アドレス軌跡を利用した計算機システムの解析法*

益田隆司** 高橋延匡**

1. まえがき

ページング機構を有する計算機システムの解析，IBM System/360 model 85 のようなキャッシュ方式の解析などを行なうためには，まずプログラムの実行時の動作を把握することが重要である．たとえば，ページング・システムを考える際，ページ・サイズをどの位にすればよいか，使用する仮想メモリ空間に対してどの位の大きさの主メモリを用意すればよいか，主メモリから2次メモリに情報をおいだす必要があるときどのようなアルゴリズムで行なえばよいか，といったことを決めるには，そこで実行されるプログラムの動作を把握しなければならない．プログラムは完全に sequcntial に動作するものでもないし，また random にアドレス空間上を行ききするものでもない．前後に参照されるアドレスのあいだにはかなりの因果関係があるはずであり，この関係をうまく把えて上記のようなシステム・パラメータを決定しなければならない．キャッシュ方式の解析の場合も同様である．

プログラム動作を把握するための最も原始的な最も正確な方法は，そのプログラムを実行したときのアドレス軌跡をとることである．実行されるプログラムの全アドレス軌跡を外部記憶装置上にとり，その記録にもとづいてプログラムの動作を把握する．

また，プログラムのアドレス軌跡はこのような目的の他にも，プログラムの tune-up 等にも利用できる．tune-up したいプログラムのアドレス軌跡をとることにより，そのプログラムのどのモジュールにどの位のオーバヘッドがかかっているかということが直ちに分かるので，それにもとづいて tune-up を行なうことができる．

この報告では，このようなアドレス軌跡の各種の利用法について，これまで発表されている報告をまとめてみる．

* Analy·is of Computer System Using Address Tracer, by Takashi Masuda & Nobumasa Takahashi (Central Research Laboratory, Hitachi, Ltd.)

** 日立製作所中央研究所

2. アドレス軌跡をとるための装置

プログラム実行時のアドレス軌跡をとるには，ソフトウェアを用いたトレーサによる方法と，ハードウェアを用いたモニタリングによる方法の2つが存在するが，ソフトウェアによる方法がより一般的である．その1例として，HITAC 5020 TSS を開発した際に作成したアドレス・トレーサ (PATTERN) を紹介する[8]．

5020 TSS は，セグメンテーション，ページング機構を有したタイムシェアリング・システムである．ページング機構下におけるプログラム動作の把握，システムの tune-up を目的として，プログラム実行時のアドレス軌跡をとるシステムを開発した．PATTERN により，5020 TSS のもとで動作する任意のプログラムのアドレス軌跡をとることが可能である．指定により，ユーザ・プログラムのみのトレース，OS (オペレーティング・システム) のみのトレース，指定したセグメントのみのトレース，全実行命令のトレース等が可能である．出力は磁気テープに行ない，テープ1巻に 50～60 万ステップのアドレス軌跡が格納できる．各命令ごとの磁気テープ上の記録内容は，命令コー

Single Instruction	Instruction Code	α | β	
	Procedure Base Register		Sequential Control Counter
Double Instruction	s_1		l_1
	s_2		l_2
	s_3		l_3
	s_4		l_4

α: Bit of indirect addressing for the first operand
β: Bit of indirect addressing for the second operand
(s_1, l_1)=(operand segment, location)
(s_2, l_2)=(indirect segment, location)
(s_3, l_3)=(second operand segment, location)
(s_4, l_4)=(second indirect segment, location)

Fig. 1 Output format of PATTERN for each instruction (HITAC 5020 TSS)

ド，命令アドレス，オペランドのアドレス，インダイレクト・アドレシングの場合はそのアドレス等である．命令ごとのフォーマットは Fig. 1 のようである．この磁気テープ・ファイルは，後に FORTRAN で処理可能である．

PATTERN の場合には，実行時の命令の軌跡をとることを主に考えているが，ある種の目的に対しては，メモリ参照の軌跡を考えた方がより正確である．1つの命令により複数個のデータが参照されるような場合にも，参照された全メモリアドレスを記録する．

ハードウェア・モニタを利用して，アドレス軌跡をとる方式も報告されている．UNIVAC 1108 マルチプロセッサ用に開発されたデータ収集装置がそれである[13]．この装置は次の3つのモードを有する．

（1） ソフトウェア・トレース・モード

プログラムの流れに応じて，ユーザ・プログラムの各所，および OS の各所で費やされる時間をうるのに利用する．データ収集装置は各とびこし命令のとびこし先のアドレスとその時刻を記録する．

（2） 命令トレース・モード

実行された命令の型，軌跡，利用されているレジスタ等に関する情報が得られる．

（3） メモリ参照モード

命令およびオペランドをうるために参照されたメモリ・アドレスの軌跡が得られる．ただし，入出力によるメモリ参照は含まない．

データ集収装置から得られるアドレス・データはドラムに格納する．収集速度は，2.5 μsec/word であるのに対し，ドラムの速度は，5 μsec/word であるので，2つの大きなバッファを設けている．データ収集は高高ドラムがいっぱいになるまでしか行なえない．その後，ドラム上の記録を磁気テープ上にはきだす．

3. プログラムの tune-up

アドレス軌跡を用いることにより，OS，言語プロセッサをはじめとする各種のプログラムの tune-up を行なうことができる．5020 TSS での経験を述べる．

5020 TSS は，稼動開始後，しばらくシステムの効率が悪く，その原因を分析する必要が生じ，OS のオーバヘッド，各 SVC，およびわりこみ処理ごとの OS のオーバヘッドにしめる割合を求めた．予想されたことではあるが，効率があがらない理由は，オンデマンド・ページングによるミッシング・ページ・フォールトの頻発と1回当りの OS 内での処理時間が大きいことであった．このとき，ページ・フォールトが生じた際の OS 内のダイナミックなプログラムの動きを把握するためにアドレス軌跡を利用した．ページ・フォールトが生じた場合，フォールトをおこした相手方のページの性質（そのページが単なる作業用ページであるか，ドラム上のページであるか，ディスク・ファイル上のページであるか，あるいはページ・テーブルであるか等），そのときの主メモリの状態等により，OS 内を走るステップ数は異なる[7]．各場合ごとのアドレス軌跡を求めることにより，OS 内の動きを正確に把握し，OS の改良に役立てた．

このようなシステムの tune-up の効果を定量的に評価するための一般的な方法が，小松らにより OS アナライザとして提案された[15]．OS アナライザの対象とするところは，必ずしも OS である必要はなく，任意のプログラムでよい．OS アナライザの考え方を概略する．

解析の対象とする OS をネットワークとして表現することを試みる．OS をいくつかの機能単位（これをモジュールとよぶ）に分割し，各モジュールをネットワークのノードに対応させる．あるモジュールから他のモジュールへの制御の移動は，対応するノードを結ぶアークとして表現する．このようにして，OS を1つのネットワーク・フローとして表現する．次に OS の実際の動作の軌跡であるアドレス軌跡を用い，ノードの系列の集合をうる．ここでノードの系列とは，OS がある処理をするために用いたモジュールの制御の移動の歴史であり，これをパスとよぶ．たとえば，1つの割込み処理の過程，あるいは，1つの SVC の処理過程が1つのパスに対応する．各パスが実行される頻度を求めることにより，各モジュール間の遷移確率が得られる．また，各モジュールの平均走行ステップ数をアドレス軌跡を利用して求める．

M_{ik} を i 番目のパスの上の k 番目のモジュールの平均実行時間とすると，i 番目のパスの平均実行時間 P_i は次式で求まる．

$$P_i=\sum_k M_{ik}$$

W_i を i 番目のパスの出現頻度（重み）とすると，OS の処理1回当りの実行時間の期待値 E は次式で求まる．

$$E=\frac{\sum_i P_i W_i}{\sum_i W_i}$$

ここで，M_{ik} が M'_{ik} に変化したときの E の値

が，E_1 から E_2 に変化したとすると，このときの改善効率 I は次式で定義される．

$$I=\frac{E_1-E_2}{E_1}\times 100.$$

さらに，モジュールの改善効率 J は次式で定義される．

$$J=\frac{M_{ik}-M'_{ik}}{M_{ik}}\times 100.$$

各モジュールごとに，J を横軸にとり，I を縦軸にとると，各モジュールの改善が全体の効率にどのような影響を与えるかという関係が得られる．5020 TSS の OS を対象とした1例を Fig. 2 に示す．Fig. 2 は OS 内の 'SCHDL', 'RTN' のモジュールを改良した場合のシステム全体の効率に与える影響を示している．

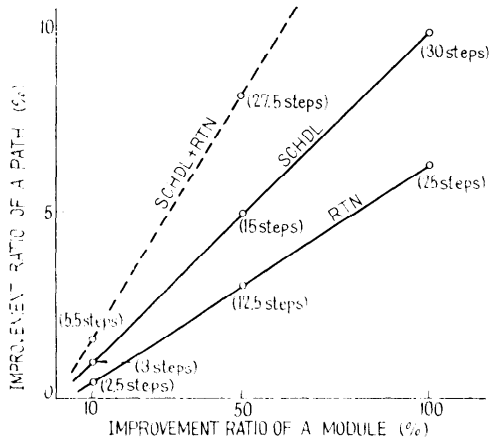

Fig. 2　Output of OS analyzer (HITAC 5020 TSS)

これと同じような実験をコンパイラの動作を把握するために行なっているが，いくつか興味深い結果が得られている．

アドレス軌跡を用いたシステムの tune-up に関連して，興味深い報告が IBM 社の Hatfield & Gerald によりなされている[14]．

仮想メモリシステムで主メモリ・サイズ以上のページを用いる場合，主メモリの有効活用ということは重要な課題である．この点に関し，従来数多く論じられてきたことは，主メモリと2次メモリのあいだのページ出し入れのアルゴリズム，すなわち，スワッピング・アルゴリズムに関することである．主メモリ上にあきページが必要になり，主メモリ上の何ページかを2次メモリにおいだす必要があるとき，どのページを選択すべきかということである．アルゴリズム良否の判定の基準は，ページ・フォールト間の走行ステップ数であり，これが大きいほどすぐれたアルゴリズムと判断される．Hatfield 等は，主メモリの有効活用をスワッピング・アルゴリズムとは異なった今1つの観点から論じている．

あるページに注目した場合，そのページが主メモリに読み込まれてから追いだされるまでに，平均的に密度高く使われるほど，そのページは主メモリ上で有効に用いられたといえる．そこでページング・システム向けのプログラムの構成が可能ではないかと考えられる．ページング・システムに適したプログラムの構成は，平均的なワーキング・セットの大きさが小さなプログラムである．ここでは，ワーキング・セットとは一定量の命令を実行する際に使用および参照されるページの集合と考えてよい．ページング・システム向きに作成したワーキング・セットの小さいプログラムが，無意識に作成したプログラムに対し，どの程度効率に影響を与えるかは興味深い．

Hatfield 等はすでに開発されているプログラムのアドレス軌跡をとり，それにもとづいてプログラムを再構成することにより，効率がどの位上がるかという実験を試みている．

プログラムを再配置可能な単位（セクタとよぶ）に分割する．セクタの大きさは，ページ・サイズの$^1/_3$～$^1/_{10}$ 程度とする．1つのページ内にできるだけ関係の深いセクタを集めることにより，そのプログラムが実行されたときのページ間の遷移の回数をできるだけ少なくし，それが結果において，平均的なワーキング・セットの大きさを小さくする効果をもつと考える．並べかえるための入力情報としては，並べかえを行なおうとするプログラムの実行時のアドレス軌跡を利用する．このアドレス軌跡内に含まれるセクタが並べかえの対象となる．ある種のコンパイラに対しこのような並べかえを行なった結果，そのコンパイラを走らせた場合のページ・フォールトの数（オンデマンド・ページングを仮定している）が，並べかえを行なう前にくらべて，$^1/_2$～$^1/_{10}$ になったと報告している．

4. シミュレーション

プログラムのアドレス軌跡を利用して，ページング方式のシミュレーション，キャッシュ方式のシミュレーション等を行なった例がいくつかある．この種のシミュレーションは，各所で行なわれているものと思わ

れるが，外部に発表されているデータは少ない．

アドレス軌跡を利用したキャッシュ方式の解析が，IBM system/360 model 85 について報告されている[10]．model 85 のキャッシュは，16 k バイトであり，1 k バイト単位に 16 個のセクタに分かれている．キャッシュ・セクタのわりあては，実行中ダイナミックに行なわれる．プログラムがキャッシュ・セクタに存在しないセクタを参照し，キャッシュがすでにいっぱいであれば，いずれかのセクタの場所に新しいセクタをわりあてなければならない．このわりあてのアルゴリズムには，LRU (Least Recently Used) を用いている．あるキャッシュ・セクタがわりあてられたとき，主メモリから 1 k バイト全体が転送されているのではなく，1 つのセクタがさらに 16 個のブロックからなっており，各々のブロックがオン・デマンド方式で読み込まれる．

キャッシュ・アプローチの良否を判定するための基本的道具として，アドレス軌跡を利用している．system/360 OS の下で動作するプログラムについて，19 本のアドレス軌跡の磁気テープ・ファイルを作成している．各ファイルは，250,000 ステップ程度のアドレス軌跡からなっている．このアドレス軌跡を用いて解析した結果の 1 例を Fig. 3, Fig. 4 に示す．Fig. 3 の

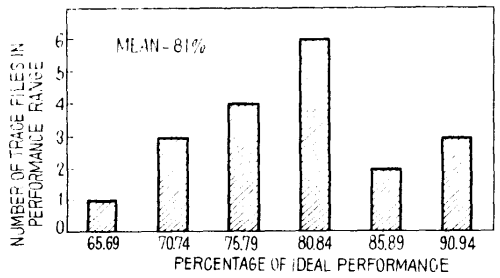

Fig. 3 Model 85 performance relative to single-level storage operating at cache speed

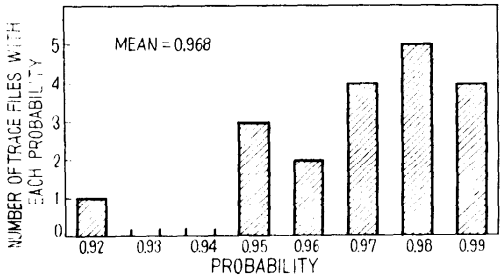

Fig. 4 Probability of finding fetched data in cache (Model 85)

横軸は，主メモリがすべてキャッシュである理想的な場合に対する効率の比をあらわす．(model 85 の主メモリのサイクル・タイムは 1.04 μsec であり，キャッシュは 80 nsec である．) 同図をみると，model 85 の効率は，主メモリがすべてキャッシュと同じスピードのメモリを有するシステムの平均 81 パーセントの効率を有する．

model 85 のキャッシュを設計する際に，キャッシュ・サイズ，セクタ・サイズおよびブロック・サイズをパラメータとして，アドレス軌跡を用いたシミュレーションを行なっている．キャッシュ・サイズ 16 k バイト，1 セクタ 1 k バイト，1 ブロック 64 バイトはこのシミュレーションから求めたものである．

この報告より 1 年前，同じ目的の論文が IBM 社の Gibson により報告されている[9]．そこでは，7000 シリーズのアドレス軌跡を 20 個のプログラムについて，各々約 3 M ステップ，テープ約 300 巻とり，解析を行なっている．また，北川らによる同様の報告もある[11]．

アドレス軌跡を利用して，ページング方式のシミュレーションを行なっているものもいくつかある．仮想メモリを有するシステムでは，主メモリと 2 次メモリのあいだの情報転送の管理はすべて OS が一括して行なうので，主メモリの管理をどのような方式で行なうかが，システムの効率に大きな影響を及ぼす．主メモリから 2 次メモリにあるページをおいだす必要があるとき，そのアルゴリズム (FIFO, LRU, RANDOM 等)，ページ・サイズ，主メモリ・サイズ等と効率 (主メモリと 2 次メモリのあいだの情報転送の回数) の関係を把握するために，アドレス軌跡を利用する[1),8)]．

シミュレーションとは若干異なるが，多段のメモリ構造を仮定したとき，あるアドレス軌跡テープに対して，各メモリ・レベルへのアクセスの頻度を，ページ・サイズ，リプレースメント・アルゴリズム，メモリ・レベルの数，各レベルでのメモリの大きさなどの関数として，簡便に求める方式を提案している論文がある[5]．この方式は上に述べたシミュレーションによるよりも結果をうるための効率はよいが，すべてのリプレースメント・アルゴリズムに対して適用できるものではなく，スタック・アルゴリズムとよばれるアルゴリズムに属するものに対してのみ適用できる．スタック・アルゴリズムに属するアルゴリズムとしては，LRU, LFU (Least Frequency Used), Optimal, RANDOM 等がこれに属する．リプレースメント・

アルゴリズムの性質を理論的に扱った報告として興味深い.

5. プログラムの性質，動作解析

前節に述べたようなシミュレーションではなく，アドレス軌跡を用いてプログラムの性質・動作を目的に応じた角度から把握しようとする試みがなされている.

Denning の "Working Set Model" はプログラム動作に関するモデルとして一般性のある唯一のものであろう[4]. アドレス軌跡に直接は関係しないのでここでは述べないが，あるプログラムのワーキング・セット $W(t,\tau)$ を時刻 $t-\tau$ から時刻 t までのあいだにそのプログラムが参照した情報の集合（ページの集合）と定義し，$W(t,\tau)$ に関する性質を論じている.

アドレス軌跡を利用したプログラムの性質に関連し，最も単純なところは命令の使用頻度の分析あたりからはじまるであろう.

プログラム動作について，ページング・システムが対象でないプログラムの動作は，SVC の系列と，SVC と SVC のあいだの時間として単純にとらえるのが普通である.

プログラム動作について数多く論じている報告は，大部分ページング・システムがその対象である．ページング・システム下でのプログラム動作を論じている報告に，(2), (3), (4), (8), (11) 等があるが，内容的には似かよったものが多い.

Fine の報告[2]は，SDC の Q-32 TSS のもとで動作するプログラムのアドレス軌跡をとり，プログラムが時間的にどのように新しいページを要求していくかを把えている．いくつかのプログラムについて，Fig. 5 のようなページ要求関数を得ている (1.6 μsec/instruction). 金田[6] はこの結果を利用して TSS のシミュレーションを行なっている.

Fig. 5 をみてもわかるとおり，プログラム開始直後のページ要求が非常に頻繁であり，Fine はこれを減らすためにプログラムの再構成が必要であるといっている. **3.** で述べた Hatfield 等の再構成に関する報告は，Fine の報告の 5 年後である.

Freibergs の報告[3] も内容は上記の論文と同じ類に属する．Table 1 に示したようなプログラムについて SVC と SVC のあいだの命令数，要求されるページ数等の統計を求めている．Table 1 のようなデータもアドレス軌跡を利用して求める．結果として，SVC 間に実行される命令数は予想外に短く，10^2～10^3 ステップのオーダである．また，この間のメモリ要求は，平均 4～6 ページと大きく，オンデマンド方式はよくないと結論している.

われわれも 5020 TSS のデータをもとにして，Fine 等と同様の実験を試みたが，その中の 1 つとして，プログラムを実行中の任意の点で切ったとき，それ以前に使用していたページとそれ以後に使用するページの一致度を種々のプログラムについて求めてみた[8]. Denning のワーキング・セット・モデルの有効性を裏付けるデータとして意味があろう.

Table 1 Classes of programs investigated by I. F. Freibergs

Class of Program	Total No. Of Instructions Traced (In Millions)	Percetage of Different Types of Instructions G(Z)	P(Z)	R(Z)	B(Z)	Ratio Of Data Words To Instruction Words (Z)**
FORTRAN Execution (11 programs)	4.02	36	20	11	33	57
Siring Processing	.28	38	23	15	24	61
Simulation (GPSS)	1.29	28	27	20	25	55
List Processing (SLIP)	1.13	26	23	19	32	49
FORTRAN Compiation In-Core Compiler (7 Programs)	1.74	28	22	14	36	54
Cobol Exection	1.89	28	15	14	43	45
AVERAGE		29	22	15	34	51

G=Instructions requiring a data word fetch in memory. (5 microseconds)*
P=Instructions requiring a data word store. (5 microseconds)
R=Instructions referring to registers only. (5 microseconds)
B=Branch, or jump, instructions. (3 microseconds)
*Average instruction time for IBM 7044.
**Data words G+P+MOVE instructions.

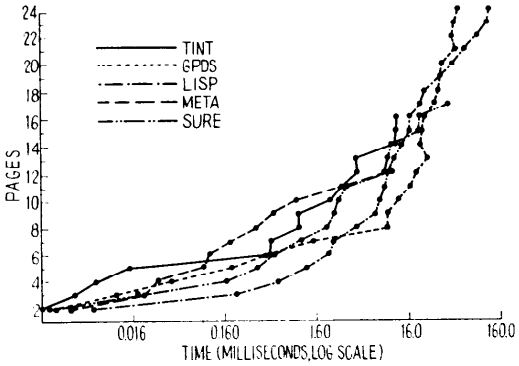

Fig. 5 Page demand (SDC)

6. むすび

アドレス軌跡を用いた計算機システムの解析法につ

いて検討した．ごく最近，IBM社は，370シリーズにすべて仮想メモリ方式をとり入れることを正式に発表した．今後，仮想メモリ方式を有する計算機システムは広く普及していくと考えられる．このようなシステムを解析する際，プログラム動作をどのように把握しモデル化するかは最も重要な問題であり，各種のプログラムのアドレス軌跡は，そのための必須の道具となる．

今後，この報告で述べたようなアドレス軌跡を利用することは，計算機システム解析のための常套手段になるものと思われる．

参 考 文 献

1) L. A. Belady, "A study of replacement algorithms for a virtual storage computer," IBM Sys. J., Vol. 5, No. 2, pp. 78-101, 1966.
2) G. H. Fine et al., "Dynamic program behavior under paging," Proc. of the 21st ACM Nat. Conf., pp. 223-228, 1966.
3) I. F. Freibergs, "The dynamic behavior of programs," Proc. of the 1968 FJCC, pp. 1163-1167, 1968.
4) P. J. Denning, "The working set model for program behavior," CACM, Vol. 11, No. 5, pp. 323-333, 1968.
5) R. L. Maltson et al., "Evaluation techniques for storage hierarchies," IBM Sys. J., Vol. 9, No. 2, pp. 78-117, 1970.
6) 金田,"タイムシェアリングシステムのシミュレーション," 信学誌, Vol. 53-c, No. 2, pp. 119-125, 1970.
7) 益田, 広沢, 吉沢, 木林,"セグメンテーション機構を有するタイムシェアリングシステムの解析," 信学誌, Vol. 54-c, No. 9, pp. 843-349, 1971.
8) 益田, 高橋, 吉沢,"ページング・マシンにおけるスワッピング・アルゴリズムの比較とプログラムの動作解析,"情報処理, Vol. No. 13, pp. 81-88, 1972.
9) D. H. Gibson, "Considerations in block-oriented system design," Proc. of the 1967 SJCC, pp. 75-80, 1967.
10) J. S. Liptay,"Structural aspects of the System/360 model 85. II. The cache," IBM Sys. J., Vol. 7, No. 1, pp. 15-21, 1968.
11) 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.
12) 飯塚,"キャッシュ・メモリ・システム(I),"情報処理, Vol. 13, No. 7, pp. 467-473, 1972.
13) D. T. Bordsen, "UNIVAC 1108 hardware instrumentation system," Proc. of the ACM workshop on system performance evaluation, pp. 1-28, 1971.
14) D. J. Hatfield and J. Gerald, "Program restructuring for virtual memory," IBM Sys. J., Vol. 10, No. 3, pp. 168-192, 1971.
15) 小松 他,"オペレーティング・システム解析の一手法,"情報処理, 受理済.

(昭和47年8月9日受付)