2006－ARC－166（8）
2006／1／24

統計処理に基づくコンパイラ協調型DVFS手法

浅井雅司[†]　池田佳路[†]　佐々木広[†]
近藤正章[†]　中村宏[†]

モバイルコンピューティングにおけるバッテリ駆動時間の延長や，高性能プロセッサの熱問題の回避などのために消費電力の低減が要求されている．現在のプロセッサにはパフォーマンスカウンタが備えられ，各種のハードウェアイベントの計測が可能となっている．本研究では，あらかじめ各種アプリケーション実行中のカウンタ情報と性能との関係を統計的に学習し，コンパイラがターゲットアプリケーションのソースコードに対し性能予測と周波数・電圧のためのランタイムコードを挿入し，性能低下を一定以下に抑えつつ，消費電力を削減するための DVFS 手法を提案する．

A Compiler Cooperative DVFS Technique based on Statistical Learning

Masashi Asai,[†] Yoshimichi Ikeda,[†] Hiroshi Sasaki,[†]
Masaaki Kondo[†] and Hiroshi Nakamura[†]

This paper proposes a novel compiler cooperated DVFS technique which is based on statical learning. In our technique, the relationships between the hardware performance counter information and the performance are learned statistically in advance. Then, the compiler inserts the runtime code for predicting the performance of the target program and sets the voltage/frequency depending on the predicted performance. Our technique can reduce a great amount of energy consumption with keeping the performance degradation to a certain ratio.

1. はじめに

近年，マイクロプロセッサの低消費電力化・低消費エネルギー化はプロセッサの設計における最も重要な課題となってきている．モバイル計算機におけるバッテリ駆動時間の延長という要求はもちろんのこと，商用サーバ，さらには科学技術計算用途などのハイエンドなプロセッサにおいても，性能低下を抑えつつ消費電力を削減することが必要不可欠な課題となっている．

高性能および低消費電力の両要求を満たすために，電源電圧と動作周波数を動的に変更する機能 (DVFS: Dynamic Voltage and Frequency Scaling) を持つプロセッサの研究が活発に行われ，Intel Pentium M[1] などで採用される SpeedStep など，DVFS 機能を備えたプロセッサが数多く登場している．DVFS を用いることでマイクロプロセッサの消費電力を効果的に削減可能であることが広く知られている．

商用のプロセッサに用いられている DVFS 手法は OS によって電源電圧・クロック周波数を決定するものであり，タスクレベルでのプロセッサ使用率をもとに，粒度の粗いものである．本研究では，タスク内において DVFS を行うことを対象としており，プログラム中での，より細かい粒度の最適化が可能となっている．

タスク内において DVFS を適用する手法はこれまでにも数多く提案されており，基本的には OS やハードウェア，またはコンパイラによって，電源電圧・クロック周波数を決定する手法である．

OS による手法[2] やハードウェアによる手法[3] では，一般的に，ある固定されたタイムインターバル毎に，実行時におけるシステムの状態を監視することによって，未来のタイムインターバルにおける電源電圧・クロック周波数を決定する．実行時の情報を用いるため，キャッシュミスなどの動的な振る舞いにも対応できる．しかし，これらの手法でのタイムインターバルの長さは前もって一意に決定する必要があり，適用するプログラムに対して独立であるため，プログラム中のフェーズの変化などに対して対応できず，効果的に DVFS を行うことが出来ないといった問題がある．

また，コンパイラによる DVFS 手法[4] は，主としてプロファイリングを用いオフラインで分析を行い，DVFS を適用するというものである．これらの手法の問題点として，プロファイリング時と実際のプログラム実行時における環境が異なっていると DVFS が有効に適用できないことが挙げられる．プロファイリングによって決定された DVFS を行うタイミングが，実行時のプロセッサのメモリ・キャッシュ構成やプログラムの入力セットなどに対して最適であるとは限らな

† 東京大学 先端科学技術研究センター
Research Center for Advanced Science and Technology, The University of Tokyo

いからである．このように，ハードウェアや OS などによる動的な手法と，コンパイラによる静的な手法にはそれぞれ一長一短があり，お互いの良い点を用いるハイブリッドな手法によってより優れた最適化を行うことが可能であると考えられる．

一方，プロセッサの機能は集積回路技術の進歩と相まって，高度で複雑な機能が実装されるようになっている．例えば，命令実行においては，深いパイプラインや投機実行，スーパースカラ，アウトオブオーダー実行などの特徴をもち，その他でもキャッシュをはじめとするメモリ階層など，ハードウェアが複雑化している．そのため，マイクロプロセッサの動作の定性的な理解や，コードの最適化が特に困難となってきている[5)6)]．ソフトウェア開発時にこれらの機能を十分に活用して性能を向上させるために，Intel Pentium II 以降，AMD Athlon 以降などでは，パフォーマンスカウンタと呼ばれる機構が備えられており，キャッシュのヒット/ミス回数や，分岐予測ミス回数などのハードウェアイベントの計測が可能となっている．

文献[7)] では，Intel Pentium M を用いた実機のプラットフォーム上において，コンパイラによる静的な解析を行い，パフォーマンスカウンタによって得られる実行時のハードウェア情報を，定性的に解析を行うことによって求めたアルゴリズムに基づいて DVFS を適用する手法を提案している．しかし一般的に，定性的な解析によるアルゴリズムを用いた手法は，プラットフォームに依存することが多い．また，今後マルチコア化などにより，さらなるハードウェアの複雑化を考えると定性的な解析は難しくなる．そのためその都度最適なアルゴリズムを提案していくことは非常に困難になる．

本稿で提案する統計処理に基づくコンパイラ協調型 DVFS 手法は，ハードウェアカウンタによって得られる実行時の動的な情報をもとに，電圧変更時の性能を予測し，性能低下を決められた範囲内に抑えた上で低周波数でプログラムを実行し，消費電力を削減する．性能の予測式は，あらかじめ統計的処理を用いた学習を行うことによって求める．提案手法では，さらに複雑化された新たなプラットフォーム上において適用する際においても，特徴的なアプリケーション群を一通り実行し，その結果をもとに学習を行うだけで電源電圧・クロック周波数変更時の性能を高い精度で予測することが可能になり，効果的な DVFS を行うことができると考えられる．

本稿では，統計処理手法である重回帰分析，提案手法の概要について述べ，評価，まとめを行う．

2. 重回帰分析

本章では本稿で用いる統計的処理についての概要を簡単に述べる．

2.1 偏回帰係数

測定の場合の数を n，従属変数を Y，p 個の独立変数を $X_i(i=1,2,\cdots,p)$ とする．従属変数の予測値 $\hat{Y}$ は，重回帰式

$$\hat{Y} = b_0 + b_1X_1 + b_2X_2 + \cdots + b_pX_p \qquad (1)$$

により求められる．

以下では独立変数が 2 個の場合を考える．予測値 $\hat{Y}$ は予測平面

$$\hat{Y} = b_0 + b_1X_1 + b_2X_2 \qquad (2)$$

上にある．

実測値 Y との差 (残差)$e_k = Y_k - \hat{Y_k}$ は正負の符号を持つので，その 2 乗和が最小になるように (最小二乗法) 独立変数にかける重み b_i(偏回帰係数, 最小二乗推定値) および定数項 b_0 を定める．つまり，次の Q を最小にする．

$$\begin{aligned} Q &= \sum_{k=1}^{n} e_k^2 = \sum_{k=1}^{n} \left(Y_k - \hat{Y_k}\right)^2 \\ &= \sum_{k=1}^{n} \{Y_k - (b_0 + b_1X_{k1} + b_2X_{k2})\}^2 \end{aligned} \qquad (3)$$

(3) 式を，b_0, b_1, b_2 で偏微分して 0 とおく．

$$\begin{cases} \frac{\partial Q}{\partial b_0} = -2\sum\{Y_k - (b_0 + b_1X_{k1} + b_2X_{k2})\} = 0 \\ \frac{\partial Q}{\partial b_1} = -2\sum X_{k1}\{Y_k - (b_0 + b_1X_{k1} + b_2X_{k2})\} = 0 \\ \frac{\partial Q}{\partial b_2} = -2\sum X_{k2}\{Y_k - (b_0 + b_1X_{k1} + b_2X_{k2})\} = 0 \end{cases} \qquad (4)$$

(4) 式に，変数 Y, X_1, X_2 の平均値を $\bar{Y}$, $\bar{X_1}$, $\bar{X_2}$ としたときの関係式

$$b_0 = \bar{Y} - b_1\bar{X_1} + b_2\bar{X_2} \qquad (5)$$

および，独立変数 X_i, X_j 間の変動・共変動

$$S_{ij} = \sum_{k=1}^{n} \left(X_{ki} - \bar{X_i}\right)\left(X_{kj} - \bar{X_j}\right) \qquad (6)$$

および，独立変数 X_i と従属変数 Y の共変動

$$S_{iy} = \sum_{k=1}^{n} \left(X_{ki} - \bar{X_i}\right)\left(X_k - \bar{Y}\right) \qquad (7)$$

を代入して整理すると，

$$\begin{cases} b_1S_{11} + b_2S_{12} = S_{1y} \\ b_1S_{21} + b_2S_{22} = S_{2y} \end{cases} \qquad (8)$$

という連立方程式が得られ，これを解くことにより偏回帰係数 b_1, b_2 が求まる．定数項 b_0 は (5) 式から求められる．以降この b_i を求めることを回帰分析を行うと表現する．

一般的には，独立変数間の変動・共変動行列を $\mathbf{S}$，独立変数と従属変数間の共変動ベクトルを $\mathbf{c}$，偏回帰係数ベクトルを $\mathbf{b}$ として，(9) 式のようになる．

$$\mathbf{Sb} = \mathbf{c} \qquad (9)$$

$\mathbf{S}$ の逆行列を $\mathbf{S}^{-1}$ とすれば，偏回帰係数は (10) 式で

求められる．

$$\mathbf{b} = \mathbf{S}^{-1}\mathbf{c} \tag{10}$$

また，上記はすべて線形についての議論であるが，指数関数や対数関数においても同様に最小二乗法を用い，回帰分析を行うことが可能である．

2.2 検　　定

回帰分析を行った場合，得られた回帰式が有意であるかの検定を行う必要がある．1 つ目は偏回帰係数と定数項について，すなわち「求められた偏回帰係数および定数項が 0 である（$b_i = 0 (i = 0, 1, 2, \cdots, p)$）」という帰無仮説を棄却できるかの検定である．棄却できない場合は求められた値が意味をなさないものとなる．2 つ目は回帰の分散分析の過程において，「分析に使用した独立変数 X_i で，従属変数 Y は説明できない」という帰無仮説を棄却できるかの検定である．棄却できない場合は回帰に用いた X_i だけでは Y を予測するのに不十分となり，回帰式自体の意味が失われる．

2.2.1 偏回帰係数と定数項の検定

まず，帰無仮説 H_0:「$b_i = 0 (i = 1, 2, \cdots, p)$」について検定する．$\mathbf{S}^{-1}$ の要素を s^{ii} と表すと，偏回帰係数 b_i の標準誤差 $SE(b_i)$ は，表 1 の MS_e を用いて

$$SE(b_i) = \sqrt{s^{ii} MS_e} \tag{11}$$

で表せ，

$$t_0 = |b_i| / SE(b_i) \tag{12}$$

は，自由度が $n - p - 1$ の t 分布に従う．b_i が有意である確率は

$$P_0 = Pr\{|t| \geq t_0\} \tag{13}$$

となり，$P_0 \leq \alpha$ のとき，帰無仮説を棄却でき，偏回帰係数は 0 でない，つまり得られた b_i を用いることは妥当であるといえる．ここで，α は有意水準とよばれ，検定の精度を表し，通常 5% の値を用いる．

また，帰無仮説 H_0:「$b_0 = 0$」についても同様に，

$$SE(b_0) = \sqrt{\left(1/n + \sum_{i=1}^{p}\sum_{j=1}^{p} \bar{X}_i \bar{X}_j s^{ij}\right) MS_e} \tag{14}$$

$$t_0 = |b_0| / SE(b_0) \tag{15}$$

$$P_0 = Pr\{|t| \geq t_0\} \tag{16}$$

となり，$P_0 \leq \alpha$ のとき帰無仮説を棄却でき，定数項は 0 でない，つまり得られた b_0 を用いることは妥当であるといえる．

2.2.2 回帰モデルの検定

回帰分析を行った場合には，通常回帰が有意か否かを確認するために表 1 の分散分析表を作り，従属変数 Y の分散は X_i による回帰によって説明できる部分（表 1 中の回帰）と，説明できない部分（表 1 中の残差）に分解し，検定を行う．H_0:「分析に使用した独立変数 X_i で，従属変数 Y は説明できない」という帰無仮説については，表 1 中の F 値は自由度が $(p, n-p-1)$ の F 分布に従うため，有意確率を P_0 とすると，$P_0 \leq \alpha$ が成立するかどうかが検定の実際の計算となる．成立すれば帰無仮説を棄却でき，分析に使用した独立変数 X_i で従属変数 Y が説明できると言えることになる．

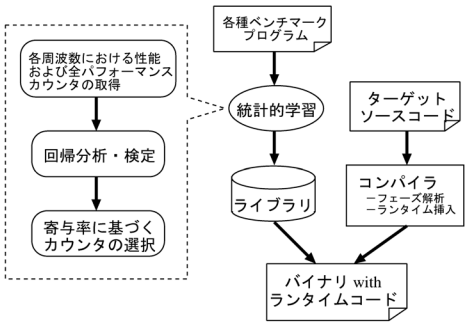

図 1　コンパイラ協調型 DVFS 手法の概要

表 1　回帰の分散分析表

変動要因	平方和	自由度	平均平方	F 値
回帰	S_r	p	$MS_r = S_r/p$	MS_r/MS_e
残差	S_e	$n-p-1$	$MS_e = S_e/(n-p-1)$	
全体	S_t	$n-1$	$MS_t = S_t/(n-1)$	

$S_e = \sum (Y_k - \hat{Y}_k)^2$,　$S_t = \sum (Y_k - \bar{Y})^2$,　$S_r = S_t - S_e = \sum b_i S_{iy}$

2.3 重相関係数と寄与率

従属変数の全変動のうち，回帰によって説明できる割合 (寄与率) は重相関係数の 2 乗 (R^2:決定係数) に等しく，次式で定義される．

$$R^2 = 1 - S_e/S_t \tag{17}$$

以下すべて，決定係数を寄与率と記す．独立変数を増やしてゆけば，寄与率はだんだんと 1 に近付くので，寄与率が高くなったのが追加された独立変数の効果なのかどうかがわからなくなる．このため，自由度調整済みの重相関係数の 2 乗 (R^{2*}) が定義される．

$$R^{2*} = 1 - \frac{S_e/(n-p-1)}{S_t/(n-1)} \tag{18}$$

$R^2 \neq 1$ である限り，R^{2*} は R^2 よりも小さい．R^{2*} が増加する限り，追加された独立変数は有効であることを意味する．

3. コンパイラ協調型 DVFS 手法

3.1 概　　要

本稿で提案するコンパイラ協調型 DVFS 手法は，パフォーマンスカウンタの情報をもとに周波数・電圧変更時の性能を予測し，性能低下を決められた範囲内に抑えつつ，低周波数でプログラムを実行し，消費電力を削減するものである．図 1 に本手法の概要を示す．

まず，あらかじめ各種アプリケーション実行中のパフォーマンスカウンタの値を取得し，それをもとに，周波数・電圧変更時の性能を予測する上で参照すべきカウンタと性能予測式を統計的に学習する．学習結果は

ライブラリに保存され，コンパイラはターゲットとなるアプリケーションのソースコードに対し，パフォーマンスカウンタの値を用いて，性能を予測するためのライブラリコールを挿入する．また，予測された性能にもとづき，周波数を変更するためのランタイムコードも挿入される．次節では，統計的学習手法，およびコンパイラによる DVFS 用コード挿入について詳述する．

3.2 統計的学習

学習を行うために，まず各種プログラム実行時の性能と，そのプラットフォームで取得可能な全てのパフォーマンスカウンタの値を DVFS 可能な全周波数の場合について計測する．

ここで，現在の動作周波数 v を，周波数 u に変更するときの性能の変化を考える．動作周波数 v での性能を y_v，変化後の周波数 u での性能を y_v，最高周波数での性能を y_M とすると，動作時 v の最高周波数の性能に対する性能比は y_v/y_M，周波数を v から u に変化させたときの性能向上率は y_u/y_v となる．したがって，現在の動作周波数 v から，周波数 u に変化させたときの最高周波数に対する性能比 Y_u を次式で定義できる．

$$(y_v/y_M) \times (y_u/y_v) = Y_u \tag{19}$$

説明変数 X_{vi} をカウンタを取得した周波数 v におけるサイクル当たりのカウンタの値とし，性能予測対象の周波数 u の最高周波数に対する性能比を，被説明変数 Y_u とする．学習フェーズではまず，この Y_u を X_{vi} で重回帰分析して回帰式 f_v^u を求める．実行時には，X_{vi} から回帰式 f_v^u を用いて性能比の予測値 $\hat{Y}_u$ を得ることができる．

対象とするプラットフォームにが q 種類のパフォーマンスカウンタを持ち，同時に $p(p \leq q)$ 個のカウンタが計測可能とすると，回帰式 f_v^u は式 (20) の線形回帰モデルを考えて f_v^u を求め，それに基づいて式 (21) より，予測値 $\hat{Y}_u$ を得た．

$$Y_u = f_v^u(X_{v1}, X_{v2}, \cdots, X_{vp}) \tag{20}$$

$$\hat{Y}_u = f_v^u(X_{v1}, X_{v2}, \cdots, X_{vp}) \tag{21}$$

ここで，対象プラットフォームにおいて設定可能な周波数を m 通りとすると，v は DVFS により動作可能な全周波数である m 通り，u は最大周波数に対する性能比を考えるので $m-1$ 通りとなる．カウンタの組み合わせが ${}_q\mathrm{C}_p$ 通り，一つのカウンタの組み合わせにつき，$m(m-1)$ 通りの回帰式が存在し，このカウンタの組み合わせの中から全体の性能比予測に最適と思われるものを，寄与率を参考に選択する．

以上の学習を行うことで，ある周波数 v で p 個のカウンタの値が得られた場合，周波数 u で動作した場合の性能比の予測値 $\hat{Y}_u$ が求められるようになる．

3.3 コンパイラによる実行時コード挿入

まず，コンパイラはターゲットアプリケーションのソースコードを解析し，ループなどの同一の挙動を示すコード領域 (以降，フェーズと呼ぶ) を見つける．あるフェーズを実行中は，最適な周波数・電圧設定も同じであると考えられるため，フェーズを単位として周波数・電圧を変更する．

```
set_freq(freq_phase_i);
start_perf_counter();
      :
(computation of phase_i)
      :
end_perf_counter();
target_freq = freq_0;
foreach freq_new (freq_1, freq_2, ...) {
   P = estimate_perf_ratio(freq_phase_i, freq_new);
   if (P < P_threshold)
      break;
   target_freq = i;
}
freq_phase_i = target_freq;
```

図 2 ランタイムコードの概要

次に各フェーズ毎にパフォーマンスカウンタを参照し，性能を予測するためのライブラリコール，および周波数変更のためのランタイムコードを挿入する．図 2 は，あるフェーズ “phase$_i$” のために挿入されるコードの概要を示したものである．具体的には，phase$_i$ を実行する直前に，周波数 (電圧) を設定するための関数 *set_freq()* および，パフォーマンスカウンタをリセットするためのライブラリコール *start_perf_counter()* を挿入する．図中，freq_phase$_i$ は周波数の設定値であり，初期値としては最高周波数を設定する．

phase$_i$ の処理終了後，カウンタの値を保持するための *start_perf_counter()* が呼ばれる．また，*estimate_perf_ratio()* は保持されたカウンタの値を回帰式 (21) に当てはめ，最大周波数で動作した場合に対する，freq$_{new}$ で動作させた場合の性能比を予測するためのライブラリコールである．あらかじめ，最高周波数に対して最低限維持すべき性能の比率を性能比閾値 ($\mathrm{P}_{threshold}$) として定義し，最大周波数の次の周波数設定値☆より順に性能比を予測しつつ，性能比閾値と比較することで，予測性能が性能比閾値未満にならない範囲で最低の周波数を求める．次回の phase$_i$ の実行時は，求められた周波数で動作する．

なお，全フェーズに対して上記の処理を行うとオーバヘッドが大きくなる恐れがあるため，比較的重い処理を行うフェーズのみに，上記の DVFS 手法を適用する．

4. 評　　価

4.1 評 価 環 境

本提案手法の有効性を調べるため，Pentium M 725

☆ 図 2 では freq$_0$ が最大周波数を表し，freq$_1$，freq$_2$ と順に周波数設定が低下することを仮定している．

プロセッサを搭載した PC を用い実験を行う．この Pentium M 725 は，32KB+32KB L1 cache, 2MB L2 cache, 400MHz bus clock であり，周波数と動作電圧の関係は表 2 のとおりである．周波数は 6 段階 ($m = 6$) であり，v =1.6GHz, 1.4GHz, 1.2GHz, 1.0GHz, 0.8GHz, 0.6GHz, u =1.4GHz, 1.2GHz, 1.0GHz, 0.8GHz, 0.6GHz となる．また，33 個のパフォーマンスカウンタ ($q = 33$) を持ち，同時に 2 つカウンタが計測可能 ($p = 2$) である．コンパイルは GCC3.3.4 (オプションは-O2) を用いて行い，カウンタ情報取得には PAPI (Performance Application Programming Interface)[8)9)] を用いた．

表 2 Pentium M 725 の周波数と動作電圧

周波数 (GHz)	1.60	1.40	1.20	1.00	0.80	0.60
動作電圧 (V)	1.484	1.420	1.276	1.164	1.036	0.956

4.2 学習用プログラムと適用評価

学習のために SPEC2000 ベンチマークの中から，整数系アプリケーションの 3 つ（176.gcc，181.mcf, 175.vpr），および浮動小数点系アプリケーションの 3 つ（179.art，183.equake，188.ammp）について，それぞれ実行時間の長い上位 5 つの関数を独立のフェーズと見なし，これら計 30 個 ($n = 30$) を今回の学習の対象とする．

また，学習が正しく行われ，ターゲットプログラムのコンパイルと実行を行う上で効果的に DVFS を行うことができるかを確認するため，学習に用いたアプリケーション以外のプログラムとしてブロッキングを行わない行列積を用い適用評価を行う．適用評価では，DVFS を行う上で特に性能比に影響すると思われる L2 キャッシュミス率を変更させて評価を行うべく，行列サイズを変化させて評価を行う．行列サイズは，行列の一辺が 50，100，500，1000 の場合を評価する．この際，各サイズの行列積を 20 回繰り返し，この繰り返しを一つのフェーズとして，周波数設定用コード，およびライブラリコールのためのコードを挿入する．

5. 評 価 結 果

5.1 統計的学習結果

前章で述べた実験環境では 2 つのカウンタ情報を同時に得ることが出来るので，カウンタは 33 個のうちの 2 個を用いた．また，回帰式 f_v^u は式 (22) の線形回帰モデルを考えて $b_{vi}^u (i = 0, 1, 2)$ を求め，それに基づいて式 (23) より，予測値 $\hat{Y}_u$ を得た．

$$Y_u = b_{v0}^u + b_{v1}^u X_{v1} + b_{v2}^u X_{v2} \quad (22)$$

$$\hat{Y}_u = b_{v0}^u + b_{v1}^u X_{v1} + b_{v2}^u X_{v2} \quad (23)$$

学習の結果，性能比 Y_u を予測するために最適なカウンタの組み合わせとして寄与率の大きい L2_DCR (Level 2 data cache reads) と L2_TCM (Level 2 cache misses) のカウンタが選択された．この組み合わせにおいて，有意水準 $\alpha = 5\%$ で検定を行たところ，帰無仮説の検定は棄却された．したがって，予測値 $\hat{Y}_u$ は次式で表される．

$$\hat{Y}_u = b_{v0}^u + b_{v1}^u\{(\texttt{L2\_DCR})\} + b_{v2}^u\{(\texttt{L2\_TCM})\} \quad (24)$$

ここで，L2_DCR と L2_TCM の 2 つのカウンタでの性能比 Y_u の寄与率を図 3 に示す．寄与率の幾何平均は 0.95 となり，実行時の 2 つのカウンタ値から性能比 $\hat{Y}_u$ が約 95% の精度で予測可能であることがわかる．

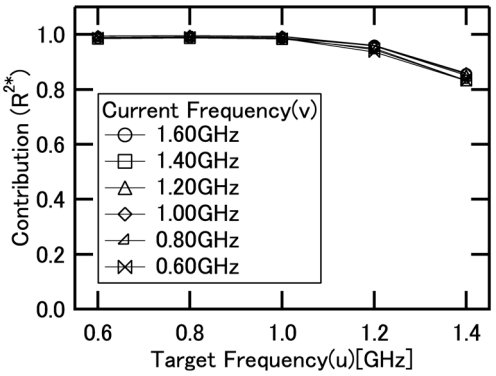

図 3 得られた回帰式の寄与率

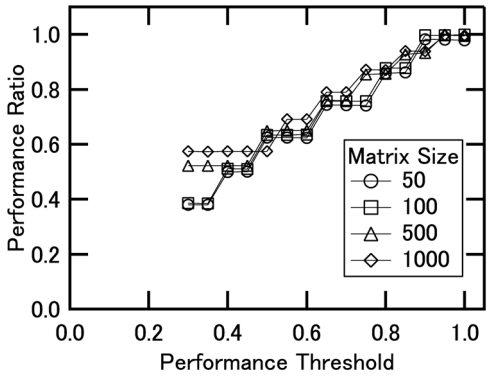

図 4 適用評価時の性能比の変化

5.2 適用評価結果

評価に用いた行列積の各行列サイズについて，性能比閾値を 0.30 から 1.00 まで 0.05 刻みで変化させた時の，実際にプログラム実行して得られた性能比の変化を図 4 に示す．ここで，性能比の基準は，DVFS 用コード，およびライブラリコールを挿入せずに，最高周波数 1.6GHz で動作させたときの性能である．

図 4 より，性能比閾値が 1.00 以外では全ての場合において，閾値以上の性能を達成しており，本手法における目的である最低限維持すべき性能を下回らないという制約は満たされている．したがって，統計的学習および，性能の予測とコンパイラによる周波数設定

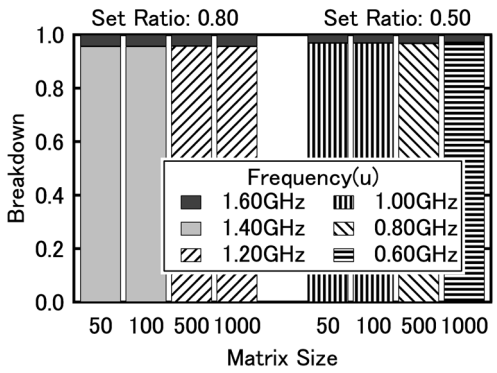

図 5 適用評価時の動作周波数の時間分布

の最適化が的確に行われていることがわかる．ただし，性能比閾値が 1.00 の場合，すなわち最高周波数である 1.6GHz で動作した場合に比べ性能低下を許さないとした場合では，わずかに性能が低下しており，行列サイズ 100 以上では 0.3%程度，行列サイズ 50 の場合では 1.3%程度性能が低下している．これは，実際には，常に 1.6GHz で動作しているにも関わらず，周波数設定用コードとライブラリコール挿入によるオーバヘッドが存在するためである．しかし，行列サイズが 100 以上程度というように，対象となるフェーズの実行が比較的重い場合には，ランタイムコード挿入によるオーバヘッドは無視できるほど小さい．

次に，どの程度周波数を低下させて実行できたかの割合を示すために，図 5 に性能比閾値 0.80，および 0.50 の場合における，実行時に各周波数で動作した時間の割合を示す．図より，最初に行列積を実行する際の周波数として，最大周波数である 1.6GHz を選択することから，1.6GHz で動作している割合が多少あるものの，大部分の時間を低い周波数で動作できていることがわかる．低い周波数で動作する場合，表 2 に示すように低い電圧で実行することができるため，実行時の消費電力削減が期待できる．たとえば，性能比閾値 0.80，行列サイズ 1000 の場合，4.4% の時間を 1.6GHz で，残りの時間は 1.2GHz で実行しているため，理論的には，プロセッサの消費エネルギーは 24.9% 削減できることになる．

このように，本稿での提案手法を用いることで，性能低下率をある閾値に抑えつつ，ほぼ最大限の消費電力削減効果を得ることができる．さらに，学習をし直すことで，異なるプラットフォームに対しても，同様に本手法を適用できることが期待される．この点で，本手法は極力性能低下を抑えつつ，消費電力を削減するための手法として，有効な手法であると考えられる．

6. まとめと今後の課題

本稿では，統計処理に基づくコンパイラ協調型 DVFS 手法を提案した．提案手法は，各種の学習対象アプリケーションに対し，性能とパフォーマンスカウンタの値との関係を重回帰分析により求め，性能を予測するための回帰式を導き，対象のソースコードに対し，カウンタ，および回帰式を用いて最高周波数に対する性能比を予測するコードを挿入し，適切な周波数・電源電圧で動作させるものである．

今後は，線形回帰モデルだけでなく，指数関数や対数関数を含むモデルに拡張することや，より多くのアプリケーションを用いて学習を行い，評価を行うことなどが課題である．

謝辞 本研究の一部は，科学技術振興機構・戦略的創造研究推進事業 (CREST) の研究プロジェクト「低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」，および文部科学省科学研究費補助金 (若手研究 (B) 17700049)，東レ科学振興会科学研究助成の支援によって行われた．

参 考 文 献

1) K. Krewell. Pentium M Hits the Street. In *Microprocessor Report*, Vol.17, No.3, March 2003.
2) K. Choi, *et al.* Fine-grained dynamic voltage and frequency scaling for precise energy and performance trade-off based on the ratio of off-chip access to on-chip computation times. In *Proc. of DATE*, Feb 2004.
3) G. Semeraro, *et al.* Dynamic frequency and voltage control for a multiple clock domain microarchitecture. In *Proc. of Micro-35*, November 2002.
4) C-H Hsu, *et al.* The design, implementation, and evaluation of a compiler algorithm for CPU energy reduction. In *Proc. of PLDI-2003*, June 2003.
5) Intel. The IA-32 Intel Architecture Software Developer's Manual Volume3: System Programing Guide, 245472. 2002.
6) J.L. Hennessy, *et al.* Computer Architecture: A Quantitative Approach, 3rd Edition. 2002.
7) Q. Wu, *et al.* A Dynamic Compilation Framework for Controlling Microprocessor Energy and Performance. In *Proc. of Micro-38*, November 2005.
8) PAPI group. PAPI Software Specification, Version 3.0.
9) S. Browne, *et al.* A Scalable Cross-Platform Infrastructure for Application Performance Tuning Using Hardware Counters. In *Proc. of SC'00*, November 2000.
10) 松原望. 統計学入門. 東京大学出版会, 1991.