社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2006－ARC－167 (26)
2006－HPC－105 (26)
2006／2／28

DVS制御による負荷不均衡のある並列プログラムの電力量削減手法

木村英明† 佐藤三久†† 堀田義彦††
朴泰祐†† 高橋大介††

PCクラスタで用いられるプロセッサにおいても，周波数と電圧を動的に変更するDVS（Dynamic Voltage Scaling）機構が利用できるようになってきた．負荷に不均衡のある並列プログラムにおいてタスク間の同期待ちの際に余裕時間が存在する場合，DVS機構を用いて周波数を適切に選択することでシステム全体の性能を低下することなく電力量を削減することができる．本論文では，非循環有向タスクグラフ（DAG）で表現される並列プログラムに対し，DVSを用いて平均的に周波数と電圧を下げることによって同期待ちの余裕時間を削減し，電力量を削減するアルゴリズムを提案する．電力を実行時にモニタ，制御するシステムを開発し，マスタ・ワーカ，ツリー型のタスクグラフを持つ実プログラムに適用しその有効性を検証した．その結果，提案アルゴリズムを適用することでアルゴリズム未適用時と比較して1%未満の性能低下で最大18.5%の電力量を削減できることが分かった．

Reducing energy of parallel programs with load imbalance by using DVS

HIDEAKI KIMURA,† MITSUHISA SATO,†† YOSHIHIKO HOTTA,††
TAISUKE BOKU†† and DAISUKE TAKAHASHI††

Recently, modern microprocessors used in PC clusters have DVS (Dynamic Voltage Scaling) mechanism which enable us to change its voltage and frequency. When there is the slack time to wait for synchronization between tasks in the execution of the parallel program, we can reduce the power by selecting an appropriate frequency by using DVS mechanism to run the tasks, without performance loss. In this paper, we propose an algorithm for directed acyclic task graph (DAG) of the parallel program to reduce the power by using DVS to slowdown the frequency seamlessly, removing the slack time for synchronization. We have developed a system for monitoring power and controlling the DVS. In our experiment, we demonstrate the effectiveness of our algorithm for master-worker and tree-task parallel programs. We found that our algorithm can reduce the power by up to 18.5% with only 1% performance loss.

1. 序　論

近年，PCクラスタ等の高性能計算システムの消費電力や電力量が増加している．消費電力の増加に伴う発熱の増加により，信頼性の低下や実装密度が低下するなどのさまざまな問題が発生する[4]．特にプロセッサの消費電力が急増しており，これがシステムの電力量を増加させる主な要因となっている．

プロセッサの周波数と電圧を動的に変更するDVS（Dynamic Voltage Scaling）機構を搭載するプロセッサが増えている．この機構を用いてプロセッサの周波数と電圧を適切に変更することにより，プロセッサの消費電力や電力量を削減することが可能である．高性能計算分野で頻繁に使用するPCクラスタにおいてもDVSが利用可能になってきている．

一方，PCクラスタ等の並列システム上においてプログラムを実行する時に待ち合わせに対する余裕時間が生じる可能性がある．この問題はタスクを均一に分配することができない時に発生する．このような問題に対し，従来は優れたタスクスケジューリングを行うことで全体の実行時間の最適化を行うとともに待ち時間を削減し負荷の均衡を図ってきた．しかしながら，最適化を行った後に待ち合わせのための余裕時間が出現する場合がある．

同期を必要とするタスク間で待ち時間が発生した場合，早くタスクを終了したノードは遊休状態となり他ノードの処理の終了を待たねばならない．どれほど早くタスクを終了したとしても，同期により後続の処理を早めることはできない．言いかえれば，同期までにタスクを終了していれば後続のタスク処理に影響を与えず，システム全体の性能は低下しない．

そこで，待ち時間のある並列プログラムに対して実

† 筑波大学 第三学群 情報学類
College of Information Sciences, Third Cluster of Colleges, University of Tsukuba
†† 筑波大学大学院 システム情報工学研究科
Graduate School of Systems and Information Engineering, University of Tsukuba

行時間を増加させずに電力量を削減する手法が考えられる．本論文では非循環有向グラフで表現されるタスクグラフを用いた並列プログラムに対し，実行時間を増加させずに電力量を削減するアルゴリズムを提案する．さらに，本アルゴリズムをマスタ・ワーカ型プログラムとツリー型のタスクグラフとなるプログラムに対して適用した．クラスタを構成する各ノードの消費電力を測定する電力測定環境を構築すると共に，電力量を削減するアルゴリズムを適用した並列プログラムをクラスタ上で実行し評価した．

Guangyu Chen らはツリーを構成する並列プログラムに対して電力量を削減する手法[1] について述べている．これは同期を必要とするノードにおいて与えられたタスクの実行時間を予測し，各ノードの周波数をクリティカルパスに合わせるよう変更するものである．我々の提案するアルゴリズムはこのアルゴリズムの拡張となっている．[1] では実機による電力測定は行われておらず，ツリー構造以外のタスクグラフについては言及されていない．

本論文の構成は以下のようになっている．2 章では余裕時間の削減による電力量削減アルゴリズムについて述べる．3 章では並列プログラムを実行するクラスタと電力測定に用いたシステムについて述べる．4 章では電力測定結果について述べる．5 章では考察と今後の検討課題について述べる．最後にまとめを述べる．

2. 待ち合わせ時間削減による電力量削減アルゴリズム

2.1 周波数・電圧と電力量

一般に，CMOS 回路のアクティブ電力は式 (1) で表される．

$$P = \alpha C V^2 f \qquad (1)$$

ここで，C は回路容量，V は電圧，f は周波数，α は回路の活性化率である．消費電力は周波数と電圧を下げることにより削減できる．

電力量は消費電力の時間積分である周波数と実行時間は反比例の関係にあり，プログラムを低周波数で実行するならば実行時間が増加する．周波数を下げることにより消費電力を下げることができるが，実行時間が増加するため電力量を削減することは難しい．一方，電圧を下げることは電力量を削減するために効果的である．電力量は電圧の 2 乗に比例するため，電圧を下げることにより電力量の大幅な削減が期待できる．ただし，プロセッサを安定に動作させるためには電圧のみを下げることはできず，周波数を共に下げる必要がある．

2.2 DVS による電力削減手法

並列プログラムにおいて同期を取るノード間で特定のノードの処理が早く終了し，同期までに余裕が発生するタスクの電力量を削減する手法を考える．ここで DVS を用いて以下の条件を同時に実現することを目的とする．

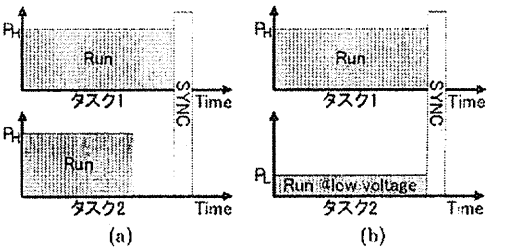

図 1　DVS による電力削減手法

- 電力量を削減する
- 実行時間を増加させない

図 1(a) に示すように，ある処理をした後に同期を必要とする 2 つのタスクを考える．タスク 1 とタスク 2 はそれぞれの処理終了後に同期を必要とする．同期を必要とするノード間でタスクの処理量が異なり，待ち時間がある場合に一方のノードは遊休状態になる．同期を行う他ノードがタスクを終了するまでこの状態は続く．このような問題に対して DVS により適切な周波数と電圧を選択することで電力量を削減する．

等しい周波数で実行した時に実行時間の短いタスクの周波数を変更する．周波数を大幅に下げた場合，実行時間の増加により同期までに処理が終了しない可能性がある．これにより，後続のタスクに影響を及ぼしシステム全体の実行時間が長くなる可能性がある．そのため，後続のタスクに影響を与えず同期までに処理を終了する周波数を求める必要がある．後続のタスクに影響を与えない周波数とは，同期までに処理を終了することが可能な周波数である．

図 1(a) について考える．実行時間の長いタスクはタスク 1，短いタスクはタスク 2 である．標準周波数 f_s でタスク 1 を実行したときの実行時間を T_1，タスク 2 を実行した時の実行時間を T_2 とする．また，実際に動作させる周波数をそれぞれ f_1，f_2 で表す．

図 1(a) においてタスク 1 がクリティカルパスとなり，全体の実行時間は T_1 によって支配される．したがって，T_1 が大きくなるか T_2 が T_1 を逆転しない限りシステム全体の実行時間に変化はない．このためにタスク 1 は周波数と電圧を変更できず，タスク 1 の周波数 f_1 は f_s となる．一方，タスク 2 については周波数と電圧を変更できる可能性がある．

同期までの余裕となる時間を余裕時間 T_{slack} とする．

$$T_{slack} = T_1 - T_2$$

上式で与えられる余裕時間を利用し，実行時間が短くなるタスク（ここではタスク 2）の周波数を変更する．

$$f_2 \geq f_s \times \frac{T_2}{T_2 + T_{slack}} \qquad (2)$$

ここで，以下の関係が成立する．

$$T_1 = T_2 + T_{slack}$$

よって，式 (2) は式 (3) に書き換えるられる

$$f_2 \geq f_s \times \frac{T_2}{T_1} \quad (3)$$

式(3)で与えられる周波数でタスクを実行した時，システム全体の実行時間は変化しない．

ここで，実際のプロセッサが動作可能な周波数はプロセッサ固有に決まっていることに注意しなければならない．一般的なプロセッサでは，周波数が低いほど対応する電圧を下げることができる．プロセッサのアクティブ電力量は電圧の2乗に比例する．例えば，電圧を70%に削減すると電力量は半減し，電圧を50%に削減すると電力量は1/4になる．このため，動作電圧を下げることは電力量を削減するために有効である．

低い電圧で安定に動作させるためには低い周波数を選択する必要がある．したがって，周波数は式(3)を満たし，プロセッサが動作可能な周波数の中で最低の値を選択する．

このように，同周波数で実行した時に同期のための待ちが発生するタスクに対して周波数と電圧を変更することで電力量を削減する．図1(b)に同期までの余裕時間を利用して周波数と電圧を削減した結果を示す．

2.3 非循環有向タスクグラフに対するアルゴリズム

タスクが実行されるプロセッサ，プロセッサ内の実行順序は適当なスケジューリングアルゴリズムにより決定しており，その順序はタスクグラフの依存関係として表現されているものとする．また，各タスクの実行時間はあらかじめ分かるものとする．

前節で述べた手法に従い，待ち時間を減少させると同時に電力量を削減するアルゴリズムを検討する．タスクグラフはタスクの集合 N とタスク間の依存関係を表す辺 V の集合で表される．タスクグラフ上の任意のタスク i において直接の後続タスクの集合 $succ(i)$，先行タスクを $pred(i)$ を以下のように定義する．

$succ(i) = \{j | i \rightarrow j$ に直接の依存関係がある $\}$

$pred(i) = \{j | j \rightarrow i$ に直接の依存関係がある $\}$

タスク i の実行時間を $T_{exec}(i)$ とする．その後，作成されたタスクグラフを元にして最も早い開始時刻（以下，最早開始時刻とする）$T_{start}(i)$，最も遅い完了時刻（以下，最遅完了時刻とする）$T_{end}(i)$ を求める．最早開始時刻 $T_{start}(i)$ は，タスクの処理順に従って

$$T_{start}(i) = max_{m \in succ(i)}\{T_{start}(m) + T_{exec}(m)\}$$

を計算することで求めることができる．一方，最遅完了時刻 $T_{end}(i)$ は，タスクの処理とは逆順に

$$T_{end}(i) = min_{n \in pred(i)}\{T_{end}(n) - T_{exec}(n)\}$$

を計算することで求められる．

最早開始時刻 $T_{start}(i)$ と最遅完了時刻 $T_{end}(i)$ から余裕時間 $D(i)$ を計算する．これは

$$D(i) = T_{end}(i) - \{T_{start}(i) + T_{exec}(i)\}$$

で与えられる．全てのタスクに対してこの計算を行う．

次に，周波数を変更するタスクを決定する．$D(i) = 0$ となるタスク i については周波数を変更することができない．なぜなら，この条件を満たすタスクはクリティカルパスに属しているためである．したがって，周波数を変更できるタスクは $D(i) > 0$ を満たすタスク i に限られる．$D(i) = 0$ を満たすタスクについては周波数を変更することができないので，これらのタスクは“決定済”とする．

次に，周波数を変更するタスク k を選択する．以下の2つの条件を満たすタスク k と経路を選択する．

(1) $succ(k)$ が全て“決定済”である

(2) 経路上のタスク実行時間 $T_{exec}(j)$ の総和を経路の長さと定義した時，“決定済”であるタスクまでの長さが最長となるタスク k とその経路

これらの条件を共に満たすタスク k に対して周波数の変更する．ここで最長となる経路長，すなわち最大実行時間を T_{path} とする．

次に，タスク k に対して周波数の変更を行う．タスク k を実行する周波数 $F(k)$ を式(4)で与える．ここで，f_s は標準周波数を表す．

$$F(k) = f_s \times \frac{T_{path}}{T_{path} + D(k)} \quad (4)$$

$F(k)$ で実行した時のタスク k の処理時間 $T'_{exec}(k)$ を式(5)で与える．

$$T'_{exec}(k) = T_{exec}(k) \times \frac{T_{path} + D(k)}{T_{path}} \quad (5)$$

$T'_{exec}(k)$ 以内でタスク k を実行すれば全体の実行時間に影響を与えない．最大実行時間 T_{path} を用いることで，複数のタスクの周波数を平均的に下げることが可能である．これにより，タスク全体で電力量を削減することができる．

周波数を決定した後，タスク k を“決定済”とする．また，周波数を変更したことによりタスク処理に要する時間が $T'_{exec}(k)$ に変更される．$T'_{exec}(k)$ を用いてタスク k の実行時間の更新を行い，最早開始時刻，最遅完了時刻も変更する必要がある．しかしながら，これらの変更が生じるタスクは周波数変更を行ったタスクに隣接する余裕時間 $E(i) > 0$ となるタスクのみである．すべてのタスクに対して余裕時間の再計算を行う必要は無い．

タスク集合 N に属する全てのタスクが”決定済”になるまで以下の作業を繰り返す．

- 最早開始時刻・最遅完了時刻・余裕時間を計算する
- $D(i) = 0$ を満たすタスクを“決定済”とする
- 周波数を変更するタスクの選択する
- 選択したタスクの周波数を変更し“決定済”とする

これにより，周波数を削減可能なタスクに対して実行時間に影響を与えずに電力量を削減できる．

2.4 非循環有向タスクグラフへの適用例

非循環有向タスクグラフに対してアルゴリズムを適用した例を示す．図2に周波数選択の様子を示す．円内の上段の数値は周波数の相対値，下段の数値は実行時間，四角内の数値は余裕時間をそれぞれ意味する．“決定済”タスクは四角内を着色して示す．

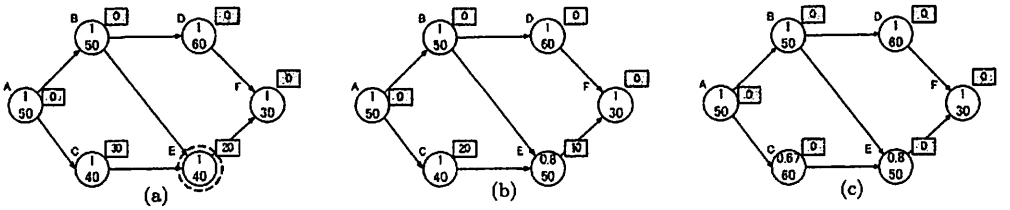

図 2　非循環有向タスクグラフに対するアルゴリズム

表 1　クラスタの構成

System	Crusoe クラスタ	Turion クラスタ
CPU	TM-5800	AMD Turion MT-34
Frequency	933	1800
Memory	256MB(SDR)	1GB(DDR)
OS	Linux 2.6.11	Linux 2.6.11
Compiler	gcc3.4.1	gcc3.4.1
NIC	Fast Ethernet	Gigabit Ethernet
Number of nodes	4	8

最早開始時刻，最遅完了時刻を求め，余裕時間を計算する．この余裕時間を用いて周波数を変更するタスクを決定する．ここで，余裕時間のあるタスクはタスク C とタスク E である．これら以外のタスクについては動作周波数を変更できず，"決定済" とする．

まず，周波数変更条件を満たすタスク E の周波数を変更する．T_{path} は 80 であり，タスク E の余裕時間は 20 である．式 (4) を用いて周波数の変更を行う．周波数は 0.8 となり，この周波数で実行した時の実行時間は式 (5) より 50 である．

実行時間が変更されたため，余裕時間を更新する．余裕時間を更新した時の様子を図 2(b) に示す．

次に，タスク C の周波数を変更する．T_{path} は 40，タスク C の余裕時間は 20 であるから，周波数は式 (4) より 0.67，実行時間は式 (5) より 60 で与えられる．

アルゴリズム適用後の各タスクの周波数，実行時間，余裕時間を図 2(c) に示す．タスク C，タスク E の周波数と電圧を下げる事で電力量が削減される．

3. 評価環境

3.1 クラスタ

Crusoe クラスタと Turion クラスタを実験用クラスタとして用い，これらに対して電力測定を行った．表 1 にクラスタの構成を示す．

Crusoe プロセッサは商用 IA-32 互換プロセッサとして初めて DVS を搭載したプロセッサである．負荷の状況に応じて自動的に周波数と電圧を変更する LongRun[5] と呼ばれる機構を搭載している．評価に用いた TM-5800 は 300[MHz] から 933[MHz] に 5 段階の動作可能な周波数が設けられている．動作させる周波数と電圧を指定する API を作成し，これを用いて周波数と電圧を変更した．プログラム中から API を呼び出すことで周波数と電圧を変更する．これにより，指定した周波数と電圧でプログラム中の特定のコードを実行できる．また，プログラム中で何度も周波数と電圧を変更することができる．

Turion プロセッサ[6] はモバイル環境向けプロセッサであり，周波数と電圧を変更する PowerNow!機構を搭載している．レジスタを操作することで周波数と電圧を直接変更する API を作成し，これを用いて周波数と電圧の変更を行った．今回用いた AMD Turion MT-34 は 800[MHz] から 1800[MHz] の 200[MHz] 間隔，6 段階の周波数で動作する．電圧は 1.2[V] から 0.9[V] の範囲で変更することができ，API によって周波数に対応した電圧を適宜選択する．

3.2 電力測定システム PowerWatch

ホール素子を用いた非接触型センサ，接続 BOX，A/D コンバータ，管理ノードからなる電力測定システムを用いて電力測定を行う．電線に電流が流れるとホール効果により電圧が発生する．この電圧をセンサで観測することで電流を測定できる．測定したデータは接続 BOX と A/D コンバータを経由し管理ノードに渡される．管理ノードには汎用 PC を用い，消費電力を計算すると共にデータの解析や編集を行う．

プロセッサの電力変動に着目して測定を行った．ATX 電源の各電圧線の中から，プロセッサに電力を供給している電圧線において消費する電力を測定する．具体的には，Crusoe プロセッサでは+5[V] 系統，Turion プロセッサは+12[V] 系統がプロセッサに電力を供給している．そのため，これらの系統に流れる電流を測定し評価に用いた．

評価には電力量と実行時間を用いた．電力量の単位として [W・scc] を用いる．

4. 評　　価

4.1 マスタ・ワーカ型プログラムに対しての評価

マスタ・ワーカ型の並列プログラムとして，固有値問題解法プログラム[7] を用いた．これは複素平面上の特定の領域内にある固有値と対応する固有ベクトルを求めるものである．マスタが問題を複数個の小問題に分割し，ワーカは割り当てられた小問題に対して計算を行い結果を返す．小問題を均等に配分できない場合に負荷不均衡が発生し，遊休状態となるノードが発生する．そこで，電力量削減アルゴリズムを用いて周波数と電圧を適切に制御することで電力量を削減する．

H_2O_3 分子の電子状態計算の際に必要となる固有値解法を行う．Turion クラスタ (4 nodes) で固有値解法プログラムを実行した時の電力遷移を示す．図 3 は

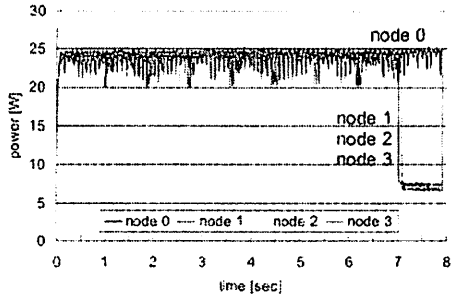

図 3 マスタ・ワーカ型プログラムを標準周波数で実行

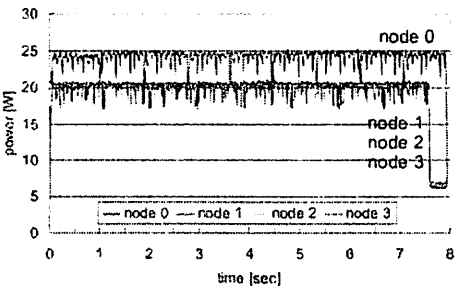

図 4 マスタ・ワーカ型プログラムにアルゴリズムを適用し実行

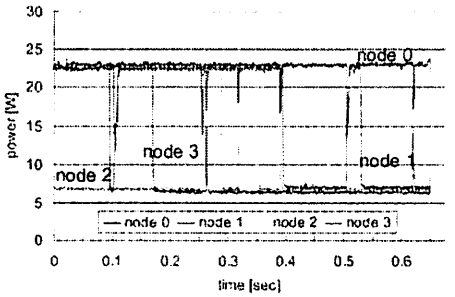

図 5 ツリー型プログラムを標準周波数で実行

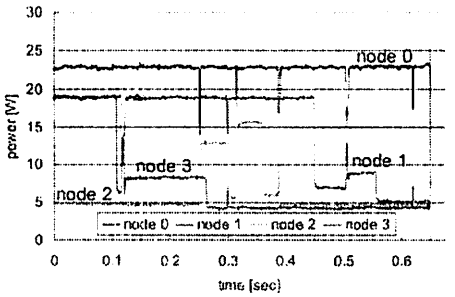

図 6 ツリー型プログラムにアルゴリズムを適用し実行

アルゴリズムを適用せずに全ノードを標準周波数である 1800[MHz] で実行した時の消費電力遷移である．**図 4** はアルゴリズムを適用し，DVS を用いて周波数と電圧を変動させた時の消費電力遷移である．

33 個の小問題を可能な限り均等に配分する．33 個の小問題を 4 ノードで並列に計算する場合，どのように配布してもわずかながら負荷不均衡が発生する．今回は node 0 に 9 個，他のノードに 8 個の小問題を配布することにする．node 0 は他のノードに比べて多くの問題が割り当てられるため実行時間が長くなる．これは図 3 より確認できる．

図 4 は提案アルゴリズムを適用した時の結果である．node 1，node 2，node 3 の周波数と電圧を下げて実行している．これらのノードは，最高周波数・最高電圧で動作したときに比べて低い消費電力で処理が行われている．また，消費電力の積分値である電力量もアルゴリズム適用前に比べて削減される．また，これらのノードの実行時間は増加し余裕時間は減少する．しかしながら，これらのタスクの実行時間は node 0 に比べて短いためシステム全体の実行時間は変化しない．

表 2 に各クラスタにおいて提案アルゴリズム未適用時と適用時のプロセッサ電力量と実行時間を示す．8 ノードで実行した場合，4 ノードで実行した時に比べて電力量の削減率が大きい．これは，周波数と電圧を変更するノード数が 7 ノードと多いことが理由として挙げられる．また，多くのノードに問題を配布すると各ノードに割り当てる小問題数が少なくなり待ち時間の相対値が大きくなる．これにより周波数と電圧を大幅に下げることができたことも理由として挙げられる．一方，実行時間の大幅な増加は見られない．

4.2 ツリー型プログラムに対しての評価

疎行列に対するコレスキー分解の並列解法プログラムに対して電力量の削減を行う．E.Rothberg のブロック化ファンアウト法[2] に基づいたコレスキー分解プログラムに対して電力量削減アルゴリズムを適用した．このプログラムは，構造解析フェーズと数値解析フェーズからなる．構造解析フェーズでは入力行列を元にツリーを構築する．数値解析フェーズでは作成したツリーに沿って行列の各要素の値を求める．実行時間の大半は数値解析フェーズであり，これを並列化することで実行時間の短縮を図る．

ツリーを構築すると共に処理量の見積もりを行う．この見積もりに対して提案するアルゴリズムを適用し，電力量を削減する．

図 5 はアルゴリズムを適用せずに全ノードを標準周波数である 1800[MHz] で実行した時の消費電力遷移である．**図 6** はアルゴリズムを適用し，DVS を用いて周波数と電圧を変動させた時の消費電力遷移である．入力データとして SPLASH ベンチマーク[3] に付属する tk15.O を用いた．

node 0 にクリティカルパスとなる全てのタスクを割り当てた．このため node 0 では周波数と電圧を下げることができず，電力量に変化は無い．一方，その他のノードはタスク間待ち合わせによる余裕時間を利

表 2　疎行列解法実行時の電力量と実行時間　(w/o) は全ノードを標準周波数で実行した時の結果

	電力量 (w/o) [W・sec]	電力量 [W・sec]	変化率 [%]	実行時間 (w/o) [sec]	実行時間 [sec]	変化率 [%]
Crusoe	1190	1091	-8.3	48.2	49.9	+3.53
Turion(4 nodes)	715	658	-8.0	7.93	7.95	+0.25
Turion(8 nodes)	782	651	-16.8	4.52	4.54	+0.44

表 3　コレスキー分解実行時の電力量と実行時間　(w/o) は全ノードを標準周波数で実行した時の結果

	電力量 (w/o) [W・sec]	電力量 [W・sec]	変化率 [%]	実行時間 (w/o) [sec]	実行時間 [sec]	変化率 [%]
Crusoe	49.2	44.7	-9.1	2.17	2.21	+1.84
Turion(4 nodes)	38.1	33.6	-11.8	0.651	0.651	± 0
Turion(8 nodes)	51.9	42.3	-18.5	0.553	0.554	+0.81

用して電力量の削減を行う．node 1 で 10.9%，node 2 で 27.0%，node 3 で 26.3%の電力量を削減した．

表 3 に各クラスタにおいて提案アルゴリズム未適用時と適用時のプロセッサ電力量と実行時間を示す．各クラスタにおいて，すべてのノードを標準周波数で実行した時と比較する．8 ノードの時に電力量が大幅に削減された理由としてはマスタ・ワーカモデルの実験と同様に周波数を変更するノードの割合が多くなったためであると考えられる．

5. 考察と課題

5.1 タスクスケジュールアルゴリズムと電力量削減アルゴリズムの関連

タスクスケジューリングアルゴリズムの多くはシステムの実行時間を最適化するとともに同期に対する待ち時間の削減に着目している．一方，提案したアルゴリズムは待ち時間を元に電力量の削減を行う．今回は，適当なタスクスケジューリングアルゴリズムによりプロセッサへのタスク割り当てが決定した後，提案したアルゴリズムを適用する手法を採った．しかしこれらのアルゴリズム間の関連が低く，提案したアルゴリズムを適用した時に電力面において最適な解を得られるとは限らない．

タスクスケジューリングアルゴリズムと電力量削減アルゴリズムを組み合わせることで実行時間を最適化するとともに電力量を最適化できる可能性がある．現在の手法では，提案アルゴリズム側で削減できる電力量はタスクの割り当て方式によって制限されることが問題である．タスクの割り当て方法の変更を含めて考慮することでこの制限が解除され，電力量をさらに削減できると考えられる．

5.2 実行時ライブラリの検討

現在は，電力量削減アルゴリズムを適用するプログラムの解析を行い周波数を変更する API をプログラム中から呼び出すことで電力量を削減している．この方式では移植性に乏しく，他のプログラムへの適用が容易ではない．

この問題に対し，タスクスケジューリングを行うライブラリを作成しライブラリに DVS 機構を利用した電力量削減手法を組み合わせる方法が考えられる．これにより，ユーザが待ち合わせ時間を考慮せずに，自動的に電力削減を行う事ができると考えられる．

6. 結　　論

タスク間の待ち合わせの際に負荷不均衡が原因となる余裕時間を用いて電力量を削減する手法を提案した．本手法は非循環有向グラフで表現されるタスクグラフに対して電力量を削減するアルゴリズムである．

マスタ・ワーカ型プログラムとツリー型タスクプログラムの 2 種について各タスクの処理時間を見積もり，提案アルゴリズムを適用した．電力測定環境を構築し，クラスタ上で並列プログラムを実行した時の消費電力を測定した．提案アルゴリズムを適用することで提案アルゴリズム未適用時と比較して 1%未満の性能評価で最大 18.5%の電力量を削減した．

謝辞　本研究の一部は科学技術振興機構・戦略的創造推進研究事業（CREST）-情報社会を支える新しい高性能情報技術-「低消費電力化とモデリング技術によるメガスケールコンピューティング」による．

参 考 文 献

1) G. Chen, K. Malkowski, M. T. Kandemir, and P. Raghavan. Reducing power with performance constraints for parallel sparse applications. In *IPDPS*, 2005.
2) Rothberg E. and A. Gupta. An efficient block-oriented approach to parallel sparse cholesky factorization. In *SC*, pp. 503-512, 1993.
3) Stanford Parallel Applications for Shared Memory (SPLASH). http://www-flash.stanford.edu/apps/SPLASH/.
4) Chung hsing Hsu and Wu chun Feng. A feasibility analysis of power awareness in commondity-based high-performance clusters. In *Cluster*, 2005.
5) Crusoe LongRun Power Management White Paper. http://www.transmeta.com/crusoe/longrun.html.
6) AMD Athlon^{TM}64 processor power and thermal data sheet, 2005.
7) T. Sakurai and H. Sugiura. A projection method for generalized eigenvalue problems. *j. Comput. Appl. Math*, Vol. 159, pp. 119-128, 2003.