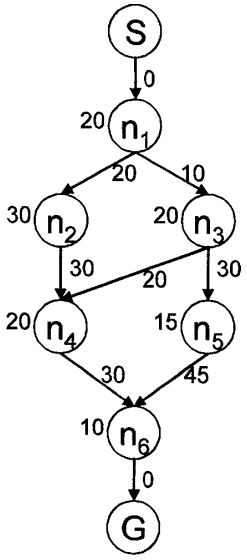
抽出された $nlist$ に含まれるタスクに対し，それぞれのタスクが持つ slack-time 内で，最も消費電力が削減される P-State で処理したときに削減される消費電力量を計算する．削減消費電力量が最大となるタスクを DVS 適用タスク n_{target} として選択し，n_{target} に DVS を適用する．DVS を適用することによって変化する n_{target} の実行時間と消費電力を更新する (12〜13 行目)．

4.4 実行開始時間と slack-time の更新

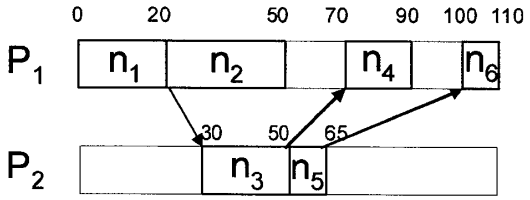
n_{target} の slack-time と，n_{target} の実行時間の増加に影響される他のタスクの est と slack-time を更新する (14 行目)．その後 n_{target} を $nlist$ から削除し，再び DVS 適用タスクの決定・適用フェーズを行う．$nlist$ の要素がなくなるか，$nlist$ 中の全てのタスクに対して DVS の適用が不可能な状態になるまで繰り返す．

4.5 議　　論

木村らの手法と比較して提案手法が有効であることを示す．3 章で述べたように，木村らの手法では通信コストが考慮されていないので，通信コストが存在する状況下でも適用できるように木村らの手法を拡張したものと提案手法を比較する．具体例として，図 8(a) に示す非循環有向タスクグラフに対して，タスク・スケジューリング・アルゴリズムである ETF アルゴリズム[7] を適用して得られた図 8(b) のスケジュールを考える．図 8(b) に示すスケジュールに対して，表 2 に



(a) タスクグラフ



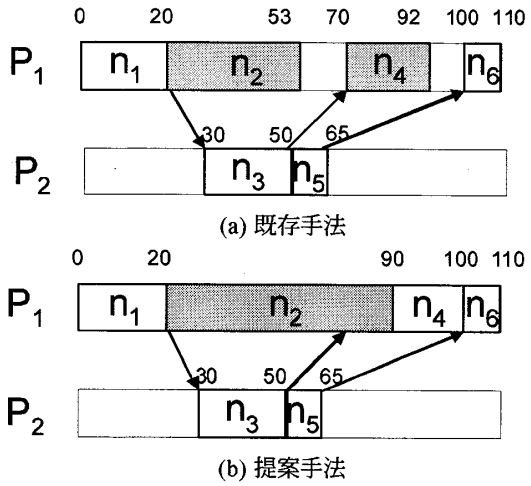
(b) スケジュール

図 8　適用対象例

Fig. 8　Examples of a task graph and a schedule

示すプロセッサの下で，木村らの手法を適用後のスケジュールを図 9(a) に示す．灰色のタスクが DVS を適用されたタスクを表す．木村らの手法では，slack-time があるタスクが連接している場合，実行開始時間が遅いタスクから順に DVS の適用可能性を判定するので，結果的にそれらのタスクに平均的に DVS を適用する．しかし，プロセッサは様々な動作周波数や動作電圧の P-State を持つため，必ずしも実行開始時間が遅いタスクから順に DVS を適用することが消費電力の削減に大きく貢献するわけではない．また，P-State は離散的な値をとり，任意の値に設定できないため，図 9(a) のように，手法適用後に slack-time が残ってしまう場合がある．

我々は，DVS 適用タスク選択機構を導入することで，手法適用後の slack-time を可能な限り減らし，かつ消費電力を大きく削減する．提案アルゴリズムの DVS 適用タスク決定・適用フェーズで述べたように，我々の手法では，slack-time が存在するタスクが連接している場合，それらのタスクの中で DVS の適用効果が大きいタスクを選択し，そのタスクに対して DVS を適用する．適用後にまだ DVS を適用できる slack-time を持つタスクが残っている場合，そのタスクに DVS を適用する．この手順を，DVS が適用できるタスク



(a) 既存手法

(b) 提案手法

図 9　消費電力削減手法適用後

Fig. 9　After applying the methods for reducing power consumption

がなくなるまで繰り返す．図 8(b) に示すスケジュールに対して，表 2 に示すプロセッサの下で，提案手法を適用した結果を図 9(b) に示す．図 9(a) のようにタスク n_2, n_4 それぞれを平均的に動作周波数を低下させて実行するよりも，タスク n_2 に対してより低い動作周波数で実行させた方が消費電力の削減に大きく貢献する．

図 9 の P_1 における [20,100] の区間に対し，表 2 の消費電力を基に消費電力量を計算すると，既存手法では 6,770[W·ut]，提案手法では 5,270[W·ut] となり，提案手法の方が削減効果が大きいことがわかる．

5. 評価実験

5.1 実験方法

提案手法の消費電力削減効果をシミュレーションにより評価する．ETF アルゴリズムのスケジューリング結果を対象に本手法を適用する．

スケジューリングの対象となるタスクグラフは，STG(Standard Task Graph set)[13] に含まれているタスク数 50, 100 のタスクグラフ各 180 種類を利用し，コストの単位は [ms] とした．STG 内のタスクグラフには通信コストが設定されていないので，通信コストを CCR(Communication to Computation Ratio)[7] と呼ばれる，タスクグラフの通信コストと処理コストの比を表す指標を用い，各々のタスクグラフに対して 10 通りの通信コストをランダムに設定した．CCR はタスクグラフの性質を表す指標であり，タスクグラフの全通信コストの和を全計算コストの和で割った値と定義されている．CCR が 0.1 の場合はタスクグラフの通信処理が少なく，1 の場合は中程度，10 の場合は多いと定義されている[7]．本実験では，CCR がこの 3 種類の値

表 3　プロセッサの電力消費モデル
Table 3　Power consumption model of the processor

P-State	周波数 [MHz]	電圧 [V]	消費電力 [W] アイドル時	消費電力 [W] 高負荷時
1	2000	1.50	40	52
2	1800	1.40	30	39
3	1000	1.10	10	13

となるように通信コストを設定し，通信量の多少による提案手法への影響を調査した．

本実験で用いる処理プロセッサは表 1 で示された AMD Athlon64 3000+であるとし，想定環境として，このプロセッサが複数台積まれたマルチプロセッサ環境のサーバを考える．実験では，4 台搭載時と，8 台搭載時の 2 種類の環境を想定する．

5.2　電力消費モデル

表 2 に示した結果と，プロセッサの動的消費電力 P に対し，動作周波数 f と動作電圧 V の間に $P \propto fV^2$ が成立することから，アイドル時にプロセッサ以外で消費される電力は約 49[W] と計算される．この値と，表 2 の結果から，動作周波数を 2000MHz から 1800MHz に低下させたときのプロセッサの消費電力は 3/4 になり，2000MHz から 1000MHz に低下させたときのプロセッサの消費電力は 1/4 に減少するというモデルで考え，以下の**表 3** に示す電力消費モデルの下で消費電力量を計算する．

一般に，DVS における周波数切替えにはオーバヘッドが伴う．例えば，Intel Speed Step では 10[μs] であり，一般的に数十 μs 程度と扱われている[14]．本評価実験においてオーバヘッドを 30[μs] と仮定すると，STG における平均タスク実行時間が 3.1[ms] であるため，その影響は 1%未満である．したがって，本稿では DVS のオーバヘッドは非常に微少であり，無視できるものとする．

5.3　実験結果

上述した電力消費モデルに基づき，ETF アルゴリズムを適用したスケジュールに対して消費電力削減アルゴリズムを適用した結果を**表 4** に示す．表中の値は手法適用前からの電力削減率の平均値を示している．表 4 には，木村らの手法適用後の効果と，我々の手法適用後の効果を示す．

実験結果より，プロセッサ数，CCR，タスク数によらず，我々の提案アルゴリズムは木村らの手法より消費電力量削減効果の向上が得られていることがわかる．また，平均して CCR が低い方が木村らの手法と比較してその向上が顕著であることが伺える．これは，CCR が低いと slack-time も小さくなるため，DVS を適用可能なタスクに対して平均的に DVS を適用するという木村らの手法では，P-State の制約を大きく受けてしまったためと考えられる．我々の手法では，P-State の制約内で DVS 適用タスク選択機構により，slack-time を最大限に活かすという性質のため，大きな向上が得られたと考えられる．また，プロセッサ数の多い方が提案手法適用前からの消費電力量削減率が大きかった．これは，プロセッサ数が多いため多くの通信が発生することにより，slack-time を持つタスクが増加し，その slack-time も増加したためと考えられる．

6. おわりに

本稿では，プロセッサの省電力技術である DVS を用い，DVS 適用タスク選択機構を導入することで，消費電力量を削減するタスク割当てアルゴリズムを提案した．我々のアルゴリズムでは，非循環有向グラフで表現されるタスクグラフに対し ETF アルゴリズムなどのタスク・スケジューリング・アルゴリズムを適用して得られたスケジュールを入力とし，消費電力量が削減されたスケジュールを出力する．並列処理の利点である処理速度の速さを悪化させないよう，スケジュール長を増加させないように DVS を適用することで消費電力の削減に貢献した．我々の提案した消費電力削減アルゴリズムでは，

(1)　DVS 適用候補タスクリスト生成，
(2)　DVS 適用タスクの決定・適用，
(3)　実行開始時間と slack-time の更新，

の三つのフェーズから構成されていて，各プロセッサの各タスクごとにこれらのフェーズを繰り返し適用する．シミュレーションによる評価実験により，プロセッサ数が 4 台の場合に平均 9.0%，プロセッサ数が 8 台の場合に平均 12.8%の消費電力削減率の向上が見られ，提案手法の有効性を確認することができた．

今後の課題として，DVS 適用時のオーバヘッドの考慮と実機での検証の二つが挙げられる．前者に関して，本稿ではオーバヘッドが微少であるとしてその影響を無視したが，扱うプログラムのタスク粒度によってはオーバヘッドが無視できない状況になることも十分に考えられる．細粒度のタスクが多く含まれるスケジュールに対しても本手法が適用できるように，DVS のオーバヘッドを考慮した消費電力削減アルゴリズムを考えることが必要である．また，後者に関して，本稿では，シミュレーションによる性能評価であるため，実際の計算機でプログラムを実行することによる消費電力量の削減効果を検証していない．検証方法として，実際の並列プログラムに対し，文献 15) にあるように，

表 4　実験結果
Table 4　Experimental results

		タスク数 50		タスク数 100	
プロセッサ数	CCR	木村らの手法	提案手法	木村らの手法	提案手法
4	0.1	7.1%	10.8%	4.8%	9.4%
	1	6.9%	10.8%	5.0%	9.3%
	10	10.3%	11.0%	9.0%	13.8%
8	0.1	11.3%	15.7%	8.9%	14.3%
	1	10.8%	14.7%	10.6%	16.5%
	10	10.8%	13.3%	11.2%	15.1%

MPI ルーチン間など，DVS ルーチンを適切な箇所に挿入することによる実現が考えられるため，それらの手法を実装し，実機で検証することが今後の課題である．

参 考 文 献

1) L.A.Barroso: “The Price of Performance”, *Queue*, Vol.3, No.7, pp.48–53 (2005).
2) X.Fan, W.D.Weber and L.A.Barroso: “Power Provisioning for a Warehouse-sized Computer”, *Proc. of the 34th Annual Int'l. Symp. on Computer Architecture*, pp.13–23 (2007).
3) Y.Zhang, X.S.Hu and D.Z.Chen: “Task Scheduling and Voltage Selection for Energy Minimization”, *Proc. of the 39th Conf. on Design Automation*, pp. 183–188 (2002).
4) AMD Corporation. Cool'in Quiet technology. `http://www.amd.com/`
5) AMD Corporation. Power now! technology. `http://www.amd.com/`
6) Intel Corporation. SpeedStep technology. `http://www.intel.com/`
7) O.Sinnen: *“Task Scheduling for Parallel Systems”*, Wiley-Interscience (2007).
8) O.Sinnen, L.A.Sousa and F.E.Sandnes: “Toward a Realistic Task Scheduling Model”, *IEEE Transactions on Parallel and Distributed Systems*, Vol. 17, No.3, pp.263–275 (2006).
9) G.Chen, K.Malkowski, M.Kandemir and P.Raghavan: “Reducing Power with Performance Constraints for Parallel Sparse Applications”, *Proc. of Workshop on High-Perfomance, Power-Aware Computing (IPDPS)*, pp.231–250 (2005).
10) 木村英明，佐藤三久，堀田義彦，朴泰祐，高橋大介: “DVS 制御による負荷不均衡のある並列プログラムの電力削減手法”, 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.SIG12(ACS15), pp.285–294(2006).
11) G.Varatkar and R.Marculescu: “Communication-Aware Task Scheduling and Voltage Selection for Total Systems Energy Minimization”: *Proc. of 2003 Int'l. Conf. on Computer-aided design (ICCAD)*, pp. 510–517 (2003).
12) 堀田義彦，佐藤三久，木村英明，松岡聡，朴泰祐，高橋大介: “PC クラスタにおける電力実行プロファイル情報を用いた DVS 制御による電力性能の最適化”, 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. SIG12(ACS15), pp.272–284(2006).
13) T.Tobita and H.Kasahara: “Performance Evaluation of Minimum Execution Time Multiprocessor Scheduling Algorithms Using Standard Task Graph Set”, *Proc. of 2000 Int'l. Conf. on Parallel and Distributed Processing Techniques and Applications (PDPTA)*, pp.745–751 (2000).
14) 宮川大輔, 石川裕: “プロセス単位電力制御機構の予備評価”, 情報処理学会研究報告（システムソフトウェアとオペレーティング・システム），Vol.2005, No.79, pp.65–72 (2005).
15) L.Choy, S.G.Petiton and M.Sato: “Toward Power-Aware Computing with Dynamic Voltage Scaling for Heterogeneous Platforms”, *Proc. of 2007 Int'l. Conf. on Cluster Computing*, pp.550–557 (2007).