資　　料

バッファ・メモリ方式のシミュレーション*

中村奉夫**　北川一***
金沢正憲***　萩原宏**

Abstract

In recent years buffer memory has become more popular in order to attain high CPU performance in large computer systems. In the analysis of the effectiveness of buffer memory in multiprogramming environments, it is important to take the influence of task-switching into consideration as well as design parameters of buffer memory.

We have made to evaluate the effectiveness of buffer memory by means of simulation In this simulation a multiprogramming model considering task-switch control is proposed and the patterns of memory reference taken by tracing are used as input data. This paper presents the method and results of the simulation in the analysis of buffer memory.

1. まえがき

計算機システムの大型化に伴って，高速で大容量の記憶装置が要求されているが，価格，技術上の制限から限界がある．このため，小容量で高速な記憶装置と，大容量で比較的低速な記憶装置を階層的に組み合わせて使用し，相互に機能を補い合う方法が採られるが，バッファ・メモリは，この考え方に基づいた高速小容量の記憶装置であり，処理装置 (CPU) とメイン・メモリの動作速度のギャップを埋めるために使用されている．すなわち，CPU からの命令やデータの読出しをできるだけこの高速のバッファ・メモリ上で行なうことにより，メイン・メモリのアクセス頻度を減らして，CPU の高速化を図っている．その場合，必要な命令やデータは，メイン・メモリよりバッファ・メモリへブロック単位で転送される．その制御はハードウェアで行なわれ，ソフトウェア上はブロックの転送を意識せずに使用できる．ところで，このようなバッファ・メモリ方式の効果を考える場合，メモリの容量，ブロックの大きさ，割当て方法，ブロック転送時間などのパラメータの他に，多重プログラム処理におけるタスク・スイッチなどによる影響が重要となる．しかし多重プログラム処理を考慮したバッファ・メモリ方式の効率解析はほとんど行なわれていない．そこで，われわれは，プログラムの走行中における CPU からの命令やデータのメモリ参照がどのように行なわれるかを測定するアドレス・トレーサを作成し，そのトレーサによって収録されたいくつかのアドレス・パターンのトレース結果を入力データとして，多重プログラム処理の環境で動作するバッファ・メモリ方式の計算機システムにおけるメモリ参照のシミュレーションを行ない，種々のパラメータとバッファ・メモリの効率の関係について解析した．

2. アドレス・トレース

2.1 トレーサによるデータ収集

プログラムの動的な振舞いを観測し，バッファ・メモリ方式のシミュレーションへの入力データとするために，プログラムの動作中に生ずる，CPU からの，そのプログラム領域内のメモリ参照を記録するアドレス・トレーサを FACOM 230-60 用に作成した．モニタ部分の動作をソフトウェア的にトレースすることは，実行速度が非常に遅くなって実際のシステムの動

* Simulation of a computer system with buffer memory, by Tomoo Nakamura, Hiroshi Hagiwara (Faculty of Engineering, Kyoto University), Hajime Kitagawa and Masanori Kanazawa (Data Processing Center, Kyoto University).

** 京都大学工学部情報工学教室
*** 京都大学大型計算機センター

Table 1　トレース・データの種類

トレース・データ	コンパイル時/実行時	使用コンパイラ・タイプ	プログラムの大きさ（キロ語）	メモリ参照回数（f）（$\times 10^4$）	P_R(%)	P_W(%)	P_X(%)	記　号
一階常微分方程式 ルンゲ・クッタ・ギル法	コンパイル	FORTRAN-Cタイプ（目的プログラムの最適化なし）	*38	22	20.0	13.0	67.0	a 1
	実　行		9	47	17.1	14.8	68.1	a 2
10次対称行列 固有値問題 ヤコビ法	コンパイル		*38	83	21.8	11.2	67.0	b 1
	実　行		14	79	31.0	10.5	58.5	b 2
	コンパイル	FORTRAN-Dタイプ（目的プログラムの最適化あり）	*38	177	21.7	11.3	67.0	c 1
	実　行		14	39	25.8	11.6	62.6	c 2
ソーティング（1,000 個の数）	コンパイル		*38	38	21.0	12.8	66.2	d 1
	実　行		9	**50	22.3	18.8	58.9	d 2
バッファ・メモリ・シミュレータ	実　行		16	**50	26.7	13.8	59.5	e 2

* FORTRAN コンパイラはセグメント構造になっており，ここでは，コンパイル時の平均コア使用量を示している．
** 実行時間が長く，また，よく似た動作をくりかえすことから，途中でうちきったことを示す．

作と相違が生ずるなどの点で困難があったので，ここでは処理プログラム・モードの動作のトレースに限った．さらに，システムの動作状況の影響を受けない独立な測定データを得るため，個々のユーザ・プログラムごとにトレースする方法を採用した．すなわち，トレースの対象となるプログラムを相対形式の状態で，アドレス・トレーサと結合編集し，実行形式にして実行させれば*，つぎのような情報がその発生順序に磁気テープ上に記録される．

（1）　メモリ参照が生じた命令のアドレスとその命令．

（2）　メモリ参照の種別，すなわち，データの読出し，書込み，命令の読出しの区別．(命令の読出しは，ジャンプまたはスキップ命令が実行され，プログラム実行のシーケンスに変更があったとき記録する．)

（3）　参照されたメモリのアドレス．(間接アドレス指定の場合は，中間的に参照したすべてのアドレスも記録する．)

（4）　モニタ・モードへ入る TRAP 命令 (SVC) の場合は，呼ばれたモニタ・モジュールの種別．

1回のメモリ参照あるいはシーケンスの変更に対して，上の（1）から（4）のデータが4語 (16 バイト) 単位で，磁気テープに出力される．

いくつかの FORTRAN プログラムのコンパイル時および実行時について，トレース・データを収録した**が，その種類を **Table 1** に示す．なお，f, P_R, P_W, P_X はつぎのような値である．($P_R+P_B+P_X=1$)

f：CPU からのメモリ参照回数，
P_R：データ読出し回数/f，
P_W：データ書込み回数/f，
P_X：命令読出し回数/f．

2.2　トレース・データによるプログラムの動作特性

アドレス・トレース・データをもとにして，つぎのような項目についてプログラムの動作特性を知ることができる．

（i）　プログラムはどの程度シーケンシャルに実行されるか．

$S([m,n])$：シーケンシャルに実行されたステップ数 s が $m \leqq s \leqq n$ である割合．

Fig. 2.1 に，$[m,n]$ として，[1,5]，[6,10]，[11,

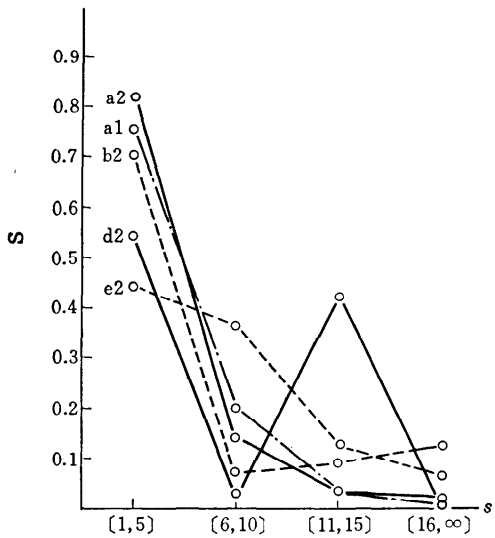


Fig. 2.1 Program dynamic behavior: sequential run length S.

* このようなトレースを行なえば，プログラムの実行速度はトレースしない場合の約50～80倍遅くなる．なお，アドレス・トレーサは，アセンブラ言語で書かれ，プログラムの大きさは約700語である．
** 京大大型計算機センターで処理されているジョブの大部分(90% 以上) は FORTRAN 言語が使用されているので，ここでは特に他の言語によるプログラムの処理中のトレースは行なわなかった．

15〕，〔16, ∞〕をとった場合の，トレース・データの一部に対するSの例を示す．

（ii） プログラムが，ジャンプまたはスキップ命令によりシーケンスの変更を生じたとき，何番地離れた所へ飛ぶか．

B(〔m, n〕)：シーケンスの変更が生じた条件のもとで，b 番地離れた所へ飛んだとき，$m \leqq b < n$ である相対的割合．($b \leqq -1, b \geqq 2$)

Fig. 2.2 に，〔m, n〕を 10 の累乗を単位として，トレース・データの一部に対するBの例を示す．*

A0; 〔$-\infty, -10^4$〕　A5; 〔2, 10〕
A1; 〔$-10^4, -10^3$〕　A6; 〔$10, 10^2$〕
A2; 〔$-10^3, -10^2$〕　A7; 〔$10^2, 10^3$〕
A3; 〔$-10^2, -10$〕　A8; 〔$10^3, 10^4$〕
A4; 〔$-10, 0$〕　A9; 〔$10^4, \infty$〕

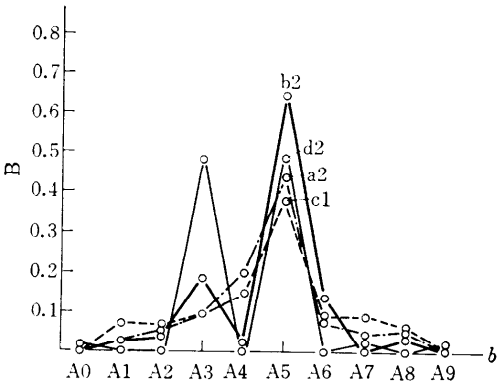


Fig. 2.2 Program dynamic behavior: jump length B.

（iii） プログラムはどの程度ローカル性をもつか．

R(j; m)：プログラムを m 語のブロックに分割したとして，あるメモリ参照時の参照ブロックが，最近使用された (most recently used) j 種類に含まれている割合．

Fig. 2.3 に，2つの例をあげる．

（iv） プログラムが動作するのに，どの程度の領域を必要とするか．

D(r; m)：プログラムを m 語のブロックに分割したとして，連続する r 回のメモリ参照中に必要とされたプログラム領域の大きさ（語数）の平均．

Fig. 2.4 に，r が 15,000 までの場合について，トレース・データの一部に対する D の例を示す．

このようにプログラムの動作特性を表わしたとき，SやBは，プログラムの内容や作り方によって相当異なるのに対して，特にRは分布の形が各プログラムと

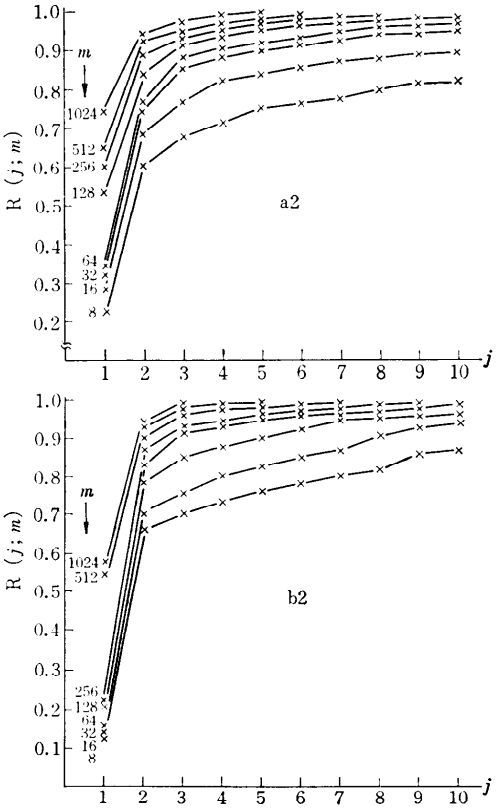


Fig. 2.3 Program dynamic behavior: memory reference R.

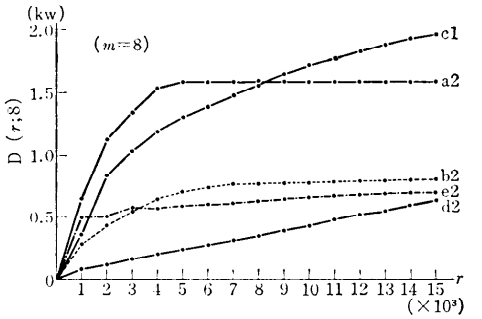


Fig. 2.4 Program dynamic behavior: memory demand D.

もによく似ており，プログラムとしての一般的な特性を示していると考えてよい．このことが，バッファ・メモリ方式やページング方式においてプログラムの違いによる影響を小さくし，その効果を安定にしていると考えられる．

3. バッファ・メモリ・シミュレータ

シミュレーションにおけるバッファ・メモリの構造

* 各プログラムにおいて，シーケンスの変更が生ずる割合は，a2: 28.8%，b2: 32.6%，d2: 14.2%，e2: 14.2% である．

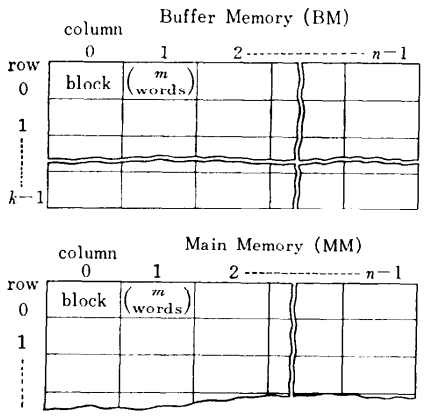


Fig. 3.1 Buffer Memory (BM) and Main Memory (MM).

は，セット・アソシアティブ・マッピング方式とする (**Fig. 3.1** 参照)．この方式では，バッファ・メモリ (以下，BM と省略する) とメイン・メモリ (以下，MM と省略する) が連続した m 語からなるブロックに分割されている．そして BM, MM の何個かのブロックはセットにして，カラムと名づけられる．マッピングは同じ番号のカラムごとに行なわれ，その対応づけに必要なアドレス表はハードウェア的に用意されているものとする．メモリ・アドレスが指定されると，アドレス表を通して BM 内のアドレスに変換される．もし必要とするアドレスが BM 上にない場合は，データの読出し時ならば，MM より CPU へデータが送られるとともに MM より BM へそのアドレスを含むブロックが転送され，その間 CPU は待たねばならない．このときのブロックの入れ換えアルゴリズムは，カラムごとに LRU (最後に使用されてから最も時間の経過したブロックを追い出す) を用いる．データの書込みの場合には，書込むデータのアドレスが BM 上になければ MM へ書込み，アドレスが BM 上にあるときにはつぎの 2 種類の書込み方式を考え，両方式を同時にシミュレートできるようにした．

(A) BM と MM の両方を更新する．(through-storing)

(B) BM のみを更新し，後に，更新されたブロックを BM 上から消すときに，MM へ転送して更新する．(post-storing)

このシミュレーションでは，1 CPU の計算機システムにおける多重プログラム処理を想定し，モニタ・タスクの走行や入出力動作によって生ずるタスク・スイッチが BM の効率に与える影響をつぎのように考慮した．すなわち，タスク・スイッチはモニタ・コール (SVC) および割込みによって引き起こされるが，前者については，トレース・データに含まれている TRAP 命令 (主として入出力動作のため) を利用する．また後者の割込みによるタスク・スイッチの発生は，システムの動作状況に依存していて，個々のトレース・データには含まれないので，平均発生間隔 (メモリ参照回数 f_t) をシミュレーションのパラメータとして与えた．そして，つぎのような 2 つの方法によりタスク・スイッチをシミュレートした．

方法 1 (BM クリア方式)

走行している複数個のプログラムのうちの 1 個に注目しているとして，そのプログラムで (i) CPU が f_t 回メモリ参照するごとに，および (ii) TRAP 命令を出すごとに，タスク・スイッチが行なわれると仮定し，その時点で，BM 上のすべてのデータをクリアする．これは，タスク・スイッチが生ずるごとに，BM 内のすべてのデータが他のプログラムによって消されるという最悪の状況を考えている．

方法 2 (プログラム・スイッチ方式)

複数個のプログラムの走行を考え，各プログラムで (i) f_t 回のメモリ参照ごとに，および (ii) TRAP 命令を出すごとに，プログラム・スイッチして順々に多重処理されてゆくものとする．このとき BM 上のデータはスイッチごとにクリアせず，スイッチされた新たなプログラムの走行によって更新されてゆくが，同一のプログラムに再びスイッチされたときには，以前に使用していた BM 上のブロックのうち，他のプログラムが使用しなかった部分はそのまま残存するとしている．この方法は，モニタ・タスクの走行を考慮しない比較的良い条件の場合を考えている．

さらに，このシミュレーションでは，CPU の先行制御が行なわれていないと仮定している．すなわち，命令の先取りによって生ずる MM から BM へのブロック転送や，書込み命令実行時における MM への書込みと次命令処理の並行動作などは考慮していない．

シミュレータのパラメータとして，ブロックの大きさ (m 語)，カラム数 (n)，カラムの大きさ (k ブロック)，BM の容量 ($C=mnk$ 語)，平均タスク・スイッチ間隔 (メモリ参照回数 f_t) をとる．これらのパラメータのうち，m, n, k, C は，BM の構造を決めるパラメータであり，f_t はオペレーティング・システ

ムの動作状況によって変化するパラメータである．そして，アドレス・トレース結果を入力データとして，種々のパラメータの組に対し，シミュレーションを行ない，メモリ参照に関するつぎのような値を求めた．

P_B：BM 上でアクセス可能であった回数/f，

P_{BR}：BM 上でデータを読出した回数/f，

P_{BW}：BM 上にデータを書込んだ回数/f，

P_{BX}：BM 上から命令を読出した回数/f，

P_{PS}：BM 上で更新されたブロックが BM 上から追い出された回数/f，

P_{BL}：BM 上へブロックの転送された回数/f，

P_F：命令およびデータの読出し回数/f．

ただし，f は CPU からのメモリ参照回数であり，$P_B=PB_{BR}+P_{BW}+P_{BX}$，$P_F=1-P_W$ である．

4. バッファ・メモリ方式の効率について

4.1 シミュレーションの結果

バッファ・メモリ方式の効率を評価するために，つぎのような値を考えた．これらはバッファ・メモリ方式の計算機システムにおけるパフォーマンスを考える上での重要な尺度となるであろう．

(i) バッファ・メモリのヒット率

BM 上での参照が可能である割合．P_B で与えられる．

(ii) バッファ・メモリの実効ヒット率 P_T，P_P

BM のアクセス時間の速さで参照が可能である割合．

through-storing 方式のとき；

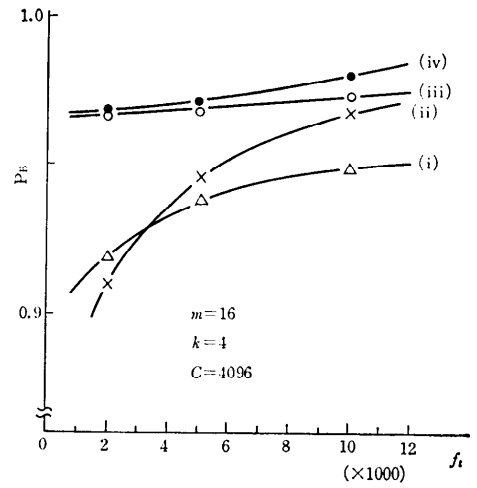
$P_T=P_B-P_{BW}$：書込み時の BM の効果が失われる．

post-storing 方式のとき；

$P_P=P_B-P_{PS}$：BM，MM 間の1ブロックの転送時間が両方向とも同じであるとすると，時間的には，BM 上にないブロックの MM からの読出しが P_{PS} だけ多くあったと考えられる．ただし，シミュレーション結果では，ほとんどすべての場合に P_{PS} は 1% 以下であり，$P_P \fallingdotseq P_B$ とみてさしつかえなかった．

(iii) 実効サイクル時間比 α

MM/BM サイクル時間比を β，1ブロック転送に要する MM サイクル数を d とすると，実効サイクル時間比 α（CPU からのメモリ参照1回当りの平均 BM サイクル数）は，つぎのように求まる．



(i) 方法1による a1
(ii) 方法1による a2
(iii) 方法2による2つのプログラム（a1, a2）の多重処理
(iv) 方法2による3つのプログラム（a1, a2, c2）の多重処理

Fig. 4.1 Effect of mean task-switch interval upon BM effectiveness.

through-storing 方式のとき；

$\alpha_T=P_T+\beta(P_W+d\cdot P_{BL})$.

post-storing 方式のとき；

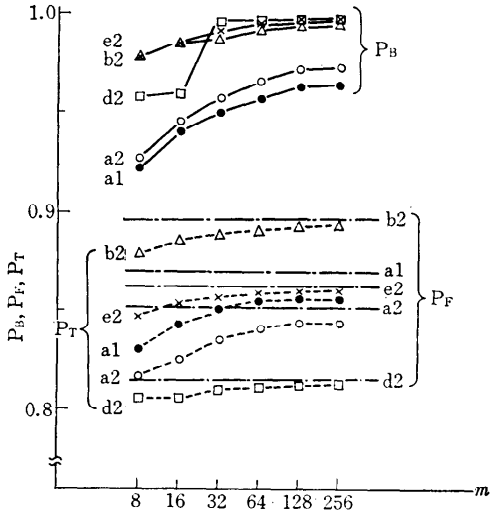
$\alpha_P=P_B+\beta\{(P_W-P_{BW})+d(P_{BL}+P_{PS})\}$.

シミュレーションの結果として，タスク・スイッチ間の平均メモリ参照回数 f_t と効率の関係をみるために方法**1**，方法**2**の効率を比較した例を **Fig. 4.1** に示す．つぎに f_t が m，C，k に与える影響をみるために方法**1**により，f_t が一定の場合の，m の変化（**Fig. 4.2**），C の変化（**Fig. 4.3**），k の変化（**Fig. 4.4**）に対する効率について示す．一方，C（同時に k）と効率の関係を，方法**1**，方法**2**の比較により示した例が **Fig. 4.5** である．また，**Fig. 4.6** には，方法**1**により，各トレース・データの違いによる効率の変化を示す．最後に，MM，BM 間のデータ転送幅（m/d）を一定して，m と α の関係を求めた例が **Fig. 4.7** である．

4.2 タスク・スイッチによる影響

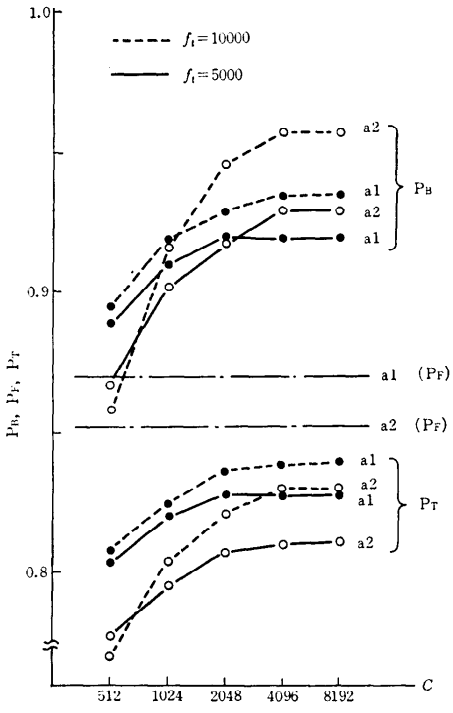
シミュレーションの結果より，タスク・スイッチの影響に関して，つぎのことが明らかになった．

(1) タスク・スイッチ間の平均メモリ参照回数 f_t を変化させたときの効率 P_B の変化の一例が **Fig. 4.1** に示されている．この例からも明らかなように，方



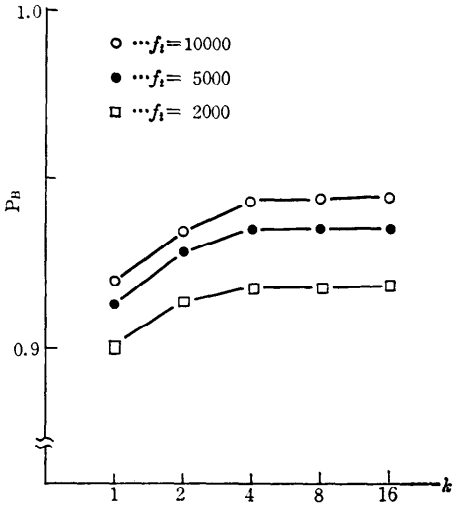
$C=4096$　$k=4$　$n=1024/m$
$f_t=5000$　方法1

Fig. 4.2 Variation of BM effectiveness with block size.



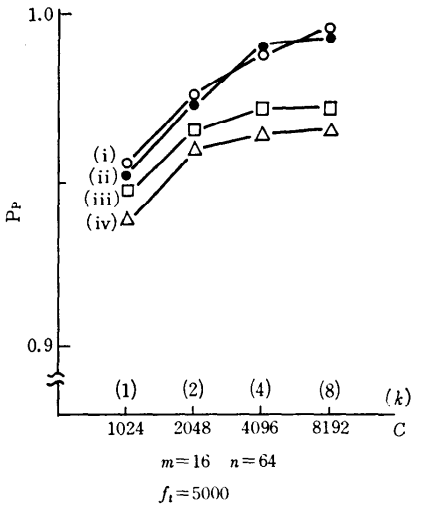
$m=8$　$k=4$　方法1

Fig. 4.3 Variation of BM effectivenss with capacity.



$C=2048$　$m=16$
$n=128/k$　a1　方法1

Fig. 4.4 Variation of BM effectiveness with column size.



$m=16$　$n=64$
$f_t=5000$

(i) 方法2による3つのプログラム（a2, b2, e2）の多重処理
(ii) 方法2による2つのプログラム（a2, b2）の多重処理
(iii) 方法1による a2, b2, e2の平均
(iv) 方法1による a2, b2の平均

Fig. 4.5 Variation of BM effectiveness with capacity and column size.

法**1**による場合（タスク・スイッチごとに BM 上のデータが完全に消える）は，f_t が小さいときに P_B が急激な減少を示すのに対し，方法**2**による場合（タ

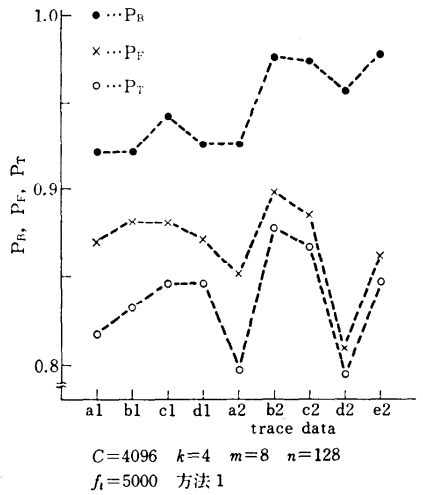


Fig. 4.6 Variation of BM effectiveness with addressing pattern.

スク・スイッチの間隔が小さいと BM 上の消されるデータの量も少ない）は，f_t が小さくなっても P_B の変化は少ない．この例での方法 **2** は，プログラム処理の多重度を 2，3 としているが，多重度をさらに大きくすれば，同じタスクに制御が戻ってくるまでに BM 内のデータが，完全に消えている確率も大きくなり，方法 **2** は方法 **1** と同じような傾向を示すこととなる．方法 **1** の場合においても，f_t が数千†より大きくなれば，タスク・スイッチが BM の効率に与える影響は小さいと考えられる．これは，一つには，プログラムが走行するために必要なブロックのほとんどが，最初の数千回のメモリ参照によって BM 上におかれるからと考えられる（**Fig. 2.4** 参照）．

（2） **Fig. 4.2** に示されているように，f_t=5,000 のとき，容量 C を一定にしてブロックの大きさ m を大きくすれば，P_B は増大する．これは，タスク・スイッチごとに BM がクリアされるため（方法 **1**），その間隔（この場合 f_t は 5,000）での，プログラムにおけるメモリ参照のローカル性に起因していると考えられる．図示されていないが，タスク・スイッチを考慮しない場合（f_t が ∞）には，C が 4 キロ語のとき，m を変化させても P_B はほとんど一定であり，さらに C がそれ以下のとき，m の増大に対して P_B が減少する傾向がみられた．ところが，実効サイクル時間比 α は，P_B が増大しても必ずしも減少しない．

† 現在，実際に使用されている OS では，タスク・スイッチ間の平均メモリ参照回数が，この程度の値になっている場合が多い．

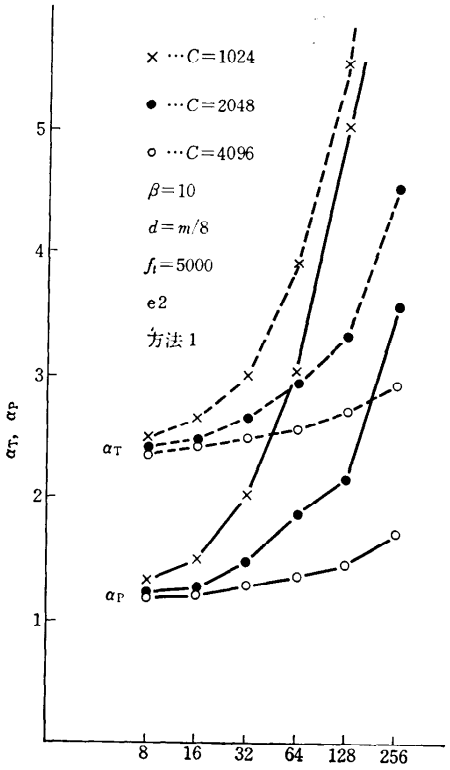


Fig. 4.7 Effective cycle time ratio vs. block size.

Fig. 4.7 に示されているように，データ転送幅が小さい値に一定のときには，ブロックの大きさ m が増大すると α が大きくなる．これによれば，タスク・スイッチを考慮しない場合と同様に，小さい m (8, 16 など) が好ましいと考えられる．

（3） タスク・スイッチを考慮する，しないにかかわらず，容量 C が大きくなれば P_B が増大することは明らかである．タスク・スイッチを考慮した場合，方法 **1** では，f_t が 1 万でも，4 キロ語程度で P_B の増加はとまる（**Fig. 4.3**）．これは，方法 **1** において BM がクリアされるまでに，BM へ転送されるデータの量が，BM の容量より小さいことによる（**Fig. 2.4** 参照）．4，8 キロ語などの大容量 BM の場合，実際の多重プログラム処理では，タスク・スイッチによって BM 上のデータが消される頻度が小さくなり，より効率が良くなることが予想される．そのような状況を想定した場合として，方法 **2** による結果例が **Fig. 4.5** に示されている．ここでは，方法 **2** によって 2 または 3 個のプログラムの多重処理を行なったときの効

率 P_P と，同じプログラムの処理を方法1により行なったときの P_P とを比較している．これをみれば，容量が1，2キロ語の場合に比べて，4，8キロ語では大きな差が現われている．カラムの大きさ k についても，容量 C の場合と同じく k が大きいほど P_B は増大する．しかしタスク・スイッチを考慮した場合，方法1では，k が4以上になると P_B はほとんど増加しない(**Fig. 4.4**)．アドレス変換機構の価格の点なども考慮すれば，k を4より大きくする必要性は小さいと考えられる．

4.3 書込み方式およびトレース・データの違いによる影響

各プログラムの動的な振舞いの違いによる効率への影響について，各トレース・データの方法1による結果を **Fig. 4.6** に示す．BM の効率は，S, B, R などのアドレス参照の動特性によってあまり大きな影響を受けず，むしろ走行中の入出力動作や必要ブロック量 (D) によって相違を生じている(例えば a2の例)．特にコンパイル時のは，実行時のに比べて全般に P_B が低くなっているが，これは，走行中に必要なブロック量が多いこと (**Fig. 2.4** 参照)，タスク・スイッチの原因となる TRAP 命令が多いことなどによる．また，コンパイラ走行時におけるデータ（原始プログラム）の違いは，それらの目的プログラムの違いに比べて，効率への影響は小さいと考えてよい．

書込み方式については，through-storing と post storing の両方式を比べた時，post-storing 方式のほうが良い効率を与えている．さらに，through-storing 方式の効率は，各プログラムにおける書込み命令の実行頻度に大きく左右されている (**Fig. 4.6** 参照)．

5. あとがき

このシミュレーションでは，バッファ・メモリをもつ計算機システムの簡単なモデルを想定し，実際のアドレス・パターン・データを用いてバッファ・メモリの効率を解析した．

実際の計算機システムにおけるバッファ・メモリ方式の効率を取り上げる場合，

- ハードウェアの面として，先行制御，
- ソフトウェアの面として，オペレーティング・システムのもとでのタスク制御

の影響を考慮に入れることが重要である．ここでの解析では，前者に関しては考慮していないが，実際に先行制御が行なわれているときには，先取りによるMMからBMへの転送があり，その一部は結果的にむだな転送を行なっているとしても，CPU の速度を向上させることになろう．また，先行制御によって MM への書込みと次命令処理の並行動作が可能となり，through-storing 方式の場合に，書込み命令頻度の違いから生ずるプログラム間の効率のばらつきが小さくなると考えられる．つぎに，後者の問題に関しては，タスク・スイッチごとに BM がクリアされる最悪のケースと，ユーザ・プログラム間だけのタスク・スイッチを考慮した比較的良い状況のケースを想定して，多重プログラム処理時の解析を行なったが，実際には，モニタ・タスクの走行や入出力による MM の書換え，割込み処理なども問題となろう．

先行制御におけるハードウェアの方式やタスク制御におけるソフトウェアの方式は，実際のシステム間でかなり違いがあり，上のようなことを考慮した効率の解析は，各システムごとの方式およびその動作環境を詳しく取り入れたシミュレーションを行なうか，または実際に動作しているシステムのモニタリングによらねばならない．さらに，今後のソフトウェアの課題として，バッファ・メモリや先行制御のハードウェアを有効に使用するプログラムの作成法を考えることも重要であろう．

なお，ここで述べたアドレス・トレースおよびバッファ・メモリのシミュレーションは，京都大学大型計算機センターでの開発計画の一つとして行なわれたものである．

参考文献

1) J. S. Liptay; Structural Aspects of the System/360 Model 85 II The Cache, IBM Systems J., Vol. 7, No. 1, pp. 15-21, 1968.
2) S. S. Sisson and M. J. Flynn; Addressing Patterns and Memory Handling Algorithms, Proc. AFIPS 1968 FJCC, Vol. 33, pp. 957-967.
3) R. M. Meade; On Memory System Design, Proc. AFIPS 1970 SJCC, Vol. 36, pp. 33-43.
4) 飯塚；キャッシュ・メモリ・システム(I)(II)，情報処理，Vol. 13, No. 7, pp. 467-473, No. 8, pp. 540-547, 1972.
5) 北川，金沢，萩原；バッファ・メモリ・シミュレーション，情報処理学会第 12 回大会, 1971.
6) 中村，北川，金沢，萩原；バッファ・メモリ方式の効率について，情報処理学会第13回大会，1972.

(昭和48年3月10日受付)(昭和48年9月13日再受付)