解　説

バーチャル・メモリ向きのプログラム構造*

益　田　隆　司**

1. まえがき

バーチャル・メモリ方式の最大の機能は，プログラマが，主メモリの大きさを意識しなくてもプログラムが作成できることであるが，逆にこの機能は，その設計方式，利用方式によって，システムの性能に大きな影響を及ぼす．プログラムが使用する総メモリ量が，主メモリの大きさに比較して，ある閾値を越えると，計算機本体の処理時間の殆んどが，主メモリと2次メモリの間のページ転送に費やされる，所謂，“thrashing”の現象が生ずることが知られている．このような現象を防ぎ，かつ，システムを効率よく動作させるために，バーチャル・メモリ方式の実験・研究段階の時期から，プログラムのページング動作，オペレーティング・システムのタスク制御，メモリ制御の方式に関して非常に数多くの研究がなされてきた．

さらにこの数年間は，その実用化に伴ない，与えられたシステム構成，オペレーティング・システムの制御方式のもとでのバーチャル・メモリ・システムに向いたプログラムの構成法，作成法の研究が数多く発表され，プログラム構成法が，性能にきわめて大きな影響を与えることが確かめられている．本文では，この点に焦点を合わせて解説したい．

2. プログラムの局所性

バーチャル・メモリ・システムのもとで，プログラムが効率よく動作するためには，そのプログラムが使用するページ数に対して，ある割合以上の主メモリ・ページを必要とし，この割合に到達するまでは，プログラムは急速に新しいページを要求し，また，これ以下の主メモリ・ページ内でそのプログラムを動作させると，2次メモリとの間のページ転送が頻発する性質があるが，大きさの等しい，機能的に同一のプログラムでも，その構造により，この割合が大きく異なる．個々のプログラムについて，この割合を下げることができれば，それらを各々，より少ない主メモリ・ページ内で効率よく動作させることができ，かつ，それにより，実質的な多重プログラムのレベルも上がり，システム全体の効率向上に寄与することになる．したがって，あるプログラムが全体として利用するページ数に比較して，そのプログラムが効率よく動作するために必要な主メモリ・ページ数が少なくてもよいプログラムを“局所性の良い”プログラムとよぶならば，局所性の良いプログラムほど，バーチャル・メモリ・システムには望ましいプログラム構造である．プログラムの局所性を定量的に表わす1つの量として，ワーキング・セットが利用できる．あるプログラムを実行中，時刻 t において，そこから過去の T 時間にそのプログラムが使用した異なったページの集合をワーキング・セットとよび，$W(t, T)$，そこに含まれるページ数をワーキング・セット・サイズとよび，$w(t, T)$ で表わす．また，$w(t, T)$ の時間平均を $\overline{w(T)}$ で表わす．

バーチャル・メモリ・システムの実用化に伴ない，局所性の良いプログラム構造を如何にして作るかが，いくつかの角度から検討されている．これらを分類すると，

（1） 開発済のプログラムを再配置可能なプログラム要素（セクタとよぶ）を単位として並べかえ，関連の強いプログラム要素を同一のページに集めるプログラム再構成法，

（2） バーチャル・メモリ・システムに向いた行列演算法，

（3） その他，バーチャル・メモリ・システムのもとでプログラムを作成する際に留意すべき事項，

に分けられる．

3. 開発済プログラムの局所性の向上方式

3.1 プログラムのメモリ使用効率

すでに開発済のプログラムをセクタを単位として並

---

* *Program Structure for Virtual Memory by Takachi MASUDA (Systems Development Laboratory, Hitachi, Ltd.,)

** (株)日立製作所　システム開発研究所

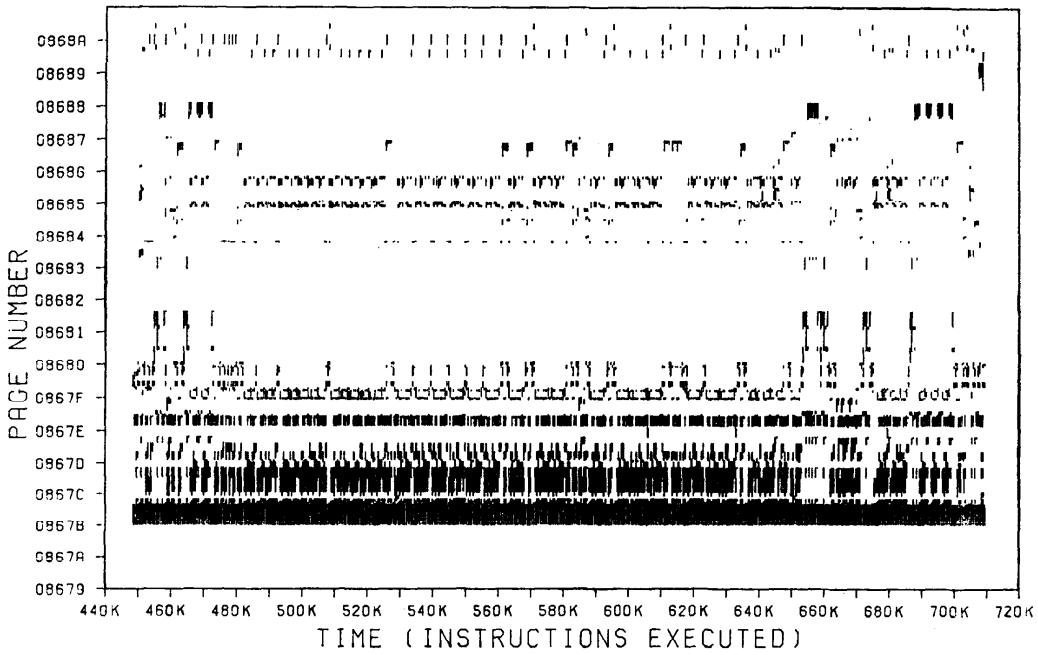


図-1 プログラムのメモリ使用状態の1例

べかえることにより，局所性を向上させる方式がいくつか報告されている[2)~8)]．本論に入る前に，あるプログラムが，その実行に伴ない，セクタ単位に見たとき，どのようにメモリを使用しているかを求めた1つの例を図-1 に示す．たて軸のきざみはページ，横軸は実行命令数を表わす．使用密度が非常に小さいページがいくつか存在していることがわかる．

プログラム再構成の目的は，関係の強いセクタをできるだけ同一ページに集めることにより，各ページ内の使用密度を大きくし，プログラムのワーキング・セットの平均的な大きさを小さくすることである．プログラム本体についての修正は，全く行う必要がないことが利点である．

3.2 プログラム再構成の方式と効果

セクタを単位として，プログラムの再構成を行うには，まず，セクタ相互間の時間的な結びつきの強さを求めねばならない．そのために，通常，再構成の対象とするプログラムを標準的な入力データのもとで実行させた場合のアドレス参照軌跡データを利用する．

Hatfield 等[2)] は，セクタ i とセンタ j の間の結びつきの強さ c_{ij} を，観測時間内のセクタ間の制御の移動または参照の絶対的な回数として定義した．しかしながら，この定義には，$\overline{w(T)}$ を小さくするという目的に対して不十分な点を含んでいる．たとえば，アドレス軌跡データの 1,000,000 命令実行の範囲で，セクタ i からセクタ j への参照が時間的に，

(1) 100,000 命令目ごとに1回ずつ，計 10 回生じた場合，

(2) 100,000 命令目から，101,000 命令目までに，集中的に 10 回生じた場合，

のいずれの場合も，$c_{ij}=10$ であるが，メモリ負荷の点から考えると両者は大幅に異なっている．(2)では i と j を同一のページに入れなかったとしても，それによるページ・フォールトの数は，通常の状態では高々2回のみであるが，(1)では最大 20 回のページ・フォールトが生ずる可能性があり，i, j を同一ページに割り当てることにより高々 10 回に減らすことができる．セクタ間の結びつきの強さの定義にこの点を考慮に入れ，ワーキング・セットの考え方を利用した方式が提案されている[4),6)]．その1つは，セクタ間の結びつきを以下のように定義する[6)]．

アドレス軌跡データ上の総命令数を N，$W_s(t, T)$ を $(t-T)$ 命令目から，t 命令目の間で利用されているセクタの集合とする．このとき，セクタ i と j の

関連度 $r_{ij}(T)$ を以下のように定義する.

$$r_{ij}(T)=\sum_{t=1}^{N}\delta_{ij}(t,T)$$

ここで,

$$\delta_{ij}(t,T)=\begin{cases}1 & \text{when } i,\ j\in W_s(t,T)\\ 0 & \text{otherwise.}\end{cases}$$

Ferrari[4] は，t が進むに伴なって，次に参照するセクタが $W_s(t,T)$ に入っていない点（この時の $W_s(t,T)$ を Critical Working Set とよぶ）に観測点をおき，そこで参照するセクタに対して上記とほぼ同様の定義をしている.

アドレス軌跡データを利用する場合の1つの問題は，入力データの変化に対して，セクタ間の相対的な関連度がどの程度影響を受けるかであるが，再構成手法が特に有効に利用できるシステム・プログラム (OS, 言語プロセッサ等)，専用プログラム等では，ある程度標準的な場合のアドレス軌跡を取れば，結果に与える影響はあまりないことが確かめられている[2),4),6)]. 最近，Ryder[8] は，より長時間のプログラム実行によりセクタ間の関連度を求めるため，OS 内部にソフトウェア・モニタを組み込み，実際のプログラムの実行中のセクタ間の移動の軌跡を収集するシステムを作成している.

プログラム再構成の次のステップは，セクタ間の関連度行列にもとづいて，各セクタをページに割りつけることである．ワーキング・セット・サイズを小さくするということから，ページ内でのセクタ間の関連度の総和が最大になるような割りつけが望ましい．これは，整数計画法により定式化できるが[6]，通常の場合，セクタ数が数百に及ぶことも多く，実用的には利用できない．これに代わる近似的な方法がいくつか提案されている.

Hatfield 等[2] は，セクタ間の関連度を，セクタ相互を結ぶバネの強さにおきかえて，力学的モデルとして解くことを試みている．また，筆者等[6] は，クラスタ分析手法を利用する方法を提案した．通常のクラスタ分析と異なるのは，各個体が各々大きさを有しており，また，1つのクラスタの大きさにページ・サイズの制約があることである.

これらの方法を利用して，実際のプログラムを再構成した場合のワーキング・セット・サイズの変化の例を図-2 に示す[6]．図中，original ordering は再構成前の平均ワーキング・セット・サイズ，H & G 法は，Hatfield 等の方法によるもの，MA 法は，セクタ間の関連度には Hatfield 等の定義しているものを利用し，セクタのページへの割りあてにはクラスタ分析法を利用したもの，WS & MA 法は，その上に，セクタ間の関連の強さにワーキング・セットの考えを利用したものである．WS & MA 法では，再構成前に比較し，$\overline{w(T)}$ が約 40% 前後減少していることがわかる.

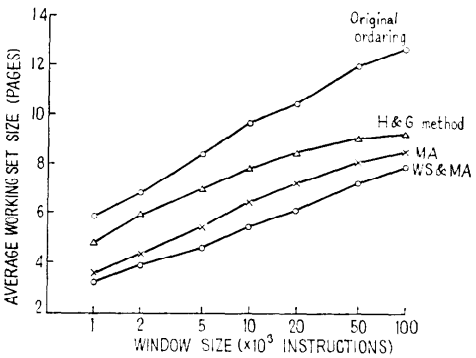


図-2 再構成アルゴリズムの比較 (FORTRAN コンパイラ)[7]

Hatfield 等は，さらに，再構成後の結果にもとづいたメモリ利用状態を，図-1 のような形でディスプレイ上に示し，その後，インタラクティブにワーキング・セット・サイズを小さくすることを試みている.

また，先に述べた Ryder は，ソフトウェア・モニタによるセクタ間の関連度にもとづいて，セクタのページへの割りつけを行う方法を，System 370, の Model 155-II で，OS/VS 2.1 の LPA (Link Pack Area) 内で動作するプログラムに対して試みた結果を報告している．その結果，実測値として，再構成により，平均的に，表-1 のような改善効果を得ている.

表-1 プログラム再構成による効率向上の割合 (Ryder[8])

Parameter	*Change with packed LPA*
Supervisor state execution time	7%
Total run time	5%
I/O interruptions	10%
Paging channel time	21%

また，開発済のプログラムの再構成による局所性の向上とは異なった方法として，Millbrandt 等[19] は，測定用具として，グラフィック・ディスプレイ，ミニコンを利用し，本体側で収集したトレース・データをただちに，ミニコン上で編集しディスプレイ上に表わすシステムを報告している．これによって，各ページの使用状態等，プログラムの局所性がきわめて容易に把握できる.

4. バーチャル・メモリ向きの行列演算法

前章には，開発済のプログラムを再構成可能なプログラム要素を単位として並べかえ，局所性を向上させる方式を述べたが，そのアプローチは，特に，繰り返し多く利用され，かつ，プロシージャ部の占める割合の大きい，各種のシステム・プログラム，専用システムのプログラムの局所性を上げるために有効である．一般のユーザ・プログラムで，その局所性が特に問題になるプログラムは，大きい次元の行列演算を行うプログラムである．

最近の報告で，村田等の報告[14),15)]は，特に，大次元行列の固有値解析法について，従来手法のバーチャル・メモリ方式への適合性を検討し，さらに，バーチャル・メモリ方式向きの固有値解析法について具体的なアルゴリズムを提案し，実プログラムでその有効性を確かめている(本会誌2月号)．より詳細については，原文に譲り，ここでは，行列演算に関して，バーチャル・メモリ・システム向きのより基本的ないくつかのプログラミング技術について述べる．これに関していくつかの報告[9)~13)]があるが，最近の Ershoff の報告[13)]がこれまでの結果を合わせてよくまとめている．Ershoff は行列演算について3つの基本的な規則を上げている．

例題には，PL/I の DO 文を利用し，行列の要素は行方向に格納されていると仮定し，また，ページ・リプレースメント・アルゴリズムとしては，LRU (Least Recently Used) を仮定する．さらに，行列の次元 N は，S をページ・サイズとして，$N<S<N^2$ の範囲を考察の対象にする．

(1) 多重の DO ループの取り扱い

A を $M\times N$ の行列，B を N 次元ベクトルとし，B の各要素に A の対応した列和を求めるプログラムを考える．Bの各要素が0にクリアされているとすると，通常のプログラムは，

```
DO COL=1 TO N BY 1;
  DO ROW=1 TO M BY 1;
    B(COL)=B(COL)+A(ROW, COL);
  END;
END;
```

となる．行列 A, B の占めるページ数を各々 p_A, p_B とすると，A, B 用の主メモリ・ページ数 k が，$2\leqslant k<p_A+p_B$ が興味の対象であり，その範囲で考えると，このプログラムは，$F=Np_A+p_B$ 回のページ・フォールトを生ずる．ところが，同一の機能を持ったプログラムを，DO 文の順序を入れかえて書くと，

```
DO ROW=1 TO M BY 1;
  DO COL=1 TO N BY 1;
    B(COL)=B(COL)+A(ROW, COL);
  END;
END;
```

とすると，$F=p_A+p_B$ となることがわかる．

規則 1：DO ループが多重になる場合の順序は，内側のループで使用される配列の添字ほど，その添字の変化に伴なう配列の要素のメモリ上の距離が小さくなるようにすること．

したがって，ユーザは，利用する言語の配列の格納順序を知っていなければならない．Guertin[10)] は，FORTRAN (配列の格納順序は列方向) で数多くの例を上げて，この点に関するプログラムの書き方を解説している．

(2) 複数行の同時処理

A, B を $N\times N$ の行列とし，B の転置行列を A に求める．通常のプログラムは，

```
DO ROW=1 TO N BY 1;
  DO COL=1 TO N BY 1;
    A(ROW, COL)=B(COL, ROW);
  END;
END;
```

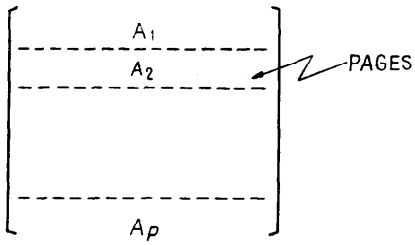
この問題に対しては，規則1は利用できない．いずれにしても，1回の外側のループの処理ごとに，A または B のいずれかの全ページが参照されてしまう．上のプログラムでは，$F=p_A+Np_B$ 回のページ・フォールトが生ずる．しかしながら，1ページ内に数行以上入る場合には，それらを同時に処理するプログラムを書くことにより，ページ・フォールトの数を減らすことができる．プログラムは省略するが，1ページに r_A 行入るとすると，$F=p_A+[N/r_A]p_B$ となる．

規則 2：最も多くのページングをひきおこす添字に関しては，ページが主メモリ内に存在するうちに，その添字の変化に伴なって使用するページ内のすべての要素を処理してしまうこと．

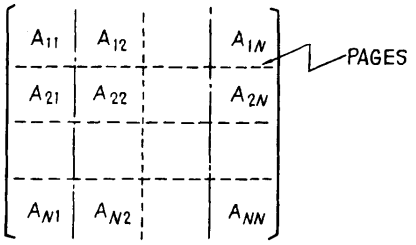
この点に関して，McKellar 等[9)] は，行列の格納法として，**図-3** (次頁参照) のような行格納法，部分行列格納法について効率の比較を行い，多くの場合，部分行列格納法の方がすぐれていることを示している．

(3) 往 復 法

ページ・リプレースメント・アルゴリズムが，LRU



(a) 行 格 納 法



(b) 部分行列格納法

図-3 行列の格納法[9]

方式（最近のシステムは，ほとんど，近似的な LRU 方式を採用している）であることを利用して，行列を一行参照するごとに，参照の方向を逆転させる．すなわち，

規則 3：ゆっくり変化する添字が，頻繁に変化する添字の右側にあるときには，遅く変化する添字が変わるごとに，早く変化する添字のきざみの符号を逆転させる．

次に，これらの規則を実際の行列演算に使用した場合の効果を，行列の転置，積等を例にして確かめている．行列の転置の例を上げると，まず，通常のプログラムは以下のようである．

```
DO ROW=1 TO N-1 BY 1;
  DO COL=ROW TO N BY 1;
    TEMP=A(ROW, COL);
    A(ROW, COL)=A(COL, ROW);
    A(COL, ROW)=TEMP;
  END;
END;
```

行列 A が 20 ページより成り，1 ページ＝512 語とする．上記のプログラムを修正して，複数行の同時処理 (Multiple Rows)，往復法 (Alternating Direction)，さらに，この両者を考慮したプログラムを書くと，ページ・フォールトの数は，図-4 に示すように変化し，

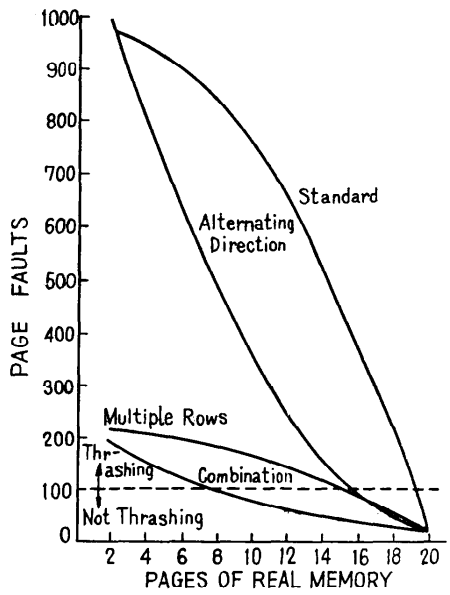


図-4 行列の転置アルゴリズムの比較 (Ershoff[13])

これらの規則の有効性が確かめられる．

5. プログラム作成上の留意事項

前章では，一般のユーザ・プログラムで，プログラムの局所性が特に問題になる大次元の行列の演算法について，いくつかの基本的な規則を述べたが，本章では，行列演算に限らず，新しくプログラムを作成する際に，局所性の高いプログラムにするために留意すべきいくつかの点について述べる．この点については，Morrison[17]，石田[18]，Millbrandt[19] 等がまとめている．以下にその代表的なものをいくつか記すと，

1) 特殊処理用のプログラム，エラー処理プログラム等，通常使用されないプログラムは，プログラムのメインの流れとは離れたところに置く．
2) データ・エリアの初期設定は，プログラムのはじめにまとめて行うのではなく，使用する直前に行った方がよい，
3) 前章の規則 1 に関連するが，データの参照は，できるだけ，格納されている順序に従う．また，逆に，参照される順序に従ってデータを格納する，
4) 同時に使用される可能性が高いプログラム，あるいは，データは，できるだけ距離的に近くおき，同一ページに配置される可能性を高くする，
5) 読み出しのみのデータと書き込みをするデータは各々まとめて互いに離しておくようにする．

6) 大きな配列，あるいは，データ・エリアを隅なくサーチするようなことをしない．できれば，細分割の可能性を考慮する，

7) あるプログラムがコールされたときには，できるだけ密度高く利用するようにする，

8) 頻繁に利用されるプログラム，バッファは，できるだけ，ページをまたがぬようにする，

9) 1つのループ内で使用する変数，配列，プログラム等は，できるだけ，連続したメモリ領域にまとめて格納するようにする，

等々である．

Morrison[17] は，さらに，これらの留意事項を，FORTRAN，COBOL，PL/I プログラム向きにもまとめている．

6. むすび

バーチャル・メモリ・システムに適した局所性の良いプログラムを作成するための方式をいくつかの角度からまとめた．

第1は，開発済のプログラムを再配置可能なプログラム要素を単位として並べかえ，各ページ内に互いに関連の強いプログラム要素を集めることにより，プログラムの局所性を高める方法である．これは，特に，繰り返し多く利用され，また，プロシージャ部のウェイトの大きいシステム・プログラム等の局所性を向上させるのに適している．

第2は，一般のユーザ・プログラムで局所性が特に問題になる大きい次元の行列の演算法について，いくつかの基本的な規則を述べた．

第3は，行列演算のほかに，バーチャル・メモリ・システムに向いた局所性の良いプログラムを書くための留意すべき点をまとめた．これらは，システム・プログラムでも，ユーザ・プログラムでも，新しいプログラムを作成する際に，共通的に考慮しておかねばならぬ点である．

参考文献

1) P, J. Denning: Virtual Memory, Computing Surveys, Vol. 2, No. 3, pp. 153～189 (1970).

2) D. J. Hatfield and J. Gerald: Program Restructuring for Virtual Memory, IBM Sys. J., Vol. 10, No. 3, pp. 168～192 (1971).

3) L, W. Comeau: A Study of the Effect of User Program Optimization in a Paging System, ACM Symp. on Ope. Sys., Principles (1967).

4) D. Ferrari: Improving Locality by Critical Working Sets, Comm. ACM, Vol. 17, No. 11, pp. 614～620 (1974).

5) D. Ferrari: Improving Program Locality by Strategy-oriented Restructuring, Proc. of IFIP congress 74, pp. 266～270 (1974).

6) T. Masuda et al.: Optimization of Program Organization by Cluster Analysis. Proc. of IFIP congress 74, pp. 261～265 (1974).

7) 益田，塩田：仮想メモリシステム向きの最適プログラム構成方式と実験，情報処理 Vol. 15, No. 9, pp. 662～669 (1974).

8) K, D. Ryder: Optimizing Program Placement in Virtual Systems, IBM Sys. J., Vol. 13, No. 4, pp. 292～306 (1974).

9) A, C. McKellar and E, G. Coffman: Organizing Matrices and Matrix Operations for Paged Memory Systems, Comm. ACM, Vol, 12, No. 3, pp. 153～165 (1969).

10) R, L. Guertin: Programming in a Paging Environment, Datamation, Vol. 18, No. 2, pp. 48～55 (1972).

11) B, S. Brown et al.: Sorting in a Paging Environment, Comm. ACM, Vol. 13, No. 8, pp. 483～494 (1970).

12) C, B. Maler: Matrix Computations with FORTRAN and Paging, Comm. ACM, Vol, 15, No. 4, pp. 268～270 (1972).

13) J, L. Ershoff: Some Programming Techniques for Processing Multidimensional Matrices in a Paging Environment, Proc. of NCC, pp, 185～193 (1974),

14) 村田：仮想記憶を意識した大次元固有値解析，京大数理解析研究所講究録第 199 号 (1974).

15) 村田，堀越：対称帯行列を三重対角化するための新アルゴリズム，情報処理，Vol. 16, No. 2, pp. 93～101 (1975).

16) A, A. Dubrulle: Sobution of the Complete Symmetric Eigenproblem in a Virtual Memory Environment, IBM J. Res. Dev., Vol. 16, No. 6, pp. 612～615 (1972).

17) J, E. Morrison: User Program Performance in Virtual Storage Systems, IBM Sys. J., Vol. 12, No. 3, pp. 216～237 (1973).

18) 石田：超大型コンピュータの技術—仮想記憶とプログラミング，bit, Vol. 15, No. 13, pp, 1407～1412 (1973).

19) W, M. Millbrandt and Juan Rodriguez-Rosell: An Interactive Software Engineering Tool for Memory Management and User Program Evaluation, Proc. of NCC, pp. 153～158 (1974).

(昭和50年7月4日受付)