計算機アーキテクチャ　80－4
（１９９０．１．２５）

キャッシュ操作明示化の提案

佐藤　正樹　　有田　隆也　　曽和　将容

名古屋工業大学　　電気情報工学科

計算機の最高性能を引き出すためには，キャッシュ・メモリは不可欠なものである．しかし，従来のキャッシュ・システムではキャッシュ・ミスがおこり，コンピュータシステムの性能向上に対する大きな障害となっている．

本研究は，キャッシュ操作をプログラムのなかに埋め込み明示化することによってキャッシュ・ミスを無くそうとするものである．本システムでは起こりうる動作は，コンパイラによって可能な限り解析されており，実行時にはすでにキャッシュ操作に関わるスケジュールがなされてしまっている．したがって，キャッシュ・ミスを極限まで少なくすることが出来る．本稿では，その原理，実現方法，本提案の優位な点などについて述べる．

The Cache under the Control of Program

Masaki SATO　Takaya ARITA　Masahiro SOWA

Department of Electrical Engineering and Computer Science
Nagoya Institute of Technology

Gokiso,Nagoya 466,Japan

Caches are becoming a more important concern in maximizing overall machine performance. Cache missesare inevitable in traditional cache systems, and they cause a loss of computer system performance.

This study focuses on reducing cache misses using the cache management instructions. The compiler analyzes the behavior of the programs and schedules the cache management instructions at appropriate places to reduce the cache misses. This paper presents principle, implementation, and advantages of the cache under the control of program.

1．はじめに

メモリの速度は，プロセッサに比べて一般に遅く，この速度の不平衡を解決する一つの方法としてメモリの階層化がある．メモリの階層化は，大容量で速度の遅いメモリと小容量で速度の速いメモリを組み合わせることにより，実効的に大容量で速度の速いメモリ実現しようとするものである．

このメモリの階層化の中で，最も重要なものの一つにキャッシュ・メモリがある．キャッシュ・メモリは，"主メモリとプロセッサの間に小容量で速度の速いメモリを設けることで，主メモリの性能を等価的に高速なものとする"，ということを狙ったものであり，この小容量で速度の速いメモリのことをキャッシュ・メモリと呼んでいる．キャッシュ・メモリは，プログラムに局所性があり，短い時間に限ればアドレスの変動する範囲が広くない場合に効果を発揮する．

近年ではクロック周波数の上昇や（パイプラインなどの並列処理による）1命令実行時間の短縮などで，プロセッサの処理能力は向上してきている．しかし，その処理能力に見合う命令やデータの供給ができるかどうかが問題となってきている．十分な命令とデータをプロセッサに供給できなければ，十分な性能を得ることができない．この十分な命令やデータの供給を行うために，キャッシュ・メモリの役割は非常に重要なものとなってきている．キャッシュ・ミスを起こすことがなければ，パイプラインを乱すことなく処理を続けることができることになる．しかし，従来のキャッシュ・メモリ方式では，キャッシュ・ミスを避けることは困難である．

本論文では，高級言語によるプログラムを静的に解析し，キャッシュ・メモリに関する操作（キャッシュ操作）をすべてプログラムによって行う方式を提案する．この方式を明示化キャッシュ・システムと呼び，そのプログラムのことをキャッシュ操作プログラムと呼ぶことにする．本方式では，プログラムでキャッシュ操作を行うので，キャッシュ・ミスのような問題が改善される可能性を持っていることを示す．

最近，マルチ・プロセッサ・システムにおけるキャッシュのコヒーレンス（キャッシュの一貫性）を制御することを目的としてProgrammable Cacheが提案されている．[1][2]キャッシュのコヒーレンス制御とは，マルチプロセッサの各キャッシュ・メモリ間や主メモリとキャッシュ・メモリ間のデータが矛盾しないようにすることである．

Programmable Cacheの方法は，次のようである．

（1）共有メモリの各アドレスにキャッシュ制御に関する（キャッシュ・メモリに入れる，入れない等の）情報を付加する．

（2）データフロー解析により得られた情報をもとに，キャッシュを制御する（キャッシュ・メモリの内容を共有メモリに書き込む，キャッシュ・メモリ内容を無効化する等の）命令を生成する．

Programmable Cacheでは，キャッシュのコヒーレンス制御のみを目的としたものであるのに対し，本研究はそれとは全く異なるものであり，命令及びデータ全てのキャッシュ操作を明示化するものである．

また，その他の研究としてキャッシュ操作が明示化された計算機が存在するものとして，参照アドレスの予測を行う方法を示したものがある．[3]参照ウィンドウ（定義：ある時点であるデータに関して，その時点あるいは以前に参照されており，その時点以降，依存関係のある別の文で参照されるような全てのデータの要素の集合）の考え方で予測し，それをプログラム作成のためのデータとしている．このデータを参考にすることで，キャッシュ操作の明示化を効率的に行うことも可能である．

本論文では，命令キャッシュの操作明示化に限って述べる．2章では，従来のキャッシュ・システムの問題点について述べる．3章では，明示化キャッシュ・システムの原理を述べる．また，4章では，3章の原理に基づきプログラム例を示す．5章では，ハードウェアの構成例を示す．最後の6章では，明示化キャッシュ・システムの特徴を明らかにする．

2．従来のキャッシュ・システムの問題点

キャッシュ・メモリの方式には種々の方式が存在するが，どの方式においてもヒット率を常に１にすることは不可能である．実行される各プログラムの性質を考慮せずに，固定的なハードウェアにより動的に行っているからである．

従来のキャッシュ・メモリでは，キャッシュ・ミスを避けることは困難である．キャッシュ・ミスが起こると，それから主メモリからキャッシュ・メモリへのデータ転送を開始する．この転送期間中プロセッサは待ち状態に入るので，それだけプログラムの実行完了が遅くなる．キャッシュ・ミスが頻繁に発生すると，遅延時間のためにパフォーマンスが悪くなり，キャッシュ・メモリをつけた意味がなくなる．したがって，キャッシュ・ミスをなくすことが計算機のパフォーマンスをあげる上で重大なことであった．

3．キャッシュ操作の明示化の原理

従来のキャッシュ・システムでは，プロセッサによる命令（データ）参照が起こり，その後にキャッシュ・ヒットの判定が行われている．つまり，要求駆動処理であることが重要な問題点であった．

明示化キャッシュ・システムでは，通常の処理（演算，転送，分岐等）を行う命令流とキャッシュ操作を行う命令流を持ち，これらの命令は並列に実行される．また，主メモリ，キャッシュ・メモリは，固定されたサイズのブロックに分割されている．分割されたキャッシュ・メモリの最小単位のことをキャッシュ・ブロックと呼ぶ．プログラムもブロックに分けられており，それぞれのブロックはプログラム・ブロック呼ばれる．

図１に明示化キャッシュ・システムで実行されるプログラムのコントロールフローによる表現例を示す．図１において，左側は通常の処理を行う命令流であり，右側はキャッシュ操作を行う命令流である．左側命令流の丸印はプログラムのブロック（複数の命令からなっている）を表し，右側命令流の丸印はキャッシュ操作命令を表している．この図で，矢印はコントロールの流れ，すなわち実行順序に関する従属関係を表し，複数本の矢印が入力されているプログラム・ブロックやキャッシュ操作命令は，その上の命令の実行後に実行可能となることを表す．

図１のプログラムは次のように動作する．まず最初に，１のキャッシュ操作命令がキャッシュ・メモリにデータを格納する．実行が終了することにより２のプログラム・ブロックと３のキャッシュ操作命令が実行可能となり，これらが並列に実行される．２のプログラム・ブロック中の命令と３のキャッシュ操作命令が終了すると，４と５の実行が可能となる．

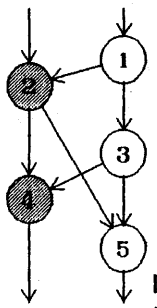


図１ 明示化キャッシュ・システムのプログラム

4．キャッシュ操作プログラム例

フル・アソシアティブ・キャッシュシステムを例にとって明示化キャッシュ・システムを説明する．図２に本プログラムの例を示す．図（ａ）は本システムのプログラムをコントロールフロープログラム風に表したものである．ここで

LMC　＄０１，０１

がキャッシュ操作命令で，主メモリ上の０１番地から始まるプログラム・ブロックをキャッシュ・ブロックの１番にロード（転送）することを表している．矢印の先にあるプログラムが通常の処理を行うプログラムである．同図（ａ）のプログラムは，主メモリ上の０１番地から始まるプログラム・ブロックを主メモリからキャッシュ・メモリに転送し，そのブロック内の命令を実行することを表している．

転送が完了したことは，キャッシュ操作命令が終了した後矢印にそってコントロールフロー処理で言うところのコントロールトークン（以後簡単にトークンという）を送ることによって処理プログラムに知らされる．同図（ｂ）はこのプログラムを実際の命令形式で表したものである．キャッシュ操作命令流から通常の処理命令流への矢印，その逆の矢印は実際には送り側の命令の後と受け側の命令の前につけたタグで表され

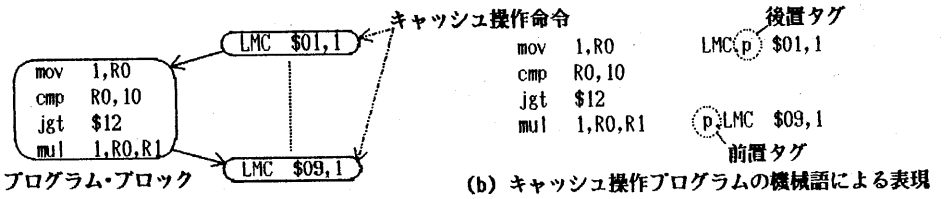
る．前につけたタグを前置タグ，後ろにつけたタグを後置タグと呼んでいる．ただし，処理プログラムの前置タグはその性質上必要としないので省略される．このようなプログラムは，一般には高級言語によって書かれたプログラムを，コンパイラによって解析することにより生成される．

図３は，①各命令の長さが同じで１，②キャッシュ・ブロックの大きさが４としたときの本システムのプログラムを示す．同図（ａ）は，通常の処理プログラムである．また，同図（ｂ）はキャッシュ操作プログラムで，

```
jgt  $08
```

はキャッシュ操作プログラムの条件ジャンプ命令であり，コンディションコードＣＣを処理プログラムから受け取りそのＣＣが"大"を表しているとき０８番地にジャンプすることを表している．ｊｍｐは無条件ジャンプ命令である．

同図（ｃ）は，高級言語による表現であり，このプログラムをコンパイラによって解析することにより（ａ），（ｂ）を得ることができる．



(a) キャッシュ操作プログラムのグラフによる表現

(b) キャッシュ操作プログラムの機械語による表現

前置タグ：プロセッサの許可があるまで，キャッシュ操作命令の実行を開始しない．

後置タグ：命令の実行後，プログラム・ブロックの実行を許可する．

図2 従属関係の表現方法

```
01:mov    1,R0
02:cmp    R0,10
03:jgt    $12
04:mul.i  1,R0,R1
05:mul    2,R0,R2
06:mul    R0,R0,R3
07:add    R1,R2,R4
08:add.i  R3,R4,R5
09:cmp    R5,50
0A:jle    $0E
0B:mul    5,R0,R6
0C:sub.i  R5,R6,R7
0D:jmp    $10
0E:mul    100,R0,R8
0F:add    R5,R8,R9
10:add.i  1,R0,R0
11:jmp    $02
```

プログラム・ブロック

図3(a) 処理プログラム

```
01:  LMC.p  $01,1
02:  jgt    $08
03:  LMC.p  $05,2
04:p.LMC.p  $09,1
05:p.LMC.p  $0D,2
06:p.LMC.p  $11,1
07:  jmp    $01
```

図3(b) キャッシュ操作プログラム

```
for i := 1 to 10 begin
    a := 1 * i;
    b := 2 * i;
    c := i * i;
    d := a + b + c;
    if d > 50 then begin
        e := 5 * i;
        f := d - e;
    end
    else begin
        g := 100 * i;
        h := d + g;
    end
end
```

図3(c) 高級言語によるプログラム

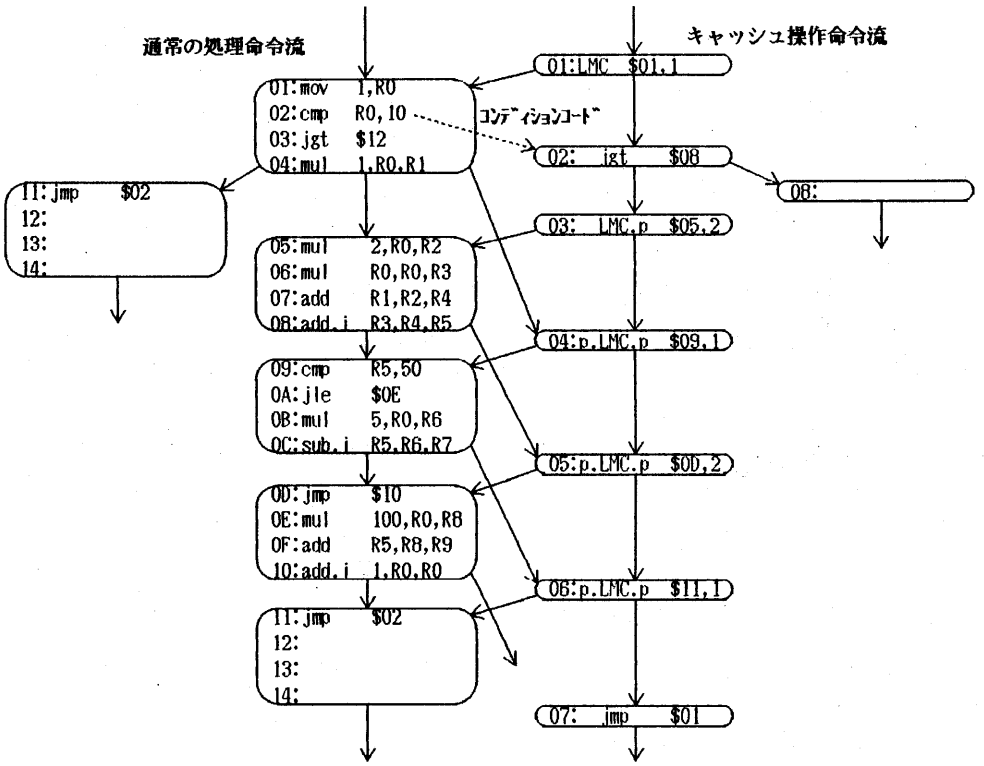


図4　キャッシュ操作プログラムの実行過程

図4はこのプログラムが実行される過程を示す図である．すなわち，このプログラムの実行は，

（1）最初にキャッシュ操作プログラムの０１番地の命令

LMC．p　＄０１，１

が実行される．この実行によって０１番地から始まるプログラム・ブロックがキャッシュ・ブロック１に転送される．この命令には後置タグｐが付いているので，その後処理プログラムを実行しているプロセッサ（処理プロセッサと呼ぶ）へキャッシュ操作命令の実行終了を示すトークンが送出される．

（2）トークンを受け取った処理プロセッサは処理プログラムの０１番地の命令

ｍｏｖ　１，Ｒ０

をキャッシュ・メモリから取り出し実行する．

（3）（2）と並列にキャッシュ操作プログラムの０２番地の命令

ｊｇｔ　＄０８

がキャッシュ操作プロセッサによってフェッチされ解釈される．この命令は，条件分岐命令であるのでコンディションコードＣＣが到着しているかどうかを調べ，到着していればそのまま命令の実行を継続し，到着していない時はその到着を待つ．

（4）（2）の命令の実行が終ると，処理プログラムの０２番地の命令

ｃｍｐ　Ｒ０，１０

が実行される．この命令はレジスタＲ０の内容と１０を比べその結果をＣＣとしてキャッシュ操作プログラムに送る．この比較命令により（3）でフェッチ，解釈した条件分岐命令の条件が確定するので，条件分岐命令の実行サイクルが始まる．

（5）（3）の条件分岐が成立した場合には，キャッシュ操作プロセッサは，キャッシュ操作プログラムの０８番地の命令を実行する．（3）の条件分岐が不成立の場合，キャッシュ操作プロセッサは，キャッシュ操作プログラムの０３番地の命令

LMC．p　＄０５，２

を実行し，実行終了を示すトークンを送出する．

（6）（4）の命令の実行が終ると，処理プログラムの０３番地の命令

ｊｇｔ　＄１２

が実行される．この命令は，（2）の命令と同じ条件で分岐を行う．

（7）（6）の条件分岐が成立した場合には，処理プロセッサは，処理プログラムの１２番地の命令を実行する．条件分岐が不成立の場合，処理プロセッサは，処理プログラム０４番地の命令

ｍｕｌ．ｉ　１，Ｒ０，Ｒ１

を実行する．この命令には後置タグｉが付いているので，キャッシュ操作プロセッサへ，このプログラム・ブロックの実行終了を示すトークンを送出する．

以下，上記のような実行が繰り返される．

５．ハードウェアの構成例

明示化キャッシュ・システムのハードウェアの構成例を図５に示す．プロセッサは，通常の処理（転送処理，演算処理，分岐処理等）を行い，キャッシュ操作プロセッサは，キャッシュ操作を行う．主メモリは，通常の処理を行う命令やデータを格納し，キャッシュ操作プロセッサ用メモリは，キャッシュ操作命令を格納する．

キャッシュ操作プロセッサとプロセッサの間では同期が必要となる．同期は，同期信号線でトークンを送受することにより行っている．トークンは１ビット，または，単なる電気信号の送出でよいので非常に高速に行うことができる．

ハードウェアの基本的な動作は，次のようである．

①プロセッサの動作

（1）プロセッサは，キャッシュ操作プロセッサからのトークンが到着していたら，プログラム・カウンタの値が示す命令をキャッシュ・メモリからフェッチし実行する．

（2）（1）で実行した命令に後置タグがついていない場合には実行を終了する．後置タグがついている場合には，キャッシュ操作プロセッサへトークンを送る．

（3）プログラム・カウンタを更新する．（1）へ行く．

②キャッシュ操作プロセッサの動作

（1）キャッシュ操作プロセッサは，プログラム・カウンタの値が示す命令をキャッシュ操作プロセッサ用メモリからフェッチする．

（2）命令に前置タグが付いていたら，プロセッサからトークンが送られているかどうかをチェックする．送られていたら命令を実行する．送られていなければ，送られるまで待つ．

（3）命令に後置タグが付いていたらプロセッサにトークンを送る．

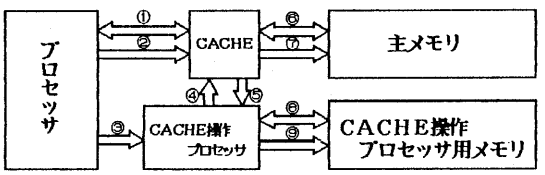
（4）プログラム・カウンタを更新する．（1）へ行く．

６．明示化キャッシュ・システムの特徴

明示化キャッシュ・システムでは，静的な解析によりキャッシュ操作プログラムを生成しているために，従来のキャッシュ・システムと比べて，以下のような優れた特徴を持っている．

［1］静的に解析可能な場合には，ヒット率を１にすることができる．

コンパイラにより静的に解析可能な場合には，キャッシュ・ミスが発生しないように全てのキャッシュ操作命令の実行スケジューリングを行うことができる．したがって，キャッシュ操作プロセッサは，処理プロセッサの必要とする命令をキャッシュ・メモリに前もって転送することができる．そのため，ヒット率を１にすることができる．



①,⑥プロセッサ用のデータバス
②,⑦プロセッサ用のアドレスバス
③同期信号線,分岐情報
④,⑤キャッシュ・メモリ制御用
⑧キャッシュ操作プロセッサ用のデータバス
⑨キャッシュ操作プロセッサ用のアドレスバス

図5 ハードウェアの一構成例

［2］キャッシュ・メモリの最適な置換が行える．

従来のキャッシュ・メモリではハードウェアにより置換を行っているため，固定的なアルゴリズムでしか行うことができない．それに対し，本システムではキャッシュ操作命令により自由に置換を行うことができるため，キャッシュ・メモリを効率よく用いることができる．その例を図６に示す．

同図（ａ）のはループを表すコントロールフロープログラムである．（ａ）の丸印はプログラム・ブロックを示している．キャッシュ・ブロックの数は４であるとする．この場合には，（ａ）のプログラムを実行するとキャッシュ内容の置換が必要となる．同図（ｂ），（ｃ）は，キャッシュ・メモリの状態遷移を表す．キャッシュ・メモリの状態は①②③④⑤の順に変化する．また，図中の＊は，プロセッサにより実行中のプログラム・ブロックを表す．

（ｂ）はフル・アソシアティブ方式で，置換アルゴリズムがＬＲＵ法の場合である．（ｂ）の場合をみると，状態①から状態②への遷移ではＡがＥに置換され，状態②から状態③への遷移ではＢがＡに置換される，というように連続して置換，すなわちキャッシュ・ミスが起こっている．

（ｃ）は明示化キャッシュ・システムの場合である．（ｃ）の場合をみると，状態②から状態③への遷移でＢがＥに置換されるだけであり，最小限の置換しか起こっていないことわかる．プログラム・ブロックＡ，Ｃ，Ｄは入れ換えられてはいない．明示化キャッシュ・システムでは，キャッシュ操作が静的にスケジューリングされているために，このようなことが可能である．

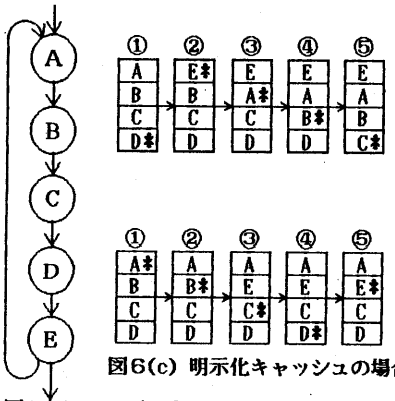


図6(c) 明示化キャッシュの場合

図6(a) ループを表すプログラム

［3］余分なチェックを必要としない．

従来のキャッシュ・システムでは，キャッシュ・メモリが参照される毎にキャッシュ・ヒット，キャッシュ・ミスの判定を行っているが，ヒット率が高い場合には，その判定の大半は無駄となる．従来のキャッシュ・システムでは，必ずキャッシュ・ヒット，キャッシュ・ミスの判定が必要であり，この無駄を取り除くことは不可能である．キャッシュ操作プログラムは，キャッシュ・ミスが起こるかどうかの判断を前もって行うことのできる情報を含んでいる．したがって，明示化キャッシュ・システムは明示化方法，ハードウェアを工夫することにより，この無駄をなくすことのできる可能性を持っていることになる．

［4］キャッシュ・ミス時の遅延時間を低減することができる．

従来のキャッシュ・システムはキャッシュ・ヒット，キャッシュ・ミスの判定終了後でなければ，主メモリからのデータ転送が開始されない．そのため，キャッシュ・ミス時にはプロセッサが待ち状態に入り，処理が止まってしまう．それに対して，本システムは要求駆動処理ではないので，キャッシュ・ミスが起こってからデータ転送が開始されるのではなく，時間的に可能な限り早く転送を開始できる．そのため，キャッシュ・ミス時の遅延時間を低減できる可能性を持っていることになる．

静的に完全に解析不可能な場合にはキャッシュ・ミスが起こるが，次のようなことを考慮することにより，キャッシュ・ミスを減らすことができる．

・分岐条件の確定を早めるような最適化を行う．キャッシュ操作プロセッサが，処理プロセッサと同じように条件分岐を行う場合には，分岐条件の確定をはやめることにより，キャッシュ操作プロセッサは先行してキャッシュ操作命令を実行できる．これにより，条件分岐の条件が成立するときも，不成立の時もキャッシュ・ミスをなくすことができる．

・条件分岐の場合に分岐先もキャッシュ・メモリ内に用意する．条件分岐の場合には実行時でなければ分岐するかどうかは分からない．分岐が成立する場合にキ

ャッシュ・ミスの発生する可能性があるが，分岐先をあらかじめキャッシュ・メモリ内へ転送しておくことにより，キャッシュ・ミスを未然に防ぐことが可能である．また，条件不成立時にキャッシュ・ミスが起こらないようにするために，条件分岐命令をプログラム・ブロックの後ろの方におくことを可能な限り避ける必要がある．

7．おわりに

本論文では，明示化キャッシュ・システムの原理，その実現方法を考察し，その特徴とハードウェアの基本構成を明らかにした．また，本文ではふれなかったが，明示化キャッシュ・システムの簡単な解析により，静的な解析が可能な場合にはヒット率を１にできることが確認されている．静的に解析が不可能な場合には，１０％程度のミスが起こることもわかった．しかし，これは何の改良も加えていない場合の解析結果であり，キャッシュ操作プログラムの最適化を行う，あるいは従来のキャッシュ・メモリに用いられているような技術を利用することにより，キャッシュ・ミスをさらに減らすことが可能であると考えられる．

本研究は３年前から始められており，データ・キャッシュのキャッシュ操作明示化に関する検討や静的に完全に解析不可能な場合の検討は既に行われている．[4]

8．参考文献

（１）Ron Cytron,Steve Karlovsky, and Kevin P. McAuliffe, "Automatic Management of Programmable Caches", Proceedings of the 1988 International Conference in Parallel Processing, (1988)

（２）Hoichi Cheong and Alexander V. Veidenbaunm ,"Stale Data Detection and Coherence Enforcement Using Flow Analysis", Proceedings of the 1988 International Conference in Parallel Processing, (1988)

（３）Gannon, Jalby, and Gallivan, "Strategies for Cache and Local Memory Management by Global Program Transformation",Journal of Parallel and Distributed Computing 5,587-616(1988).

（４）佐藤，"キャッシュ・メモリの操作明示化に関する基礎研究"，昭和 63 年度 名古屋工業大学卒業論文 ,(1989)