# 動的計画法を用いたキャッシュフラッシュの最悪タイミングの探索手法

宮本寛史[†] 飯山真一[††] 冨山宏之[†]
高田広章[†] 中島浩[††]

近年，ハードリアルタイムシステムにおいてもキャッシュメモリ（以下，キャッシュ）の利用が望まれており，キャッシュを考慮した最悪実行時間解析が必要になっている．マルチタスキング環境では，タスク切替えによってフラッシュされるキャッシュライン数はそのタイミングに依存し，それによって実行時間が変化する．そのため，キャッシュを考慮した最悪実行時間解析を行うには，最も実行時間が長くなるタスク切替えタイミング（以下，最悪フラッシュタイミング）を求めることが必要である．本論文では，最悪フラッシュタイミングの探索手法を提案し，その有効性を示す．

# A Search Algorithm of Worst-Case Cache Flush Timings Using Dynamic Programming

Hiroshi Miyamoto,[†] Shinichi Iiyama,[††] Hiroyuki Tomiyama,[†]
Hiroaki Takada[†] and Hiroshi Nakashima[††]

In recent years, the use of cache memory has been desired in hard real-time systems in order to reduce the memory access time. To enable it, accurate analysis of the worst-case execution time considering cache flushes is necessary since the cache may be flushed by preempting tasks in a multitask environment. This paper proposes a method to find the worst-case timing of cache flushes and demonstrates its effectiveness.

## 1. はじめに

リアルタイムシステムは，出力される結果の正確さに加えて，処理が決められた時刻（デッドライン）までに終了するなどの時間制約を満たすことを求められるシステムである．タスクが時間制約を満たすかを確かめるには，プログラムの最悪実行時間を求めることが重要である．

リアルタイムシステムを構築する際には，最悪実行時間を安全かつ正確に求めることが望まれるが，複雑なアーキテクチャを持ったプロセッサでは，最悪実行時間を正確に求めることは一般には難しい[1)]．特に，キャッシュメモリ（以下，キャッシュ）に関しては，動作の予測が難しく，最悪実行時間を正確に求めることを困難にすることから，ハードリアルタイムシステムにおいては使用が難しかった．しかしながら近年，ハードリアルタイムシステムにおいてもキャッシュの利用が望まれるようになっており，キャッシュを考慮した安全で正確な最悪実行時間解析が必要になっている．

多くのリアルタイムシステムではマルチタスクシステムが採用されているが，この場合，あるタスクがキャッシュしたメモリブロックを，別のタスクがフラッシュする可能性がある．また，フラッシュによって生じるコストは，フラッシュのタイミング（以下，フラッシュタイミング）によって異なる．そのため，最悪実行時間を求めるためには，すべてのフラッシュタイミングの組合せに対して，その合計コストの最大値を求める必要がある．

プログラムの静的パス解析を利用して，最悪実行時間を求めるための手法は，数多く研究されている．また，それを用いてキャッシュのフラッシュコストの最大値を求める手法についても，多くの研究がある[2)~6)]．しかし，静的パス解析を用いた最悪実行時間解析は，実際には実行する可能性のないパスの実行時間を最悪

† 名古屋大学大学院情報科学研究科情報システム学専攻
Department of Information Engineering, Graduate School of Information Science, Nagoya University
†† 豊橋技術科学大学情報工学系
Department of Information and Computer Sciences, Toyohashi University of Technology

実行時間としてしまう場合があり，悲観的な結果となることが多い．そのため，このような手法が実用化されている例はほとんど報告されていない．

それに対して現実のシステム開発においては，様々な入力パターンを与えながらプログラムの実行時間を計測し，その中の最大値に一定の安全率を見込んで，最悪実行時間と見なす方法が広く用いられている．ところが，同じ手法でキャッシュのフラッシュコストの最大値を見積もるためには，様々なタイミングでフラッシュを起こして計測することが必要となり，現実的ではない．

そこで我々は，特定の入力パターンによる実行トレースから，フラッシュコストが最大になるフラッシュタイミング（以下，最悪タイミング）を効率的に求める手法を提案する．提案手法では，すべての入力パターンに対する最悪実行時間の保証はできないものの，上述した最悪実行時間を求めるための現実的な手法と組み合わせて使うことができ，実用性が高いと考えられる．すなわち，入力パターンを与えてプログラムの実行時間を計測するたびに，提案手法により最悪タイミングを求め，そのときのフラッシュコストを加算することで，キャッシュフラッシュの効果を考慮に入れることができる．

本論文では，2 章において関連研究について述べる．3 章で最悪タイミング探索問題を定義する．4 章では，定義した最悪タイミング探索問題に対して効率的な探索手法を提案する．5 章では，提案手法を用いて最悪タイミング探索を行い，評価を行う．

## 2. 関 連 研 究

現在，キャッシュメモリはリアルタイム性の低いシステムで用いられていることが多い．そのため実際の組み込みシステム開発においては，入力パターンを変えながら実行時間を計測しその中の最大値に一定の安全率を見込んで各タスクの最悪実行時間とするといった計測的な手法がとられている場合が多い．

キャッシュを利用した場合，他のタスクがキャッシュをフラッシュすることで生じるコスト（フラッシュコスト）によって実行時間が延びるため，入力パターンだけでなく，他のタスクがキャッシュをフラッシュするタイミング（フラッシュタイミング）もタスクの実行時間に影響を与える．そのためフラッシュタイミングを変えて実行時間を計測する必要がある．

しかしながら，フラッシュタイミングは他のタスクによって決定されるため，シミュレーションなどによっても任意のタイミングでフラッシュが発生する状況を作り出すことが困難である．結果としてフラッシュコストの見積りの精度は低くなるため，よりリアルタイム性の高いシステムでは，従来の計測的な手法を適用することは適切ではない．

キャッシュを考慮した最悪実行時間解析手法の研究として静的パス解析を用いた研究がある．Lee らの手法[2),3)] は第 1 段階として，プログラムの基本ブロックごとにフラッシュコストを求める．具体的には静的パス解析を行い，各基本ブロックに対してその時点でフラッシュが生じたときにキャッシュミスを増加させる可能性のあるメモリブロックの数を計算する．第 2 段階では第 1 段階で求めたフラッシュコストを用いて線形計画問題を解くことで最悪応答時間を求める．文献 4)，5) ではそれぞれ Lee らの手法を基本としながら，フラッシュコストの見積りをより正確にすることで最悪応答時間を正確に求める手法が提案されている．文献 6) では静的パス解析を行った後，整数線形計画法を適用して，最もキャッシュラインを多く使用する実行パスを求めることでキャッシュを考慮した最悪性能解析を行う手法が提案されている．

これらの静的パス解析を用いた最悪実行時間解析は，実際には実行する可能性のないパスの実行時間を最悪実行時間としてしまう場合があり，悲観的な結果となることが多い．そのため，このような手法が実用化されている例はほとんど報告されていない．

本論文が提案する手法は，特定の入力パターンから得られた実行トレースに対してフラッシュタイミングの組合わせを網羅的に探索することによって最悪タイミングを安全に求めることができる．提案手法は従来使われてきた手法と組み合わせて用いる．すなわち，入力パターンを変えながら実行時間を測定した後，その過程で得られた実行トレースに対して本手法を適用することにより，キャッシュがフラッシュされる影響を加味することができる．従来行われてきた方法と異なり，フラッシュが生じる可能性のあるタイミングはすべて網羅される．そのため解析に用いた実行トレースに対しては，フラッシュコストを安全に見積もることができる．これを多くの実行トレースに対してこれを行うことで最悪実行時間の見積りの精度が高くなり，高いリアルタイム性が要求システムに対しても適用できると考えられる．

## 3. 問 題 定 義

本論文ではシングルレベル，ライトスルーキャッシュを前提とする．また，解析を行うにあたって，ある入力パターンを与えて実行したときの実行トレース，そ

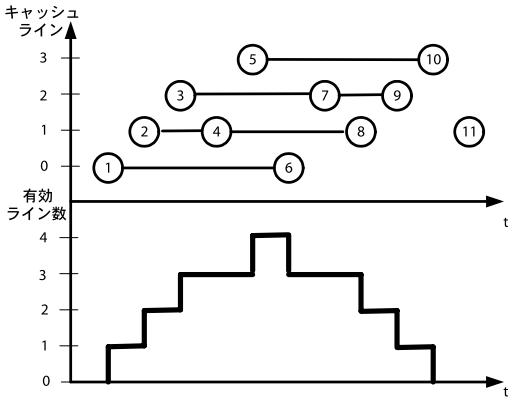



**図 1** 有効なライン
Fig. 1 Effective cache lines.

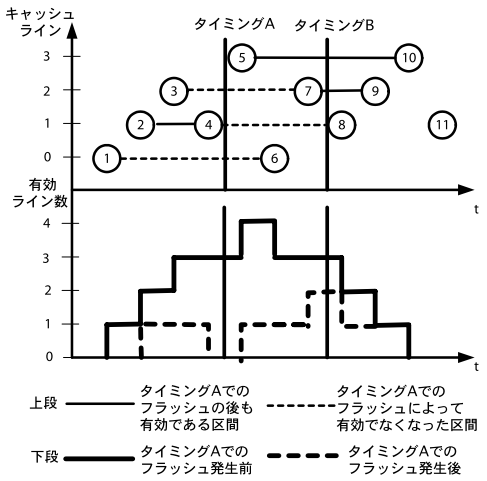



**図 2** フラッシュコスト
Fig. 2 Flush cost.

れから得られるキャッシュアクセス☆のログが得られているとする．本章では，この前提に基づいて最悪タイミング探索問題を定義する．まず，有効なラインの概念を説明し，フラッシュコストを定義する．次に，最悪タイミング探索問題を定義する．

### 3.1 有効なライン

キャッシュラインが将来参照されるメモリブロックを保持している間，そのラインは**有効**であるという☆☆．**図 1** は，キャッシュラインの有効性を視覚的に表現する．図 1 の横軸は時刻で，上段の縦軸はキャッシュライン番号を，下段の縦軸は有効ライン数を示している．上段に示される円はアクセスを意味し，円の中の数字は何番目のアクセスであるかを示している．この図の上段では，キャッシュラインが有効である期間を実線で示している．たとえば，上段の 1 番目のアクセスと 6 番目のアクセスはキャッシュライン 0 へのアクセスであり，その間が実線になっているため 1 番目のアクセスから 6 番目のアクセスまでの間キャッシュライン 0 は有効であることを意味している．

### 3.2 フラッシュコスト

本論文ではライトスルーキャッシュを想定しているため，フラッシュコストはフラッシュによって増加するキャッシュミスペナルティのみである．キャッシュミスが増加するのは，有効なラインがフラッシュされた場合である．有効でないラインは，将来参照しないメモリブロックを保持しているため，フラッシュされてもキャッシュミスは増加しない．本論文ではコストを安全側に見積もるために，プリエンプトされた時点で有効なキャッシュラインはすべてフラッシュされると想定する．そのため，フラッシュコストはプリエンプトされた時点での有効ライン数にキャッシュミスペナルティ（以降では，キャッシュミスペナルティを 1 とする）をかけたものになる．**図 2** では，タイミング A においてフラッシュが生じるとキャッシュライン 0，1，2 が有効であるため，そのコストは 3 である．

### 3.3 最悪タイミング探索

最悪タイミング探索は，フラッシュコストの和が最大になるタイミングの組合せの探索である．しかし，2 回以上のフラッシュが生じる場合，単純に有効ライン数のグラフから各タイミングの有効ライン数を求める方法はとらない．あるタイミングにおけるフラッシュは他のタイミングにおける有効ライン数も減少させるためである．図 2 でこれを説明する．図中タイミング A におけるフラッシュコストは 3 である．次に図中タイミング B におけるフラッシュコストを求める時点では，タイミング A におけるフラッシュによって，図上段において点線で示される部分でキャッシュラインが有効でなくなっている．タイミング B における有効ライン数は点線で示されるように減少するため，タイミング B におけるフラッシュコストは 2 になる．

2 回以上フラッシュが生じると想定する場合，あるフラッシュタイミングの組合せに対してフラッシュコストの和を求める際に，個々のタイミングにおけるコストを求めると同時にそのフラッシュによって有効でなくなるラインを求め，そのうえで次のフラッシュによるコストを求めるという手順を踏む．本論文では，

☆ 命令フェッチ，ロード/ストア命令によるアクセス．また，キャッシュミス発生時には，必ずそのメモリブロックをロードすると想定する．

☆☆ 通常，ラインが有効であるとはそのラインがデータを保持していることを意味するが，本論文ではここで定義した意味で使用する．

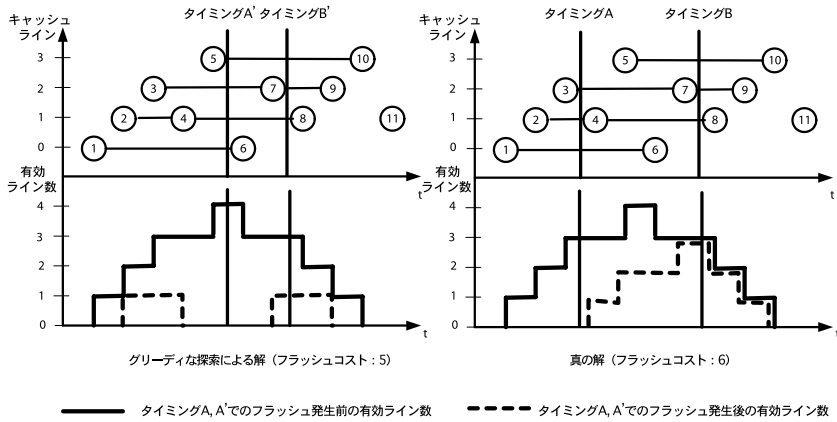



**図 3** 真の解が求められない場合
Fig. 3 Failure of greedy algorithm.

最悪タイミング探索を上述した手順に従って求められるフラッシュコストの和を目的関数として，これを最大化するタイミングの組合せを探索する組合せ最適化問題として考える．

## 4. 提案手法

本章では，上述した最悪タイミング探索問題を効率的に解く手法を提案する．まず，探索手法の方針について説明する．次にその方針に従って問題を形式的に扱えるようにするための表記法や定義について説明する．最後に，動的計画法を利用したアルゴリズムとそれをより効率的なものにする手法を提案する．

### 4.1 方針

本論文では，探索手法は前章で定義した最悪タイミング探索問題に対して確実に真の最悪タイミングが求めることができるものにする．そのため，高速であっても，確実に真の最悪タイミングが求められないアルゴリズムは適切ではない．たとえば，最悪タイミング探索をグリーディに行うアルゴリズムとして，有効ライン数が最大であるタイミングの選択とそのタイミングにおけるフラッシュを繰り返して最悪タイミングを求めるアルゴリズムが考えられる．アクセス回数が $N$ 回でフラッシュ回数が $F$ 回の場合，有効ライン数最大のタイミングの探索には $O(N)$ 個のタイミングを探索する必要があり，それを $F$ 回繰り返すため計算量は $O(NF)$ である．フラッシュが発生するタイミングの組合せの総数は ${}_NC_F$ であるため，これを全数探索する場合の計算量は $O(N^F)$ である．よって，グリーディな探索アルゴリズムは全数探索する場合と比較して高速であるといえるが，真の最悪タイミングが求められない場合がある．**図 3** は，グリーディな探索アルゴリズムでは真の最悪タイミングが求められない例である．フラッシュ回数が 2 回の場合の最悪タイミングは，図に示されるタイミング（タイミング A，タイミング B）であり，フラッシュコストはそれぞれ 3 であるため，その和は 6 になる．グリーディな探索アルゴリズムでは，有効ライン数が最大になるタイミングの探索を繰り返すため，1 つ目は有効ライン数が最大になるタイミング（タイミング A'）を選択し，2 つ目のフラッシュタイミングには，タイミング A' におけるフラッシュの後での有効ライン数が最大になるタイミング（タイミング B'）が選択される．フラッシュコストはそれぞれ，4 と 1 であるためフラッシュコストの和は 5 になり，真の最悪タイミングを求めることができていない．

前章で述べたように個々のタイミングにおけるフラッシュのコストは，他のタイミングにおけるフラッシュに影響を受けて独立には定まらないため，最悪タイミング探索問題は本質的にすべてのタイミングの組合せの探索が必要な問題であるといえる．そのため本論文では，網羅的な探索を基本とした効率的なアルゴリズムを提案する．

### 4.2 表記法

本節では表記法を説明するが，最悪タイミング探索は，アクセス回数が $N$ のアクセスログに対して，フラッシュ回数が $F$ である場合の最悪フラッシュタイミングとフラッシュコストを求めるものとして説明する．

#### 4.2.1 フラッシュタイミングとアクセス

$t$ 番目のアクセスを $a_t, 1 \leq t \leq N$ と表記し，$a_t$ と $a_{t+1}$ の間のタイミングを $t$ と表記する．また，タイミング $t$ に対して，その直後のアクセス $a_{t+1}$ と同じキャッシュラインへの次のアクセスを $A$ とすると，$A$

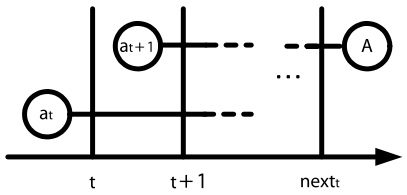



**図 4** フラッシュタイミングとアクセス
Fig. 4 The flush timings and accesses.

の直前のタイミングを $next_t$ と表記する．**図 4** にこれらのアクセスとタイミングの関係を図示する．$a_1$ の前と $a_N$ の後のタイミングにおいて有効ライン数は必ず 0 であり，最悪フラッシュタイミングの候補から除いてもよい．しかし，後述するフラッシュコストの定義で $a_1$ の前のタイミングであるタイミング 0 を利用する場合があるため $(0 \leq t < N)$ の範囲の $N$ 個のタイミングをフラッシュタイミングの候補とする．

#### 4.2.2 フラッシュコスト

タイミング $\hat{t}$ でフラッシュが生じた状態で，次のフラッシュがタイミング $t$ で生じた場合のフラッシュコストを次のように表記する．

$C(\hat{t}, t)$

この表記は，$t$ 以前にフラッシュが生じている場合のコストを示すものである．しかし，$t$ 以前でフラッシュが生じていない場合のフラッシュコストも $C(0,t)$ と表記することで表現することが可能である．タイミング 0 におけるフラッシュのコストは 0 であり，タイミング $t$ におけるフラッシュのコストに影響を与えないため，フラッシュが生じていない状態と等しいと考えてよいためである．

#### 4.2.3 最悪コスト

タイミング $t$ でフラッシュが生じた状態で $t$ 以降で $f$ 回のフラッシュが生じる場合のフラッシュコストの和の最大値を最悪フラッシュコスト（以下，最悪コスト）とし，$Cw(t,f)$ と表記する．また，$Cw(t,f)$ は，以下のように再帰的に定義することができ，本論文ではこの式を計算することで最悪コストを求めている．

$$Cw(t,f) = \begin{cases} 0 & \\ f = 0 & \\ \max\limits_{t<t'<N} \{C(t,t') + Cw(t', f-1)\} & \\ f > 0 & \end{cases} \quad (1)$$

### 4.3 動的計画法を利用したアルゴリズム

本節では，最悪コスト $Cw(0,F)$ を動的計画法によって計算できることを説明する．動的計画法とは，

```
/*最悪コストを求めるアルゴリズム*/
Cw[N][F+1];
for(t=N-1;t>=0;t--){
    tにおけるフラッシュ;
    for(f=1;f<=F;f++){
        max = 0;
        for(t'=t+1;t'<=N-1;t'++){
            if(max<C(t,t')+Cw[t'][f-1]){
                max = C(t,t')+Cw[t'][f-1];
            }
        }
        Cw[t][f]=max;
    }
}
```

**図 5** 動的計画法を利用したアルゴリズム
Fig. 5 The algorithm with dynamic Programing 1.

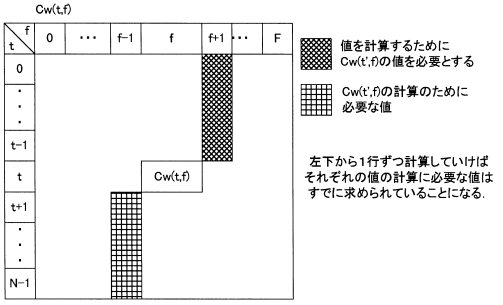



**図 6** アルゴリズム 1 の動作
Fig. 6 The behavior of Algorithm 1.

部分解を計算して表に記憶しておき，後でその値が必要になったときに表を引くことで，効率的に元の問題の解を求めることのできる手法である．式 (1) より，最悪コスト $Cw(t,f)$ は部分解 $Cw(t', f-1)$ を計算することで求められるため，この手法を計算に適用できる．**図 5** に $Cw(0,F)$ を動的計画法によって求めるアルゴリズムを示す．

アルゴリズムの動作を説明する．式 (1) より，$Cw(t,f)$ の値を計算するときには**図 6** が示す範囲の値が求められている必要がある．図 5 に示したアルゴリズムは，$Cw(N-1,0)$ から計算し，表の左下から 1 行ずつ計算していくことになるため，それぞれの値を計算するときには必要な値がすべて求められていることになる．

次にこのアルゴリズムの計算量を考える．まず，‘‘$t$ におけるフラッシュ; ’’ について，$t$ でフラッシュされた状態で $t'$ においてフラッシュが生じた場合のコスト（C(t,t')）を計算するために必要なデータ構造の変更を行う操作であり，計算量 $O(1)$ で行える．また，$C(t,t')$ の計算も $t'$ が 1 ずつ変化するため $O(1)$ で行える．$C(t,t') + Cw[t'][f-1]$ の最大値を求めるためには図 6 に示される部分を走査する必要がある

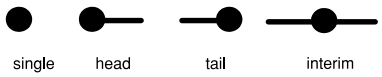



**図 7** アクセスの分類
Fig. 7 The classification of accesses.

ため，計算量が $O(N)$ 必要である．これをすべての $t, f, 0 \leq t \leq N-1, 0 \leq f \leq F$ に対して行うため全体の計算量は $O(N^2F)$ である．

### 4.4 探索空間の絞込み

前節では，最悪コストの計算に動的計画法を適用できることを説明した．本節では探索空間を絞り込むことによって，より効率的に最悪タイミングを求める手法を提案する．

3 章において，図 1 を使って最悪タイミング探索を図式的に説明した．この図においてそれぞれのアクセスは以下に示すように，その前後におけるキャッシュラインの有効性から**図 7** に示す 4 種類に分類できる．

*single*

11 番目のアクセスのように，そのアクセスの前後でキャッシュラインが有効でないアクセス．

*head*

1，2，3，5 番目のアクセスのように，そのアクセスより前ではキャッシュラインは有効ではなく，そのアクセスの後ではキャッシュラインが有効になっているアクセス．

*tail*

6，8，9，10 番目のアクセスのように，そのアクセスより前ではキャッシュラインは有効で，そのアクセスの後ではキャッシュラインが有効ではないアクセス．

*interim*

4，7 番目のアクセスのように，そのアクセスの前後でキャッシュラインが有効になっているアクセス．

これらの各アクセスの分類は静的に定められるものではない．例として図 2 において，8 番目のアクセスはタイミング A におけるフラッシュが生じる前は *tail* に分類されるが，フラッシュの後では *single* に分類される．

このアクセスの分類を利用することで，最悪コストを計算するときの探索空間を絞り込むことができる．**図 8** に効率化されたアルゴリズムを示す．このアルゴリズムの動作は基本的に前節で示したものと同じであり，最悪コストの表を下から順に埋めていく．しかしながら，前節で示したアルゴリズムが基本的に $Cw(t,f)$ の定義に従って図 6 に示される部分をすべて走査するのに対して，効率化されたアルゴリズムは，**図 9** に示すように走査する範囲を限定しており，限定できる範囲は，$a_t$ がどの種類に分類されるかによって決定される．

```
/*最悪コストを求めるアルゴリズム*/
Cw[N][F+1];
for(t=N-1;t>=0;t--){
    tにおけるフラッシュ;
    if(at+1がsingle またはtail){
        for(f=1;f<=F;f++) Cw[t][f]=Cw[t+1][f];
    }else{
        for(f=1;f<=F;f++){
            max = Cw[t+1][f];
            for(t'=t+1;t'<=nextt;t'++){
                if(max<C(t,t')+Cw[t'][f-1]){
                    max = C(t,t')+Cw[t'][f-1];
                }
            }
            Cw[t][f]=max;
        }
    }
}
```

**図 8** 動的計画法を利用したアルゴリズム 2
Fig. 8 The algorithm with dynamic Programing 2.

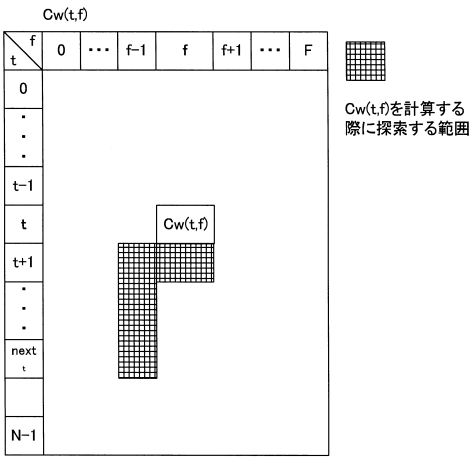



**図 9** アルゴリズム 2 の動作
Fig. 9 The behavior of Algorithm 2

走査する範囲を限定できる理由について説明する．まず，$C(t,t')$ は以下の式を満たす．

$a_{t+1}$ が *single* または *tail* である場合．

$$C(t,t') = C(t+1,t') \tag{2}$$

$a_{t+1}$ が *head* または *interim* である場合．

$$C(t,t') = \begin{cases} C(t+1,t')+1 & t+1 < t' \leq next_t \\ C(t+1,t') & next_t < t' < N \end{cases} \tag{3}$$

式 (2) は，$a_{t+1}$ が *single* または *tail* である場合では，

$t$ でフラッシュが生じる場合と $t$+1 でフラッシュが生じる場合とでは，それ以降で生じるフラッシュに対する影響が同じであるため，コストが等しくなることを示している．式 (3) は，$a_{t+1}$ が *head* または *interim* である場合では，コストが等しくなる範囲とコストが 1 だけ増える範囲があることを示している．この性質を利用して走査する範囲を限定できることを説明する．まず，$a_{t+1}$ が *single* または *tail* である場合は，$Cw(t,f) = Cw(t+1,f)$ としている．これは，式 (2) と最悪コストの定義の式 (1) より，自明である．次に $a_{t+1}$ が *head* または *interim* である場合は，$Cw(t+1,f)$ と $max_{t<t'\leq next_t}\{C(t,t') + Cw(t',f-1)\}$ を含めた中での最大値を $Cw(t,f)$ としている．これは，次の定理が成り立つためである．

**定理**

$a_{t+1}$ が *head* または，*interim* である $t, t < N-1$ に対して以下の式が成り立つ．

$$\begin{aligned} &Cw(t,f) \\ &= \max\{\max_{t<t'\leq next_t}\{C(t,t') + Cw(t',f-1)\}, \\ &\qquad Cw(t+1,f)\}\} \end{aligned} \tag{4}$$

**証明**

$$\begin{aligned} &Cw(t,f) \\ &= \max_{t<t'<N}\{C(t,t') + Cw(t',f-1)\} \\ &=\max\{\max_{t<t'\leq next_t}\{C(t,t') + Cw(t',f–1)\}, \\ &\quad \max_{next_t<t'\leq N}\{C(t,t') + Cw(t',f–1)\}\} \end{aligned}$$

式 (3) より，$C(t,t') = C(t+1,t'), t+1 < t' \leq next_t$

$$\begin{aligned} &=\max\{\max_{t<t'\leq next_t}\{C(t,t') + Cw(t',f–1)\}, \\ &\quad \max_{next_t<t'\leq N}\{C(t+1,t') + Cw(t',f–1)\}\} \end{aligned}$$

ここで式 (3) より，$C(t,t') = C(t+1,t')+1, t+1 < t' \leq next_t$ であるため以下が成り立つ．

$$\begin{aligned} &\max_{t+1<t'\leq next_t}\{C(t+1,t'), Cw(t',f-1)\} \\ &\leq \max_{t+1<t'\leq next_t}\{C(t,t'), Cw(t',f-1)\} \end{aligned}$$

よって

$$\begin{aligned} &Cw(t,f) \\ &=\max\{\max_{t<t'\leq next_t}\{C(t,t') + Cw(t',f-1)\}, \\ &\qquad Cw(t+1,f)\}\} \end{aligned}$$

□

以上のように，$Cw(t,f)$ を計算するためには定義に従ってすべての範囲を探索する必要はなく，探索する範囲を限定して効率的に計算することができる．

次に，このアルゴリズムの計算量を求める．図 9 には $Cw(t,f)$ を計算するときに走査する必要がある部分を示している．ある $t$，$f$ に対して $Cw(t,f)$ を求めるときの計算量は $next_t - t$ の平均値を $L_{mean}$ とすると $O(L_{mean})$ であり，これをすべての $t$，$f$ に対して行うため，全体の計算量は $O(NFL_{mean})$ であることが分かる．ここで，$L_{mean}$ は同じキャッシュラインへのアクセスが前後のアクセスと時間的にどの程度離れているかを表すものである．プログラムにおけるデータアクセスには一般に時間的局所性が見られることが多いため，$L_{mean}$ が $N$ に比例して大きくなるとは考えにくく，むしろ定数であると予想されるためアルゴリズムの計算量は実質的に $O(NF)$ に近くなると考えられる．

最後に $Cw(t,f)$ の値を保持する表のメモリ使用量について述べる．前節のアルゴリズムでは表のメモリ使用量は $O(NF)$ であるが，探索空間の絞り込みを行うアルゴリズムでは，計算に必要なデータ以外は保持しないように最適化することでメモリ使用量を減らすことが理論的に可能である．

図 9 に示すように $Cw(t,f)$ を計算するときに必要なデータは $Cw(t+1,f-1), ...Cw(next_t,f-1)$ のみである．そのため，ある $t$，$f$ に対して $Cw(t,f)$ を求めるために必要なデータに要するメモリ容量は $O(next_t - t)$ である．これが最大になる場合のメモリ容量を確保しておけばよいため，$next_t - t$ の最大値を $L_{max}$ とすると，表のメモリ使用量は $O(L_{max}F)$ に抑えることができる．ただし，このメモリ使用量を削減する最適化の有効性は，問題のサイズやマシンのメモリ容量によって異なるため，それらを考慮して適用を決めるべきである．次章における評価では問題のサイズからメモリ使用量が物理メモリに収まる範囲であると判断し，この最適化を適用していない．

## 5. 評　　価

我々は，4 章で示した探索手法に基づいたプログラムを作成し，実際に最悪タイミング探索を実行し，探索時間などの評価を行った．

### 5.1 探索時間の評価

評価は，ベンチマークを実行して得られるキャッシュアクセスログを入力として最悪タイミング探索を行って探索時間を調査した．キャッシュアクセスログを得るツールは SimpleScalar ツールセット[7] を改造して作成した．ベンチマークには，SPEC95 ベンチマーク[8] を使用した．評価はアクセス回数を変えて行うため任意のアクセス回数のキャッシュアクセスログを得る必要がある．本論文では SPEC95 ベンチマークを使用

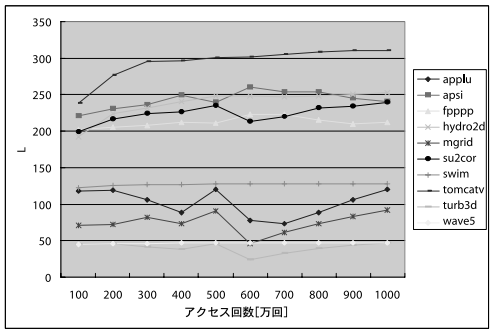



**図 10** $L_{mean}$ の変化（fp）
Fig. 10 The transition of $L_{mean}$ (fp).

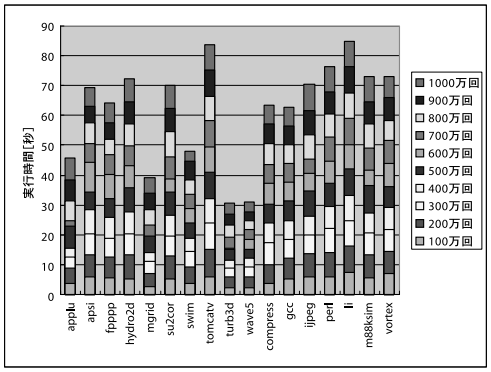



**図 11** アクセス回数による探索時間の推移
Fig. 11 The transition of search time by access.

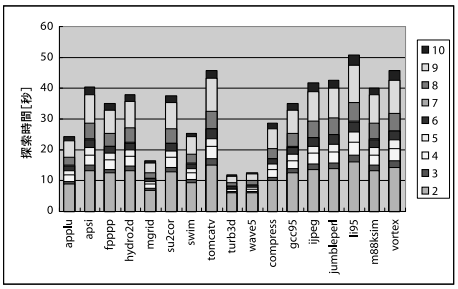



**図 12** フラッシュ回数による探索時間の推移
Fig. 12 The transition of search time by flush.

する場合，アクセス回数が 1,000 万回のキャッシュアクセスログをあらかじめ得ておき，そこから任意のアクセス回数に相当する部分を抜き取って使用した．最悪タイミング探索ツールは，CPU が Xeon 2.53 GHz，メモリ 1 GB のマシンを用いて Cygwin 上で実行し，time コマンドを用いて探索時間を計測した．

### 5.2 評価結果

前章において，探索空間の絞り込みを行うアルゴリズムは計算量が $O(NFL_{mean})$ であるが $L_{mean}$ が定数であるため，実際には $O(NF)$ になると予想した．これを検証するために，SPEC95 ベンチマークの各ベンチマークのアクセス回数が 100 万回から 1,000 万回までのアクセスログに対して，$L_{mean}$ を調査した．**図 10** に fp の結果を示す．この結果からは，多くのベンチマークにおいて $L_{mean}$ はアクセス回数によらず，一定であることが分かる．それ以外のベンチマークも一定値に収束していく傾向が見られる．前章ではデータアクセスにおける時間的局所性の存在から $L_{mean}$ の値が一定であると予想したが，実際の調査によってもほぼ一定であるか，一定値に収束する傾向があることが確認できた．また，$L_{mean}$ の値が数百程度であることが確認できたため，探索空間を絞り込む効果は非常に大きいと考えられる．なぜなら，探索空間の絞り込みを行わない場合は，平均して $\frac{N}{2}$ の範囲を探索する必要があるためである．アクセス回数が 100 万回の場合では，$\frac{1}{1000}$ に探索空間を絞り込むことができているということになる．

次に，実際に最悪タイミング探索を実行し，探索時間を調査した．**図 11** にアクセス回数が 100 万回から 1,000 万回までのログに対して，フラッシュ回数が 2 回生じると想定して最悪タイミング探索を行ったときの探索時間を示しているが，ほぼ $N$ に比例して探索時間が推移していることが分かる．

**図 12** は，アクセス回数が 200 万回のログに対して，フラッシュ回数が 2 回から 10 回生じる場合の最悪タイミング探索を実行し，その探索時間を調査したものである．キャッシュアクセスが 200 万回行われるプログラムに対して，10 回フラッシュが発生する場合でも 1 分以下で探索できることが分かる．この場合でもメモリ使用量は約 140 MB であったため，仮にフラッシュ回数が 50 回程度ならば，メモリ使用量は物理メモリ上に収まる程度で済むと考えられ，高速な探索が行えることが期待できる．また，仮にアクセス回数が 100 万回の場合ならば，フラッシュ回数が約 100 回の場合でも探索は数分で行えると思われる．100 MIPS のプロセッサで数ミリ秒実行されるタスクのキャッシュアクセス回数が 100 万回程度であり，このタスクを 100 回プリエンプトするタスクがあるとすると，その周期は数十マイクロ秒程度であると考えられる．このようなタスクがキャッシュをフラッシュする影響までも解析できることになるため，タスクセットやスケジューリングアルゴリズムによって差はあるものの，本手法は多くのシステムにおいて適用可能であるといえる．

### 5.3 グリーディな探索手法との比較

グリーディな探索アルゴリズムによる解の誤差を調査した．アクセス回数が 100 万回のキャッシュアクセ

**表 1** グリーディな探索手法の精度
Table 1 Accuracy of greedy algorithm.

| ベンチマーク名 | hydro2d | su2cor | swim |
|---|---|---|---|
| 一致しないパターン数 | 2 | 2 | 1 |
| 最大誤差 (%) | 2.1 | 5.7 | 0.4 |
| ベンチマーク名 | tomcatv | ijpeg | li |
| 一致しないパターン数 | 1 | 1 | 2 |
| 最大誤差 (%) | 3.3 | 0.2 | 0.3 |

スログ 10 パターンに対し，フラッシュ回数が 2 回の場合の最悪タイミング探索を行い，真の最悪コストを求められなかったパターンの数とそのときの最悪コストの誤差を測定した．**表 1** にその結果を示す．この表には最悪コストを求められなかったパターンが 1 つ以上あるベンチマークの結果のみをあげており，調査した 17 のベンチマークのうち 11 のベンチマークはグリーディな探索手法によってすべてのパターンの真の最悪コストを求めることができた．しかし，最大で 6%の誤差が生じる場合がある．そのため，グリーディな探索アルゴリズムに対して，提案手法は真の最悪タイミングを確実に求められるという点で妥当であることが証明された．

## 6. おわりに

本論文では，キャッシュアクセスログからキャッシュフラッシュの最悪タイミングを効率的に求める手法について考察した．まず，有効なライン，フラッシュコストなどを定義した．

続いて，最悪タイミング探索問題を定式化し，動的計画法によって最悪タイミングを求める手法を提案した．また，提案手法が探索空間を絞り込む手法の適用によって，実質的な計算量が $O(NF)$ になることを示した．

最後に提案手法の評価を行った．実際に最悪タイミング探索を行った結果，探索に要した時間は $O(NF)$ に従うことが確認できた．また，アクセス回数が 200 万回でフラッシュ回数が 10 回の場合でも最悪タイミング探索を 1 分以下で行うことができ，本手法は多くの組み込みシステムにおいて適用可能であるといえる．

今後の課題としては，実際のハードリアルアイムアプリケーションに適用して評価することや，フラッシュコストを見積もる方法をさらに正確なものにすることなどがあげられる．

**謝辞** 本研究の一部は（株）半導体理工学研究センターの支援による．また，動的計画法の適用に関して指摘してくださった ACS 論文誌査読者に感謝します．

## 参考文献

1) Puschner, P. and Burns, A.: A Review of Worst-Case Execution-Time Analysis, *Real-Time Systems*, Vol.18, No.2/3, pp.115–128 (2000).
2) Lee, C.-G., Hahn, J., Seo, Y.-M., Min, S.L., Ha, R., Hong, S., Park, C.Y., Lee, M. and Kim, C.S.: Analysis of Cache-Related Preemption Delay in Fixed-Priority Preemptive Scheduling, *IEEE Trans. Comput.*, Vol.47, No.6, pp.700–713 (1998).
3) Lee, C.-G., Hahn, J., Seo, Y.-M., Min, S.L., Ha, R., Hong, S., Park, C.Y., Lee, M. and Kim, C.S.: Enhanced Analysis of Cache-related Preemption Delay in Fixed-priority Preemptive Scheduling, *Proc. 18th IEEE Real-Time Systems Symposium*, pp.187–198 (1997).
4) Tan, Y. and Mooney, V.: Integrated Intra- and Inter-Task Cache Analysis for Preemptive Multi-tasking Real-Time Systems, *Proc. 8th International Workshop on Software and Compilers for Embedded Systems*, pp.182–199 (2004).
5) Negi, H.S., Mitra, T. and Roychoudhury, A.: Accurate estimation of cache-related preemption delay, *Proc. 1st IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, pp.201–206 (2003).
6) Tomiyama, H. and Dutt, N.: ILP-Based Program Path Analysis for Bounding Worst-Case Inter-Task Cache Conflicts, *IEICE Trans. Inf. Syst.*, Vol.E87-D, No.6, pp.1582–1587 (2004).
7) SimpleScalarLLC. http://www.simplescalar.com/
8) SPEC — Standard Performance Evaluation Corporation. http://www.specbench.org/

(平成 17 年 4 月 28 日受付)
(平成 17 年 8 月 3 日採録)

**宮本 寛史**

2003 年名古屋大学工学部電気電子情報工学科卒業．現在同大学大学院情報科学研究科情報システム学専攻博士前期課程に在学．組み込みリアルタイムシステムに関する研究に従事．

**飯山　真一**（正会員）

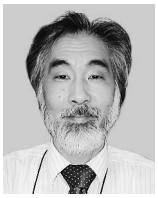
2005 年豊橋技術科学大学大学院電子・情報工学専攻博士後期課程修了．現在，リアルタイムスケジューリング理論とその応用に関する研究に従事．博士（工学）．

**冨山　宏之**（正会員）

1999 年 3 月九州大学大学院システム情報科学研究科博士後期課程修了．同年米国カリフォルニア大学アーバイン校客員研究員．2001 年（財）九州システム情報技術研究所研究員．2003 年名古屋大学大学院情報科学研究科講師．現在同助教授．SOC や組み込みシステムの設計技術に関する研究に従事．電子情報通信学会，ACM，IEEE 各会員．博士（工学）．

**高田　広章**（正会員）

名古屋大学大学院情報科学研究科教授．東京大学助手，豊橋技術科学大学助教授等を経て，2003 年 4 月より現職．博士（理学）．組み込みシステム開発技術の研究に従事．ITRON 仕様の標準活動に参加し，$\mu$ ITRON4.0 仕様等のとりまとめを行う．オープンソースの ITRON 仕様 OS 等を開発する TOPPERS プロジェクトの会長をつとめる．2000 度坂井記念特別賞受賞．情報処理学会組込みシステム研究グループ主査．電子情報通信学会，ソフトウェア科学会，ACM，IEEE 各会員．



**中島　　浩**（正会員）

1981 年京都大学大学院工学研究科情報工学専攻修士課程修了．同年三菱電機（株）入社．推論マシンの研究開発に従事．1992 年京都大学工学部助教授．1997 年豊橋技術科学大学教授．並列計算機のアーキテクチャ等並列処理に関する研究に従事．工学博士．1988 年元岡賞，1993 年坂井記念特別賞受賞．情報処理学会計算機アーキテクチャ研究会主査，同論文誌コンピューティングシステム編集委員長等を歴任．IEEE-CS，ACM，ALP，TUG 各会員．