SMT プロセッサにおける実行効率を向上するリアルタイムスケジューリング

加藤真平[†] 小林秀典[††] 山崎信行[†]

SMT プロセッサでは，ハードウェア資源の競合によって実行効率が変動するので，リアルタイムタスクのスケジュールが容易ではない．また，同時に実行するタスクの組合せによっては，実行効率が低下してしまい，スループットを十分に向上できない場合も考えられる．そこで，本論文では，SMT プロセッサにおけるリアルタイム性の保証およびスループットの向上を実現するために，実行効率を考慮するリアルタイムスケジューリング方式を提案し，その有効性を評価する．提案方式である U-Link スケジューリングは，周期タスクの特性を利用して，つねに同じタスクどうしを同時に実行することで，実行効率の変動を抑制する．また，同時に実行するタスクの組合せを決定する際に，実行効率が高くなる組合せを選択することでスループットを向上する．本論文では，さらに，これまで困難とされてきた SMT 実行におけるタスクの実行時間の予測およびスケジュール可能性についても述べる．評価の結果，従来のリアルタイムスケジューリングアルゴリズムに比べて，U-Link スケジューリング方式のアルゴリズムである UL-DEDF が，実行効率の変動を抑制し，デッドラインミスを起こすことなくスループットを最大 40%向上できたことを示す．

Real-Time Scheduling to Increase Execution Efficiency on SMT Processors

Shinpei Kato,[†] Hidenori Kobayashi[††] and Nobuyuki Yamasaki[†]

It is not straightforward to schedule real-time tasks on SMT processors, since the execution efficiency fluctuates due to hardware resource competition. Also, some combinations of the co-scheduled tasks degrade the execution efficiency and it cannot increase throughput sufficiently. This paper proposes a real-time scheduling scheme for SMT processors which deals with the execution efficiency to increase throughput without missing the deadline, then evaluates its effectiveness. The proposed scheme, U-Link Scheduling, bounds the fluctuation of the execution efficiency by limiting the combination of the co-scheduled tasks using the characteristic of periodic tasks. It increases throughput by choosing the combinations of the co-scheduled tasks so as to boost the execution efficiency when it builds the combinations. Also, this paper describes how to estimate the execution time of tasks and provide the schedulability analysis. The evaluation result shows that UL-DEDF, an algorithm of U-Link Scheduling, bounded the fluctuation of system efficiency, then increased throughput about 40% over the existing real-time scheduling algorithms without missing the deadline.

1. はじめに

リアルタイムシステムでは，周期タスクのデッドラインを保証するために，リアルタイムスケジューリングが重要となる．このとき，スケジューリングアルゴリズムだけでなく，与えられたタスクセットがスケジュール可能か否かを判定できる手法が必要である．また，状況によっては，システムに新たに到着したタスクの受入れを制御する機構[15]も必要になる．さらに，近年では，システム全体のスループットも要求される場面が増えている．ヒューマノイドロボットを例にとると，センサ/アクチュエータ制御といった時間を厳密に守る必要のあるハードリアルタイムタスクと，画像処理や自然言語処理といったスループットを重視するソフトリアルタイムタスクが混在している．それゆえ，限られた資源量および低消費電力という制限のもと，リアルタイム性および高スループットが同時に要求される．

システムのスループットを向上するために実行の並列度を高めるプロセッサ技術として Simultaneous

† 慶應義塾大学
Keio University
†† キャノン株式会社
Canon Corporation

Multithreading（SMT）[19),20)] がある．SMT は，スーパースカラに細粒度マルチスレッディングを統合した実行方式であり，ハードウェア的に複数のスレッドを同時に実行する．本論文では，混乱を避けるために，ハードウェアで同時実行可能なスレッドを論理プロセッサ（LP: Logical Processor）と呼ぶことにする．SMT は，命令バッファやリオーダバッファ，キャッシュ，各種演算器等をスレッド間で共有するため，同じハードウェア資源量で従来のスーパースカラやマルチスレッディングに比べて IPC を 2〜3 倍向上できることが知られている[4)]．SMT 実行方式を利用することで，動作周波数を上げることなくシステムのスループットを向上できるため，近年の組込みリアルタイムシステムにおいても，SMT プロセッサを有効利用することが考えられる．

SMT プロセッサでは，各 LP 間でハードウェア資源の競合が発生し，実行の効率が低下・変動してしまうことが知られている[7),14),16),23)]．リアルタイムシステムにおいては，実行効率の変動は周期タスクの実行時間の変動を引き起こす．つまり，最悪実行時間（WCET: Worst-Case Execution Time）と最良実行時間（BCET: Best-Case Execution Time）の差が大きくなってしまう．リアルタイムスケジューリングは，各周期タスクの WCET を基にスケジュールを行うため，実行時間の変動が大きい場合にはアイドル時間が増加してしまい，システムを効率的に利用することができない．また，スケジュール可能なタスクの数は，予測実行時間の値に大きく依存する．より多くのタスクを処理するためには，タスクの実行時間が短くなる，すなわち実行効率が高くなるタスクの組合せでスケジュールする必要がある．

本論文では，SMT プロセッサにおけるリアルタイム性の保証およびスループットの向上を目的としたリアルタイムスケジューリング方式を提案する．具体的には，SMT プロセッサの実行効率を考慮し，デッドラインミスを起こすことなく，できるだけ多くのタスクを処理することを目標とする．

2. 背景および関連研究

文献 16) では，SMT プロセッサにおいて，システムのスループットを向上させ，レスポンスタイムを短縮するスケジューリング手法として，SOS（Sample, Optimize, Symbiosis）が提案されている．SOS は同時に実行するタスクの組合せを変化させながら，ハードウェアのパフォーマンスカウンタから得られる情報をサンプリングする．そして，得られた情報から最適なタスクの組合せでスケジューリングを行う．このように，論理プロセッサの実行効率を考慮したタスクの組合せでスケジューリングを行うことで，スループットの向上を実現し，レスポンスタイムを最大で 17%改善している．その後，文献 17) において，各タスクが優先度に従って各論理プロセッサのスループットを分配するように SOS を拡張し，システムのスループットを最大で 40%，レスポンスタイムを 33%改善することに成功している．

文献 14) で提案された Thread-Sensitive スケジューリングも，文献 16) と同様に論理プロセッサの実行効率を考慮したスケジューリング手法である．Thread-Sensitive スケジューリングでは，同時に実行する最適なタスクの組合せを選択するために，論理プロセッサの状態をフィードバックしている．フィードバックの対象は，システムにおける性能基準によって異なるが，IPC に基いた Thread-Sensitive スケジューリングは，ラウンドロビンのような従来の手法に比べ最大で 17%のスループット向上を達成している．

文献 23) では，論理プロセッサ間の相性を定義し，相性が良くなるようにタスクをスケジュールしている．具体的には，キャッシュミスが，SMT プロセッサの最も大きな性能低下を引き起こす原因であると考え，キャッシュのヒット率が高くなるようにタスクを切り換えることで，最大 96%の速度向上を実現している．

これらの研究は，システムのスループットに焦点を当てたものであった．また，スループット向上のために，同時に実行するタスクの組合せをシステム実行時に動的に切り替えているが，リアルタイムスケジューリングでは優先度に従ってタスクを切り替える必要があるので，優先度を無視した動的な切替えはリアルタイム性の破綻の原因となり難しいと考えられる．

文献 7) では，ソフトリアルタイムタスクを対象とし，リアルタイムスケジューリングの観点から，SMT プロセッサにおけるスループットおよびデッドラインミス率について考察している．まず，マルチプロセッサで利用されてるグローバルスケジューリング方式およびパーティショニング方式を，SMT プロセッサ用に応用して比較している．これら 2 つの方式の比較は，過去にマルチプロセッサの分野でも議論されているが[2),8),18)]，ここでは，SMT プロセッサの特徴にも着目して比較している．次に，スループットに関して，SMT 特有の Symbiosis Factor [16)] を考慮する方式（Symbiosis-Aware）および Symbiosis Factor を考慮しない方式（Symbiosis-Oblivious）の比較を行っている．グローバルスケジューリング方式のアルゴリズム

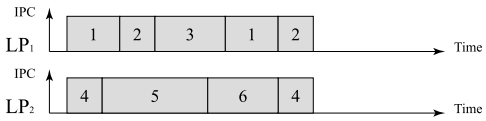


図 1 マルチプロセッサにおけるスケジューリング例
Fig. 1 Scheduling example on multiprocessors.

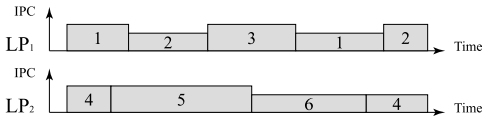


図 2 SMT プロセッサにおけるスケジューリング例
Fig. 2 Scheduling example on SMT processors.

として EDF-US[$\frac{M}{2M-1}$][5]（以下，EDF-US と略す），パーティショニング方式のアルゴリズムとして EDF-FF[12] を用いて評価を行った結果，Symbiosis-Aware 方式とグローバルスケジューリング方式を組み合わせたとき，最も良い性能を示したとしている．しかしながら，実行時間の予測やスケジュール可能性に対する具体的なアプローチは提案していない．また，Symbiosis Factor を求めるためには，SMT 実行における IPC が既知である必要があるが，それを求めるのは非常にコストが高い．

マルチプロセッサを対象としたリアルタイムスケジューリングでは，シングルスレッド実行における予測実行時間を想定して，**図 1** のようなスケジュールを想定する（図中の番号はタスクの番号）．一方，SMT プロセッサでは，実行効率の変動を考慮し，**図 2** のように実行されることを考慮する必要がある．たとえば，タスク 1，タスク 2，タスク 4 は，同時に実行するタスクが異なることから実行効率が変動しているので，シングルスレッド時の予測実行時間を用いることはできない．また，タスク 2 は，タスク 4 と同時に実行する場合と比べて，タスク 6 と同時に実行した場合に実行効率が低下している．スループットを十分に向上するためには，実行効率が高くなるタスクの組合せで実行する必要があることが分かる．EDF-US や EDF-FF 等の既存のリアルタイムスケジューリング手法は，SMT プロセッサにおける実行効率の変動を考慮していないので，リアルタイム性を保証することが難しく，また，スループットも十分に向上することができないと考えられる．

3. U-Link スケジューリング

本論文では，SMT プロセッサにおいて，リアルタイム性を保証し，かつ，実行効率を十分に向上できるリアルタイムスケジューリング方式を提案する．提案手法である U-Link スケジューリングは，周期タスクの利用率（予測実行時間と周期の割合）に基づいて同時に実行するタスクの組合せを固定する．これにより，競合するハードウェア資源の組合せを制限し，実行効率の変動を緩和することができる．また，実行効率が高くなるタスクの組合せで実行することで，安定して高いスループットを保つことができる．

ここで，同時に実行するタスクの組合せを固定しても，入力値や実行パスの変化，キャッシュ等が原因となり，実行効率の変動を完全に抑制することはできないことに注意されたい．我々の知る範囲では，これらの問題を考慮したユニプロセッサの WCET 解析に関しては多くの文献が存在するが[3),6),11),13)]，SMT プロセッサにおいて実行効率の変動を抑制する効果的な手法は提案されていない．実際，SMT 実行方式の仕組みから，実行効率の変動を完全に抑制することは困難である．本論文では，入力値や実行パス，キャッシュ等による実行効率の変動を完全に抑制することは論文の範囲外とする．また，最適なタスクの組合せ，すなわち，実行効率が最大となるタスクの組合せを求めることは NP 完全問題であり，複雑なアルゴリズムが必要になる．本論文では，簡単な計算で済み，かつ，実行効率を十分に向上することを考えているため，最適な組合せを求めることは論文の範囲外とする．

本章では，まず，提案するリアルタイムスケジューリング理論で仮定するシステムモデルを示す．次に，各タスクに対する実行効率を定義し，実行効率を基に予測実行時間を計算する方法を示す．また，実行効率を求めるために，相対命令利用率およびユニット競合率を定義する．そして，同時に実行するタスクを組合せを求めるアルゴリズムおよびその組合せをスケジュールするアルゴリズムを提案し，最終的にスケジュール可能性について述べる．

3.1 システムモデル

本論文で仮定する SMT プロセッサには，M 個の論理プロセッサ LP_1〜LP_M が存在する．演算器等の機能ユニットに関しては，汎用性を考慮して，L 個の種類があるものとし，λ_1〜λ_L で表すことにする．機能ユニット λ_l の数を $N(\lambda_l)$，機能ユニット λ_l のレイテンシを $D(\lambda_l)$ とする．たとえば，整数演算ユニット（ALU）を 4 個，浮動小数点演算ユニット（FPU）を 2 個，分岐ユニット（BU）を 2 個，メモリアクセスユニット（MAU）を 1 個持っている SMT プロセッサがあったとすると，機能ユニットの種類は 4 個なので $L = 4$ となる．また，λ_1，λ_2，λ_3，λ_4 を，それぞれ，ALU，FPU，BU，MAU とすると，$N(\lambda_1) = 4$，

$N(\lambda_2) = 2$，$N(\lambda_3) = 2$，$N(\lambda_4) = 1$ となる．各ユニットの個数は，各ユニットの並列可能な命令数を指している．

システム開始時にはタスクセット τ が与えられる．τ は，N 個の周期タスク ($\tau_1 \sim \tau_N$) の集合であり，ハーモニックとする．すなわち，すべてのタスクの周期は互いに整数倍であるとする．システム実行時に新しいタスクが到着することはない．すべてのタスクは互いに独立に処理を行い，どの時点でもプリエンプト可能である．各タスク τ_i は，(C_i, T_i, I_i) というタプルで表される．C_i は，シングルスレッド実行における実行時間を指す．T_i は，タスクの周期であり，相対デッドラインでもある．I_i は，上述した各機能ユニット λ_l を利用する命令数を返す関数であり，命令関数と呼ばれる．すなわち，$I_i(\lambda_l)$ は，τ_i に含まれる命令のうち，機能ユニット λ_l を利用する命令数である．これらの値は事前に分かっているものとし，各タスクのタプルはシステム設計者が指定するものとする．τ_i のシングルスレッド実行時の利用率は，$U_i = \frac{C_i}{T_i}$ で表し，全体で τ_i が実行する割合を意味する．また，本論文では，シングルスレッド実行時の各タスクの利用率の合計 $U(\tau)$ をシステム負荷と定義する．

$$U(\tau) = \sum_{\tau_i \in \tau} U_i$$

3.2 スケジューリング方式

U-Link スケジューリング（**図 3**）は，**同時実行セット**（Co-scheduled set）と呼ばれるタスクの集合を複数個構築する．任意の k 番目の同時実行セット τ'_k は，$N(\tau'_k)$ 個の要素を含んでおり ($1 \le N(\tau'_k) \le M$)，各要素の利用率がお互いに等しい，または近くなるように構成される．τ'_k に所属する任意の i 番目の要素 $\tau'_{k,i}$ は，1 つ以上のタスクから構成されるタスクの集合であり，**複合タスク**と定義する ($1 \le i \le N(\tau'_k)$)．複合タスクは，$\tau'_{k,i} = (C'_{k,i}, T'_{k,i})$ というタプルで表される．$T'_{k,i}$ は複合タスクの周期であり，式 (1) で求めることができる．$C'_{k,i}$ は複合タスクの SMT 実行における実行時間を表す．$C'_{k,i}$ の求め方については，3.3 節で詳しく述べる．

$$T'_{k,i} = \min\{T_j | \tau_j \in \tau'_{k,i}\} \quad (1)$$

同時実行セット内の A 番目の要素は，**軸タスク**（Axis task）と呼ばれる特別な要素であり，必ず単一のタスクとなる．本論文では $A = 1$ とした．軸タスク $\tau'_{k,A}$ の利用率は，τ'_k に含まれる他の複合タスクよりも大きい．表記の簡略化のため，$\tau'_{k,A}$ を α_k と表記する場合もある．S 個の同時実行セットがあったとすると，S 個の軸タスク $\{\alpha_1, \alpha_2, ..., \alpha_S\}$ があり，その集合を**軸セット**（Axis set）α と定義する．軸タスク以外の複合タスクを一般タスクと呼ぶことにする．

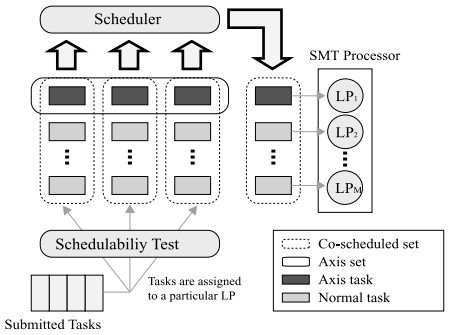


図 3 U-Link スケジューリング
Fig. 3 U-Link Scheduling.

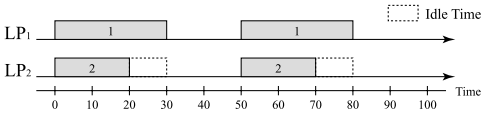


図 4 アイドル時間の発生
Fig. 4 Idle time generation.

同時実行セットの各要素を，単一タスクではなく複合タスクとすることで，U-Link スケジューリングにおけるアイドル時間を削減することができる．たとえば，タスク $\tau_1 = (30, 50, I_1)$ とタスク $\tau_2 = (20, 50, I_2)$ を含む同時実行セットを考える（この例では関数 I は用いない）．**図 4** に示すように，τ_1 の実行中は τ_2 以外のタスクは実行できないので，τ_2 の実行後にアイドル時間が発生してしまう．この場合，$\tau_3 = (5, 25, I_3)$ のような利用率が 1/5 以下のタスクは，τ_2 の利用率が 2/5 なので $5/25 + 20/50 \le 30/50$ となり，τ_2 および τ_3 は複合タスクを構成できる．このように，複合タスクの概念により，アイドル時間を減らすことができ，システムの効率的な利用が可能となる．

スケジューラは，1 つの同時実行セット τ'_k を選択し，その i 番目の複合タスク $\tau'_{k,i}$ を i 番目の論理プロセッサ LP_i で実行する．同時実行セットの構築手法については 3.4 節で，複合タスクのスケジュール方法およびスケジュール可能性については 3.5 節で詳しく述べる．

3.3 実行効率および予測実行時間

本論文では，複合タスク $\tau'_{k,i}$ の中のタスク τ_j の実行効率 $E(\tau_j \in \tau'_{k,i})$ を定義する ($\frac{1}{M} \le E(\tau_j) \le 1$)．SMT プロセッサの実行効率は，同時に実行しているタスクが持つ命令の種類や数およびプロセッサが持つ機能ユニットの種類や数に大きく依存する．そこで，実行効率を求めるために，相対命令利用率およびユニッ

ト競合率を定義する．本節では，相対命令利用率およびユニット競合率について説明し，それらを基に実行効率および予測実行時間を算出する方法を示す．

相対命令利用率 $R_l(\tau_j \in \tau'_{k,i})$ とは，複合タスク $\tau'_{k,i}$ の中のタスク τ_j の全命令の中で，機能ユニット λ_l を利用する命令の割合を，利用率に関して軸タスク α_k により正規化したものである．これにより，ある一定期間内で，τ_j が実行する命令のうち，機能ユニット λ_l を利用する命令の割合を求めることができる．また，複合タスク $\tau'_{k,i}$ に含まれるタスクの相対命令利用率の総和が，複合タスクの相対命令利用率 $R_l(\tau'_{k,i})$ となる．

$$R_l(\tau_j \in \tau'_{k,i}) = \frac{I_j(\lambda_l)}{\sum_{m=1}^{L} I_j(\lambda_m)} \times \frac{U_j}{U_{k,A}} \quad (2)$$

$$R_l(\tau'_{k,i}) = \sum_{\tau_j \in \tau'_{k,i}} R_l(\tau_j) \quad (3)$$

次に，ユニット競合率 $F_l(\tau_j \in \tau'_{k,i})$ とは，複合タスク $\tau'_{k,i}$ に含まれるタスク τ_j に関して，機能ユニット λ_l 度の利用低下率を意味し，式 (4) で求められる．$F_l(\tau_j \in \tau'_{k,i})$ は 1 以上の値であり，1 のときは競合きは起きていないということになる．ただし，同時実行セット τ'_k は，M 個の複合タスクから構成され，$R_l(\tau'_{k,1}) \leq R_l(\tau'_{k,2}) \leq ... \leq R_l(\tau'_{k,M})$ と整列しているものとする ($R_l(\tau'_{k,0}) = 0$)．紙面の都合上，式 (4) の A および B は，それぞれ，以下の式で表すものとする．A は，機能ユニットに対する競合の程度である．B は，競合の程度 A が発生する割合である．下記の例を用いて，ユニット競合率についてもう少し詳しく説明する．

$$F_l(\tau_j \in \tau'_{k,i}) = \min\left\{1, \frac{\sum_{m=1}^{M}(A \times B)}{R_l(\tau_j)}\right\} \quad (4)$$

$$A = \max\left\{1, \frac{M - m + 1}{N(\lambda_l)}\right\}$$

$$B = \Big[\min\{R_l(\tau_j), R_l(\tau'_{k,m})\} - \min\{R_l(\tau_j), R_l(\tau'_{k,m-1})\}\Big]$$

例 1. 5 つの複合タスク $\tau'_{k,1}$ 〜 $\tau'_{k,5}$ が存在し，機能ユニット λ_l に対する相対命令利用率がそれぞれ，$\frac{8}{10}$，$\frac{6}{10}$，$\frac{6}{10}$，$\frac{2}{10}$，$\frac{1}{10}$ であるとする．λ の数は $N(\lambda_l) = 2$ であり，$\tau'_{k,3}$ に含まれるタスク τ_j の相対命令利用率が $R_l(\tau_j) = \frac{4}{10}$ であるとき，τ_j の λ_l に対するユニット競合率 $F_l(\tau_j \in \tau'_{k,3})$ を求めたいとする．式 (4) は，**図 5** のように再帰的な計算を行う．最初の再帰処理では，全体の $\frac{1}{10}$ の割合で，2 個の機能ユニットに対して 5 個のタスク間で競合が発生する．よって，$A = \frac{5}{2}$，$B = \frac{1}{10}$ である．2 回目の再帰処理では，全体の $\frac{1}{10}$ の割合で，2 個の機能ユニットに対して 4 個のタスク間で競合が発生する．よって，$A = \frac{4}{2}$，$B = \frac{1}{10}$ である．同様に，その後の再帰処理を計算すると，$F_l(\tau_j \in \tau'_{k,3}) = \{\frac{5}{2}\times(\frac{1}{10}-0)+\frac{4}{2}\times(\frac{2}{10}-\frac{1}{10})+\frac{3}{2}\times(\frac{4}{10}-\frac{2}{10})+\frac{2}{2}\times(\frac{4}{10}-\frac{4}{10})+1\times(\frac{4}{10}-\frac{4}{10})\}\div\frac{4}{10} = \frac{15}{20}\div\frac{4}{10} = \frac{15}{8}$ となる． □

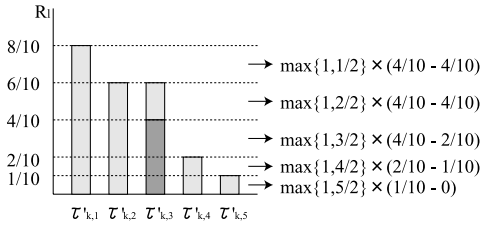


図 5 ユニット競合率
Fig. 5 Unitcompetition ratioy.

タスク τ_j の実行効率は，各命令のレイテンシの総和と，各ユニット競合率に各ユニットのレイテンシを乗じた値の総和との比率であり，式 (5) で表される．

$$E(\tau_j \in \tau'_{k,i}) = \frac{\sum_{l=1}^{L} D(\lambda_l)}{\sum_{l=1}^{L}\{F_l(\tau_j) \times D(\lambda_l)\}} \quad (5)$$

このとき，タスク τ_j の SMT 実行における予測実行時間 $\ddot{C}_j$ は，シングルスレッド実行における予測実行時間および実行効率を用いて式 (6) で求められる．また，利用率 $\ddot{U}_j$ は，式 (7) で求められる．

$$\ddot{C}_j = \frac{C_j}{E(\tau_j \in \tau'_{k,i})} \quad (6)$$

$$\ddot{U}_j = \frac{\ddot{C}_j}{T_j} \quad (7)$$

複合タスク $\tau'_{k,i}$ の予測実行時間 $C'_{k,i}$ および利用率 $U'_{k,i}$ は，それぞれ，式 (8) および式 (9) で求められる．

$$C'_{k,i} = \sum_{\tau_j \in \tau'_{k,i}} \left\{\ddot{U}_j \times T'_{k,i}\right\} \quad (8)$$

$$U'_{k,i} = \frac{C'_{k,i}}{T'_{k,i}} \quad (9)$$

ここで，式 (10) を満たすことに注意されたい．

$$U'_{k,i} = \sum_{\tau_j \in \tau'_{k,i}} \ddot{U}_j \quad (10)$$

同時実行セット τ'_k の平均実行効率は，$\bar{E}(\tau_k)$ と表記され，式 (11) で求めることができる．

Algorithm UL-FFDE
Input: Task set τ
Output: Co-scheduled set τ' and axis set α

Begin
1 : Let $umax(\tau) = (\tau_i : U_i = \max\{U_j | \tau_j \in \tau\})$;
2 : Let $etest(\tau'_{k,i}, \tau_j) = \bar{E}(\{\tau'_{k,l} | l < i\} \cup \{\tau'_{k,i} \cup \tau_j)\})$;
3 : Let $utest(\tau'_{k,i}, \tau'_{k,j}) = (U'_{k,i} : \tau'_k = \tau'_k \cup \tau'_{k,j})$;
4 : $k \leftarrow 1$;
5 : **while** $\tau \neq \emptyset$
6 : $\tau'_{k,1} \leftarrow umax(\tau)$;
7 : $\alpha_k \leftarrow \tau'_{k,1}$;
8 : $\tau = \tau - umax(\tau)$;
9 : **for** $2 \leq i \leq M$
10 : sort $\{\tau_j \in \tau\}$ as $etest(\tau'_{k,i}, \tau_j) \geq etest(\tau'_{k,i}, \tau_{j+1})$;
11 : **for each** $\tau_j \in \tau$
12 : **if** $utest(\tau'_{k,i} \cup \tau_j, \tau'_{k,i} \cup \tau_j) \leq utest(\alpha_k, \tau'_{k,i} \cup \tau_j)$
13 : $\tau'_{k,i} = \tau'_{k,i} \cup \tau_j$;
14 : **end if**
15 : **end for each**
16 : **end for**
17 : $k \leftarrow k + 1$;
18 : **end while**
End

図 6 UL-FFDE アルゴリズム
Fig. 6 The algorithm UL-FFDE.

$$\bar{E}(\tau'_k) = \frac{\sum_{\tau_j \in \tau'_k} C_j}{\sum_{\tau_j \in \tau'_k} \ddot{C}_j} \tag{11}$$

3.4 同時実行セットの構築

同時実行セットは，各要素の SMT 実行時の利用率がお互いに等しい，または近くなるように構成される．スループットを向上するためには，同時実行セットを構築する際に，3.3 で導入した実行効率を考慮する必要がある．本論文で提案するアルゴリズム **U-Link First-Fit in Decreasing Efficiency（UL-FFDE）** は，各同時実行セットの平均実行効率に基づいて同時実行セットを構築する．そのアルゴリズムを**図 6** に示す．

まず，τ の中で最も大きな利用率を持ったタスクを軸タスク $\alpha_k\,(= \tau'_{k,1})$ として選択する（6〜7 行目）．そして，このタスクを集合 τ から除く（8 行目）．次に，一般タスクについて求める．各タスク τ_j に対して，τ_j を複合タスク $\tau'_{k,i}$ に挿入したと仮定し，その場合の τ'_k の平均実行効率が高い順番に τ を整列する（10 行目）．そして，先頭のタスクから順番に以下の作業を行う．タスク τ_j を含めた複合タスク $\tau'_{k,i}$ の SMT 実行における利用率 $U'_{k,i}$ が，軸タスクの利用率以下であれば（12 行目），τ_j を複合タスク $\tau'_{k,i}$ の要素とする（13 行目）．これは，最適組合せ問題における First-Fit 手法に似ている．この作業を繰り返し，1 つの同時実行セットを構築する（9〜16 行目）．そして，次の同時実行セット用にインデックスを移動する（17 行目）．

UL-FFDE は，アルゴリズム自体は簡単であるが，計算量が $O(n^2)$ であり，比較的時間のかかるアルゴリズムである．しかしながら，本論文では新たなタスクが到着しないことを仮定しているので，同時実行セットの構築はシステムの開始時に 1 度だけである．さらに，システム起動前に静的に計算することも可能であるため，UL-FFDE のオーバヘッドがリアルタイムシステムに影響を与えることは考えにくい．また，UL-FFDE は，平均実行効率が良くなる順に整列して，First-Fit で複合タスクおよび同時実行セットを構築している．Best-Fit で挿入してしまうと，利用率に関しては向上が望めるが，平均実行効率が低下してしまい，全体のスループットが低下してしまう恐れがある．文献 22) によると，実際のヒューマノイドロボット等のアプリケーションでは，タスクの種類および周期の組合せはそれほど多くはないので，First-Fit に対して Best-Fit による利用率の劇的な向上は望めない．一方，平均実行効率は，タスクの性質によって大きく低下する可能性があるので，平均実行効率が高い順に First-Fit で挿入する方が適当であると考えられる．また，最適なタスクの組合せを求めるには全数探索のようなアルゴリズムが必要であり，6 行目の UL-FFDE における軸タスクの選択をすべてのタスクに対して試

行する必要がある．全数探索の計算量は $O(n^3)$ となるため，UL-FFDE に比べると現実的ではないと考えられる．以上の検討より，本論文の範囲外であるスループットの最大化を考慮しない範囲では，UL-FFDE は同時実行セットを構築するのに妥当なアルゴリズムである．

3.5 タスクスケジューリング

3.5.1 仮想タスク化

スケジューリングの観点から，同時に実行するタスクの周期および予測実行時間は，すべて均一であることが望ましい．しかしながら，3.4 節で提案した UL-FFDE は，スループットの高い同時実行セットを構築することはできるが，同時実行セット内の複合タスクの周期および予測実行時間がすべて等しくなるとは限らない．これは，UL-FFDE の問題ではなく，複数の周期タスクを同時に実行するというアプローチに対する潜在的な問題であると考えられる．このような場合には，U-Link スケジューリングにおいて，あるタスクの実行時間が各周期で変動してしまう可能性がある．たとえば，$\tau_1 = (10, 40, I_1)$，$\tau_2 = (20, 120, I_2)$，$\tau_3 = (10, 120, I_3)$ が与えられたとき，これらが 1 つの同時実行セットに割り当てられたとする．このとき，**図 7** の τ_1 のように，実際の実行時間が変動してしまう可能性がある．

3.3 節で提案した予測実行時間は，SMT 実行における統計的な平均値となるため，上記の例では $\ddot{C}_1 = 14$ となる．しかしながら，τ_1 の最初の周期では，実行時間は 18 であり，SMT 実行における予測実行時間よりも大きくなってしまっている．実際の実行時間が予測実行時間より大きいと，デッドラインミスが発生する可能性があることは明白である．そこで，本論文では，**仮想タスク化**という概念を導入し，上述の問題に対処する．仮想タスク化とは，利用率は同じであるが，周期および実行時間が元のタスクより短いタスクであると見なすことを意味する．仮想タスクの周期（仮想周期）は，所属する同時実行セット内の最も短いタスク周期となり，仮想予測実行時間はその周期に利用率を乗じた値となる．複合タスク $\tau'_{k,i}$ の仮想周期 $T''_{k,i}$ は式 (12) で表され，仮想予測実行時間 $C''_{k,i}$ は式 (13) で表される．

$$T''_{k,i} = \min\left\{T'_{k,j} | \tau'_{k,j} \in \tau'_k\right\} \tag{12}$$

$$C''_{k,i} = C'_{k,i} \times \frac{T''_{k,i}}{T'_{k,i}} \tag{13}$$

元のタスクの実行は，仮想実行時間が経過すると一度サスペンドされ，次の仮想周期で再び再開される．よって，仮想タスクの実行は元のタスクと等価である．先の例では，τ_2 および τ_3 は周期 40 の仮想タスクと見なされ，連続する 3 つの仮想周期で τ_2 および τ_3 が構成される．仮想タスク化により，図 7 のスケジュールは，**図 8** のようになり，すべてのタスクの実行時間を一定の予測実行時間に抑えることが可能となる．

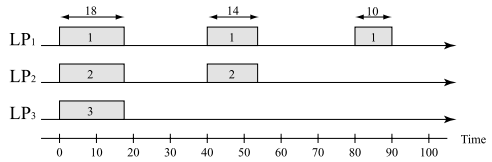


図 7 時間時間の変動
Fig. 7 Fluctuation of execution time.

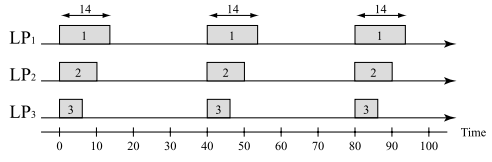


図 8 仮想タスク化
Fig. 8 Pseudo task division.

Algorithm UL-DEDF
Input: The set of axis tasks α
Output: The set of running tasks $\{\gamma_i | i = 1, 2, .., m\}$
Begin
1: **if** there is no ready task in α
2: $\{\gamma_i\} \leftarrow \emptyset$;
3: **else**
4: choose α_i from α based on EDF;
5: $\gamma_1 \leftarrow \alpha_i$;
6: **for each** $\{\tau'^i_k | k > 1\}$
7: **if** there is no ready task in τ'^i_k
8: $\gamma_k \leftarrow \emptyset$;
9: **else**
10: choose τ_j from τ'^i_k based on EDF;
11: $\gamma_k \leftarrow \tau_j$;
12: **end if**
13: **end for each**
14: **end if**
End

図 9 UL-DEDF アルゴリズム
Fig. 9 The algorithm UL-DEDF.

3.5.2 スケジューリングアルゴリズム

U-Link スケジューリングでは，同時実行セットのスケジュールおよび複合タスクのスケジュールが必要になる．本論文で提案する **U-Link Double Earliest Deadline First（UL-DEDF）**では，同時実行セットおよび複合タスクの両方のスケジュールに EDF [10] を利用する．**図 9** に，UL-DEDF のアルゴリズムを示す．

まず，実行可能な軸タスクが存在しなければ，実行するタスクセットは空である（2 行目）．実行可能な軸

タスクが存在すれば，軸セット α から 1 つの軸タスク α_i を EDF に従って選択する（4 行目）．軸タスク α_i が決定されると，同時実行セット τ'_i が得られるので，そのほかの同時実行する複合タスクも選択されたことになる．ここで，複合タスクに実行可能なタスクが存在しなければ，対応する LP における実行タスクは存在しない（8 行目）．実行可能なタスクが存在すれば，複合タスク内から 1 つのタスク τ_j を EDF に従って選択する（10 行目）．

3.5.3 スケジュール可能性

リアルタイム性を保証するためには，スケジューリングアルゴリズムのスケジュール可能性を定式化する必要がある．本節では，UL-DEDF のスケジュール可能性に関する定理を示す．そのために，まず，以下の 2 つの補題を与える．

補題 1. すべての複合タスクを単一のタスクであると仮定する．このとき，タスクセット τ は，式 (14)，式 (15)，式 (16) を満たしていれば，UL-DEDF によってスケジュール可能である．

$$\sum_{\tau'_{k,i}\in\alpha} U'_{k,i} \leq 1 \tag{14}$$

$$\forall k, U'_{k,A} \geq \max\{U'_{k,i} | i \neq a\} \tag{15}$$

$$\forall k, T'_{k,A} \leq \min\{T'_{k,i} | i \neq a\} \tag{16}$$

証明 軸セットは，EDF によってスケジュールされるので，式 (14) が成り立てば，軸セットはスケジュール可能である．よって，一般タスクがデッドラインミスを起こさなければ，タスクセットはスケジュール可能である．U-Link スケジューリングでは，一般タスクは軸タスクと同期して実行されるので，式 (16) より，軸タスク $\tau'_{k,A}$ は，一般タスク $\tau'_{k,i}$ の各周期で $x = \frac{T'_{k,i}}{T'_{k,A}}$ 回リリースされる．同時実行セットに含まれるタスクの周期はハーモニックであるため，$\tau'_{k,A}$ の x 回目の周期の終わりが $\tau'_{k,j}$ のデッドラインになる．軸セットはスケジュール可能なので，$C'_{k,A} \times x \geq C'_{k,i}$ であれば，$\tau'_{k,A}$ に同期して実行する $\tau'_{k,i}$ もスケジュール可能になる．式 (15) より，$C'_{k,A} \times x \geq C'_{k,i}$ は成り立つので，補題は真である． □

補題 2. 補題 1 が成立するとき，$\tau'_{k,i}$ に含まれるすべてのタスクもスケジュール可能である．

証明 軸タスクを $\tau'_{k,A}$ とすると，τ'_k がハーモニックであることから，あるタスク $\tau_j \in \tau_{k,i}$ のデッドラインまでに $\tau'_{k,A}$ が実行する時間は，$C'_{k,A} \times \frac{T_j}{T'_{k,A}}$ である．EDF の性質から，$\{\tau_p | T_p \leq T_j\}$ に対して，式 (17) が成立すれば，$\tau'_{k,i}$ に含まれるすべてのタスクはスケジュール可能となる．

$$C'_{k,A} \times \frac{T_j}{T'_{k,A}} \geq \ddot{C}_i + \sum \ddot{C}_p \times \frac{T_i}{T_p} \tag{17}$$

式 (17) は，以下のように変形される．

$$U'_{k,A} \times T_j \geq \ddot{C}_j + \sum \ddot{U}_p \times T_j$$

$$U'_{k,A} \geq \ddot{U}_j + \sum \ddot{U}_p \tag{18}$$

式 (15) および式 (10) より，式 (18) は成立するので，与えられた補題は真である． □

補題 1 および補題 2 より，UL-DEDF のスケジュール可能性に関して以下の定理を導くことができる．

定理 1. 式 (14)，(15)，(16) をすべて満たすタスクセットは，UL-DEDF によってスケジュール可能である．

UL-FFDE で構築された同時実行セットは，式 (14) および式 (15) を満たす．仮想タスク化により周期が式 (12) となり，式 (16) も満たす．以上より，UL-FFDE で構築された同時実行セットは，仮想タスク化のもとで，UL-DEDF によりスケジュール可能である．

4. 評　　価

本節では，文献 7) でも使われている EDF-FF および EDF-US を比較対象とし，UL-DEDF が，SMT プロセッサにおいて，リアルタイム性を低下させることなく，システムのスループットを向上できたことを示す．具体的には，UL-DEDF におけるタスクの平均実行時間が短縮されたことを示し，それにより，デッドラインミスを起こすことなく EDF-FF および EDF-US より多くのタスクを処理できたことを示す．

4.1 評 価 環 境

評価に使用する SMT プロセッサ[21] の主要な構成を**表 1** に示す．同時に命令をフェッチできるスレッド数が 8 であるが，同時に命令を発行できるスレッド数は 4 であるため，4-way SMT として利用する．コン

表 1 SMT プロセッサの構成
Table 1 Outline of the SMT Processor.

Clock Frequency	100 MHz
Fetch width	8
Issue width	4
Integer register	32-bit × 32-entry × 8-set
Integer renaming register	32-bit × 64-entry
FP register	64-bit × 8-entry × 8-set
FP renaming register	64-bit × 64-entry
ALU	4 + 1 (Divider)
FPU	2 + 1 (Divider)
64-bit ALU	1
FP Vector Units	1 (4 FPU × 2 line)
Branch Unit	2
Memory Access Unit	1

```
/* P is a feedback period.
   Kp, Ki and Kd are constant.*/
void PID_control(input)
{
  e(t) = setpoint - input;
  sum_e(t) = sum_e(t-P) + e(t);
  delta = Kp * e(t)
        + Ki * sum_e(t)
        + Kd * [{e(t) - e(t-P)} / P];
  output = prev_output + delta;
  return output;
}

/* ROW and COL are constant.
   data[][] is a global variable. */
void MEM_access(char img[ROW*COL])
{
  for(i=0; i < ROW; i++)
    for(j=0; j < COL; j++)
      data[i][j] = img[i*COL + j];
}
```

図 10 評価に用いるタスク
Fig. 10 Tasks for the evaluation.

テキストスイッチのオーバヘッドは考慮に入れない．

評価に用いる周期タスクには，整数演算の多い PID 制御タスク（PID-INT）および浮動小数点演算の多い PID 制御タスク（PID-FP），そして，メモリアクセスを中心とする配列処理タスク（MEM）を利用する．

PID-INT と PID-FP は，ロボットのモータ制御等で利用され，実際に厳密なリアルタイム性が必要なタスクである．MEM は，画像処理等，比較的に計算量の多いリアルタイムタスクを想定したタスクである．これらのタスクの擬似コードを**図 10** に示す．PID-INT と PID-FP は，それぞれ擬似コード内の変数が整数型と浮動小数点型になっている．

PID-INT と PID-FP の周期は {100 μs, 200 μs, 400 μs} から選択し，MEM の周期は {2 ms, 4 ms, 8 ms} から選択する．シングルスレッド実行におけるタスクの予測実行時間は，事前評価により計測するものとし，正確な WCET 解析は本論文の範囲外とする．PID-INT および PID-FP のシングルスレッド実行における実行時間を計測したところ，7.87 μs および 8.46 μs であったので，両方とも 10 μs を設定した．MEM に関しては，図 10 にある ROW と COL の値を変えて 2 種類用意する（MEM1，MEM2）．MEM1 および MEM2 のシングルスレッド実行における実行時間は，それぞれ，89.77 μs と 180.12 μs であったので，MEM1 は 100 μs，MEM2 は 200 μs と設定した．

表 2 各種命令数
Table 2 The number of each instruction type.

	ALU	ALU D	FPU	FPU D	MAU	BU
PID-INT	39	1	0	0	12	1
PID-FP	15	0	24	1	15	1
MEM 1	27	0	0	0	31	5
MEM 2	49	0	0	0	62	10

表 3 各種命令レイテンシ
Table 3 The delay of each instruction type.

ALU	ALU D	FPU	FPU D	MAU	BU
1	9	3	12	30	1

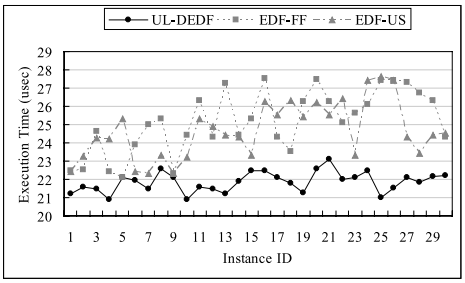


図 11 実行時間の変動
Fig. 11 Fluctuation of execution time.

そして，タスクの利用率の合計が，設定されたシステム負荷に近くなるように，周期と実行時間の組合せを決定した．

評価において想定した各種命令数を**表 2** に，各種命令のレイテンシを**表 3** に示す．各種命令数は，逆アセンブルを行い，事前に静的に数えて算出した．本論文では単純なタスクを想定しているため，分岐処理等を考慮したプログラムフロー解析による正確な実行命令数の算出は範囲外とする．ただし，MEM タスクに関しては，命令数誤差を軽減するために，Loop Unrolling を行ったうえで，各種命令数を求めた．ALU D および FPU D は，それぞれ除算器を表している．

4.2 実行時間に関する評価

図 11 は，システム負荷 400%の実験において無作為に選んだ 1 つのタスクに関して，各アルゴリズムでスケジュールした場合の実行時間の変移を示している．また，3.3 節で述べた U-Link スケジューリングモデルにおける理論的な予測実行時間も示している（UL-Estimation）．このタスクの相対命令利用率およびユニット競合率を**表 4** に示す．タスクの種類は PID-FP であり，表 4 から FPU と MAU の競合率が高いことが分かる．

UL-DEDF は，同時に実行するタスクの組合せを制

表 4 相対命令利用率（RIR）とユニット競合率（UCR）
Table 4 Relative Instruction Rate (RIR) and Unit Competition Rate (UCR).

	ALU	ALU D	FPU	FPU D	MAU	BU
RIR	0.29	0	0.36	0.02	0.23	0.02
UCR	2.41	1.00	3.33	1.51	3.19	1.47

限してスケジュールするため，EDF-FF や EDF-US に比べて実行時間の変動を抑制できたことが分かる．しかしながら，同時に実行するタスクの組合せを制限しても，入力値の大きさやキャッシュの状態，実行パスの変化等，様々な原因によって実行時間は変動してしまうため，実行時間を十分に安定させることはできなかった．本評価で用いたタスクの性質から，入力値や実行パスが大きく変化することは考えられないので，図 11 で発生した実行時間の変動の主な原因はキャッシュのヒット率であったと考えられる．

表 2 にあるように本評価では，相対命令利用率およびユニット競合率を求める際に，すべての Load/Store 命令がメモリを参照すると仮定した．しかしながら，Load/Store 命令がキャッシュを利用する場合には，そのレイテンシは著しく削減されるので，相対命令利用率およびユニット競合率を求める際に，機能ユニットにキャッシュの項目を追加した方が効果的である．そして，キャッシュのヒット率から，Load/Store 命令をメモリアクセスとキャッシュアクセスに区別することで，より正確な相対命令利用率およびユニット競合率を求めることができると考えられる．ここで，ユニプロセッサではデータのキャッシュヒット率を予測する手法[11),13)] が提案されているが，SMT プロセッサでは，さらにスレッド間でキャッシュを共有するため，キャッシュの動作の予測が困難であることが問題になる．この点で，U-Link スケジューリングは，同時に実行するタスクの組合せを制限することで，参照の局所性によりキャッシュラインの入れ替えを軽減することができるので，従来のスケジューリング手法よりもキャッシュのヒット率の変動を抑えることができると考えられる．しかしながら，キャッシュのヒット率を調節することに関しては，ソフトウェアによるアプローチには限界がある．よって，キャッシュのヒット率の変動を抑制するためには，実行時に動的にパイプラインの処理を単純化する手法[1)] や WCET 解析とキャッシュの関係を考慮したアウトオブオーダ実行の設計[9)] 等，ハードウェアによるアプローチが効果的であると考えられる．

UL-DEDF では，WCET は 23.11 μs，BCET は 20.9 μs であり，実行時間の最大変動量（WCET − BCET）は 2.22 μs と小さく，その変動の範囲は EDF-FF および EDF-US よりも制限されているのが分かる．また，このタスクの予測実行時間は 24.41 μs であり，一度も実際の実行時間が予測実行時間を超えることはなかった．紙面の都合上，ここでは示さないが，これはすべてのタスクに対しても同様であった．一方，EDF-FF では，WCET が 20.13 μs，BCET が 27.53 μs であり，実行時間の最大変動量が 7.40 μs と大きく，その変動の範囲にも制限がない．EDF-US においても，WCET が 27.64 μs，BCET が 22.34 μs であり，実行時間の最大変動量が 5.30 μs となり，UL-DEDF よりも大きい．リアルタイムスケジューリングでは，タスクの予測実行時間に WCET を用いることが多いので，WCET と BCET の差が大きいとアイドル時間が増加し，システムの利用効率は大きく低下してしまう．これでは，スループット向上という SMT プロセッサの利点を活かすことができない．よって，実行効率を考慮することでタスクの実行時間の変動を抑制できる UL-DEDF の効果は大きいと考えられる．

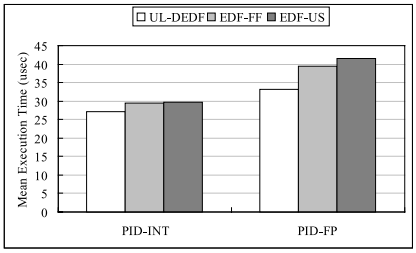


図 12 PID タスクの平均実行時間
Fig. 12 Mean execution time of PID tasks.

次に，タスクの各種類（PD-INT，PD-FP，MEM1，MEM2）ごとの平均実行時間の評価をとり，考察する．

図 12 は，各アルゴリズムにおける PID 制御タスクの平均実行時間および 3.3 節で述べた予測実行時間を示している．デッドラインミスを起こしたタスクは計測対象としなかった．PID-INT の平均実行時間は，UL-DEDF で 20.24 μs，EDF-FF で 24.32 μs，EDF-US で 26.43 μs であった．この結果を見ると，UL-DEDF がわずかに実行時間が短いが，各アルゴリズムで大きな差は見られなかった．これは，PID-INT で多く利用する ALU が，他の PID-FP や MEM1 および MEM2 においても頻繁に利用するためであると考えられる．PID-FP に関しては，EDF-FF および EDF-US での平均実行時間が，それぞれ 29.32 μs および 31.31 μs であるのに対して，UL-DEDF では 23.22 μs になっていることから，UL-DEDF の効果が，PID-INT よりも大きくなっていることが分かる．これは，UL-DEDF

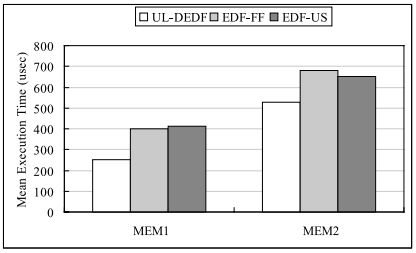


図 13 MEM タスクの平均実行時間
Fig. 13 Mean execution time of MEM tasks.

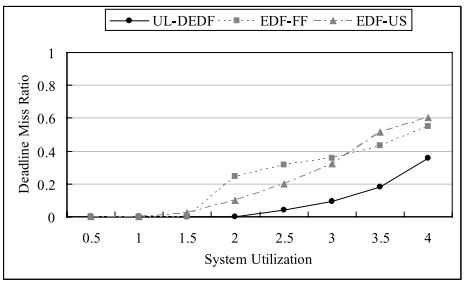


図 14 システム負荷によるデッドラインミス率
Fig. 14 Effect of system load on deadline miss ratio.

の方が FPU を効率的に利用できたことを表している．

図 13 は，各アルゴリズムにおける MEM1 と MEM2 の平均実行時間および予測実行時間を示している．UL-DEDF は，EDF-FF と EDF-US に比べて MEM1 および MEM2 の実行時間を約 100〜130 μs 短縮している．MEM タスクは，扱う時間の単位が大きいので，UL-DEDF が短縮できる実行時間も非常に大きい．メモリアクセスのレイテンシは他の演算に比べて大きいため，効率的にスケジュールできれば，短縮できる実行時間も大きくなる．よって，UL-DEDF は，メモリアクセスを多く行うタスクに対して，より効果的であるといえる．この結果から，UL-FFDE によって構築された同時実行セットを用いることで，UL-DEDF は，実行効率の高いタスクの組合せでスケジュールすることが可能になり，タスクの平均実行時間を短縮できたことが分かる．

図 11，図 12，図 13 を見ると，今回の評価に関しては，U-Link スケジューリングモデルにおける予測実行時間は，実際の実行時間よりも過大に評価を行っていた．これは，モデル化の際のユニット競合率の見積りが大きかったことに起因する．3.3 節で提案したユニット競合率の算出方法では，同じユニットを利用する命令がつねに競合することを仮定している．たとえば，タスク 1 とタスク 2 が両方とも 4 つの整数演算命令を持っていたとする．これらの整数演算命令が，タスク 1 では初めの方に，タスク 2 では最後の方に実行された場合，競合する確率は小さい．しかしながら，提案した算出方法では，これら 4 つの整数演算命令はすべて競合すると仮定している．そのため，本論文で提案したユニット競合率は実際の競合率よりも大きい値を見積もっているといえる．一方，ユニット競合率を過小評価すると予測実行時間も過小評価されるため，スケジューラビリティ判定が適切に動作せず，デッドラインミスを引き起こす可能性がある．この点を考慮すると，競合率を大きく見積もったことでデッドラインミスの発生を防止できたとも考えられる．

4.3 リアルタイム性に関する評価

図 14 は，システムに投入するタスクの利用率の合計（システム負荷）を変化させていった場合の，各アルゴリズムのデッドラインミス率を示している．

$$\text{デッドラインミス率} = \frac{\text{デッドラインミスの総数}}{\text{全タスクのリリースの総数}}$$

UL-DEDF は，システム負荷が 200%まではデッドラインミスを起こさずにすべてのタスクをスケジュールできた．一方，システム負荷が 200%の時点で，EDF-FF および EDF-US でのデッドラインミス率は，それぞれ 24.36%および 9.83%であった．EDF-FF は，各 LP を容量が 1 の箱，各タスクのシングルスレッド実行における利用率をアイテムの容量として，First-Fit を実行する．よって，投入されたタスク数が少ない場合には，各 LP の利用率に偏りが出てしまう．たとえば，EDF-FF では，First-Fit の性質上，LP_1，LP_2 の利用率が 1 に近く，LP_3 の利用率が小さいという状況が発生する．このとき，SMT 実行によってタスクの実行時間は延長するので，LP_1 の実行効率がある一定以上低下すると，LP_1 に割り当てられたタスクのデッドライン率は急激に増える．このような理由から，システム負荷 200%において，EDF-FF のデッドラインミス率が EDF-US に比べて大きくなっていると考えられる．

システム負荷が増加するにつれて各アルゴリズムにおけるデッドラインミス率も増加しているが，どのシステム負荷においても，UL-DEDF でのデッドラインミス率は EDF-FF および EDF-US よりも低い．最終的にシステム負荷が 400%になると，UL-DEDF の 35.95% に対して，EDF-FF では 55.13%，EDF-US では 59.94% であった．この結果から，同じタスクセットを利用した場合，EDF-FF および EDF-US に比べると，UL-DEDF は，デッドラインミスを起こすことなく多くのタスクを処理でき，スループットが高いといえる．

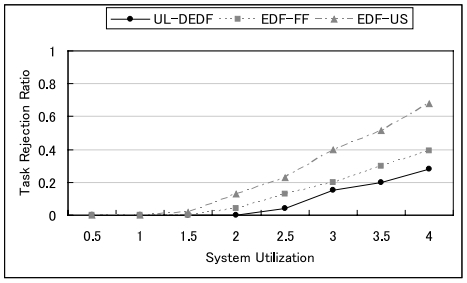


図 15 タスク拒否率（スケジュール可能性判定有）
Fig. 15 Task rejection ratio with a feasibility test.

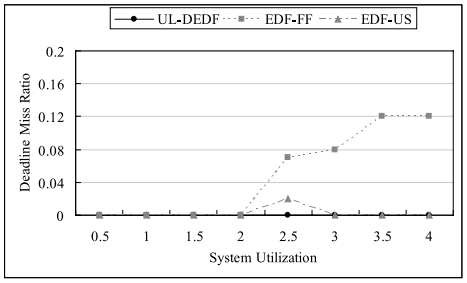


図 16 デッドラインミス率（スケジュール可能性判定有）
Fig. 16 Deadline miss ratio with a feasibility test.

先の評価により，UL-DEDF と UL-FFDE を組み合わせることで，高いスループットを達成できることを示した．次に，スケジュール可能性を判定する機構を実装し，UL-DEDF が，デッドラインミスを起こすことなく，従来手法より多くのタスクを処理できたことを示す．具体的には，タスクセットを構築する際に，タスクを 1 つずつ追加していき，各タスクに対してスケジュール可能性判定を行う．ここで，EDF-FF および EDF-US のスケジュール可能性判定には，SMT 実行における予測実行時間が必要になる．しかしながら，これらのアルゴリズムでは，SMT 実行における実行時間を予測する手法は提案されていない．そこで，単純に，UL-DEDF で利用する予測実行時間を用いることにする．スケジュール可能性判定は，各アルゴリズムにおいて，以下の手順で行われる．

UL-DEDF：UL-FFDE で同時実行セットを構築するとき，式 (14) と式 (15) を満たしていればスケジュール可能．式 (16) に関しては，仮想タスク化を行うことで，考慮する必要はなくなる．

EDF-FF：First-Fit によってタスクを LP_j に割り当てたとき，式 (19) を満たしていればスケジュール可能．

$$\sum_{\tau_i \in LP_j} \ddot{U}_i \leq 1 \tag{19}$$

EDF-US：タスクセットが，式 (20) を満たしていればスケジュール可能．

$$\ddot{U}_i \leq \frac{M^2}{2M-1} \tag{20}$$

スケジュール可能性判定によって許可されなかったタスクはスケジューリングの対象となるタスクセットに含まれることはない．また，デッドラインミスを起こしたタスクは，その後のスケジュールを崩さないために，実行を終了するものとする．

図 15 は，設定されたシステム負荷における，各アルゴリズムのタスク拒否率を示している．タスク拒否率は以下の式で求めるものとする．

$$\text{タスク拒否率} = \frac{\text{受入れを拒否されたタスクの総数}}{\text{システムに投入されたタスクの総数}}$$

結果を見ると，どのシステム負荷においても，UL-DEDF のタスク拒否率が最も低い．最大で，タスク拒否率を EDF-FF より 11%，EDF-US より 40%ほど削減している．これは，UL-DEDF が最も多くのタスクを処理できることを意味している．EDF-US のスケジュール可能性判定式である式 (20) は，スケジュール可能なシステム負荷の上限が低いため，EDF-US におけるタスク拒否率は大きくなってしまったと考えられる．最終的にシステム負荷が 400%になると，EDF-US は約 70%のタスクを拒否してしまう．EDF-FF では，式 (19) にあるように，スケジュール可能性判定式は EDF-US ほど厳格ではない．しかしながら，どのシステム負荷においても，タスク拒否率は UL-DEDF より平均的に 10%ほど高い．これは，タスクを LP に割り当てる際のパーティショニング手法の効率の差であると考えられる．UL-DEDF とともに用いる UL-FFDE は，結果として各 LP の利用率が均一になるようにタスクを割り当てている．一方，EDF-FF で用いる First-Fit は，最初に見つかった十分な空き容量を持つ LP にタスクを割り当てるので，各 LP の利用率に偏りが出てしまう．結果として，UL-FFDE に比べると，各 LP の空き容量が多くなってしまう．タスク拒否率が低いということは，より多くのタスクを処理できたことであり，スループットは向上したといえる．

次に，タスクの受入れを拒否することによって，実際にデッドラインミスの発生を防止できたか否かについて評価した．**図 16** は，図 15 の実験と同じタスクセットを用いたときのデッドラインミス率を示している．結果を見ると，UL-DEDF ではデッドラインミスが発生していないことが分かる．よって，予測実行時間が十分であったことを意味する．EDF-FF では，システム負荷が 250%を超えたところから，約 10%のデッド

ラインミス率が検出されている．これは，スケジューリング方式自体のスループットに差があるため，UL-DEDF では十分であった予測実行時間が，EDF-FF では十分でなかったと考えられる．一方，EDF-US では，システム負荷が 250%のときに約 2%のデッドラインミス率が検出された．EDF-US では，スケジュール可能なシステム負荷の上限が非常に小さいため，システムに余分なアイドル時間が余っている．よって，実際のタスクの利用率の合計がスケジュール可能なシステム負荷の上限を超えても，余分なアイドル時間にタスクを処理できるので，実際の実行時間が予測実行時間を超えてしまった場合でも，いくつかのタスクは，デッドラインまでに実行を完了できたと考えられる．

5. 結　　論

本論文では，SMT プロセッサにおける実行効率を考慮し，リアルタイム性の保証と実行効率の向上を実現する U-Link スケジューリングを提案し，その有効性を評価した．

ヒューマノイドロボット等で実際にリアルタイム性を要求される PID 制御タスクおよび配列処理タスクを用いた評価では，実行時間の変動を抑制し，既存の手法に比べて，PID 制御タスクの平均実行時間を最大約 20%，配列処理タスクの平均実行時間を最大約 35%短縮することができた．また，スケジュール可能性判定を用いて，タスクの受入れを制御したところ，デッドラインミスを起こすことなく，最大約 40%多くのタスクを処理することができた．

今後の課題としては，本論文では，同時実行セット内のタスクの周期がハーモニックである仮定したが，汎用的なタスクセットにも対応できるようにスケジューリング理論を改善することがあげられる．また，評価において，実行効率を求めるために各タスクが持つ各種命令数を逆アセンブルすることによって静的に求めた．今後は，プログラムフロー解析やハードウェア内に機能ユニットごとのカウンタ等を用いて，より正確な実行効率の事前評価を行い，その有効性を検証していく予定である．実行効率の変動や向上に関しては，本論文ではキャッシュの動作は考慮しなかった．しかしながら，実際に実行効率を考えるうえでは，キャッシュの動作が非常に重要になってくる．今後は，キャッシュの動作を考慮した予測実行時間の算出方法や，キャッシュのヒット率の変動を抑制する方法について考えていく予定である．

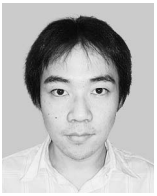
謝辞 本研究は，日本学術振興会の支援による．また，本研究の一部は，科学技術振興機構 CREST の支援による．

参 考 文 献

1) Anantaraman, A., Seth, K., Patil, K., Rotenberg, E. and Mueller, F.: Virtual Simple Architecture (VISA): Exceeding the Complexity Limit in Safe Real-Time Systems, *Proc. International Symposium on Computer Architecture*, pp.350–361 (2003).
2) Andersson, B. and Jonsson, J.: Fixed-Priority Preemptive Multiprocessor Scheduling: To Partition or not to Partition, *Proc. International Conference on Real-Time Systems and Applications*, pp.337–346 (2000).
3) Arnold, R., Mueller, F., Whalley, D. and Harmon, M.: Bounding Worst-Case Instruction Cache Performance, *Proc. Real-Time Systems Symposium*, pp.172–181 (1994).
4) Eggers, S.J., Emer, J.S., Levy, H.M., Lo, J.L., Stamm, R.L. and Tullsen, D.M.: Simultaneous Multithreading: A Platform for Next-Generation Processors, *IEEE Micro*, Vol.17, pp.12–19 (1997).
5) Goossens, J., Funk, S. and Baruah, S.: Priority-Driven Scheduling of Periodic Task Systems on Multiprocessor, *Real-Time Systems*, Vol.25, pp.187–205 (2003).
6) Healy, C., Whalley, D. and Harmon, M.: Integrating the Timing Analysis of Pipelining and Instruction Caching, *Proc. Real-Time Systems Symposium*, pp.288–297 (1995).
7) Jain, R., Hughes, C.J. and Adve, S.V.: Soft Real-Time Scheduling on Simultaneous Multithreaded Processors, *Proc. Real-Time Systems Symposium*, pp.134–145 (2002).
8) Lauzac, S., Melhem, R. and Mosse, D.: Comparison of Global and Partitioning Schemes for Scheduling Rate Monotonic Tasks on a Multiprocessor, *Euromicro Workshop on Real Time Systems*, pp.188–195 (1998).
9) Li, X., Roychoudhury, A. and Mitra, T.: Modeling Out-of-Order Processors for Software Timing Analysis, *Proc. Real-Time Systems Symposium*, pp.92–103 (2004).
10) Liu, C.L. and Layland, J.W.: Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment, *J. ACM*, Vol.20, pp.46–61 (1973).
11) Liu, J. and Lee, H.: Deterministic Upperbounds of the Worst-Case Execution Times of Cached Program, *Proc. Real-Time Systems Symposium*, pp.182–191 (1994).
12) Lopez, J., Garcia, M., Diaz, J. and Garcia, D.: Utilization Bounds for EDF Scheduling on

Real-Time Multiprocessor Systems, *Real-Time Systems*, Vol.28, pp.39–68 (2004).
13) Min, S.K.S. and Ha, R.: Efficient Worst Case Timing Analysis of Data Caching, *Proc. Real-Time Technology and Applications Symposium*, pp.230–240 (1996).
14) Parekh, S., Eggers, S. and Levy, H.: Thread-Sensitive Scheduling for SMT Processors, Technical report, Dept. of Computer Science and Engineering, University of Washington (2000).
15) Ramamritham, K. and Stankovic, J.A.: Dynamic Task Scheduling in Distributed Hard Real-Time Systems, *IEEE Software*, Vol.1, pp.65–75 (1984).
16) Snavely, A. and Tullsen, D.M.: Symbiotic Jobscheduling for a Simultaneous Multithreading Architecture, *Proc. International Conference on Architectural Support for Programming Languages and Operating Systems*, pp.234–244 (2000).
17) Snavely, A., Tullsen, D.M. and Voelker, B.: Symbiotic Jobscheduling with Priorities for a Simultaneous Multithreading Processor, *Proc. ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, pp.66–76 (2002).
18) Srinivasan, A., Holman, P., Anderson, J.H. and Baruah, S.: The Case for Fair Multiprocessor Scheduling, *Proc. International Parallel and Distributed Proceesing Symposium*, pp.114–123 (2003).
19) Tullsen, D.M., Eggers, S.J. and Levy, H.M.: Simultaneous Multithreading: Maximizing On-Chip Parallelism, *Proc. Annual International Symposium on Computer Architecture*, pp.392–403 (1995).
20) Tullsen, D.M., Eggers, S.J., Levy, H.M., Lo, J.L. and Stamm, R.L.: Exploiting Choice: Instruction Fetch and Issue on an Implementable Simultaneous Multithreading Processor, *Proc. Annual International Symposium on Computer Architecture*, pp.191–202 (1996).
21) Yamasaki, N.: Responsive Multithreaded Processor for Distributed Real-Time Systems, *Journal of Robotics and Mechatronics*, Vol.17, pp.130–141 (2005).
22) 松井俊浩, 比留川博久, 石川　裕, 山崎信行, 加賀美聡, 堀　俊夫, 金広文男, 斉藤　元, 稲邑哲也：ヒューマノイド・ロボットのための実時間分散情報処理, 実時間処理に関するワークショップ, pp.1–7 (2004).
23) 内倉　要, 笹田耕一, 佐藤未来子, 加藤義人, 大和仁典, 中條拓伯, 並木美太郎：SMT プロセッサにおけるスレッドスケジューラの開発, 情報処理学会論文誌, Vol.46, pp.150–160 (2005).

(平成 18 年 1 月 27 日受付)
(平成 18 年 6 月 11 日採録)



加藤　真平（学生会員）

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．現在同博士課程に在籍．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．

小林　秀典

1977 年生．2000 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博士課程修了．博士（工学）．同年キャノン株式会社に入社．オペレーティングシステム等の開発に従事．

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 月より同助教授．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．