数 値 解 析 28－3
（１９８９．３．１４）

階層型メモリーを共用するシステムでの並列処理

寒 川　光
日本アイ・ビー・エム（株）　ＮＩＣマーケティング・センター

IBM の ES/3090 は、最大６個までプロセッサを持つことのできるシステムである。ES/3090 のメモリー空間は通常のメモリー（中央記憶機構）の他に、上位にキャッシュ・メモリー、下位に拡張記憶機構の２つのバッファー領域をそなえた仮想メモリー空間としてサポートされる。通常 FORTRAN プログラムからはこの２つのバッファー領域を明示的に考慮する書き方はしないが、本報告では IBM VS FORTRAN 第２版のベクトル機能と多重タスク処理機能を用いて、これらのシステム的特性を考慮したプログラムを作成し、ベクトル計算、並列計算を駆使した状態での記憶階層の性能を測定してみる。

Parallel Computing on a Shared Hierarchical Memory System

Hikaru Samukawa
NIC Marketing Center, IBM Japan, Ltd.
Shinjuku Sumitimo Bldg., 6-1, Nishi-Shinjuku 2-chome, Shinjuku-ku, Tokyo 163, Japan

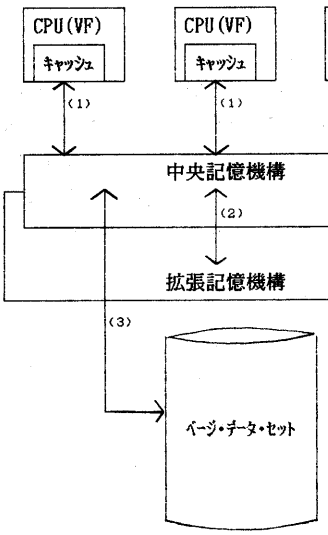
IBM ES/3090 can have up to 6 processors. Its address space comprises two buffer areas, high-speed buffer (cache memory) and expanded storage, in addition to central storage (memory as a standard meaning). Usually, FORTRAN program does not explicitly specify these buffer areas, but in this report, the performance of these buffer areas is tested under the vector and parallel computational environment using vectorization function and multi-tasking facility provided by IBM VS FORTRAN version 2.

1．はじめに

並列処理のためのアルゴリズムは、メモリーを共用するシステム（shared memory system）を前提とするものと、メモリーが個々のプロセッサに属するシステム（message passing system）を前提とするものとで異なる。IBM の ES/3090 は、最大６個までプロセッサを持つことのできるシステムであり、IBM の VS FORTRAN の多重タスク処理機能、またはパラレル FORTRAN を用いることにより、ひとつのジョブが同時にこれらのプロセッサを使用することができる。この場合、メモリー共用型の並列処理システムに該当する。また、個々のプロセッサにはベクトル機構(VF:Vector Facility)を取り付けることができるため、ベクトル処理も並列処理とともに用いることができる。[1)]

メモリーは FORTRAN プログラムからは単一の仮想アドレス空間として扱われるが、実際にはキャッシュ（高速緩衝記憶機構）、中央記憶機構（CS：Central Storage）、拡張記憶機構（ES：Expanded Storage）、ページ・データ・セットの４とおりの階層をもっている(図1)。この記憶階層の制御はシステムによって行われ、FORTRAN プログラムからは明示的には制御しない。これらの記憶階層を考慮したプログラムは考慮せずに書かれたプログラムよりも実行速度が速い。行列と行列の積において、Liu[2)]らはメモリー階層が増えるに従って行列を小行列に分割してループのネストを深くするアリゴリズムの有効性を述べている。

本報告では、並列処理の環境で記憶階層を考慮したときのプログラムの姿や、ES/3090 の記憶階層についてテストした例について述べる。



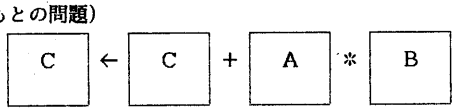
	移動方法（タイミング）	移動する者	移動時間
(1)	命令実行中	H／W	ユーザーのCPU時間
(2)	ページ・フォールト割込み ページ移動命令（同期命令）	ＯＳ	システムのCPU時間
(3)	ページ・フォールト割込み 入出力割込み	ＯＳ	システムのCPU， 入出力時間

図１　ＥＳ／３０９０の記憶階層

2．テスト・プログラム

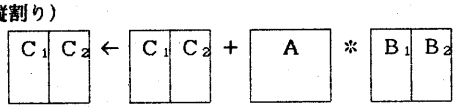
行列と行列の乗加算サブルーチンは、行列を任意のサイズの小行列に分割して、それぞれの小行列に対して再び同じインターフェースをもつサブルーチンを呼び出すことで処理できる。この特徴を利用すると、プログラムからメモリーをアクセスする順序を容易に変更することが可能で、階層型メモリーの性能測定に好都合である。

（もとの問題）



$C \leftarrow C + A * B$

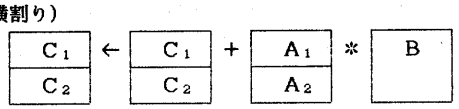
（縦割り）



$C_1 \leftarrow C_1 + A * B_1$

$C_2 \leftarrow C_2 + A * B_2$

（横割り）



$C_1 \leftarrow C_1 + A_1 * B$

$C_2 \leftarrow C_2 + A_2 * B$

2．1　ベクトル化

行列の乗加算は３重にループ・ネストした問題である。

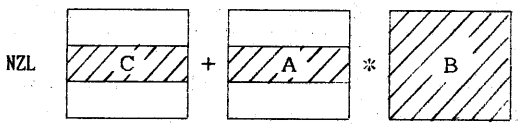
$$c_{ij} \leftarrow c_{ij} + \sum_{k=1}^{m} a_{ik} \cdot b_{kj} \quad , \quad \text{where} \quad i=1,l,\ j=1,n$$

ベクトル化はｉ，ｊ，ｋいずれの添字についても可能であるが、ここではｉを選ぶ。ES/3090 ベクトル機構では256 (ES/3090 の機種によっては128) 以下のベクトル長のベクトル演算を繰り返すことで、ハードウェアのもつベクトル長よりも長い論理ベクトル長のベクトル演算を処理する。この繰り返しループをセクショニング・ループと呼ぶ。セクショニング・ループを演算　$y_i \leftarrow y_i + \alpha * x_i$　を例に、擬似的なベクトル命令を用いて説明する（なおここで用いる擬似命令はベクトル・アーキテクチャーの定める厳密なもの[3]とは異なる）。

```
［セクショニング・ループ］
  yi ← yi＋α＊xi    ,  where  i=1,l (l>256)
        LD      FR, α  ……  αを浮動小数点レジスターへロード
     ┌(sectioning loop)
     │  Vload   VR,(yi)  ……  yのセクションをベクトル・レジスターへロード
     │  VM&A    VR,FR,(xi)……  xのセクションのα倍をベクトル・レジスター（y）に加える
     │  VStore  VR,(yi)  ……  ベクトル・レジスターの内容をyへストア
     └  Branch
   ＶＲ： ベクトル・レジスター，ＦＲ：浮動小数点レジスター
  (yi)： メモリー上のベクトルyiのセクション、１セクションは256以下の要素を持つ。
```

したがって、３重のループのうちｉに関するループがベクトル化され２重になるが、ｌ＞256の問題ではセクショニング・ループが追加され、再び３重のループの問題になる。セクショニング・ループは通常コンパイラーによって自動的につくられるが、ここではプログラムで明示的に取り扱う。行列乗加算にセクショニング・ループを形成するサブルーチンを示す。

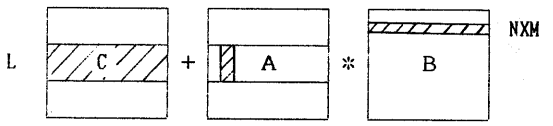
```
      Subroutine DMPYSE(A,LDA,B,LDB,C,LDC,L,M,N)
      double precision A(LDA,M),B(LDB,N),C(LDC,N)
      common/system/LVS …… ベクトル・レジスター長
        NZ=MIN(L,LVS)
        NII=(L-1)/NZ+1
          do II=1,NII
            I=(II-1)*NZ+1
            NZL=MIN(NZ,L-I+1)
            call DMPY__ (A(I,1),LDA,B,LDB,C(I,1),LDC,NZL,M,N)
          end do
        return
```



２．２　キャッシュ容量の考慮

ES/3090 では CPU 内部に高速メモリー（キャッシュ）を備えている。ベクトル命令のオペランドのキャッシュ・ヒット率を高めるために、行列の乗加算を小行列の乗加算の集まりとして取扱うことで、計算速度を向上させることができる[4]。セクショニングの処理が行われた状態で次のサブルーチンを使用すると、キャッシュ容量の考慮により、キャッシュ・ヒット率を高めることができる。

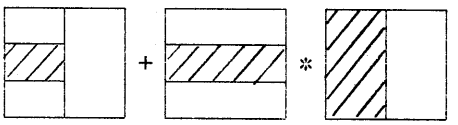
```
Subroutine DMPYAQ(A,LDA,B,LDB,C,LDC,L,M,N)……Lは256以下
double precision A(LDA,M),B(LDB,N),C(LDC,N)
common/system/ …,NCACHE ……  有効キャッシュ容量
NX=NCACHE/L
NMAT=(M-1)/NX+1
  do JJ=1,NMAT
    J=(JJ-1)*NX+1
    NXM=MIN(NX,M-J+1)
    call DMULAD(A(1,J),LDA,B(J,1),LDB,C,LDC,L,NXM,N)
  enddo
return
```



(L*NXM) ＜ 有効キャッシュ容量

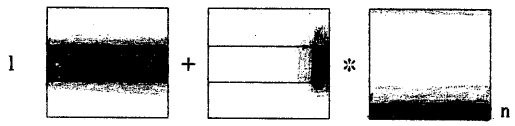
２．３　メモリー階層の考慮

前述のアルゴリズムではセクショニング・ループごとに行列Ｂ全体を参照する。メモリーの階層間に速度差が大きい場合、問題を縦割りにして、セクショニング・ループを通して高速のメモリー階層だけで計算を続けるように考慮することができる。すなわち、考慮すべき２つのメモリー階層間で、速い方のメモリーの容量を与えられると、ＡおよびＣのセクションと使用するＢとがこの容量以下になるように問題を縦に分割して、乗加算サブルーチンをループの中から呼び出すような乗加算サブルーチンを作ることができる。



（斜線部）＜（高速メモリー容量）

このように取扱うことで、行列Ｂを低速メモリーから高速メモリーへ移動する回数を、セクション数［L/256］回から１回に減ずることができる（行列Ａの稼動回数はこのループの回る回数に増える）。この新しいループを記憶容量ループと呼ぶことにする。このループを考慮しない場合、行列データが高速メモリー階層に存在する確率を濃度をつけて表示してみる。これはメモリー資源の管理を LRU：Least Recently Used アルゴリズムによっているからである。



(l*n) < 有効キャッシュ容量

(キャッシュ・ループが最後の小行列A'を計算中の場合)

２．４　並列化

IBM VS FORTRAN 多重タスク処理機能[5)]を用いて、縦割りにされた複数の行列演算を ES/3090-300S の複数の CPU で並列処理する。ここでは多重タスク処理機能のうち JCL (ジョブ制御言語) で与えたタスク数をプログラムに取込む NTASKS、並列計算サブルーチンを始動する DSPTCH、並列計算している複数のタスクの終了を待つ SYNCRO の３つのライブラリー・ルーチンを利用して、縦割りにされた複数の乗加算を並列処理する乗加算サブルーチンを作る。

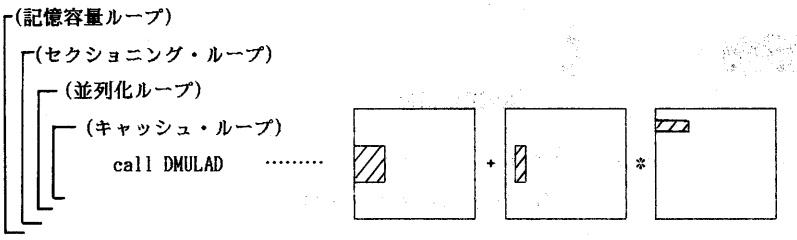
```
Subroutien DMPYPA(A,LDA,B,LDB,C,LDC,L,M,N)
double precision A(LDA,M),B(LDB,N),C(LDC,N)
dimension IARG(30)
call NTASKS(NCPU)
NN=MIN(NCPU,N)
NCOL=(N-1)/NN+1
   do IP=1,NN
      J1=(IP-1)*NCOL+1
      J2=MIN(IP*NCOL,N)
      IARG(IP)=J2+J1+1
      call DSPTCH('DMPY__',A,LDA,B(1,J1),LDB,C(1,J1),LDC,L,M,IARG(IP))
    enddo
call SYNCRO
return
```

ここに表われるループを並列化ループと呼ぶ。

なお、サブルーチン DSPTCH を介して始動された並列サブルーチン DMPY__ はサブタスクとして実行されるためにスケジュールされ、オペレーティング・システムの制御のもとに実行される。サブタスクは論理的なタスクであり、タスクの数が実際の CPU の数と一致している必要はない。

２．５　ループの順序

４つのループをそれぞれＰ（並列化ループ）、Ｒ（記憶容量ループ）、Ｓ（セクショニング・ループ）、Ｃ（キャッシュ・ループ）で表わし、最内側にベクトル長256以下を扱う乗加算ルーチンを配したプログラムを考える。たとえば、Ｒ/Ｓ/Ｐ///Ｃの順に作った６重のループは



を意味し、Pのループで作られたタスクが3つあることを示す。

2．6　データ・タッチ

階層型メモリーのうちキャッシュは CPU に属するが、他の3つの記憶階層はシステムにひとつで全 CPU から共通にアクセスされる。これらの階層間のデータの移動はオペレーティング・システムからがページ単位（4096バイト＝512ダブル語）に行う。この時オペレーティング・システムによる割込みが発生する。並列処理の目的は経過時間の短縮にあるため、割込みの影響を測定、排除したり次のステップで使用されるデータを早めに中央記憶機構に移動させるためのサブルーチンを用意した。

［行列の指定された小行列を中央記憶域に入るためのサブルーチン］

```
      Subroutine TOUCH (A,LDA,NROW,NCOL)
      double precision A(LDA,NCOL)
        do J=1,NCOL
          I=1
10        X=A(I,J)
          I=I+512
          if(I.LT.NROW) GO TO 10
        Y=A(NROW,J)
        end do
      return
```

3．テスト環境と結果

ES/3090 の最大構成は6つのプロセッサを備えた600S型である。この型では中央記憶機構、拡張記憶機構の取付け可能な最大容量はそれぞれ、512MB，2048MB である。このような大きな資源を備えたシステムを夜間のバッチ処理などでフルに活用して大規模な数値計算を行うケースも増えつつある。本報告では3つの CPU（3ベクトル機構付き）を備えた 3090-300S、キャッシュ容量 128KB（各CPU）、中央記憶 256MB、拡張記憶 512MB を専有使用して経過時間とカーネル（サブルーチンDMPYAQ，2.2 キャッシュ容量の考慮）の CPU 時間を計測した。比較の便宜上計算速度は行列サイズがｎのときの浮動小数点演算数$2n^3$を CPU 時間、経過時間で除し、それぞれ CPU Mflops，Elapsed Mflops で表わす。並列計算の Elapsed Mflops は CPU の数で割った1CPU あたりの Elapsed Mflopsもあわせて使用する。なお、オペレーティング・システムは MVS/XA を用いた。

プログラムは最内側の2重ループ（ベクトル長256以下の乗加算）サブルーチン：DMULAD のみアセンブラー言語を用いて書き、他はすべて VS FORTRAN 第2版、リリース3を用いた。

また、システム構成を縮小し、CPU 1台、中央記憶 128MB、拡張記憶なしのシステムで ページ・データ・セットとのページングに着目するテストも行った。

３．１　拡張記憶機構

プログラムの実行に用いられる部分（ページ）は中央記憶機構上に存在しなくてはならない。参照されたページが中央記憶機構になく、拡張記憶機構にある場合は、オペレーティング・システムによりこのページは中央記憶機構に移動（ページ・イン）される。ただし、中央記憶機構に空きエリアがないときは、LRU (Least Recently Used) アルゴリズムに基づいて、中央記憶域上のページを拡張記憶機構へ移動（ページ・アウト）した後、ページ・インを行うため、２回の移動が必要となる。この移動はページ・データ・セットに対するページングと異なり同期命令により行われるため、入出力割込みは発生せず、高速に処理される。

また、ページ移動に使用される CPU 時間はユーザ・プログラムの CPU 時間にはカウントされない。したがって、ページ移動による遅れは、単独でジョブを実行したときの経過時間と CPU 時間との差によって知ることができる。プログラムの仮想記憶領域のうち中央記憶域から外れた部分を拡張記憶域に溜めているわけであるから、拡張記憶域に対してこのような使い方をしている限り、ページ・データ・セットのバッファー領域と考えられる。ユーザー・プログラムから見た場合、半導体メモリー（キャッシュ）を装備したディスクをページ・データ・セットとして用いた場合と類似するが、オペレーティング・システムからは入出力処理ルーチンを経由せずにページ移動される点に大きな相異がある。オペレーティング・システムの入出力処理は、入出力そのものの実行と完了待ちのほかに、タスク・スイッチやプロトコルの解釈など、CPU に対しても多くの仕事を要求する。これが理由で、はキャッシュ付きページ・データ・セットでは拡張記憶機構のような高速のバッファー領域を実現することはできない。

・問題のサイズとループ順序

行列の大きさをＡ，Ｂ，Ｃともに4500×4500（先導次元 LDA，LDB，LDC は4501）に選ぶと、倍精度のためのプログラムは約 460MB のデータ領域を必要とする。この領域の約半分が拡張記憶域を用いて実行される。ループの順序を次の３通りに選んだ。

(1)　R/S/P///C

(2)　R/P///S/C

(3)　P///S/C

・結　果

結果は(3)が 102.9 CPU Mflops，298.7 (99.57) Elapsed Mflops (1 CPU) で最も速く、他は CPU で約2.5%、Elapsed で４%程遅かった。(1)，(2)のケースのＲループで考慮した利用可能な高速記憶域サイズ（中央記憶域）は150MBとしたため、ｎ＝4500の行列を２つに縦割りしている。この影響で CPU Mflops 値も下がっているものと考えられる。ｎ＝4500では中央記憶域が容量として十分であり、行列Ｂと、行列Ａ，Ｃのセクションを収めきれる。したがって、(1)～(3)の比較の中でＲループを設けた意味はこのサイズの問題では見られない。

・ページ移動のオーバーヘッド

(3)のケースに着目してページ移動に要する時間を考察する。ここではセクショニング・ループが１つ進むごとに行列ＡとＣの２列に１列の割合（256要素は１ページの1/2）で新しいページ（平均4500ページ）が要求されていると考えられる。このページ移動のために 102.9 CPU Mflops が 99.57 Elapsed Mflops に約3.2%遅くなったとすると、

１セクションの計算（経過）時間 …… 610(秒) ÷ [4500 / 256] ＝ 33.9(秒)

１ページの移動時間（インとアウト）…… 33.9(秒)× [3.2(%)/100] ÷ 4500(ページ)＝241(μ秒)

と見積もることができる。この時間（241μ秒）はオペレーティング・システムが１ページを移動するのに要する時間と、並列処理中の他の CPU が中央記憶域へのアクセスをロックされる時間から構成される。前者は、3090のベース・モデルで約75μ秒[6]であるため、ここ（Ｓモデル）ではサイクル・タイムの換算（18.5→15.0 n秒の短縮から75→61μ秒）とベクトル・レジスターのセーブ、リストアの時間(約5μ秒）を加えて66μ秒と仮定する。この仮定にもとずき次式から後者を約27μ秒と見積もることができる。

(241－66×２）÷４ ＝ 27（μ秒）

・ベクトル命令実行中のページ移動

ここで用いた乗加算サブルーチンはキャッシュ容量を考慮したものであるため、ページ移動のためにオペレーティング・システムが走ると、キャッシュの中味が変化して、計算を再開したときのキャッシュのヒット率の低下を招く可能性がある。また、ベクトル命令実行中に発生した割込みは、ベクトル・レジスターの内容をセーブ、リストアする分だけスカラー命令実行中の割込みより作業が多くなる。この影響を測定する目的で、(1)のR/S/P///CのSループに行列Aと行列Cの使用する部分をデータ・タッチしてから DMPYPA を呼び出すようにして実行した。結果は、有意な差としては測定されなかった。この理由は、最内側におかれたサブルーチンが、倍精度で1つのベクトル・レジスターしか使っていないこと、割込みの頻度が十分小さいことのためと考えられる。

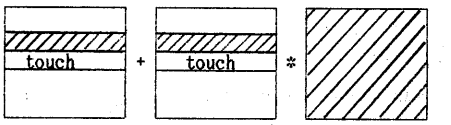
・結　論

この問題では、ページ移動回数が毎秒265回（インとアウト）と少ないため、ユーザー・プログラムからは拡張記憶域を中央記憶域の延長と見做してあまり意識しないで利用できる。

3．2　並列タスクによる先行ページング

数値計算ルーチンでは次のステップで使用するデータがわかっている場合が多い。したがって、要求された時点で初めて下位のメモリー階層から上位のメモリー階層へデータ転送を行う LRU アルゴリズムは、遅いアルゴリズムといえる。データ・タッチ・サブルーチンを並列タスクから使用するとこの転送を早めに処理することができる。S／Cと構成したとき、セクショニング・ループに次のセクションをデータ・タッチするサブ・ルーチンを入れる。

```
do II=1,NII
  I=(II-1)*NZ+1
  NZL=MIN(NZ,L-I+1)
    if(II.LT.NII) then
      call DSPTCH('TOUCH',A(I+NZL,1),NZL,M)
      call DSPTCH('TOUCH',C(I+NZL,1),NZL,N)
    endif
  call DMPYAQ(A(I,1),LDA,·····
  call SYSNCRO
end do
```



中央記憶 128MB、拡張記憶ゼロの3090-180S型（1 CPU）で行列サイズ 2300の問題を解くとページングが発生し、104 CPU Mflops の計算速度が 46 Elapsed Mflops の経過速度に低下する。この問題に対し、プログラム例のように並列タスクで'TOUCH'を呼んぶとページングが非同期に行われ 62 Elapsed Mflops に恢復した。このプログラムは文脈的には奇妙なプログラムと映るかもしれないが、処理は外部ファイルに対する非同期入出力に類似した馴染み深いものである。この方法が効果を発揮するのはメモリー共用型のシステムで（並列タスクが他のタスクの使用するデータを上位の階層に移動可能で）あるためである。また、1 CPU モデルでも効果を発揮するのは、入出力割込みが発生するからである。この例では計算速度とページ入出力との速度差が大きく、あまり実用的な方法とは言いがたい。しかし、FORTRAN 入出力を使用しているプログラムの経過時間を短縮したい時など、簡便な入出力の非同期化としての並列タスクの利用は一考に値するものといえる。

4．おわりに

FORTRAN 言語は、1台のプロセッサが1枚のメモリー空間を対象に、数式を素直にプログラムしてゆくのに都合のよいかたちに作られた。キャッシュという言葉が象徴するように、高速のバッファー領域を用いて計算速度を高める方法は、基本的にはユーザー・プログラムからは見えないかたちでシステムに組入れられてきた。本報告で触れた拡張記憶域も同様の性格をもつものである。プロセッサの計算能力が高められたベクトル計算では、これまであまり気にしないで済んでいた記憶階層間の速度差を考慮して、数式表現に忠実な計算順序を崩してまで、システム

に都合のよい順序に変更する場合がある。本来プログラムからは見えないように作られたシステムのカラクリを考慮するのであるから、表面的な文脈からプログラムの意図は乖離しがちである。

一方並列計算の考え方も FORTRAN が最初にデザインされた時代には存在せず、変化するメモリー内容を複数のCPUが参照する場合など、プログラムがシステムのカラクリを汲み取った文脈が登場する（並列化のためのサブルーチン：DMPYPA で扱った配列 IARG(IP) のように）。

このように並列化も記憶階層の考慮もともに FORTRAN 本来の姿とはそぐわない面を持っているが、本報告では計算速度のためにあえて両者をプログラミングにとり入れる方法を試みた。ここでは階層間のデータ移動量を少なくすることと、割込みによる計算の遅れを少なくすることが要求される。この要求をみたすように作られたプログラムは数式を忠実にトレースする単純なプログラムとは異なった複雑なものとなる。特に先行ページングを試みた例は、オペレーティング・システムの採用している LRU アルゴリズムを前提にした方法で、(しかもデータ・タッチ・サブルーチン：TOUCH の内側のループの代入文などは、コンパイラーの最適化能力が優れると、実行されない可能性もあり）エレガントなプログラムとは言いがたい。プログラミング・エレガンスと高速性能の追及とはウラハラの関係にあると諦めてしまえばそれまでだが、あえて両者の歩み寄る余地を個人的意見として述べ結びにかえさせていただきたい。

今回扱ったプログラムがエレガントでないのは、プログラムの文脈とその意図とが、FORTRAN プログラムからは見てはいけないシステムの性格まで潜ったところで繋がっているからと思われる。これまでシステムは、仮想記憶方式とプログラムからは見えないバッファーとによって、より広いメモリー空間と高速性とを提供してきたが、 FORTRANの言語規格は意外に高速性をとり入れる方向には進化していないように思われる。ここで言語規格の方に、プログラムが明示的に記憶階層を指定する方法をとり入れ、たとえば言語（FORTRAN）にデータのプライオリティを指定する機能を定義して、プログラムから逐次不要となったデータを高位のメモリー階層から排除してゆけるようするとすっきりするかもしれない。これはたとえば先行ページングの例ではエレガントにゆきそうである。このような言語記述の定義は逆に、システムに LRU アルゴリズム以外の記憶階層間のデータ移動を行う機能を装備させるきっかけを与えられるかもしれない。

【参考文献】

1) D.H.Gibson, D.W.Rain and H.F.Walsh, "Engineering and scientific processing on the IBM 3090", IBM Systems Journal Vol 25, No.1, pp36-50 (1986).

2) Bowen Liu and Nelson Strother, "Programming in VS Fortran on the IBM 3090 for Maximum Vector Performance", IEEE COMPUTER, June 1988, pp6-76.

3) IBM System/370 Vector Operations, SA22-7125, IBM Corporation (1986).

4) H.Samukawa, "Programing style on the IBM 3090 Vector Facility considering both performance and flexibility", IBM Systems Journal Vol 27, No.1, pp453-474 (1988).

5) IBM VS FORTRAN Version 2 Programming Guide, SC26-4222, IBM Corporation (1986).

6) S.G.Tucker, "The IBM 3090 System : An overview", IBM Systems Journal Vol 25, No.1, pp4-19 (1986).