統計情報に基づく動的電源電圧制御手法

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

プログラムの実行に関して種々の最適化に対する要求が強く求められてきており，特に実行時における最適化が重要視されている．我々はハードウェアから得られる動的な情報と，コンパイラから得られるプログラムのフェーズの区切りといった静的な情報を用いて精度の高い最適化手法を確立する．本稿では，実行時における最適化のためのモデリングに従来のような定性的な解析ではなく，定量的なデータを取得し，統計的な学習を用いるという手法を提案する．また，本手法を DVFS に適用し，許容される下限の性能を最高周波数時の性能との相対値，すなわち性能比閾値として与えたときに，許される範囲内で最適な周波数・電源電圧を選択し，消費電力を削減することを目的とする．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・電圧変更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．

Dynamic Voltage Scaling Method Based on Statistical Analysis

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

This paper proposes a novel DVFS technique based on statistical analysis. Our approach is hybrid in which dynamic hardware information and static information such as program behavior given by the compiler are used. In our technique, the relationships between the performance and the performance counter information are learned statistically in advance. Then, the compiler inserts the runtime code for predicting the performance of the target program and sets the appropriate frequency/voltage depending on the predicted performance. Our technique can reduce a great amount of energy consumption with keeping the performance degradation to a certain ratio which the user can expect.

1. はじめに

近年，プログラムの実行に関して種々の最適化が強く求められてきているが，とりわけ実行時における最適化の重要性が増してきている．一般に実行時，つまり動的な最適化手法を静的な最適化手法と比べた場合の利点は，データセットの違いや，ハードウェア構成などの違いからくる動的な振舞いに対処することが可能であり，効率的な実行を提供できるという点にある．たとえば最適化の対象として適切な周波数・電源電圧の選択や，キャッシュや命令キューのサイズ，マルチコア・マルチスレッドプロセッサにおける実行スレッド数の選択などがあげられる．アプローチの手段も様々であり，ハードウェア的なアプローチから，コンパイラ・OS などのソフトウェア的なものまでと多岐にわたっている．

OS による手法[9] やハードウェアによる手法[19] では，一般的に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舞いにも対応できる．しかし，フェーズの変化などのプログラムの情報を明示的に利用することはできないため，最適化が困難な場合がある．

また，コンパイラによる手法は，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うというものである[8]．これらの手法の問題点として，たとえばデータセットが異なっていた場合に，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振舞いが異なっていると有効に最適化が行えないことがあげられる．

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る．近年ではお互いの良い点を用いるハイ

† 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
☆ 現在，文部科学省
Presently with Ministry of Education, Culture, Sports, Science and Technology

ブリッドな手法が提案されており，より優れた実行時における最適化を行うことが可能であると考えられている．

一方で，こういった最適化を行うためには計算機の振舞いを解析・理解し，どのような場合にどういった最適化を行うかというモデルを立てることが必要である．しかし，計算機システムは年々加速的に複雑化してきている．たとえば命令実行においては，スーパパイプラインや高度な分岐予測によるアウト・オブ・オーダ実行などが行われている．また，SMT や CMP といったワンチップ上でリソースを共有しつつ複数のスレッドを実行可能なプロセッサも登場し，多階層のキャッシュメモリを有している．こういった事情から，計算機の振舞いを定性的に理解することが非常に困難になってきている[3),11)]．さらに複雑化していく計算機システムにおいて，定性的な解析を用いた最適化手法を生み出していくための時間およびコストは増大する一方である．たとえば，文献 20) では，あるハードウェアカウンタの情報を指標として性能モデルを作成し，周波数・電源電圧を制御する DVFS 手法が提案されているが，プラットフォームが異なり，もし当該ハードウェアカウンタが備えられていない場合は，新たに性能モデルを作成し直す必要がある．また，当該カウンタがある場合でも，性能モデルに用いられている数式のパラメータなどは，プラットフォームごとに探索する必要があり，定性的にそれらを構築することには限界があると考えられる．

そこで我々は，このような問題に対処しつつ種々の最適化を可能にするために，計算機システムの振舞いを定量的に解析し実行時に最適化を行う手法を提案する．具体的には，計算機の振舞いを示す様々なハードウェアイベントの定量的な値から統計的な学習を行い最適化実行のためのモデルを作成する．そのうえで，実行時にはそれらハードウェアイベントにおける値を指標として最適化を実行する．ハードウェアイベントの定量的な測定には，最近のほとんどのプロセッサに搭載されているパフォーマンスカウンタと呼ばれる機構を用いる．パフォーマンスカウンタにより，キャッシュのヒット/ミス回数や，分岐予測ミス回数などがソフトウェアなどから計測可能となっている．

本稿では提案手法を Intel Pentium M [15)] を用いた実機のプラットフォーム上で DVFS 手法として適用し，適切なモデルを統計的に確立し，最適な周波数・電源電圧の選択を実現する．本稿で提案する統計処理に基づく DVFS 手法は，パフォーマンス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うえでなるべく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も，特徴的なアプリケーション群をひととおり実行し，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．

本稿の構成は以下のとおりである．次章において，統計的学習に用いる重回帰分析について説明し，3 章では提案する統計情報に基づく DVFS 手法について述べる．4 章では評価環境および評価条件について説明し，5 章で評価結果を示す．6 章で統計情報に基づいた実行時最適化手法の応用可能性について言及し，7 章で関連研究についてまとめ，8 章で本稿のまとめと，今後の課題について述べる．

2. 重回帰分析

本章では本稿で用いる統計的処理である重回帰分析[12),21)] 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．

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

$$\hat{Y} = b_0 + b_1 X_1 + b_2 X_2 \tag{1}$$

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 2 乗法）独立変数にかける重み b_i（偏回帰係数，最小 2 乗推定値）および定数項 b_0 を定める．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

$$b_0 = \bar{Y} - b_1 \bar{X_1} + b_2 \bar{X_2} \tag{2}$$

および，独立変数 X_i，X_j 間の変動・共変動 $S_i j$ と独立変数 X_i と従属変数 Y の共変動 $S_i y$ を代入して整理すると，

$$\begin{cases} b_1 S_{11} + b_2 S_{12} = S_{1y} \\ b_1 S_{21} + b_2 S_{22} = S_{2y} \end{cases} \tag{3}$$

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 は式 (2) 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．

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 2 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 2 つ行う必要がある．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．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．

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．b_i が有意である確率は

$$P_0 = Pr\{|t| \geq t_0\} \tag{4}$$

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 は有意水準と呼ばれ，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，$P_0 \leq \alpha$ 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検定を行う．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 が説明できるといえることになる．

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）に等しい．以下すべて，決定係数を寄与率と記す．寄与率とは実際値と理論値の差を表しており，回帰モデルによって説明できた予測値の変動の割合を示す．寄与率が 100%に近い値であるほど，モデルのあてはまり具合いがいいことを意味する．独立変数を増やしてゆけば，寄与率は徐々に 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かどうかが分からなくなる．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．$R^2 \neq 1$ 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．

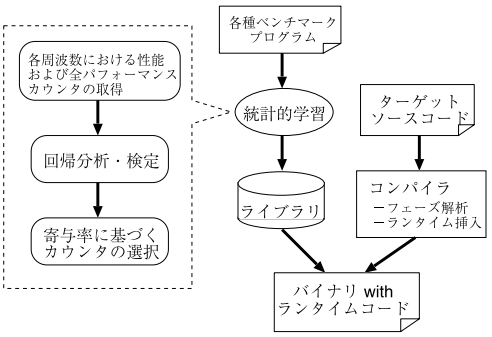

図 1 統計情報に基づく DVFS 手法の概要
Fig. 1 Overview of the proposed DVFS method based on statistical analysis.

3. 統計情報に基づく DVFS 手法

3.1 概　　要

動的電源電圧制御（*Dynamic Voltage and Frequency Scaling: DVFS*）手法は，電源駆動/バッテリ駆動の別，あるいはプロセッサのタスク処理要求の負荷などに応じて，動的にプロセッサのクロック周波数と電源電圧を調節する手法である．バッテリ駆動時間を長くしたい，あるいはメモリからのデータ待ちのため行うべきタスクが少なくプロセッサのアイドル状態が長いような場合には，プロセッサの周波数・電源電圧を下げ消費電力削減を狙う．CMOS 半導体のスイッチングに起因する消費電力は電源電圧の 2 乗に比例するため，プロセッサを低電圧で動作させることによって，大きな消費電力削減効果が期待される．本稿で提案する統計情報に基づく 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うえ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，予測された性能に基づき，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

本手法では，ターゲットのアプリケーションにおいて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）実行中における，最適な周波数・電圧設定は一定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．学習を行うために，各種プログラムのすべてのフェーズの実行時の性能と，そのプラットフォームで取得可能なすべてのパフォーマンスカウンタの値を DVFS 可能な全周波数について計測する．

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_u，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．

$$(y_v/y_M) \times (y_u/y_v) = Y_u \quad (5)$$

説明変数 X_{vi} をカウンタの値を取得した周波数 v におけるサイクルあ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．学習段階ではまず，この Y_u を X_{vi} で重回帰分析を行い回帰式 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（$p \leq q$）個のカウンタを計測可能であるとすると，式 (6) の線形回帰モデルを想定し回帰式 f_v^u を求め，それに基づいて式 (7) より，予測値 $\hat{Y}_u$ が得られる．

$$Y_u = f_v^u(X_{v1}, X_{v2}, \cdots, X_{vp}) \quad (6)$$

$$\hat{Y}_u = f_v^u(X_{v1}, X_{v2}, \cdots, X_{vp}) \quad (7)$$

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．カウンタの組合せが ${}_qC_p$ 通り，1 つのカウンタの組合せにつき，m $(m-1)$ 通りの回帰式が存在し，このカウンタの組合せの中から全体の性能比予測に最適と思われるものを，2.3 節で述べた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．

```
set_freq(freq_phase_i);
start_perf_counter();
      :
(computation of phase_i)
      :
end_perf_counter();

if (freq_phase_i ≠ freq_0)
  set_freq(freq_0);

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 = freq_new;
}
freq_phase_i = target_freq;
```

図 2 ランタイムコードの概要
Fig. 2 Pseudo-code of the run-time code.

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フェーズを発見する．次にコンパイラは各フェーズごとに，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 freq$_0$ を設定する．

phase$_i$ の処理終了後，カウンタの値を取得するための *end_perf_counter()* が呼ばれる．その後すぐに，*set_freq()* を呼び，現在の周波数が最高周波数でない場合，周波数を最高周波数に設定する．これは，後に続くランタイムコード，およびそれに続くフェーズ以外の処理が低い周波数で実行されてしまうと，大幅に性能が低下する恐れがあるため，フェーズの終了後には最高周波数に戻すように実装したためである．しかし，一方で周波数変更の回数が増加するため，周波数変更のオーバヘッドにより，かえって性能が悪化してしまうことも考えられる．したがって，後の評価ではフェーズの終了時に最高周波数に戻す場合，および戻さない場合の両者を評価し比較する．また，*estimate_perf_ratio()* は取得されたカウンタの値を回帰式 (7) にあてはめ，最大周波数での実行性能に対する周波数 freq$_{new}$ での実行性能比を予測するためのライブラ

表 1 Intel Pentium M 760 プロセッサのクロック周波数と電源電圧の関係
Table 1 Relationships between clock frequency and supply voltage of Intel Pentium M 760 processor.

Processor Clock [GHz]	2.00	1.86	1.73	1.60	1.46	1.33	1.20	1.06	0.80
Processor Core Vdd [V]	1.356	1.308	1.260	1.228	1.196	1.164	1.132	1.084	0.988

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（$P_{threshold}$）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

本手法の有効性を評価するため，Intel Pentium M 760 プロセッサを搭載した PC において実験を行う．この Pentium M プロセッサは，32 + 32 KB L1 cache, 2 MB L2 cache, 400 MHz bus clock であり，周波数と動作電圧の関係は**表 1** のとおりである．周波数は 9 段階（$m = 9$）であり，v = 2.00, 1.86, 1.73, 1.60, 1.46, 1.33, 1.20, 1.06, 0.80 GHz, u = 1.86, 1.73, 1.60, 1.46, 1.33, 1.20, 1.06, 0.80 GHz 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 つのカウンタ値を計測可能（$p = 2$）である．カウンタ値の取得には PAPI（Performance Application Programming Interface）[1),7)] を用いる．

4.2 学習用プログラムと適用評価

様々な未知のアプリケーションの性能を予測するため，学習には様々な特徴を持つアプリケーションを選択する必要がある．また本稿で用いた Pentium M はキャッシュサイズも大きく比較的高性能な汎用のマイクロプロセッサであるため，キャッシュミス率が高くなるような大きなデータセットを持ち，なおかつ整数系，浮動小数点系にまたがり広い分野のアプリケーションを網羅しているベンチマークとして SPEC CPU2000 [2)] を選択する．SPEC CPU2000 の中から，整数系アプリケーションの 9 つ（gzip, vpr, gcc, mcf, eon, gap, vortex, bzip2, twolf），および浮動小数点系アプリケーションの 7 つ（swim, mgrid, mesa, art, equake, ammp, apsi）について，それぞれ 1 回あたりの実行時間が長い関数をフェーズとして学習を行う．この際，GNU gprof を用い実行時間が長い上位の関数を対象として，人手でフェーズを抽出した．なお，各ベンチマークは，ref, train, test の 3 種類のデータセットを入力として実行することで，カウンタ値の計測を行い，それら各々を独立のフェーズと見なして学習の対象とする．また，行列積演算とベクトル積演算のプログラムにおいてそれぞれ行列サイズ，ベクトルサイズを変更して実行・計測しこれらも学習の対象とする．上記によって計 341（$n = 341$）の学習用フェーズについてカウンタ値の計測を行った．それぞれ，コンパイルには GCC3.3.4（オプションは `-O2`）を用いた．

次に，提案手法を用いた DVFS 手法によって最適な周波数・電源電圧を選択し，消費電力を削減できることを確かめるための評価実験を行う．評価プログラムには SPEC CPU2000 の整数ベンチマークの mcf と bzip2，また浮動小数点ベンチマークの swim と mgrid（それぞれ，データセットには ref を用いる）および倍精度の行列積演算プログラム（行列 1 辺のサイズ = 50, 600, 1000）の 7 つを用いる．これら評価用のプログラムは，統計的な学習による結果をもとにキャッシュミス率に着目し，その値が比較的高いものから，低いものまで，振舞いの異なるベンチマークを含むように選択されている．また，本手法に関して十分な議論が可能となるように，これら評価用プログラムの適用評価における結果は，想定どおりに周波数・電源電圧を選択できたもの，一見性能予測がはずれているように見えるもの，そして本手法における限界を示しているものを含んでいる．

評価の際には，leave-one-out 法を用い，対象のベンチマークを除き学習をした学習結果を用い適用評価を行う．たとえば mcf の適用評価では，すべての学習用プログラムから mcf のみを除いたプログラム群を学習の対象とし学習を行った後に，mcf において評価を行うものである．これらの学習結果をもとに，性能を予測するためのライブラリコード，周波数変更のためのランタイムコードをソースコードに挿入する．また，性能比閾値 $P_{threshold}$（= 1.0, 0.9, 0.8, 0.5）の

☆ 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．

表 2 寄与率の大きいカウンタの組合せ
Table 2 Counter pairs of high contribution ratio.

Rank	Counters	Contribution Ratio
No. 1	L2 store misses & L2 total cache misses	0.868
No. 2	Requests for exclusive access to clean cache line & L2 total cache misses	0.848
No. 3	L2 instruction cache misses & L2 total cache misses	0.846

4 通りについて評価を行う．応用によってはフェーズごとに異なる性能比閾値を与える必要があり，また閾値もソフトリアルタイム制約を考えた場合には，絶対に守らなくてもよいと考えられる．しかし，本評価の第 1 の目的は，統計的にカウンタの値を学習することで，周波数を変更した際の計算システムの性能をモデル化できることを示すことである．そのため，プログラム全体に対して単一の性能比閾値を与えるというシンプルな評価を行い，その閾値を守れたかどうかで手法の有用性を議論する．

本評価で設定した課題が，そのまま現実的な最適化対象になりうることは少ないと考えられる．しかしながら，リアルタイム制約下での低消費電力化として，OS などによってまずは最高周波数で動作させ，デッドラインまでの時間を考慮しつつ次の動作周波数を決定するような場合には，本評価で設定した課題が応用できると考えられる．また，その際にはフィードバック制御を行うことができるため，性能比閾値は絶対守らなければならないものではなくなる．

5. 評価結果

5.1 統計的学習結果

前章で述べた評価実験環境では 2 つのカウンタ値を同時に取得することができるため，性能予測のためのカウンタは 33 個のうちの 2 個を用いた．また，回帰式 f_v^u は式 (8) 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 (9) より，予測値 $\hat{Y}_u$ を得た．

$$Y_u = b_{v0}^u + b_{v1}^u X_{v1} + b_{v2}^u X_{v2} \quad (8)$$

$$\hat{Y}_u = b_{v0}^u + b_{v1}^u X_{v1} + b_{v2}^u X_{v2} \quad (9)$$

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合せとして寄与率の最も大きい `L2 STM`（Level 2 store misses）と `L2 TCM`（Level 2 total cache misses）のカウンタの組合せが選択された．この組合せにおいて，有意水準 $\alpha = 5\%$で検定を行ったところ，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．

$$\hat{Y}_u = b_{v0}^u + b_{v1}^u(\texttt{L2 STM}) + b_{v2}^u(\texttt{L2 TCM}) \quad (10)$$

学習の結果，上記の 2 つのカウンタ値を用いた場合のそれぞれの周波数での性能に対する寄与率の幾何平均は 0.868 となり，実行時には 2 つのカウンタ値から性能 Y_u が約 86.8% の精度で予測可能であることが分かる．また，参考として寄与率の大きかったカウンタの組合せ上位 3 組，およびそのときの寄与率の幾何平均を**表 2** に示す．どの組合せにも Level 2 total cache misses が含まれている．L2 キャッシュミス率が高い場合，プロセッサがメモリからのデータを待ってストールする時間が長くなり，周波数を落としても性能があまり落ちないという知見がよく知られており，それと一致する結果となった．また，L2 store misses なども寄与率が高いものとして出現している．L2 キャッシュミス率が高い場合でも，ロードミスの回数が多いのか，ストアミスの回数が多いのかによって性能に与える影響は変わってくる．たとえばロードミスの回数が支配的ならば，ストアミス回数が支配的な場合に比べてストール時間は長くなり，周波数を落としたときの性能の変化は小さくなると考えられる．したがって，L2 キャッシュミス中のロードミスとストアミスの比率を見るために，L2 store misses という指標が選択されたと考えられる．また，L2 instruction cache misses のミス率が高い場合には，ミスから復帰した際に周波数が低いとパイプラインへの命令供給が遅くなってしまい，周波数を低くすると性能にマイナスに影響してしまう．したがって，これも性能への影響を予測するうえで重要な指標になっているためと思われる．なお，2 番目の組に含まれている Requests for exclusive access to clean cache line については，このカウンタが何を意味するかが不明であり，現時点での考察は難しい．

5.2 適用評価結果

本節では，適用評価の結果を示し，考察を行う．本稿では提案手法をフェーズとして選ばれたコード領域についてのみ用いたため，フェーズがアプリケーション全体に対してどの程度の割合を占めるかが全体の結果に大きく影響する．したがって，以下ではまずフェーズにおける結果のみを示す．なお，以下の評価におけるフェーズとは図 2 に示す “computation of phase$_i$” 領域だけでなくその外側のランタイムコードを含むものとする（つまり，図 2 全体）．したがって，ランタイ

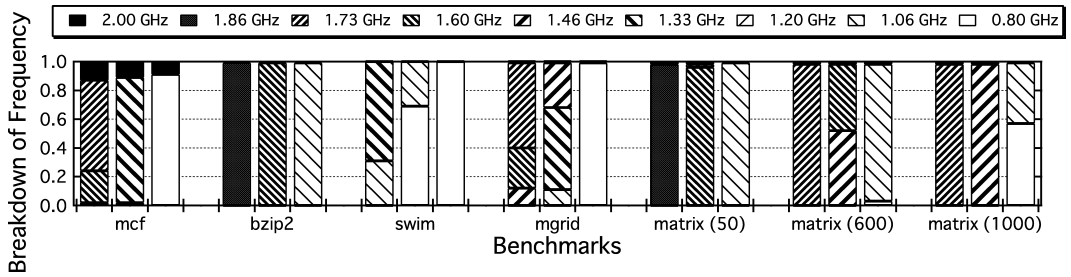

図 4 適用評価時の周波数の内訳
Fig. 4 Frequency breakdown.

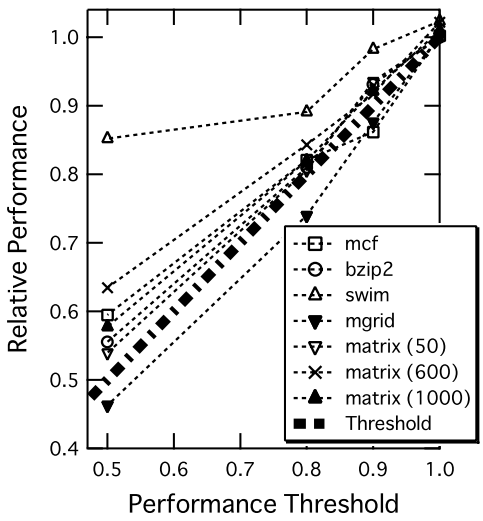

図 3 適用評価時の性能比の変化
Fig. 3 Variance of relative performance at applicable evaluation.

ムコードのオーバヘッドは評価結果に含まれている．以下図と表の中で，行列のサイズが 50 の行列積演算については matrix (50)，サイズが 600，1000 の場合も同様に matrix (600)，matrix (1000) と記す．各性能比閾値を設定し，本手法を適用した場合の最高周波数 2.00 GHz に対する性能比を**図 3** に示す．また，**図 4** にそれぞれの性能比閾値に対して，どの周波数でどれだけの割合動作したかの内訳を示す．それぞれのアプリケーションについて 3 つ示されている棒グラフは左から，性能比閾値 $P_{threshold} = 0.9$，0.8，0.5 の順に対応する．また，$P_{threshold} = 1.0$ のときはつねに最高周波数で動作するため，図には示していない．また，フェーズにおけるアプリケーションごとの特徴を**表 3** にまとめた．この表にはアプリケーションから抽出したスタティックなフェーズの数，それらがダイナミックに実行された回数の総和，また最高周波数時のフェーズの実行時間が全体の実行時間のうち占める割合，最高周波数でのフェーズの実行時間が示されている．スタティックなフェーズの数は対象となるプログラムから選択したフェーズの個数を，またダイナミックなフェーズの数はそれらが実際の実行中に何回実行されたか，つまり図 2 に示すランタイムコードが何回実行されたかを示している．図 3 において点線で描かれている直線は性能比がちょうど性能比閾値と一致する場合を表しており，この直線より上側にプロットされている点は最低限維持すべき性能を下回らないという評価実験の制約を満たしている．

図より mgrid を除く 6 つすべてのアプリケーションにおいて，目的を達成していることが分かる．これら 6 つのアプリケーションのうち swim を除いた 5 つについては，比較的各性能比閾値に近い性能で実行されており，高い精度で性能が予測でき，電源電圧と周波数を適切に選択することで，可能な限り消費電力が削減できていることが期待される．特に，行列積演算に対しては 3 つの異なるサイズすべてにおいて性能比閾値が 0.9，0.8 の場合は最大で 4%の差，性能比閾値が 0.5 においても最大で 12%程度の差しか生じていない．

一方，mgrid は各性能比閾値に対して，3–7%程度下回る性能で実行されている．この原因を調べるため，まず挿入したランタイムコードや周波数変更のオーバヘッドについて調査した．その結果 1 回あたりのランタイムコードは数 μs，周波数変更は平均して 1 段階あたり約 50 μs かかることが分かった．また，mgrid において性能比閾値を 0.9，0.8，0.5 と設定した際に周波数の遷移回数☆は合計でそれぞれ 58,084，140,120，280,130 回であった．すなわちランタイムコードおよび周波数変更によるオーバヘッドは性能比閾値 0.9 の場合 58084 × 約 50 [μs] = 約 2.97 [s] となる．同様に，性能比閾値 0.8，0.5 の場合のオーバヘッドはそれぞれ約 7.11 [s]，14.2 [s] である．これらのオーバヘッドによる時間は実行時間全体においてそれぞれ約 1.1，2.2，

☆ たとえば，最高周波数である 2.00 GHz から 2 段階遅い 1.73 GHz に遷移する場合の遷移回数は 2 回と表現する．

表 3 アプリケーションごとの特徴
Table 3 Execution characteristics of each benchmark.

Benchmarks	# of static phases	# wf dynamic phases	phase rate [%]	Time (whole) [s]
mcf	7	34	32.0	45.6
bzip2	7	758	85.0	40.3
swim	4	2399	86.4	262.9
mgrid	18	19025	91.8	238.5
matrix (50)	1	40	100.0	159.5
matrix (600)	1	40	100.0	411.7
matrix (1000)	1	40	100.0	516.3

表 4 最高周波数時に対する：消費エネルギーの相対値 ( ) 内は ED 積
Table 4 Relative energy against max frequency (ED product).

	性能比閾値	mcf	bzip2	swim	mgrid	matrix (50)	matrix (600)	matrix (1000)
フェーズのみ	0.9	85.7 (92.0)	93.1 (100.0)	69.2 (70.7)	84.2 (96.5)	93.2 (99.7)	86.7 (94.7)	86.7 (94.8)
	0.8	77.5 (89.0)	82.2 (101.5)	56.5 (63.7)	74.0 (100.6)	82.5 (102.3)	78.2 (96.6)	78.2 (95.5)
	0.5	57.2 (97.5)	64.1 (115.6)	53.2 (63.1)	53.2 (116.0)	64.4 (119.4)	64.1 (99.6)	64.2 (98.4)
プログラム全体	0.9	95.3 (98.3)	94.1 (100.0)	74.6 (74.7)	85.4 (96.9)	93.2 (99.7)	86.7 (94.7)	86.7 (94.8)
	0.8	92.5 (96.6)	84.4 (101.3)	61.8 (68.6)	75.7 (100.6)	82.5 (102.3)	78.2 (96.6)	78.2 (95.5)
	0.5	82.6 (99.6)	67.3 (113.3)	58.6 (68.1)	55.1 (114.8)	64.4 (119.4)	64.1 (99.6)	64.2 (98.4)

2.7%を占めており，点線を下回った一因である．しかし，このオーバヘッドだけでは性能比閾値を 3–7%下回った理由を完全に説明できていない．ここで，学習用プログラムを mgrid のみに限定し，回帰分析による統計的学習を行った結果，mgrid において最も相関が高いパフォーマンスカウンタの組合せは L1 data cache accesses と L2 total cache misses で，その寄与率は 95.3%であった．L1 data cache accesses は相関の高いパフォーマンスカウンタの組合せの上位 10 組に含まれており，mgrid の性能を予測するうえで非常に重要な指標であることが分かる．また，本適用評価で用いた L2 total cache misses と L2 store misses の組合せは mgrid に対する寄与率が 67.4%，相関の高さは 41 番目であり mgrid の性能を予測するには適していない指標であることが分かった．このように mgrid は学習に用いた多くのアプリケーションとは異なった振舞いをする．統計的学習によってすべてのアプリケーションに対して性能を正確に予測可能，つまり寄与率が 100%となることは現実的にありえないため，mgrid の適用評価結果は本手法のある限界点を示しているといえる．

このように性能を正確に予測することができないアプリケーションへの対策として，実際に本手法を適用した際にその性能を測定し，予測がはずれているようであればフィードバック制御を行い，次にそのフェーズが実行される際には予想される周波数より 1 段階異なる周波数を選択するなどして，動的に対応していくことなどが考えられる．次に，性能比閾値よりもはるかに高い性能を達成している swim について見てみると図 4 の右の棒グラフ，つまり性能比閾値 0.5 のとき，ほぼ 100%，最低周波数である 0.80 GHz で動作していることが分かる．このことから，swim においては性能予測がはずれているわけではなく，最低周波数で動作した場合においても性能がほとんど低下しないため，性能比閾値よりも高い性能を達成する結果になったということが分かる．

また，**表 4** に，最高周波数で動作した場合に対するそれぞれの性能比閾値を設定した場合の消費エネルギーの相対値 [%] と ED 積 [%] を示す．この表に示す値は，図 4 に示すそれぞれの周波数の実行時間の内訳と，表 1 に示すそれぞれの周波数に対する電源電圧の値を用い，消費エネルギーは電源電圧の 2 乗に比例するという関係式から算出した理論値である．

表の上の 3 行はフェーズのみの消費エネルギーの相対値と ED 積をそれぞれ性能比閾値 0.9，0.8，0.5 の場合に対して示しており，下の 3 行がアプリケーション全体の消費エネルギーの相対値と ED 積を示している．表から性能比閾値が 0.9 と設定した場合において平均して約 15%程度消費エネルギーを削減可能である．なるべく性能低下を引き起こさずに消費電力を削減するという要求は多く，本手法はそのような場合に特に効果的である．bzip2 や mgrid など ED 積の評価では 100%を上回ってしまう場合もあるが，リアルタイムアプリケーションなど一定の性能を達成できればよいという状況では，性能低下をある程度許容しつつ，消費電力削減のためにできる限り低い周波数で動作させることが目的となる．さらに，そのような場合において最高周波数に対する性能が正確かつ動的に予

表 5 性能比閾値 $P_{threshold} = 0.9$ のときの最高周波数時に対する性能比
Table 5 Relative performance against max frequency when $P_{threshold} = 0.9$.

	Condition	mcf	bzip2	swim	mgrid	matrix (50)	matrix (600)	matrix (1000)
フェーズのみ	normal	89.7	93.0	97.7	87.2	93.4	91.4	91.4
	keep *frequency*	85.2	93.7	98.7	87.4	93.5	91.4	91.4
	L2 TCM only	89.0	93.8	99.8	89.1	93.5	90.1	91.6
プログラム全体	normal	95.8	94.0	98.0	88.1	93.4	91.4	91.4
	keep *frequency*	91.5	94.1	93.8	87.9	93.5	91.4	91.4
	L2 TCM only	96.5	94.6	99.8	89.9	93.5	90.1	91.6

測でき，所望の性能が実行時に変わるような場合にも対応できる本手法の有効性は高いと考えられる．

アプリケーション全体について見てみると，mcf 以外の 6 つについてはフェーズのみの結果と比べてもほぼ変わらない削減率であり，提案手法は高い消費電力削減能力を有しているといえる．

5.3 比較評価

次に，フェーズの終了直後に最高周波数に戻すことによるオーバヘッド，ならびにその後の処理を最高周波数以外で実行した場合の影響を調べるため，フェーズ終了直後に最高周波数に設定する処理を省いたランタイムコード（図 2 中の *end_perf_counter()* の直後の *set_freq(freq$_0$)* を除いたもの）を用いて評価を行う．さらに，最も性能への寄与率が高いカウンタである L2 TCM（Level 2 total cache misses）のみを用いた評価を行い，カウンタを複数個用いることの効果を調べる．なお，この場合のランタイムコードは図 2 と同様のものを用いる．単一のカウンタ L2 TCM を用いた場合の学習の結果，寄与率の幾何平均は 0.839 であり，本評価環境においては L2 TCM のみでも比較的高い精度での性能予測が可能であると考えられる．

これら 2 通りの評価結果と，これまで評価してきた手法との結果を比較するために，性能比閾値 0.9 と設定した場合の最高周波数に対する性能比を**表 5** に示す．表中，normal がこれまでの評価結果を示し，keep *frequency* がフェーズの終了直後に最高周波数に戻さない場合の評価，L2 TCM only が L2 TCM のみを用いた評価の結果である．まず，keep *frequency* の結果と normal を比較してみると，すべてのアプリケーションにおいて大きな差は見られないことが分かる．これら評価に用いたアプリケーションにおいては，ほぼプログラム全体をフェーズに区切ることができたため，フェーズの終了後に低い周波数で動作することによる性能の低下がほとんど見られないためだと考えられる．ただし，mcf に関しては若干 normal よりも低い性能となっている．表中，“フェーズのみ” と “プログラム全体” におけるそれぞれの性能比閾値での消費エネルギーの相対値を比較すると mcf は特にその値に開きがあり，フェーズ以外の領域が多いことが分かる．このことから，フェーズの終了後に最高周波数に戻さなかったため低い周波数で動作する時間があり，その影響が周波数・電源電圧を変更するのに要する時間よりも支配的であったと考えられる．したがって，このようなプログラムの場合には，フェーズの終了直後に最高周波数に戻すことが性能低下を防ぐためには必要だといえる．次に，normal と L2 TCM only の結果を比較すると，mgrid 以外では大きく性能が変わらない結果となった．本評価に用いたプラットフォーム，プログラムでは L2 TCM のみだけでも，比較的高い精度で性能を予測可能であるといえる．mgrid はカウンタを 2 つ用いた場合よりも性能比閾値に近い結果となっており，これはカウンタ L2 store misses を用いない方がより正確に性能を予測できたためだと考えられる．

6. 統計情報に基づく実行時最適化の可能性

本稿では，全ハードウェアカウンタの情報を基に統計処理を行うことで実行時最適化をする手法を提案し，それを DVFS 手法に対して適用したが，従来より個別の最適化や性能などのモデル化のために統計情報を利用した研究が行われている．たとえば，文献 18) では，統計的な解析により SMT プロセッサにおけるリソース競合の影響をモデル化している．この研究では，Pentium 4 プロセッサを用い，線形の回帰分析を用いてハードウェアカウンタの値からリソース競合による性能の低下を予測する式を導いており，高い精度でリソース競合を予測できることが示されている．これを利用すれば，たとえば SMT プロセッサで複数のプロセスを同時に実行する際に，リソース競合による性能低下が少なくなるようなプロセスの組を実行時に見つけることもでき，OS のタスクスケジューリング手法として統計処理に基づく実行時最適化手法が応用できると考えられる．

一般的には，ハードウェアカウンタの値を用い統計的に学習を行うことで，周波数変更時の性能予測だけでなく，様々な場面において性能モデルや，各種のモ

デル構築を行うことが可能である．何らかの最適化を行う場合に，性能上，あるいは性能と電力との関係においてトレードオフが存在することが多く，そのような場合に最適化手法のオン／オフ，あるいは最適化にかかるパラメータの調整などを実行時に行えば，プログラムに応じてきめ細かな最適化を行うことができることも多い．たとえば，文献 5) においては，動的にキャッシュサイズを変更することによってデータへのアクセスサイクルを変更可能なキャッシュ構成を提案し，実行しているプログラムのキャッシュのミス率にあわせたキャッシュサイズの最適化を行うことで，性能を低下させることなく消費電力を削減する手法を提案している．また，文献 4) では Pipeline Balancing（PLB）と呼ぶ，同時発行命令数が可変なアーキテクチャ手法を提案している．

このような手法を対象として，本稿で提案するように，各パラメータ時の性能や電力とハードウェアカウンタとの関係を統計的に学習することで，容易に実行時最適化を行うことが可能であると考えられる．したがって，本手法は適用範囲が広く，DVFS 手法だけでなく幅広い最適化に対しても有効に働くと期待される．

7. 関 連 研 究

ハードウェアベースの DVFS 手法の歴史は古く，組み込みの分野からマイクロアーキテクチャの分野において広く研究が行われている[14),17),19)]．ソフトウェア的な手法としては OS が負荷状態を監視することによって制御される手法や[9)]，プロファイルを用いてプログラムの振舞いを解析し，定性的なモデルを立てることによって周波数・電源電圧の制御を行う手法も行われている[13)]．またそのほかに，OS による手法[9)] やハードウェアによる手法[19)] では，一般的に，ある固定されたタイムインターバルごとに，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周波数・電源電圧を決定するといった研究もさかんである．

本稿ではコンパイラによる静的な解析と，ハードウェアから得られる動的な情報を用いるハイブリッドな手法を提案しているが，似たようなアプローチでコンパイラによってプログラムをフェーズに区切り，プログラムの実行時にハードウェアの情報を用いて IPC を予測し，DVFS を行う研究もある[8)]．そのほかにも，実行時の最適化という観点から，ランタイム最適化コードという，実行時の状況により HW 構成の変更や実行コードの選択などを行うためのコードを用いた研究も存在する．これらは dynamic compilation と呼ばれ，実行時の状況により，命令シーケンスそのものを変更することが可能である．IBM DAISY[10)] や Intel IA32EL[6)]，Intel PIN[16)] などの dynamic compilation 用のソフトも多く開発されている．

文献 20) では，Intel PIN を用い，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．1 章でも述べたように，本研究における第 1 の目的は消費電力を削減することではなく，パフォーマンスカウンタから得られるハードウェアイベントの値を統計的に学習することにより，性能を予測するために必要なパフォーマンスカウンタの選択と予測式を自動的に求められることを示すことである．そのうえで，与えられた性能比閾値を下回らない中で最低の周波数で動作させ，消費電力削減を実現している．したがって性能低下の予測式を人手でモデル化している文献 20) とは，その点において異なっている．また，文献 20) で評価に用いている Pentium M プロセッサと本稿で使用している Pentium M とは製造プロセスをはじめ，キャッシュ構成やとりうる周波数・電源電圧の値などが異なっており，単純に定量的な値を比較をすることはできない．よって，本手法の，文献 20) に対する優位性は，文献 20) における性能などの予測式を統計的学習によって自動的に求めることができる点である．

また，定性的な解析が困難である SMT（Simultaneous MultiThreading）プロセッサにおけるリソース競合をモデル化するために統計的な手法を用いて定量的に解析する研究も行われている[18)]．

8. まとめと今後の課題

プログラムの実行に関して種々の最適化が強く求められてきており，特に実行時における最適化が重要視されている．我々はハードウェアから得られる動的な情報と，コンパイラによって得られるプログラムのフェーズの変化といった静的な情報を用いてより精度の高い最適化を行う．本稿では，実行時における最適化のためのモデルを立てる際に従来のような定性的な解析ではなく，定量的なデータを取得し統計的な学習を用いるという手法を提案した．また，本手法を広く知られている低消費電力化手法である DVFS に適用し，許容される下限の性能を最高周波数時の性能との相対値，すなわち性能比閾値として与えたときに，許される範囲内の比較的低い周波数・電源電圧を選択し，

消費電力が効果的に削減できることを確認した．このように，評価プラットフォームにおいてパフォーマンスカウンタの値と性能の関係を統計的に学習することによって，未知のプログラムの性能を予測するために必要なカウンタおよび，予測式を自動的に求められること，および適用評価の結果，高い精度で性能を予測できることを示すことができた．

本手法の手順として，まず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く．導いた回帰式，およびカウンタの値から最高周波数に対する性能比を予測するランタイムコードを対象のソースコードに対して挿入し，実行時にパフォーマンスカウンタの値とランタイムコードから適切な周波数・電源電圧を決定し動作させるものである．

本手法における今後の課題として，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，より優れた，オーバヘッドの少ないランタイムコードを提案することがあげられる．本評価においては Pentium M を用いたシステムのみにおいてその有用性を確かめるにとどまったが，異なるプラットフォーム上で評価を行い提案手法の有効性を確認することは今後の大きな課題である．また，他の定性的に解析しにくい最適化対象に対して統計的にモデルを立てて最適化を行うことなども今後の課題である．

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) PAPI group: PAPI Software Specification, Version 3.0.
2) The Standard Performance Evaluation Corporation (SPEC). http://www.specbench.org
3) Intel Corporation: IA-32 Intel Architecture Software Developer's Manual Volume 3: System Programming Guide (2002).
4) Bahar, R.I. and Manne, S.: Power and energy reduction via pipeline balancing, *ISCA* (2001).
5) Balasubramonian, R., Albonesi, D.H., Buyuktosunoglu, A. and Dwarkadas, S.: Memory hierarchy reconfiguration for energy and performance in general-purpose processor architectures, *MICRO* (2000).
6) Baraz, L., Devor, T., Etzion, O., Goldenberg, S., Skaletsky, A., Wang, Y. and Zemach, Y.: IA-32 Execution Layer: A two-phase dynamic translator designed to support IA-32 applications on Itanium-based systems, *MICRO* (2003).
7) Browne, S., Dongarra, J., Garner, N., London, K.S. and Mucci, P.: A Scalable Cross-Platform Infrastructure for Application Performance Tuning Using Hardware Counters, *SC* (2000).
8) Chheda, S., Unsal, O.S., Koren, I., Krishna, C.M. and Moritz, C.A.: Combining compiler and runtime IPC predictions to reduce energy in next generation architectures, *Conf. Computing Frontiers* (2004).
9) Choi, K., Soma, R. and Pedram, M.: 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, *DATE* (2004).
10) Ebcioglu, K. and Altman, E.R.: DAISY: Dynamic Compilation for 100% Architectural Compatibility, *ISCA* (1997).
11) Hennessy, J.L. and Patterson, D.A.: Computer Architecture: *A Quantitative Approach*, 3rd Edition, Morgan Kaufmann (2002).
12) Hogg, R.V., Craig, A. and McKean, J.W.: *Introduction to Mathematical Statistics*, 6th Edition, Prentice Hall (2004).
13) Hsu, C.-H. and Kremer, U.: The design, implementation, and evaluation of a compiler algorithm for CPU energy reduction, *PLDI* (2003).
14) Hughes, C.J. and Adve, S.V.: A Formal Approach to Frequent Energy Adaptations for Multimedia Applications, *ISCA* (2004).
15) Krewell, K.: Pentium M Hits the Street, *Microprocessor Report*, Vol.17 (2003).
16) Luk, C.-K., Cohn, R.S., Muth, R., Patil, H., Klauser, A., Lowney, P.G., Wallace, S., Reddi, V.J. and Hazelwood, K.M.: Pin: Building customized program analysis tools with dynamic instrumentation, *PLDI* (2005).
17) Marculescu, D.: On the use of microarchitecture-driven dynamic voltage scaling (2000).
18) Moseley, T., Grunwald, D., Kihm, J.L. and Connors, D.A.: Methods for Modeling Resource Contention on Simultaneous Multithreading Processors, *ICCD* (2005).
19) Semeraro, G., Albonesi, D.H., Dropsho, S., Magklis, G., Dwarkadas, S. and Scott, M.L.: Dynamic frequency and voltage control for a multiple clock domain microarchitecture, *MICRO* (2002).

20) Wu, Q., Martonosi, M., Clark, D.W., Reddi, V.J., Connors, D., Wu, Y., Lee, J. and Brooks, D.: A Dynamic Compilation Framework for Controlling Microprocessor Energy and Performance, *MICRO* (2005).
21) 松原 望：統計学入門，東京大学出版会 (1991).

(平成 18 年 5 月 8 日受付)
(平成 18 年 7 月 13 日採録)

佐々木 広（学生会員）
2003 年東京大学工学部計数工学科卒業．2005 年同大学大学院情報理工学系研究科修士課程修了．現在，同大学院工学系研究科博士課程在学中．

浅井 雅司
2004 年東京大学工学部計数工学科卒業．2006 年同大学大学院情報理工学系研究科修士課程修了．現在，文部科学省科学技術・学術政策局政策課勤務．

池田 佳路（学生会員）
2005 年東京大学工学部計数工学科卒業．現在，同大学大学院情報理工学系研究科修士課程在学中．

近藤 正章（正会員）
1998 年筑波大学第三学群情報学類卒業．2000 年同大学大学院工学研究科博士前期課程修了．2003 年東京大学大学院工学系研究科先端学際工学専攻修了．博士（工学）．独立行政法人科学技術振興機構戦略的創造研究推進事業 CREST 研究員を経て，現在東京大学先端科学技術研究センター特任助手．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，ディペンダブルコンピューティングの研究に従事．電子情報通信学会，IEEE，ACM 各会員．

中村 宏（正会員）
1985 年東京大学工学部電子工学科卒業．1990 年同大学大学院工学系研究科電気工学専攻博士課程修了．工学博士．同年筑波大学電子・情報工学系助手．同講師，同助教授を経て，1996 年より東京大学先端科学技術研究センター助教授．この間，1996〜1997 年カリフォルニア大学アーバイン校客員助教授．高性能・低消費電力プロセッサのアーキテクチャ，ハイパフォーマンスコンピューティング，ディペンダブルコンピューティング，ディジタルシステムの設計支援の研究に従事．情報処理学会より論文賞（平成 5 年度），山下記念研究賞（平成 6 年度），坂井記念特別賞（平成 13 年度）各受賞．IEICE，IEEE，ACM 各会員．