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

2007－ＳＬＤＭ－129／2007－ＥＭＢ－4（17）
2007／3／16

マルチプロセッサ用実時間スケジューリングにおける周波数制御手法

船岡 健司† 加藤 真平† 山崎 信行†

† 慶應義塾大学大学院 理工学研究科
〒 223-8522 神奈川県横浜市港北区日吉 3-14-1
E-mail: †{funaoka,shinpei,yamasaki}@ny.ics.keio.ac.jp

あらまし 組込システムへの要求が多様化するにつれて，プロセッサの消費電力増加が重大な問題となっている．プロセッサの周波数と電圧を制御することにより消費電力を抑えることが可能となり，システムの価値を高められる可能性がある．実時間スケジューリングの周波数制御手法に関する研究は，主にシングルプロセッサが対象とされてきた．本論文では，マルチプロセッサの実時間スケジューリングアルゴリズムとして最適な LLREF を拡張することによる周波数制御手法を提案する．シミュレーションにより提案アルゴリズムと理論上最適なアルゴリズムを比較した結果，システムの負荷が高い場合に，提案アルゴリズムによるプロセッサ周波数の無駄は 0.2%以下であることを示した．
キーワード 実時間システム，実時間スケジューリング，消費電力，周波数制御

Frequency Scaling in Real-Time Scheduling on Multiprocessors

Kenji FUNAOKA†, Shinpei KATO†, and Nobuyuki YAMASAKI†

† Graduate School of Science and Technology, Keio University
3-14-1 Hiyoshi, Kouhoku-ku, Yokohama, Kanagawa 223-8522 Japan
E-mail: †{funaoka,shinpei,yamasaki}@ny.ics.keio.ac.jp

Abstract As requirements for embedded systems are widely spread, the most serious limitation on these systemss is increase of electricity consumption. To control frequency and voltage of processors can decrease electricity consumption and improve worth of systems. Researches of frequency scaling techniques for real-time scheduling are mostly focused on single processors. In this paper, we provide the frequency scaling techniques which are an extension of LLREF, an optimal real-time scheduling algorithm on multiprocessors. Simulation results show that the difference between our algorithm and optimal algorithm is less than 0.2% when loads of systems are high.
Key words Real-Time Systems, Real-Time Scheduling, Electricity Consumption, Frequency Scaling

1. 序 論

テクノロジの発展と革新的なアイディアにより，プロセッサの性能は劇的に向上している．しかしながら，プロセッサの性能向上に起因して，消費電力や熱の増加が大きな問題となっている．したがって，命令レベルの並列性向上が限界に到達しており，スレッドレベルの並列性に着目したマルチプロセッサが注目されている．本論文におけるマルチプロセッサとは，複数のプロセッサを利用するものだけではなく，1 プロセッサに複数の実行単位を有する Simultaneous Multithreading（SMT）[1] や Chip Multiprocessing（CMP）[2] を含む．

携帯端末のような組込システムにおいて，駆動時間はシステムの価値を決定する．ほとんどのシステムにおいて，プロセッサは最も消費電力の大きい要素であることから，プロセッサの消費電力削減によってシステムの価値を高められる可能性がある．プロセッサの多くは CMOS 回路によって形成されており，動作周波数の最大値は動作電圧に依存する．CMOS 回路によって形成されているプロセッサの消費電力 E は，周波数 f と電圧 V の二乗に比例（$E \propto fV^2$）[3] することから，周波数と電圧を制御することにより消費電力を抑えることが可能となる．

また，組込システムには，ロボットの制御や画像処理のように時間制約を持つシステムが多い．このようなシステムでは，実時間スケジューリングと周波数制御を組み合わせることにより，時間制約を守った上で消費電力を抑えることが必要である．実時間スケジューリングの周波数制御手法に関する研究は，主にシングルプロセッサが対象とされてきた [4], [5].

マルチプロセッサを対象とした実時間スケジューリング理論では，最適なアルゴリズムとして PD^2 [6] と EKG [7], LLREF [8] が知られている．理論的には全て最適であるが，スケジュール手法が異なる．PD^2 は，Pfair [9] という厳しい制約を満たすた

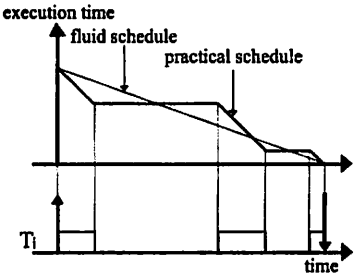

図 1　Fluid スケジュール

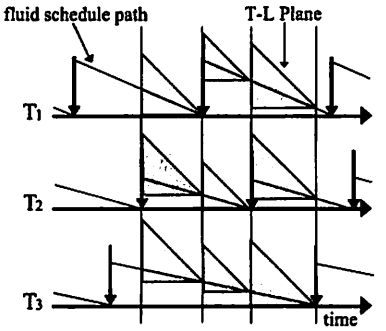

図 2　LLREF の概念図

め，時間を固定長に分割してスケジュールを行う．しかしながら，時間分割することにより，スケジュールの際に大きなオーバヘッドが発生する可能性がある．EKG は，タスクをプロセッサに割振っていき，割振れなかった時にタスクを分割して割振り実行する．この手法では，システムの利用率が低い場合，負荷が特定のプロセッサに偏る可能性がある．LLREF は，時間をタスクのデッドラインごとに分割し，T-L Plane と呼ばれる二等辺三角形を利用してスケジュールを行う．固定長の時間分割によらないアルゴリズムであり，PD^2 よりオーバヘッドを削減できる可能性がある．また，全てのプロセッサを均等に利用することが可能である．本論文では，オーバーヘッドが小さく周波数制御に向いていると考えられる LLREF において，時間制約を守りながら消費電力を削減することに焦点を当てる．

2. LLREF スケジューリングアルゴリズム

LLREF [8] は，マルチプロセッサシステム用の最適な実時間スケジューリングアルゴリズムである．理論的に，LLREF でスケジュールできないタスクセットは，いかなるアルゴリズムをもってしてもスケジュールできない．LLREF は，fluid スケジュール [10] の概念を利用して最適なスケジュールを実現する．fluid スケジュールは，図 1 に示されるように，常に一定の割合でタスクの実行を行う概念的なスケジューリング手法である．

図 2 は，LLREF によるスケジュールの概念図である．LLREF は，全タスクのデッドラインによって区切られた時間の中で，T-L Plane（Time and Local Execution Time Domain

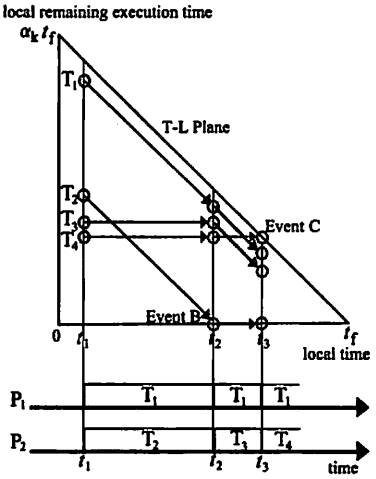

図 3　LLREF のスケジュール例

Plane）と呼ばれる三角形を利用して，その中でスケジュールを行う．T-L Plane の右下端を fluid スケジュールと一致させることにより，全てのデッドラインを守ることが可能である．全てのトークンが三角形の右下端に到達可能であることを局所的に実行可能であるという．局所的に実行可能であれば，全てのタスクのデッドラインを守ることが可能である．

T-L Plane の詳細を図 3 の上に示す．T-L Plane は，横軸に時間，縦軸にタスクの局所的な残り実行時間をとる直角三角形である．タスクは，現在時刻とタスクの局所的な残り実行時間の位置示すトークンに対応する．全てのタスクは，T-L Plane の中で局所的な残り実行時間を全て実行しなくてはならない．局所的な残り実行時間は，タスクを実行するとトークンが右下に移動して減少する．タスクを実行しないとトークンが右に移動して変化しない．トークンが三角形の底辺と斜辺（no local laxity diagnal）に移動したとき，それぞれイベント B とイベント C が発生する．タスクの到着（T-L Plane の左端）とイベント B，イベント C をまとめてイベントと呼ぶ．実行するタスクの判断は，イベントが発生したときに行い，優先度の高いタスクを選択する．LLREF は，局所的な残り実行時間が大きいタスクに高い優先度を与えて実行するアルゴリズムである．これにより最適なスケジュールを可能としている．

LLREF のスケジュール例を図 3 に示す．図のように 4 つのタスク T_i と 2 つのプロセッサ P_k が存在するシステムを想定する．時刻 t_1 において，局所的な残り実行時間の大きい T_1 と T_2 を選択する．時刻 t_2 まで実行していくと，タスク T_2 が三角形の底辺まで到達することにより，イベント B がが発生する．ここでも同様に，局所的な残り実行時間の大きい T_1 と T_3 を選択する．さらに実行していくと，時刻 t_3 にタスク T_4 が三角形の斜辺まで到達することにより，イベント C が発生する．ここでは，T_1 と T_4 を選択する．この操作を繰り返すことにより，全てのトークンを三角形の右下端まで到達させる．

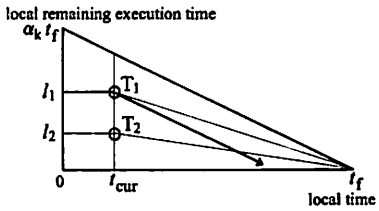

図 4 T-L Plane ($\alpha_k = 0.5$)

3. T-L Plane Scaling

本論文では，T-L Plane の斜辺の傾きを制御を周波数制御に対応させる手法を提案する．また，周波数制御条件下でのスケジュール可能性について示す．プロセッサの周波数が最大値の半分である場合の T-L Plane の例を図 4 に示す．タスクを実行すると斜辺の傾きに沿って局所的な残り実行時間が減少する．

3.1 システムモデル

T-L Plane における j 番目 ($j = 0, 1, 2, \ldots, f$) のイベントの発生時刻を t_j とする．ただし，t_0 はタスクの到着である．

システムは，M 個のプロセッサ $\mathbf{P} = \{P_1, P_2, \ldots, P_M\}$ から成る．それぞれのプロセッサ P_k の正規化した周波数を $\alpha_k (0 \leqq \alpha_k \leqq 1)$ とする．システムによって α_k を独立に決定できるシステムもあれば，そうでないシステムも存在する．全プロセッサの α_k の総和を $M_\alpha = \sum_{P_k \in \mathbf{P}} \alpha_k$ と定義する．

システムには，システムが処理しなければならないタスクの集合であるタスクセット $\mathbf{T} = \{T_1, T_2, \ldots, T_N\}$ が存在する．それぞれのタスクは独立しており，いつでも横取り（実行の一時中断）やマイグレーション（タスクのプロセッサ間の移動）が可能である．タスク T_i の到着時刻を a_i とする．周期タスク T_i は，周期 p_i と最悪実行時間 c_i に特徴付けられる．タスク T_i をプロセッサ P_k で実行したとき，タスク T_i は単位実行時間あたり最大で u_i/α_k のプロセッサ時間を要求する．タスク T_i の相対デッドラインは，周期 p_i と等しい．タスクは，このデッドラインまでに実行を終えなければならない．タスク T_i の利用率を $u_i = c_i/p_i$ $(0 < u_i \leqq 1)$，タスクセット $\mathbf{T}$ の利用率を $U = \sum_{T_i \in \mathbf{T}} u_i$，タスクセット $\mathbf{T}$ に含まれるタスクの利用率の最大値を $U_{\max} = \max_{T_i \in \mathbf{T}} \{u_i\}$ と定義する．T-L Plane において，タスク T_i の時刻 t_j における局所的な残り実行時間を $l_{i,j}$，局所的な利用率を $r_{i,j} = \frac{l_{i,j}}{t_f - t_j}$ と定義する．また，時刻 t_j における局所的な利用率の和を $S_j = \sum_{T_i \in \mathbf{T}} r_{i,j}$ と定義する．

3.2 スケジュール可能性

本節では，全てのプロセッサの周波数が等しく $\alpha (= \alpha_1 = \alpha_2 = \ldots = \alpha_M)$ であるシステムを想定する．SMT や CMP のように，チップ上に複数の実行単位があるプロセッサでは，実行単位ごとに周波数や電圧を制御できないものも存在する．このようなシステムにおけるスケジュール可能性について示す．

提案する周波数決定アルゴリズム DecideUniformFrequency を図 5 に示す．この条件下でのスケジュール可能性に関する定理を以下に示す．余白の都合から証明については省略する．証明方法については，LLREF のものとほぼ同様である．

アルゴリズム: DecideUniformFrequency

```
1: foreach 1...M as k
2:     α_k = max{U_max, U/M}
8: end foreach
```

図 5 均一周波数決定アルゴリズム

アルゴリズム: AcceptanceTest

```
1: when T_i arrives at time t such that t_{e-1} < t ≦ t_e
2:   if U + U_i > M
3:     reject T_i
4:   else
5:     accept T_i
6:     T = T ∪ {T_i}
7:     if p_i ≧ t_f − t_e
8:       a_i = t_e
9:     else
10:      a_i = t_f
11:    end if
12:  end if
13: end when
```

図 6 タスク受け入れ判定

[定理 1] イベントが時刻 t_e で発生したとする．$S_{e-1} \leqq \alpha M$ であれば，$S_e \leqq \alpha M$ である．

証明 省略 ■

[定理 2] LLREF によってタスクセットがスケジュール可能であるための必要十分条件は，$U \leqq \alpha M \cap U_{\max} \leqq \alpha$ である．

証明 省略 ■

周波数決定アルゴリズム DecideUniformFrequency は，定理 2 よりスケジュール可能な周波数決定を行うアルゴリズムである．これにより，周波数を独立に設定できないシステムにおいて，周波数 α を最小化することができる．

3.3 動的なシステム

前節では，タスクセットが変化しない静的なシステムを想定した．タスクセットが動的に変化するシステムでは，それにあわせて周波数も動的に変更しなくてはならない．周波数の変更はいつでも行えるわけではなく，変更のタイミングを誤るとタスクがデッドラインを守れない可能性がある．

初めに，タスクが消滅した際の周波数決定の定理を示す．タスクが消滅するとタスクセットの利用率は減少することから，タスクの到着よりも簡単に考えることが可能である．

[定理 3] タスク T_i が時刻 $t\,(t_{e-1} < t \leqq t_e)$ に消滅したとする． 時刻 t_e に周波数決定 DecideFrequencySimple を行えば，このタスクセットはスケジュール可能である．

証明 定理 1 より帰納法を利用する．イベントの時刻 t_e に周波数の決定を行い，周波数が α_{new} となったとする．よって，$S_e \leqq \alpha_{\text{new}} M$ が成立する．定理 1 より，次のイベントが発生する時刻 t_{e+1} でも $S_{e+1} \leqq \alpha_{\text{new}} M$ の関係が成立する．帰納法より局所的に実行可能であり，fluid スケジュールパスでスケジュールすることが可能である． ■

アルゴリズム: ClassifyTasks

Require: $u_1 \geqq u_2 \geqq \ldots \geqq u_N$
1: $u = U$
2: $\mathbf{T}^{\text{heavy}} = \{\}$
3: $\mathbf{T}^{\text{light}} = \mathbf{T}$
4: **foreach** 1...N **as** i
5: **if** $u - u_i \leqq M - i$ **then**
6: $\mathbf{T}^{\text{heavy}} = \mathbf{T}^{\text{heavy}} \cup \{T_i\}$
7: $\mathbf{T}^{\text{light}} = \mathbf{T}^{\text{light}} \backslash \{T_i\}$
8: $u = u - u_i$
9: **else**
10: **break**
11: **end if**
12: **end foreach**

図 7 タスク分類アルゴリズム

次に新しいタスクが到着した際の周波数決定について述べる．新しいタスクを受け入れるかどうか判断するタスク受け入れ判定のアルゴリズムを図 6 に示す．このタスク受け入れ判定によるスケジュール可能性に関する定理を以下に示す．

[定理 4] 新しいタスク T_i が時刻 $t\,(t_{e-1} < t \leq t_e)$ に到着したとする．ただし，$U + U_i \leq M$ とする．タスク T_i の周期 p_i が $p_i \geqq t_f - t_e$ であるならば，タスク T_i の到着時刻 $a_i = t_e$ とする．タスク T_i の周期 p_i が $p_i < t_f - t_e$ であるならば，タスク T_i の到着時刻 $a_i = t_f$ とする．タスクの到着時刻 a_i に周波数決定 DecideFrequencySimple を行えば，このタスクセットはスケジュール可能である．

証明 定理 3 と同様である． ■

定理 3 と定理 4 により，システムの実行中に動的なタスクセットの変更があってもスケジュール可能である．

3.4 周波数の最小化

本節では，全てのプロセッサの周波数を独立に変更可能で静的なシステムにおける周波数制御手法を述べる．前節における周波数制御のスケジュール可能性は，$U_{\max}$ に大きく依存していた．そのため，利用率の大きなタスクがボトルネックとなり，周波数を十分に下げることができない可能性がある．

この問題を解決するため，タスクをヘビータスクとライトタスクのグループに分類する．ヘビータスクに分類されたタスクは，1 つのプロセッサで 1 つのタスクを実行する．ヘビータスク T_i をプロセッサ P_k で実行するときの周波数は，$\alpha_k = u_i$ である．ライトタスクに分類されたタスクは，ヘビータスクを実行していないプロセッサで実行する．ライトタスクは，残りのプロセッサで LLREF によりスケジュールを行い，プロセッサの周波数を DecideUniformFrequency と同様に決定する．

ヘビータスクの集合を $\mathbf{T}^{\text{heavy}}$，ライトタスクの集合を $\mathbf{T}^{\text{light}}$，ヘビータスクの利用率の和を U^{heavy}，ライトタスクの利用率の和を U^{light}，ヘビータスクの利用率の最大値を $U_{\max}^{\text{heavy}}$，ライトタスクの利用率の最大値を $U_{\max}^{\text{light}}$ と定義する．以下，簡単にためにヘビータスクの数 $|\mathbf{T}^{\text{heavy}}|$ を L と表す．

タスクセットの分類アルゴリズムと周波数決定アルゴリズムを図 7 と図 8 に示す．あらかじめ，全タスクを利用率の降順にソートして，利用率の大きなタスクをヘビータスクに分類する．タスクをヘビータスクに分類してもスケジュール可能であれば，そのタスクをヘビータスクとする．ライトタスクには，利用率の小さいタスクが残ることにより，$U_{\max}^{\text{light}}$ を小さくすることが可能となる．以下にこれらのアルゴリズムに関する定理を示す．

アルゴリズム: DecideFrequency

Require: $u_1 \geqq u_2 \geqq \ldots \geqq u_N$
1: **foreach** 1...M **as** k
2: **if** P_k is assigned a heavy task **then**
3: $\alpha_k = u_k$
4: **else**
5: $\alpha_k = U^{\text{light}}/(M - L)$
6: **end if**
7: **end foreach**

図 8 周波数決定アルゴリズム

[補題 5] アルゴリズム ClassifyTasks によってタスクを分類して，アルゴリズム DecideFrequency によって周波数を決定したとき，$U = M_\alpha$ である．

証明

$$
\begin{aligned}
M_\alpha &= \sum_{k=1}^{L} \alpha_k + \sum_{k=L+1}^{M} \alpha_k \\
&\Leftarrow \begin{cases} \alpha_k = u_k & 1 \leq k \leq L \\ \alpha_k = U^{\text{light}}/(M-L) & L < k \leqq M \end{cases} \\
&= U^{\text{heavy}} + (M-L)\frac{U^{\text{light}}}{M-L} = U
\end{aligned}
$$

■

[定理 6] $U > M_\alpha$ であればスケジュール不可能である．

証明 ヘビータスクは必ずスケジュール可能であるため，ライトタスクのスケジュール可能性について考える．

$$
\begin{aligned}
&U^{\text{light}} - \sum_{k=L+1}^{M} \alpha_k \\
&\quad \Leftarrow U^{\text{heavy}} = \sum_{i=1}^{L} \alpha_k \\
&\quad = U^{\text{heavy}} + U^{\text{light}} - \sum_{k=1}^{L} \alpha_k - \sum_{k=L+1}^{M} \alpha_k \\
&\quad = U - M_\alpha > 0 \\
&\quad \Rightarrow U^{\text{light}} > \sum_{k=L+1}^{M} \alpha_k
\end{aligned}
$$

定理 2 より $U > M_\alpha$ であればスケジュール不可能である． ■

[定理 7] アルゴリズム ClassifyTasks とアルゴリズム DecideFrequency によって，周波数の和 M_α は最小化される．

証明 補題 5 と定理 6 より明らか． ■

定理 4 より，ClassifyTasks と DecideFrequency によって，タスクセットをスケジュール可能な最低限の周波数を割当可能である．よって，理論的には最適なアルゴリズムである．

表 1　システムの選択可能な周波数

	システム 1	システム 2	システム 3
1	0.5	0.5	0.36
2	0.75	0.6	0.55
3	1.0	0.7	0.64
4		0.8	0.73
5		0.9	0.82
6		1.0	0.91
7			1.0

4. 評　　価

周波数を自由に設定可能なプロセッサでは，前節から理論的には周波数を最小化することが可能であることを示した．しかしながら，実際のプロセッサでは設定可能な周波数が制限される場合がある．本節では，シミュレーションによってタスクセットの利用率と周波数の和の関係を示す．これにより，提案アルゴリズムの有効性を示す．

4.1 評価方法

評価に利用するシステムは以下の通りである．システムのプロセッサ数を $M = 8$ とする．表 1 は，評価に利用するシステムである．それぞれのプロセッサは，表 1 に示された α の値をプロセッサ毎に独立して設定可能である．システム 1 では，周波数を 3 段階に設定可能である．システム 2 では，周波数をシステム 1 よりも細かく 6 段階に設定可能である．システム 3 では，周波数を細かく設定可能であり，他のシステムよりも低い周波数 0.36 を選択可能である．

タスクセットは，タスクの利用率が一様分布となるように生成した．タスクの周期 p_i は $[1, 100]$ の整数値から，実行時間は $[1, p_i]$ の整数値から一様乱数によって決定する．タスクをタスクセットに追加していき，タスクセットの利用率が目標値を超えてしまったらタスクを破棄する．タスクが連続して 10 回破棄されたらタスクセットの利用率が目標値となるようなタスクを生成して，これを評価に利用するタスクセットとする．

それぞれのタスクセットで ClassifyTasks と DecideFrequency によってプロセッサの周波数を決定して，frequency ratio を求める．frequency ratio の定義を式 (1) に示す．frequency ratio が 1 のとき，最低限の周波数で実行できている．frequency ratio が大きいほど，システムに無駄があることを意味する．

$$\text{frequency ratio} = \frac{M_\alpha}{U} \tag{1}$$

選択可能な周波数は表 1 に限られていることから，Decide-Frequency によって決定した周波数以上となる選択可能な最小値をプロセッサの周波数とする．タスクセットの利用率が [0.5,8.0] の 0.5 毎に，それぞれ 1000 のタスクセットを生成して frequency ratio の平均値を求めた．

比較対象は，理論通り制限なく自由に周波数を設定可能な Optimal アルゴリズムである．Optimal アルゴリズムでは，常に frequency ratio が 1 となる．周波数の和を低くするという観点からは，frequency ratio を Optimal アルゴリズムの値に近づけるほど優れたアルゴリズムであると言える．

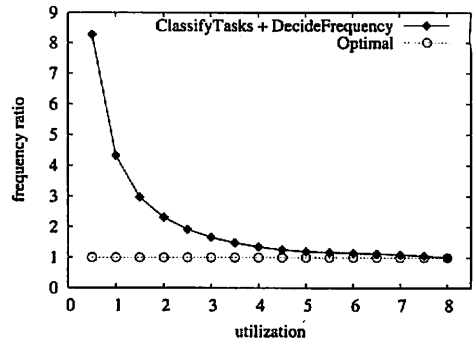

図 9　システム 1 の評価結果

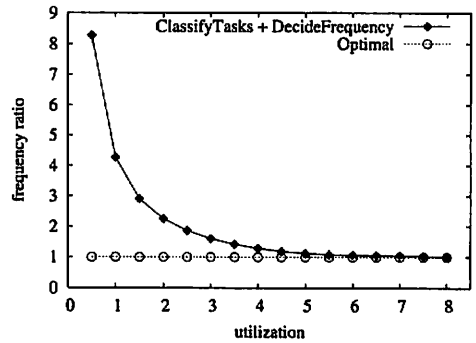

図 10　システム 2 の評価結果

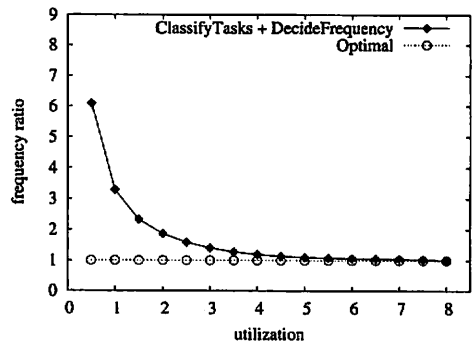

図 11　システム 3 の評価結果

4.2 評価結果

評価の結果を図 9 から図 11 に示す．横軸はタスクセットの利用率，縦軸は評価指標の frequency ratio である．どの結果もタスクセットの利用率が高くなるほど frequency ratio が Optimal の値に近づいている．逆に，タスクセットの利用率が低いと frequency ratio の値が大きくなるという傾向が見られる．これは，システムの選択可能な周波数が表 1 に示したように一定以上に限られているからである．よって，システムの利用率が低い場合に無駄が多くなるということが読み取れる．こ

のように，プロセッサには動作可能な最低電圧があり，一定以上に下げることができない場合が多い．どのシステムでも，タスクセットの利用率が設定可能な最低周波数のプロセッサ数倍を超えたあたりから減少が緩やかになっている．全てのシステムにおいて，タスクセットの利用率が 4 を超えると，Optimal と比較して frequency ratio が約 0.25 以下の差となっている．

次に選択可能な周波数と frequency ratio の関係を見る．システム 1 とシステム 2 は，選択可能な周波数の数が大きく異なる．ここでは，システム 2 の方が最大で約 0.08 ほど frequency ratio を低くすることが可能となっている．また，システム 2 の方が低い frequency ratio で実行可能となっている．設定可能な周波数の数が多いほど柔軟に周波数を決定可能であることから，妥当な結果であると言える．また，システム 2 とシステム 3 では，設定可能な最小の周波数が異なる．システム 2 とシステム 3 を比較すると，システム 3 の方が最大で 2.18 ほど frequency ratio を低くすることが可能となっている．この差は，タスクセットの利用率が低いほど顕著に表れている．タスクセットの利用率が高くなるほど，この差は小さくなっている．低い周波数を設定可能であれば，その周波数で低い利用率のタスクを実行することが可能であることから，このような結果になったと考えられる．よって，frequency ratio は設定可能な最小の周波数によって大きく変化するということが読み取れる．

最後に，システムの利用率が高い場合に着目する．どのシステムでも，システムの利用率が高くなると frequency ratio の差は小さくなっていっている．特に利用率が最大の 8 となると，Optimal との差が全てのシステムで約 0.002 以下となっている．

LLREF は，従来のアルゴリズムでスケジュール不可能な利用率の高いタスクセットをスケジュール可能である．評価結果より，提案アルゴリズムはそのような利用率の高いタスクセットを扱うシステムの周波数制御に有効である．

5. 結　論

本論文では，最適なマルチプロセッサ用の実時間スケジューリングアルゴリズム LLREF を拡張することによる周波数制御手法を提案した．また，周波数制御手法の理論的なスケジュール可能性に関する定理を示した．我々の知る限り，提案手法は，最適なマルチプロセッサ用の実時間スケジューリングアルゴリズムの初めての周波数制御手法である．

本論文において，システムの周波数を独立に設定可能なシステムとそうでないシステムについて検討を行い，それぞれのシステムに付いて周波数決定アルゴリズムを示した．また，システムの周波数を独立に設定可能なシステムでは，周波数を最小化可能であることを示した．また，システムの周波数が独立に設定不可能なシステムについて，タスクセットが動的に変化するシステムの受け入れ可能性について述べた．

タスクセットが変化しない静的なシステムでは，実行開始時に周波数を決定し，実行時には周波数決定アルゴリズムによるオーバーヘッドがない．また，タスクセットが変化する動的なシステムでは，タスクセットの受け入れ判定が $O(1)$ で，周波数の変更は $O(M)$ で可能である．基本的にシステム構成が決まることによって M は定数となることから，本論文で示した全てのアルゴリズムは定数時間で実行可能である．

今後の課題としては，提案アルゴリズムを柔軟なものにする手法を考えていきたい．周波数を独立して設定可能であり，タスクセットが変化する動的なシステムの受け入れ決定手法を提案したい．本論文における周波数を独立して設定可能な手法は，LLREF と異なるスケジュールを行うことから，独立して設定できないシステムのように単純化することはできない．

本論文の提案手法は，実際の実行時間が最悪実行時間と等しい場合は有効である．しかしながら，時間制約を守るために最悪実行時間は実際の実行時間よりも大きく設定することが多い．また，SMT のようなプロセッサでは，資源の競合から実際の実行時間が大きく変動する可能性がある．よって，実行時間の変動に対応した動的な周波数制御手法を提案したい．これにより，実際の負荷に応じた周波数制御を行うことが可能となり，システムの消費電力を抑えることが可能となる．

最後に，これらの提案手法を実機で実装して実用性を確認したい．実際のシステムでは，スケジュールや周波数変更などのオーバーヘッドも考慮しなくてはならない．よって，オーバーヘッドも考慮した実装を行う必要がある．

謝辞 本研究は，科学技術振興機構 CREST の支援による．

文　献

[1] D. M. Tullsen, S. J. Eggers and H. M. Levy: "Simultaneous Multithreading: Maximizing On-Chip Parallelism", In Proc. of the 22nd Annual International Symposium on Computer Architecture, pp. 392–403 (1995).

[2] K. Olukotun, B. A. Nayfeh, L. Hammond, K. Wilson and K. Chang: "The Case for a Single-Chip Multiprocessor", In Proc. of the 7th International Conference on Architectural Support for Programming Languages and Operating Systems, pp. 2–11 (1996).

[3] T. D. Burd and R. W. Brodersen: "Energy Efficient CMOS Microprocessor Design", In Proc. of the 28th Annual Hawaii International Conference on System Sciences, pp. 288–297 (1995).

[4] J. A. Stankovic, C. Lu and S. H. Son: "The Case for Feedback Control Real-Time Scheduling", In Proc of the 11th Euromicro Conference on Real-Time Systems, pp. 11–20 (1999).

[5] P. Pillal and K. G. Shin: "Real-Time Dynamic Voltage Scaling for Low-Power Embedded Operating Systems", In Proc. of the ACM Symposium on Operating Systems Principles, pp. 89–102 (2001).

[6] J. H. Anderson and A. Srinivasan: "Early-Release Fair Scheduling", In Proc. of the 12th Euromicro Conference on Real-Time Systems, pp. 35–43 (2000).

[7] B. Andersson and E. Tovar: "Multiprocessor Scheduling with Few Preemptions", In Proc. of the 12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, pp. 322–334 (2006).

[8] H. Cho, B. Ravindran and E. D. Jensen: "An Optimal Real-Time Scheduling Algorithm for Multiprocessors", In Proc. of the 27th IEEE Real-Time Systems Symposium, pp. 101–110 (2006).

[9] S. K. Baruah, N. K. Cohen, C. G. Plaxton and D. A. Varvel: "Proportionate Progress: A Notion of Fairess in Resource Allocation", Algorithmica, **15**, 6, pp. 600–625 (1996).

[10] P. Holman and J. H. Anderson: "Adapting Pfair Scheduling for Symmetric Multiprocessors", Journal of Embedded Computing, **1**, 4, pp. 543–564 (2005).