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　64–74 (Mar. 2009)

Rate Monotonicに基づくマルチプロセッサ用リアルタイムスケジューリング

武田　瑛[†1]　加藤真平[†1]　山崎信行[†1]

近年，組み込みリアルタイムシステムにおいてもマルチプロセッサ技術の利用が一般的になりつつある．このような背景から，マルチプロセッサにおいてCPUを100%利用できる最適なリアルタイムスケジューリングアルゴリズムが提案されているが，多くのコンテキストスイッチやタスクマイグレーションが発生し，それらのオーバヘッドにより実用性の面で問題視されている．一方で，従来の単純なアルゴリズムでは高いスケジュール可能性を実現することができない．本論文では，従来の単純なアルゴリズムであるRate Monotonic（RM）を基にしたスケジューリングアルゴリズムRMZLを提案する．提案するアルゴリズムは，高い予測性や小さいジッタ，少ないオーバヘッドなどのRMの長所を残しつつ，スケジュール可能性を向上させるものである．シミュレーション評価により，提案アルゴリズムは大きなオーバヘッドを要することなく従来のRMを基にしたスケジューリングアルゴリズムよりも多くのタスクをスケジュール可能であることを示す．

Real-time Scheduling Based on Rate Monotonic for Multiprocessors

Akira Takeda,[†1] Shinpei kato[†1]
and Nobuyuki Yamasaki[†1]

In recent embedded systems multiprocessor platforms are commonly used. Due to this background, optimal real-time scheduling algorithms which can use full system utilization have been proposed, but these algorithms generate a number of context switches and task migrations that incur significant overhead and are often considered not to be practical due to the overhead. Meanwhile existing simple algorithms cannot improve the schedulability. This paper propose a new multiprocessor real-time scheduling algorithm based on global Rate Monotonic (RM) which is one of simple conventional algorithms. Our algorithm remains the merit of RM such as high predictability, low jitter, and low overhead, and also improves the schedulability. The simulation evaluation shows that our algorithm outperforms the existing global RM based algorithm in the schedulabiltiy point of view.

1. はじめに

今日の組み込みリアルタイムシステムは，ロボットやユビキタスアプリケーション等の出現により，高性能なプラットフォームを必要とする傾向にある．特に，ヒューマノイドロボットには，より高度の知能や認識能力，俊敏な運動性能，そして高い安全性が求められる[1)]．発熱や消費電力といった点を考慮すると，従来のシングルプロセッサの動作周波数を上げることは難しいため，高い情報処理能力を得るためには，同時細粒度マルチスレッディング（SMT）[2)]やチップマルチプロセッシング（CMP）[3)]などの高並列アーキテクチャを用いることが好ましいと考えられる．また，ヒューマノイドロボットには実環境の力学に追随して反応するためのリアルタイム性能が求められるため，そのような高並列アーキテクチャ上でリアルタイムスケジューリングを行うことが必要である．しかし，Rate Monotonic（RM）[4)]やEarliest Deadline First（EDF）[4)]のような従来のシングルプロセッサで最適なスケジューリングアルゴリズムは，マルチプロセッサでもはや最適ではなく，アーキテクチャの並列性を十分に利用できないことが知られている[5)]．マルチプロセッサにおけるリアルタイムスケジューリング手法は今日広く議論されている．

リアルタイムスケジューリングは，主に動的優先度方式と静的優先度方式に分類される．動的優先度方式は，高いスケジュール可能性を達成できるが，タスクのコンテキストスイッチが多発し，オーバヘッドが大きくなる傾向がある．一方静的優先度方式は，スケジュール可能性は低いが，タスクのディスパッチ処理にかかるオーバヘッドを小さくすることができる．また，静的優先度方式には，高い予測性とタスクのジッタが小さいという重要な利点がある．先のロボット制御の例では，タスクは，タイミングのずれが制御不安定性につながるハードリアルタイムタスク，マルチメディア処理などのより長い周期で起動されるソフトリアルタイムタスク，行動計画などの非リアルタイムタスクに分けられる．このような多種のタスクが存在するシステムでは，高負荷時に低優先度の非リアルタイムタスクから必ずデッドラインミスするという予測性が重要である．また，制御タスクは，つねに一定のタイミン

†1 慶應義塾大学
Keio University

グで実行されなければならず，ジッタが小さいことが求められる．

本論文の目標は，マルチプロセッサ上のリアルタイムシステムにおいて，高い予測性や小さいジッタといった静的優先度の長所を継承しながら，大きいオーバヘッドを発生させることなくスケジュール可能性を向上させることである．そのため，静的優先度アルゴリズムである RM にゼロ余裕時間ルール[6] を適用したアルゴリズムを提案する．そして，ハードリアルタイムシステムにおいて不可欠なスケジュール可能性判定式を示す．最後にシミュレーションにより，提案したアルゴリズムをタスクセットのスケジュール成功率とタスクのプリエンプション数の観点から評価する．

本論文の構成を以下に示す．2 章では，本論文で仮定するシステムモデルと用語の定義を述べる．次に，3 章でマルチプロセッサにおけるリアルタイムスケジューリングアルゴリズムに関する関連研究およびそれらの問題点について述べる．4 章で提案するアルゴリズムを述べ，5 章でそのスケジュール可能性解析について述べる．6 章では提案アルゴリズムのデッドラインからの遅延時間の上限値の解析について述べる．その後，7 章でシミュレーションによる評価結果を述べ，最後に 8 章で結論と今後の課題について述べる．

2. システムモデル

本論文では，マルチプロセッサリアルタイムスケジューリングの研究における一般的なシステムモデルを仮定する．システムは m 個のプロセッサまたはコアから構成されるものとする．そして n 個のタスクから構成されるタスクセット $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$ がシステムに与えられる．各タスク τ_i は (C_i, T_i) というタプルで定義される．C_i は最悪実行時間であり，T_i は周期である．タスクは周期の短い順，つまり RM において優先度の高い順にソートされているものとする．つまり，$T_1 \leq T_2 \leq \ldots \leq T_n$ が成り立つ．τ_i の利用率を $U_i = C_i/T_i$ で表す．また，タスクセット τ に含まれるタスクの合計利用率を $U(\tau) = \sum_i U_i$ で表す．すなわち，$U(\tau)$ はシステム全体の負荷を表す．ここで，$U(\tau)/m$ をシステム利用率と呼ぶことにする．タスクは一連のジョブを周期的に生成する．タスク τ_i の k 番目のジョブを $\tau_{i,k}$ と表す．$\tau_{i,k}$ は時刻 $r_{i,k}$ でリリースされ，デッドライン $d_{i,k}$ は次のジョブのリリース時刻とする．すなわち，$d_{i,k} = r_{i,k+1} = r_{i,k} + T_k$ とする．時間 t において，ジョブ $\tau_{i,k}$ の残り実行時間を $C_{i,k}(t)$ とするとき，$\tau_{i,k}$ の余裕時間（laxity）を $L_{i,k}(t) = d_{i,k} - t - C_{i,k}(t)$ と定義する．余裕時間はそのジョブの緊急度を表し，緊急度は余裕時間が少なくなるほど増す．余裕時間が負であることはデッドラインミスを意味する．余裕時間が負のときの余裕時間の絶対値を特に遅延時間（tardiness）と呼ぶ．遅延時間はジョブのデッドラインからの遅れを意味する．ジョブ $\tau_{i,j}$ がレディ状態であるが，すべてのプロセッサが他の優先度の高いジョブによって使用されているとき，ジョブ $\tau_{i,j}$ はそれらのジョブにブロックされているという．タスクはプリエンプト可能で互いに独立しているものとし，いかなるタスクも複数のプロセッサで実行することはできないものとする．また，1 度システムの実行が開始されたら，新たなタスクが到着したり，すでにシステムに存在するタスクが消滅したりすることはないものとする．

タスクセットがスケジュール可能であるとは，スケジューリングアルゴリズムによってすべてのジョブがデッドラインをミスすることなく実行を完了できることをいう．あるスケジューリングアルゴリズムに対するスケジュール可能性判定とは，入力にタスクセットが与えられ，出力に与えられたタスクセットがスケジュール可能であるかどうかを返すアルゴリズムである．ヒューマノイドロボットのようなハードリアルタイムタスクを含むシステムでは，スケジューリングアルゴリズムがスケジュール可能性判定を有している必要がある．一方，ソフトリアルタイムタスクは多少のデッドラインミスは許容されるが，タスクの品質（QoS）を保証するために大きな遅延は許されない．よってそれらのタスクをスケジュールするアルゴリズムは，タスクの遅延時間の上限値を事前に解析できるアルゴリズムを有することが望ましい．また，タスクの平均の反応時間が重要なシステムの場合，スケジューリングアルゴリズムが仕事量を保存することが望ましい．仕事量を保存するとは，実行可能なタスクが存在するときにプロセッサをアイドル状態にしないことをいう．

システムはメモリ共有型のマルチプロセッサシステムとし，各プロセッサがデータとコードを共有できるものとする．タスクのプリエンプションやプロセッサ間の移動の時間コストはアーキテクチャに大きく依存するため，本論文では考えないこととする．提案アルゴリズムの評価では，時間コストではなくプリエンプションの頻度を評価指標とする．

3. 関 連 研 究

マルチプロセッサ用スケジューリングは，タスクをプロセッサに割り当てる方式として，パーティショニング方式とグローバルスケジューリング方式の 2 つに大別できる．パーティショニング方式は，あらかじめ静的に各プロセッサにタスクを割り当て，各プロセッサで独立にスケジュールを行う方式である．一方グローバルスケジューリング方式は，タスクをシステム全体で 1 つのレディキューに入れ，高優先度のタスクから動的に各プロセッサにスケジュールする方式である．パーティショニング方式は，実装や計算が容易であり，タスクのプロセッサ間の移動が起きないのでオーバヘッドが低いという利点があるが，利用可能なシ

ステム利用率は必ず 50%以下であることが証明されている[7]．また，タスクを静的にプロセッサに割り当ててしまうため，プロセッサの負荷が偏りやすい．一方グローバル方式は，タスクのプロセッサ間の移動によるオーバヘッドが大きい半面，マルチプロセッサの並列性を最大限利用できるためにタスクの平均反応時間が短いという長所がある．また，高いスケジュール可能性を達成することができ，現存する最適スケジューリングアルゴリズムのすべてはこのグローバルスケジューリング方式である．

マルチプロセッサシステムにおける最適なリアルタイムスケジューリングアルゴリズムとしては，Pfair[8]，LNREF[9]，EKG[10] の 3 つがこれまで提案されてきている．Pfair スケジューリングでは，各タスクを非常に短い時間（クォンタム）を持つサブタスクに分割し，それらのサブタスクで優先度スケジューリングを行う．よって，最悪の場合，毎ティックタスクのコンテキストスイッチやマイグレーションが起こり，スケジューラのオーバヘッドが大きくなってしまう．LNREF アルゴリズムは，このような Pfair スケジューリングの制限を緩和したアルゴリズムであるが，基本は Pfair スケジューリングと同じプロセッサシェアリング型のアルゴリズムであるため，やはりオーバヘッドが大きい．EKG アルゴリズムは，最適な 3 つのアルゴリズムのうち最もオーバヘッドが低いと考えられるが，Next-fit パーティショニングアルゴリズムに基づいているため，低負荷時にはタスクが特定のプロセッサに偏ってしまうという問題点がある．本論文では，評価の際にこれらの最適アルゴリズムと提案アルゴリズムのオーバヘッドの違いについて見る．

これらの最適なスケジューリングアルゴリズムは動的優先度アルゴリズムに分類されるため，制御タスクのような正確なタイミングが重視されるタスクを含むシステムには好ましくない．一方，RM は静的優先度アルゴリズムに分類され，その簡潔さや高い予測性の点で実際のシステムにおいてしばしば利用されるアルゴリズムである．しかしながら，特にグローバルスケジューリング方式の RM は，Dhall's Effect[5] により利用可能な CPU 利用率が非常に低い．プロセッサの数を m とすると，グローバル RM の利用可能なシステム利用率は $1/m$ である．そこで Andersson[11] らは，グローバル RM において $m/(3m-2)$ より利用率が高いタスクを最高優先度にすることにより，利用可能なシステム利用率を $m/(3m-2)$ まで改善するアルゴリズム RM-US$[m/(3m-2)]$ を提案した．しかしながら，RM-US アルゴリズムは RM がスケジュール可能なタスクセットをスケジュールできないことがあり，結果としてスケジュール可能性の低下を招くことがある．

従来から用いられている単純なスケジューリングアルゴリズムとしては，RM のほかに EDF や Least Laxity Algorithm（LLA）[12] があげられる．EDF と LLA はともにシングルプロセッサで最適なアルゴリズムであるが，マルチプロセッサでは EDF は利用可能なシステム利用率が低く，LLA は EDF に比べてスケジュール可能性が高い反面コンテキストスイッチが頻発しオーバヘッドが高い．Cho[6] らは，EDF と LLA を組み合わせることで，大きいオーバヘッドを発生させることなく EDF のスケジュール可能性を向上させるアルゴリズムとして ED/LL（EDF/Least Laxity）と EDZL（EDF until Zero Laxity）を提案した．ED/LL は，通常のスケジューリングでは EDF を用いて，余裕時間が 0 のタスクが存在する場合には LLA を用いるアルゴリズムである．EDZL は，通常のスケジュールでは EDF を用いてスケジューリングを行い，余裕時間が 0 のタスクが存在した場合にはそのタスクを最高優先度にするアルゴリズムである．特に EDZL は，少ないオーバヘッドで比較的高いスケジュール可能性を得ることができる．また，EDF でスケジュール可能なタスクセットは EDZL でも必ずスケジュール可能であるという重要な性質を持っている．EDZL で用いられている余裕時間が 0 のタスクを最高優先度にするというルールはゼロ余裕時間ルール（Zero Laxity Rule）と呼ばれる．

4. スケジューリングアルゴリズム

提案アルゴリズム RMZL（Rate Monotonic until Zero Laxity）はゼロ余裕時間ルールを RM に適用したものである．つまり，通常は RM によりジョブをスケジュールし，ジョブの余裕時間が 0 になったらそのジョブを最高優先度にする．RMZL アルゴリズムを**図 1** に示す．まず，余裕時間が負であるジョブが存在すると，そのジョブはデッドラインミスを起こしているので，タスクをレディーキューから外す（2–3 行目）．次に，余裕時間が 0 である各ジョブについて，それぞれ以下の処理を行う（5–15 行目）．アイドル状態のプロセッサが存在するならば，そこでタスクを実行する（6–7 行目）．アイドル状態のプロセッサがない場合は，余裕時間が正のジョブがあればそのジョブをプリエンプトする（8–10 行目）．なければ，余裕時間が 0 でかつ自ジョブよりも周期の長いジョブをプリエンプトする（11–13 行目）．これらのいずれにも一致しない場合は，何もしない．結果として，そのジョブは余裕時間が負となり，上で述べたようにレディーキューから外される．すべてのジョブの余裕時間が正ならば，それらは通常の RM によってスケジュールされる（16–18 行目）．

図 2 は 2 プロセッサ上でタスクセット $\tau = \{\tau_1 = (2,3), \tau_2 = (2,3), \tau_3 = (2,3)\}$ を RM と RMZL でスケジュールした例である．すべてのタスクの周期は等しいので，RM スケジュールと RMZL スケジュールともに τ_1，τ_2 が実行を開始する．RM スケジュールでは，$t = 2$ まで τ_1 と τ_2 が実行してしまうため，τ_3 はデッドラインミスを起こしてしまう．しか

```
1.  while TRUE do
2.    if there is any job with the negative laxity then
3.      remove it from the ready queue;
4.    end if
5.    while there is any job τ_{i,j} with zero-laxity do
6.      if there is an idle processor then
7.        execute τ_{i,j} on it;
8.      else if there is any job with a positive laxity then
9.        preempt the job with a positive laxity
10.         and the largest period;
11.     else if there is any other job τ_{k,l} with zero laxity
12.       and larger period than τ_{i,j} then
13.       preempt τ_{k,l}
14.     end if
15.   end while
16.   if every job has a positive laxity then
17.     perform RM
18.   end if
19. end while
```

図 1　RMZL アルゴリズム
Fig. 1　RMZL algorithm.

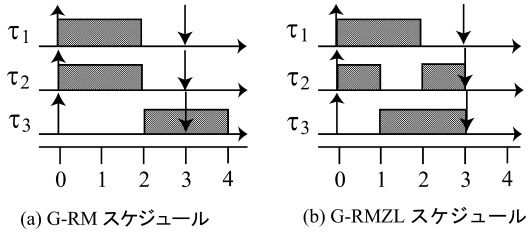

図 2　(a) RM スケジュールと (b) RMZL スケジュールの例
Fig. 2　Examples of (a) RM schedule and (b) RMZL schedule.

しながら RMZL スケジュールでは，$t = 1$ において τ_3 の余裕時間が 0 となり，最高優先度となる．よって τ_2 はプリエンプトされ，τ_3 が実行を開始する．$t = 2$ では τ_1 が実行を終了するので，τ_2 が実行を再開する．結果として，RMZL スケジュール例ではすべてのタスクがデッドラインミスすることなく実行できていることが分かる．

RMZL は，RM と同様に仕事量を保存するアルゴリズムである．すなわち，実行されていないレディージョブがある限りプロセッサはアイドル状態にならない．よって，仕事量を保存しないアルゴリズムに対してタスクの平均反応時間が良い．仕事量を保存しないアルゴリズムである最適スケジューリングでは，低負荷時に平均反応時間が仕事量を保存するアルゴリズムに比べて著しく悪化する．また，ゼロ余裕時間ルールは安全（safety）ルールである．つまり，RM と RMZL のスケジュールが異なってくるのは RM がデッドラインミスを起こす場合のみであり，RM でスケジュール可能なタスクセットは RM と RMZL ではまったく同じようにスケジュールされる．この性質から，RMZL は，RM でスケジュールできない場合を除いては，予測性が高い，タスクのジッタが小さいなどの RM の長所をあわせ持つアルゴリズムである．よって，制御タスクなど正確なタイミングが求められるハードリアルタイムタスクと利用率が高く高処理能力が求められるソフトリアルタイムタスクが混在するシステムに有効なアルゴリズムであると考えられる．一方，従来のグローバル RM アルゴリズムである RM-US は，RM でスケジュール可能なタスクセットがスケジュール不能になる場合がある．この例として RM および RM-US によるタスクセット $\tau = \{\tau_1 = (1, 2), \tau_2 = (3, 4), \tau_3 = (3, 4)\}$ のスケジュールを**図 3** に示す．RM スケジュールでは周期の短い τ_1 が先にスケジュールされ，結果としてデッドラインをミスすることなく実行できるが，RM-US スケジュールでは利用率の高いタスクである τ_2 と τ_3 が先に実行されてしまい，結果 τ_1 がデッドラインミスを起こしていることが分かる．

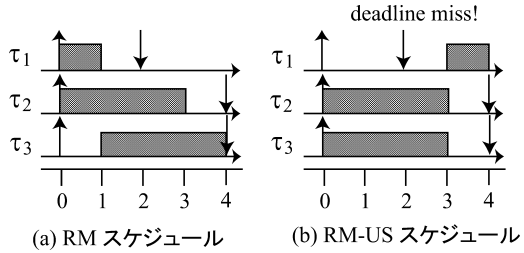

図 3　(a) RM スケジュールと (b) RM-US スケジュールの例
Fig. 3　Examples of (a) RM schedule and (b) RM-US schedule.

同じゼロ余裕時間を用いている EDZL と RMZL を比較すると，RMZL は静的優先度アルゴリズムを基にしているため，予測性が高い．また，現時点でのスケジュール可能性判定による評価では，RM の方が EDF よりもより多くのタスクセットをスケジュール可能であると判定することができる[13]．よって，RMZL の方が EDZL よりも多くのタスクをスケジュール可能であると判定することができると考えられる．シミュレーションによる評価でこのことを示す．

5. スケジュール可能性解析

本章では，RMZL のスケジュール可能性判定条件について述べる．以下の解析はグローバルスケジューリング方式アルゴリズムの反応時間解析（RTA）[13] および EDZL のスケジュール可能性解析[14] を基にしている．

解析のために 2 つの用語を定義する．これらの用語はマルチプロセッサのスケジュール可能性解析においてしばしば用いられる．

定義 1（干渉長）．区間 $[a,b)$ におけるタスク τ_k への干渉長 $I_k(a,b)$ は，τ_k が m 個以上の高優先度タスクによってブロックされ実行できない区間の合計長を表す．また，区間 $[a,b)$ におけるタスク τ_k へのタスク τ_i の干渉長 $I_k^i(a,b)$ は，τ_k が τ_i にブロックされ実行できない区間の合計長を表す．

定義 2（仕事量）．区間 $[a,b)$ におけるタスク τ_k の仕事量 $W_k(a,b)$ は，与えられたスケジュールにおいてタスクが区間 $[a,b)$ で実行しなければならない実行量を表す．

干渉長に関する補題を以下に示す．

補題 1. すべてのグローバルスケジューリング方式のアルゴリズムに対して下式が成り立つ．

$$I_k(a,b) \geq x \Leftrightarrow \sum_{i \neq k} min\left(I_k^i(a,b), x\right) \geq mx$$

証明．EDF のスケジュール可能性解析に関する論文[15] の補題 4 の証明と同じである． □

解析の大まかな流れを以下に示す．各タスク τ_k が最大反応時間を得るときのジョブを J_k^* とする．まず，ジョブ J_k^* について，区間 $[r_k^*, r_k^* + R_k^{ub}]$ における τ_k への干渉長 I_k の上限値を求める．そして干渉長の上限値 I_k^{ub} と τ_k の実行時間から，τ_k の反応時間の上限値 R_k^{ub} を求める．τ_k のジョブの余裕時間の下限値 L_k^{lb} は $R_k^{ub} - T_k$ で求められる．RMZL では，$m+1$ 個以上のジョブの余裕時間が 0 になると，1 つ以上のタスクが実行できず余裕時間が負となり，デッドラインミスを起こしてしまう．よって，RMZL でデッドラインミスが起きる必要条件は，$m+1$ 個のタスク τ_k について $L_k^{lb} <= 0$ が成り立ち，かつそのうちの 1 つのタスクについて $L_k^{lb} < 0$ が成り立つことである．

では，まず干渉長の上限値 I_k^{ub} を求める．

補題 2. 区間 $[a,b)$ におけるタスク τ_i の τ_k への干渉長 $I_k^i(a,b)$ は，$[a,b)$ における τ_i の仕事量 $W_i(a,b)$ を超えない．

証明．タスクは実行しているときのみ他タスクに干渉する．よって干渉長と仕事量の定義から明らか． □

そこで，干渉するタスク τ_i の仕事量の上限値 W_i^{ub} を求める．

補題 3. m 個のプロセッサ上で，タスクセット $\tau = \{\tau_1, \ldots, \tau_n\}$ を RMZL でスケジュールすると，タスク τ_k を干渉するタスク τ_i の区間 $[r_k^*, r_k^* + R_k^{ub})$ における仕事量 $W_i^{ub}(r_k^*, r_k^* + R_k^{ub})$ は下式で求められる．

$$W_i^{ub}(r_k^*, r_k^* + R_k^{ub}) = W_i^{ub}(R_k^{ub})$$
$$= \begin{cases} n_i(R_k^{ub})C_i & \\ + \min(C_i, R_k^{ub} + T_i - C_i - n_i(R_k^{ub})T_i) & (i < k) \\ C_i & (i > k) \end{cases} \quad (1)$$

ただし，

$$n_i(R_k^{ub}) = \left\lfloor \frac{R_k^{ub} + T_i - C_i}{T_i} \right\rfloor \quad (2)$$

とする．

証明．最初に，干渉されるタスク τ_k より周期の長いタスク τ_i（$i > k$）について見ていく．RMZL スケジュールでは，τ_i は，余裕時間が正のときは τ_k より優先度が低いため，τ_i が τ_k をブロックするためには，余裕時間が 0 になり，ジョブが最高優先度になる必要がある．したがって，τ_i の仕事量が最大となるためには，ジョブが周期の最後に C_i だけ実行される必要がある．このとき．**図 4** から分かるように，区間 $[r_k^*, r_k^* + R_k^{ub})$ における τ_i の仕事量（図 4 の斜線部分）は，$T_i \geq T_k$ から，必ず C_i 以下である．よって，仕事量 $W_i^{ub}(r_k^*, r_k^* + R_k^{ub})$ は下式を満たす．

$$W_i(r_k^*, r_k^* + R_k^{ub}) \leq C_i \quad (3)$$

次に，タスク τ_k より周期の短いタスク τ_i（$i < k$）について見ていく．リリース時刻が区

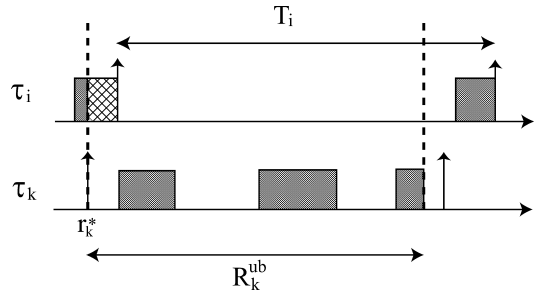

図 4　$i > k$ の場合
Fig. 4　Case in which $i > k$.

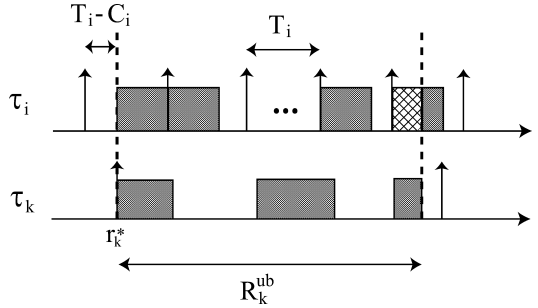

図 5　$i < k$ の場合
Fig. 5　Case in which $i < k$.

間より前で，デッドラインが区間内にあるジョブの仕事量の合計は，それらのジョブが周期の最後に C_i すべて実行されたときが最も大きい．また，リリース時刻が区間内で，デッドラインが区間より後にあるジョブの仕事量は，周期の最初に C_i すべて実行されたときが最も大きい．このとき，タスクの仕事量が最大となるのは，**図 5** のように，リリース時刻が区間より前で，デッドラインが区間内にあるジョブのリリース時刻とジョブ J_k^* のリリース時刻 r_k^* が一致する場合である．

このとき，$n_i(R_k^{ub})$ を区間 $[r_k^*, r_k^* + R_k^{ub})$ において最悪実行時間 C_i をすべて実行できるジョブの数とすると，図 5 より n_i は以下で計算できることが分かる．

$$n_i(R_k^{ub}) = \left\lfloor \frac{R_k^{ub} + T_i - C_i}{T_i} \right\rfloor$$

一方，区間の最後のジョブの仕事量（図 5 の斜線部分）は $\min(C_i, R_k^{ub} + T_i - C_i - n_i(R_k^{ub})T_i)$ である．よって，タスク τ_i の仕事量は，下式を満たす．

$$W_i(r_k^*, r_k^* + R_k^{ub}) \leq n_i(R_k^{ub})C_i + \min(C_i, R_k^{ub} + T_i - C_i - n_i(R_k^{ub})T_i) \quad (4)$$

以上，式 (3)，(4) より，補題が示された．□

補題 3 より，区間 $[r_k^*, r_k^* + R_k^{ub})$ における干渉長の上限値 $I_k^i(R_k^{ub})$ が求められる．しかしながら，干渉長の定義より，タスク τ_k は $R_k^{ub} - C_k$ より長い干渉は受けない．そうでなければ，τ_k の反応時間は R_k^{ub} を超えてしまう．このことから，以下の補題が示される．

補題 4. タスク τ_k の反応時間は，下式を満たせば R_k^{ub} を超えない．

$$\sum_{i \neq k} \min(I_k^i(R_k^{ub}), R_k^{ub} - C_k + 1) < m(R_k^{ub} - C_k + 1) \quad (5)$$

証明．式 (5) が成り立つとすると，補題 1 より，下式が成り立つ．

$I_k(R_k^{ub}) < (R_k^{ub} - C_k + 1)$

したがって，J_k^* は $R_k^{ub} - C_k$ より長い干渉を受けない．よって，干渉長の定義より，J_k^* は R_k^{ub} までに実行を完了することができる．□

補題 4 より，τ_i の干渉長の上限値は $min(W_i^{ub}, R_k^{ub} - C_k + 1)$ となる．以上から，RMZL におけるタスクの反応時間の上限値が求められる．

定理 1（RTA for RMZL）．RMZL でスケジュールされたマルチプロセッサシステムのタスク τ_k の反応時間の上限値は，下式の R_k^{ub} に関して $R_k^{ub} = C_k$ から始まる不動点反復法を解くことによって求められる．

$$R_k^{ub} \leftarrow C_k + \left\lfloor \frac{1}{m} \sum_{i \neq k} \hat{I}_k^i(R_k^{ub}) \right\rfloor$$

ただし，

$\hat{I}_k^i(R_k^{ub}) = \min(W_i^{ub}(R_k^{ub}), R_k^{ub} - C_k + 1)$

であり，$W_i^{ub}(R_k^{ub})$ は式 (1) で求められる値である．

証明．背理法により証明する．収束した R_k^{ub} の値より τ_k の反応時間が大きいと仮定する．収束した R_k^{ub} の値は以下で求められる．

$$R_k^{ub} = C_k + \left\lfloor \frac{1}{m} \sum_{i \neq k} \min(W_i^{ub}(R_k^{ub}), R_k^{ub} - C_k + 1) \right\rfloor$$

補題 2 より，$W_i^{ub}(R_k^{ub}) \geq I_k^i(r_k^*, r_k^* + R_k^{ub})$ が成り立つ．したがって，

$$R_k^{ub} \geq C_k + \left\lfloor \frac{1}{m} \sum_{i \neq k} \min(I_k^i(R_k^{ub}), R_k^{ub} - C_k + 1) \right\rfloor$$

仮定より，補題 4 の逆から，下式がいえる．

$$R_k^{ub} \geq C_k + \left\lfloor \frac{1}{m} m(R_k^{ub} - C_k + 1) \right\rfloor = R_k^{ub} + 1$$

この式は矛盾しており，よって背理法より定理が証明された．□

RMZL では，$m+1$ 個以上のジョブの余裕時間が 0 になると，1 つ以上のタスクが実行できず余裕時間が負となり，デッドラインミスを起こしてしまう．よって，RMZL でデッドラインミスが起きる必要条件は，$m+1$ 個のタスク τ_k について $L_k^{lb} \leq 0$ が成り立ち，かつ

そのうちの 1 つのタスクについて $L_k^{lb} < 0$ が成り立つことである．このことから，RMZL のスケジュール可能性解析が導かれる．

定理 2（RMZL test）．タスクセット $\tau = \{\tau_1, \ldots, \tau_n\}$ は，（少なくとも $m+1$ 個のタスクが下式を満たし，かつそのうちの *1* つのタスクが下式の不等号 $<$ を満たす）ことがなければ，m 個のプロセッサシステムにおいて RMZL によりスケジュール可能である．

$$L_k^{lb} = T_k - R_k^{ub} \leq 0$$

ただし R_k^{ub} は定理 1 で求められる値である．

定理 2 の判定は改善することができる．干渉するタスク τ_i の反応時間が求まっているならば，τ_i の余裕時間の下限値 L_i^{lb} を考慮することにより，τ_i の干渉長を少なく見積もることができる．**図 6** は，L_i^{lb} を考慮した場合の図である．式 (1) と (2) は以下のように書き換えられる．

$$W_i^{ub}(r_k^*, r_k^* + R_k^{ub}) = W_i^{ub}(R_k^{ub})$$
$$= \begin{cases} n_i(R_k^{ub})C_i + \min(C_i, & \\ R_k^{ub} + T_i - C_i - L_i^{lb} - n_i(R_k^{ub})T_i) & (i < k) \\ C_i & (i > k) \end{cases} \quad (6)$$
$$n_i(R_k^{ub}) = \left\lfloor \frac{R_k^{ub} + T_i - C_i - L_i^{lb}}{T_i} \right\rfloor$$

よって，改善された RMZL の RTA とスケジュール可能性判定が示される．

定理 3（Refined RTA for RMZL）．RMZL でスケジュールされたマルチプロセッサシステムのタスク τ_k の反応時間の上限値は，下式の R_k^{ub} に関して $R_k^{ub} = C_k$ から始まる不動点反復法を解くことによって求められる．

$$R_k^{ub} \leftarrow C_k + \left\lfloor \frac{1}{m} \sum_{i \neq k} \hat{I}_k^i(R_k^{ub}) \right\rfloor$$

ただし，

$$\hat{I}_k^i(R_k^{ub}) = \min(W_i^{ub}(R_k^{ub}), R_k^{ub} - C_k + 1)$$

であり，$W_i^{ub}(R_k^{ub})$ は式 (6) で求められる値である．

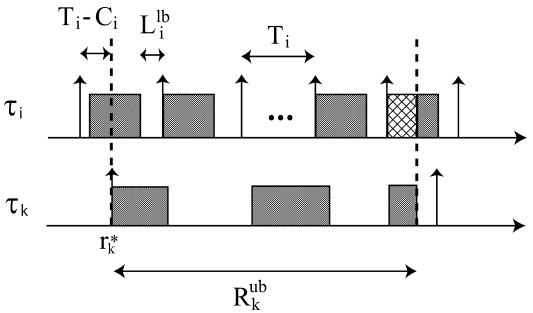

図 6　$i < k$ の場合（改善版）
Fig. 6　Case in which $i < k$ (refined version).

定理 4（Refined RMZL test）．タスクセット $\tau = \{\tau_1, \ldots, \tau_n\}$ は，（少なくとも $m+1$ 個のタスクが下式を満たし，かつそのうちの *1* つのタスクが下式の不等号 $<$ を満たす）ことがなければ，m 個のプロセッサシステムにおいて RMZL によりスケジュール可能である．

$$L_k^{lb} = T_k - R_k^{ub} \leq 0$$

ただし R_k^{ub} は定理 3 で求められる値である．

6. タスクのデッドラインからの遅延時間

ヒューマノイドロボットのようなハードリアルタイムタスクとソフトリアルタイムタスクが混在するシステムでは，ハードリアルタイムタスクのスケジュール可能性を保証すると同時に，ソフトリアルタイムタスクの遅延時間の上限値を解析できることが望ましい．マルチプロセッサシステムにおいてタスクの遅延時間の上限値を求める研究は数多く行われている．

遅延時間の上限値は，前節で求めた反応時間の上限値を用いて求めることができる．

定理 5（Tardiness bound for RMZL）．*RMZL* でスケジュールされたマルチプロセッサシステムのタスク τ_k の遅延時間の上限値は，下式で求められる．

$$tardiness(\tau_k) = R_k^{ub} - T_k$$

ただし，R_k^{ub} は定理 *3* で求められる値である．

7. シミュレーション評価

本章では，RMZL が大きなオーバヘッドを要することなく高いスケジュール可能性を達成できることを示す．スケジュール可能性に関する評価指標は下式で表されるスケジュール成功率（Success Ratio）とする．

$$\text{Success Ratio} = \frac{\#\text{ of successfully scheduled task sets}}{\#\text{ of scheduled task sets}}$$

また，オーバヘッドに関する評価指標はタスクのプリエンプション数とする．評価対象のアルゴリズムとしては，静的優先度グローバルスケジュール方式である RM および RM-US，静的優先度パーティショニング方式である RM-FFDU（First Fit in Decreasing Utilization）とする．また，スケジュール可能性の評価ではゼロ余裕時間ルールを用いた動的優先度グローバルスケジューリング方式である EDZL を，プリエンプション数の評価では最適スケジューリングアルゴリズムである LNREF，EKG を評価対象に含めることとする．代表的な最適アルゴリズムである Pfair は理論的に最もプリエンプション数が多く実用性が低いと考えられるため，評価対象には含めないこととする．

7.1 評価環境

シミュレーションは，m，U_{max}，U_{min}，U_{total} の 4 つのパラメータによって決定する．m はプロセッサ数，U_{max} と U_{min} は各々与えられるタスクセットに含まれるタスクの利用率の最大値と最小値である．U_{total} はタスクセットに含まれるタスクの利用率の合計である．システム利用率（System Utilization）は U_{total}/m と定義する．1 つの m，U_{max}，U_{min} の組合せに対して，システム利用率 30%から 100%まで各々1,000 個のタスクセットを投入して，スケジュール成功率を計測した．これらのパラメータに関しては様々な組合せが考えられるが，現在はデュアルコアが一般的になってきているため，将来の組み込み用のプラットフォームの規模を考慮して，プロセッサ数は 4，8，16 の 3 通りとした．冒頭で述べたように，本論文で主に対象としているヒューマノイドロボットでは制御タスクやマルチメディア処理を行うタスク，イベントタスクなど様々なタスクが存在する．したがって各タスクの利用率の範囲は $(U_{max}, U_{min}) = (1.0, 0.01)$ とした．タスクセット τ は以下のように生成した．$U(\tau) \leq U_{total}$ である限り，新しいタスクを τ に追加していく．各タスク τ_i の利用率 U_i は $[U_{max}, U_{min}]$ の範囲で一様分布で生成した．最後に生成されるタスクの利用率のみ，$U(\tau) = U_{total}$ となるように調節した．ヒューマノイドロボットでは，制御タスクは 1 ms 以下のサイクルで実行され，ソフトリアルタイムタスクは 33 ms ほどのフレーム周期で実行される．よって，周期 T_i を $[100, 3000]$ の範囲の一様分布で生成した．タスクの実行時間 C_i は $C_i = U_i T_i$ として得られる．シミュレーション時間は $[0, 1000000)$ とした．

タスクセットのスケジュール成功の定義を以下に述べる．パーティショニング方式の RM-FFDU は，タスク割当てが成功したらデッドラインをミスすることなくスケジューリングできるアルゴリズムである．よって，タスクセットに含まれるタスクをすべてプロセッサに割り当てることができたならばスケジュール成功であると定義した．タスクをプロセッサに割り当てる際に考慮するプロセッサごとの利用率上限は，タスク数を n として，$U_{ub} = n(2^{1/n} - 1)$ で求められる上限値[4] を用いた．一方，グローバルスケジューリング方式の RM，RM-US，RMZL，EDZL は，アルゴリズムのスケジュール可能性判定でスケジュール可能とされた場合にスケジュール成功とする方法と，実際にタスクセットをスケジュールしデッドラインミスを起こすことなくシミュレーションを終了した場合にスケジュール成功とする方法の 2 つがある．本評価では両方の定義を用い，たとえば RM では前者の定義を用いた場合のスケジュール成功率を RM-test，後者の定義を用いた場合のスケジュール成功率を RM-sim という具合に表記することとした．RM と RM-US のスケジュール可能性判定は Baker[16] の判定法を用いた．すなわち，タスクセットは $\sum_i U_i \leq (m/2)(1 - U_{max}) + U_{max}$ を満たせば m 個のプロセッサ上で RM によりスケジュール可能である．また，タスクセットは m 個のプロセッサにおいて，利用率が λ より高いタスクが k 個あり，それら以外のタスクの合計利用率が $((m-k)/2)(1-\lambda) + \lambda$ 以下であれば，RM-US[λ] でスケジュール可能である．本評価では，RM-US[$m/(3m-2)$] を用いた．また，RMZL のスケジュール可能性判定は定理 4 を用いた．EDZL のスケジュール可能性判定は Cirinei ら[14] の判定法を用いた（複雑であるため，詳細は述べない）．

7.2 スケジュール可能性に関する評価結果

シミュレーション結果を**図 7** に示す．RM-test は，他のアルゴリズムに比べてスケジュール成功率が非常に低かったため省略してある．まず，実際にスケジュールしたときの結果について見ると，RMZL-sim と EDZL-sim はどのプロセッサ数においても最もスケジュール成功率が高く，RMZL-sim と EDZL-sim のスケジュール成功率はほとんど変わらなかった．このことは，ゼロ余裕時間ルールの有効性が RM を基にした場合でも変わらないことを示している．また，他のアルゴリズムの成功率がプロセッサ数の増加にともない減少するのに対して，RMZL-sim と EDZL-sim のスケジュール成功率は逆に増加していた．これは，プロセッサ数が増えることで，同時に存在できるゼロ余裕時間のタスクが増加し，よりデッドラインミスしにくくなったためだと考えられる．ここで，RMZL および EDZL では同時に存在するゼロ余裕時間のタスクがプロセッサ数 m を超えるとデッドラインミスすることに注意されたい．一方，RM-US-sim は，特に $m = 4$ の場合で，RM-sim よりスケジュール成功率が低下することがあった．これは RM-US が，RM でスケジュールできないタスクセットをスケジュール可能にする半面，RM でスケジュールできるはずの多くのタスクをスケジュール不能にしてしまうことを示している．この点で，RMZL が RM-US よりも優れていることが分かる．実際にスケジュールした場合で見ると，パーティショニング方式である RM-FFDU が最もスケジュール成功率が低かった．このことは，ソフトリアルタイムシ

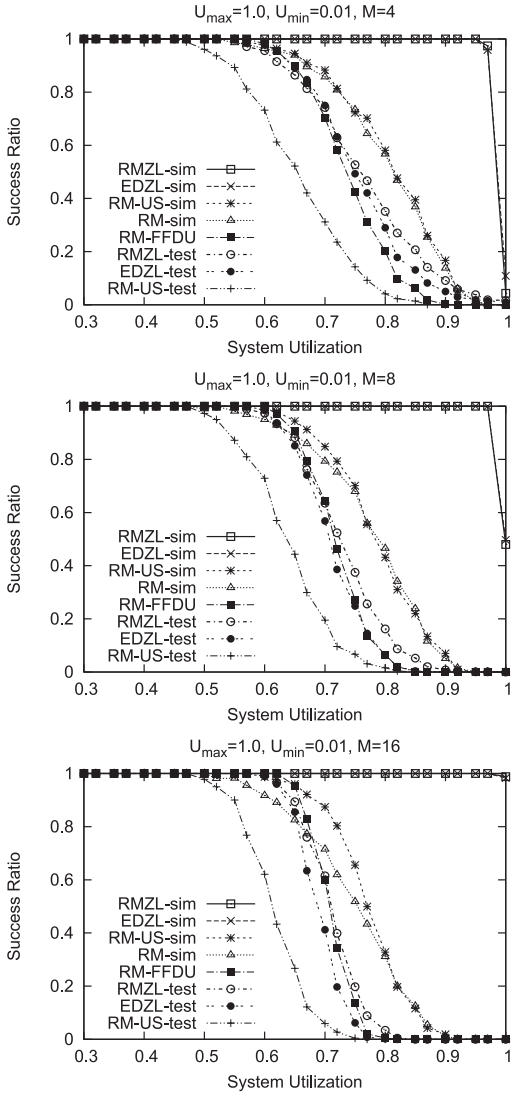

図 7　システム利用率に応じたスケジュール成功率
Fig. 7　Success ratio as a function of system utilization.

ステムにおけるグローバルスケジューリング方式アルゴリズムの優位性を示している．

一方，スケジュール可能性判定を行った場合の結果を見ると，グローバルスケジューリング方式のすべてのアルゴリズムは，実際にスケジュールした場合よりも大幅にスケジュール成功率が減少していた．このことは，グローバルスケジューリング方式の RM および EDF を基にしたアルゴリズムのスケジュール可能性判定が非常に悲観的なものであることを示している．RMZL においてもそれは例外ではなく，また実際にスケジュールした場合の結果と異なりプロセッサ数によって成功率が減少していた．このことから，より厳密なスケジュール可能性判定は今後の課題とする．しかしながら RMZL-test のスケジュール成功率は，他のすべての RM に基づくグローバルスケジュール方式アルゴリズムの成功率よりも高く，また RM-FFDU の成功率とほとんど同じであった．このことは，グローバルスケジューリング方式が有効でないと考えられていたハードリアルタイムシステムにも RMZL が有効であることを示している．また，同じゼロ余裕時間ルールを適用している EDZL-test よりも RMZL-test の方が高いスケジュール可能性を達成できた．

7.3　タスクプリエンプション数の評価結果

次に各アルゴリズムのプリエンプション数について議論する．プリエンプション数のシミュレーション結果を**図 8** に示す．公平のため，実際にスケジュールした場合にスケジュール成功率が 100%であるシステム利用率でのプリエンプション数のみプロットしてある．最適アルゴリズムである LNREF を EKG を比べると，EKG の方がプリエンプション数は少なかった．しかしながら，最適アルゴリズムの中で最もプリエンプション数が少ない EKG においても，プロセッサが 16 個のときに最大で RMZL の 11 倍ものプリエンプションが発生する結果となった．最適アルゴリズムはタスク数にプリエンプション数が比例する傾向にあるため，よりプロセッサ数とタスク数が増えた場合にはこの倍率をさらに上回ると考えられる．将来搭載されるプロセッサがメニーコアへと移行しタスク数が増加した場合にこの性質は問題となる．

一方，RMZL とパーティショニング方式である RM-FFDU と比較すると，プリエンプション数の比は最大で 5.3 倍であった．しかしながら，最適でないグローバルスケジューリング方式アルゴリズムである RM，RM-US プリエンプション数はいずれのプロセッサ数の場合でも RMZL とほとんど変わらなかった．よって，ゼロ余裕時間を適用したことによるコンテキストスイッチのオーバヘッドは無視できることが分かる．

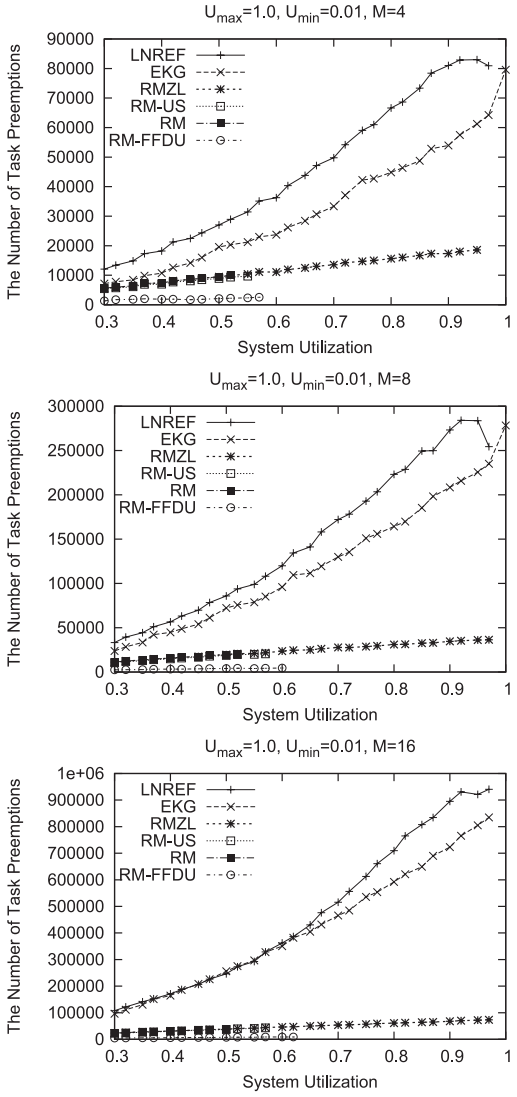

図 8　システム利用率に応じたタスクプリエンプション数
Fig. 8　Number of preemptions as a function of system utilization.

8. 結論および今後の課題

本論文では，マルチプロセッサ上のリアルタイムシステムにおいて，高い予測性や小さいジッタといった静的優先度の長所を継承しながら，スケジュール可能性を向上させるアルゴリズム RMZL を提案した．このアルゴリズムは，ヒューマノイドロボットなどに代表される，正確なタイミングが求められるハードリアルタイムタスクと利用率が高く高処理能力が求められるソフトリアルタイムタスクが混在するシステムに有効なアルゴリズムであるといえる．ハードリアルタイムタスクの保証に必要なスケジュール可能性判定について述べ，またソフトリアルタイムタスクの品質向上に必要な遅延時間の上限値についても述べた．シミュレーションによる評価では，RM-FFDU よりも最大で約 5.3 倍のプリエンプションを発生させてしまうことが分かった．しかしながら，RM および RM-US と比べるとプリエンプション数はほとんど変わらないことから，プリエンプションを多発させることなくスケジュール可能性を向上できたといえる．

以下に今後の課題を示す．評価で見たように，本論文で示したスケジュール可能性判定はグローバルスケジューリング方式の性質とはいえ非常に悲観的なものである．また，本論文では RMZL で利用可能なシステム利用率の上限を解析できていない．スケジューリングアルゴリズムの比較はこの利用率の上限によってなされるため，この解析は重要であると考えられる．また，余裕時間が 0 になったことをどのタイミングでどのように検知するかといった実装上の問題も解決する予定である．

謝辞　本研究の一部は科学技術振興機構 CREST の支援によるものであることを記し，謝意を表す．また，本研究の一部は文部科学省グローバル COE プログラム「環境共生・安全システムデザインの先導拠点」によるものであることを記し，謝意を表す．

参 考 文 献

1) Matsui, T., Hirukawa, H., Yamasaki, N., Ishikawa, H., Kagami, S., Kanehiro, F., Saito, H. and Inamura, T.: Distributed Real-Time Processing for Humanoid Robots, *Proc. 11th International Conference on Embedded and Real-Time Computing Systems and Applications*, pp.205–210 (2005).
2) Tullsen, D.M., Eggers, S.J. and Levy, H.M.: Simutaneos Multithreading: Maximizing On-Chip Parallelism, *Proc. 22nd Annual International Symplsium on Computer Architecture*, pp.392–403 (1995).
3) Olukotun, K., Nayfe, B., Hammond, L., Wilson, K. and Chang, K.: The Case

for a Single-Chip Multiprocessor, *Proc. International Conference on Architectural Support for Programming Languages and Operating Systems*, pp.2–11 (1996).
4) Liu, C. and Layland, J.: Scheduling algorithms for multiprogramming in a hard real-time environment, *J. ACM*, Vol.20, pp.46–61 (1973).
5) Dhall, S.K. and Liu, C.L.: On a real-time scheduling proglem, *Operations Research*, Vol.26, pp.127–140 (1978).
6) Cho, S., Lee, S.K., Han, A. and Lin, K.J.: Efficient Real-Time Scheduling Algorithms for Multiprocessor Systems, Vol.E85-B, pp.2859–2867 (2002).
7) Andersson, B. and Jonsson, J.: The Utilization Bounds of Partitioned and Pfair Static-Priority Scheduling on Multiprocessors are 50%, *Proc. 15th Euromicro Conference on Real-Time Systems*, pp.33–40 (2003).
8) Baruah, S., Cohen, N., Plaxton, C.G. and Varvel, D.: Proportionate progress: A notion of fairness in resource allocation, *Algorithmica*, Vol.15, pp.600–625 (1996).
9) Cho, H., Ravindran, B. and Jensen, E.D.: An Optimal Real-Time Scheduling Algorithm for Multiprocessors, *Proc. 27th IEEE Real-Time Systems Symposium*, pp.101–110 (2006).
10) Andersson, B. and Tovar, E.: Multiprocessor Scheduling with Few Preemptions, *Proc. 12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pp.322–334 (2006).
11) Andersson, B., Baruah, S. and J.J.: Static-priority Scheduling on Mutiprocessors, *Proc. 22nd IEEE Real-Time Systems Symposium*, pp.193–202 (2001).
12) Leung, L.Y.: A New Algorithm for Scheduling Periodic, Real-Time Tasks, *Algorithmica*, Vol.4, pp.209–219 (1989).
13) Bertogna, M. and Cirinei, M.: Response-Time Analysis for globally scheduled Symmetric Multiprocessor Platforms, *Proc. 28th IEEE International Real-Time System Symposium*, pp.149–158 (2007).
14) Cirinei, M. and Baker, T.P.: EDZL Scheduling Analysis, *Proc. 19th Euromicro Conference on Real-Time Systems*, pp.9–18 (2007).
15) Bertogna, M., Cirinei, M. and Lipari, G.: Improved Schedulability Analysis of EDF on Multiprocessor Platforms, *Proc. 17th Euromicro Conference on Real-Time Systems*, pp.209–218 (2005).
16) Baker, T.P.: Multiprocessor EDF and Deadline Monotonic Schedulability Analysis, *Proc. 24th IEEE Real-Time Systems Symposium*, pp.120–129 (2003).

(平成 20 年 7 月 23 日受付)
(平成 20 年 10 月 3 日採録)

武田　瑛

1984 年生．2007 年慶應義塾大学理工学部情報工学科卒業．現在，同大学大学院理工学研究科開放環境科学専攻修士課程在籍．リアルタイムシステム，オペレーティングシステム等の研究に従事．

加藤　真平（正会員）

1982 年生．2004 年慶應義塾大学理工学部情報工学科卒業．2006 年同大学大学院理工学研究科開放環境科学専攻修士課程修了．博士（工学）．現在同大学訪問研究員．リアルタイムシステム，オペレーティングシステム等の研究に従事．

山崎　信行（正会員）

1966 年生．1991 年慶應義塾大学理工学部物理学科卒業．1996 年同大学大学院理工学研究科計算機科学専攻博士課程修了．博士（工学）．同年電子技術総合研究所入所．1998 年 10 月慶應義塾大学理工学部情報工学科助手．同専任講師を経て 2004 年 4 月より同准教授．現在，産業技術総合研究所特別研究員を兼務．並列分散処理，リアルタイムシステム，システム LSI，ロボティクス等の研究に従事．