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，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，同期待ちの余裕時間を削減するとともに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．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18.5%の電力量を削減できることを評価実験により確認した．

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 uniformly, 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-based parallel programs. We found that our algorithm can reduce the power 18.5% with only 1% performance loss in our evaluation.

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が増加している．消費電力の増加にともな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等のさまざまな問題が発生する[5]．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．

近年，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きた．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．最近では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が利用可能になりつつある．

一方，PCクラスタ等の並列システムにおいてプログラムを実行するときに待合せのための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ときに発生する．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を最適化する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合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なければならない．どれほ

† 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

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．いいかえれば，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，システム全体の実行時間は増加しない．

そこで，待ち時間のある並列プログラムに対して実行時間を増加させずに電力量を削減する手法を考える．本論文ではタスクグラフが非循環有向グラフとなる並列プログラムに対して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を構成す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ともに，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，各ノードの電力を評価した．

近年，DVS を高性能計算分野に応用し電力を削減する手法が数多く提案されている．

Chen ら[1] はツリー型のタスクグラフ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て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．また，Chen らは実機による電力測定を行っていない．ツリー以外の一般的なタスクグラフについても言及していない．

Kappiah ら[7] は繰り返し実行されるルーチンの実行履歴から周波数を決定するシステムを提案している．Hsu ら[6] は，高性能計算向けの Run-Time システムを提案している．これらの手法は，いずれもシステムの状態を監視し続ける必要がある．我々が提案する手法はタスクの割当て時に動作周波数を決定するため，システムの状態を監視する必要がない．

Ge ら[4] は，通信時の周波数を下げることにより，少ない性能低下で電力量の削減を実現する手法を提案している．この手法では，プロファイル情報を用いて適切な周波数を選択する．そのため，実行時間や電力のプロファイル情報が必要となる．我々の提案する手法は，処理時間の予測を用いるためプロファイル情報を必要としない．

本論文の構成は以下のようになっている．2 章では待合せ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 章では評価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. 待合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) で与えられる．

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

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．電力量は電圧の 2 乗に比例するため，電圧を下げることは電力量の削減に効果的である．たとえば，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．

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ない．したがって，電圧と周波数を同時に変更する必要がある．

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と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場合について考える．このような問題に対し，タスクの電力量を削減することができる．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．

- 電力量を削減する．
- 実行時間を増加させない．

図 1 のように，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．タスク割当てアルゴリズムの多くは，各ノードのタスク処理完了時刻が等しくなるようタスクの割当てを行う．しかしながら，必ずしもタスクの完了時刻が等しくなるとは限らない．ノードの処理完了時刻が異なる場合，一方のノードでは待ちが発生し遊休状態になる．これは，同期相手となるノードがタスクを終了するまで続く．

このような問題に対し，DVS により適切な周波数を選択することで電力量を削減することができる．同じ周波数で実行したときに待ちが発生するノードに対し，周波数の変更を行う．周波数を下げすぎた場合，実行時間が大幅に増加し同期までにタスクを終了しない可能性がある．これが後続の処理に影響しシステム全体の実行時間を長くする可能性がある．後続の処理に影響を与えないためには，同期までに処理を終了すればよい．

図 1 を用いて周波数の決定方法を検討する．図 1 において実行時間の長いタスクはタスク 1，実行時間の

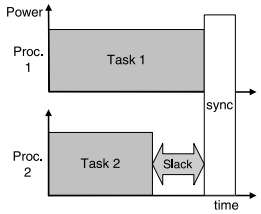

図 1 電力量最適化前
Fig. 1 Power profile before optimization.

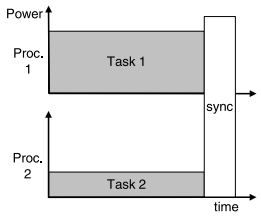

図 2 電力量最適化後
Fig. 2 Power profile after optimization.

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 を実行したときの実行時間を 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 とする．プロセッサが次の処理を行えず遊休状態でいる時間を余裕時間 T_{slack} と定義する．

システム全体の実行時間を増加させないために，タスク 1 はそのままの周波数で実行しなければならない．したがって，タスク 1 の周波数 f_1 は f_s となる．

タスク 2 では，同期までに処理を終了する範囲で周波数を下げることができる．タスク 2 の周波数 f_2 が式 (2) を満たすとき，実行時間は逆転しない．

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

ここで，実際のプロセッサが動作可能な周波数は離散であることに注意しなければならない．一般的には，周波数が低いほど対応する電圧を下げることができ，電力量を削減することができる．したがって，式 (2) を満たし，かつプロセッサが動作可能な最低の動作周波数を選択する．

このように，T_{slack} を用いて周波数を変更し，これに合わせて電圧を下げることで電力量を削減する．**図 2** に周波数を変更し電力量を削減した様子を示す．

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予測できるものとする．

2.2 節で述べた手法に従い，待ち時間を減少させるとともに電力量を削減する．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 $succ(i)$，先行タスク集合 $pred(i)$ 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．

タスクグラフにおいて最も早く開始可能な時刻（以下，最早開始時刻とする）$t_{start}(i)$，処理を完了していなければ後続の処理に影響を与える最も遅い時刻（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下式を計算することで求める．

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

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

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

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．これは式 (3) で与えられる．タスク集合 N を構成するすべてのタスクに対して式 (3) を計算することで，余裕時間を求める．

$$D(i) = t_{end}(i) - \{t_{start}(i) + T_{exec}(i)\} \qquad (3)$$

次に，周波数を変更するタスクを決定する．周波数を変更可能なタスクは余裕時間 $D(i) > 0$ を満たすタスク i である．余裕時間が 0 であるタスクはクリティカルパスに属するため，周波数を下げるとシステム全体の実行時間が増加する．このため，$D(i) = 0$ を満たすタスクは周波数が “決定済み” であることを示すフラグを立てる．

次に，周波数を変更するタスク k を選択する．

(1) $succ(k)$ がすべて “決定済み” である．
(2) (1) を満たすタスクから $pred(i)$ が “決定済み” であるタスク i までのタスク列を考える．タスク列が複数に分岐する場合はタスク列に属するタスクの実行時間の総和が最大となる列を選択し，このタスク列を経路と定義する．経路に属するタスクの合計実行時間を T_{path} としたとき，T_{path} が最大となるタスクを求める．

上記の条件を満たすタスクをタスク k とし，その経路の実行時間 T_{path} を求める．

次に，タスク k の周波数を変更する．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 は標準の周波数を表す．

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

$F(k)$ で実行したときのタスク k の処理時間 $T'_{exec}(k)$ は式 (5) で与えられる．

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

$T'_{exec}(k)$ 以内でタスク k を実行すれば全体の実行時間に影響しない．実行時間が最長となる経路に要する時間 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，電力を削減可能な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 “決定済み” 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．タスク k の実行時間を更新し，最早開始時刻，最遅完了時刻を再計算する．これらの変更が必要となるタスクはタスク k から “決定済み” であるタスクまでの一部タスクに限られる．すべてのタスクに対して余裕時間を再計算する必要はない．

タスク集合 N に属するすべてのタスクが “決定済み” 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 “決定済み” とする．
- 周波数を変更するタスクを選択する．
- 選択したタスクの周波数を変更し “決定済み” とする．

周波数を適切に制御することでシステム全体の実行時間を増加させることなく電力量を削減することが可能である．

2.4 ツリー型のタスクグラフへの適用例

タスクグラフがツリー型となるプログラムに対して本アルゴリズムを適用する．

図 3，**図 4**，**図 5** および **図 6** に周波数選択の様子を示す．それぞれの左図において円内上段の数値は周波数の相対値，下段はタスク処理に要する時間を意味する．隣接する四角内の数値は余裕時間を表し，余裕時間がなく周波数変更を行えないタスクやすでに周波数変更を完了したタスク，すなわち “決定済み” であるタスクについてはこれを網掛けして示す．右図は左図で示されるタスクグラフを実行したときの時間経過と消費電力の関係を示す．消費電力は式 (1) によって導かれるものとした．プロセッサへのタスク割当ては右図のように行われるものとし，各タスクは左図の矢印に沿った順番で処理されるものとする．また，各プロセッサが最初に処理するタスクの前に同期をとるものとし，この同期を仮想的な “決定済み” タスクとする．

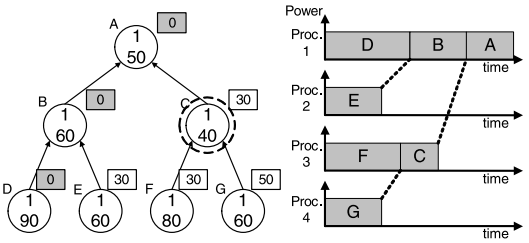

図 3 ツリー型タスクグラフへの適用例（1）
Fig. 3 Example of program represented by tree (1).

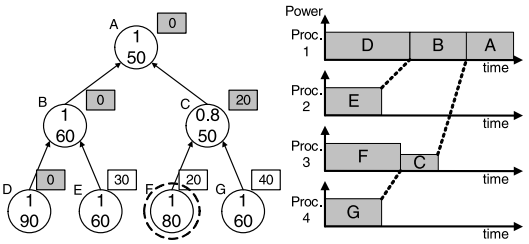

図 4 ツリー型タスクグラフへの適用例（2）
Fig. 4 Example of program represented by tree (2).

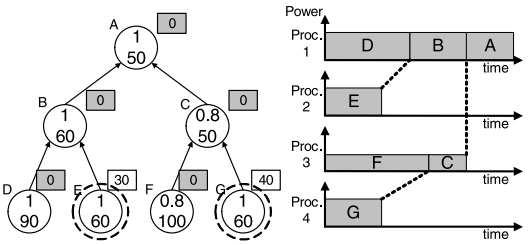

図 5 ツリー型タスクグラフへの適用例（3）
Fig. 5 Example of program represented by tree (3).

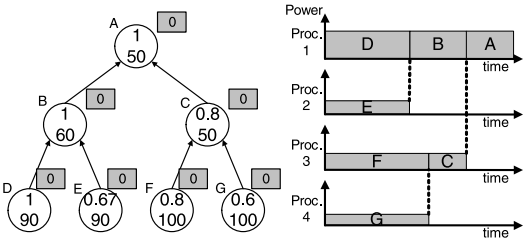

図 6 ツリー型タスクグラフへの適用例（4）
Fig. 6 Example of program represented by tree (4).

すべてのタスクに対して最早開始時刻と最遅完了時刻を求め余裕時間を計算し，“決定済み” フラグを立てる．このときの様子を図 3 に示す．

図 3 において周波数の変更を行うタスクを決定する．後続タスクがすべて “決定済み” であるタスクは

タスク C とタスク E である．タスク C から “決定済み” であるタスクまでの経路は，タスク C→ タスク F であり 120 の時間を要する．タスク E が要する時間は 60 であるため，ここではタスク C を選択する．タスク C の周波数は式 (4) より 0.8 となり，タスク C の処理に要する時間は式 (5) より 50 となる．実行時間の変更にともない余裕時間を更新する．更新後の状態を図 4 に示す．

図 4 において，選択可能なタスク中で “決定済み” であるタスクまでの実行時間が最も長いタスクはタスク F である．したがって，このタスク F の周波数を調整する．タスク F は式 (4) より 0.8 の周波数で実行し，実行時間は 100 となる．その後，実行時間と余裕時間を更新する．図 5 にタスク F の周波数選択後の状態を示す．

タスク E，タスク G についても同様の処理を行う．選択可能なタスクが複数存在する場合，どちらを先に選択してもよい．これらの処理を繰り返すことで図 6 に示す状態が得られ，すべてのタスクの余裕時間が 0 になる．

実際の環境に適用する場合，周波数と電圧を合わせて下げることでこれを実現する．しかしながら，プロセッサはすべての周波数で動作可能であるとは限らない．システム全体の実行時間を増加させないために，計算で求めた周波数よりも高い周波数を選択しなければならない可能性がある．

2.5 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**図 7**，**図 8** および**図 9** に示す．左図において円内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．“決定済み” タスクは四角内を網掛けして示す．また，右図は左図で示されるタスクグラフを実行したときの時間経過と消費電力の関係を示す．プロセッサへのタスク割当てはすでに行われているものとし，各タスクは左図の矢印に沿って処理されるものとする．

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．したがって，これら以外のタスクについては動作周波数を変更できない．

すべての後続タスクが “決定済み” であるタスクはタスク E であるため，タスク E の周波数を変更する．T_{path} はタスク E とタスク C の実行時間の合計である 80，余裕時間は 20 である．式 (4) によると，周波数は 0.8 となる．このときの実行時間は式 (5) より 50 である．実行時間に変更が生じたため，余裕時間を更新する．余裕時間を更新後の状態を図 8 に示す．

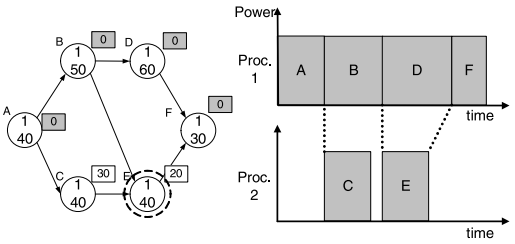

図 7 非循環有向タスクグラフに対する適用例（1）
Fig. 7 Example of program represented by DAG (1).

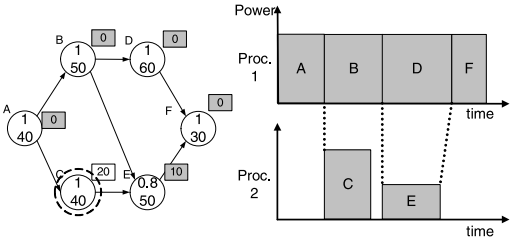

図 8 非循環有向タスクグラフに対する適用例（2）
Fig. 8 Example of program represented by DAG (2).

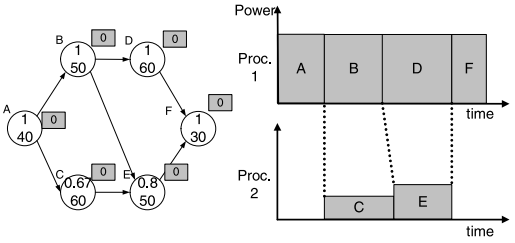

図 9 非循環有向タスクグラフに対する適用例（3）
Fig. 9 Example of program represented by DAG (3).

次に，タスク C の周波数を変更する．タスク C の実行時間は 40，余裕時間は 20 であるから，式 (4) を用いて周波数を 0.67 に変更する．また，実行時間は式 (5) より 60 になる．アルゴリズム適用後の各タスクの状態を図 9 に示す．タスク E の開始時刻は遅れるが，タスク E はタスク D との同期までに処理を完了するため全体の実行時間は増加しない．

これより，タスク C とタスク E の処理に要する電力量が削減されることが分かる．また，クリティカルパスに合わせて周波数を変更しているためシステム全体の実行時間は変化しない．

3. 評価環境

3.1 クラスタ

Turion クラスタと Crusoe 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．

Turion プロセッサ[9] はモバイル環境向けプロセッ

表 1 クラスタの構成
Table 1 Power-scalable clusters.

	Turion cluster	Crusoe cluster
CPU	Turion MT-34	TM-5800
Frequency	1800 MHz	933 MHz
Memory	DDR 1 GB	SDRAM 256 MB
Network	Gigabit Ethernet	Fast Ethernet
Kernel	Linux 2.6.11	Linux 2.6.11
Compiler	gcc 3.4.1	gcc 3.4.1
MPI	LAM 7.1.1	LAM 7.1.1
# of nodes	4 or 8	4

表 2 各プロセッサの動作可能周波数と電圧
Table 2 Voltage and frequency of processors.

Turion MT-34		TM-5800	
frequency [MHz]	voltage [mV]	frequency [MHz]	voltage [mV]
1800	1200	933	1350
1600	1150	800	1250
1400	1100	667	1200
1200	1050	533	1100
1000	1000	300	900
800	900		

サであり，周波数と電圧を変更する PowerNow!を搭載している．MSR（Model Specific Register）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．これにより，プログラム中の特定のコードを指定した周波数と電圧で動作させることが可能である．また，同一の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．

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 機構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 [8] と呼ばれる機構を搭載している．周波数と電圧を直接指定する API を作成し，これを用いて周波数と電圧を変更した．

今回評価に用いたプロセッサの周波数と電圧の組を**表 2** に示す．

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 PowerWatch を構築した．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．これをセンサで観測することで電流を測定できる．測定した電流データは接続 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，電力データを保存するとともにデータの解析や編集を行う．

クラスタの動作に応じた電力を取得するために，プログラムの実行と電力測定のタイミングをそろえる必要がある．**図 10** に電力測定のフローチャートを示す．クラスタ上で並列プログラムを実行する前に，クラスタ内の特定のノードが PowerWatch に対してメッセージを送信する．PowerWatch がこのメッセージを受信すると，応答メッセージを返すとともに電力測定を開始する．これにより，並列プログラム開始と電力測定開始のタイミングをそろえ，並列プログラムの処理と消費電力を対応付けることができる．

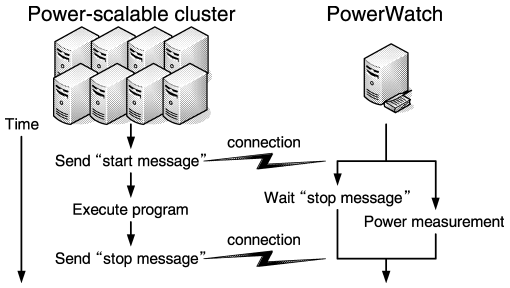

図 10 電力測定システム PowerWatch
Fig. 10 PowerWatch: How to measure the power.

電力測定には ATX 電源の各電圧線を利用した．システムの構成要素によって供給される電圧が異なる．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．

主にプロセッサに電力を供給する系統と他の系統にわけて予備評価を行った．Turion クラスタにおいて標準周波数で計算を行ったとき，プロセッサでは 22.9 W の電力を消費した．一方，最低周波数で計算を行ったときに 7.83 W の電力を消費した．プロセッサ以外で消費する電力はどのような周波数を選択した場合でも約 14 W であった．したがって，プロセッサは最大でシステム全体の 62%の電力を消費することが分かる．Crusoe クラスタではプロセッサが 2 W から 7 W の間で動作するのに対し，プロセッサ以外ではつねに 1.5 W 程度の電力を消費する．

以降の評価では 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着目している．これは，DVS を用いてもプロセッサの消費電力が変化するのみであり，他の構成要素の電力は変化しないためである．

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 [10] を用いた．これは複素平面上の

表 3 固有値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ときの結果
Table 3 Energy and execution time using our algorithm (eigenvalue solver).

	energy(w/o) [Wsec]	energy [Wsec]	ratio [%]	time(w/o) [sec]	time [sec]	ratio [%]
Crusoe	1120	1020	−8.9	48.3	48.4	+0.21
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

特定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．マスタは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を計算し結果を返す．小問題を均等に配分できない場合に終了時刻に差が発生し，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ことを試みる．

H_2O_3 分子の電子状態計算の際に必要となる固有値計算を行う．このとき，負荷ができる限り均等になるよう小問題を配布する．各ワーカに割り当てる小問題数により，実行時間を予測できる．

Turion クラスタ（4 nodes）で固有値解法プログラムを実行したときの消費電力遷移を示す．**図 11** は提案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 1,800 MHz で実行したときの結果である．**図 12** は提案アルゴリズムを適用し，DVS を用いて周波数を変化させて実行した結果である．

問題を 33 個の等規模な小問題に分割して評価を行った．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に不均衡が発生する．今回は node 0 に 9 個，他のノードに 8 個の小問題を配布した．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，処理に要する時間が他のノードと比べて長くなる．これは図 11 より確認できる．

図 12 は提案アルゴリズムを適用したときの結果である．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，標準の周波数で動作したときに比べて低い消費電力でタスクを実行している．また，電力量もアルゴリズム適用前に比べて削減されている．これらのノードの計算時間は増加しているが，node 0 の計算時間と比べて短いためシステム全体の実行時間に影響しない．動作周波数をさらに下げると，同期までに処理を終了することができず，システム全体の実行時間が増加する．

表 3 に各クラスタにおいて標準の周波数で実行したときと提案アルゴリズム適用時の電力量合計値と実行時間を示す．Crusoe クラスタで 8.9%，Turion クラスタ（4 nodes）で 8.0%，Turion クラスタ（8 nodes）で 16.8%の電力量が削減された．8 ノードで実行した

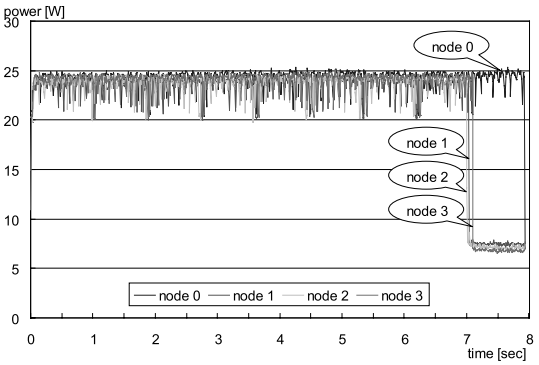

図 11 標準周波数実行時の消費電力遷移（固有値解法）
Fig. 11 Power profile of the program executed at standard frequency.

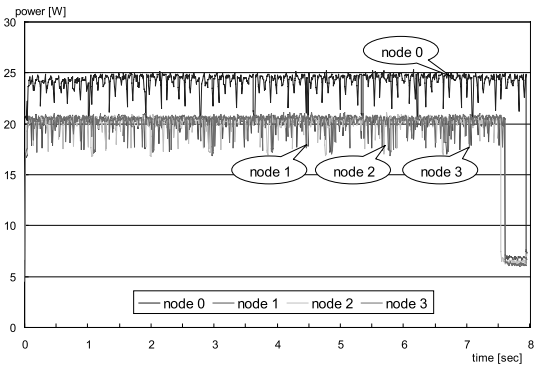

図 12 提案アルゴリズム適用時の消費電力遷移（固有値解法）
Fig. 12 Power profile of the optimized program.

場合，4 ノードで実行したときに比べて電力量の削減率が大きい．これは，低い周波数で計算を行うノード数が 7 ノードと多いことが理由としてあげられる．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余裕時間が相対値に大きくなる．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あげられる．一方，いずれの結果も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．Rothberg のブロック化ファンアウト法[2] に基づくアルゴリズムを用いたコレスキー分解プログラムに対し，提案アルゴリズム

表 4 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ときの結果
Table 4 Energy and execution time using our algorithm (Cholesky decomposition).

	energy(w/o) [Wsec]	energy [Wsec]	ratio [%]	time(w/o) [sec]	time [sec]	ratio [%]
Crusoe	48.1	45.0	-9.1	2.176	2.178	+0.09
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

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もとにツリー型のタスクグラフを構築し，プロセッサへのタスク割当てを行う．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．

タスクグラフを構築するとともに実行時間を予測する．今回は，実行時間として計算時間と通信時間を用いた．計算時間は計算量に比例するものとし，これは処理する行列のサイズによって求めた．通信時間は転送するデータサイズに比例するものとした．実測により計算性能と通信性能を求めておき，これらを係数として計算量と通信量に乗じた．これらの和を予測した実行時間とし，これを用いて提案アルゴリズムを適用した．

Turion クラスタを用いて疎行列に対する並列コレスキー分解を行う．**図 13** は全ノードを標準周波数である 1,800 MHz で実行したときの消費電力遷移である．**図 14** はアルゴリズムを適用し，DVS を用いて周波数を変更しながら処理を行ったときの電力遷移である．入力データとして SPLASH ベンチマーク[3] に付属する tk15.o を用いる．

クリティカルパスとなるタスクすべてを node 0 に割り当てた．このため node 0 では周波数と電圧を下げることができない．一方，その他のノードはタスク間待合せのための余裕時間を利用して電力量を削減する．

表 4 に各クラスタにおいて標準の周波数で実行したときと提案アルゴリズムを適用したときの電力量合計値と実行時間を示す．Crusoe クラスタで 9.1%，Turion クラスタ（4 nodes）で 11.8%，Turion クラスタ（8 nodes）で 18.5%の電力量が削減された．8 ノードのときに電力量が大幅に削減された理由としてはマスタ・ワーカモデルの結果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．いずれの結果も，実行時間の大幅な増加は見られない．

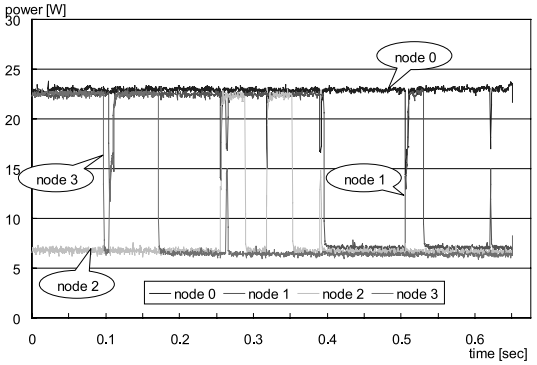

図 13 標準周波数実行時の消費電力遷移（コレスキー分解）
Fig. 13 Power profile of the program executed at standard frequency.

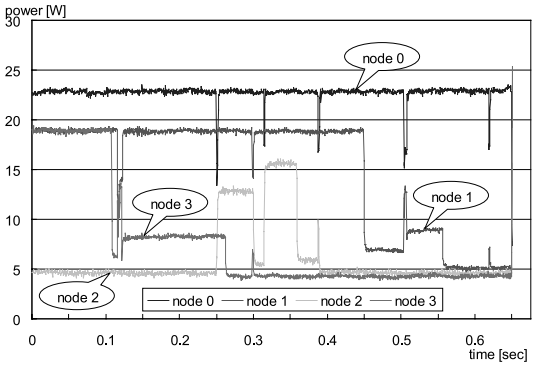

図 14 提案アルゴリズム適用時の消費電力遷移（コレスキー分解）
Fig. 14 Power profile of the optimized program.

5. 考　　察

5.1 並列プログラムに対する適用結果

4.1 節のマスタ・ワーカ型プログラムではワーカに配布する問題数に応じて実行時間を予測した．すべてのノードが同じ周波数で動作したときに各ノードの処理完了時刻が異なる場合，一部のノードに余裕時間が発生する．DVS を使用し，周波数を下げることで余裕時間を削減した．周波数を下げたことにより各ノードの実行時間は増加するが，この増加は余裕時間内に抑えてある．周波数を下げると同時に電圧を下げることにより電力量を削減することができる．一方で提案アルゴリズムを適用する場合，周波数を決定するフェー

表 5 計算時間のみで予測を行ったときの電力量と実行時間
Table 5 Energy and execution time of the optimized program using only the calculation time.

	energy [Wsec]	ratio [%]	time [sec]	ratio [%]
Crusoe	43.0	−10.6	2.246	+3.22
Turion (4 nodes)	33.5	−12.1	0.672	+3.23
Turion (8 nodes)	41.4	−20.2	0.586	+5.97

ズが挿入されるため実行時間が若干増加する．

4.2 節ではツリー型タスクを構築する並列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．各タスクの余裕時間を用いて周波数の変更を行い，その結果電力量は削減された．実行時間はわずかに増加している．これはマスタ・ワーカ型プログラムの結果と同様に周波数を決定するたのフェーズによるものであると考えられる．

周波数の変更には一定時間のオーバヘッドを要する．評価に用いたプロセッサでは数十 μsec 程度の時間で周波数の変更を行うことができた．これは評価に用いたタスクの粒度に比べて十分小さいものであるため無視できるものとした．しかしながら，粒度の細かい問題に対しては周波数変更のオーバヘッドを考慮しなければならない．

周波数を変更することにより余裕時間が 0 となることが望ましいが，評価結果ではそのような場面はあまり見られない．これは，プロセッサが動作可能な周波数が制限されていることが原因である．今回の評価では，計算で求めた周波数で動作できない場合，求めた周波数よりも高い周波数を選択した．これは，システムの実行時間を増加させないためである．このため，若干ながら余裕時間が残った．

5.2 実行時間予測と電力量削減アルゴリズム

我々が提案したアルゴリズムは実行時間の予測が重要である．実行時間の予測に失敗すると，周波数を変更することにより全体の実行時間が増加する危険性がある．

4.2 節では，問題に対して通信時間を考慮して実行時間を考えた．通信時間は並列プログラムにおける主要な要素であり，実際の実行時間に大きく影響する．

図 15 に通信時間を考慮せず，計算時間のみを予測し提案アルゴリズムを適用したときの消費電力遷移を示す．予測方法を変更した以外は図 14 の実験環境と同様である．

node 0 にクリティカルパスとなるすべてのタスクを割り当てた．そのため，node 0 はつねに最高周波数で計算および通信を行うことが期待される．しかしながら，node 0 が一時的に遊休状態になり，消費電力が下がっていることを確認できる．これは node 1 が同期までにタスクを終了しなかったためである．この遊休状態である時間によって全体の実行時間が増加する．一方，同期までに処理を終了できない低い周波数を選択したノードが存在したため，図 14 の結果と比較して電力量削減幅は大きくなっている．

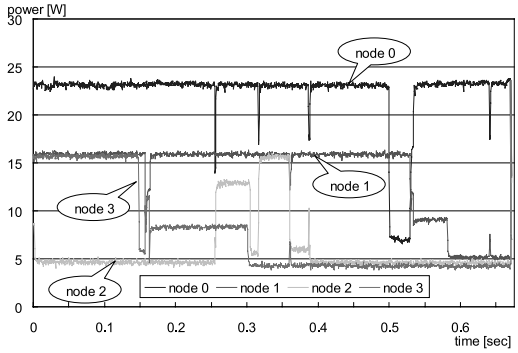

図 15 計算時間のみで予測を行ったときの消費電力遷移
Fig. 15 Power profile of the optimized program using only the calculation time.

表 5 に計算による実行時間のみを予測し，提案アルゴリズムを適用したときの電力量，実行時間を示す．Crusoe クラスタで 10.6%，Turion クラスタ（4 nodes）で 12.1%，Turion クラスタ（8 nodes）で 20.2%の電力量が削減された．いずれの結果も通信時間を予測しなかったときと比べて電力量削減幅が大きくなっているが，全体の実行時間が増加している．これは，実行時間の予測に失敗し同期までに処理を終了する周波数より低い周波数で動作したタスクが存在したことが原因である．

5.3 プロセッサ台数と電力量

プロセッサ台数と電力量の関係について検討する．Turion クラスタで固有値解法プログラムを実行したときの結果において，標準周波数実行時は 4 ノードで実行した際の電力量が 8 ノード実行時に比べて低いが，提案アルゴリズム適用時は逆転現象が発生する．ノード数の増加により各ノードに割り当てられるタスクが減少し，余裕時間が相対的に大きくなる．そのため，周波数と電圧の下げ幅が大きくなり電力量を大幅に削減できることが理由である．一方，コレスキー分解プログラムはツリー型のタスクグラフを構成するためタスクの並列性が低く，多くのプロセッサを用いて

も実行時間を短縮しにくい．そのため，遊休状態であるプロセッサの消費電力によってシステム全体の電力量が上昇し，固有値解法プログラム実行時のような電力量の逆転現象は見られなかった．

このように，周波数だけではなく台数を変更することによっても電力量が変化する．適切なプロセッサ台数を選択し周波数を調整することで，さらに電力量を削減できる可能性がある．

5.4 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

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，本論文で提案したアルゴリズムは待ち時間をもとに電力量の削減を行う．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当てが決定した後に提案したアルゴリズムを適用する手法をとった．しかし，これでは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ときに電力面において最適な結果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．タスクの割当て方法の変更を考慮することでこの制限がなくなり，実行時間を保ちつつ電力量をさらに削減できる可能性がある．

5.5 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．

これに対し，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 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，タスクスケジューリングを最適化しても同期のための余裕時間が生じたときに電力量を削減できる．

6. 結　　論

並列プログラムにおいて，タスクの完了時刻が異なり一部のノードに余裕時間が発生することがある．この余裕時間を用いて電力量を削減する手法を提案した．すでにツリー型のタスクグラフを構成する並列プログラムに対して電力量を削減するアルゴリズムは提案されている．本論文で提案するアルゴリズム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・ワーカ型とツリー型タスクグラフを構成する 2 種類の並列プログラムに対して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ときの消費電力を測定した．いずれのプログラムに対しても処理量を適切に見積もることで，実行時間を大幅に増加させることなくの電力量を削減できることが確認できた．また，本アルゴリズムは対象とするプロセッサに依存しない．搭載するプロセッサの異なる 2 種のクラスタに対してともに実行時間を大幅に増加させることなく電力量を削減した．提案アルゴリズムを適用することで標準の周波数で実行したときと比較して 1%未満の性能低下で 18.5%の電力量を削減できることを評価実験より確認した．

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) Chen, G., Malkowski, K., Kandemir, M.T. and Raghavan, P.: Reducing power with performance constraints for parallel sparse applications, *High-Performance, Power-Aware Computing in International Parallel Distributed Processing Symposium* (2005).
2) Rothberg, E. and Gupta, A.: An efficient block-oriented approach to parallel sparse cholesky factorization, *Supercomputing Conference*, pp.503–512 (1993).
3) Stanford Parallel Applications for Shared Memory (SPLASH). http://www-flash.stanford.edu/apps/SPLASH/
4) Ge, R., Feng, X. and Cameron, K.W.: Performance-constrained distributed dvs scheduling for scientific applications on power-aware clusters, *Supercomputing Conference* (2005).
5) Hsu, C.H. and Feng, W.C.: A feasibility analysis of power awareness in commondity-based high-performance clusters, *Cluster Computing* (2005).
6) Hsu, C.H. and Feng, W.C.: A power-aware run-time system for high-performance computing, *Supercomputing Conference* (2005).
7) Kappiah, N., Freeh, V.W. and Lowenthal,

D.K.: Just in time dynamic voltage scaling: Exploiting inter-node slack to save energy in mpi programs, *Supercomputing Conference* (2005).
8) Crusoe LongRun Power Management White Paper. http://www.transmeta.com/crusoe/longrun.html
9) AMD Athlon^{TM}64 processor power and thermal data sheet (2005).
10) Sakurai, T. and Sugiura, H.: A projection method for generalized eigenvalue problems, *J. Comput. Appl. Math.*, Vol.159, pp.119–128 (2003).

(平成 18 年 1 月 27 日受付)
(平成 18 年 5 月 6 日採録)

木村 英明（学生会員）

昭和 58 年生．平成 16 年小山工業高等専門学校電子制御工学科卒業．平成 18 年筑波大学第三学群情報学類卒業．現在，同大学大学院システム情報工学研究科在学中．低消費電力クラスタシステムに関する研究に従事．

佐藤 三久（正会員）

昭和 34 年生．昭和 57 年東京大学理学部情報科学科卒業．昭和 61 年同大学大学院理学系研究科博士課程中退．同年新技術事業団後藤磁束量子情報プロジェクトに参加．平成 3 年通産省電子技術総合研究所入所．平成 8 年新情報処理開発機構並列分散システムパフォーマンス研究室室長．平成 13 年より，筑波大学システム情報工学研究科教授．同大学計算科学研究センター勤務．理学博士．並列処理アーキテクチャ，言語およびコンパイラ，計算機性能評価技術，グリッドコンピューティング等の研究に従事．IEEE，日本応用数理学会各会員．

堀田 義彦（学生会員）

昭和 54 年生まれ．平成 15 年筑波大学第三学群情報学類卒業．現在，同大学大学院システム情報工学研究科在学中．クラスタシステム等の低消費電力化に関する研究に従事．

朴 泰祐（正会員）

昭和 36 年生．昭和 59 年慶應義塾大学工学部電気工学科卒業．平成 2 年同大学大学院理工学研究科電気工学専攻後期博士課程修了．工学博士．昭和 63 年慶應義塾大学理工学部物理学科助手．平成 4 年筑波大学電子・情報工学系講師，平成 7 年同助教授，平成 16 年同大学大学院システム情報工学系助教授，平成 17 年同教授，現在に至る．超並列計算機アーキテクチャ，ハイパフォーマンスコンピューティング，クラスタコンピューティング，グリッドに関する研究に従事．平成 14 年度および平成 15 年度情報処理学会論文賞受賞．日本応用数理学会，IEEECS 各会員．

高橋 大介（正会員）

昭和 45 年生．平成 3 年呉工業高等専門学校電気工学科卒業．平成 5 年豊橋技術科学大学工学部情報工学課程卒業．平成 7 年同大学大学院工学研究科情報工学専攻修士課程修了．平成 9 年東京大学大学院理学系研究科情報科学専攻博士課程中退．同年同大学大型計算機センター助手．平成 11 年同大学情報基盤センター助手．平成 12 年埼玉大学大学院理工学研究科助手．平成 13 年筑波大学電子・情報工学系講師．平成 16 年筑波大学大学院システム情報工学研究科講師．博士（理学）．並列数値計算アルゴリズムに関する研究に従事．平成 10 年度情報処理学会山下記念研究賞，平成 10 年度，平成 15 年度情報処理学会論文賞各受賞．日本応用数理学会，ACM，IEEE，SIAM 各会員．