オペレーティング・システム 50－5
（１９９１．３．１５）

明示化キャッシュメモリの性能評価

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，計算機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中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静的に解析されており，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．それによってキャッシュ・ミスを可能な限り少なくすることができる可能性を持つ．本稿では，明示化キャッシュの概要とソフトウェアシミュレータによる性能評価の結果について述べる．

Performance Evaluation of a 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 misses are 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 the outline of the cache under the control of program and the results of performance evaluation by software simulation.

1．はじめに

近年，プロセッサはクロック周波数の上昇や命令実行時間の短縮などにより性能向上が図られている．プロセッサの最高性能を引き出すためには，プロセッサへの命令やデータの十分な供給が不可欠の問題である．そのためにキャッシュ・メモリが用いられており，その役割は重要性を増してきている．

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．それは，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キャッシュ操作が行われていることに起因する．また，従来のキャッシュ・メモリ方式では，キャッシュ・ミスが発生した後に，主メモリからキャッシュ・メモリへのデータ転送が開始される．つまり，キャッシュ操作は本質的に要求駆動処理であるということができる．このデータ転送期間中プロセッサは待ち状態に入るとともに，パイプライン処理の流れも乱れるため，それだけプログラムの実行が遅くなる．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向上させる上で重要なことである．

本研究はキャッシュ・ミスを，キャッシュ操作をプログラムの中に埋め込み明示化することによって無くそうとするものである．つまり，従来固定的なアルゴリズムでハードウェアによって行っていたキャッシュ操作を，実行される各プログラムに応じてソフトウェアによって行うのである．この方法をキャッシュ操作の明示化と呼び，そのためのプログラムをキャッシュ操作プログラムと呼んでいる．キャッシュ操作プログラムは，高級言語で書かれたプログラムをコンパイル時に静的に解析することによって作られる．作られたキャッシュ操作プログラムは，演算や転送などの通常の処理と並列に実行される．したがって，従来のキャッシュ・メモリ方式のような要求駆動処理ではないので，キャッシュ・ミスの発生しないようにキャッシュ操作を通常の処理より先行させることも可能である．これによって，明示化キャッシュは，キャッシュ・ミスを極限まで少なくできる可能性を持つ．

本論文では，第２章で明示化データ・キャッシュの原理，ハードウェア構成，特徴について述べる．また，第３章では，ソフトウェア・シミュレータによる性能評価の結果について述べる．

2．明示化データ・キャッシュ

2．1 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やデータの参照が起こった後で，キャッシュにヒットしたかどうかの判定が行われている．それに対し，明示化キャッシュ・システムでは通常の処理を行う命令流とキャッシュ操作を行う命令流を持っている．これらはそれぞれ処理命令流，キャッシュ操作命令流と呼ばれており，またこれらのプログラムは処理プログラム（ＰＰＲ），キャッシュ操作プログラム（ＣＨＰＲ）と呼ばれている．

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・グラフによる表現例を示す．図１において，左側の流れが通常の処理を行う命令（ＰＰＲ）であり，右側がキャッシュ操作を行う命令（ＣＨＰＲ）である．左側の丸印はプログラムのブロック（１命令以上からなっている）を表し，右側の丸印は１キャッシュ操作命令を表している．この図で，矢印（アーク）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，複数本のアーク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．このことは，コントロール・トークンと呼ばれる制御信号がアーク上を流れることによってなされる．

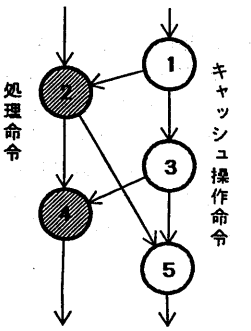

図１ 明示化プログラムのグラフによる表現

図１のプログラムは次のように実行される．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．格納後，実行の終了を示すトークンをアーク上に送出することによって，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・ブロック中の命令すべてと３のキャッシュ操作命令が終了すると，トークンがそれぞれのアーク上に送出され，４のプログラム・ブロックと５のキャッシュ操作命令の実行が可能となる．以上のように処理プログラムとキャッシュ操作プログラムは，お互いに同期をとりながら並列に実行される．

2．2 ハードウェア構成

データ・キャッシュ操作の明示化を行った計算機（明示化キャッシュ計算機）の構造を図２に示す．ＰＵはプロセッサ，ＤＣＨＭはデータ・キャッシュ・メモリ，Ｍは主メモリ，ＣＨＰＵはキャッシュ操作プロセッサ，ＣＨＰＵＭはキャッシュ操作プロセッサ用メモリである．主メモリ（Ｍ）には処理プログラムやデータが格納されており，キャッシュ操作プロセッサ用メモリ（ＣＨＰＵＭ）にはキャッシュ操作プログラム（ＣＨＰＲ）が格納されている．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（ＣＨＰＲ）を実行し，ＰＵは処理プログラム（ＰＰＲ）を実行する．

データ・キャッシュ・メモリ（ＤＣＨＭ）は，ブロックと呼ばれるｍワードからなるメモリｎ個から構成されている．各ブロックにはＤＣＢ１からＤＣＢｎの名前がつけられており，

これをブロックナンバと呼ぶことにする．この1ブロックを構成するmワードがデータ・キャッシュ・メモリと主メモリ間を移動する基本単位であり，この大きさはラインサイズ（あるいはブロックサイズ）と呼ばれている．主メモリ内のデータは，当然のことながらmワード単位で移動させられるので，mワード毎の大きさに分割されている．

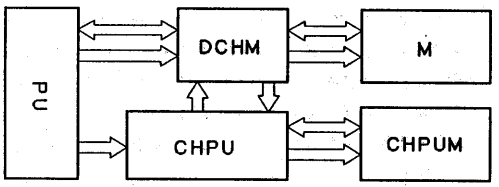

図2　ハードウェアの構成例

2．3 キャッシュ操作命令

次に示す命令が，データ・キャッシュ操作命令である．

（1）主メモリの内容をデータ・キャッシュ・メモリへロードする命令（ロード命令）

LMC　アドレス，キャッシュ・ブロック・ナンバ

（2）データ・キャッシュ・メモリの内容を主メモリへストアする命令（ストア命令）

SCM　キャッシュ・ブロック・ナンバ

（3）分岐命令

JUMP　アドレス　：無条件ジャンプ命令
BRANCH　アドレス　：条件ジャンプ命令

（4）コール命令

CALL　アドレス　：コール命令
RETURN　：リタン命令

2．4　明示化方法の分類

キャッシュ操作命令のうちロード命令とストア命令とを以下の様に分類することで，明示化方法を3種類検討する．

（1）　ロード命令（主メモリ→キャッシュ・メモリ）

ロード命令は，

LMC　OP1，　OP2

のように表現される．OP1は主メモリのアドレスを示し，OP2はキャッシュ・メモリのブロックナンバを示す．この命令によって，OP1の示す主メモリのブロックからOP2の示すキャッシュ・メモリのブロックへデータの転送を行う．

このオペランドOP1，OP2のとり得る値によって，LMC命令を分類すると，次のようになる．

LMC　OP1,	OP2	例
S	S	LMC　100, 1
S	SD	LMC　100
S	D	LMC　100, R1
D	S	LMC　R1, 1
D	SD	LMC　R1
D	D	LMC　R1, R2

この表で，Sはオペランドを静的に決定することを意味し，Dはオペランドを動的に決定することを意味する．また，SDはオペランドを決めるアルゴリズムを静的に決め，そのアルゴリズムによってオペランドを動的に決定することを意味する．表に示した以外の組み合わせも考えられるが，検討する意味はない．また，R1，R2はレジスタを表わし，その値は動的に決定される．

また，本論文でOP2にDを指定する方法についての検討は行っていない．

現在のプログラミング・スタイルでは，S－S型命令（OP1がSで，OP2がSの命令）のみで明示化プログラムを作成できない．したがって，S型命令（OP1がSの命令）とD型命令とを組み合わせて，明示化プログラムを作成しなければならない．S型命令とD型命令の組み合せは，

（S－S，　D－S），（S－S，　D－SD）
（S－SD，D－S），（S－SD，D－SD）

の4通り考えられる．しかし，（S－S，D－SD）と（S－SD，D－S）の組み合せは実現できない．片方の命令のOP2がSであり，もう一方の命令のOP2がSDである場合，Sを決定できないからである（SDは実行時に決定されるので，Sを静的に決定できない）．したがって，実現できる組み合せは，

（S－S，　D－S），（S－SD，D－SD）

の2通りである．

（2）　ストア命令（キャッシュ・メモリ→主メモリ）

ストア命令は，

SCM　OP1，　OP2

のように表現される．OP1はキャッシュ・メモリのブロックナンバーを示し，OP2は主メモリのアドレスを示す．この命令によって，OP1の示すキャッシュ・メモリのブロックからOP2の示す主メモリのブロックへデータの転送を行う．

OP2の指定を行うことには意味がない．なぜなら，OP2は静的に決定されるものであって，動的に決定されたり，一定のアルゴリズムによって決定されるものではないからである．オペランドOP1のとり得る値によって，SCM命令を分類すると，次のようになる．

SCM　OP1	例
S	SCM　100
SD	
D	SCM　R1

この表中の記号の意味は，ＬＭＣ命令の場合と同様である．ＯＰ１がＳＤの場合には，ＳＣＭ命令は省略可能であり，ストアに関してその実行タイミング，実行内容は動的に決定されることになる．

また，本論文でＯＰ１にＤを指定する方法についての検討は行っていない．

（３） 明示化方法

（１），（２）よりＬＭＣ命令とＳＣＭ命令の組み合わせは，

［１］ＬＭＣ命令……（Ｓ－Ｓ，Ｄ－Ｓ）型
　　ＳＣＭ命令……Ｓ型
［２］ＬＭＣ命令……（Ｓ－Ｓ，Ｄ－Ｓ）型
　　ＳＣＭ命令……ＳＤ型
［３］ＬＭＣ命令……（Ｓ－ＳＤ，Ｄ－ＳＤ)型
　　ＳＣＭ命令……ＳＤ型

の３つになる．これらの方法をそれぞれ完全静的決定，半静的決定，動的決定と呼ぶことにする．

（４） キャッシュ操作プログラム例

図３にキャッシュ操作プログラムの例（完全静的決定の場合）を示す．同図（ａ）は，高級言語による表現であり，同図（ｂ）は処理プログラムであり，同図（ｃ）はキャッシュ操作プログラムである．同図（ｄ）はコントロール・フロー・グラフによる表現である．

２．５ 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，ヒット率が理論的に最高となる．

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データをキャッシュ・メモリに前もって転送することができる．そのため，ヒット率を理論的に最高の値にすることができる．

（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４に示す．

同図（ａ）はループを表すプログラムである．（ａ）の丸印はプログラム・ブロックを示している．また，ＡＢＣＤＥは，各プログラム・ブロックで使用されるデータ・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（１）（２）（３）（４）（５）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．この場合をみると，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がわかる．データ・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．

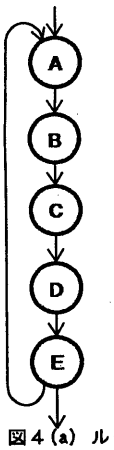

図４(a) ループを示すコントロール・フロー・グラフ

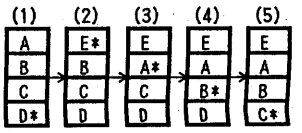

図４(b) フル・アソシアティブ方式，ＬＲＵ置換法の場合

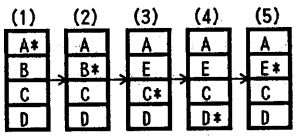

図４(c) 明示化キャッシュの場合

（３）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かった．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った．それに対して，本システムは，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（キャッシュ操作を先行して実行することが可能）．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が大きい．

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．

（４）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．

（５）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．このようなことをプログラムで指定できることが大きな特徴となっている．

（６）レジスタ間接アドレシングを行うレジスタの値の確定を早めるような最適化を行う．

図５(a)に最適化前，(b)に最適化後のプログラムを示す．最適化の基本的な考え方は，

アドレス生成を行っている命令の実行を早めるように，その命令を前方（プログラムリストの上方）へ移動する．

である．図５の場合，同図(a)中のadd命令（３行目）を前方（同図(b)の２行目）へ移動している．現在のところ局所的な命令順序の入れ換えのみを考慮しているため，大域的にみると最適化にならない場合も存在し得る．

```
begin
    ans1 := 0;
    ans2 := 1;
    for i:=1 to 4 do begin
        ans1 := ans1 + a[i];
        ans2 := ans2 * a[i]
    end
end
```

図３(a) 高級言語によるプログラム

```
01:  move     x-1,R1
02:  move     0,R2
03:  move     1,R3
04:  add.d    R1,1,R1
05:d.load     R1,R4
06:  add      R4,R2,R2
07:  mul      R4,R3,R3
08:  noteq    R4,x+3
09:  branch   04
0A:d.store    R2,y
0B:  store.d  R3,y+1
```

図３(b) 処理プログラム

```
01:c.LMC.c    R1,DCB1
02:  branch   01
03:  LMC.c    y,DCB2
04:c.SCM      DCB2
```

図３(c) キャッシュ操作プログラム

```
  ADD.d   DataA, R10, R10
d.LOAD    R10, R11
  ADD.d   Temp, R2, R12
d.STORE   R11, R12
```

図５(a) 最適化前

```
  ADD.d   DataA, R10, R10
  ADD.d   Temp, R2, R12
d.LOAD    R10, R11
d.STORE   R11, R12
```

図５(b) 最適化後

３．性能評価

３．１ ソフトウェア・シミュレータの概要

性能評価を行うために，従来のデータ・キャッシュをもつ逐次計算機用（ｎｃ－１），完全静的決定，半静的決定，動的決定の各場合の明示化キャッシュ計算機用（ｅｃ－１，ｅｃ－２，ｅｃ－３）の４種類のシミュレータを作成した．

これらのシミュレータはＣ言語で記述され，ワークステーションのＸ－Ｗｉｎｄｏｗ上で動作する．

今回作成した４種類のシミュレータでは次のことを仮定している．

・プロセッサの命令は，命令キャッシュより供給され，そのヒット率を１とする．

・命令キャッシュに関するシミュレーションは行わない．

・プロセッサ内はパイプライン方式をとらない．

・キャッシュ・メモリには，ライトバッファが存在し，書き込みに要する時間は０であるとする（ＳＣＭ命令のフェッチ時間は考慮するが，本来の処理（キャッシュ・メモリから主メモリへのデータ転送）を行う時間は０である）．

（１）ｎｃ－１の仕様

・プロセッサは逐次実行を行う．

・キャッシュ・メモリの構成方法として，フル・アソシアティブ方式，セット・アソシアティブ方式，ダイレクト・マップド方式を採用可能．

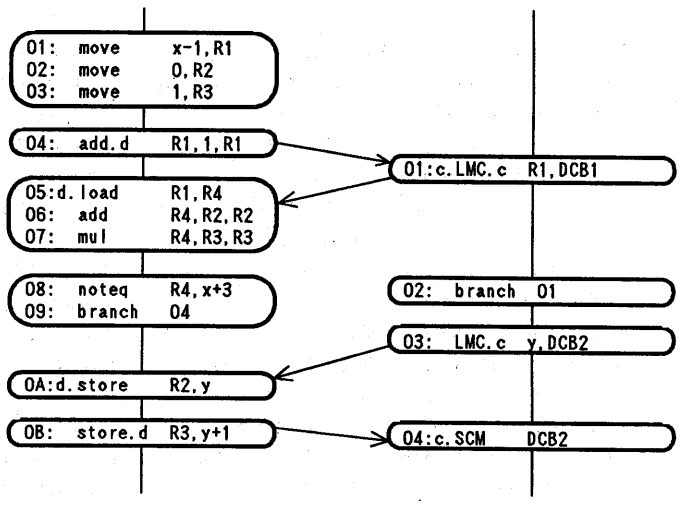

図３(d) コントロール・フロー・グラフによる表現

・キャッシュ・メモリの置換方法として，ＬＲＵ法，ＦＩＦＯ法．ＲＡＮＤＯＭ法を採用可能．
・キャッシュ・メモリの書き込み方式は，コピーバックのみをインプリメント．
（２）ｅｃ－１，ｅｃ－２，ｅｃ－３の仕様
・プロセッサ，キャッシュ操作プロセッサの並列実行．
・命令ストリーム間の同期は機械命令単位で行う．
・同期通信のための時間は０とする．
・ｅｃ－３のみ，ｎｃ－１と同様なキャッシュ・メモリを持つ．

３．２ 評価プログラム

評価に用いたプログラムは，以下の１５種類である．
・ソート７種類．
(1)バブルソート:bubblesort.
(2)分布数えソート:distsort.
(3)ヒープソートによって:heapsort.
(4)クイックソート(非再帰版):qs.
(5)クイックソート(再帰版):quicksort.
(6)シェルソート:shellsort.
(7)単純挿入法ソート:strinssort.
・検索３種類．
(8)配列中のデータを二分検索する:binarysearch.
(9)行列中のデータを検索する:matrixsearch.
(10)配列中のデータを逐次検索する:seqsearch.
・その他５種類．
(11)ハッシュを用いデータをテーブルへ登録する:hash.
(12)行列の乗算（アルゴリズム１）:matmult.
(13)行列の乗算（アルゴリズム２）:matrixmult.
(14)データをマージする:merge.
(15)素数を求める:sieve.

３．３ 評価の前提

明示化データ・キャッシュの評価を行うためにいくつかの条件を設定する．まず最初に，キャッシュ・メモリのブロック数を制限するかどうかによって，
・制限なし：キャッシュ操作を明示化する際に，キャッシュ・メモリのブロックを最大限利用する．
・制限あり：キャッシュ操作を明示化する際に，キャッシュ・メモリのブロックを２に限定する．
の２つの場合について評価を行う．また，２．４で述べたように，明示化方法を３種類検討する．
・完全静的決定：ＬＭＣ命令の第２オペランド，ＳＣＭ命令のオペランドを完全に静的に決定する（ＬＭＣ命令が(Ｓ－Ｓ，Ｄ－Ｓ)型であり，ＳＣＭ命令がＳ型）．
・半静的決定：ＬＭＣ命令の第２オペランドを静的に，ＳＣＭ命令のオペランドを動的に決定する（ＬＭＣ命令が(Ｓ－Ｓ，Ｄ－Ｓ)型であり，ＳＣＭ命令がＳＤ型）．
・動的決定：ＬＭＣ命令の第２オペランド，ＳＣＭ命令のオペランドを動的に決定する（ＬＭＣ命令が(Ｓ－ＳＤ，Ｄ－ＳＤ)型であり，ＳＣＭ命令がＳＤ型）．
である．ラインサイズは２，４ワードの場合について行った．また，最適化を行った場合と最適化を行っていない場合についても行った．最適化については，２．６参照．キャッシュ・メモリのアクセスタイムと主メモリのアクセスタイムの比が，１:１，１:２，１:３，１:４，１:５の各場合について評価を行った．

また，逐次計算機では，明示化キャッシュ計算機の場合と同容量のキャッシュ・メモリを使用する．キャッシュ・メモリの方式はフル・アソシアティブ方式であり，置換法はＬＲＵ法であり，書き込み方式はコピーバック方式である．

３．４ 評価結果

（１）実行時間の比較

表１，２に評価結果の一部を示した．表１はキャッシュ・メモリ・ブロックの数を制限しない場合であり，表２はキャッシュ・メモリ・ブロック数を２に制限した場合である．また，表１，２ともラインサイズは４であり，最適化はなしであり，キャッシュ・メモリと主メモリのアクセスタイムの比は１：５である．表中の数値は，実行時間比を示している．実行時間比は，従来のキャッシュ・メモリを持つ逐次計算機の実行時間に対する明示化キャッシュ計算機の実行時間の比である．実行時間比が小さいほど，明示化キャッシュ計算機の性能が高いといえる．

［ブロック数の制限をしない場合］表１参照．
・完全静的決定の場合，明示化キャッシュ計算機の性能の高い例が７ある．その中でもシェルソートは１５％程度実行時間が短くなっている．逆にクイックソート（非再帰版）では，実行時間が２倍程度になっている．これは，書き込みをどのタイミングで行うか（実行タイミング），どのキャッシュブロックを主メモリに書き込むか（実行内容）をＳＣＭ命令によって静的に決定していることに起因する．つまり，静的解析では書き込みの実行タイミングと実行内容を特定できない場合が存在するからである．クイックソート（非再帰版）の場合には，多くの書き込みが存在するため，（静的解析で特定できない場合が多く存在するため）ＳＣＭ命令も多く存在する．そのためＳＣＭ命令のフェッチ時間がネックとなっている．また，ラインサイズが２から４へ変化することによって，実行時間比の大きくなったプログラムは７例であり，その他の例では変化ないか，小さくなった．ラインサイズの変化によって，実行時間比がどのように変化するか一概には決められない．

・半静的決定の場合，明示化キャッシュ計算機の性能の高い例が１１ある．完全静的決定の場合と同様に，シェルソートは１５％以上実行時間が短くなっている．残りの４例については，１０％程度実行時間が長くなっている．これは，キャ

ッシュ・ブロック・ナンバをＬＭＣ命令によって静的に決定していることに起因する．つまり，キャッシュ・ブロック・ナンバを静的に決定することによって，キャッシュ・メモリ全体を有効に使えないからである．また，ラインサイズが２から４へ変化することによって，実行時間比の大きくなったプログラムは１例のみであり，その他の例では変化ないか，小さくなった．ラインサイズの変化によって，実行時間比は小さくなる傾向にあるといえる．

・動的決定の場合，明示化キャッシュ計算機の性能の高い例が９ある．完全静的決定の場合と同様に，シェルソートは１５％以上実行時間が短くなっている．残りの６例については，従来のキャッシュ・メモリと変わっていない．半静的決定の場合に比べ，実行時間が長くなる例がいくつかある．その理由としては，メモリのアクセスパターンによって，従来のキャッシュ・メモリの問題点（無駄な置換がおこる）が発生したことによると思われる．また，ラインサイズが２から４へ変化することによって，実行時間比の大きくなったプログラムはなかった．したがって，ラインサイズの変化によって，実行時間比は小さくなるといえる．

３つの場合を比較すると実行時間比は，

動的決定＜半静的決定＜完全静的決定

の順に小さくなっている．半静的決定の場合と動的決定の場合との間の大小関係は一概には決まらないが，全体的にみると動的決定の場合の方が優れている．

［ブロック数を２に制限した場合］表２参照．

この場合は，ブロック数を制限しない場合とほぼ同様なことがいえ，実行時間比は，

動的決定＜半静的決定＜完全静的決定

の順に小さくなっている．

program	完全静的	半静的	動的
binarysearch	1.05	1.05	0.99
bubblesort	0.94	0.88	0.82
distsort	0.94	0.85	0.92
hash	1.00	0.95	0.97
heapsort	1.03	0.93	0.90
matmult	0.99	0.97	1.00
matrixmult	0.99	0.98	1.00
matrixsearch	0.99	0.99	0.99
merge	0.94	0.83	0.88
qs	2.24	1.10	1.00
quicksort	1.23	0.95	0.93
seqsearch	1.00	1.00	1.00
shellsort	0.83	0.80	0.80
sieve	1.29	0.99	1.00
strinssort	1.32	1.08	1.00

表１　実行時間比
（ブロック数の制限なし，ラインサイズ：４ワード，
最適化なし，アクセスタイムの比 ＝ １：５）

program	完全静的	半静的	動的
binarysearch	1.05	1.05	0.99
bubblesort	0.94	0.88	0.82
distsort	0.99	0.92	0.90
hash	0.97	0.95	0.96
heapsort	0.93	0.92	0.90
matmult	0.95	0.94	1.00
matrixmult	0.95	0.95	1.00
matrixsearch	0.99	0.99	0.99
merge	0.88	0.88	0.88
qs	1.27	1.09	1.00
quicksort	1.00	0.88	0.92
seqsearch	1.00	1.00	1.00
shellsort	0.76	0.73	0.80
sieve	1.09	0.99	1.00
strinssort	1.27	1.09	1.00

表２　実行時間比
（ブロック数の制限あり，ラインサイズ：４ワード，
最適化なし，アクセスタイムの比 ＝ １：５）

（２）最適化の効果

表３，４に評価結果の一部を示した．表３はキャッシュ・メモリ・ブロックの数を制限しない場合であり，表４はキャッシュ・メモリ・ブロック数を２に制限した場合である．また，表３，４ともラインサイズは４であり，キャッシュ・メモリと主メモリのアクセスタイムの比は１：５である．表中の数値は，最適化実行時間比を示している．最適化実行時間比は，最適化前の実行時間に対する最適化後の実行時間の比である．最適化実行時間比が小さいほど，最適化の効果が大きいといえる．

［ブロック数の制限をしない場合］表３参照．

・完全静的決定，半静的決定，動的決定の各場合について，最適化実行時間比を平均するとそれぞれ０．９７，０．９７，０．９５となる．どの場合も，数％～１０％程度最適化の効果の出ていることがわかる．しかし，半静的決定の場合，最適化実行時間比が２例で１をこえている．これは，最適化を局所的に行っていることに起因すると思われる．つまり，最適化する際に大域的な考慮を行っていないので，最適化を行ったことによって関係のない部分に悪影響を与えたものと考えられる．

［ブロック数を２に制限した場合］表４参照．

・完全静的決定，半静的決定，動的決定の各場合について，最適化実行時間比を平均するとそれぞれ０．９６，０．９６，０．９５となる．ブロック数を制限しない場合と同様に，数％～１０％程度最適化の効果の出ていることがわかる．

４．おわりに

本論文では，明示化データ・キャッシュの概要を示し，従来のデータ・キャッシュ・メモリを持つ計算機用，明示化データ・キャッシュ・メモリを持つ計算機用のシミュレータを作成し，評価を行った．

主メモリからキャッシュ・メモリへデータの転送を行う命令（ロード命令），キャッシュ・メモリから主メモリへデータの転送を行う命令（ストア命令）の実現方法を考慮し，明示化方法を３種類検討した．各方法の異なる点は，ロード命令やストア命令のオペランドの指定方法である．オペランドの値を“静的に決定する”か“動的に決定する”かによって，それらの方法は区別される．１番目の方法（完全静的決定）はロード命令，ストア命令ともオペランドを静的に決定し，２番目の方法（半静的決定）はロード命令のオペランドを静的に，ストア命令のオペランドを動的に決定し，３番目の方法（動的決定）はロード命令，ストア命令ともオペランドを動的に決定する．評価の結果，どの方法によって明示化を行っても，明示化データ・キャッシュはプログラムの実行時間を数％～１５％程度短縮できる可能性を持つことがわかった．各方法の優位性について，一概に言うことはできないが，完全静的決定の場合より半静的決定，動的決定の場合の方が優れていることは，明らかになった．また，プログラムを最適化することによって，実行時間がさらに数％程度短縮されることも明らかになった．

program	完全静的	半静的	動的
binarysearch	1.00	1.00	1.00
bubblesort	0.94	0.94	0.93
distsort	0.94	0.94	0.94
hash	1.00	0.98	0.99
heapsort	0.99	1.09	0.99
matmult	0.92	0.92	0.92
matrixmult	0.90	0.90	0.89
matrixsearch	1.00	1.00	1.00
merge	0.98	0.96	0.96
qs	1.00	0.91	0.91
quicksort	0.96	1.03	0.95
seqsearch	1.00	1.00	1.00
shellsort	1.00	1.00	1.00
sieve	0.94	0.93	0.93
strinssort	1.00	0.92	0.91

表３　最適化実行時間比
（ブロック数の制限なし，ラインサイズ：４ワード，アクセスタイムの比 ＝ １：５）

program	完全静的	半静的	動的
binarysearch	1.00	1.00	1.00
bubblesort	0.94	0.94	0.93
distsort	0.95	0.94	0.94
hash	1.00	0.99	0.99
heapsort	0.99	1.00	0.99
matmult	0.92	0.92	0.92
matrixmult	0.90	0.90	0.86
matrixsearch	1.00	1.00	1.00
merge	0.97	0.97	0.97
qs	0.93	0.92	0.91
quicksort	0.96	0.95	0.95
seqsearch	1.00	1.00	1.00
shellsort	1.00	1.00	1.00
sieve	0.94	0.92	0.93
strinssort	0.93	0.92	0.91

表４　最適化実行時間比
（ブロック数の制限あり，ラインサイズ：４ワード，アクセスタイムの比 ＝ １：５）

参考文献

［１］佐藤，有田，曽和，"キャッシュ操作明示化の提案"，情報処理学会研究報告，90-ARC-80 (1990.1).

［２］佐藤，曽和，"並列処理によるキャッシュ操作の明示化"，並列処理シンポジウムJSPP'90，pp.1-7 (1990.5).

［３］佐藤，曽和，"明示化キャッシュ・システムの性能評価"，電気関係学会東海支部連合大会　講演論文集，pp.675 (1990.10).

［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).

［５］佐藤，"キャッシュメモリの操作明示化に関する研究"，平成 元 年度 名古屋工業大学修士論文，(1991).