情報処理学会論文誌　コンピューティングシステム　Vol. 2　No. 1　96–109 (Mar. 2009)

L4 マイクロカーネルにおける省電力スケジューラの開発

林　　和宏[†1]　金井　　遵[†2]
丸山勝巳[†3]　並木美太郎[†4]

近年，サーバや組み込み機器などのあらゆる分野における省電力化技術の必要性が高まっている．我々は，これらの電力問題を解決する手段の 1 つとして，L4 マイクロカーネル上で DVFS 制御により省電力化を行うスケジューラ機構を提案する．本スケジューラは，L4 スレッド単位で実行時情報に基づいた性能予測を行い，ユーザによってあらかじめ定められた性能制約を満たす範囲内で最適な CPU 周波数を選択することにより省電力化を図る．また，性能予測モデルを実行環境に応じて自動的に最適化する学習機構，および性能予測に誤差が生じる際にこれを補正するフィードバック機構を有する．本論文では，本スケジューラの設計と実装および評価について述べ，ユーザレベルの OS サーバをアプリケーションプログラムなどと同等に電力制御することによるマイクロカーネルにおける本システムの適用性ならびに省電力化の可能性について考察する．評価では，I/O アプリケーション実行時に最大 34%の CPU 消費エネルギー量の削減を実現した．

Implementation of a Power-saving Scheduler on L4 Microkernel

Kazuhiro Hayashi,[†1] Jun Kanai,[†2]
Katsumi Maruyama[†3] and Mitaro Namiki[†4]

Recently, the low power technology is needed in many fields such as server computers or embedded systems. We propose a power saving scheduler with DVFS (Dynamic Voltage and Frequency Scaling) on L4 microkernel. The scheduler reduces power consumption by predicting the performance of each L4 thread based on its run-time information and selecting the best voltage and frequency that satisfies the performance constraint defined by users. The scheduler has the “learning system” optimizing the performance estimation model automatically in each environment, and the “feedback system” compensating the performance estimation error. This paper describes the design, implementation and evaluation of the scheduler, and then, considers the applicability of the scheme to L4 microkernel and the possibility of power saving in microkernel by applying power-aware scheduling to user level OS servers similarly to applications. In evaluation, we achieved 34% CPU energy reduction of an I/O application.

1. はじめに

近年，コンピュータシステムにおける電力問題が深刻化している．サーバやクラスタシステムにおいては，マシン性能の向上にともなってその消費電力と発熱が増大し，電力コストおよび冷却のための空調コストが大きな問題となっている．また，バッテリ駆動型の組み込み機器やポータブル機器などにおいては，電池残量を効率的に使用するための電力制御技術が必須となる．

このような消費電力の問題に対し，省電力化技術に関する多くの研究・開発が行われている．これらの技術は，基本的にハードウェアによる自律的な電力制御と，ハードウェアの提供する機能を利用したソフトウェアによる電力制御の 2 つに大別される．ハードウェアレベルでの省電力化手法では，回路レベルでの細粒度な設計・制御が行えるほか，クロック供給のないときにも消費されるリーク電流を削減するための技術の構築が可能である．その反面，ソフトウェアによって仮想化されるタスクなどの各種資源に関する情報の取得が困難であり，システム内に存在する有益な情報に基づいて最適な電力制御を行うことが難しい．一方で，ソフトウェアレベルでの省電力化技術では，手段自体はハードウェア機能の実装に依存するものの，デバイスの使用状況やその使用者であるタスクなどの管理情報を用いた柔

†1 東京農工大学大学院工学府情報工学専攻
Department of Computer and Information Sciences, Graduate School of Engineering, Tokyo University of Agriculture and Technology
†2 東京農工大学大学院工学府電子情報工学専攻/日本学術振興会特別研究員（DC2）
Department of Electronic and Information Engineering, Graduate School of Engineering, Tokyo University of Agriculture and Technology/Research Fellow of the Japan Society for the Promotion of Science (DC2)
†3 国立情報学研究所
National Institute of Informatics
†4 東京農工大学大学院共生科学技術研究院
Division of Systems and Information Technology, Institute of Symbiotic Science and Technology, Tokyo University of Agriculture and Technology

軟な電力制御を実現できる．

ハードウェアによる電力制御の例としては，電源状況に応じて自動的に電圧とクロック数を変更する Intel 社の SpeedStep 技術や，回路内で部分的に電力供給を停止させることで消費電力を削減するパワーゲーティング[12] などがある．また，ソフトウェアレベルでの電力制御としては，CPU の周波数・電圧動的調整機能 DVFS（Dynamic Voltage and Frequency Scaling）を用いた OS などによる省電力化技術が代表的である．DVFS 機能の具体例として，Intel 社の拡張版 Intel SpeedStep [4]，AMD 社の Cool'n'Quiet および PowerNow!，Transmeta 社の LongRun などがあげられる．DVFS 機能を利用したソフトウェアによる電力制御の例としては，Linux カーネル 2.5 以降に搭載されている CPUFreq 機構，ARM 社の IEM ソフトウェアなどがあり，このほかにも DVFS に関する多くの研究がなされている．

Linux における CPUFreq においては，CPU 負荷や温度，動作プログラムの種類などといった情報に基づいて，指定された電力制御ポリシに従った DVFS 制御を行っている．また，CPU 速度・電圧調整に関する過去の論文[3] では，一定間隔ごとに CPU 負荷を計算し，結果に基づいて CPU 速度を決定する種々のポリシを提案している．しかし，これらの手法はタスクごとの動作挙動や I/O などの情報を考慮しておらず，システム状態を 1 つのブラックボックスとして扱っていることになる．ハードウェアの使用頻度や CPU 負荷などが異なる多様なプログラムが複数同時実行されるようなコンピュータシステムにおいては，タスクごとの動作特性を考慮した電力制御を行うことはきわめて重要である．加えて，上記の例では CPU 周波数の低下に際する性能面での保証がなされていない．実用的なシステムへの適用，特にリアルタイムなどの時間制約の厳しい状況下での実装を考えたときには，DVFS 制御によるプログラム実行性能への影響を予測し，ある程度のパフォーマンスを保証できる能力が求められる．このために，性能の定量的なモデル化といった作業が必要になる．

パイプラインステージ統合と DVFS を複合した研究[13] では，目標となるスループットを設定し，これを満たす最小の CPU 周波数・電圧を選択するという手順を踏むことで，周波数選択における性能保証を行っている．また，ハードリアルタイムシステムを対象とした DVFS 制御手法[9] においては，EDF や RM といったスケジューリングに対しタスクのリアルタイム性を保証したうえで周波数・電圧決定を行う複数のアルゴリズムが提案されている．これらの論文では，いずれも性能の比率が直接周波数比となるものとして予測を行っている．しかし，実際のタスク性能はメモリアクセスや I/O といった多くの外的要因に左右されるため，CPU 周波数比のみを用いた予測では最適結果が得られない可能性が高い．特に，上記要因によって何らかのレイテンシが生じる場合には，周波数低下による性能への影響が減少し，省電力化の可能性が高まる．したがって，DVFS 制御における性能予測は，周波数以外の外的要因も同時に考慮したうえで行うことが理想的である．タスクごとの動作特性を考慮したうえで性能をモデル化し，その予測に基づいた DVFS 制御を行う既存の手法として，性能指標となる CPI（Cycles Per Instruction）とメモリレイテンシとの間で線形重回帰モデルを構築することにより，DVFS 制御における実行時性能予測を行う研究[7] や，リアルタイムシステムにおいてタスクのメモリアクセス率をモニタリングし，タスク実行時間の予測に利用する DVFS フィードバック手法に関する研究[10]，処理時間の CPU 依存性によってワークロードを 2 種類に分割し，各々で周波数決定ポリシを変えて制御する MPEG デコード処理のための研究[2] などが存在する．

本論文では，コンピュータシステムにおける電力問題を解決する手段の 1 つとして，OS レベルでの定量的な性能予測モデルに基づく DVFS 制御手法を提案する．先行研究[6] では，性能予測モデルの自動学習と，これを用いたプログラムごとの最適な動作周波数選択を行うスケジューラ機構を Linux カーネル上で実現している．本スケジューラは，Linux プロセス単位でその動作特性に基づいた性能予測を行い，ユーザにより定められた性能制約を保証する範囲内で周波数を調整するため，プログラムごとの動作特性および周波数低下による性能への影響を考慮した最適な電力制御が可能である．なお，ここでいう性能制約とはあくまで省電力化における性能の目標値であり，本制約に対してシステムがデッドライン保証や制約超過時の対処を行うことを意味するものではない．

本研究では，先行研究[6] の提案する省電力機構をマイクロカーネル OS 上で実現する．マイクロカーネルに着目する主な理由は，その OS 構成におけるタスク実装の特異性にある．カーネル機能が最小限に抑えられるマイクロカーネル OS ではその他のサービスがユーザレベルで実装されるため，スケジューラによる個々の OS タスクに対する DVFS 制御が可能になる．前述した関連研究はいずれもアプリケーションに対する DVFS 制御の例であるが，これらの理論をマイクロカーネルに適用することにより，OS 処理に対しても電力の最適化を行うシステムを実現することができる．そこで本研究では，マイクロカーネル内のスケジューラに対して省電力機構を実装し，実機上での評価を通じてマイクロカーネルシステムにおける省電力化の可能性について考察する．

2. 本研究の目標

マイクロカーネル OS では，カーネルはプロセス間通信（IPC）や割込み通知といった必

要最低限の機能を持ち，ファイルシステムやデバイスドライバ，割込みハンドラといったOS 機能の大半がユーザレベルの OS サーバとして実装される．このため，カーネルスケジューラはアプリケーションタスクだけでなく，OS サーバに対してもその動作特性に基づいた電力制御を行うことが可能である．これに対し，一般的なモノリシックカーネル OS においては，デバイスドライバなどの多くの機能が 1 つのカーネルプログラムとして存在し，割込みや例外，システムコールを通じて不定期に呼び出されるため，スケジューラによってこれらの OS 処理を個別に電力制御することは困難である．このため，マイクロカーネル OS はモノリシックカーネル OS に比べて OS 処理の細粒度な電力制御が行えるという点でより高い省電力化効果が期待できる．そこで本研究では，マイクロカーネルに対して先行研究[6] の提案する省電力スケジューラを実装することにより，プログラムの実行時性能を考慮した効率的な省電力化を実現するとともに，各種 OS 処理についてもユーザプログラムと同等に電力制御するシステムを構築し，その評価を行うことを目標とする．

実装対象には，著名なマイクロカーネルである L4[1] を利用する．一般的に，マイクロカーネル OS においては，OS 機能がユーザタスクとして分離されて実装されるために IPC やそれにともなうカーネル・ユーザモードの切替えが頻繁に発生し，モノリシックカーネル OS と比較して性能低下を招きやすいという欠点がある．L4 は高速な IPC の実装によって OS サーバ間通信の高速化を実現しており，数あるマイクロカーネルの中でも特に性能面で優れているという特徴を持つ．

本研究では，L4 をベースとする 2 つの OS，L^{4}Linux および LP49[8] に対して省電力スケジューラを導入する．L^{4}Linux は，ドイツ・ドレスデン工科大学の DROPS プロジェクト，LP49 は日本・国立情報学研究所（NII）の分散 OS プロジェクト H_2O によって開発が進められているマイクロカーネル OS である．L^{4}Linux，LP49 はそれぞれ Fiasco，Pistachio と呼ばれる異なる L4 マイクロカーネルを用いて実装されている．両 OS の全体構成を**図 1** および**図 2** に示す．L^{4}Linux では，ユーザ空間にページャや割込み管理といった資源管理サーバを配置し，その上で Linux カーネルとしての機能を持つユーザタスクが Linux サーバとして動作する．ハードウェア制御や例外といった機能は，Linux サーバ内で L4 や資源管理サーバの提供する API を用いて仮想化される．また，LP49 では，ユーザ空間にページャなどの基本的な資源管理サーバを配置し，その上で Plan9 のカーネル機能を持つ Core プロセスと呼ばれるユーザタスクが動作する．さらにその上で，シェルやファイルサーバ，アプリケーションプログラムなどが個別の L4 タスクとして実装される．本研究では，Fiasco および Pistachio の両マイクロカーネル内のスケジューラに対して省電力機構を実装し，Linux

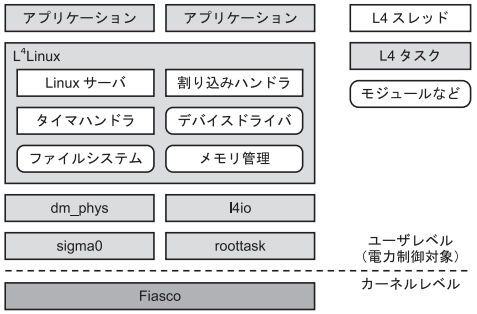

図 1　L^{4}Linux のシステム構成
Fig. 1　Structure of L^{4}Linux.

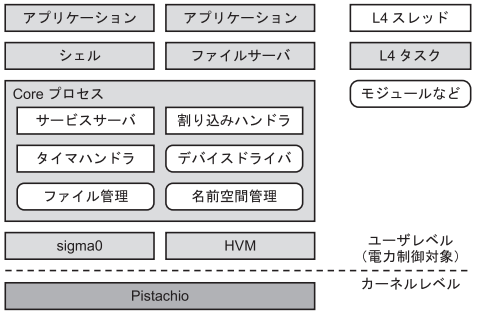

図 2　LP49 のシステム構成
Fig. 2　Structure of LP49.

サーバや Core プロセス，各種資源サーバといった各種 OS タスクとその内部スレッドに対しても，アプリケーションプログラムと同等に電力制御を行う．このとき，L^{4}Linux においてはカーネルとは別に Linux サーバ内にもユーザ定義のスケジューラが存在するが，L^{4}Linux 内に存在するすべてのプログラムはカーネルスケジューラによって管理されるために，ユーザ定義スケジューラに対して修正を行う必要性はない．

実装および評価には，Intel Pentium M プロセッサを使用する．本 CPU には，拡張版 Intel SpeedStep テクノロジ（EIST）が搭載されており，最大 9 段階の周波数調整が可能である．また，命令実行数や I/O，メモリアクセス，TLB ミスなどといった種々のイベントを同時に 2 種類計測できるパフォーマンスカウンタを備えており，タスク動作挙動の調査

に用いることができる．

3. 設　　計

本章では，L4 マイクロカーネルにおける省電力スケジューラのシステム設計について述べる．本研究における省電力化の具体的な手法は，先行研究[6] の提案する内容と同等である．ただし，実装対象である L4 のスケジューラでは，すべてのプログラムの実行単位は L4 スレッドとして管理されており，この L4 スレッドに対して電力制御を施すことにより，図 1 および図 2 に示すような OS サーバプログラムに対しても省電力化を実現できる点で，先行研究とはその制御対象範囲が大きく異なる．

3.1 性能予測モデルに基づく省電力化手法

DVFS 制御により CPU 省電力化を図る場合，電圧低下による消費電力の削減と周波数低下による性能への影響のトレードオフが問題になる．本スケジューラでは，保証されるべき最低限のプログラム性能をユーザがあらかじめ指定し，本制約を満たす範囲内で周波数・電圧低下による省電力化を行うという方針をとる．以下では，ユーザによって決定される性能制約を性能閾値と呼び，τ と表記する．

プログラムの実行時性能を考慮した省電力スケジューリングを行ううえで，何らかの形で性能を定量化する必要がある．一般的に，性能の指標として IPS（Instructions Per Second）が用いられることが多く，本研究においてもプログラムの実性能 p として IPS の値を利用する．プログラムの実行コードがループなどによってつねに一様であると仮定した場合，同一周波数 f 下での IPS の値はつねに一定であると予想される．ここで，選択可能な最高 CPU 周波数 $f_{\max}$ におけるプログラムの実行時性能を $p_{f_{\max}}$，実性能を p としたとき，性能閾値 τ を最高周波数時の性能に対して実性能が満たすべき性能比

$$\tau = \frac{p}{p_{f_{\max}}} \tag{1}$$

と定義する．また，プログラムが同一コードを実行するという前提下では，性能比は実行時間比の逆数と等価となる．すなわち，つねに最高周波数で動作したときのプログラムの実行時間を $t_{f_{\max}}$，実際の実行時間を t とすると，性能閾値 τ は

$$\tau = \frac{t_{f_{\max}}}{t} \tag{2}$$

と表される．たとえば，性能閾値が 50%である場合，つねに最高周波数で動作したときの半分の性能，すなわち 2 倍の実行時間でプログラムが終了するような制約の下で周波数・電圧調整を行えばよいことになる．

実際にプログラムの動作周波数を決定する際には，各周波数でプログラムを実行した場合に予測される性能が性能閾値を満たせるかどうかを判別する必要がある．本スケジューラでは，各周波数に対する予測性能の算出に下記のような線形回帰モデルを用いる．

$$\hat{y}_f = b_{0,f} + b_{1,f}x \tag{3}$$

ここで，説明変数 x を L4 スレッドの動作特性を表す実行時パラメータ，目的変数 $\hat{y}_f$ を周波数 f で実行したときに予想される最高周波数時に対する性能比とする．本スケジューラは，まず L4 スレッドの実行時情報 x を取得し，式 (3) より各周波数 f における予測性能比 $\hat{y}_f$ を計算する．次に，得られた $\hat{y}_f$ のうち，ユーザより定められた性能閾値 τ に対して

$$\hat{y}_f \geq \tau \tag{4}$$

を満たすような最低の f をスレッドの動作周波数として選択する．説明変数 x は，性能比 $\hat{y}_f$ に対して相関が強いパラメータであることが望ましい．先行研究[6),11)] では，L2 キャッシュミスヒット回数が性能に対して支配的であることを述べており，性能予測モデルの説明変数として 1 命令あたりの L2 キャッシュミス率を用いている．性能予測モデルを用いた DVFS 制御に関するこのほかの論文[10] においても，予測モデルにおける説明変数としてキャッシュミスを利用した例が存在する．そこで，本研究においても説明変数 x として L2 キャッシュミス率を利用するものとし，実行性能に影響するパラメータとしてスレッド単位でモニタリングを行う．実装・評価に用いる Pentium M プロセッサのパフォーマンスカウンタでは，実行命令数および L2 キャッシュミス回数の計測が可能であるため，本スケジューラではこれらの 2 つの動的情報を用いて IPS および L2 キャッシュミス率の算出を行うものとする．

L4 スレッドごとの動作周波数決定は，スレッド挙動，すなわち IPS およびキャッシュミス率の変化を検出するために，ある程度の頻度で実行する必要がある．本スケジューラでは，毎回のコンテキストスイッチ時に各スレッドの動作周波数を決定するものとする．したがって，DVFS 制御に関する主要な処理はいずれもコンテキストスイッチ時にのみ行われる．

3.2 学 習 機 構

3.1 節の性能予測モデルは，異なる動作特性を持つ複数のプログラムを実行し，実際に得られる統計情報をもとに回帰分析によって構築する必要がある．モデル構築の 1 つの手段として，事前に収集した統計情報を用いてあらかじめ回帰係数を静的に決定しておく方法がある．しかし，回帰係数の最適な値は異なるプラットホームやアーキテクチャの間で同一とは限らず，特定の環境下であらかじめ構築された予測モデルを他のシステム上で汎用的に利用することは好ましくない．学習機構では，ユーザが実行した複数のテストプログラム

に対して統計データを収集し，回帰分析を行うことで性能予測モデルを自動的に構築する．これにより，ユーザが独自の環境下で学習を行うことで，最適な性能予測モデルを構築できる機能を提供する．

学習機構の動作時には，コンテキストスイッチごとに学習対象の L4 スレッド実行時の式 (3) における説明変数と目的変数のデータを取得する．性能予測モデルは，周波数 f ごとに構築する必要があるため，コンテキストスイッチ時にはスレッドの動作周波数を 1 段階ずつ変更し，各周波数 f におけるデータ (x_f, y_f) を収集する．回帰係数 $b_{0,f}$，$b_{1,f}$ は，すべてのスレッドに対して得られたデータ (x_f, y_f) から最小二乗法により決定する．先行研究[6]においては，存在するすべてのプロセスに対して学習を行っていたが，本研究では，ユーザが学習対象となるスレッドを指定できるようにし，学習に最適な必要最低限のプログラムに対してのみ学習を行うものとする．

性能予測モデルの回帰係数をより正確に求めるために，学習には説明変数となる L2 キャッシュミス率の値が異なるような複数のプログラムを用いることが望ましい．これらのプログラムは，省電力時に実際に実行されるアプリケーションと必ずしも同一である必要はない．また，省電力化するアプリケーションの組合せごとに学習を行う必要はない．なぜなら，異なるキャッシュミス率を持つ複数のプログラムセットに対して学習を実行すると，学習データに基づいて性能予測モデル (3) の係数が決定され，得られた予測モデルを用いて別のアプリケーションに対しても性能を予測できるためである．したがって，あらかじめ用意した適当なプログラムセットに対して学習を実行するだけで，様々なアプリケーションに対して性能予測に基づく省電力化が可能となる．

3.3　フィードバック機構

3.1 節における性能予測モデル (3) は，L2 キャッシュミス率が性能に大きく影響することを前提として構築される．しかし，実際のプログラム性能はこのほかの様々な要因に大きく依存する場合も考えられ，キャッシュミス率のみがつねに性能に対して支配的であるとは限らない．たとえば，I/O バウンドなプログラムにおいては，全体の実行時間に占める I/O 待ちの割合が大きくなるため，I/O に関する情報はキャッシュミス率などと比べて性能により相関の強いパラメータとなることが予想される．したがって，キャッシュミス率などの単一のパラメータのみを用いて多様な動作挙動を示す種々のプログラムの実行時性能を正確に予測することは困難であり，結果として性能見積り誤差による消費電力の浪費または過度な時間超過を招く恐れがある．

そこで，キャッシュミス率以外の要因が性能に大きく影響するようなプログラムが実行された場合に，必要に応じて性能予測における誤差を補正するフィードバック機構を実装する．フィードバック機構では，理想性能と実性能の誤差を自動的に検出し，これを補正するように動作周波数を調節することで，実性能のさらなる最適化を行う．ここでいう理想性能とは，プログラムを最高周波数で実行したときの性能 $p_{f_{\max}}$ に対する比が性能閾値 τ と等しくなるような性能 p_{ideal} を指す．すなわち，

$$p_{\text{ideal}} = \tau p_{f_{\max}} \tag{5}$$

となる．まず，理想性能 p_{ideal} の算出に必要な $p_{f_{\max}}$ を計測する．このため，フィードバック適用時には各スレッドは最低 1 回のタイムスライスを最高周波数で動作する必要がある．次に，コンテキストスイッチごとに式 (5) より p_{ideal} を計算し，実性能 p_{real} との誤差 p_{diff} を算出する．ここで，

$$p_{\text{diff}} = p_{\text{real}} - p_{\text{ideal}} \tag{6}$$

とする．最終的に，毎回のコンテキストスイッチにおける p_{diff} の総和 $\sum p_{\text{diff}}$ を 0 に近づけるように周波数を 1 段階ずつ変更する．今，$\sigma = \sum p_{\text{diff}}$ とすると，σ が条件式

$$\sigma_{\min} \leq \sigma \leq \sigma_{\max} \tag{7}$$

を満たす場合，実性能と理想性能に大きな誤差はないものとして周波数調整を行わない．なお，$\sigma_{\min}$ および $\sigma_{\max}$ はそれぞれ σ の 0 に対する許容誤差の最小値，最大値とする．

$$\sigma < \sigma_{\min} \tag{8}$$

となるとき，実性能が理想性能を下回ることから性能制約が満たされていないと判断し，動作周波数を 1 段階上げる．

$$\sigma > \sigma_{\max} \tag{9}$$

となるとき，理想性能を上回るパフォーマンスにより過度な電力消費が行われていると見なし，さらなる省電力化のために周波数を 1 段階下げる．以上の周波数調整によって，性能予測が外れるようなケースにおいても実性能の補正による適切な省電力化が可能となる．

上記のフィードバック制御は，式 (5) および式 (6) から分かるように，最高周波数時に計測した性能 $p_{f_{\max}}$ を基準として行われる．ここで，プログラム実行中に $p_{f_{\max}}$ の値が大きく変化した場合，過去に計測した $p_{f_{\max}}$ の値を用いてフィードバック制御を継続すると適切な周波数選択ができなくなる可能性がある．そこで，フィードバック処理中にはスレッドの実行状態をキャッシュミス率や IPS の値としてつねに監視し，これらが大きく変化した場合には 1 度 $p_{f_{\max}}$ を再計測してから制御を再開するという手法をとる．

3.4　全 体 構 成

システムの全体構成を**図 3** に示す．省電力スケジューラの動作モードとして，3.2 節の学

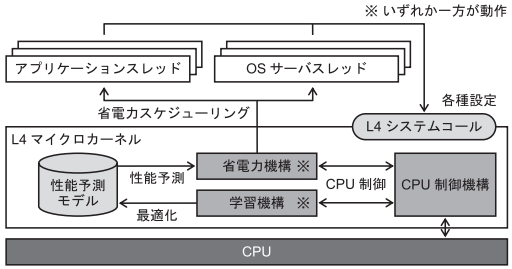

図 3　省電力スケジューラの全体構成
Fig. 3　Structure of power-saving scheduler on L4.

習機構が動作する学習モードと，3.1 節の手法に基づいて省電力化を行う省電力モードの 2 つを実装する．実際にどのモードで動作するかは，ユーザが自由に選択できる．

学習モードでは，3.2 節の内容に従って性能予測モデルの自動構築を行う．このとき，ユーザは学習に用いるプログラムをスケジューラに通知し，実際にプログラムを 1 回ずつ実行するだけで，システム環境に応じた最適な予測モデルを構築できる．

省電力モードでは，学習によって得られた性能予測モデルを利用し，3.1 節の手法に基づいて L4 スレッドごとの DVFS 制御を行う．また，必要に応じてフィードバック機構を適用し，キャッシュミス率以外の要因が性能に大きく影響するような場合にも，実性能の補正により最適な動作周波数を選択する．このとき，スケジューラはすべてのアプリケーションスレッドならびに OS サーバスレッドをその種類に関係なく独立した対等の L4 スレッドとして管理し，各々の事前の動作情報のみに基づいて個別に最適周波数を決定する．このため，通常複数のアプリケーションと依存関係のある OS 処理に対して，アプリケーションに独立な OS 処理専用のスレッドという形で省電力スケジューリングを行うことが可能となり，結果としてアプリケーションスレッドと OS サーバスレッドを独立に扱うことによる OS 処理を細分化した電力制御が実現できる．

CPU 周波数・電圧の設定や，スレッド実行時のパフォーマンスカウンタの値の取得といった処理は，アーキテクチャごとに設計・実装される CPU 制御機構を介して行う．また，本スケジューラの機能をユーザが利用する際，動作モードの選択や学習プログラムの指定，性能閾値の設定やフィードバック適用の有無といった何らかの要求をシステムに対して通知する必要がある．このためのインタフェースとして，L4 内にスケジューラ制御用のシステムコールを新たに作成し，ユーザプログラムはこれを利用することで本スケジューラに対して各種設定を行う．

4. 実　　装

3 章の設計内容に従い，L4 マイクロカーネル内に省電力スケジューラを実装した．実装対象として用いた Fiasco および Pistachio は，細かい実装方法や設計理念の相違はあるものの，L4 としての基本原理は同じである．ソースコードはいずれも C++で記述されており，カーネル内の多くの OS コンポーネントがクラスとして抽象化されている．本研究では，図 3 に示されるような電力制御に関する機構を 1 つのクラスとして L4 内に新たに定義し，スケジューラなどの他の既存処理から本クラスの提供する電力制御メソッドを呼び出す形で実装を行った．以下では，本クラスを DVFS クラスと呼ぶ．

学習機構および省電力機構は，DVFS クラスの主要メソッドとして実装される．両機構による処理はいずれもコンテキストスイッチ時にのみ行われるため，L4 スケジューラのコンテキストスイッチコードから DVFS クラスのハンドラメソッドを呼び出し，モードに応じた処理を行う形で実装した．電力制御においてスレッドごとに必要となるデータについては，DVFS クラスをスレッドの実体となるクラスのメンバとすることで保持した．また，CPU 制御機構はアーキテクチャ依存コードとして隔離して実装することによりシステムの移植性を高めた．

DVFS クラスのほかに，ユーザが省電力スケジューラに対して各種設定を行うためのシステムコールインタフェースの作成を行った．L4 に本システム制御用のシステムコールエントリを新たに追加し，そこから DVFS クラスのハンドラメソッドを実行する形で実装した．また，ユーザが本システムを効率的に操作できるようにするためのシステムコールライブラリを作成した．主要な API を**表 1** に記す．以上の電力制御に関する既存コードへの修正は，L4 外部の OS サーバに対しては基本的に不要であり，結果的に Fiasco および Pistachio をベースとするあらゆるシステムに対して本スケジューラを汎用的に適用可能である．

実装した省電力スケジューラに対して，コンテキストスイッチ時の追加コード実行によるオーバヘッドの測定を行った．スケジューリング処理全体（`schedule`）のクロックサイクル数に占める追加コードの割合と，CPU・メモリベンチマークおよび I/O ベンチマークの実行時間に占める追加コードの割合の最大値を**表 2** に記す．ここで，CPU・メモリベンチマーク，I/O ベンチマークにはそれぞれ 5 章の評価で用いている `matrix`，`disk` を使用した．なお，`matrix` は，1,500 × 1,500 の 2 つの `int` 型行列の積を求めるプログラム，`disk` は `/dev` 以下の HDD デバイスファイルに対し `fread` により合計 100 MB のデータを 1 MB

表 1　システムコールライブラリの関数例
Table 1　Examples of system call functions.

dvfs_set_mode	動作モードを指定する．
dvfs_feedback_on	フィードバックを有効にする．
dvfs_feedback_off	フィードバックを無効にする．
dvfs_learning_on	対象スレッドの学習を有効にする．
dvfs_learning_off	対象スレッドの学習を無効にする．
dvfs_set_thd	対象スレッドの性能閾値を設定する．

表 2　コンテキストスイッチのオーバヘッド
Table 2　Overheads of L4 context switch.

	L^4Linux	LP49
クロック数（schedule）	2,187	4,747
クロック数（追加部分）	1,555 (71%)	1,424 (30%)
実行時間比率（CPU ベンチ）	0.16%	0.05%
実行時間比率（I/O ベンチ）	0.85%	0.12%

ずつ読み取るプログラムである．表 2 に示すように，CPU・メモリベンチマーク実行時におけるオーバヘッドは両 OS で 0.2%未満と十分に小さい．I/O プログラム実行時には，マイクロカーネル OS の構造上 OS サーバ間で IPC が頻繁に発生し，コンテキストスイッチ回数が増加するため，結果的にオーバヘッドが CPU・メモリベンチマークに比べて大きくなるが，それでも実行時間に占める比率は 1%に満たない．また，表 2 の実行時間比率に関する内容はあくまで全テストケース中の最悪値であり，他の多くの場合ではさらに小さいオーバヘッドとなるため，本システムの実装による致命的な遅延は発生していないと考えることができる．

5. 評　　価

省電力スケジューラを実装した L^4Linux および LP49 に対し，実機上での電力測定によるシステム評価を行った．CPU は Intel Pentium M 760 を使用し，0.8〜2.0 GHz の周波数と各々で利用可能な最小電圧の組からなる計 9 段階の動作レベルを用いた．ベンチマークプログラムには，行列演算 matrix，整数演算 Dhrystone，MiBench より高速フーリエ変換 FFT とハッシュ計算 SHA，およびディスク読み込みプログラム disk を用いた．また，本システムの適用性を明確にするため，TAR アーカイバや SQL サーバ，AAC エンコーダといったより実用的なアプリケーションに対する電力評価を行った．

表 3　学習時間 [s]
Table 3　Learning time [s].

matrix	29.20
Dhrystone	36.67
FFT	19.37
SHA	25.38
合計	110.62

表 4　性能予測式 (3) における回帰係数の学習結果
Table 4　Obtained regression coefficients.

f [GHz]	L^4Linux		LP49	
	$b_{0,f}$	$b_{1,f}$	$b_{0,f}$	$b_{1,f}$
0.79	0.40	43.35	0.39	31.61
1.06	0.53	36.80	0.53	29.02
1.20	0.60	32.14	0.59	26.27
1.33	0.67	28.47	0.66	22.02
1.46	0.74	23.22	0.73	17.47
1.60	0.80	17.92	0.80	13.64
1.73	0.87	10.54	0.86	9.28
1.86	0.93	4.31	0.93	4.64
2.00	1.00	0.00	1.00	0.00

まず，キャッシュミス率の異なる 4 つの CPU・メモリバウンドなプログラム matrix，Dhrystone，FFT，SHA を学習モードで実行し，性能予測モデル (3) の構築を行った．学習に要した時間の詳細を**表 3** に，学習の結果得られた回帰係数 $b_{0,f}$ および $b_{1,f}$ の値を**表 4** にそれぞれ記す．ここで，回帰係数 $b_{0,f}$ はキャッシュミス率が 0 に近いようなプログラムを実行したときの周波数 f ごとの予測性能比，$b_{1,f}$ はキャッシュミス率の増加に対する予測性能比の増加率となる．$b_{0,f}$ については，両 OS 間で同等の結果となったが，$b_{1,f}$ については低周波数のモデルほど L^4Linux の方が値が大きくなる結果となった．これは，キャッシュミス発生時に要するオーバヘッドが L^4Linux の方が長く，キャッシュミス率の高いプログラムを実行した際の周波数低下による性能への影響が LP49 に比べて小さいためであると考えられる．この結果から分かるように，性能予測モデルは OS やハードウェアプラットホームなどのプログラム実行環境によって変化する可能性がある．したがって，システム環境に適合する性能予測モデルを学習により動的に構築することは，最適な電力制御を行ううえで重要であるといえる．

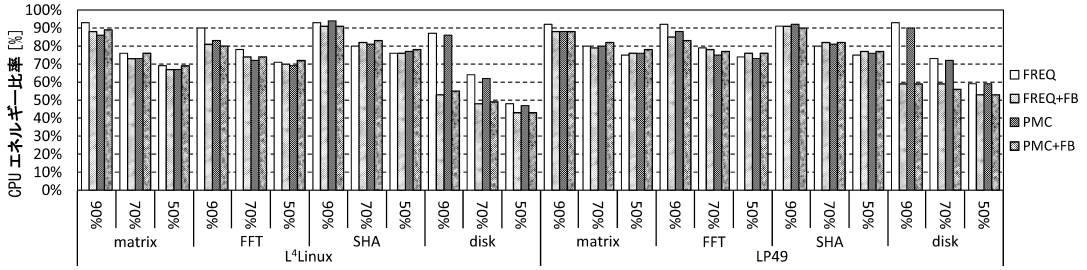

図 4　ベンチマークごとの CPU エネルギー消費量の比率
Fig. 4　CPU energy ratios of the highest frequency of each benchmark.

表 5　省電力化方式
Table 5　Power-saving methods.

方式名	性能予測モデル	フィードバック
FREQ	式 (10)	無効
FREQ+FB	式 (10)	有効
PMC	式 (3)	無効
PMC+FB	式 (3)	有効

次に，表 4 の回帰係数からなる性能予測モデルを用いて省電力機構を動作させ，ベンチマーク実行時の電力評価を行った．本研究の提案する性能予測モデルの優位性を確認するため，比較対象として，性能比が純粋に周波数比となるような単純な予測モデル

$$\hat{y}_f = \frac{f}{f_{\max}} \tag{10}$$

を用いた方式を実装した．評価では，式 (10) および式 (3) の 2 つの性能予測モデルとフィードバックの有無を組み合わせた**表 5** に示す 4 つの方式を用いて，ベンチマーク実行に要する CPU エネルギー消費量の比較を行った．CPU エネルギー消費量の値は，ベンチマーク実行時における HDD などの外部装置を除いたマザーボード全体の平均電力 P_{mb} から CPU 以外の平均電力 P_{ecpu} を差し引いた値と，ベンチマークの実行時間 t の積として算出した．すなわち，CPU エネルギー消費量 E_{cpu} は

$$E_{\mathrm{cpu}} = t(P_{\mathrm{mb}} - P_{\mathrm{ecpu}}) \tag{11}$$

となる．P_{mb} の値は，ホール素子による非接触型の電流計測システムにより実測した．ここで，P_{ecpu} の値はデータシート[5] に示される電力理論値に対して，同環境下で測定したシステム電力との差分として算出した．

`matrix`，`FFT`，`SHA` および `disk` を実行したときの CPU エネルギー消費量の測定結果を**図 4** に示す．結果は，性能閾値を 90%，70%，50%に設定した際の各方式におけるエネルギー比率を，最高周波数時を 100%とした百分率で示してある．また，性能閾値に対する実性能の誤差を測定した結果を**図 5** に示す．ここで，性能閾値 τ における実行時間 t_τ の性能誤差率 e_τ は

$$e_\tau = \frac{t_{100\%}}{t_\tau} - \tau \tag{12}$$

として算出している．つまり，実性能が性能閾値以上であれば e_τ は正となり，以下であれば負となる．

5.1　CPU・メモリベンチマークに対する評価

`matrix` や `FFT` といったキャッシュミス率の高いベンチマークプログラムについては，FREQ 方式に比べ PMC 方式の方がより高い省電力効果が得られている．また，性能閾値に対する実行時間の超過率となる負方向への性能誤差は最大で −5%程度であり，いずれの方式においても性能制約を大きくオーバすることなく省電力化を実現できている．したがって，キャッシュミス率などの性能に支配的な情報を説明変数とする性能予測モデルを用いた DVFS 制御方式を適用することで，L4 マイクロカーネルにおいてもより効果的な CPU 省電力化を実現できる可能性が示された．一方で，`SHA` などのキャッシュミス率の低いプログラムについては，2 つのモデル式 (3) および (10) による予測値が同等となるため，FREQ 方式と PMC 方式の間で大きな差はないという結果となった．

先行研究[6] における Linux 上での電力評価と比較した場合でも，CPU・メモリベンチ

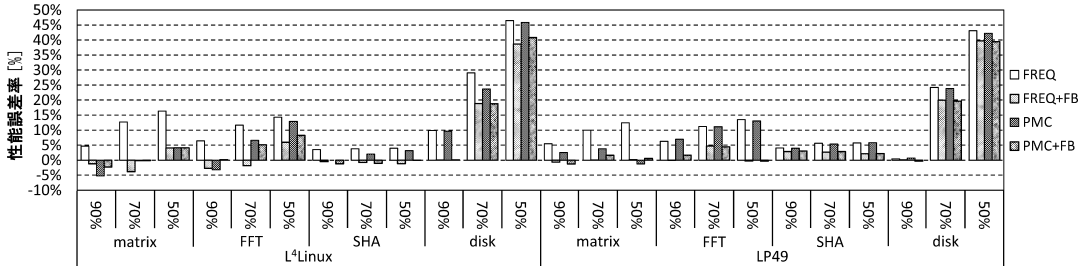

図 5　ベンチマークごとの性能誤差率
Fig. 5　Error ratios of real performances of each benchmark.

マークに関しては同様の結果が得られている．CPU・メモリバウンドなタスクの実行については，OS 構造に依存する要素がきわめて少ないため，マイクロカーネル OS においても Linux と同様の電力挙動となることは容易に予想できる．本評価は，これを明確にした結果であるといえる．

5.2　I/O ベンチマークに対する評価

`disk` では，L^{4}Linux および LP49 の双方において，フィードバック適用時に FREQ 方式や PMC 方式に比べ高い省エネルギー効果が得られている．特に，L^{4}Linux において性能閾値を 90%に設定した場合では，FREQ 方式に比べ 32%のエネルギー削減を実現している．また，性能閾値を 70%，50%に設定した際に性能誤差率が正に大きくなっているが，これは `disk` の実行時間が性能閾値によらずほぼ一定なために，性能閾値が低くなるほど理想性能に対する実性能の誤差が増加していることを示している．結果的には，性能を低下させることなく，フィードバック制御による大きなエネルギー削減を実現していることになる．

`disk` 実行時には，ディスクアクセス要求や割込みによって多くの OS サーバが頻繁に動作することになる．ディスクデータの読み取りに関する OS 内部の処理は，L^{4}Linux と LP49 の間で細かい実装方法は異なるもののその基本的な構造・手順は同様である．L4 においては，すべての割込みハンドラはユーザレベルの IRQ スレッドとして実装され，カーネルからの割込み通知メッセージ受信することでその処理を実行する．L^{4}Linux においては Linux サーバ内に，LP49 においては Core プロセス内にそれぞれ最終的な割込み処理を実行する IRQ スレッドが実装されている．`disk` 実行時には，プログラム全体の実行時間に対する IRQ スレッドの割合が，L^{4}Linux では 91.6%，LP49 では 87.6%とともに 9 割近くを占めていることが分かった．

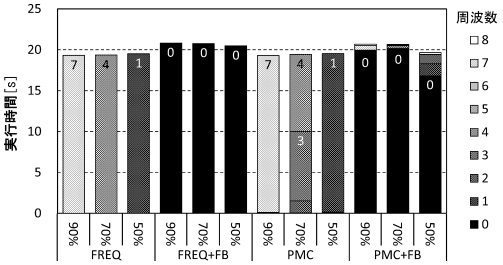

図 6　`disk` 実行時の IRQ スレッドの実行時間内訳（L^{4}Linux）
Fig. 6　Execution time breakdown of IRQ handler thread for the execution of `disk` (L^{4}Linux).

これらの IRQ スレッドは，割込み処理の中でハードディスクに対してデータの入力命令を発行するため，CPU 周波数に影響しない I/O レイテンシに大半の実行時間を消費する．このため，IRQ スレッドの実性能比は，CPU 周波数にかかわらずつねに 100%に近い状態となる．フィードバック機構では，実性能が理想性能よりも高い場合，式 (9) より周波数を下げ続けるため，性能閾値が 100%より低い今回のケースでは，IRQ スレッドはほぼすべての実行時間を最低周波数で動作することになる．**図 6** に，L^{4}Linux における IRQ スレッドの周波数ごとの実行時間内訳を示す．ここで，最高周波数を 8（2.0 GHz），最低周波数を 0（0.8 GHz）とする．FREQ+FB および PMC+FB 方式においては，性能閾値によらず実行時間のほぼすべてを最低周波数で動作していることが確認できる．また，L^{4}Linux 上で `disk` 実行時に動作する主なスレッドの CPU エネルギー消費量を**表 6** に記す．なお，表内の合計値は `disk` 実行時に消費される全エネルギーとほぼ同等である．このうち大部分のエ

表 6　disk 実行時のスレッドごとの CPU エネルギー消費量 [J]（L⁴Linux）
Table 6　CPU Energy of each thread for the execution of disk [J] (L⁴Linux).

スレッド	FREQ			FREQ+FB			PMC			PMC+FB		
	90%	70%	50%	90%	70%	50%	90%	70%	50%	90%	70%	50%
IRQ スレッド	317.6	227.9	168.1	140.3	140.2	137.7	314.7	216.1	168.1	150.0	143.4	141.7
Linux サーバ	2.0	1.5	1.2	1.8	1.4	1.8	1.4	1.3	1.2	2.0	1.8	1.7
割込み管理サーバ	17.7	13.9	11.8	10.3	9.6	10.5	17.6	13.3	11.9	10.6	9.8	10.6
カーネルアイドル	0.4	0.1	0.1	0.3	0.3	0.3	0.4	0.2	0.2	0.3	0.3	0.3
合計	337.7	243.4	181.3	152.8	151.5	150.2	334.1	230.9	181.4	162.9	155.2	154.2

ネルギーを IRQ スレッドが消費しており，フィードバック適用時にはこれを大幅に削減できていることが分かる．上記の結果は，マイクロカーネル構成により細分化された OS 処理から高レイテンシな要素を検出し，これに対して最適な動作周波数の選択が行えていることを意味する．結果的に，フィードバックを適用することによって，性能低下を招くことなく最大限の CPU 省エネルギー化を実現できたことになる．本成果は，OS サーバを個別に省電力スケジューリングすることで OS 処理の電力最適化を実現した結果であり，本システムのマイクロカーネルへの有用性を示す結果であるといえる．

5.3　Linux における電力評価との比較

先行研究[6] による Linux 上での disk の評価では，本研究と同様，フィードバック適用時に高い省エネルギー効果を得ている．disk 実行時には HDD アクセスが頻繁に発生するため，その動作挙動は OS 構造に大きく依存する．本節では，L⁴Linux と Linux のそれぞれにおける disk 実行時の電力挙動を比較し，省電力スケジューラの両 OS 構造に対する適用効果の相違を確認する．

図 7 に，disk を単一で実行（disk×1），2 つを同時実行（disk×2），SHA と同時実行（disk+SHA）した際の CPU エネルギー消費率を示す．実験では，L⁴Linux と Linux の両 OS を同一ハードウェア環境上で動作させ，同一の計測方法により評価した．なお，5.2 節の評価と同様，周波数低下に対する disk の実行時間の変化量が小さいために，すべてのケースで性能誤差率は性能閾値が低いほど正に大きくなる結果となった．disk のみを単一あるいは複数同時に実行した場合には，L⁴Linux と Linux のいずれもフィードバックが有効に働き，両 OS 間で同程度の省電力効果が得られていることが確認できた．L⁴Linux では，disk を 2 つ同時実行にした際に IRQ スレッドが複数の disk アプリケーションから共有される形になるが，このような場合においても単一の IRQ スレッドを適切に電力制御することで OS 処理の効果的な省電力化が実現できていることが分かる．

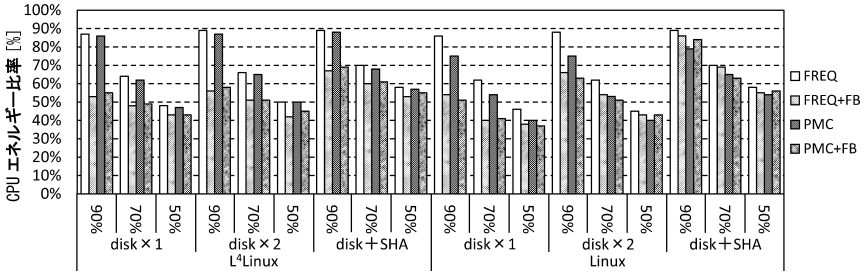

図 7　disk 実行に関する CPU エネルギー消費率の比較
Fig. 7　Comparison of CPU energy ratios about disk.

一方で，disk と SHA を同時実行した場合，L⁴Linux では disk 単体実行時と同様にフィードバック効果が高いのに対し，Linux においてはフィードバック適用によるエネルギー削減率が低い結果となっている．これは，HDD 割込みに対するハンドリング処理の実装が OS 間で異なることに起因する．L⁴Linux においては，5.2 節で述べたように，HDD 割込みに対応する IRQ スレッドがカーネルから割込み通知メッセージを受信することで，1 スレッドコンテキストとして割込み処理を実行する．これに対し Linux では，割込み発生時に動作中のプロセスがカーネルモードで割込みハンドラを実行するため，スケジューラは割込み処理を独立した 1 つのプロセスコンテキストとして扱うことができない．

図 8 および **図 9** は，それぞれ L⁴Linux，Linux において disk+SHA 実行時に動作するスレッド・プロセスの実行時間内訳を示している．図 8 から分かるように，L⁴Linux における IRQ スレッドの実行時間はほぼ一定であり，また CPU ネックなベンチマークである SHA は性能閾値の低下にともなって実行時間が単調増加している．この IRQ スレッドと SHA スレッドの動作挙動は，disk ベンチマークと SHA ベンチマークをそれぞれ個別に実験したと

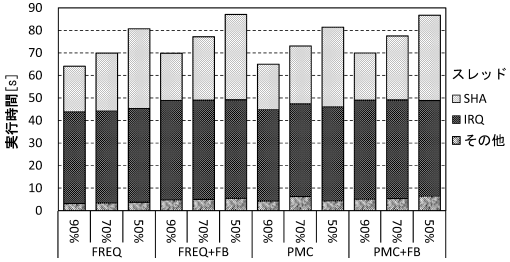

図 8　disk+SHA 実行時のスレッドごとの実行時間内訳（L^{4}Linux）
Fig. 8　Execution time breakdown of each thread for the execution of disk+SHA (L^{4}Linux).

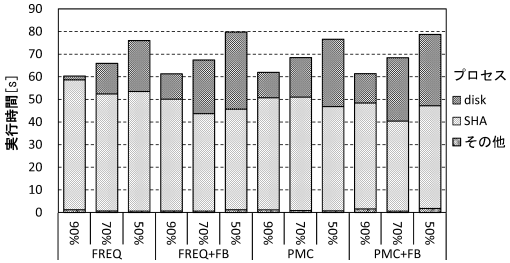

図 9　disk+SHA 実行時のプロセスごとの実行時間内訳（Linux）
Fig. 9　Execution time breakdown of each process for the execution of disk+SHA (Linux).

きと同等の実行時間・周波数となっており，これらのベンチマークを並行実行した今回の実験においても各スレッドに対して適切な電力制御が行えていることが分かる．

一方，図 9 では，L^{4}Linux と同様の挙動を示すはずの SHA の実行時間が性能閾値に関係なく不規則に変化している．また，I/O システムコールを連続して発行するのみで，本来周波数低下の影響が小さいはずの disk プロセスの実行時間が性能閾値低下にともなって増加している．これらの結果から，Linux においては，割込みハンドラによる I/O 処理が SHA や disk のコンテキストの一部として実行されており，スケジューラは異なる動作特性をあわせ持つコンテキストを単一プロセスとして扱いながら電力制御を行っていることが分かる．3.3 節で述べたように，フィードバックではプロセス挙動の変化を検出した場合に最高周波数へと移行するため，割込み処理と本来の処理が交互に現れる SHA などのプロセスを頻繁に高周波数で動作させ，結果的に全体のエネルギー削減率が低下したと考えられる．

以上から，disk+SHA のようにアプリケーション処理と OS 処理が混在する実行環境においては，これらを不本意に統合制御する可能性のあるモノリシックカーネル OS に比べ，各々を個別に電力制御可能なマイクロカーネル OS の方が，より高い省電力効果が期待できるといえる．

5.4　実用アプリケーションを用いた評価

本研究の提案する省電力スケジューラの実システム上での使用における適用範囲を明確にするため，より実用的なアプリケーションを用いた評価を行った．実験では，下記に示す 3 つのアプリケーションを L^{4}Linux へ移植し，前節までと同様の方法で CPU エネルギー消費率および性能誤差率の測定を行った．

- TAR
  ファイル数 2,520 個，合計サイズ 62 MB からなる LP49 のソースコードを tar コマンドにより TAR 形式に圧縮する．
- SQL
  ローカルの MySQL サーバ上にあらかじめ作成した 280 MB のデータベースに対し，1 万件の検索処理を発行する．
- AAC
  再生時間 1 分 38 秒，ビットレート 1,411 Kbps，ファイルサイズ 16.5 MB の WAV 形式の音声データを，FAAC エンコーダを用いて AAC 形式に変換する．

結果を**図 10** および **図 11** に示す．ファイル操作やデータアクセスによってディスク入出力が多発する TAR や SQL などのアプリケーションでは，特に性能閾値が高い場合にフィードバックが有効に働き，大幅に消費エネルギーを削減できていることが分かる．性能閾値 90%においては，TAR では PMC+FB で 34%，SQL では FREQ+FB で 18%のエネルギー削減を実現しており，本システムの有用性を示す結果であるといえる．性能誤差に関しては，disk ベンチマークと同様の理由で正に増加しており，性能の低下なく省電力化を行えていることが分かる．一方で，アプリケーション実行中の CPU 負荷が比較的高い AAC では，各方式間でエネルギー消費はほぼ同等となった．

結論として，本システムを用いることで高い省電力効果が得られる実用アプリケーションは，フィードバックを有効にした TAR および SQL となる．TAR に示されるようなファイル圧縮や SQL のようなデータベース処理などといったデータアクセス頻度の高いアプリケーションに対する本システムの適応性は高く，少ない性能低下で高い CPU 省電力効果を得ることができる．反面，AAC などを代表とする音声や動画のエンコード・デコードといった高

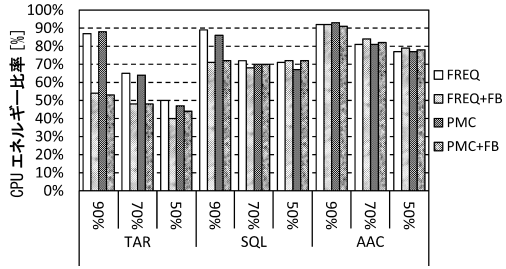

図 10　アプリケーションごとの CPU エネルギー消費量の比率
Fig. 10　CPU energy ratios of the highest frequency of each application.

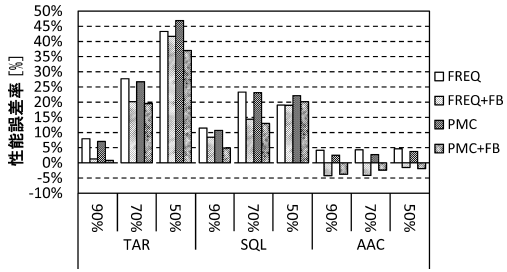

図 11　アプリケーションごとの性能誤差率
Fig. 11　Error ratios of real performances of each application.

い CPU パワーを必要とするアプリケーションに対しては，本システムを用いることによる大きな省電力効果は期待できない．これは，CPU 負荷の高いプログラムの性能が純粋に周波数に比例するため，式 (10) のような単純な予測モデルを用いた場合にも最適な周波数決定が行えてしまうことが原因である．また，5.1 節や先行研究[6] の評価結果にもあるように，`matrix` や `FFT` といったメモリバウンドなベンチマークでは，PMC 方式，すなわち学習を用いることによって FREQ 方式よりも高い省電力効果が得られる場合がある．特に，頻繁に挙動が変化するようなフィードバックに適さないアプリケーションに対しては，学習がより有効に働くような場合も十分に考えられる．このため，本システムによる省電力化をより効果的に行うためには，それぞれの適応範囲に応じて省電力方式を使い分けることが重要と考える．

6. 関連研究

Linux における CPUFreq や，CPU 周波数・電圧変更についての以前の研究[3] では，設定された電力ポリシに従った CPU 負荷に基づく電力制御を行っていた．しかし，5 章の結果が示すように実際のプログラムの動作挙動は多種多様であり，これを区別せず CPU の利用状況のみをもとに DVFS 制御を行う場合，ワークロードの変化に迅速に対応する細粒度な省電力化を実現できない．

プログラム実行性能に制約のあるようなシステムにおける DVFS 制御の例[9),13)] では，周波数比が直接性能比となるものとして性能を予測していた．これと同様の予測手法である 5 章の FREQ 方式では，本研究の提案手法である PMC 方式と比較して，キャッシュミス率の高いプログラムに対して省電力効果が低い結果となった．本研究の手法では，キャッシュミス率を考慮した性能予測に加えフィードバックによる実性能の誤差補正によって，周波数と性能の依存性におけるメモリ・I/O レイテンシの影響を検出し，結果的により高い省電力効果を実現している．

本研究と同様に，メモリアクセス率を説明変数とした回帰モデルを利用して性能や実行時間の予測を行う研究[7),10)] も存在する．しかし，いずれの手法も，予測における I/O などのメモリアクセス以外の要因に対する考慮がなされておらず，また，個々の OS 処理を細分化して制御することも不可能である．これに対して本研究では，学習機構によって性能予測モデルの動的な構築が可能であるほか，フィードバック機構を適用することによって，I/O などが性能に大きく影響するような場合においても動作周波数の最適化を行うことができる．さらに，OS サーバごとの電力制御が可能であるため，I/O 実行やシステムコール発生時に頻繁に動作する OS 内部処理に対しても細粒度な電力制御が可能である．

7. おわりに

本論文では，L4 マイクロカーネルを対象とした DVFS 省電力スケジューラの設計と実装，および評価について述べた．マイクロカーネル OS の内部構成に着目することで，本スケジューラでは，アプリケーションタスクのみでなく各種 OS サーバに対しても同等に省電力スケジューリングを行い，OS 処理に対する細粒度な電力制御を実現している．評価では，特に I/O ベンチマーク実行時にフィードバック機構による顕著な省エネルギー効果が得られ，従来の単純な性能予測に基づく省電力手法と比較して最大で 34%の CPU エネルギー削減を実現した．また，評価結果の原因について分析することでマイクロカーネルに対

する本システムの有用性を示し，OS 処理の細分化の重要性について述べることによりマイクロカーネル構成における省電力化の可能性について考察した．

今後の課題としては，まず性能予測モデルの改善があげられる．本研究においては，L2 キャッシュミス率のみを用いて性能予測を行っていたが，このほかにも I/O 情報などの複数のパラメータからなる重回帰モデルを構築し，より高精度な性能予測を実現していくことなどが考えられる．また，展望として，組み込み分野などへの適用を想定し，リアルタイムシステム性を考慮した省電力スケジューリングの設計・実装を検討している．現システムにおいては，プログラムがユーザによって定められた性能閾値を確実に満たすことが保証されていない．リアルタイムシステムへの本スケジューラの導入を考えたとき，性能閾値や予測モデルといった種々の概念をどのような形でデッドライン保証や実行時間予測といった理論に適用するかという問題がある．これらの内容について検討し，本システムをより実用的な省電力技術へと発展させることは，今後の大きな研究課題の 1 つである．

参 考 文 献

1) Au, A. and Heiser, G.: L4 User Manual (1999). http://l4hq.org/docs/manuals/l4uman.pdf
2) Choi, K., Soma, R. and Pedram, M.: Off-Chip Latency-Driven Dynamic Voltage and Frequency Scaling for an MPEG Decoding, *Proc. 41th IEEE/ACM Design Automation Conference*, pp.544–549 (2004).
3) Govil, K., Chan, E. and Wasserman, H.: Comparing Algorithms for Dynamic Speed-Setting of a Low-Power CPU, Technical Report TR-95-017, ICSI (1995).
4) Intel: Enhanced Intel SpeedStep Technology for the Intel Pentium M Processor (2004). http://www.intel.com/design/intarch/papers/30117401.pdf
5) Intel: Intel Pentium M Processor with 2-MB L2 Cache and 533-MHz Front Side Bus (2005). http://download.intel.com/design/mobile/datashts/30526202.pdf
6) 金井　遵，佐々木広，近藤正章，中村　宏，天野英晴，宇佐美公良，並木美太郎：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，情報処理学会論文誌：コンピューティングシステム，Vol.49, No.SIG2(ACS21), pp.20–36 (2008).
7) Lee, S.-J., Lee, H.-K. and Yew, P.-C.: Runtime Performance Projection Model for Dynamic Power Management, *Proc. 12th Asia-Pacific Computer Systems Architecture Conference*, Vol.4697, pp.186–197 (2007).
8) 丸山勝巳，佐藤好秀：連携処理のためのコンポーネント型 OS LP49，情報処理学会研究報告，2008-OS-108, Vol.2008, No.35, pp.123–130 (2008).
9) Pillai, P. and Shin, K.G.: Real-Time Dynamic Voltage Scaling for Low-Power Embedded Operating Systems, *Proc. 18th ACM Symposium on Operating Systems Principles*, pp.89–102 (2001).
10) Poellabauer, C., Singleton, L. and Schwan, K.: Feedback-Based Dynamic Voltage and Frequency Scaling for Memory-Bound Real-Time Applications, *Proc. 11th IEEE Real Time and Embedded Technology and Applications Symposium*, pp.234–243 (2005).
11) 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).
12) 関　直臣，Zhao, L.，徐　慧，池淵大輔，小島　悠，長谷川揚平，天野英晴，香嶋俊裕，武田清大，白井利明，中田光貴，宇佐美公良，砂田徹也，金井　遵，並木美太郎，近藤正章，中村　宏：MIPS R3000 プロセッサにおける細粒度動的スリープ制御の実装と評価，情報処理学会研究報告，2008-ARC-176, Vol.2008, No.1, pp.71–76 (2008).
13) 嶋田　創，安藤秀樹，島田俊夫：パイプラインステージ統合と DVS の併用による消費電力の削減，情報処理学会論文誌：コンピューティングシステム，Vol.48, No.SIG3(ACS17), pp.75–87 (2007).

(平成 20 年 7 月 23 日受付)
(平成 20 年 11 月 29 日採録)

林　　和宏（学生会員）

2008 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．現在，同大学大学院工学府情報工学専攻博士前期課程在学中．

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．並列分散処理，低消費電力化，シンクライアント等に関するシステムソフトウェアの研究に従事．

丸山　勝巳（正会員）

1968 年東京大学工学部電子工学科卒業．1970 年同大学大学院修士課程修了．1970〜1995 年 NTT 電気通信研究所．電子交換機の研究実用化，ITU 国際標準化等に従事．現在，国立情報学研究所教授．実時間システム，最適化コンパイラ，並行オブジェクト指向，分散 OS 等の研究に従事．1982 年電電公社総裁表彰．1993 年本学会論文賞．電子情報通信学会フェロー．博士（工学）．

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．