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

2007－ＳＬＤＭ－129／2007－ＥＭＢ－4（15）
2007／3／16

優先度付きSMTにおけるデッドラインからの遅延時間を保証可能なソフトリアルタイムスケジューリング

武田 瑛† 船岡 健司†† 加藤 真平†† 山崎 信行†,††

†慶應義塾大学理工学部情報工学科
††慶應義塾大学大学院理工学研究科開放環境科学専攻
〒223–8522 横浜市港北区日吉 3–14–1
E-mail: †{takeda,funaoka,shinpei,yamasaki}@ny.ics.keio.ac.jp

あらまし 優先度付きSMTとは，ハードウェアで各スレッドに優先度を設け，優先度の高いスレッドに優先的にプロセッサ資源を割り当てることができる技術である．優先度付きSMTは，高スループットを保ちながらリアルタイム処理を実現できるという点で，ソフトリアルタイムシステムの要件を満たしている．本論文では，優先度付きSMT上のソフトリアルタイムシステムを対象としたリアルタイムスケジューリングアルゴリズムを提案する．そして，システムの品質(QoS)の要求を満たすために，タスクのデッドラインからの遅延時間(Tardiness)の上限値を解析する．シミュレーションによる評価を行い，Tardinessの上限値の厳密さを議論する．

キーワード 優先度付きSMT，ソフトリアルタイムシステム，スケジューリングアルゴリズム，遅延時間

Soft Real-Time Scheduling that Bounds Deadline Tardiness on Prioritized Simultaneous Multithreading

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

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

Abstract The prioritized SMT is a technique that places the priority to a thread by hardware and preferentially allocates the processor resources to the higher priority thread. This technique meets the requirements of soft real-time systems due to be able to realize real-time processing with high throughput. This paper proposes a real-time scheduling algorithm for soft rea–time systems on the prioritized SMT and analyzes the upper bound of the tardiness of tasks in order to meet QoS requirement of the systems. We discuss the tightness of the tardiness bound in the evaluation with the tick-level simulation.

Key words Prioritized SMT, Soft Real-Time System, Scheduling Algorithm, Tardiness

1. 序 論

今日の組込みシステムでは，マルチメディア処理などのソフトリアルタイムシステムの重要性が増してきており，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．しかしながら，ほとんどの組込みシステムにはハードウェア資源と消費電力の制限があるため，プロセッサの動作周波数を上げることは難しい場合が多い．このような背景から，近年ではSimultaneous Multithreading (SMT) [7] やChip Multiprocessing (CMP) [6] のようにスレッドレベル並列性を抽出してシステムのスループットを向上する技術が組込みシステムでも利用されるようになってきている．とりわけSMTは，複数のスレッド間でハードウェア資源を共有することにより資源の利用率が向上できるため，低い動作周波数で高いスループットを実現できる．

SMTは，スレッド間でハードウェア資源を共有しているため，同時に実行されるタスクによって実行時間が大きく変動する．時間予測性が重要なリアルタイムシステムにおいて，このような実行時間の変動は大きな問題となる．我々は，SMTの各スレッド実行に優先度を導入した優先度付きSMT(RMT)を提案した[9]．優先度付きSMTを用いることで，システム内のタスク数がハードウェア内のスレッド数よりも少ない場合には，オペレーティングシステムのサポートなしに固定優先度に

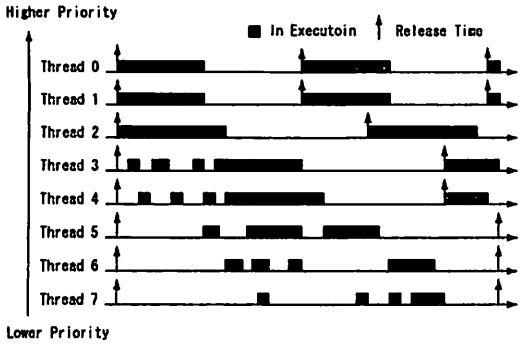

図 1 優先度付き SMT 実行

よるリアルタイム実行が可能である．しかしながら，タスク数が多くなった場合には，ソフトウェアによるリアルタイムスケジューリングを行う必要がある．優先度付き SMT 機構を用いたハードリアルタイムスケジューリングアルゴリズムとしては RM-SPU が提案されたが [5]，RM-SPU はスケジュール可能性を保証するために各タスクの利用率に制限がある．優先度付き SMT の利点は高いスループットを保ちながらリアルタイム実行ができる点であるため，このような制限は望ましくない．

本論文では，優先度付き SMT を対象とし，より多くのタスクをスケジュール可能なソフトリアルタイムスケジューリングアルゴリズムを提案する．ソフトリアルタイムアプリケーションでは，多少のデッドラインミスは許容されるが，一定のサービス品質 (QoS) を保証することが重要である．QoS を保証するための要件は，デッドラインミス数やデッドラインミス率の制限などがあるが，本論文ではデッドラインからの遅延時間 (Tardiness) の上限を保証することを考える．Tardiness の上限値を解析する研究は，マルチプロセッサの分野で数多く行われている [1],[3],[8]．ビデオプレーヤーを例に挙げると，一定のフレームレートは保証されなければならないが，数ミリ秒のジッタはビデオの品質に重大な影響を及ぼさない．音声の品質はそのようなジッタにより敏感だが，音声データは多くの場合バッファリングされるので，多少の遅延時間はバッファされたデータを再生することによって補われる．

2. 優先度付き SMT

優先度付き SMT は，SMT の技術をリアルタイムシステムで利用するために提案され，我々が研究開発を行っている Responsive Multithreaded Processor (RMT Processor) に実装されている [9]．優先度付き SMT では，ハードウェアでスレッドに優先度を付加でき，複数のスレッドでハードウェア資源が競合した場合に，優先度の高いスレッドに優先的に資源を割り当てることができる．図 1 に見られるように，優先度付き SMT では優先度の高いスレッドが優先度の低いスレッドよりも優先的に実行される．

真垣らは，各スレッドの短期的な IPC を監視し，各スレッドが優先的に利用できる資源量を調節する IPC 制御機構を提案した [10]．この機構により，各スレッドの IPC の目標値をソフトウェアから設定可能である．この機構では，各スレッドの IPC を厳密に保証することはできないが，本論文ではソフトリアルタイムシステムを対象としているため，平均的な保証で十分であると考えられる．そこで本論文では，各スレッドの IPC が保証できるものと仮定する．

3. スケジューリングアルゴリズム

本論文では，優先度付き SMT におけるソフトリアルタイムスケジューリングアルゴリズムとして，EDF-sh(EDF with task splitting and executing higher priority) を提案する．デッドラインからの遅延時間 (Tardiness) は，全てのスケジューリングアルゴリズムで上限値を解析できるとは限らない [8]．優先度付き SMT では，タスクが違う実行効率で実行されると実行時間が予測不可能となり，Tardiness を保証することが困難である．EDF-sh は，論理プロセッサ (スレッド) の優先度及び実行効率を固定し，タスクを一定の論理プロセッサに割り当てることにより，タスクの実行時間を一定にし，Tardiness の保証を実現させる．また，高い利用率のタスクを高い実行効率の論理プロセッサに割り当てることにより，タスク受理率を向上させる．

3.1 システムモデル

システム開始時に，タスクセット τ を与える．τ は，1 から N までインデックス付けされた N 個の周期タスク $(\tau_1, \tau_2, ..., \tau_N)$ の集合であり，システム実行時に新しいタスクが到着することはない．各タスク τ_i を (C_i, T_i) というタプルで表す．C_i は，τ_i の単一スレッド実行時の実行時間とし，T_i は τ_i の周期とする．各タスクは周期ごとに実行されるが，この時の各周期の実行の単位をジョブと呼ぶ．タスク τ_i の k 番目のジョブを $\tau_{i,k}$ と表す．時刻 d のデッドラインをもつジョブ $\tau_{i,j}$ が時刻 t で実行を完了した場合，$\tau_{i,j}$ の Tardiness を $tardiness(\tau_{i,j}) = max(0, t-d)$ と定義する．τ_i の利用率を，$U_i = \frac{C_i}{T_i}$ と定義する．各タスクの利用率の合計を $U(\tau) = \sum_{\tau_k \in \tau} U_k$ で表し，これをタスクセットの利用率と呼ぶことにする．本論文では，タスクはすべて周期タスクとし，すべてのジョブの相対デッドラインが周期と等しいとする．また，ジョブのデッドラインミスが同じタスクの次のジョブのリリースを遅延させないものとする．

本論文では，ハードウェアのスレッドを論理プロセッサと呼ぶことにする．システムには，M 個の論理プロセッサ $(X_1, X_2, ..., X_M)$ が存在する．論理プロセッサ X_i の実行効率 e_i を単一スレッド実行時の IPC を 1 とした時の相対 IPC と定義する．ここで，$0 < e_i \leq 1$ と $e_i > e_{i+1}$ が常に成り立つとする．各論理プロセッサの実行効率は IPC 制御機構によって一定に制御されるものとする．この制御により，優先度付き SMT におけるスケジューリングをプロセッサ間で異なる速度を持つマルチプロセッサにおけるスケジューリングに還元できる．X_j で実行されているタスク τ_i の実行時間は C_i/e_j である．そこでタスクセット τ の利用率は $U(\tau) \leq \sum_{i=1}^{M} e_i$ を満たすものとする．そうでなければ，Tardinss は無制限に増加してしまうためである．

3.2 タスク割当てフェーズ

まず，タスクを各論理プロセッサに割り当てるアルゴリズムを図 2 に示す．このアルゴリズムは，論理プロセッサ上のタス

```
1.  for p in 1..M do
2.    τ[p] := {}    U[p] := 0
3.  end for
4.  p := 1
5.  Order tasks by nonincreasing utilization
6.  for i in 1..N do
7.    if U[p] + C_i/T_i ≦ e[p] then
8.      τ[p] := τ[p] ∪ {τ}    U[p] := U[p] + C_i/T_i
9.    else
10.     if p = M then   declare FAILURE
11.     end if
12.     split task τ_i into τ_i' and τ_i'' such that
13.       T_i' = T_i
14.       T_i'' = T_i
15.       C_i' = (1 − U[p]) * T_i'
16.       C_i'' = (C_i/T_i − C_i'/T_i') * T_i''
17.     if C_i'/(T_i'·e[p]) + C_i''/(T_i''·e[p+1]) > 1 then   declare FAILURE
18.     end if
19.     τ[p] := τ[p] ∪ {τ_i'}    U[p] := U[p] + C_i'/T_i'
20.     τ[p+1] := τ[p+1] ∪ {τ_i''}    U[p+1] := U[p+1] + C_i''/T_i''
21.     p := p+1
22.   end if
23. end for
```

図2　タスク割当て

クの合計利用率がそれぞれの実行効率を超えないようにタスクを割り当てる．これは，マルチプロセッサスケジューリングにおけるパーティショニングに類似しているが，文献[1]のアプローチのように，タスクを分割し2つの論理プロセッサでスケジュールすることができる．このことにより，論理プロセッサの容量を Tardiness が解析できる限界まで利用することができる．また，高い利用率のタスクを高い実行効率の論理プロセッサに割り当てることで，タスク受理率を向上させる．

まず，全タスクを利用率の高い順にソートする(5行目)．そして，タスク τ_i を現在の論理プロセッサ p に割り当てようと試みる．現在の論理プロセッサ上のタスクの合計利用率 U[p] と τ_i の利用率の合計が p の実行効率 e[p] を超えないならば(7行目)，その τ_i を p に割り当てる(8行目)．もしそうでないなら，タスクを τ_i',τ_i'' の2つに分割し(12〜16行目)，論理プロセッサ p と p+1 に割り当てる(19〜20行目)．ここで，τ_i' 及び τ_i'' は仮想的なタスクであり，τ_i の実行のための区間を確保することが目的であることに注意されたい．すなわち，τ_i' と τ_i'' がスケジュールされた区間で τ_i が実行される．最後に，次のタスクのために論理プロセッサのインデックスを移動する(21行目)．このアルゴリズムは，タスクを割り当てられる論理プロセッサがなくなった場合(10行目)，または分割した2つのタスクが並行に実行されることが避けられない場合(17行目)，失敗する．

このアルゴリズムを用いて，$\tau_1=(8,20)$，$\tau_2=(2,5)$，$\tau_3=(3/10)$，$\tau_4=(1,4)$，$\tau_5=(4,20)$，$\tau_6=(2,20)$，$\tau_7=(1,20)$ の7個のタスクを3つの論理プロセッサ X_1，X_2，X_3(実行効率はそれぞれ 1，1/2，1/4) に割り当てた例を図3に示す．この例では，τ_3 が X_1 と X_2 に，τ_5 が X_2 と X_3 に分割されている．

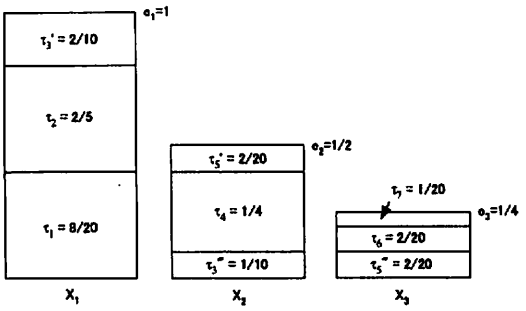

図3　タスク割当ての例

```
1.  for p in 1..M do
2.    when any split task arrives on p do
3.      t0 := current time
4.      t1 := next time a split task arrives on p
5.      calc_interval // here we calculate timea[p] and timeb[p]
6.      if firsttask[p] then
7.        execute firsttask[p] on processor p during [t0, timea[p])
8.      end if
9.      schedule unsplit tasks with EDF during [timea[p],timeb[p])
10.     if lasttask[p] then
11.       execute lasttask[p] on processor p during [timeb[p], t1)
12.     end if
13.   end do
14. end for
```

図4　タスクの実行

3.3　タスク実行フェーズ

次は実行アルゴリズムを説明する．このアルゴリズムは，各論理プロセッサ間で独立に，割り当てられたタスクを EDF でスケジュールする．例外として，分割されたタスクに関しては，並列に実行されないようにスケジュールをおこなう必要がある．分割タスクは，文献[2]と同様に，同じ論理プロセッサに割り当てられた非分割タスクよりも優先的に実行される．

実行アルゴリズムを図4に示す．各論理プロセッサごとに，任意の分割タスクが到着した時に(2行目)，t_0 を現在の時刻，t_1 を次に分割タスクが到着する時刻とし(3〜4行目)，関数 calc_interval で timea[p] と timeb[p] を計算する(5行目)．そして，もし firsttask[p] が存在すれば区間 [t_0,timea[p]) で firsttask[p] を実行し(6〜8行目)，区間 [timea[p],timeb[p]) で非分割タスクを EDF にしたがって実行する(9行目)．lasttask[p] が存在すれば，区間 [timeb[p],t_1) で lasttask[p] を実行する．

分割タスクは区間 [t_0,timea[p]) 及び [timeb[p],t_1) で実行される．timea[p] と timeb[p] を算出するのが関数 calc_interval である．関数 calc_interval を図5に示す．τ_i' が t_0 から $\frac{C_i'}{T_i'}\cdot\frac{t_1-t_0}{e_p}$，$\tau_i''$ が t_1 まで $\frac{C_i''}{T_i''}\cdot\frac{t_1-t_0}{e_p}$ 実行されるように timea[p] と timeb[p] を計算している．

この実行アルゴリズムで図3での分割タスクを実行した例を図6に示す．例えば，区間 [0,10) では τ_3' は X_1 で時刻 10 まで $\frac{2}{10}\cdot\frac{10-0}{1}=2$ 実行され，τ_3'' は X_2 で時刻 0 から $\frac{1}{10}\cdot\frac{10-0}{1/2}=2$ 実

```
1.  procedure calc_interval is
2.  begin
3.    if there is a task τ_i' assigned to p then
4.      lasttask[p] := τ_i'
5.      timeb[p] := t1 − (C_i'/T_i') * (t1 − t0) / e[p]
6.    else
7.      lasttask[p] := NULL
8.      timeb[p] := t1
9.    end if
10.   if there is a task τ_i'' assigned to p then
11.     firsttask[p] = τ_i''
12.     timea[p] := t0 + (C_i''/T_i'') * (t1 − t0) / e[p]
13.   else
14.     firsttask[p] := NULL
15.     timea[p] := t0
16.   end if
17. end
```

図5 関数 calc_interval

行される.

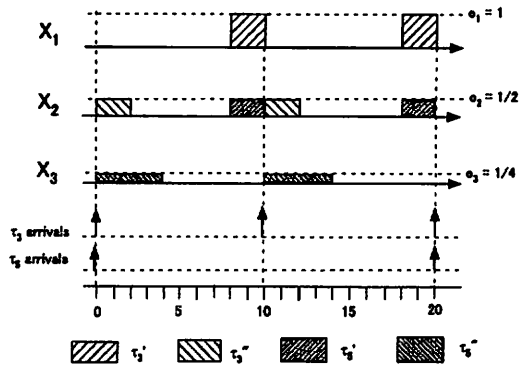

図6 分割タスクの実行例

4. Tardiness の上限値

前節で述べたように，EDF-sh では，分割タスクはデッドラインミスを起こさない．また，分割タスクが存在しない論理プロセッサでは，全タスクが EDF によってスケジュールされるためデッドラインミスが発生しない．よって，論理プロセッサに分割タスクが存在する場合の非分割タスクの Tardiness の上限値を求めればよい．

以下の定理が示される．

定理 1. X_k に割り当てられた分割タスクを τ_i''，τ_j' とすると，X_k に割り当てられたタスクの Tardiness は，高々

$$\Delta = \frac{C_i'' + C_j'}{e_k - U_i'' - U_j'} \tag{1}$$

である．

証明. 背理法で証明する．つまり，X_k に割り当てられた非分割タスク τ_q のジョブ $\tau_{q,l}$ の Tardiness が Δ を超えると仮定する．t_d, t_c, t_0 を以下のように定義する．

$t_d \equiv$ ジョブ$\tau_{q,l}$の絶対デッドライン (2)

$t_c \equiv t_d + \Delta$ (3)

$t_0 \equiv t_c$以前で X_kが「アイドル状態」または
「t_dより後のデッドラインを持つ非分割タスクが
実行されている状態」である最後の瞬間 (4)

また，τ_{split} と $\tau_{unsplit}$ をそれぞれ X_k に割り当てられた分割タスク，非分割タスクの集合とする．さらに，$demand(\tau, t_0, t_c)$ を区間 $[t_0, t_c)$ において τ のジョブが実行できる最大時間とする．

まず，$demand(\tau_{split}, t_0, t_c)$ と $demand(\tau_{unsplit}, t_0, t_c)$ を求める．分割タスクは非分割タスクより高優先度であるので，式 (4) より，t_0 までに実行を終了している．よって，区間 $[t_0, t_c)$ で実行される分割タスク τ_i''，τ_j' は区間 $[t_0, t_c)$ でリリースされる．$[t_0, t_c)$ でリリースされる τ_i'' のジョブの数は高々$\lceil \frac{t_c - t_0}{T_i''} \rceil < (\frac{t_c - t_0}{T_i''} + 1)$，$[t_0, t_c)$ でリリースされる τ_j' のジョブの数は高々$\lceil \frac{t_c - t_0}{T_j'} \rceil < (\frac{t_c - t_0}{T_j'} + 1)$ である．したがって，X_k での τ_i'' の実行時間が $\frac{C_i''}{e_k}$，τ_j' の実行時間が $\frac{C_j'}{e_k}$ であるから，

$$\begin{aligned} demand(\tau_{split}, t_0, t_c) &< \frac{C_i''}{e_k}\left(\frac{t_c - t_0}{T_i''} + 1\right) + \frac{C_j'}{e_k}\left(\frac{t_c - t_0}{T_j'} + 1\right) \\ &= \frac{1}{e_k}\left((U_i'' + U_j')(t_c - t_0) + C_i'' + C_j'\right) \end{aligned} \tag{5}$$

一方，仮定より，式 (2)〜(4) から，区間 $[t_0, t_c)$ に X_k で実行している非分割タスクのすべてのジョブは t_0 以降にリリースされ，かつ t_d 以前のデッドラインをもっている．そのような非分割タスク τ_m の数は高々$\lfloor \frac{t_d - t_0}{T_m} \rfloor \leq \frac{t_d - t_0}{T_m}$ であるから，

$$\begin{aligned} demand(\tau_{unsplit}, t_0, t_c) &\leq \sum_{\tau_m \in \tau_{unsplit}} \left\lfloor \frac{t_d - t_0}{T_m} \right\rfloor \cdot \frac{C_m}{e_k} \\ &\leq \frac{1}{e_k}(t_d - t_0) \sum_{\tau_m \in \tau_{unsplit}} \frac{C_m}{T_m} \\ &\leq \frac{1}{e_k}(t_d - t_0)\left(e_k - U_i'' - U_j'\right) \end{aligned} \tag{6}$$

よって，

$$\begin{aligned} &demand(\tau_{split} \cup \tau_{unsplit}, t_0, t_c) \\ &< \frac{1}{e_k}\left((U_i'' + U_j')(t_c - t_d) + C_i'' + C_j'\right) + t_d - t_0 \end{aligned} \tag{7}$$

$T_{q,l}$ が t_c までに実行を終了していないことから，

$$t_c - t_0 < demand(\tau_{split} \cup \tau_{unsplit}, t_0, t_c) \tag{8}$$

である．よって，

$$t_c - t_0 < \frac{1}{e_k}\left((U_i'' + U_j')(t_c - t_d) + C_i'' + C_j'\right) + t_d - t_0 \tag{9}$$

したがって，

$$t_c - t_d < \frac{C_i'' + C_j'}{e_k - U_i'' - U_j'} = \Delta \tag{10}$$

一方，(2) と (3) の定義より，$t_c - t_d = \Delta$ である．これは上式と矛盾している。以上より，定理 1 が成り立つ． □

定理 1 によって各タスクの Tardiness の上限値が示された．以下に示される定理 2 によって，EDF-sh におけるタスクセットの Tardiness の上限値が保証できる．定理 2 は，定理 1 により明らかである．

定理 2. EDF-sh では，$U(\tau) \leq \sum_{k=1}^{M} e_k$ であるとすると，タスクセット τ の Tardiness は高々

$$\max_{1 \leq k \leq M} \frac{C_i'' + C_j'}{e_k - U_i'' - U_j'} \tag{11}$$

である．

5. Tardiness の上限値の縮小法

式 (1) で表される Tardiness の上限値を最小にするようにタスクを割り当てることは NP 困難である．ここでは，Tardiness の上限値を縮小するためのタスク割当てのヒューリスティックスを述べる．また，タスクの分割を制限することによって Tardiness の上限値を縮小する方法も述べる．

5.1 タスク割り当てのヒューリスティックス

3.2 で述べたタスクの割当てアルゴリズムは，next fit bin-packing algorithm [4] と同様だが，タスクを 2 つに分割することを許している．また，タスクを割り当てる前に利用率の高い順にソートすることにより，利用率の高いタスクを実行効率の高いプロセッサに割り当てる．このタスク割当て方法を **NFDU**(Next-Fit in Decreasing Utilization) と呼ぶことにする．

式 (1) から，Tardiness の上限値を縮小する 1 つの方法は，利用率の低いタスクを分割する方法である．これは，NFDU で割り当てていき，タスクを分割しなければならなくなったときに，割り当てていないタスクの中で最も利用率が低いタスクを分割させることにより可能である．すでにタスクは利用率の高い順に並んでいるので，今現在のタスクから走査し，現在のプロセッサの残りの容量よりも高い利用率の中から最も利用率の低いものを選べばよい．このタスク割当て方法を **NFLU**(Next-Fit with Lower Utilization) と呼ぶことにする．

Tardiness の上限値を縮小するもう 1 つの方法は，実行時間の少ないタスクを分割する方法である．NFDU で割り当てていき，タスクを分割しなければならなくなったときに，割り当てていないタスクの中で最も実行時間が小さいタスクを分割すれば，これを実現できる．現在のプロセッサの残りの容量よりも高い利用率の中から最も実行時間が小さいタスクを選べば良い．このタスクの割当て方法を **NFLE**(Next-Fit with Low Execution time) と呼ぶことにする．

利用率の低いタスクを分割するには，タスクを利用率の高い順に first fit で各論理プロセッサに割り当て，割り当てることができない場合分割する方法もある．この方法を **FFDU**(First Fit in Decreasing Utilization) と呼ぶことにする．FFDU は，分割しなければならないタスクの利用率が比較的高い場合に，2 つの論理プロセッサに分割できない可能性がある．このような場合に，3 つ以上のプロセッサに分割することを許せば，タスクを割り当てられる．次章の評価で見るように，FFDU は Tardiness の上限値を最も縮小させることができる．しかし，FFDU はアルゴリズムが複雑であり，またタスクのプロセッサ間の移動 (migration) が多くオーバーヘッドが高い．さらに，タスクの分割の最小単位には制限があり，実際には分割できない場合も存在する．次章で述べる評価では，タスクが理想的に分割できるものとする．

5.2 タスク分割の制限

式 (1) は，論理プロセッサに 2 つの分割タスクが存在する場合の Tardiness の上限値である．論理プロセッサに存在できる分割タスクを 1 つに制限すれば，式 (1) の Tardiness Bound を $\frac{C_i'}{e_k - U_i''}$ に縮小することができる．また，分割タスクが存在しない論理プロセッサ上のタスクはデッドラインミスしない．そのため，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にも，ハードリアルタイムタスクの存在する論理プロセッサでのタスクの分割を禁止することによって，本アルゴリズムを適用できる．

6. シミュレーションによる評価

本章では，式 (1) の Tardiness の上限値の理論的な厳密さを評価するために，ハードウェア資源の競合などは考慮に入れず，全てのタスクの実行をティック単位で設定可能な Tick-level シミュレーションを行った．

6.1 シミュレーションモデル

RMT Processor に実装されている優先度付き SMT は，8 つのスレッドから同時に命令をフェッチ可能であるが，命令発行は 4 つのスレッドからしか行わない．そこで本評価では，論理プロセッサ数を 4 個とする．実行効率をどのように予測し決定するかは今後の課題であるが，本評価では各論理プロセッサの実行効率をそれぞれ 0.8，0.7，0.3，0.2 と決定した．タスクの利用率は [0.01,0.2] の範囲の一様分布，タスクの実行時間は $[1, c_{max}]$ の範囲の一様分布で生成した．タスクの利用率と実行時間を決めれば，周期は自動的に求まる．c_{max} で 1000 個のタスクセットを生成した．それぞれのタスクセットで，全論理プロセッサの実行効率の和 (=2.0) を超えない限りタスクをタスクセットに追加した．それぞれのヒューリスティックスで，タスクを論理プロセッサに割り当て，式 (11) で Tardiness の上限値を算出した．また，各ヒューリスティックスについて実際にタスクをスケジュールし，Tardiness を観測した．本来であればハイパーピリオドと呼ばれる全タスクの周期の最小公倍数の時間が必要であるが，ハイパーピリオドは値が非常に大きくなりコンピュータでは計算できない場合があるため，実際のスケジュールは 100,000 までとした．

6.2 各ヒューリスティックスの比較

図 7 は各ヒューリスティックスの Tardiness の上限値の平均を示している．すべてのヒューリスティックスの Tardiness の上限値は最大実行時間 c_{max} に従って大きくなっていった．分割タスクの実行時間や利用率を考慮していない NFDU は，それらを考慮した NFLU や NFLE に比べて 2 倍程度の Tardiness の上限値となってしまった．NFLU と NFLE を比較すると，10%以内のごくわずかな差であるが，NFLE の Tardiness の上限値の方が低

い．しかし，利用率の範囲を変化させるとNFLUのTardinessの上限値の方が低くなることもあり，NFLEとNFLUのどちらが良いかは一概には言えない．4つのヒューリスティックスの中でTardinessの上限値が最小となるのはFFDUであり，図7ではnext-fitの最良であるNFLEの20%ほどの低いTardinessの上限値が得られる．このことにより，分割せざるを得なくなるまで分割せず，利用率の低いタスクを分割するfirst-fitの有効性が示される．

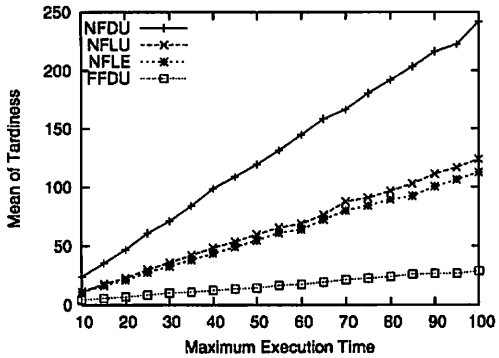

図7 各ヒューリスティックスの比較

6.3 Tardinessの上限値と観測値の比較

図8は，Tardinessの上限値と実際に観測されたTardinessの比が最も1に近い時の，Tardinessの上限値と実際のTardinessを示している．c_{max}が15，20，25，85の時には実際のTardinessが上限値に非常に近づいているが，このような場合でも実際のTardinessが上限値を超えないことがわかる．Tardinessの上限値と観測値が共に高い時には，上限値は厳密に観測値を見積もれているとは言えない．今回の評価では時間の都合上スケジュール時間を100,000としたが，コンピュータで処理しうる最大のスケジュール時間で厳密な評価を取る必要があり，このことは今後の課題とする．

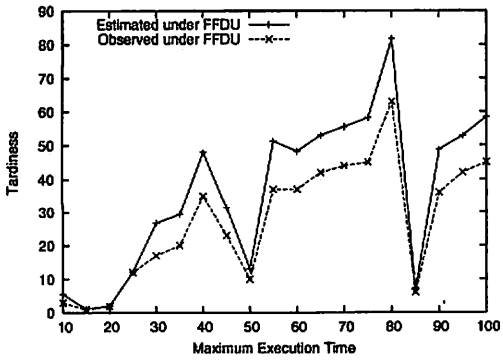

図8 FFDUのTardiness Boundと実際のTardiness

7. 結論及び今後の課題

本論文では，優先度付きSMT上におけるソフトリアルタイムスケジューリングアルゴリズムとしてEDF-shを提案した．EDF-shは，各論理プロセッサの実行効率を固定し，タスクを一定の論理プロセッサに割り当てることによって，Tardinessの上限値を解析可能とする．また、利用率の高いタスクを実行効率の高い論理プロセッサに割り当てることによって，利用率が高いタスクの受理率を高め，システムのスループットを向上させることができる．シミュレーションによる評価では，解析したTardinessの上限値が，上限値と観測値が共に低い場合にはほぼ厳密に見積もれていることを示した．提案したアルゴリズムにより，多少のデッドラインミスは許容されるが一定の品質を要求するソフトリアルタイムタスクを扱うことが可能となる．また，タスクの分割の制限により，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の両方を扱うことが可能である．

今回の評価では，スケジュール時間を制限したため，解析したTardinessの上限値が本当に厳密であるかどうかの評価は今後の課題とする．また，今回はTick-levelシミュレーションにより評価を行ったが，ハードウェアの資源の競合を考慮に入れた評価を行う必要がある．さらに，各論理プロセッサの実行効率を適切に決定する方法も考える必要がある．

謝辞 本研究は，科学技術振興機構CRESTの支援による．

文 献

[1] J. Anderson, V. Bud, and U. Devi. An EDF-based Scheduling Algorithm for Multiprocessor Soft Real-Time Systems. In *Proc. of the 17th Euromicro Conference on Real-Time Systems*, pages 199–208, July 2005.

[2] 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*, pages 322–334, August 2006.

[3] U. Devi and J. Anderson. Tardiness Bounds under Global EDF Scheduling on a Multiprocessor. In *Proc. of the 26th IEEE Real-Time Systems Symposium*, pages 330–341, December 2005.

[4] S. K. Dhall and C. L. Liu. On a real-time scheduling proglem. *Operations Research*, 26:127–140, 1978.

[5] S. Kato and N. Yamasaki. Study of Real-Time Scheduling under Prioritized Simultaneos Multithreading. In *Proc. of Work in Progress Session of the 27th IEEE International Real-Time System Symposium*, pages 97–100, December 2006.

[6] K. Olukotun, B. Nayfe, L. Hammond, K. Wilson, and K.Chang. The Case for a Single-Chip Multiprocessor. In *Proc. of Intl. Conf. on Architectural Support for Programming Languages and Operating Systems*, pages 2–11, 1996.

[7] D. M. Tullsen, S. J. Eggers, and H. M. Levy. Simutaneos Multithreading: Maximizing On-Chip Parallelism. In *Proc. of the 22nd Annual International Symplsium on Computer Architecture*, pages 392–403, June 1995.

[8] P. Valente and G.Lipari. An Upper Bound to the Lateness of Soft Real-Time Tasks Scheduled by EDF on Multiprocessors. In *Proc. of the 26th IEEE Real-Time System Symposium*, pages 311–320, 2005.

[9] N. Yamasaki. Responsive Multithreaded Processor for Distributed Real-Time Sytems. In *Journal of Robotics and Mechatronics*, volume 17, pages 130–141, 2005.

[10] 真垣郁男，伊藤務，山崎信行. Responsive Multithreaded Processorの命令供給機構およびIPC制御機構の設計と実装. 第7回組込みシステム技術に関するサマーワークショップ, pages 71–78, 2005.