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

2006－ARC－169 (30)
2006／8／2

統計情報に基づく実行時最適化の検討

佐々木 広† 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提案する．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最適化のためのランタイムコードを挿入し，実行時に最適化を行う．本稿では，提案手法を用いて効果的な 1) DVFS および 2) HW プリフェッチの制御を行い，その有効性を実証する．

Dynamic Optimization Method based on Statistical Analysis

HIROSHI SASAKI,† YOSHIMICHI IKEDA,† MASAAKI KONDO† and HIROSHI NAKAMURA†

The importance and demand of optimization techniques of program execution is growing strongly. Especially, dynamic optimization techniques are emphasized heavily. We propose a dynamic optimization 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 this paper, we apply our proposed technique to 1) DVFS and 2) control of hardware prefetch mechanism on a real system and demonstrate the effectiveness.

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る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．例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挙げられる．

OS による手法[7]やハードウェアによる手法[14]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る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[6]．これらの手法の問題点として，例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る舞いが異なっていると有効に最適化が行えないことが挙げられる．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る舞いを解析・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．例えば命令実行においては，スーパーパイプラインや高度な分岐予測によるアウト・オブ・オーダー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，計算機の振る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][9]．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．

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る舞

† 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

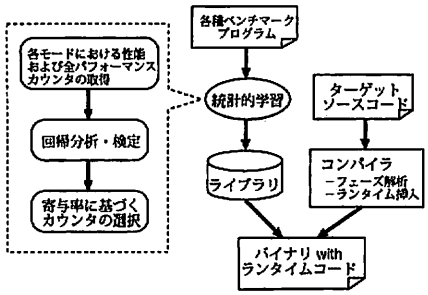

図 1 統計情報に基づく実行時最適化手法の概要

いを定量的に解析し実行時に最適化を行う手法を提案している[15]．具体的には，計算機の振る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．その上で，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．これまでに，提案手法を「統計情報に基づく動的電源電圧制御手法」として，Intel Pentium M[11] における実機のプラットフォーム上 DVFS 手法として適用し，適切なモデルを統計的に確立し，効果的な周波数・電源電圧の選択を実現している[15]．

以前の評価は，Pentium M を用いた単一のプラットフォーム上においてのみ行っており，提案手法の有効性を示すには不十分な面があった．そこで，本稿では提案手法を用いた DVFS 手法の従来プラットフォームにおける評価に加えて，AMD Opteron を用いたプラットフォーム上での評価を行う．さらに，DVFS 以外の最適化に対しても提案手法を用いて適切なモデルが構築できることを示すために，ハードウェアプリフェッチの有効/無効 (on/off) を動的に制御することによる性能の最適化について検討する．

2. 統計情報に基づく実行時最適化手法

本節では，我々の提案する統計処理に基づく実行時最適化手法について説明する．

2.1 概　　要

図 1 に提案手法の概要を示す．まず，あらかじめ最適化対象の各モード (例えば，取りうる各周波数・電圧) での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と性能を取得する．それを基に，モードを変更した際に性能がどう変化するか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，予測された性能に基づき，モードを変更するためのランタイムコードも挿入する．次節以降では，統計的学習手法，およびコンパイラによるコード挿入について詳述する．

```
set_mode(mode_phase_i);
start_perf_counter();
    :
(computation of phase_i)
    :
end_perf_counter();

for (j = 1; j != num_mode; j++) {
  P_j = estimate_perf(mode_phase_i, mode_j);
}

mode_phase_i = select_mode(P);
```

図 2 ランタイムコードの概要

2.2 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) 実行中では，最適な性能が得られるモードは一定であると考えられるため，このフェーズを単位として各モードを選択する．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設定可能な各モードにおいて計測する．

得られたカウンタ値を独立変数とし，異なるモードへ変更した際の性能比率を従属変数として重回帰分析[10] を行うことで，性能を予測するための回帰式を求める．それを用い，現在のカウンタの値から次のフェーズにおける各モードでの性能を予測し，次のフェーズでは最適と予測されたモードで実際に実行を行う．なお，用いた統計手法の詳細については，文献[15] を参照されたい．

2.3 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，およびモード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，モード (例えば周波数や，プリフェッチの on/off) を設定するための関数 *set_mode()* 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，$mode\_phase_i$ はモードの設定値である．そして，$phase_i$ 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．

また，*estimate_perf* は取得されたカウンタの値を回帰式に当てはめ，現在のモードのカウンタ値から，他のモード ($mode_j$) 時の性能 (P_j) を予測するためのライブラリコールである．

次に，select_mode(P) により，予測された各モードの性能値から，この phase_i の次回実行時にどのモードを用いるべきかを決定する．なお，select_mode() は実際にどのような目的で最適化を行うかに依存するコードとなる．例えば，文献[15] における DVFS 手法では，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として定義し，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ものである．

なお，全フェーズに対して上記の処理を行うとオーバー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，上記を適用する．

2.4 DVFS 手法への適用

提案する統計情報に基づく実行時最適化手法を，DVFS に適用する例を示す．この DVFS 手法は，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ことが目的である．

まず，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．

$$(y_v/y_M) \times (y_u/y_v) = Y_u \quad (1)$$

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．2.3 節で述べた統計的学習ではまず，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，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 (2) より，予測値 $\hat{Y}_u$ が得られる．

$$\hat{Y}_u = f_v^u(X_{v1}, X_{v2}, \cdots, X_{vp}) \quad (2)$$

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 ${}_qC_p$ 通り，1 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，回帰によって説明できる割合を意味する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. 評　　価

提案手法の有効性を評価するため，DVFS 手法における周波数制御を行う．さらに，ハードウェアプリフェッチの on/off 制御に適用するためのモデルを構築し，その妥当性について検討する．

3.1 評価環境

DVFS の評価では，Intel Pentium M 760，および AMD Opteron 150 プロセッサを搭載した PC において実験を行う．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．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．Opteron プロセッサは 64+64 KB L1 cache, 1 MB L2 cache であり，周波数と動作電圧の関係は表 1 のとおりである．本 Opteron プロセッサはパフォーマンスカウンタを 29 個持っており，同時に 4 つのカウンタ値を計測可能 ($p = 4$) である．

プリフェッチの評価では，Intel Xeon 3.06 GHz プロセッサを用い，その構成は 12 KB Trace cache, 8 KB L1 D cache, 512 KB L2 cache, 1 MB L3 cache である．

カウンタ値の取得には PAPI(Performance Application Programming Interface)[1][5] を用いる．

3.2 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．そこで，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み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．Opteron については，挙動が不自然な整数ベンチマークをいくつか除き，学習を行った．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，実

表 1 Relationships between clock frequency and supply voltage

Pentium M	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
Opteron	Processor Clock [GHz]	2.40	2.20	2.00	1.80	1.60	1.40	1.20	1.00	
	Processor Core Vdd [V]	1.500	1.450	1.400	1.350	1.300	1.250	1.200	1.150	

際に性能がどう変化するか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，また浮動小数点ベンチマークの swim (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 (行列一辺のサイズ = 50, 600, 1000) の 5 つを用いる．

また，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結果を用い適用評価を行う．例えば 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，モード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，DVFS 手法において最高周波数に対して最低限維持すべき性能の比率である性能比閾値は 1.0, 0.9, 0.8, 0.5 の 4 通りについて評価を行う．

4. 評価結果

4.1 統計的学習結果

Pentium M は 2 つのカウンタ値を同時に取得することが出来るため，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．同様に，Opteron は性能予測のために 4 個のカウンタを用いた．また，回帰式 f_v^u は線形回帰モデルを考えて $b_{vi}^u (i = 0, 1, 2)$ を求め，それに基づいて予測値 $\hat{Y}_u$ を得た．

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 Pentium M では寄与率の最も大きい L2 STM (Level 2 store misses) と L2 TCM (Level 2 total cache misses) のカウンタが選択された．また，Opteron では L2 STM (Level 2 store misses), BR MSP (Conditional branch instructions mispredicted), TOT INS (Instructions completed), そして L1 TCR (Level 1 instruction cache reads) が選択された．したがって，それぞれの予測値 $\hat{Y}_u$ は次式で表される．

$$\hat{Y}_u = b_{v0}^u + b_{v1}^u(\mathrm{L2\ \ STM}) + b_{v2}^u(\mathrm{L2\ \ TCM}) \quad (3)$$

$$\hat{Y}_u = b_{v0}^u + b_{v1}^u(\mathrm{L2\ \ STM}) + b_{v2}^u(\mathrm{BR\ \ MSP}) + b_{v3}^u(\mathrm{TOT\ \ INS}) + b_{v4}^u(\mathrm{L1\ \ TCR}) \quad (4)$$

学習の結果，Pentium M 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．同様に，Opteron における寄与率の幾何平均は 0.733 となる．

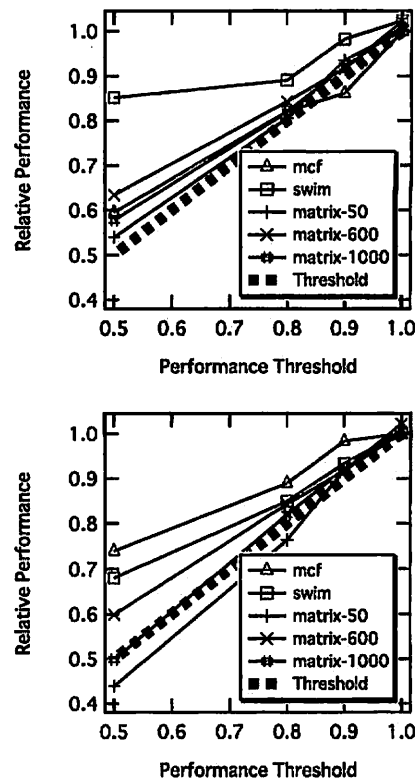

図 3 適用評価時の性能比の変化 (上: Pentium M 下: Opteron)

4.2 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，紙面の都合，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，ランタイムコードのオーバー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

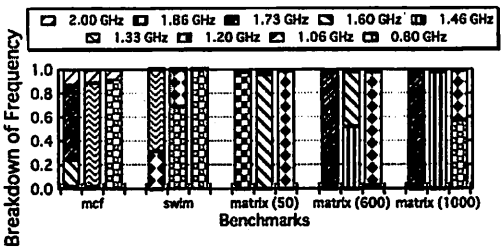

図 4 適用評価時の周波数の内訳 (Pentium M)

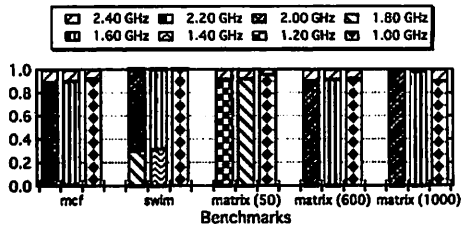

図 5 適用評価時の周波数の内訳 (Opteron)

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に対する性能比を図 3 に示す. また, 図 4, 図 5 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 の時は常に最高周波数で動作するため, 図には示していない. 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 Pentium M においては mcf, Opteron においては matrix-50 を除く全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. これらのうち swim を除いた 4 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いる. 図より, 評価に用いた 2 つのプラットフォーム上で提案手法を用いることによって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.

また, 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一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 図 5 の両方において右の棒グラフ, 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 2 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 [%] を示す. この表に示す値は, 図 4 図 5 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 Pentium M で約 15%程度, Opteron で約 10.0%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.

4.3 プリフェッチ手法の学習結果

次に, Xeon を用いたプリフェッチによる性能最適化のためのモデル構築の学習結果を示す. 学習の結果, ハードウェアプリフェッチ on/off 時の性能を予測するために最適と選択されたカウンタの組み合わせは BR_MSP (Conditional branch instructions mispredicted) と TLB_IM (Instruction translation lookaside buffer misses) であった. しかし, この際の寄与率は 0.3 程度と非常に低く, 統計的検定を通らないため, 性能を予測するには至らないと考えられる. この理由としては, 今回 PAPI を用いて取得できるカウンタの値からでは, プリフェッチの on/off 時の性能を予測することが難しいからだと考えられる. プリフェッチを用いたときに性能が向上するか否かは, メモリへのアクセスパターンが非常に重要な役割を果たすことが知られているが, 現在取得可能なカウンタの値から, これらが予想できるとは考えにくい. 今後の展望としては, PAPI とは異なるソフトウェアを用いることによってバスのアクセスについて種々のカウンタ値を取得することが可能であることが分かっており, それを利用することによって性能予測のモデルを構築することが第一に挙げられる.

5. 関連研究

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[6]. その他にも, 実行時の最適化という観点から, ランタイム最適化コー

表 2 最高周波数時に対する消費エネルギーの相対値

	性能比閾値	mcf	swim	matrix-50	matrix-600	matrix-1000
Pentium M	0.9	85.7	69.2	93.2	86.7	86.7
	0.8	77.5	56.5	82.5	78.2	78.2
	0.5	57.2	53.2	64.4	64.1	64.2
Opteron	0.9	88.3	85.3	94.0	88.3	85.0
	0.8	77.5	73.4	82.5	77.4	74.3
	0.5	62.2	58.9	60.6	62.0	58.8

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[8] や Intel IA32EL[4], Intel PIN[12] などの dynamic compilation 用のソフトも多く開発されている．

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[13]．

6. 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，2 つの異なるプラットフォーム上で評価を行い，その有効性を確かめた．加えて，ハードウェアプリフェッチによる性能の最適化のためのモデルを提案手法により構築し，検討を行った．この結果，現在取得可能なカウンタ値からは，十分なモデルを構築するには至らないと言うことが分かった．今後は，プリフェッチの on/off 時の性能を予測するのに適していると思われるバス周りのカウンタを取得することによってモデルを構築，適用評価を行う必要があると考えられる．

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，オーバーヘッドの少ないランタイムコードを提案することが挙げられる．また，他の定性的に解析しにくい最適化対象に対して実機のみならず，シミュレーションを用いることによっ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」，文部科学省科学研究費補助金 (若手研究 (B) 17700049) および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) 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., in *MICRO*, 2003.
5) Browne, S., Dongarra, J., Garner, N., London, K. S. and Mucci, P.: A Scalable Cross-Platform Infrastructure for Application Performance Tuning Using Hardware Counters., in *SC*, 2000.
6) 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., in *Conf. Computing Frontiers*, 2004.
7) 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., in *DATE*, 2004.
8) Ebcioglu, K. and Altman, E. R.: DAISY: Dynamic Compilation for 100% Architectural Compatibility., in *ISCA*, 1997.
9) Hennessy, J. L. and Patterson, D. A.: *Computer Architecture: A Quantitative Approach, 3rd Edition*, Morgan Kaufmann, 2002.
10) Hogg, R. V., Craig, A. and McKean, J. W.: *Introduction to Mathematical Statistics ,6th Edition*, Prentice Hall, 2004.
11) Krewell, K.: Pentium M Hits the Street, *Microprocessor Report*, Vol.17 (2003).
12) 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., in *PLDI*, 2005.
13) Moseley, T., Grunwald, D., Kihm, J.L. and Connors, D.A.: Methods for Modeling Resource Contention on Simultaneous Multithreading Processors., in *ICCD*, 2005.
14) 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., in *MICRO*, 2002.
15) 佐々木広, 浅井雅司, 池田佳路, 近藤正章, 中村宏：統計情報に基づく動的電源電圧制御手法, in *SACSIS*, 2006.