論　文

オンデマンド・ページング・システム動作解析のためのアドレス・パターン・ジェネレータ*

村　岡　洋　一**

Abstract

An address pattern generator based on a new concept called a two stack model is proposed in this paper. Paging behavior of an address pattern generated by this generator is compared with that of an example of the actual address pattern and its usability is indicated.

To show the effectiveness of the generator, paging behavior simulation of a multiprogrammed on-demand paging system is performed using the address pattern generator.

The main emphasis is put on the comparison of various scheduling algorithms. It is shown that to give the equal CPU time to each program in a multiprogramming mix, memory allocation rather than time slicing plays a key role. Associated overhead to accomplish the above goal is also described.

1. まえがき

オンデマンド・ページング・システムの性能の解析は，これまでも確率モデルやシミュレーションによって行われていた．前者はシステムの定常状態のふるまいを把握するのに適しているが，その場合でも多重処理される各プログラムの性格は同一であるという仮定を置くのが普通であって，また過渡的な現象の解析も一般に困難である．実際のプログラムの実行トレース・データを使ったシミュレーションもいろいろ行われている例えば[1]．これらのシミュレーションは1つのプログラムのふるまい（例えばメモリ・サイズとページ・フォールト率の関係）に注目して行われており，その限りにおいては完全に正しい結果が得られる．しかし，スケジューリング手法の検討等のように，あらかじめ定めた性格のプログラムの多重処理を解析した方が便利な場合にはこれだけでは不充分である．

これらに代わる手段としてアドレス・パターン・ジェネレータ（APG と略す）を開発したので，その概要と APG を使ったシミュレーション実施例をここに報告する．APG は実際のプログラムのメモリ・アクセス・アドレス・パターン（プログラムが発生するメモリ・アドレスの数字例）に擬似したパターンを乱数を使って発生させるもので，これを使ってディスクリート・イベント・シミュレータを駆動する．もちろん APG にプログラム実行上有意義なアドレス・パターンを発生させることは不可能であるが，我々の関心はページングのふるまいにあるので，この点で実プログラムを擬似できれば充分である．同様のアドレス・パターン・ジェネレータは他にも報告されているが[2)3)]，2章に述べる理由によって我々のジェネレータの方がより現実に近いものと確信する．APG のパラメータを変化することによって，各種のページング特性を持つアドレス・パターンが容易に発生できる．このアドレス・パターン・ジェネレータの実施例として，多重処理時のページングのふるまいのシミュレーション例を示した．これまで1プログラムのページングのふるまいのシミュレーションの結果は種々報告されているが，多重処理についてのデータはほとんど無い．

2. アドレス・パターン・ジェネレータ

アドレス・パターン・ジェネレータはプログラムの実行に伴ない発生するメモリ・アドレスを，命令ステ

* An Address Pattern Generator for Multiprogrammed On-demand Paging System Simulation by Yoichi MURAOKA (N. T. T. Yokosuka Electrical Communication Laboratory)
** 日本電信電話公社　横須賀電気通信研究所データ通信研究部

ップ対応に発生する．我々はシステムのページングのふるまいに関心があるので，発生されるアドレスはページ番号のみで充分である．アドレス・パターン・ジェネレータとしてはすでにLRUスタック・モデル[2]，単純（VSL）モデル[2][3]等が提案されており，現実のプログラムのページングのふるまいを良く擬似するアドレス・パターンを発生することが報告されている[2]．

LRUモデルではプログラムのページを，一番最近に使われたものから順にスタックに入れて保持している．各スタックの位置 s_i には確率 p_i が対応する．プログラムのページ数を N とすればスタックの大きさもNであり，確率は $p_1 \geq p_2 \geq \cdots \geq p_N$ が成立するように割り付けられる．ただし $\sum_{i=1}^{N} p_i = 1$ である．1ステップごとに乱数 r ($0 < r \leq 1$ の一様乱数) が発生され，$\sum_{i=1}^{K-1} p_i < r$ かつ $\sum_{i=1}^{K} p_i \geq r$ が成立するようなスタック位置Kが決定される（便宜上 $\sum_{i=1}^{0} p_i = 0$ とする）．このスタック位置Kに入れられているページ番号xがこのステップのメモリ・アドレスとして発生され，xがスタックの一番最初の位置に持ってこられスタックの再構成がなされる．

VSLモデルはLRUモデルの縮退型と考えられる．スタックの特定の位置 s_L と確率 λ が決められる．スタックの位置 s_i には $i \leq L$ なら確率 $p_i = \lambda/L$ が，$i > L$ なら確率 $p_i = (1-\lambda)/(N-L)$ が割り付けられる．メモリ・アドレスの発生機構はLRUモデルと同一である．

LRUモデルにおいてはスタックの各位置について確率を与える必要があるのに比べて，VSLモデルでは2つのパラメータ L と λ のみを定めるのみで良いので，実用上はVSLモデルの方が優れている．しかしVSLモデルは次の欠点を持つ．プログラムの性格の一つとしてWorking Setサイズがある[4]．このWorking Setサイズと Windowサイズの関係を **Fig. 1** に例示した．同図のようなWorking Setサイズの立上りの遅いプログラムのアドレス・パターンをVSLモデルで擬似するのは困難である．Ⓐの部分を擬似するにはVSLモデルの L の値をほぼ h に等しくし，かつ λ の値を1に極端に近づければ良いが，このようにすればスタック位置 s_1 から s_L までのページがほぼ $1/L$ の確率で使われることになり，単純に考えれば同図の点線のごとくWindowサイズが L でWorking Setサイズが既に L になってしまう．またこの逆に λ の値を小さくすれば同図一点鎖線のごとく単調に増加してしまう結果となる．この欠点を補なうために我々は，VSLモデルの延長である2スタック・モデルを開発してこれを使ってAPGを作成した．

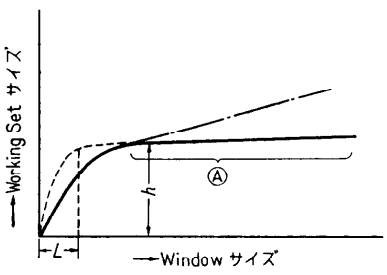


Fig. 1 Relation between working set size and window size

2スタック・モデルでは，VSLモデルのスタックを **Fig. 2** に示すように，3つの部分に分割する．本モデルのページ番号発生アリゴリズムを **Fig. 3**（次頁参照）に示した．本モデルで $L_2 = N$, $\lambda_3 = 1$ とすれば，VSLモデルに縮退できる．プログラムのWorking Setサイズと Windowサイズの関係から，パラメータ値 $L_1, L_2, \lambda_1, \lambda_2$ および λ_3 を求めるめやすを **Fig. 4**（次頁参照）に示した．本モデルでは短い時間間隔では小スタック内から次にアクセスするアドレスを選ぶが，長期間にわたっては中スタック内からアドレスが選ばれることになる．従って Fig. 1 に示すような立ち上りの遅い Working Set 特性を持つアドレス・パターンの発生も可能である．

2スタック・モデルは次のように解釈できる．プログラムは通常少数（L_1）のページを使って実行している．適当な期間これを続けると，ジャンプによってプログラムの他の部分に制御が移行する．この場合まったく無関係な部分に移行するのではなく，これまで使

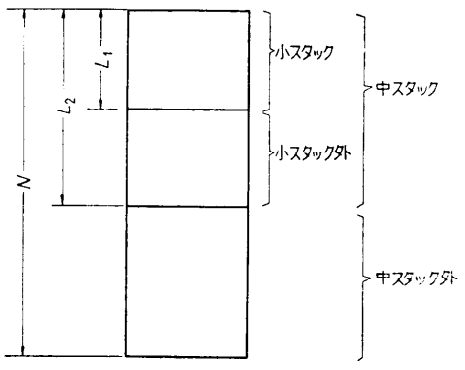


Fig. 2 Stack configuration for a two stack model

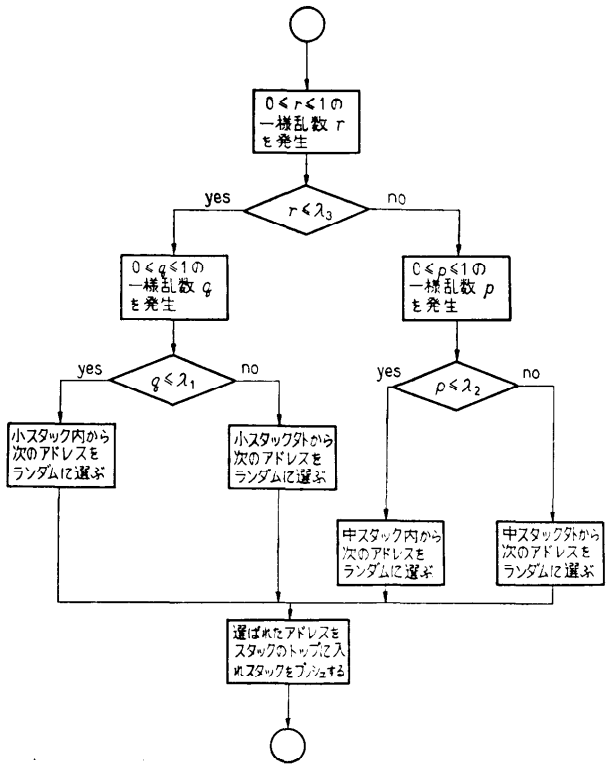
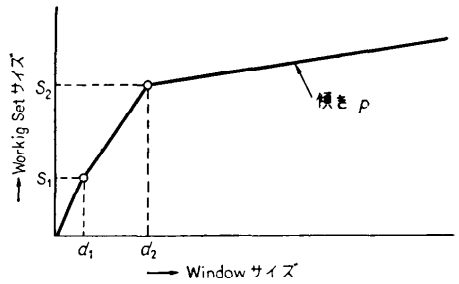


Fig. 3 Flow of address pattern generator

用していた部分と相関関係のある範囲（L_2）内でジャンプする．小スタックは Working Set 特性の立ち上り部の性格を決め，中スタックはなだらかな部分（すなわち Fig. 1 のⒶの部分）の性格を決める．これに対して VSL モデルではジャンプ対象範囲はプログラム全体である．LRU モデルは一番最近に使用されたページ程ひき続いてアクセスされる可能性が大きいというローカリティの原理に基づいているが，先きに述べたように N 個のパラメータ $\lambda_1, \lambda_2, \cdots, \lambda_N$ の値を決める必要があるので実用性に乏しい．

APG がどれだけ良く実際のプログラムのアドレス・パターンを擬似できるかを次に例で示す．**Fig. 5** は APG によって発生させたアドレス・パターン（$\lambda_1=0.998$, $\lambda_2=0.993$, $\lambda_3=0.996$, $L_1=5$, $L_2=40$, $N=200$）と，実際のプログラムの実行トレースによって求めた Working Set サイズと Window サイズの関係を示す．また **Fig. 6** は上記２アドレス・パターンについて，ページ・フォールト率を LRU アルゴリズムで測定した結果である．最後に APG によって発生させたアドレス・パターン（パラメータは上記のもの



$L_1 = S_1$
$L_2 = S_2$
$\lambda_1 = 1 - S_2/d_2$
$\lambda_2 = p/(1-\lambda_3)$
$\lambda_3 = 1 - 1/d_1$

Fig. 4 Estimation of parameters for address pattern generator

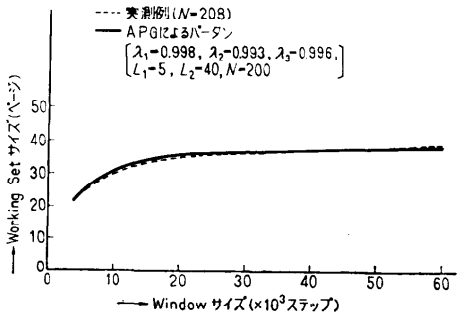


Fig. 5 Relation between working set size and window size-experimental result

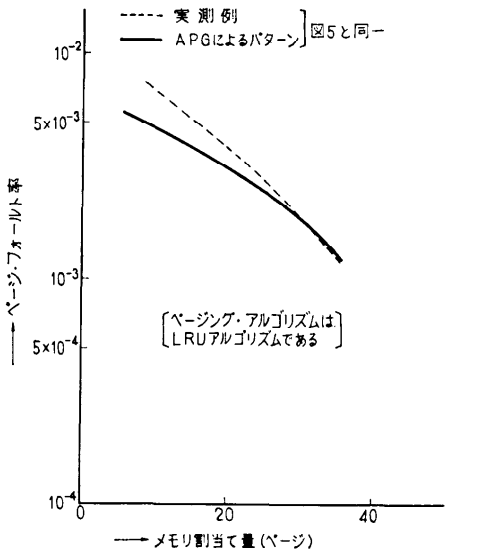


Fig. 6 Relation between paze fault rate and size of memory allocated-experimented result

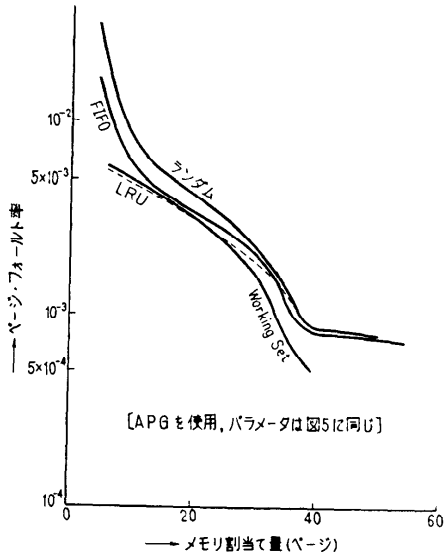


Fig. 7 Comparison of pazing algorithms experimental result

と同一) を使ってページング・アルゴリズムを比較したのが **Fig. 7** である. もちろん, この結果によってページング・アルゴリズムの優劣は判定できない. しかし, 一般に Working Set アルゴリズムが最適で LRU アルゴリズムがこれに続き, ランダム・アルゴリズムが一番劣るといわれている傾向が, Fig. 7 にも現れている. このことから逆に APG によるアドレス・パターンはページングのふるまいに関する限り, 実際のプログラムを良く擬似できることが分かる.

一般にシミュレーションは, 結果の(i)正当性と(ii)一般性の立証が難かしい. APG においては, Fig. 5, 6 および 7 から判断して少なくとも結果の定性的な正当性は保証されていると考えた. 一般性を保つために同一のシミュレーションを異った乱数系列を使用して繰り返し, 同じ傾向の結果が得られることを確認した. 従って本論文中のデータはシミュレーション結果の代表例である.

3. アドレス・パターン・ジェネレータの実施例

3.1 シミュレーションの概略

アドレス・パターン・ジェネレータ等を使ってシミュレーションする利点として,

(i) 従来の解析的手法では分からなかった現象が発見できる,

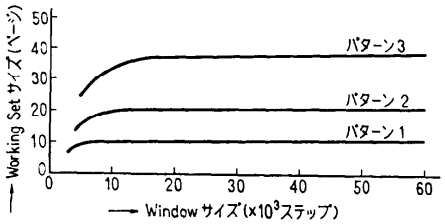


Fig. 8 Working set characteristics of three address patterns

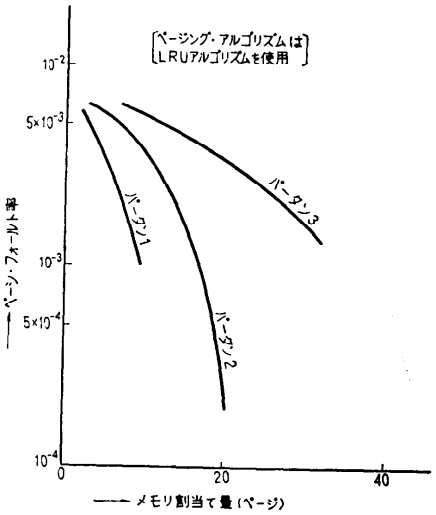


Fig. 9 Paging behavior of three address patterns

(ii) スケジューリング等の動的な効果が評価できる,

等があげられる. 本章ではアドレス・パターン・ジェネレータの実施例として, 多重処理時のページングのふるまいのシミュレーション結果を示す. シミュレーションに使ったアドレス・パターンは次の3つである (**Fig. 8, 9** 参照).

(1) パターン1

$\lambda_1=0.998$, $\lambda_2=0.9993$, $\lambda_3=0.995$,
$L_1=1$, $L_2=10$, $N=200$.

(2) パターン2

$\lambda_1=0.998$, $\lambda_2=0.9993$, $\lambda_3=0.995$,
$L_1=2$, $L_2=20$, $N=200$.

(3) パターン3

$\lambda_1=0.998$, $\lambda_2=0.9993$, $\lambda_3=0.995$,
$L_1=5$, $L_2=40$, $L=200$.

これらのパラメータは実際のプログラムの測定例を参考にして決めたが, 必ずしも一般性があるとはい

い難い，従って以下に示す結果はあくまで実施例である．

3.2 ページング・アルゴリズムのグローバルな適用

M 個のプログラムを多重処理する場合にページング・アルゴリズムxの適用方法として大別すると次の2案がある．

(1) ローカルな適用（ローカルxアルゴリズムと呼ぶ）．

各プログラムに固定サイズのメモリを割り付けて，このメモリ内で各プログラム毎に独立してxを適用する．

(2) グローバルな適用（グローバルxアルゴリズムと呼ぶ）．

特に各プログラムに固定サイズのメモリを割り付けることはせずに，全プログラムのページを同等に扱ってxを適用する．

ページング・アルゴリズムxとして LRU (Least Recently Used) アルゴリズムを使い，パターン3のプログラムを多重処理する場合について，(1)と(2)の比較を行なった結果が **Fig. 10** である．同図で1プログラム当りのメモリ量mは，多重度が M でメモリ総量が V なら V/M である．Fig. 9 と Fig. 10 を比較するとグローバル LRU アルゴリズムの方が，ローカル LRU アルゴリズムよりも優れていることがわかる．すなわちグローバルに適用すると，ページングのふるまい上は1プログラム当りの実効的な使用メモリ量は V/M より大きくなる．例えば Fig. 10 の場合には実効的なメモリ量と実際のメモリ量との比 e は，

$$e=1.0+0.4(M-1)$$

となる．

3.3 CPU 使用時間

本節では，CPU 使用時間の割当てとページング・アルゴリズムとの関係をシミュレーションで調べる．本節の目的は多重処理時におけるタイム・スライスの効果を見ることにある．

多重処理時に，タイム・スライス無しにプログラムを FIFO でディスパッチする (**Fig. 11** の処理) と，性格の良い（すなわち Working Set サイズの，小さい）プログラムのみが優遇される結果になることが予想される．これをシミュレーションで確めたのが **Fig.**

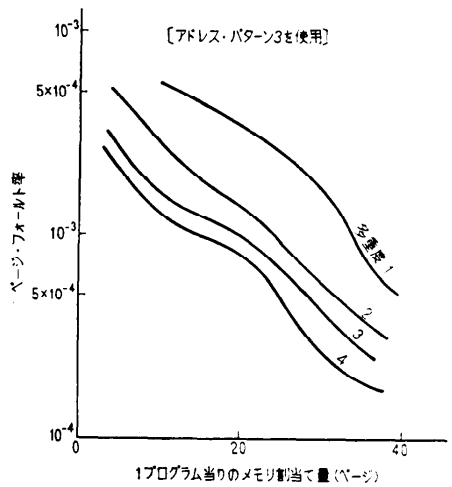


Fig. 10 Effect of global LRU Algorithm

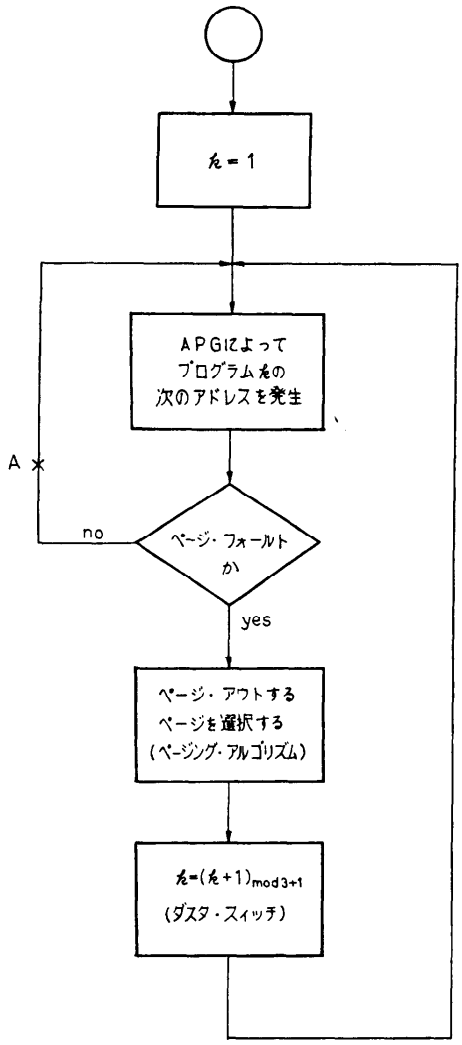


Fig. 11 Flow of simulation

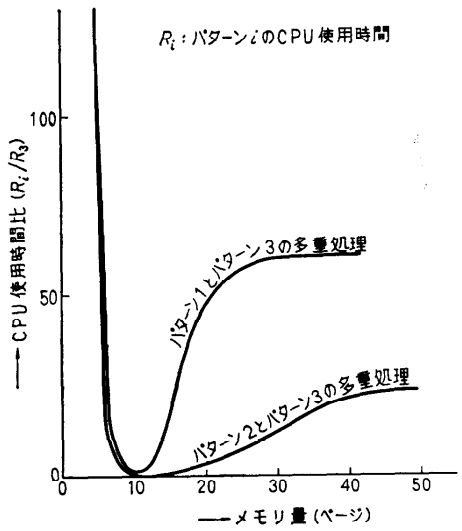


Fig. 12 Run time ratio of programs in multiprogramming mix

12]である．同図は Fig. 11 とグローバル LRU アルゴリズムを使った場合の結果で，アドレス・パターン3のプログラムと1のプログラムまたは3と2のプログラムの多重処理を行なった．この結果を見ると，Working Set サイズの小さいプログラム程多くの CPU 時間を使用する傾向にあることが分かる．

CPU 使用時間の均等割当ての実現手段の一つとしてタイム・スライスがある．これによって Working Set サイズの小さいプログラムがページ・フォールトを起すことなく，長い時間 CPU を独占するのを防ぐことができる．Fig. 11 の処理にタイム・スライス処理を含めて（A点でプログラムがディスパッチされてからの走行時間の積算と，この値があらかじめ定めたタイム・スライス値を超えた場合のタスク・スィッチ処理を行なう），パターン2と3のプログラムの多重処理のシミュレーションを行なった結果が **Fig. 13** である．ページングの入出力時間は0としてある．Fig. 13 には，

（1） CPU 使用時間の比．

各タイム・スライス値を使った時のパターン2の CPU 使用時間とパターン3の CPU 使用時間の比．

（2） ページ・フォールト回数の比．

タイム・スライスを使った場合と使わない場合の，システム全体のページ・フォールト発生回数の比．

の両者が示してある．これから次の観察が得られる．

（1） タイム・スライスは CPU 使用時間均等化に有効である．

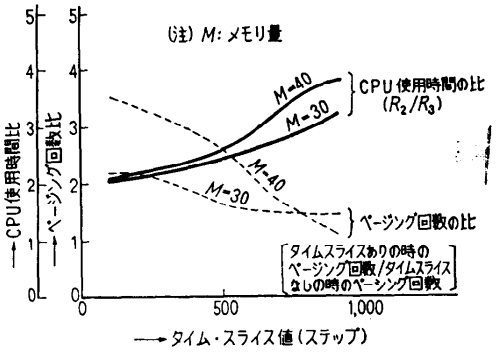


Fig. 13 Effect of time slicing

（2） 代償としてページ・フォールト回数が増大する．

後者の理由は，Working Set サイズの大きいプログラムのディスパッチされる契機が増大することにあると思われる．

3.4 スケジューリング・アルゴリズムの比較

最後にスケージュリング・アルゴリズムの比較を，シミュレーションで試みた例を示す．シミュレーションはパターン1，2および3の多重処理を対象とした．比較したスケジューリング・アルゴリズムは次の4つである．

（1） グローバル LRU アルゴリズムのみ．

（2） グローバル LRU アルゴリズムとタイム・スライス(1000 ステップ) の併用．

（3） PFR アルゴリズムとタイム・スライス(1000 ステップ) の併用．

PFR アルゴリズムでは，ページアウトすべきページはページ・フォールト率が最少のプログラムから，LRU アルゴリズムで選ばれる．ページングのための入出力時間は 10^4 ステップとした．以上3つのスケジューリング・アルゴリズムを，例としてプログラムに対する CPU 使用時間の均等割当ての観点から比較して，**Table 1**（次頁参照）に結果を示した．同表から次の観察ができる．

（1） CPU 使用時間を均等割当てにするには，タイム・スライスを使うことが有効であるが，代償として CPU 使用率が低下する．

（2） メモリ量が大きい時には，PFR アルゴリズムは使用率を低下させることなく，CPU 使用時間均等割当てを達成するのに有利である．

Table 1 Comparison of scheduling algorithms

メモリ量 ＼ スケジューリング・アルゴリズム ＼ 比較項目		グローバル LRU	グローバル LRU +タイムスライス	PFR +タイムスライス
ページ 15	CPU 使用時間比	1/0.8/0.7*	1/0.8/0.7	—
	CPU 使用率	0.056	0.056	—
45	CPU 使用時間比	1/0.003/0.002	1/0.7/0.04	1/0.9/1
	CPU 使用率	0.85	0.88	0.36
60	CPU 使用時間比	1/0.003/0.002	1/0.8/0.9	1/0.9/1
	CPU 使用率	0.85	0.88	0.54
80	CPU 使用時間比	1/0.002/0.001	1/0.7/0.7	1/0.8/0.7
	CPU 使用率	0.95	0.88	0.93

* パターン1の使用時間／パターン2の CPU 使用時間／パターン3の使用時間の比，ただしパターン1の CPU 使用時間を1とした．

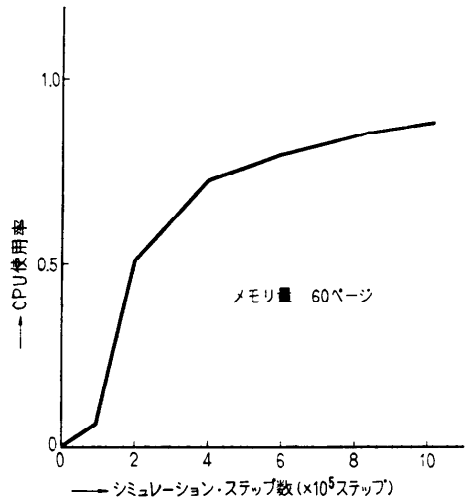


Fig. 14 An example of dynamic behavior of simulation

4. まとめ

新しいアドレス・パターン・ジェネレータと，その実施例を示した．3章にも述べたとおり，アドレスパターン・ジェネレータを使えば，従来の解析的手法では分からなかった現象が発見できる．その例として，ページング・アルゴリズムのグローバルな適用の効果と，タイム・スライスの影響のシミュレーション例を示した．さらに3章には，スケジューリング・アルゴリズムの比較例も示した．3章にも断わったように，使用したアドレス・パターンの一般性が保証されていないので，同章の結果もあくまで参考である．しかし，パターンの選択に注意すれば，種々の性格を持つプログラムを走行させた場合のオンデマンド・ページング・システムの動作解析が可能となる．

APG によるシミュレーション速度は実時間の約100倍で，その立ち上りも **Fig. 14** に例を示したように極めて速い．少ないパラメータで異なったアドレス・パターンを容易に発生できるので，これまで解析的手段のみでは充分行なえなかった現象の解明に，大いに有効と考えられる．

参考文献

1) J. W. Boyse: Execution Characteristics of Programs in a Page-on-Demand System, CACM, Vol. 17, No. 4, pp. 192～196 (1974).
2) J. R. Spirn & P. J. Denning: Experiments with Program Locality, Proc. FJCC, pp. 611～621 (1972).
3) 飯塚，照井：新しいアドレスパターン発生方式によるキャッシュ・メモリのシミュレーション，情報処理，Vol. 14, No. 9, pp. 669～676 (1973).
4) P. J. Denning: The Working Set Model for Program Behavior, CACM, Vol. 11, No. 5, pp. 323～333 (1968).

(昭和49年11月1日受付)
(昭和50年1月25日再受付)