2007－OS－106 (2)
2007／8／3

統計情報に基づく省電力Linuxスケジューラ

金井 遵[†] 佐々木 広[††] 近藤正章[††]
中村 宏[††] 並木美太郎[†]

本稿では，統計情報に基づき性能予測を行い，設定された性能の範囲内でプロセス毎に省電力化を行うLinuxスケジューラの設計と評価について述べる．性能予測には，ハードウェアカウンタの値を説明変数とした回帰分析を用いて定量的にモデル化を行う方法を採用した．これにより，機械的に性能予測に用いるモデルを立式できるようにし，定性的な性能のモデル化が困難な計算機環境への展開を容易にした．さらには，性能予測のみでは性能の精度に限度があるため，実行時に予測性能と実性能の差を補正するフィードバック機構を設計・実装し，より高い性能精度を達成した．これにより，性能予測が外れるような場合にも，許容性能範囲内でさらに省電力化を達成すると共に，リアルタイム性を向上させた．本スケジューラは，プロセス単位での省電力化が可能であり，多様なプログラムが実行される環境においても最適な電力制御を行うことができる．加えて，基準性能の設定は実時間を基準として行うため，リアルタイムシステムにも応用可能である．実装には，近年サーバ環境から組込み機器まで多くの場合に利用されるLinuxシステムを用い，様々なアーキテクチャへの展開を容易な設計とした．評価より，性能予測のみで最大58%，性能予測とフィードバック機構を併用することで最大65%と大幅な電力量削減に成功した．

Energy-Efficient Scheduler by Statistical Analysis for Linux

Jun Kanai ,[†] Hiroshi Sasaki ,[††] Masaaki Kondo ,[††]
Hiroshi Nakamura [††] and Mitaro Namiki [†]

This paper describes the implementation and evaluation of an energy-efficient scheduler by statistical analysis for Linux. Estimating performance is based on statistical analysis by regression analysis and hardware counters as regression parameters. Applying our method for variable environments is easy. Additionally, we designed and implemented of "feedback system" which revise an error between estimated performance and real performance. Electricity consumption is optimized by governing electric power of each process. A performance threshold is established based on real-time, therefore our system is able to apply soft real-time system. We implemented this scheduler for Linux which is used for variable environments such as server, embedded systems and so on. Evaluations show that estimating performance reduces 58% of energy, and estimating performance with the feedback system reduces 65% of energy.

1. はじめに

近年，計算機の性能向上と複雑化に伴って，システムの消費電力は増加の一途をたどっている．一方で，計算機のユビキタス化にともない，携帯電話やPDAを初めとした情報端末，ノートパソコンなどのバッテリ駆動のデバイスの省電力化の要求が高まっている．また，バッテリ駆動のデバイスだけでなく，クラスタ環境や家庭内・企業などで利用する計算機，家電機器を初めとした組込み機器などにおいても，運用コスト，熱設計，環境問題などの面から省電力化が求められている．このように，省電力化は今後の情報システム開発にとって大きな課題である．

このような背景から，省電力化の研究や製品開発が盛んに行われている．これらを大別すると，ハードウェアレベルでの手法，ソフトウェアレベルの手法に分類することができる．ハードウェアによる手法では，プロセス技術の改善による省電力化から，バッテリ状況に応じて動作周波数や電源電圧を変更するIntel社のSpeedStep[1]，MPUの使用状況によって動作周波数や電源電圧を自動的に変更するTransmeta社のLongrun[2]やIntel社のEIST(Enhanced Intel Speedstep Technology)[1]まで，様々な手法が存在する．しかしながら，ハードウェアのみによる手法では，様々な動的情報を利用した複雑な制御を行うことや，OSなどのソフトウェアから得られる情報から制御を行うことが難しい．

一方，ソフトウェアによる省電力化の手法は，ハードウェアが提供する動的な電源電圧・周波数制御機構であるDVFS(Dynamic Voltage and Frequency Scaling)機構と連携した手法が中心である．汎用OSで広く利用されている単純な例としては，Linuxシステムに搭載されたcpufreq[3]モジュールと各種governorシステムを併せた例や，AMDによるWindows用のCool'n'Quietが存在する．例えば，ondemand governorやconservative governorおよびCool'n'Quietでは，CPU負荷に合わせて周波数制御を行う．これは，性能予測を行えるものではないほか，システム全体で周波数制御を行うためプロセス単位で挙動が異なる場合などには利用に適さ

† 東京農工大学
Tokyo University of Agriculture and Technology
†† 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

ない．アプリケーションによって性能要件が異なる場合も多く，性能を落とさずにできるだけ省電力化したい場合などにはこの手法では不十分である．特に，周波数変更時の性能予測が行えないことは，リアルタイムシステムのように守るべき性能が決まっているアプリケーションでの利用に向かない．一方で，一定時間で処理結果を返す必要のある各種サーバや，携帯電話やAV家電，ゲーム機器などの組込みシステムなど，リアルタイム性が必要な機器の重要性は増大し，それらの省電力化も求められている．近年，サーバから組込み機器までさまざまなアーキテクチャ向けにLinuxシステムを利用する例が増加しているが，OSレベルでは上記で述べたような単純な省電力化機構のみ利用されている．プロセス単位の性能予測による細かい電力制御を行うことで，性能を落とさずにさらなる省電力化の達成が期待できるほか，リアルタイム性を有するシステムとすることができる．

性能予測を行い，性能をできるだけ落とさずに省電力化を目指す研究として，多くの研究[6)10)]や，実際のシステムに適用した例[7)8)]が存在する．性能予測を行う方法として，定性的に対象システムの性能をモデル化する方法と，定量的にモデル化する方法が存在する．しかし，定性的にシステムを分析する方法では，モデル式算出の機械化が困難で，多くのプラットホームへの展開は難しい．加えて，近年の計算機システムの複雑化に伴い，定性的な性能のモデル化が困難になっているほか，定量的・定性的なモデル化の手法ともに，求められる性能予測の精度には限界がある．精度の高い性能予測と，最適な周波数選択による設定された性能の実現は，省電力化とリアルタイム性の実現において重要な要素である．また，性能をモデル化したものを実OSに適用した例においても，マルチプロセス動作を考えられていない場合もある．プロセス単位で性能要件・最適な電力制御が異なる場合もあり，これを考慮することができるOSレベルでの省電力化ではプロセス単位での電力制御が必須である．

以上より，既存のシステムにおける問題点をまとめると，OSレベルでの省電力化手法では，プロセス単位での省電力化が適しているにもかかわらず，広く利用されている手法ではシステム全体で一律に電圧周波数制御を行っているため電力の無駄が多いこと，また，性能予測を行うことができないためリアルタイムシステムでの利用に向かないことを挙げることができる．また，先行研究における問題点は，性能予測のための式のモデル化が定性的な手法では困難・不十分であること，性能のモデル化に基づく性能予測のみでは複雑化するシステムの性能予測精度に限度があること，マルチプロセス動作まで考慮されたモデルが少ないことなどを挙げることができる．

そこで，性能予測の方法として，性能予測のモデル化を容易・機械的に行うことのできる，定量的なモデル化手法に注目した．本研究の目標として，このモデルを用いてプロセス単位でDVFSを適用し，従来より細かい粒度で制御を行うことで，実際に広く利用されるLinuxシステムにソフトリアルタイム性を有する省電力化環境を提供する．これにより，設定性能に対して精度の高い性能を実現することで，リアルタイム性を確保しつつ，消費電力を極力抑えることを目標とする．さらに，複雑化する計算機システムでは性能予測に限界があるため，性能予測が外れるような場合にも，実行時に周波数選択を最適化し，さらなる省電力化とリアルタイム性を実現するシステムとすることを目標とする．

筆者らはまず，ハードウェアから得られる各種情報を元に，定量的に性能予測式をモデル化した．さらに，このモデルを元にプロセス単位でDVFS制御を行い，与えられた性能閾値を上回る性能を保ちつつ，省電力化を達成するLinuxスケジューラを設計・実装した．さらに，モデル化された性能予測のみでは限界がある場合にも，動的に実性能を周波数選択に反映し，実行時に動的最適化を行うフィードバック機構を設計・実装し，総合的に省電力化するスケジューラとした．これにより，さらなる省電力化，リアルタイム性の実現が期待できる．最後に，本スケジューラについて評価を行い，性能や省電力化の達成度を検証した．本稿ではこれらの詳細について述べる．

2. 統計情報に基づく性能予測

著者らの研究[5)]では，プログラムをコンパイラによりフェーズに分割し，ハードウェアから得られる情報を元に，フェーズ単位で精度の高い性能予測を行う方法を提案している．性能予測には，あらかじめ多数のプログラムを実行し，統計的な学習を行い，あるハードウェアカウンタの値と性能の間の相関関係を回帰分析によりモデル化する，定量的な手法を用いている．従来のような定性的な解析によりモデル化するわけではないため，この方法を用いることで，性能に支配的な要因が異なるような様々なプラットフォームで容易かつ機械的に性能予測式を立式することができる．複雑化する計算機システムにおいて，性能の定性的な解析は難しくなっており，本方式はそのような環境でも容易に精度の高い性能予測を行うことができるため，有用な方式である．また，説明変数であるハードウェアカウンタを変え，回帰分析を行い，モデルの当てはまりの良さ（決定係数）を調べることで，性能に支配的なハードウェアカウンタの要因について機械的に算出できることも本方式の利点である．

論文ではさらに，上記の方法で立式した性能予測式から，ターゲット周波数毎にフェーズ単位で性能予測を行い，プログラムを与えられた性能閾値を下回らない最低周波数で動作させることで，省電力化を行う方法を提案している．性能予測を行うためには，様々なプログラムを様々な周波数で動作させ，統計的な学習を行い，性能予測式をモデル化する必要がある．これにはPentiumMプロセッサにおける2個のパフォーマンスカウンタをサイクル数で正規化したものを説明変数，フェーズの実行時間の最高周波数時との相対比（つまり性能比）を目的変数として学習を行い，重回帰分析を行っている．重回帰分析を行った性能予測式は，パフォーマンスカウンタの値を計測した周波数（変更前の周波数）をv，性能予測を行う周波数（変更後の周波数）をu，最高周波数時との性能比をY_v^u，パフォーマンスカウンタの値をサイクル数で割ったものをX_{v1}，X_{v2}，係数をb_{v0}^u，b_{v1}^u，b_{v2}^uとして次式のように表すことができる．

$$Y_v^u = b_{v0}^u + b_{v1}^u X_{v1} + b_{v2}^u X_{v2} \quad (1)$$

PentiumMプロセッサでは，パフォーマンスカウンタを用いて多くのイベント発生回数を計測することができる．しかし，同時に計測できるイベントの数は2個までであるため，論文では，全てのパフォーマンスカウンタの組み合わせについて統計を取り，重回帰分析を行い，最も決定係数の高い，つまりモデルの当てはまりの良いパフォーマンスカウンタの組み合わせを性能予測に利用している．実験環境では，評価

より，Level 2 total cache misses と，Level 2 store misses のカウンタの組み合わせが最も性能に支配的であることがわかっている．

本方式は，汎用的かつ，高い精度で性能予測を達成しているが，性能に支配的な要素の異なる，つまり性能予測に用いるパフォーマンスカウンタの値が性能を説明しないプログラムが動作した場合には性能予測と実性能に差が生じるという欠点がある．また，ユーザモードプログラムとしての実装であるため，複数のプロセスが動作した場合，プロセス毎に最適な周波数を選択できない．さらには，本方式はフェーズ単位で性能予測を行っているが，並列動作するプロセスの組み合わせによって，リソース競合などによりコンパイラ挿入によるフェーズが必ずしも最適な統計取得・周波数制御の単位とは限らない．これらは，OS レベルでの実装により，プロセス単位での制御を行うことで解決することができる．

3. 統計情報に基づく Linux スケジューラ

本研究では，前章で述べた統計情報に基づく性能予測から省電力化を行う Linux スケジューラを設計・実装した．さらに，論文[5] では扱っていない，マルチプロセス時の動作をプロセス単位での電源電圧・周波数制御を行うことで改善した．性能予測に関して，従来のアルゴリズムと比べ，性能予測間隔の粒度を OS にとって最適なものにするとともに，性能予測式の数を減らすことで性能予測時，性能予測式立式時の計算量を改善した．さらに，実時間閾値という概念を導入することで，システム内のプロセス動作状況・負荷状況にかかわらず，実時間を基準として可能な限り一定の性能を指定・充足できるようにし，ソフトリアルタイム性を有するシステムの設計とした．加えて，性能予測が大きく外れるような場合も想定し，性能差を補正するフィードバック機構の導入を行い，性能閾値に対する実性能の精度向上を実現した．これにより，許容される性能範囲内でさらなる省電力化が望めるとともに，性能予測のみでは性能閾値を下回るような場合にも，性能補正を行うことでリアルタイム性を確保できる．本章ではこれらの設計・実装について述べる．

3.1 スケジューラの全体構成

本研究で設計した Linux スケジューラの全体構成を図 1 に示す．

本スケジューラはプロセス単位で最適な周波数を算出し，プロセス毎に周波数制御を行う．つまり動作周波数は，コンテキストスイッチ毎に変更される．スケジューラはパフォーマンスカウンタ管理機構，フィードバック機構，性能予測機構，実時間閾値変換機構，プロセス単位 DVFS 制御機構からなる．パフォーマンスカウンタ管理機構はプロセス単位の性能予測に用いるパフォーマンスカウンタの集計を行う．性能予測機構ではパフォーマンスカウンタの値と統計情報による性能予測式から各周波数時の性能予測を行い，ユーザにより設定された性能閾値から最適な周波数を選択する．また，単一の性能予測式のみでは，性能を決定する要素が大量に存在する近年の計算機において性能予測には限度があり実性能と予測性能に誤差が発生するため，フィードバック機構では，その予測性能と実性能の差を検出し，性能予測機構で選択された周波数に変更を加え，予測性能と実性能の誤差を補正する．実時間閾値変換機構では，実時間を基準として性能の閾値を設定できるようにし，システムの負荷状況にかかわらず，一定の実時間で処理を終了させるような周波数を選択するようにし，リアルタイム性を確保する．プロセス単位 DVFS 制御機構では，実際に選択された周波数から CPU を制御し，プロセス単位での周波数制御を行う．

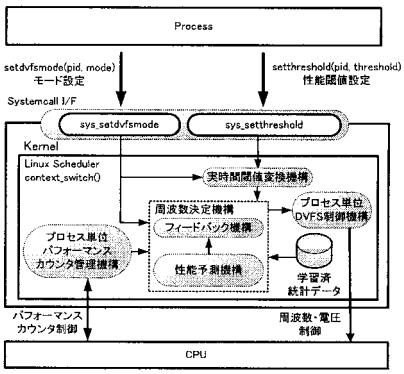

図 1　スケジューラの全体構成

ユーザプログラムからは，システムコールを利用することで各種の設定を行う．性能の閾値設定には setthreshold システムコール，オプション設定には setdvfsmode システムコールを追加した．これらのシステムコールはプロセス ID を指定してプロセス単位で設定を行えるほか，プロセス ID に-1 を設定するとデフォルトの性能閾値を設定でき，プロセスに性能閾値やオプションが設定されていない場合に利用することができる．オプションには主に表 1 に示した設定項目を指定できる．なお，デフォルト設定を行う場合や，自プロセス以外の特定プロセスの設定を行うには，保護のために，設定を行うプロセスが CAP_SYS_ADMIN ケーパビリティを持っている必要がある．

本スケジューラは，Linux スケジューラのコンテキストスイッチ部に各種処理を追加する．次節以降では各部の詳細について述べる．

3.2 性能予測機構

本スケジューラでは，3 章で述べた統計情報に基づく性能予測式から性能予測を行う．ただし，Linux スケジューラに性能予測機構を組込みにあたり，幾つか改善を加えている．

まず，既存アルゴリズムでは，コンパイラが挿入するフェーズ単位で性能予測・設定を行っていたが，本研究ではタイムスライスをフェーズと見なし，タイムスライス単位で性能予測と最適な周波数の算出を行う．これは，マルチプロセスで動作するような場合，プロセス同士の組み合わせによって，同一フェーズ内でも，リソース競合などにより，プログラムの振る舞いが大きく変化する可能性があるためである．キャッシュヒット率を例にとれば，キャッシュヒット率が高いプログラム A が単体で実行されていたとき，後からワーキングセットが大きく，かつキャッシュヒット率が低いようなプログラム B が実行された場合には，リソース競合によりプログラム

表 1　設定可能なオプション（抜粋）

パラメータ	内容
ENABLE_REALTIME_THD	実時間性能閾値設定を有効にする
DISABLE_REALTIME_THD	実時間性能閾値設定を無効にする
ENABLE_FEEDBACK	フィードバック機構を有効にする
DISABLE_FEEDBACK	フィードバック機構を無効にする

B のキャッシュヒット率が大幅に低下するような場合が存在するためである．このような場合にも，タイムスライス単位で性能予測を行い直すことで対処が可能であるため，このような設計とした．

さらに，既存アルゴリズムでは，説明変数の正規化方法としてサイクル数を用いていた．しかし，この方法では回帰式は，変更前の周波数別・変更後の周波数別に式が必要となるため，周波数を n 段階に変更可能なシステムでは，$n(n-1)$ 通りの回帰式を求める必要がある．そこで，筆者らは説明変数の正規化の方法として，パフォーマンスカウンタから得られるユーザモード実行命令数を用いることとした．ユーザモード実行命令数あたりのパフォーマンスカウンタの変動値（例えば L2 キャッシュミス回数）を用いることでプロセッサの周波数に依らず，同一のコードを実行すれば説明変数の値は同じになる．これにより，求める回帰式の数は変更後の周波数別だけで良いため n 通りで良い．また，これにより PentiumM ではパフォーマンスカウンタの個数は 2 個であるため，説明変数に利用できるのは 1 変数のみになる．しかし，論文[5]によれば，本システムでは L2 Total Cache Misses のみでも精度の高い性能予測が可能である．ターゲット周波数 u 時における最高周波数時に対する性能比 Y_u の予測は，あらかじめ回帰分析により求めた係数 b_{u0}，b_{u1} と説明変数となるパフォーマンスカウンタの値 X_1 と，ユーザモード実行命令数 $X_{instructions}$ より次式で行う．

$$Y_u = b_{u0} + b_{u1}(X_1/X_{instructions}) \quad (2)$$

さらに，性能比 Y_u とシステムコールにより設定された性能閾値 Thd を比較し，

$$Y_u \geqq Thd \quad (3)$$

となる最低周波数の u を動作周波数とする．

3.3 実時間閾値設定機構

筆者らの先行研究[5]では，マルチプロセスでの動作が考えられておらず，同じ性能閾値を設定したとしても，プロセスに割り当てられる時間によって，計算終了までに要する実時間は変化する．しかしながら，特にリアルタイム性が要求されるようなプログラムにおいては，動作しているプロセス数や負荷にかかわらず，一定時間で処理が完了することが必要である．また，人間にとってもプロセス時間ではなく，実時間を基準として性能閾値を設定できた方が理解しやすいため，このような仕組みを提供する．

ここで，ユーザが与えた性能閾値を Thd とする．$Thd = 1$ となるのは，最高周波数で動作し，さらに全ての時間が該当プロセスの実行に割り当てられた場合の性能とする．このとき，プロセスが実際に満たすべき閾値 $RealThd$ は，総時間 T_{all} と T_{all} 中に該当プロセスに割り当てられた時間 $T_{Process}$ から，

$$RealThd = Thd/(T_{Process}/T_{all}) \quad (4)$$

で求めることができる．さらに，性能閾値 $RealThd$ を満たすように性能予測を行い，最適な周波数を選択する．これにより，該当プロセスに割り当てられる時間にかかわらず，つまり，システムの負荷状況にかかわらずに実時間を基準とした閾値を設定することができるようになる．ただし，全時間中に占めるプロセス時間は過去のデータより算出される値であるため，未来についてもこの値が保証されるわけではない．また，性能予測についても必ずその性能でプログラムが動作することを保証しない．つまり，本システムはリアルタイム性を可能な限り満たすソフトリアルタイムとなる．

Linux システムにおいては，予備実験より $T_{Process}$ をタイムスライス (3ms〜20ms) とすると，$T_{Process}/T_{all}$ の変動が比較的大きくなってしまうため，$RealThd$ が変動し，実性能と性能閾値との誤差が大きくなる．そこで，精度を高めるため，コンテキストスイッチが発生する毎に計算するのではなく，複数回に一度，$T_{Process} \geqq 100[ms]$ となった場合に再計算するようにしている．

3.4 フィードバック機構

本研究では性能予測式として，L2 キャッシュヒット率などの特定のハードウェアカウンタの値を用いる．しかしながら，性能に大きく影響する要素はプログラムによって一定とは限らず，性能予測に用いない要素が性能を大きく左右する場合，性能予測と実性能とに大きな差異が発生することもあり得る．実性能が性能閾値を上回る場合には，さらに周波数を下げることで，許容される性能の範囲内でさらに省電力化を行うことができる可能性がある．一方で，実性能が性能閾値を下回ることは，リアルタイム性が要求されるアプリケーションにおいては好ましくない．そこで，本スケジューラに実性能と性能閾値に差が発生する場合に補正を行うフィードバック機構を設計・実装した．本機能は，プログラムの振る舞いの変化を評価式により検出し，その都度，基準となる性能を計測し，その性能に対して実性能の達成度を評価する．その結果を周波数選択に反映させることで，性能の閾値に実性能をより近づける仕組みとなっている．以下ではこれらの詳細について述べる．

ここで，計算機の性能を示す指標として，IPS(Instructions Per Second) が広く用いられている．筆者らは，本スケジューラが利用するプログラムの性能の指標として，単位時間あたりのユーザモード実行命令数 UIPS(Usermode Instructions Per Second) という値を導入した．つまり，最高周波数時の UIPS を $UIPS_{max}$ とした場合，プロセスが満たすべき UIPS，$UIPS_{ideal}$ は，

$$UIPS_{ideal} = UIPS_{max} \times RealThd \quad (5)$$

となる．よって，実性能と性能閾値との命令数差 err は，動作周波数下で時間 T_{ts} での実行命令数を $Nins_f$ として，

$$err = UIPS_{ideal} \times T_{ts} - Nins_f \quad (6)$$

となる．この err の総和を 0 に近づけるように，周波数を調整することで実性能と性能閾値との差をなくすことができる．一方で，本機能を利用するためには，プログラムを最高周波数で動作させ，基準となる $UIPS_{max}$ を実測する必要がある．

本スケジューラでは，図 2 のように，最高周波数時の性能を実測する「性能計測モード」，性能予測式から求めた最適な周波数で動作する「予測モード」，フィードバック機構により性能を調整する「フィードバックモード」があり，条件に従ってタイムスライス毎に状態遷移する．

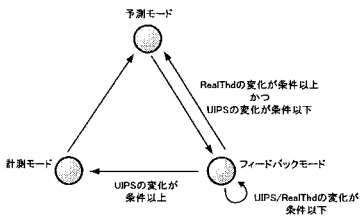

図 2 各モードの状態遷移

性能計測モードでは，プログラムを最高周波数で動作させ$UIPS_{max}$の測定を行い，基準となる性能とする．測定後，次のタイムスライスでは予測モードに遷移する．

予測モードでは，3.2節で述べた性能予測機構により性能予測を行い，得られた周波数で動作する．測定後，次のタイムスライスではフィードバックモードに遷移する．

フィードバックモードでは，実性能と性能閾値から求めた理想的な性能との命令数差を計算し，なるべく差が0に近づくように周波数設定を行う．具体的には，errの総和S_{err}が$S_{err} < 0$の場合には周波数を一段階上げ，$S_{err} \geqq ThdH_{ins}$の場合には，周波数を一段階下げる．ここで，$RealThd$が変化した場合には最適な周波数が変わる．フィードバックモードのみでも周波数補正は可能であるが，最適な周波数に収束するまでに時間がかかるため，$RealThd$の差が$ThdH_{RealThd}$以上になった場合には予測モードに遷移し，周波数の再予測を行うこととした．本実装では，$ThdH_{RealThd}$として，PentiumMの周波数が7%刻みで変更可能であるため，余裕を持たせ，5%を選択している．

また，$UIPS_{max}$はプログラムの振る舞いによって常時変化する．誤った$UIPS_{max}$を性能の基準として利用し続けると，性能閾値と実性能に誤差が生じることになる．そこで，プログラムの振る舞いの変化を検出し，性能計測モードに遷移し，$UIPS_{max}$を再計測する必要がある．一方で，頻繁に性能計測モードに遷移すると，性能計測モードは最高周波数で動作するため，与えられた性能閾値よりも性能が大幅に上回り，省電力化において不利になると考えられる．プログラムの振る舞いの変化の検出は必要最低限にする必要がある．本スケジューラでは，以下の2種の検出方法を用いて，プログラムの振る舞いの変化を検出することとした．

- 予測性能の変化
  現在と，$UIPS_{max}$を計測した時の，性能予測に用いるパフォーマンスカウンタの値が大きく異なる場合，$UIPS_{max}$が異なる可能性が高い．そこで，性能計測モード時および，現在のパフォーマンスカウンタ値をユーザランド実行命令数で正規化したものをそれぞれ，次回のターゲットとなる周波数の性能予測式に入れ，差が$ThdH_{estimate}$以上になった場合には，プログラムの振る舞いが変化したと見なす．$ThdH_{estimate}$は許容される最大の性能誤差を示すが，ここでは5%としている．
- 実性能の変化
  予測式に利用しない要素が性能に支配的な要因の場合，予測性能の変化が実性能の変化を検出できない．そこで，実性能の変化に関してもプログラムの振る舞いの変化の検出対象とした．しかし，動作周波数は逐次変更されるため，UIPSからプログラムの振る舞いの変化を全て検出するのは困難である．そこで，プログラムが同一内容であれば，次式を常に満たすことに注目した．ここで，$maxfreq$は最高周波数，$targetfreq$はターゲット周波数，$UIPS_{max}$は最高周波数時のUIPS，$UIPS_{target}$はターゲット周波数時のUIPS，$Base_UIPS_{target}$はプログラムの振る舞い変化後にターゲット周波数で初めて動作したときのUIPSとする．これらはあらかじめ計測しておく．

$$\frac{targetfreq}{maxfreq} \leqq \frac{UIPS_{target}}{UIPS_{max}} \leqq 1 \qquad (7)$$

$$\frac{Base\_UIPS_{target} - UIPS_{target}}{Base\_UIPS_{target}} = 0 \qquad (8)$$

プログラムが同一内容であれば，(7)式はターゲット周波数で動作時の最高周波数時との性能比は，周波数比以上1以下になること，(8)式は同一周波数であれば，IPSは同等になることを示している．許容誤差Thd_{uips}を考慮してもなお，(7)式，(8)式を満たさない場合には，プログラムの振る舞いが変化したと見なす．Thd_{uips}は性能の許容誤差であり，同じく5%としている．

以上のいずれかの条件を満たす場合，プログラムの振る舞いが変化したとみなし，性能計測モードに遷移する．命令によって実行時間の長さは変わるため，実行命令数ベースの差の総和であるS_{err}は，その際にクリアする．

3.5 実　　装

以上で述べた設計に基づき，Linux Kernel 2.6.18をベースとし，スケジューラに機能追加を行った．ターゲットとなるアーキテクチャはx86とし，特に電源電圧・周波数制御はCPUやチップセット依存となるため，Dothanコアの PentiumM(FSB 533MHz) をターゲットとした．既存のカーネルソースに対するコード変更量は，追加システムコールに関する宣言部分を除くとスケジューラに関する記述がある/kernel/sched.cに加えた2行のみであり，そのほかの追加部分は新規ファイルにまとめることで，頻繁にバージョンアップされる最新Linuxカーネルへの追随も容易である．本スケジューラに関するソースコード総行数と，本スケジューラを利用した場合のオーバヘッドを計測した結果について表2に示す．本カーネルでは，コンテキストスイッチ毎に統計処理・性能予測・周波数切り替えを行うが，表2に示したように，総プログラム実行時間に占めるコンテキストスイッチ時のオーバヘッドは0.01%程度と十分に小さい．

また，Linuxカーネルではアーキテクチャ依存部は/archディレクトリ以下に格納され，他アーキテクチャへの移植が容易な構造となっている．本スケジューラにおいても，パフォーマンスカウンタ制御や電源電圧・周波数制御がアーキテクチャ依存になるため，これらのコードは/arch以下に分離し，x86以外のアーキテクチャへの移植が容易な構造とした．

なお，今回はPentiumMを実装のターゲットにしているが，近年多くのCPUにおいてパフォーマンスカウンタを設けており，これらを利用することで他のアーキテクチャでも汎用的に利用できる方法である．また，CPUのパフォーマンスカウンタだけでなく，I/Oに関する情報や，OSから得られるsleep_avgなどの情報などを用いて，複数の説明変数から重回帰分析を行うことでさらに精度の高い性能予測が可能になると予想できる．

なお，本スケジューラではあらかじめ複数のプログラムについて実行し，パフォーマンスカウンタの値と性能の統計を取り，性能とパフォーマンスカウンタの間の相関関係を求めておく必要がある．そこで，筆者らはあらかじめ行列乗算 (行列サイズ1500 × 1500) とMiBench(FFT/SHA/Dijkstra/QuickSort)について統計を取り，各周波数で動作した場合の回帰式を求めている．利用するパフォーマンスカウンタの要素は先行研究[5]

表2　スケジューラの基本情報

ソースコード総行数	951[行]
コンテキストスイッチ所要サイクル（全て）	3720[cycle]
コンテキストスイッチ所要サイクル（追加部分のみ）	3082[cycle] (83[%])
総時間中コンテキストスイッチ時間（オーバヘッド）	0.01[%]

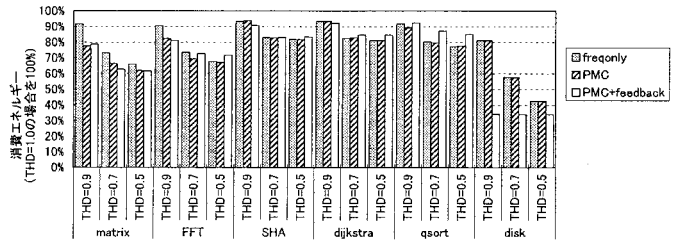

図 3　各モードの閾値別消費エネルギー比

にて最も決定係数が大きい結果となった，L2 Cache Total Misses を利用している．

なお比較評価のため本システムでは，以上で述べたパフォーマンスカウンタの値から性能予測を行い，フィードバック機構を併用するモード（PMC+feedback(fb) モード），統計情報を用いずに周波数がそのまま性能比になると仮定し，性能を予測するモード (freqonly モード)，統計情報とパフォーマンスカウンタの値から性能を予測し，フィードバック機構は無効となるモード (PMC モード) を選択し，利用することができる．freqonly モードと，PMC モードでは，3.4 項で述べた性能計測モードとフィードバックモードには遷移しない．

4. 評　　価

本スケジューラについて種々の評価を行った．本章ではこれらの詳細について述べる．評価には，PentiumM 760 を搭載した PC を利用した．PentiumM 760 で設定可能な周波数・電圧および，それらの周波数条件下での行列乗算プログラム実行時の消費電力を非接触型電力測定装置で実測した結果を表 3 に示す．以降の評価では，freqonly モード，PMC モード，PMC+feedback(fb) モードのそれぞれで評価を行う．

4.1 消費電力量に関する評価

本スケジューラについて，性能閾値を変化させ，行列乗算と MiBench(FFT，SHA，dijkstra，quicksort)，ディスク読み込みのベンチマークを実行した場合の CPU 消費エネルギーの計測を行った．消費エネルギーの測定結果を図 3 に，電力性能の指標である性能閾値 1.0 の場合を基準とした性能閾値 0.9 の場合のエネルギー遅延積（EDP）の比を表 4 に示す．

結果より，全てのモードにおいて消費エネルギー削減が実現できている．特に，性能閾値 0.5 の場合，PMC+feedback モードで最大 65%程度の消費エネルギー削減が達成できており，本方式の有用性を示す結果である．また，ディスク読込みや matrix，FFT，SHA，dijkstra の一部では，性能予測と実性能に差が発生するため，フィードバックモードが特に有効に働き，PMC のみの予測と比べ，最大 45%と大幅な消費エネルギーの削減に成功している．年々複雑化するシステム環境下や，さまざまな律速要素のプロセスが動作する環境下では，単一の予測式での予測精度は説明変数を増やした場合でも限界があるため，フィードバック機構が有用と考えられる．

表 3　PentiumM 760 の基本情報

動作クロック [GHz]	プロセッサ電源電圧 [V]	消費電力 [W]
2.00	1.356	23.96
1.86	1.308	19.96
1.73	1.260	17.32
1.60	1.228	15.52
1.46	1.196	13.96
1.33	1.164	12.64
1.20	1.132	11.44
1.06	1.084	9.996
0.80	0.988	7.716

表 4　THD=0.9 の場合のエネルギー遅延積比 [%]（数値は大きいほど良）

	matrix	FFT	SHA	dijkstra	qsort	disk
freqonly	104	106	100	100	102	124
PMC	113	113	99	100	101	124
PMC+feedback	114	110	99	98	96	293

SHA や matrix などプログラム内でのキャッシュヒット率が大きく変わらないベンチマークでは，フィードバック機構の有無で消費エネルギーの差は小さい．一方で，律速要素の値が頻繁に変化するプログラムは苦手である．FFT や qsort などプログラム中でのキャッシュヒット率が頻繁に変化するベンチマークでは PMC モードと比べてフィードバックを有効にすると，消費電力に最大約 7%程度の差が現れている．これは，プログラムの振る舞いが変わったことを検出した場合に，基準となる最高周波数時の性能を計測するために，最高周波数で動作することが原因である．これを改善するためには，プログラムの振る舞いの変化検出回数・箇所を最適化するために，コンパイラによるフェーズ挿入が有効かと思われる．しかしながら，最大 7%となる消費電力の増大はディスク読込みプログラムでフィードバック機構を有効にした場合に改善する 45%に比べて小さい．

電力性能の指標である EDP を見れば，ディスク読込みや matrix，FFT などにおいては，特に性能を落とさず省電力化できていることがわかる．特にディスク読込みでは，フィードバック機構により，大幅に EDP が改善している．EDP ベースで見た場合でも，性能予測が外れる場合にフィードバック機構が特に有効に働く．

以上より，基本的にはフィードバック機構は有用である．しかし，たとえば純粋に CPU バウンドになるプロセスのみを動かすような予測が外れない場合で，さらに特に消費電力を重視する場合にはフィードバック機構を無効にするといった使い分けも有効であると考えられる．しかしながら，今回のベンチマークプログラムは基礎データを取るため，データ入出力にファイル入出力を伴わないようにしている．実際のプログラムにおいては，CPU バウンドと言われるアプリケーションにおいてもファイル入出力を伴うなど，同一プログラム中で I/O バウンドとなる部分が存在するような場合が多いと考えられる．そのため，一般に CPU バウンドとなると言われるプログラムが多く走る環境においても，一概にフィードバッド機構が無用であるというわけではない．

4.2 スケジューラの閾値に対する精度の評価

設定された性能閾値に対して精度の高い実性能を達成することで，省電力化およびリアルタイム性の向上に繋がる．そ

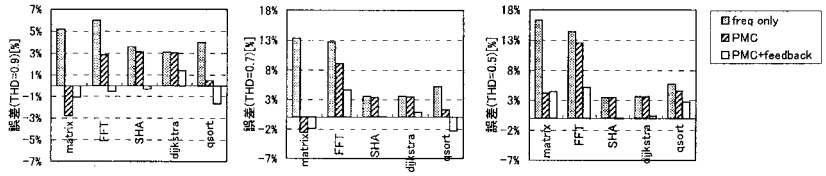

図 4　各モードの性能誤差

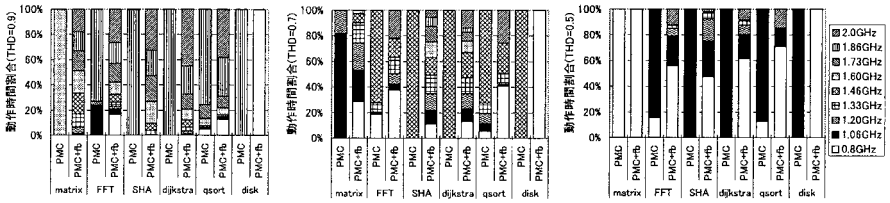

図 5　各モードの動作周波数内訳

こで，本スケジューラの精度の評価を行うため，行列乗算と MiBench(FFT，SHA，dijkstra，QuickSort)，およびディスク読み込みプログラムのベンチマークを用いて，性能閾値を与え，実性能との差を調べる評価を行った．

性能閾値を 0.9，0.7，0.5 に設定した場合の実性能との誤差を計測した結果を図 4 に（周波数が性能にほとんど影響しない結果となるディスク読込みプログラムは除く），性能閾値を 0.9，0.7，0.5 に設定した場合の動作周波数の内訳を図 5 に示す．また，それぞれのモードの正確性を比較するために，それぞれの性能閾値で誤差の指標となる二乗平均平方根誤差を計算したもの（ディスク読込みプログラムは除く）を図 6 に示す．

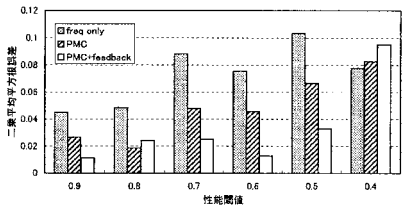

図 6　各モードの閾値別性能誤差

結果より，多くの場合で PMC+feedback モードが最も正確，次に PMC モードが正確であることがわかり，本システムの有用性を示す結果となっている．特に，ディスク I/O がボトルネックになり，性能予測が大幅に外れるディスク読込みプログラムでは，閾値 0.9 の動作周波数の内訳において，PMC モード，freqonly モードに比べて大幅に低い周波数で動作しており，省電力化できている．また，PMC+feedback モードでは，最低周波数で動作し，それ以上性能を下げられない場合を除き，誤差が設計での述べた許容誤差 5%以内に収まっていることが確認できる．

しかしながら，quicksort では，多くの場合，フィードバック機構を利用しない方が正確な結果となっている．これは，quicksort が頻繁にキャッシュヒット率が変わるベンチマークであり，フィードバックで基準とする最高周波数時の実性能が頻繁に変化するため，その変化を追跡しきれていないためである．許容誤差の修正で最高周波数時の性能の正確な追跡は可能だが，その際に実際に最高周波数で動作させ実性能を計測しなければならないため，性能が性能閾値に比べ高くなってしまい，結局誤差が発生してしまう．また，性能閾値 0.4 で PMC+feedback モードの誤差が大きいのは，プログラムの振る舞い（キャッシュヒット率や IPS の値）が大きく変わる場合に，基準性能の計測のため，最高周波数で動作した結果，最終的に性能が高くなってしまうためである．これを改善するためには，プログラムの振る舞いの変化検出回数・箇所を最適化するために，先行研究[5] で述べられているコンパイラによるフェーズ挿入が有効と考えられる．

このように，多くのアプリケーションにおいて統計情報に基づく性能予測とフィードバック機構の組み合わせは効果を発揮するが，キャッシュヒット率などの性能を支配する要因の値が頻繁に変化するようなアプリケーションに限っては苦手な傾向にある．

4.3　マルチプロセス時の精度評価

リアルタイム性を要求するアプリケーションにおいては，システムの負荷状況にかかわらず，指定した時間までに処理を終えることが重要である．そこで，複数のアプリケーションを同時に動作させ，性能閾値に対する実性能精度に関する評価を行った．行列乗算プロセスを PMC+feedback モードで 1〜5 プロセス動作させ，性能閾値を 0.2 倍とした場合の計算終了までの実時間を計測した結果を図 7 に，そのときの動作周波数の内訳を図 8 に示す．ここで図 7 の結果は，最高周波数で 1 プロセスを動作させた場合に要する計算終了時間の 5 倍を基準（理想値）とし，相対値で示す．

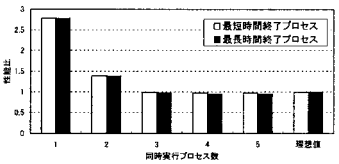

図 7　マルチプロセス動作時の性能（実時間性能指定の正確性評価）

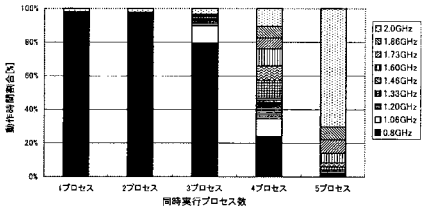

図 8　マルチプロセス動作時の動作周波数時間内訳（matrix1500）

結果より，性能閾値を満たせるプロセス数，つまりプロセス数 3 以上での最大誤差は 4%となった．この結果は，設計時に設定した最大許容誤差である 5%の範囲内におさまっており，マルチプロセス動作時にも十分な精度で実行できていることがわかる．また，実時間は体感速度にも大きく影響すると考えられ，本システムが負荷状況にかかわらず，高い精度かつ，プロセッサ能力に余裕がある場合には低い周波数で実行できていることは，本システムの有用性を示している．

5. 関連研究

Linux システムに搭載される従来の省電力化システム cpufreq と各種 governor は，システムの負荷状況から電圧・周波数制御を行う．しかし，本システムと違いプロセス毎の細かい性能要求に対して対処することが出来ない．また，性能予測を行うものではないため，リアルタイムシステムの利用には向かない．本システムは，性能予測とフィードバック機構を組み合わせ，精度の高い性能実現が可能である．

また，性能をできるだけ落とさずに省電力化を目指す研究としては，CPU のアイドル時間を用いて，周波数変更時の性能低下をモデル化した研究[6] や，その情報を実 Linux システムに適用した研究[7]，リアルタイムシステムに応用した研究[8] などが活発である．しかしながら，近年の CPU は高度化しており，パイプライン実行やアウトオブオーダ実行などにより，CPU の正確なアイドル時間の算出は困難になっており，これらの手法による正確性には疑問が残る．例えば，論文[7] では，CPU アイドル時間の目安として Linux のプロセス毎の待ち時間の指標である sleep_avg を参照しているが，あくまでも sleep_avg に現れるのはプロセスが待ち状態の時間であり，メモリボトルネックになるようなアプリケーションが走ることが多い環境など，全てのシステムで利用できるわけではない．

一方，近年の CPU の高度化に伴うアイドル時間算出の困難化により，アイドル時間以外から性能を予測する研究も存在する．例えば，論文[10] ではメモリのアクセス時間から性能低下を予測し，メモリバウンドなアプリケーションにおいても性能低下を抑えつつ，電力消費の削減を達成している．一方で，モデル化は定性的な解析に基づいており，他システムやメモリ以外がボトルネックになる環境においては，モデルの再構築が必要になるという欠点がある．

これらに対し本システムは，ハードウェアカウンタの値と回帰分析を用いて定量的に性能のモデル化を行うため，複雑なシステムにおいてもモデル化は容易かつ機械的に行うことができる．また，メモリアクセスだけでなく，様々な律速要素に対して対処が可能である．加えて，性能予測の限界に対してもフィードバック機構により対処することができるため，より実用的なシステムとなっている．

6. おわりに

本稿では，Linux 向けに統計情報に基づき性能予測を行い，フィードバック機構を有する省電力化スケジューラの設計・実装について述べ，評価を行った．本スケジューラは性能予測のモデル化が容易であり，フィードバック機構による実行時性能の最適化も可能であるため，多くのアーキテクチャへの実用的な省電力化プラットフォームの提供が可能である．また，システムの負荷状況にかかわらず，省電力化を行いながらも，性能実現の精度は高く，リアルタイムシステムへの応用も可能である．さらに評価より，性能予測のみで最大 58%，フィードバック機構を有効にすることで最大 65%の消費エネルギーの削減を実現し，本方式の有用性を実証した．

今後の予定として，現在手作業にて行っている性能予測式の立式の自動化，フィードバック機構のアルゴリズムの最適化により，一部，消費電力が増大している部分の改善，多くのアーキテクチャへの展開を挙げることができる．

参考文献

1) Intel: Enhanced Intel SpeedStep Technology for the Intel Pentium M Processor, http://www.intel.com/design/intarch/papers/30117401.pdf
2) Transmeta: LongRun2 Technologies, http://www.transmeta.com/tech/longrun2.html
3) Dominik Brodowski: Linux kernel CPUfreq subsystem, http://www.kernel.org/pub/linux/utils/kernel/cpufreq/cpufreq.html
4) Intel: Intel Pentium M Processor with 2-MB L2 Cache and 533-MHz Front Side Bus Datasheet, http://download.intel.com/design/mobile/datashts/30526202.pdf
5) 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, 情報処理学会論文誌, Vol.47, No.SIG18(ACS16), pp.80–91(2006).
6) M. Weiser, B. Welch, A. Demers, and S. Shenker: Scheduling for reduced CPU energy, Proc. of Symposium on Operating Systems Design and Implementation, (1994).
7) 宮川大輔, 石川 裕: 電力制御スケジューラのプロトタイプ実装, 情報処理学会研究報告,SigOS, Vol.103, pp.109–115(2006).
8) W.Yuan, K.Nahrstedt: Energy-efficient soft real-time CPU scheduling for mobile multimedia systems, Proc. of the 19th ACM Symposium on Operating Systems Principles(SOSP'03), pp.149–163(2003).
9) F.Gruian: Hard real-time scheduling for low-energy using stochastic data and DVS processors, Proc. of the 2001 international symposium on Low power electronics and design, pp.46–51(2001).
10) Q.Wu, M.Martonosi, D.W.Clark, V.J.Reddi, D.Connors, Y.Wu, J.Lee, D.Brooks: A Dynamic Compilation Framework for Controlling Microprocessor Energy and Performance, Proc. of the 38th annual IEEE/ACM International Symposium on Microarchitecture, pp.271–282(2005).