性能予測モデルの学習と実行時性能最適化機構を有する省電力化スケジューラ

金 井 遵[†1]　佐 々 木 広[†2]　近 藤 正 章[†2]
中 村 宏[†2]　天 野 英 晴[†3]
宇佐美 公良[†4]　並木 美太郎[†1]

本稿では，統計情報に基づき性能予測を行うモデルを自動的に最適化し，このモデルを用いて，設定された性能の範囲内でプロセスごとに省電力化を行う 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性能予測式の最適化を行う方法を採用した．これにより，プログラムを実行するのみで計算機環境ごとに性能予測式を最適化する機構を設計・実装した．さらに，省電力化時には，最適化した性能予測に基づき最適な周波数でプログラムを実行するとともに，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，より高い性能精度の達成とさらなる省電力化を達成した．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，時間制約のあるシステムにも応用可能である．実装には広く利用される Linux システムを用い，評価より，単純な周波数比による性能予測に比べて，学習による性能予測のみで CPU ベンチマークでは最大 17%，性能予測とフィードバック機構を併用することで I/O ベンチマークでは最大 47%，CPU ベンチマークでは最大 23%の電力量を削減できた．

Energy-efficient Scheduler with Adaptive Performance Prediction and Performance Optimization

Jun Kanai,[†1] Hiroshi Sasaki,[†2] Masaaki Kondo,[†2]
Hiroshi Nakamura,[†2] Hideharu Amano,[†3] Kimiyoshi Usami[†4]
and Mitaro Namiki[†1]

This paper describes the implementation and evaluation of an energy-efficient Linux scheduler using adaptive statistics of performance. The scheduler examines relationships between the performance and hardware counters using regression analysis. The relationships are determined via executions of benchmark programs for each computer environment. Additionally, the scheduler executes each process with suitable voltage and frequency by adaptive performance prediction. Furthermore, we designed and implemented of “feedback system” that revise an error between the estimated performance and the real performance to achieve more energy efficiency. The performance threshold is established using real-time, therefore the scheduler is able to apply for soft real-time system. Evaluations show that the performance prediction reduces 17% of energy in a CPU benchmark, and the performance prediction with the feedback system reduces 47% of energy in an I/O benchmark, and 23% of energy in a CPU benchmark.

†1 東京農工大学
Tokyo University of Agriculture and Technology
†2 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
†3 慶應義塾大学大学院理工学研究科
Graduate School of Science and Technology, Keio University
†4 芝浦工業大学工学部
Department of Information Science and Engineering, Shibaura Institute of Technology

1. はじめに

近年，計算機の性能向上と複雑化にともなって，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 をはじめとした情報端末，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，

家電機器をはじめとした組み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さかんに行われている．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，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 システムをあわせた例や，AMD による Windows 用の Cool'n'Quiet が存在する．たとえば，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．これらは，周波数変更時の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，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み込みシステムなど，処理に時間制約がある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，サーバから組み込み機器まで様々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．

これより，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ともない，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，省電力化と時間制約の達成において重要な要素である．さらに，マルチプロセス動作を考えた場合，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．

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かつ，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，ユーザの計算機の利用方法や計算機の構成に最適な性能予測のモデルを自動的に最適化することを目標とする．さらに，この自動的に最適化した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 システムに性能要件を達成しつつ省電力化を行う環境を提供する．さらに，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性能要件達成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，プログラムを実行した際に，ハードウェアから得られる各種情報をもとに，定量的に性能予測式を自動的に最適化する Linux スケジューラを設計・実装した．さらに，このモデルをもとにプロセス単位で 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機構を本スケジューラに実装した．加えて，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させ，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．最後に，本スケジューラについて評価を行い，性能精度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

筆者らの研究[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もとに，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求める，定量的な手法を用いている．定量的な手法を用いることで，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，モデルのあてはまりの良さ（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．

本稿ではさらに，上記の方法で立式した性能予測式から，ターゲット周波数ごと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，性能予測式を最適化する必要がある．これにはハードウェアから得られる各種情報のカウンタ（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_{vq}（$q = 1 \sim p$），回帰係数を b_{vq}^u（$q = 0 \sim p$）として次式のように表すことができる．

$$Y_v^u = b_{v0}^u + \sum_{q=1}^{p} b_{vq}^u X_{vq} \qquad (1)$$

実装・評価において利用している 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，本稿では，すべてのパフォーマンスカウンタの組合せについて統計をとり，2 変数による重回帰分析を行い，最も決定係数の高い，つまりモデルのあてはまりの良いパフォーマンスカウンタの組合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 のカウンタの組合せが最も性能に支配的であることが分かっている．

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，プロセスごと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合せによって，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．また，本方式はシステムごと・ユーザごとに最適な性能予測式を構築することが比較的容易であるが，OS に組み込むにあたり，カーネルの改変はユーザにとって困難であるので，性能予測式を自動的に学習，適用できることが望ましい．

3. 統計情報に基づく Linux スケジューラ

本章では本研究で設計・実装を行った，省電力化 Linux スケジューラの概要と全体構成について述べる．

3.1 概　要

本研究では，計算機環境やユーザの計算機利用方法に応じて，統計情報に基づく回帰分析により性能予測式を自動的に最適化し，この性能予測モデル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

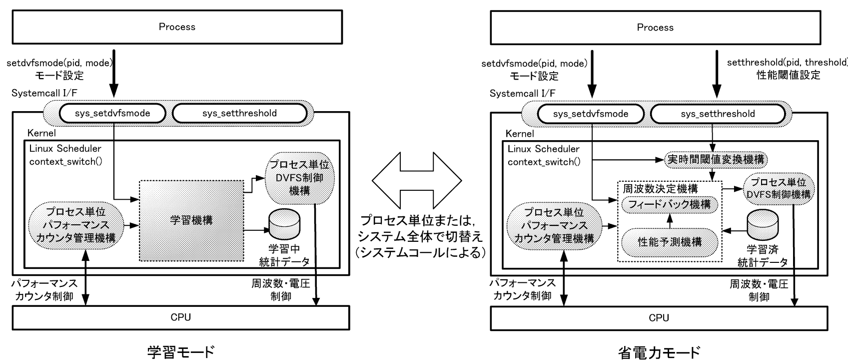

図 1　スケジューラの全体構成
Fig. 1　The architecture of DVFS scheduler.

とで性能要件を達成できる．本章ではこれらの設計・実装について述べる．

3.2　スケジューラの全体構成

本スケジューラは，性能予測式から省電力化を行う省電力モードと，性能予測式を自動的に最適化する学習モードからなる．これらのモードは，システムコールにより，プロセス単位または，システム全体で切り替えることができる．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.

省電力モードでは学習モードで最適化した性能予測式を用いて，ユーザが与えた性能制約の中で実際に省電力化を行う．省電力モードではプロセス単位で最適な周波数を算出し，プロセスごと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ごとに変更される．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と，プロセス単位 DVFS 制御機構は学習モードと共通である．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予測精度には限界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，その性能閾値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，性能閾値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，時間制約を達成する．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プロセス単位で CPU の周波数制御を行う．

一方，学習モードでは，省電力モードで用いる性能予測式の学習を行う．スケジューラはプロセス単位でパフォーマンスカウンタと性能の統計をとり，プロセス切替えごとに回帰分析を行い，性能予測のための回帰方程式の回帰係数を算出し，予測式を最適化する．スケジューラは，パフォーマンスカウンタ管理機構，学習機構，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で回帰分析に用いるパフォーマンスカウンタの集計を行う．学習機構は動作周波数ごとに回帰分析を行い，最適な回帰係数を求め，パフォーマンスカウンタの値と性能の関係を導出する．プロセス単位 DVFS 制御機構では，各動作周波数について性能予測式の最適化を行う必要があるため，プロセス単位での周波数制御を行う．

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，学習・省電力化モードの切替えや，各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やオプションを設定でき，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 ケーパビリティを保持している必要がある．

本スケジューラは，Linux スケジューラのコンテキ

表 1 設定可能なオプション（抜粋）
Table 1 Options for DVFS scheduler.

パラメータ	内容
SETMODE_STUDY	学習モードに設定
SETMODE_PREDICTION	省電力化モードに設定
パラメータ（省電力化モードのみ）	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

ストスイッチ部に各種処理を追加する．次章以降では各部の詳細について述べる．

4. 統計情報に基づく性能予測と実行時最適化機構による省電力化

本章では省電力モードにおける各機構の詳細について述べる．

4.1 性能予測機構

本スケジューラでは，2 章で述べた性能予測モデルを用いて，ユーザが与えた許容性能範囲内で最適な周波数を選択し，動作する．ただし，Linux スケジューラに性能予測機構を組み込むにあたり，2 章の先行研究[5] の方式から，性能予測式の最適化方法や回帰分析における説明変数や目的変数の定義にいくつか改善を加えた．

まず，性能予測式は，ターゲット周波数ごとに立式する．各式における目的変数は最高周波数動作時に対する性能比，説明変数は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となる．ここで，計算機の性能を示す指標として，IPS（Instructions Per Second）が広く用いられており，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．目的変数として，この UIPS の比を用いる．

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，説明変数（PMC 値/サイクル数）の値は，計測する周波数によって変わる．たとえば，PentiumM の L2 キャッシュミス回数を例にとれば，同一のコードを実行した場合でも，メモリ入出力に要するサイクル数は動作周波数ごとに変わるため，説明変数の値が変わる．つまり，この値から他の周波数で動作した場合の性能を予測する場合，説明変数と性能（最高周波数に対する UIPS 比）の関係を示す係数（式 (1) における係数 b^u_{vq}）は説明変数計測時の周波数ごとに変わるため，説明変数を計測したときの周波数別，かつ性能予測を行うターゲットの周波数別に用意する必要がある．つまり，周波数を s 段階に変更可能なシステムでは，$s(s-1)$ 通りの回帰方程式を求める必要があり，計算コストが大きい．

そこで，筆者らは説明変数をユーザモード実行命令数で正規化した．これにより，同一のコードを実行すれば，説明変数（PMC 値/ユーザモード実行命令数）の値は説明変数計測時の周波数 v によらず一定となる．つまり，説明変数と性能の関係を示す係数 b^u_{vq} は，ターゲット周波数 u が同一であれば同じとなり，用意する式の数はターゲット周波数別の s 通りでよい．

性能とは処理に要する時間で定義できるため，時間と直接の関係があるサイクル数で正規化した方が正確に振舞いを反映できるが，サイクル数と実行命令数には強い相関関係があり，実行命令数で正規化しても大きな影響はないことが分かっている[12]．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，PentiumM によるシステムでは L2 Total Cache Misses のみでも精度の高い性能予測が可能である．

また，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，プロセスどうしの組合せによって，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 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あるためである．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り，このような設計とした．

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（UIPS）比 Y^u の予測は，あらかじめ学習モードの回帰分析により求めた回帰係数 b^u_0，b^u_1 と説明変数となるパフォーマンスカウンタの値 PMC と，ユーザモード実行命令数 $Uins$ より次式で行う．

$$Y^u = b^u_0 + b^u_1(PMC/Uins) \qquad (2)$$

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，

$$Y^u \geqq Thd \qquad (3)$$

となる最低周波数の u を動作周波数とする．この性

能閾値は，プロセスをどの程度の性能で実行するか，つまり性能要件を指定するものである．$Thd = 1$ となるのは，最高周波数で動作し，さらにすべ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．性能閾値は，全実時間が該当プロセスに割り当てられた場合の最高周波数時の UIPS に対する比，つまり最高性能に対する性能比として指定する．たとえば,「性能をなるべく落とさずに省電力化する」といった場合には 0.9 程度を,「最大限省電力化する」といった場合には，0.1 程度を指定するなどといった利用方法になる．最高性能を基準としたのは，最大性能以外を性能基準とすると，最大性能の基準性能比の値がアプリケーションの特性によって変わり,「性能をなるべく落とさずに省電力化する」ことの指定が難しいためである．

4.2　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，特に時間制約のある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，利用者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く，このような仕組みを提供する．与えられたタイムスライスにかかわらず，同じ処理は同じ実時間で終了させるのが本システムでいう，リアルタイムと考えた．

しかし性能閾値は，全実時間が該当プロセスに割り当てられると仮定して指定した．このとき，たとえば，同時に実行されるプロセス数が 5 プロセスあり，全時間中に該当プロセスに割り当てられる時間が 1/5 になったとする．この場合，同一周波数である処理を終了させるためにかかるプロセス時間は，すべての時間が 1 プロセスに割り当てられる場合と同一だが，実時間は 5 倍となる．同じ実時間，つまり 1/5 のプロセス時間で処理を終了させるためには，元の 5 倍の性能で動作する必要がある．実時間閾値とは，該当プロセスが，割り当てられるプロセス時間によらず同一の実時間で処理を終了させるのに必要な，実時間に対して満たすべきプロセス時間基準の閾値のことである．

ここで，ユーザが与えた性能閾値を Thd とする．このとき，マルチプロセス環境において，すべての時間が該当プロセスの実行に割り当てられた場合と同等の実時間で同じ処理を終了させるのに必要なプロセス時間基準の性能閾値 $TrueThd$ について考える．総時

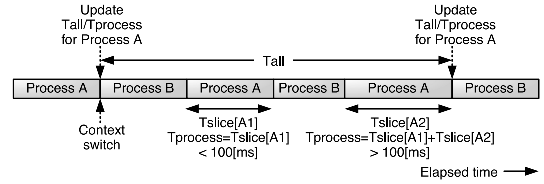

図 2　$T_{Process}/T_{all}$ の更新タイミング
Fig. 2　The timing diagram for updating $T_{Process}/T_{all}$.

間を T_{all}，T_{all} 中に該当プロセスに割り当てられた時間 $T_{Process}$ とすると，全時間に占めるプロセス時間の割合 R は，

$$R = T_{process}/T_{all} \tag{4}$$

となる．全時間が該当プロセスに割り当てられた場合と同じ実時間で処理を終了させるためには，そのプロセスは元の性能 Thd の $1/R$ 倍の性能で動く必要がある．つまり，プロセスが実時間に対して満たすべきプロセス時間基準の閾値 $TrueThd$ は，

$$TrueThd = Thd/R \tag{5}$$

で求めることができる．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．

なお，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（3〜20 ms）とすると，式 (4) の，全時間に占めるプロセス時間の割合 R の変動が比較的大きくなってしまうため，$TrueThd$ 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，**図 2** のようにコンテキストスイッチ複数回に 1 度，再計算する．プロセス A の R を考えた場合，前回，R を計算した時点を時間 0 とする．プロセス A を実行した時間の総和が 100[ms] 以上になったのち，コンテキストスイッチが初めて起きた瞬間の実時間を T_{all}，プロセス時間の総和を $T_{Process}$ として R を計算する．

今回，実装対象とした Linux においてはリアルタイムタスクにおけるデッドラインをスケジューラに対して通知する手段がない．そこで，今回は，時間制約のある Linux プロセスに対し，時間制約を守ることのできる性能閾値をあらかじめ見つけておき，その閾値を外部から与える方法をとった．これにより，ソースコードを改変せずに，可能な限り時間制約を満たしつつ，負荷によってプロセスに与えるタイムスライスを

可変にすることで，省電力化が可能となる．ただし，周期処理などの場合では，処理1回あたりの処理量が可変であると，性能閾値として与える値によっては，時間制約を満たせない可能性が高まる．時間制約の要件によっては，省電力面では不利になるが，処理量が最大となる場合でも時間制約を満たすことのできる性能閾値を与える必要がある．

4.3 フィードバック機構

本研究では性能予測式の説明変数として，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，予測性能と実性能に大きな差異が発生することもありうる．このとき，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，性能制約のあ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，プログラムの振舞いの変化を評価式により検出し，そのつど，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，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．

すでに述べたとおり，本研究において性能指標の単位として UIPS を導入した．最高周波数時の UIPS を $UIPS_{max}$ とした場合，プロセスが実時間に対して満たすべき UIPS, $UIPS_{ideal}$ は，性能閾値を $TrueThd$ として，

$$UIPS_{ideal} = UIPS_{max} \times TrueThd \qquad (6)$$

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，

$$err = UIPS_{ideal} \times T_{ts} - Nins_f \qquad (7)$$

となり，err の総和 S_{err} 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減らすことができる．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．

そこで，本スケジューラでは，**図 3** のように，最高周波数時の性能を実測する「性能計測ステート」，性能予測式から求めた最適な周波数で動作する「予測ステート」，フィードバック機構により性能を調整する「フィードバックステート」があり，条件に従ってタイムスライスごとに状態遷移する．

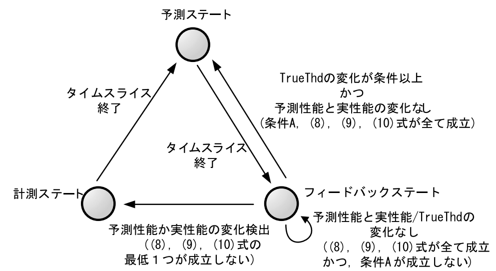

図 3 省電力化モードでの各ステートの状態遷移
Fig. 3 The state diagram in energy-efficient mode.

性能計測ステートでは，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ステートに遷移する．

予測ステートでは，4.1 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．次のタイムスライスではフィードバックステートに遷移する．

フィードバックステートでは，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 $S_{err} < ThdL_{ins}$ の場合，周波数を 1 段階下げ，$S_{err} \geqq ThdH_{ins}$ の場合，周波数を 1 段階上げる．ここで，$TrueThd$ が変化した場合には最適な周波数が変わる．フィードバックステート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，$TrueThd$ の差が $ThdH_{TrueThd}$ 以上になった場合（条件 A）に予測ステートに遷移し，周波数の再予測を行う．本実装では，$ThdH_{TrueThd}$ として，5%を選択している．

また，$UIPS_{max}$ はプログラムの振舞いによって常時変化する．先行研究[5] においては，この処理内容の変化の検出をコンパイラによるフェーズ挿入により行っていた．しかし，本研究においては処理内容の変化および，プロセスどうしのリソース競合などによる最高性能の動的要因による変化を自動的に検出し，最高周波数時の性能の変化に追随するようにした．これにより，実性能の最高周波数時の性能を基準とした理想性能への正確な追随が可能となる．結果的に，性能精度の実現や，省電力化が可能となる．実際には，プログラムの振舞いの変化を検出した場合には，性能計測ステートに遷移し，$UIPS_{max}$ を再計測する．しかし，頻繁に性能計測ステートに遷移すると，性能計測

ステート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．一方，処理内容の変化検出を甘くし，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．よって，プログラムの振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 つの変化を，プログラムの振舞いの変化として検出することとした．

- 予測性能の変化
  PMC_i, $UIns_i$ を，タイムスライス i 終了時のそれぞれパフォーマンスカウンタの値，ユーザモード命令実行数とする．タイムスライス now は現在のタイムスライス番号, $base$ は $UIPS_{max}$ を測定した際のタイムスライス番号とする．PMC_{now}, PMC_{base} が大きく異なる場合，$UIPS_{max}$ が異なる可能性が高い．パフォーマンスカウンタの値をユーザモード実行命令数で正規化した値, $PPI_{base} = PMC_{base}/UIns_{base}$, $PPI_{now} = PMC_{now}/UIns_{now}$ をそれぞれ，式 (2) による次回のターゲットとなる周波数 u の性能予測式 $Y^u = f_{est}(X)$ に入れる．ただし，$PMC/Uins = X$ とする．この差が E_{est} 以上になった場合，つまり式 (8) を満たさない場合，プログラムの振舞いが変化したと見なす．E_{est} は許容される最大の性能相対誤差を示すが，ここでは 5%としている．
  $$|f_{est}(PPI_{now}) - f_{est}(PPI_{base})| \leqq E_{est} \tag{8}$$
- 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舞いの変化をすべ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，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，同一周波数で UIPS は同等になることに注目した．このとき，許容する相対誤差 E_{uips} を考慮してもなお，条件を満たさない場合，つまり式 (9)，(10) を満たさない場合にプログラムの振舞いが変化したと見なす．
  ここで，$Base_UIPS_u$ は，フィードバックステートに遷移してから最初に周波数 u で動作したタイムスライス終了時の UIPS 値とし，この値は他のステートに遷移した場合にはクリアされる．$target$ は現在のタイムスライスでの周波数を示す．つまり，$UIPS_{target}$ は最新のタイムスライスでの UIPS となる．また，$targetfreq$ はターゲット周波数，$maxfreq$ は最高周波数の値とする．
  $$\frac{targetfreq}{maxfreq} - E_{uips} \leqq \frac{UIPS_{target}}{UIPS_{max}} \leqq 1 + E_{uips} \tag{9}$$
  $$\left|\frac{Base_UIPS_{target} - UIPS_{target}}{Base_UIPS_{target}}\right| \leqq E_{uips} \tag{10}$$
  E_{uips} は性能の許容誤差であり，同じく 5%としている．

以上いずれかの変化を検出した場合，プログラムの振舞いが変化したと見なし，性能計測ステートに遷移する．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．

4.4 固定優先度スケジューリングクラスにおける省電力化と周期処理

Linux のリアルタイムアプリケーションにおいては，スケジューリングに固定優先度（SCHED_FIFO や SCHED_RR）スケジューリングクラスを用い，かつ高い優先度を指定する．固定優先度スケジューリングクラスの特殊性は，連続的にプロセスを実行したとしても優先度が変動しないため，待ち状態でなければ，実行状態となる可能性が高いことと考えられる．本スケジューラは実行されたプロセスについて，性能閾値の範囲内で省電力化を試み，他の実行可能状態のプロセス数などについての配慮は行わない．よって，実行状態となる可能性の高いプロセスほど性能閾値の実現可能性が高くなる．そのため，一般的に優先度の高い SCHED_FIFO や SCHED_RR のプロセスでは性能閾値の実現可能性や，その範囲内での省電力化の実現度が SCHED_OTHER プロセスに比べて高いことになる．これは，本スケジューラが固定優先度スケジューリングクラスにおいても有効であることの証左といえる．残りの優先度の低いプロセスについては，優先度の高いプロセスが実行されない時間の範囲内でできる限り性能閾値実現と省電力化を行う．

よって優先度の高いプロセスに低い性能閾値が設定されていた場合，優先的に低い周波数で実行したことで，優先度の低いプロセスが性能閾値を守れない可能性はある．しかし，優先度という概念から，これで良いという理念のもとに本スケジューラでは省電力化を行っている．

ここで，時間制約があり，固定優先度スケジューリングクラスを用いる典型例として，周期処理を取り上げる．周期処理においては，次の周期処理開始時間までに処理を終えればよい場合が多い．そこで，性能閾値として周期処理における周期を T_{period} とし，時間制約をこの値とする．さらに周期処理における 1 回の処理の命令数を N とし，最高周波数でこの処理は $T_{maxfreq}$ で終了するとする．このとき，性能閾値 Thd として T_{period} で処理が終了できる最低の閾値を指定したとすると，

$$Thd = T_{maxfreq}/T_{period} \tag{11}$$

$$UIPS_{max} = N/T_{maxfreq} \tag{12}$$

が成立する．ここで，時間 T_{all} 中での該当プロセスの実行時間を T_{proc} とし，さらに本文中式 (7) における，err は $T_{ts} = T_{proc}$，式 (6)，式 (11)〜(12) より，

$$\begin{aligned} err &= UIPSideal \times T_{proc} - Nins_f \\ &= UIPS_{max} \times (Thd/R) \times T_{proc} - Nins_f \\ &= N \times T_{period}/T_{all} - Nins_f \end{aligned} \tag{13}$$

となる．時間制約を守れれば，$N = Nins_f$，$T_{all} = T_{period}$ となるので，$err = 0$ となる．この場合に $ThdH_{ins} < 0$ として定数を与えることで，フィードバック機構が周波数を下げ，省電力化を行う．一方，次の周期処理開始時間までに処理が終了しない場合には，$N > Nins_f$，$T_{all} > T_{period}$ のいずれかまたは両方を満たす（たとえば，固定優先度，優先度最大の場合は自ら sleep/yield しない限りプロセスが切り替わらないため $N = Nins_f$，$T_{all} > T_{period}$ となる）こととなり，$err > 0$ となるため，デッドラインに間に合わせるように，フィードバック機構が周波数を上げることで，性能を調整する．

5. 性能予測式の学習による最適化

4.1 節で述べた省電力モードにおける性能予測機構で利用する性能予測式 (1) の回帰係数は，あらかじめ複数のプログラムを実行し，パフォーマンスカウンタの値と性能の間の相関関係を回帰分析に基づき，解析することで求める．筆者らの研究[12]では，この回帰分析とカーネルへの性能予測式の適用には一部ユーザの手による解析，実装が必要であり，手間が大きかった．そこで本研究では，システムコールにより学習モードに設定し，ベンチマークプログラムを実行するだけで，自動的に性能予測式における回帰係数を最適化し，性能予測に適用する学習機構を設計・実装した．

以降では回帰分析による性能予測式の具体的な最適化方法について述べる．本スケジューラはタイムスライスごとに，動作周波数 u おける計測回数 i 番目の実性能 $UIPS_i^u$ とユーザモード実行命令数 $UIns_i^u$，パフォーマンスカウンタ 1〜p の変動値 P_{iq}^u（$q = 1$〜p）を計測する．ここで，説明変数 X_q^u を，命令あたりのパフォーマンスカウンタ変動値，目的変数 Y^u を最高周波数での性能 $UIPS^{maxfreq}$ に対する比として，周波数 u での性能予測を行う回帰方程式を次式のように定義する．

$$Y^u = b_0^u + \sum_{q=1}^{p} b_q^u X_q^u \tag{14}$$

さらに，目的変数 Y^u の n 個のサンプル $Y_i^u = UIPS_i^u / UIPS^{maxfreq} (i = 1〜n)$ と，これに対応する説明変数 X_q^u（$q = 1$〜p）の n 個のサンプル $X_{iq}^u = P_{iq}^u / UIns_i^u$ が測定されたとする．ただし，$UIPS_i^u$ と $UIPS^{maxfreq}$ は同一のコードを実行した場合に計測する．このとき，平方和・積和行列 $\left[S_{jk}^u\right]$（$j, k = 1$〜p）と，積和偏差ベクトル $\left[S_{yj}^u\right]$（$j = 1$〜p）を，

$$\begin{aligned} \left[S_{jk}^u\right] &= \sum (X_{ij}^u - \overline{X_j^u})(X_{ik}^u - \overline{X_k^u}) \\ &= \sum X_{ij}^u X_{ik}^u - \overline{X_k^u} \sum X_{ij}^u \\ &\quad - \overline{X_j^u} \sum X_{ik}^u + n \overline{X_j^u} \times \overline{X_k^u} \end{aligned} \tag{15}$$

$$\begin{aligned} \left[S_{yj}^u\right] &= \sum (Y_i^u - \overline{Y^u})(X_{ij}^u - \overline{X_j^u}) \\ &= \sum Y_i^u X_{ij}^u - \overline{X_j^u} \sum Y_i^u \\ &\quad - \overline{Y^u} \sum X_{ij}^u - n \overline{Y^u} \times \overline{X_j^u} \end{aligned} \tag{16}$$

とすると，偏回帰係数ベクトル $\left[b_q^u\right]$（$q = 1$〜p）は，

$$\left[b_q^u\right] = \left[S_{jk}^u\right]^{-1} \left[S_{yj}^u\right] \tag{17}$$

で求めることができる．さらに，

$$b_0^u = \overline{Y^u} - \sum_{q=1}^{p} b_q^u \overline{X_q^u} \tag{18}$$

となる．$\left[S_{jk}^u\right]$，$\left[S_{yj}^u\right]$ と，$\left[b_q^u\right]$ は，$i = 1$〜$n-1$ の，$\sum X_{ij}^u$，$\sum X_{ik}^u$，$\sum X_{ij}^u X_{ik}^u$，$\sum Y_i^u$，$\sum Y_i^u X_{ij}^u$ と，n 個目のサンプル Y_n^u，X_{nq}^u から求めることができる．よって，回帰方程式も $n-1$ までの解析データと n 個目のサンプルから求めることができる．つまり，1 回の回帰方程式更新に要する計算量やメモリ使用量はデータ数 n によらず，説明変数の個数 p と周波数変更可能段階数 s のみに依存する．計算量は s 個の p 元連立一次方程式 (17) を解く計算量 $O(sp^3)$ となる．しかし，PentiumM においては，先行研究[5]より $p = 2$ 程度でも十分精度の高い性能予測が可能であり，計算量は小さい．

すでに述べたように，本システムにおいては説明

変数は 1 変数で性能予測を行っている．単回帰分析（$p=1$）のとき，式 (17)，(18) は，

$$b_1^u = S_{y1}^u / S_{11}^u = \frac{\sum (X_{i1}^u - \overline{X_1^u})(Y_i^u - \overline{Y^u})}{\sum (X_{i1}^u - \overline{X_1^u})^2} = \frac{\sum X_{i1}^u Y_i^u - \overline{Y^u} \sum X_{i1}^u - \overline{X_1^u} \sum Y_i^u + n\overline{X_1^u} \times \overline{Y^u}}{\sum {X_{i1}^u}^2 - 2\overline{X_1^u} \sum X_{i1}^u + n\overline{X_1^u}^2} \quad (19)$$

$$b_0^u = \overline{Y^u} - b_1^u \overline{X_1^u} \quad (20)$$

となる．PentiumM において，計算量は，$s(s-1)$ 個の式を立式する必要があった従来方式に比べ，本方式では s 個の立式でよいため，$s=9$ より，1/8 となる．

なお，すでに述べたとおり，目的変数は最高周波数時との性能比になるため，同一コードを実行したときの性能を比較する必要がある．そのため，先行研究[12]では，同一のプログラムをすべての周波数で複数回実行させ，同一箇所で計測を行っている．しかしながら，これには学習に時間がかかるほか，手間も大きい．そこで，本カーネルでは，プログラム 1 回の実行でコンテキストスイッチごとに周波数を変動させ，説明変数の値が連続して同一の場合には，同一コードが動作していると仮定し，学習を行うようにしている．具体的には相対誤差 $E_{pmcdiff}$ を考慮して，取得データが下記式を満たすとき学習を行う．

$$\left| \frac{X_i^{targetfreq} - X_i^{maxfreq}}{X_i^{maxfreq}} \right| \leqq E_{pmcdiff} \quad (21)$$

以上で述べた定義を適用した，学習モードに関する擬似コードを**図 4** に示す．図 4 の処理はコンテキストスイッチごとに行う．

6. 実　　装

前章までに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，同じ x86 の中でも特に電源電圧・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 533 MHz）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，省電力モードで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

```
context_switch(next) //next:次のプロセス
  next.pmcpinst[next.freq] ← getpmc() //命令あたりの PMC 変動値取得
  next.uips[next.freq] ← getuips()    //UIPS 取得

  pmcdiff ← next.pmcpinst[MAX_FREQ] - next.pmcpinst[next.freq]
  if |pmcdiff| > Epmcdiff
    // PMC 変化検出（最初から再度統計取得）
    next.freq ← MAX_FREQ
  elseif next.freq = 1
    // 学習実行
    study++
    for i ← 1..MAX_FREQ
      // Σ x[i], Σ y[i], Σ xy[i], Σ x^2[i] 更新
      y      ← next.uips[i] / next.uips[MAX_FREQ]
      sx[i]  ← sx[i] + next.pmcpinst[i]
      sy[i]  ← sy[i] + y
      sxy[i] ← sxy[i] + next.pmcpinst[i] * y
      sx2[i] ← sx2[i] + next.pmcpinst[i] * next.pmcpinst[i]
      ax     ← sx[i] / study
      ay     ← sy[i] / study
      // b0[i], b1[i] 更新
      b1[i] ← (sxy[i] - ay*sx[i] - ax*sy[i] + study*ax*ay) /
                 (sx2[i] * ax * sx[i] +  study * ax * ax)
      b0[i] ← ay - b1[i] * ax

    next.freq ← MAX_FREQ
  else
    // 統計取得継続（次の周波数）
    next.freq ← next.freq - 1

  setfreq(next.freq)
  resetpmc()
```

図 4 性能予測式の学習アルゴリズム
Fig. 4 The learning algorithm for performance prediction.

表 2 スケジューラの基本情報
Table 2 DVFS scheduler specifications.

項目	値
ソースコード総行数	1,191[行]
コンテキストスイッチ所要サイクル（すべて）	4,693[cycle]
コンテキストスイッチ所要サイクル（追加部分のみ）	3,504[cycle] (75[%])
総時間中コンテキストスイッチ時間（オーバヘッド）	0.01[%]

ごとに統計処理・性能予測・周波数切替えを行うが，表 2 に示したように，総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 から得られるプロセスの待ち時間 sleep_avg などの情報などを用いて，複数の説明変数

表 3 学習結果
Table 3 Learing results.

ベンチマーク	実行時間 [s]	学習実行回数	L2TCM/inst
matrix	30.0	81	3.76×10^{-3}
FFT	18.5	20	2.46×10^{-3}
SHA	24.6	60	6.41×10^{-6}
Dijkstra	56.3	62	4.75×10^{-5}
QuickSort	11.5	6	1.56×10^{-3}

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モードを学習モードに設定し，複数のプログラムを実行し，性能予測に用いる回帰方程式の回帰係数を学習しておく必要がある．そこで，あらかじめ行列乗算（行列サイズ 1500×1500）と MiBench [11]（FFT/SHA/Dijkstra/QuickSort）について 1 回ずつ実行し，学習を行った．利用するパフォーマンスカウンタの要素は先行研究[5] におい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．学習時の各プログラムの実行時間，学習実行回数，説明変数として利用する命令あたりの平均 L2 キャッシュミス回数（L2TCM/inst）は，**表 3** のようになった．

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方式（PMC+feedback(fb) 方式），統計情報を用いずに周波数比がそのまま性能比になると仮定して性能予測を行い，フィードバック機構を併用する方式（freqonly+feedback(fb) 方式），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方式（PMC 方式），統計情報を用いずに周波数比がそのまま性能比になると仮定して性能を予測し，フィードバック機構は無効となる方式（freqonly 方式）を選択し，利用することができる．freqonly 方式，PMC 方式では，4.3 節で述べた性能計測ステートとフィードバックステートには遷移しない．

7. 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 4** に示す．以降の評価では，freqonly 方式，PMC 方式，freqonly+feedback(fb) 方式，PMC+feedback(fb) 方式のそれぞれで評価を行う．

表 4 PentiumM 760 の基本情報
Table 4 PentiumM 760 specifications.

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

7.1 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（THD）を変化させ，行列乗算と MiBench（FFT，SHA，dijkstra，QuickSort），SPEC CPU2000（181.mcf，171.swim，183.equake），ディスク読み込みのベンチマークを実行した場合の CPU 消費電力量の計測を行った．性能閾値は最高周波数で動作した場合に対する許容性能を指定する．たとえば，最高周波数時に 1 秒で終了するプログラムに対して，2 秒で終了すればよい場合，性能閾値は 0.5 となる．消費電力量の測定結果を，性能閾値 1.0 の場合，つまりつねに最高周波数で動作する場合の電力量比として**図 5** に示す．

結果より，周波数制御を行わない場合に比べ，すべての方式において消費電力量削減が実現できている．特に性能閾値 0.9 のとき，freqonly 方式に比べ，PMC+feedback 方式ではディスク読み込みベンチマークで 45%，CPU ベンチマーク 171.swim で 23%の消費電力量削減を実現できており，本方式の有用性を示す結果である．特に，CPU/メモリバウンドの MiBench や行列乗算，SPEC CPU2000 では，freqonly 方式に比べて，PMC 方式，freqonly+feedback 方式，PMC+feedback 方式で消費電力量の削減量はそれぞれ，平均 6.6%，2.3%，4.6%と性能予測が特に有効に働き，PMC 方式が最も消費電力量を削減できている．また，freqonly 方式と PMC 方式を比べた場合，最大 17%程度の消費電力量削減をパフォーマンスカウンタを利用した性能予測により実現している．ただし，183.mcf（性能閾値 0.9）では 37%の消費電力量を削減できているが，後述の理由によりこれは除外している．一方，ディスク読み込みプログラムを加えた場合は，平均 5.9%，5.3%，7.0%の削減と，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．

また，ディスク読み込みや matrix, FFT, 171.swim, 183.equake の一部では，PMC による性能予測と実性能に差が発生するため，フィードバック機構が特に有効に働き，PMC のみの予測と比べ，消費電力量の削

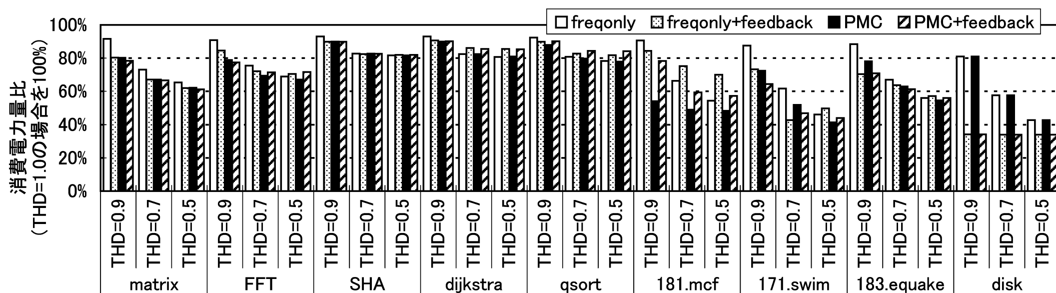

図 5 各方式の閾値別消費電力量比
Fig. 5 Energy consumption results.

減に成功している．年々複雑化するシステム環境下や，様々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．

また，SHA 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電力量の差は小さい．一方で，性能閾値が小さい場合，律速要素の値が頻繁に変化するプログラムはフィードバック機構にとって苦手である．実際に，FFT や QuickSort 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 方式と比べてフィードバックを有効にすると，消費電力量が最大 10%程度増加している．これは，プログラムの振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，プログラムの振舞いの変化検出回数・箇所を最適化するために，論文 5) で述べられているコンパイラによるフェーズ挿入や，フィードバックアルゴリズムの変更が有効と考えられる．

freqonly+feedback 方式と PMC+feedback 方式では，PMC+feedback 方式の方が平均して消費電力量を削減できており，フィードバック機構を併用する場合にもパフォーマンスカウンタによる性能予測が有効である．これは，次節の評価より，計測ステートから予測ステートに移行後，正確に性能を予測でき，フィードバックに移行したとき，freqonly+feedback 方式に比べて，最適な周波数への収束速度が速いことと，パフォーマンスカウンタの変化によるプログラムの振舞い検出により，基準性能が正確に追跡できており，動作周波数を適切に下げることができるためである．特に，FFT や QuickSort のような頻繁にキャッシュヒット率が変わるベンチマークで最大 16%の差が現れている．

実際のプログラムにおいては，CPU バウンドといわれるアプリケーションにおいてもファイル入出力をともなうなど，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．このように 1 つのプログラムの中でも律速要素が変わることも考えられる．よって，フィードバック機構のみ，パフォーマンスカウンタによる性能予測のみの一方では不十分であり，性能予測がよく当たり，さらに基準性能が頻繁に変わる場合には PMC 方式，性能予測が外れる場合には PMC+feedback 方式と使い分けるのが良いと考えられる．

また，Linux で利用される従来の CPU 省電力化環境である cpufreq との比較についても行った．評価に用いる governor は，システムが低負荷時には周波数を低下させ，省電力化する ondemand を利用した．

結果より，cpufreq+ondemand における消費電力量は，CPU/メモリベンチマークでは対常時最高周波数動作で 100〜99%程度となり，cpufreq+ondemand と，本スケジューラの PMC+feedback（性能閾値 0.9）の消費電力量を CPU/メモリベンチマークで比較した場合，CPU/メモリベンチマークでは最大 36%，平均 20%ほど本スケジューラの方が消費電力量削減を実現している．対してディスク読み込みベンチマークでは，ほぼ同等の結果となった．

7.2 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つながる．そこで，本スケジューラの精度の評価を行うため，各種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 6** 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み込みベンチマークは除く），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 7** に示す．ここで誤

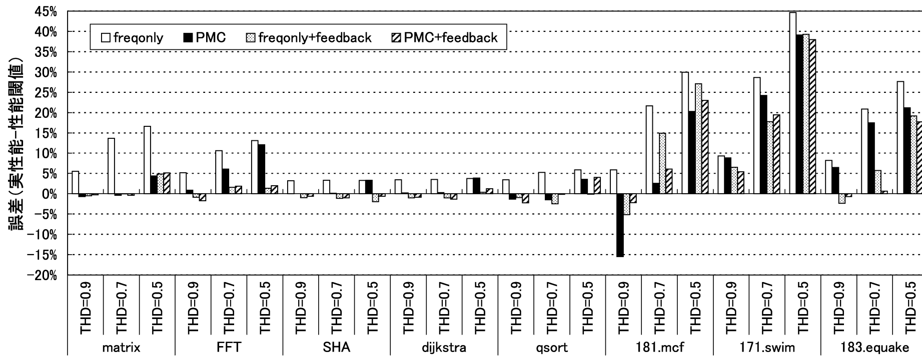

図 6 各方式の性能誤差
Fig. 6 Performance error results of each method.

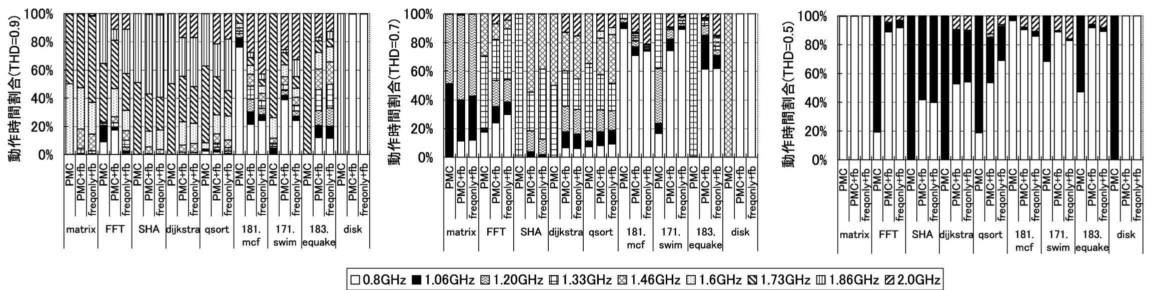

図 7 各方式の動作周波数内訳
Fig. 7 Frequency statistics of each method.

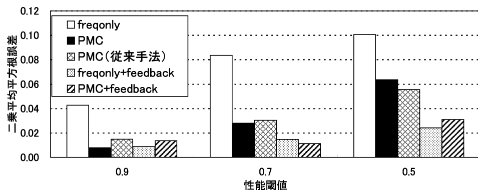

図 8 各方式の閾値別性能誤差
Fig. 8 Performance error results of varying thresholds.

差とは，実性能（最高周波数時に対する相対性能比）から性能閾値を引いた値と定義する．つまり，誤差が正であれば，性能閾値を上回る性能，すなわち閾値から期待される時間よりプログラムが早く終了したことを示す．また，それぞれの方式の精度を比較するために，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（MiBench）を**図 8** に示す．二乗平均平方根誤差については，従来[12] の同一プログラムを全周波数で複数回実行し，同一コードの箇所で統計をとり，回帰分析を行って性能予測式を最適化した場合の比較もあわせて示す．

結果より，多くの場合で PMC+feedback 方式，freqonly+ feedback 方式が最も正確，次に PMC 方式が正確であり，本システムの有用性を示す結果となっている．特に，171.swim，183.equake やディスク 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み込みベンチマークでは，閾値 0.9 の動作周波数の内訳において，PMC 方式，freqonly 方式に比べて特に低い周波数で動作しており，省電力化できている．一方，PMC 方式の 181.mcf（性能閾値 0.9）では，freqonly 形式に比べて大幅に省電力化できていたが，性能閾値を大きく下回っている．しかし，フィードバックを有効にすることにより，性能誤差は 5%以下となっている．

また，フィードバックが有効な方式では，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，ほとんどで誤差が設計で述べた許容誤差 5%以内に収まっていることが確認できる．matrix や FFT，171.swim，183.equake などの一部では性能予測のみでは性能閾値より高い性能を示す．フィードバック機構を併用することで，許容性能内でさらに消費電力量を削減できた例である．

フィードバック機構を有効にした場合，最適な周波数（つまり理想性能）への収束時間は，予測ステートで予測性能により選択した周波数での実性能と理想性能の差による．周波数はコンテキストスイッチご

とに 1 段階ずつ変更するため，その差が小さいほど収束速度が速くなる．図 6 を見ると，多くのベンチマークにおいて，freqonly 方式よりも，PMC のほうが高い精度で性能予測を行えている．つまり，フィードバック機構を有効にしたとき，理想性能を実現できる周波数への収束速度は，freqonly+feedback 方式よりも，PMC+feedback 方式のほうが速い．図 6 より，PMC 方式に比べて，freqonly 方式では，matrix，FFT，171.swim などでは予測性能が大きく性能閾値を上回っている．実際に，図 7 の周波数の内訳を見た場合，freqonly 方式よりも，PMC 方式のほうが低い周波数を選択するので，feedback を有効にした場合，PMC+feedback 方式のほうが低い周波数（最適周波数）への到達時間が短く（収束速度が速く），その周波数で動作する時間が長くなる．これにより省電力化に寄与している．

しかしながら，PMC+feedback 方式では QuickSort の性能閾値 0.5 などにおいて PMC 方式に比べ，性能誤差が大きくなっている．これは，図 7 の結果より，QuickSort では頻繁にキャッシュヒット率が変化し，計測ステートに遷移し，最高周波数で動作する割合が大きくなるためである．freqonly+feedback 方式では，パフォーマンスカウンタの値を利用しないため，式 (8) の予測性能の差によるプログラムの振舞い変化を検出できないため，計測ステートに移行しにくい．QuickSort においては，パフォーマンスカウンタの変動が性能にそれほど大きく影響しないため，偶然性能誤差は小さくなっている．しかし，他の箇所においては予測性能差によるプログラムの振舞いの変化検出が消費電力量や性能精度に好影響を与えている箇所もあるため，式 (8) をプログラムの振舞い変化の検出対象から外すのは好ましくない．この誤差は式 (8) における許容する誤差を大きくすることで，改善が可能である一方，他のベンチマークや性能閾値において悪影響を及ぼすことも考えられる．

また，従来方式と比べても遜色のない予測精度を実現している．理想は，従来のように同一のコードを複数の周波数で複数回実行し，統計をとる方法だが，本方式でも十分な精度を実現している．学習時には，PentiumM760 において周波数は 9 段階に設定可能であるため，従来方式ではそれぞれのプログラムについて 9 回ずつ実行する必要があったが，本学習機能では，それぞれのプログラムを 1 回ずつ実行するのみでよく，学習に必要な時間は従来方式の 1/9 と大幅な削減に成功している．学習時にも，通常の計算機利用方法との差はない．よって，一般ユーザにとっても利用は容易であり，それぞれのユーザの利用方法や計算機環境に応じて性能予測式を最適化することも容易である．

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合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．消費電力量面でも性能精度面でも，PMC 方式と PMC+feedback 方式を場合によって切り替えて利用することが最適である．

7.3 マルチプロセス時の精度評価

時間制約のあ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の精度に関する評価を行った．

評価に用いるアプリケーションとしては，既存の行列乗算プログラムおよび，同じ行列乗算で時間制約を想定した固定優先度の周期処理プログラムを作成し，これらを同時に実行し，精度評価を行った．

周期処理プログラムは，従来の行列乗算をもとに，これを 1500 の段階に分割し（1 行分の行列を求める操作を 1 段階とする），一定時間間隔で周期処理を行い，残り時間は sleep するようなプログラムとした．処理間隔は 80[ms] 間隔（12.5[Hz]）であり，最高周波数で 1 段階の処理は約 16[ms] で終了し，残り時間は sleep する．時間制約は次の周期が来る前に処理を終えることとする．つまり，最高周波数では時間制約を満たしつつ，最大 5 プロセスの同時実行が可能である．スケジューリングクラスは SCHED_FIFO とし（精度は SCHED_RR でもほぼ同じであった），優先度はスケジューリングクラス内での最大値とする．

評価としては，PMC+feedback 方式で，通常の行列乗算プロセスを 1〜6 プロセス動作させた場合，および，2 プロセス（1 プロセスが周期処理プロセス），6 プロセス（1 プロセスが周期処理プロセス），6 プロセス（5 プロセスが周期処理プロセス）のそれぞれの場合について計測を行った．性能閾値は 0.2 倍としている．

計算終了までの実時間を計測した結果を**図 9** に，そのときの動作周波数の内訳を**図 10** に示す．ここで図 9 の結果は，通常の行列乗算プロセスを最高周波数で 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．

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 5 以下での最大誤差は 4%となった．

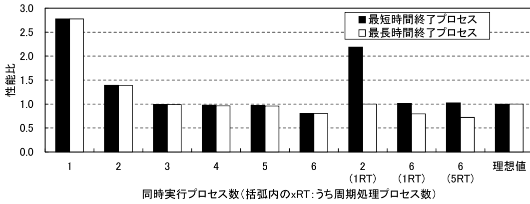

図 9 マルチプロセス動作時の性能（実時間性能指定の精度評価）
Fig. 9 Performance error results of multi-process.

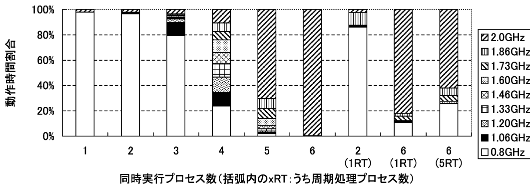

図 10 マルチプロセス動作時の動作周波数時間内訳
Fig. 10 Frequency statistics of multi-process.

この結果は，設計時に設定した最大許容誤差である5%の範囲内に収まっており，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．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．

また，周期処理プロセスは 2(1RT)，6(1RT)，6(5RT) のすべてにおいて高い精度で実行が行えており，特に優先度の高い固定優先度スケジューリングクラスのプロセスは通常のプロセスに比べて設定性能の達成度が高いことが分かった．加えて，全段階のうち，84[ms] 以内に処理が終わったのは，2(1RT)，6(1RT) で 100%，6(5RT) で 98%であった．ここで 84[ms] としたのは，Linux のスケジューリング間隔が 4[ms] であるためである．この数値は本スケジューラを介さない場合とほぼ同等であった．よって本スケジューラ導入による時間制約に対する達成度の低下はほぼないといえる．時間制約の達成だけでなく，2(1RT) では，周期処理プロセス，通常プロセス両方において省電力化が行われている．特に，周期処理プロセスの省電力化は，4.4 節で述べたように，フィードバック機構によるものである．本スケジューラは周期処理プロセスにおいても時間制約の達成と省電力化を同時に達成した．

8. 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ごとの細かい性能要求に対して対処することができない．また，性能予測を行うものではないため，性能制約のあるアプリケーションでの利用には向かない．また，7 章で述べたとおり，メモリバウンドアプリケーションで省電力化が行われないといった弱点もある．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ほか，様々な律速要素に応じた周波数制御が可能である．

Windows においては，デバイスごとに細かい電源制御インタフェースの規定を行っており，一定時間利用がない場合には電源制御を行うことで省電力化を実現している．CPU の省電力化環境には，AMD の Cool'n'Quiet などといった省電力化ソフトウェアが存在する．しかし，これらも内容的には Linux の cpufreq とほぼ同等で，システム全体で周波数制御，本研究と違い，CPU・メモリバウンドアプリケーションで省電力化が行われない，といった問題が存在する．

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精度には疑問が残る．たとえば，論文 7) では，CPU アイドル時間の目安として Linux のプロセスごと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アプリケーションが走ることが多い環境など，すべてのシステムで利用できるわけではない．

一方，近年の CPU の高度化にともな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．たとえば，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，消費電力量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予測式の最適化を行うため，複雑なシステムにおいても最適化は自動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．

9. おわりに

本稿では，Linux 向けに統計情報に基づいて自動的に性能予測式を最適化し，それを用いた性能予測とフィードバック機構により，性能の最適化を行う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式の最適化が容易であり，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，省電力化を行いながらも，システムの負荷状況にかかわらず高い性能精度を実現し，ある程度，時間制約のあるシステムへの応用も可能である．さらに評価より，単純な周波数比による性能予測（freqonly 方式）に比べて，学習による性能予測のみで CPU ベンチマークでは最大 17%，性能予測とフィードバック機構を併用することで I/O ベンチマークでは最大 47%，CPU ベンチマークでは最大 23%の消費電力量の削減を実現し，本方式の有用性を実証した．

今後の課題として，まずフィードバック機構のアルゴリズムの最適化，特に基準性能の変化速度，頻度に応じたフィードバックポリシの変更がある．これにより，評価で一部，消費電力量が増大している部分の改善や，精度の改善が期待できる．また，組み込み環境への展開など，多くのアーキテクチャへの移植・評価をあげることができる．

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「革新的電源制御による超低電力高性能システム LSI の研究」によるものである．

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) Brodowski, D.: 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：コンピューティングシステム，Vol.47, No.SIG18(ACS16), pp.80–91 (2006).
6) Weiser, M., Welch, B., Demers, A. and Shenker, S.: Scheduling for reduced CPU energy, *Proc. Symposium on Operating Systems Design and Implementation*, pp.13–23 (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 2006-OS-103, pp.109–115 (2006).
8) Yuan, W. and Nahrstedt, K.: Energy-efficient soft real-time CPU scheduling for mobile multimedia systems, *Proc. 19th ACM Symposium on Operating Systems Principles* (*SOSP'03*), pp.149–163 (2003).
9) Gruian, F.: Hard real-time scheduling for low-energy using stochastic data and DVS processors, *Proc. 2001 international symposium on Low power electronics and design*, pp.46–51 (2001).
10) 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, *Proc. 38th annual IEEE/ACM International Symposium on Microarchitecture*, pp.271–282 (2005).
11) Guthaus, M.R., Ringenberg, J.S., Ernst, D., Austin, T.M., Mudge, T. and Brown, R.B.: MiBench: A free, commercially representative embedded benchmark suite, *Proc. IEEE 4th Annual Workshop on Workload Characterization*, pp.3–14 (2001).
12) 金井　遵，佐々木広，近藤正章，中村　宏，並木美太郎：統計情報に基づく省電力 Linux スケジューラ，情報処理学会研究報告 2007-OS-106, pp.9–16 (2007).

(平成 19 年 7 月 23 日受付)
(平成 19 年 12 月 13 日採録)

金井　遵（学生会員）

2006 年東京農工大学工学部情報コミュニケーション工学科卒業．2007 年同大学大学院工学府情報工学専攻博士前期課程修了．現在，同大学院工学府電子情報工学専攻博士後期課程在学中．

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．

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）．同年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．2004年東京大学先端科学技術研究センター特任助手，2007年同特任准教授．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，現在，東京大学先端科学技術研究センター准教授．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．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，ディジタルシステムの設計支援の研究に従事．特に，回路，アーキテクチャ，OSの協調による高性能・低消費電力コンピュータシステムの実現に興味を持つ．現在，情報処理学会計算機アーキテクチャ研究会主査．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．

天野　英晴（正会員）
1981年慶應義塾大学理工学部電気工学科卒業．1986年同大学大学院理工学研究科電気工学専攻博士課程修了．現在，慶應義塾大学理工学部情報工学科教授．工学博士．計算機アーキテクチャの研究に従事．

宇佐美公良
1982年早稲田大学理工学部電気工学科卒業．1984年同大学大学院理工学研究科修士課程修了．同年（株）東芝入社．半導体技術研究所にてマイクロプロセッサの研究開発に従事．1993〜1995年スタンフォード大学客員研究員．2000年早稲田大学より博士（工学）を授与．2003年芝浦工業大学工学部情報工学科助教授．2005年より同大学教授．研究分野は，システムLSIの低消費電力設計技術．IEEE，ACM，電子情報通信学会会員．

並木美太郎（正会員）
1984年東京農工大学工学部数理情報工学科卒業．1986年同大学大学院修士課程修了．同年4月（株）日立製作所基礎研究所入社．1988年東京農工大学工学部数理情報工学科助手．1989年電子情報工学科助手．1993年11月電子情報工学科助教授．1998年4月情報コミュニケーション工学科助教授．現在，東京農工大学大学院共生科学技術研究院教授．博士（工学）．オペレーティングシステム，言語処理系，ウィンドウシステム等のシステムソフトウェア，並列処理，コンピュータネットワークおよびテキスト処理の研究・開発・教育に従事．ACM，IEEE各会員．