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

2007－OS－105 (1)
2007／4／5

タスクの予測により コンテキストスイッチを投機実行する手法に関する検討

永田真稔† 小林良太郎† 島田俊夫†

本稿では，リアルタイムOSのコンテキストスイッチに着目して，タスク切り替え時間を削減する手法を提案する．提案手法では，まず，コンテキストをプロセッサ内に複数保持する．後続タスクの予測を行うハードウェアを導入し，通常の処理と並列に後続タスクを予測する．予測されたタスクのコンテキストスイッチを投機的に行い，タスク切り替え時間を隠蔽する．以上の機構を評価した結果，予測機構の履歴数を5として後続タスクを予測した場合，予測精度は65%～81%となった．また，このときタスク切り替えに要する時間の削減量が16%～20%となることがわかった．

Discussion about Technique for Speculatively Switching Context by Predicting Task

Masatoshi Nagata,† Ryotaro Kobayashi† and Toshio Shimada†

In this paper we propose a mechanism that reduces time for context switch in real-time operating systems. The mechanism holds multiple contexts in the processor. The next task after a current task is predicted. Necessary time for context switch can be hided since the context switch for the predicted task is performed in parallel with normal processing. The evaluation results show that the prediction accuracy range over 65%-81% and the ratio of the reducing time is 16%-20% when the size of the history table is five.

1. はじめに

リアルタイムOSは自動車の制御において重要な役割を果たしている．近年，自動車の搭載プロセッサ数が増加しており，安全制御など複雑で高速な動作をプロセッサ同士が連携して行う場合，処理速度が問題になりつつある．複数のプロセッサを1つに統合し，搭載するプロセッサの総数を減らすことで問題の困難さを軽減できる．複数のプロセッサが処理する仕事を1つのプロセッサで実行すると，プロセッサ当たりの処理タスク数が増加し，タスク制御が複雑になる．さらにタスク切り替えの頻度が増加し，タスク切り替え時間の総和が増加し，全タスクの実行時間も増加する．以上のことから，タスク切り替え時間を削減することが必要である．

そこで，本論文では，タスク切り替え時間を削減するために，タスク切り替え処理の一部であるコンテキストスイッチに着目し，コンテキストスイッチのオーバーヘッドを隠蔽することにより，コンテキストスイッチのオーバーヘッドを削減する手法を提案する．具体的には，まず，プロセッサ上で複数のコンテキストを保持するレジスタセットを備える．次に，タスク実行中に，タスクの遷移パターンを用いて後続タスクを予測し，予測されたタスクのコンテキストスイッチを投機的に行う．

2章ではコンテキストスイッチについて説明する．3章ではコンテキストスイッチを投機的に行う手法を提案する．4章で評価環境を述べ，5章で評価する．6章では関連研究を述べて，7章で本論文をまとめる．

2. コンテキストスイッチ

コンテキストスイッチに着目した理由は3つある．第1の理由として，タスク切り替えの際には必ず，コンテキストスイッチが発生するためコンテキストスイッチ時間を削減できれば，タスク切り替え時間の短縮につながる．第2の理由として，コンテキストスイッチは，どのタスク切り替えにおいても，同じ処理を行うので，新たに追加するハードウェア機構が複雑にならない．第3の理由として，コンテキストスイッチはメモリにアクセスを行い，キャッシュミスをする確率が高い[1)]．メモリアクセス時間が長くなるので投機実行によりメモリオーバヘッドを隠蔽できれば効果が大きい．

図1では，タスクBの実行中に，システムコールによってリアルタイムOSが呼び出され，タスクBより優先度の高いタスクAに実行が遷移している．

以降，本稿では，図1のタスクBのようにシステムコールによってOSが呼び出され，実行が中断されたタスクのことを先行タスクと呼び，図1のタスクAのようにOSによって次に実行することが決定されたタスクのことを後続タスクと呼ぶ．また，本稿では，

† 名古屋大学大学院工学研究科
Graduate School of Engineering, Nagoya University

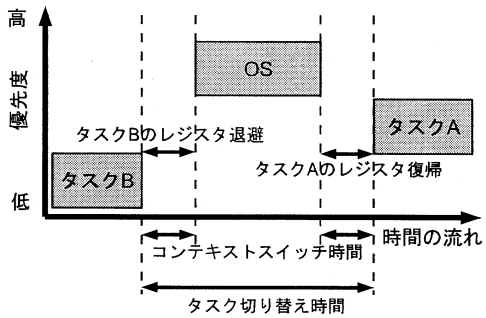


図 1　コンテキストスイッチ

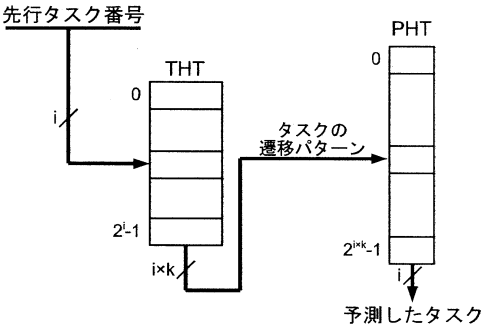


図 2　後続タスク予測機構

コンテキストスイッチとして，先行タスクのレジスタ退避と後続タスクのレジスタ復帰のみに注目し，その他のコンテキストスイッチの処理は従来通りとする．

先行タスクのレジスタ退避は，システムコールによって呼び出されたリアルタイム OS が先行タスク中断時に先行タスクのコンテキストをメモリに書き込むことで行われる．後続タスクのレジスタ復帰は，リアルタイム OS によって後続タスクが決定し，タスクを切り替える際に，メモリからコンテキストを読み出して，レジスタセットに書き込むことである．

3. 提 案 手 法

コンテキストスイッチを投機的に行うために，コンテキストを保持する 3 つのレジスタセットと，タスクの遷移パターンを用いる後続タスク予測機構を利用する．

提案手法では，先行タスク実行時に，後続タスク予測機構によって予測されたタスクのコンテキストを，3 つのレジスタセットの内の 1 つに読み出しておく．これにより，後続タスクのレジスタ復帰を投機的に行う．次に，システムコールによってリアルタイム OS が呼び出された時に，3 つのレジスタセットの内で先行タスクと後続タスクが使用していないレジスタセットを用いて先行タスクのレジスタ退避時間を隠蔽する．最後に，OS を実行する．OS 実行中に先行タスクのレジスタ退避を行い，リアルタイム OS によって後続タスクが決定し予測が成功したら，既にレジスタ復帰を行っていたレジスタセットを用いてレジスタ復帰時間を隠蔽する．

3.1 タスクの予測方法

図 2 に後続タスク予測機構の構成を示す．後続タスク予測機構では，タスクの遷移パターンを用いて後続タスクを予測するために，二つの表を用いる．一つは，各タスクに対して過去に遷移したタスクの履歴を記録する表で，タスク履歴表 (THT:Task History Table) と呼ぶ．もう一つは，タスクの遷移パターンに対して後続タスクを記録する表で，パターン履歴表 (PHT:Patern History Table) と呼ぶ．

後続タスクを予測する方法は，実行中の先行タスク番号を用いて THT を参照し，履歴パターンを得る．この履歴パターンを用いて PHT を参照し，予測後続タスクを得る．次に，後続タスクの情報を用いて THT と PHT を更新する．

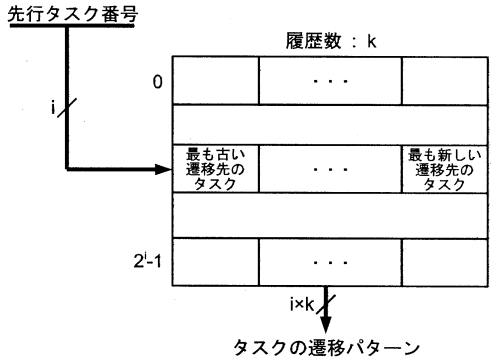


図 3　THT の構成

図 3 に THT の構成を示す．THT はタスクの遷移パターンを保持するテーブルである．THT の情報は先行タスク番号をインデックスとしてアクセスする．THT の更新は，後続タスクが決定するたびに，インデックス先のエントリを 1 タスク分左シフトし，右端に後続タスクを書き込む．タスク番号のビット長を $i$bit，THT が保持することが可能な履歴数を k とすると，タスクの遷移パターンのビット長は $i \times k$ ビットとなる．

図 4 に PHT の構成を示す．PHT は，タスクの遷移パターンをインデックスとして，遷移パターンが過去に遷移した後続タスクを保持し出力するテーブルである．PHT を更新するには，後続タスクが決定するたびに，対応するエントリに後続タスクを書き込む．PHT の総履歴数は，タスクの遷移パターンのビット長が $i \times k$ ビットなので，2^{ik} となる．

後続タスク予測機構の動作を具体例で説明する．図 1 のように，タスク B からタスク A にタスクが切り替わる場合を考える．図 5 に THT と PHT の値の変化を示す．まず，タスク B が実行中に，後続タスクを予測する場合について説明する．タスク B を用いて，図 5 の (a) の THT を参照すると，「ACAC」というタスクの遷移パターンを得る．このタスクの遷移パターンを用いて，図 5 の (a) の PHT を参照すると，「C 」という予測した後続タスクを得る．

THT の更新方法を説明する．図 5 の (a) の THT の対応するエントリ「B」にある各値「ACAC」を，左

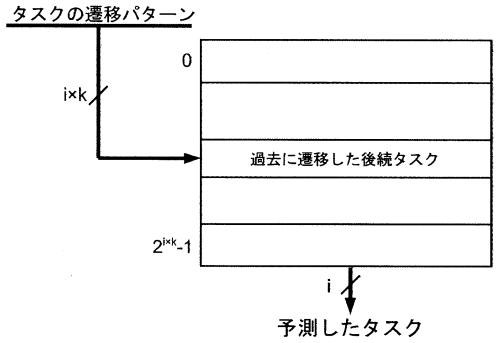
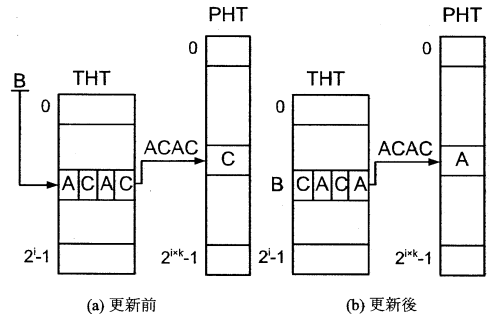


図 4　PHT の構成



(a) 更新前　　(b) 更新後

図 5　THT と PHT の値の更新方法

にタスク 1 個分シフトする．空いた右端に後続タスク「A」を書き込むことにより，図 5 の (b) の THT のように「CACA」となる．

PHT の更新方法を説明する．図 5 の (a) の PHT の対応するエントリ「ACAC」の値を「A」に更新し，図 5 の (b) の PHT を得る．

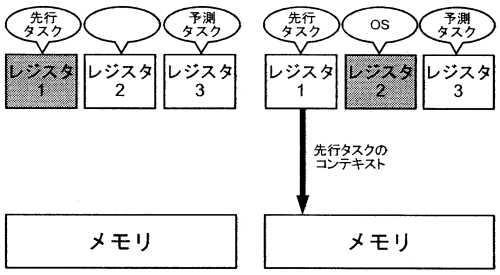
3.2　先行タスクのレジスタ退避時間の隠蔽方法

図 6 に先行タスクのレジスタ退避を行うときの，コンテキストスイッチの様子を示す．図 6 のレジスタ 1〜3 は，コンテキストを保持するレジスタセットを，その上の吹き出しは，各レジスタセットに保持されたコンテキストを示す．レジスタセットの内，色塗りされたレジスタセットは，そのときプロセッサで使用しているレジスタセットを示す．また，図 6 では，レジスタ 1 で先行タスクを実行し，レジスタ 3 に，先行タスク実行時に予測後続タスクのコンテキストをメモリから読み出し，保持した状態を表す．

図 6 を用いて，先行タスクのレジスタ退避時間の隠蔽方法を説明する．システムコールによって呼び出されたリアルタイム OS は，使用されていない図 6 のレジスタ 2 を用いて実行を開始する．OS 実行と並行して図 6 のレジスタ 1 に保持された先行タスクのコンテキストを，ハードウェア機構を用いて，メモリ上のスタックに待避する．

3.3　後続タスクのレジスタ復帰時間の隠蔽方法

提案手法では，リアルタイム OS によって，後続タスクが決定したら，予測後続タスクのコンテキストを OS 実行中にハードウェア機構でロードしたレジスタセットを用いて，後続タスク実行を開始し，レジスタの復帰時間を隠蔽する．予測が成功した場合と，失敗した場合について説明する．



(a) 先行タスク中断時　　(b) リアルタイムOS実行時

図 6　先行タスクのレジスタ退避時間の隠蔽方法

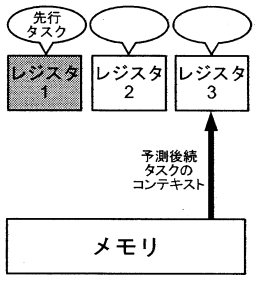
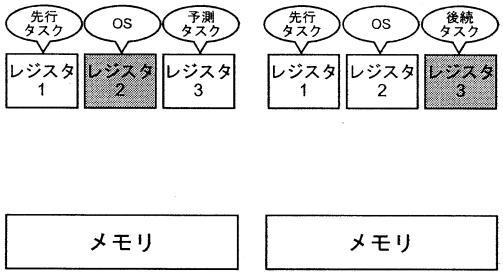


図 7　先行タスク実行時のコンテキストの移動

後続タスク予測機構が予測に成功した場合

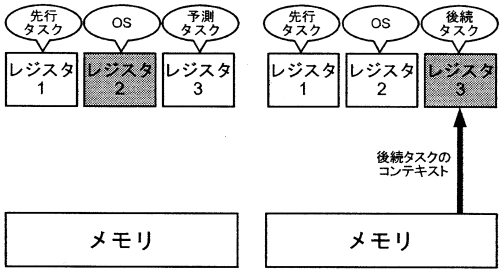
図 7 に先行タスク実行中の様子を示し，図 8 に後続タスク予測機構が予測に成功した場合の様子を示す．図 7 と図 8 のレジスタ 1〜3 は，コンテキストを保持するレジスタセットを，その上の吹き出しは，各レジスタセットに保持されたコンテキストを示す．レジスタセットの内，色塗りされたレジスタセットは，そのときプロセッサで使用しているレジスタセットを示す．また，図 7 では，レジスタ 1 で先行タスクを実行している状態を表し，図 8 では，レジスタ 1 で先行タスクを実行し，レジスタ 3 に，先行タスク実行時に予測後続タスクのコンテキストをメモリから読み出し，レジスタ 2 で OS を実行した状態を表す．

図 7 と図 8 を用いて，後続タスク予測機構が予測に成功した場合を説明する．予測タスクの実行に必要なレジスタデータは，先行タスク実行中にハードウェア機構により，先行タスクの実行と並行に，使用していない図 7 のレジスタ 3 にロードしておく．タスク切り替えを伴うシステムコールが呼ばれ，リアルタイム OS によって後続タスクが決定する．後続タスクが決定したら，後続タスクと，先行タスク実行中に，予測器によって予測されたタスクとを比較し，予測成功と分かった場合は後続タスクのコンテキストが図 8 のレジスタ 3 に存在するため，プロセッサは図 8 のレジス



(a) OS実行時　　(b) OSの処理中断時

図 8　後続タスクのレジスタ復帰時間の隠蔽方法 (予測成功時)



(a) OS実行時　　(b) OSの処理中断時

図 9　後続タスクのレジスタ復帰時間の隠蔽方法 (予測失敗時)

タ 3 を用いて後続タスクを実行する．多くの場合後続タスクのレジスタ復帰時間を隠蔽することができる．

後続タスク予測機構が予測に失敗した場合

図 9 に後続タスク予測機構が予測に失敗した場合の様子を示す．図 9 のレジスタ 1〜3 は，コンテキストを保持するレジスタセットを，その上の吹き出しは，各レジスタセットに保持されたコンテキストを示す．レジスタセットの内，色塗りされたレジスタセットは，そのときプロセッサで使用しているレジスタセットを示す．また，図 9 では，レジスタ 1 で先行タスクを実行し，レジスタ 3 に，先行タスク実行時に予測後続タスクのコンテキストをメモリから読み出し，レジスタ 2 で OS を実行した状態を表す．

図 9 を用いて，後続タスク予測機構が予測に失敗した場合を説明する．まず，タスク切り替えを伴うシステムコールが呼ばれて，先行タスクが中断されたときに，リアルタイム OS によって後続タスクが決定する．後続タスクが決定したら後続タスクと予測タスクを比較し，予測に失敗したと分かった場合，リアルタイム OS によって，通常通りメモリから後続タスクのコンテキストを読み出し，プロセッサが使用するレジスタセット図 9 のレジスタ 3 に書き込む．予測に失敗した場合のミスペナルティは発生しない．

4. 評価環境

提案機構によるタスク切り替え時間の削減量を求めるために，以下のような方法を用いた．

表 1　トレースカーの状態

番号	状態名	操作
1	Straight	直進する
2	RightCurve	右に曲がる (タスク 3 より 曲がる角度が小さい)
3	RightSharpCurve	右に曲がる (タスク 2 より 曲がる角度が大きい)
4	RightRecovery	右に曲がりながら ラインを探す
5	LeftCurve	左に曲がる (タスク 3 より 曲がる角度が小さい)
6	LeftSharpCurve	左に曲がる (タスク 2 より 曲がる角度が大きい)
7	LeftRecovery	左に曲がりながら ラインを探す
8	Uturn	U ターンする

(1)　コース上のライン沿って動くトレースカーを用いて状態遷移を調査

(2)　C 言語で記述した後続タスクを予測する機構のシミュレータを用いて，状態遷移から後続タスクを予測する機構の予測精度を測定

(3)　(2) で測定した予測精度からタスク切り替え時間の削減量を計算

本章では，まず，状態遷移を調査するためのトレースカーについて 4.1 節で述べる．次に，予測精度からタスク切り替え時間の削減量を計算する方法について 4.2 節で述べる．

4.1 トレースカー

後続タスクを予測する機構の予測精度を測定するために，トレースカーにおいて直進，左折などの制御を行うプログラム上の関数 (状態) をタスクとして用いた．トレースカーは，ラインを検出する 3 つのセンサからの情報を元にモータの回転数を制御することにより，ラインに沿って動く．このトレースカーをラインに沿って走行するよう制御するために，8 つの状態を用いる．8 つの状態と操作方法との対応やトレースカーの状態とタスク番号との対応を表 1 に示す．

トレースカーが走行するコースを 2 つ用意し，以下の 4 つのモデルで状態遷移を調査する．

- COURSE1：
  図 10 のコースを時計回りで走行
- COURSE2：
  図 10 のコースを反時計回りで走行
- COURSE3：
  図 11 のコースを走行
  モデル COURSE4 とはスタートの位置が異なる．
- COURSE4：
  図 11 のコースを走行
  モデル COURSE3 とはスタートの位置が異なる．

4.2 予測精度からタスク切り替え時間の削減量を計算する方法

前節の 4 つのモデル COURSE1〜4 をトレースカー

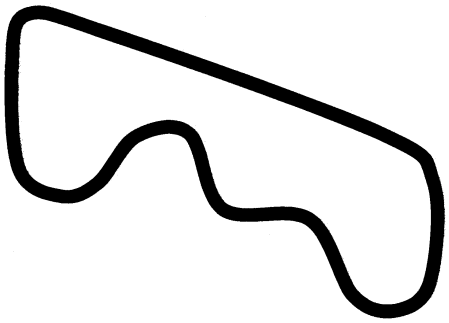


図 10　トレースカーのコース 1

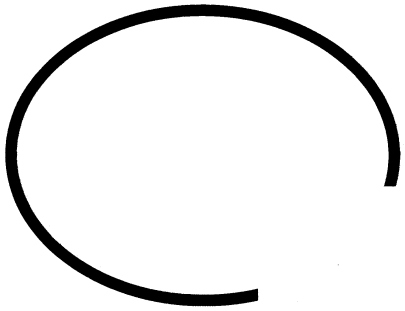


図 11　トレースカーのコース 2

が走行することにより，状態遷移データを取得した．この状態遷移データを用いて，後続タスク予測機構のシミュレータを動作させ，それぞれのコースに対する予測精度を求める．この予測精度からタスク切り替え時間の削減量を求める．提案手法では，タスクの予測が成功した場合は，3 つのレジスタセットによって，コンテキストスイッチ時間が削減され，タスク切り替え時間が短くなる．一方，タスクの予測が失敗した場合は，リアルタイム OS でコンテキストスイッチを行うため，コンテキストスイッチ時間は従来のままである．

タスク切り替えに要する時間に対する削減量を求めるために，コンテキストスイッチ時間とタスク切り替え時間の割合を求める．これにより，タスク切り替えに要する時間に占める，提案手法の最大削減時間の割合を求めることができる．

タスク切り替え時間とコンテキストスイッチ時間は，プロセッサの動作周波数が 25MHz である文献[2] では，それぞれ 11μs と 2.9μs であり，プロセッサの動作周波数が 600MHz である文献[3] では，それぞれ 480ns と 125ns である．これらよりコンテキストスイッチ時間は，タスク切り替え時間の 1/4 と仮定する．よって，タスク切り替えに要する時間に占める，提案手法の最大削減時間の割合は，1/4 となる．

次に，タスク切り替えに要する時間に占める，提案手法によるタスク切り替えの削減時間の割合を求める．提案手法では，予測に成功すれば，タスク切り替え時間を削減でき，予測に失敗すれば，タスク切り替え時間は従来のままとなる．これらにより，提案手法では，予測に成功した分だけタスク切り替え時間を削減することができる．以上により，タスク切り替えに要する時間に占める，最大削減時間の割合 $R_{max\_reduction}$ と，後続タスクを予測する機構のヒット率 P_{hit} との積が，タスク切り替えに要する時間に占める，提案手法によるタスク切り替えの削減時間の割合 $T_{reduction}$ となる．予測精度から，タスク切り替えに要する時間に占める，提案手法によるタスク切り替えの削減時間の割合を求める式を，式 (1) に示す．

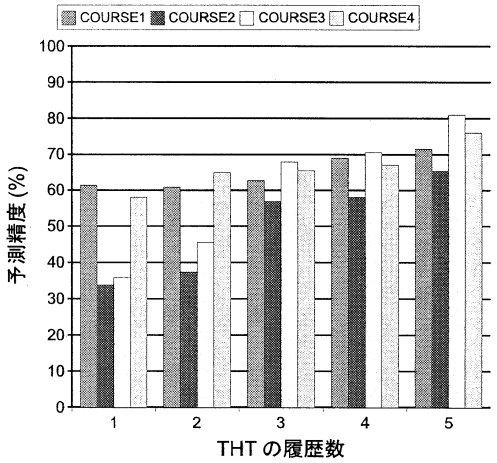


図 12　タスクを予測する機構の予測精度

$$T_{reduction} = P_{hit} \cdot R_{max\_reduction} \quad (1)$$

また，従来手法のタスク切り替えに要する時間で正規化した，提案手法のタスク切り替えに要する時間 T_{total} を式 (2) に示す．

$$T_{total} = 1 - P_{hit} \cdot R_{max\_reduction} \quad (2)$$

なお，式 (1) と式 (2) における，タスク切り替えに要する時間に占める，最大削減時間の割合 $R_{max\_reduction}$ は，1/4 とした．

5. 評価結果

5.1 予測精度

図 12 に，4.1 節で説明したトレースカーを用いて，後続タスクを予測する機構の予測精度を評価した結果を示す．図の横軸は，THT の履歴数を示し，縦軸は，後続タスクを予測する機構の予測精度を示す．4 本組になった棒グラフは，左から順に 4.1 節で説明した各モデル COURSE1〜4 で測定を行った場合である．

図 12 から，後続タスクを予測する機構の予測精度は，THT の履歴数を 5 にした場合で，65%〜81% となることがわかる．

また，図 12 より，後続タスクを予測する機構では，THT の履歴数を増やすと，予測精度が高くなるのがわかる．この理由を次のように考察する．後続タスクを予測する機構は，各タスクの遷移パターンの繰り返しを検出し，予測する機構であるため，実際のタス

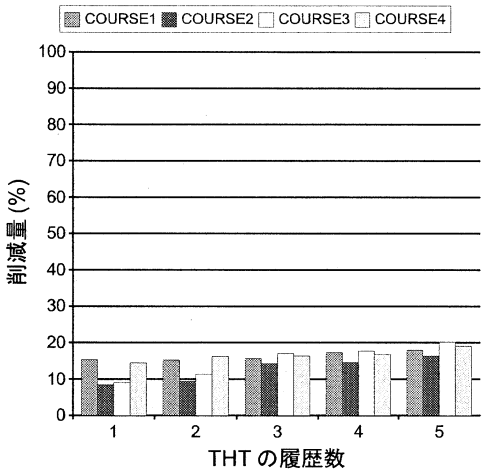


図 13 タスク切り替え時間の削減量

クの遷移パターンが，記録できる履歴長より長い繰り返しパターンを有する場合，タスクを予測する機構では，繰り返しをとらえることができない．そのために，THT の履歴長をふやせば，記録できる履歴長が増え，予測精度が向上する．

予測精度を向上させるためには，THT の履歴数を図 12 で示した値の 5 より増やして，予測精度を測定をし，予測精度の向上が飽和する THT の履歴数を見つけなければならない．しかし，THT の履歴数を増やすと，PHT のテーブルサイズが大きくなりすぎるという問題が発生する．例えば，タスク番号のビット長を $i$bit，THT の履歴長を k とした場合の THT の総履歴数を考える．まず，履歴数を 2 倍にする前の PHT の総履歴数は 2^{ik} となる．次に，履歴数を 2 倍にすると先ほどの k が $2k$ となるので，PHT の総履歴数は 2^{2ik} となり，これは，履歴数を 2 倍にする前の PHT の総履歴数の 2 乗である．よって，THT の履歴数が増加すると，PHT の総履歴数は，指数関数的に増加する．以上より，今後 PHT の総履歴数を削減する機構を追加する必要がある．

5.2 タスク切り替えに要する時間の削減量

図 13 に，4.1 節で説明したトレースカーを用いて，タスク切り替えに要する時間に占める，提案手法による削減量の割合を評価した結果を示す．図 13 の横軸は，THT の履歴数を示し，縦軸は，タスク切り替えに要する時間に占める，提案手法による削減量の割合を示す．4 本組になった棒グラフは，左から順に 4.1 節で説明した各モデル COURSE1～4 で測定を行った場合である．

図 13 では，4.2 節で説明した計算方法で，タスク切り替えに要する全実行時間に占める，提案手法による削減量の割合を求めた．そのため，図 13 では，タスクを予測する機構の予測精度が向上する場合と同じように，THT の履歴数を増やすと，タスク切り替えに要する全実行時間の削減量が大きくなるのが分かる．そのため，タスクを予測する機構の予測精度が向上すれば，タスク切り替えに要する全実行時間の削減量も大きくなる．また，図 13 から，提案手法により，タスク切り替えの削減時間の割合は，THT の履歴数を 5 にした場合で，16%～20% となることがわかる．

6. 関連研究

タスク切り替え時間を削減する手法として，複数のコンテキストを用いて実現する手法[4] や，リアルタイム OS の処理をハードウェアで実装する手法[5)6)] がある．これらの機構では，リアルタイム OS がタスク切り替え処理をするときに，リアルタイム OS の代わりに，タスク切り替えの処理の一部を専用のハードウェアで行っている．

これらに対して，我々の手法では，リアルタイム OS が主として，タスク切り替え処理を行い，提案機構は，リアルタイム OS をサポートする位置づけにある．しかしながらタスク切り替え時間をさらに削減するためにはさらなる OS 機能のハードウェア化が必要であることが分かる．

7. まとめ

コンテキストスイッチに着目して，タスク切り替え時間を削減する手法を提案した．提案機構では，コンテキストスイッチのオーバーヘッドを削減できるため，タスク切り替え時間を削減することができる．その結果，THT の履歴数を 5 とした場合で，16%～20% のタスク切り替え時間を削減できた．本論文では論旨を簡単にするために 3 つのレジスタセットを用いた．しかしながら対象が組み込みシステムであることを考えるとハードウェア量の削減が望ましい．レジスタセットを 2 つにしてもメモリアクセスを伴うため速度の低下は許容範囲の可能性があるので，今後このことも追及したい．

参考文献

[1] Jeffrey C. Mogul, Anita Borg, “The Effect of Context Switches on Cache Performance,” WRL Technical Note TN-16, 1990.

[2] “V850 シリーズ開発環境,” NEC エレクトロニクス株式会社, 2005.

[3] 中村健真, “Interface 6 月号,” CQ 出版株式会社, p60-71, 2006.

[4] 田中 清史, 松本 尚, 平木 敬, “Casablanca: 実時間処理 RISC コアの設計と実装,” 情報処理学会研究報告, ARC Vol.99, NO.100, p51-56, 1999.

[5] 仲野 巧, アンディウタマ, 坂橋 光義, 塩見 彰睦, 今井正治, “リアルタイム OS の VLSI 化とその評価,” 電子情報通信学会論文誌, NO.8, p679-686, 1995.

[6] 森 久直, 坂巻 佳壽美, 重松 宏志, “組み込み制御システム向けリアルタイム OS のハードウェア化, ” 東京都立産業技術研究報告第 8 号, p55-58, 2005.