仮想記憶における有効実記憶容量推定工具とその設計†

木村 泉†† 高木茂行††† 清水二郎††††

仮想記憶方式の計算機システムにおける有効実記憶容量（システムの満足な動作をさまたげない範囲で利用者プログラムがもち得る最大のワーキングセットの大きさ）を推定するための簡便な工具（Ratfor プログラム）を示す．人工的にワーキングセットを発生させて経過時間を測り，ワーキングセットの大きさと経過時間の関係をグラフにえがく．またこの工具の設計上の問題について論ずる．この工具は限られた割り当て時間の中でなじみのない機種について使われることが多いと考えられるので，Kernighan らのソフトウェア工具の原則からははずれた作りかたが必要である．本文は，便利な工具を提供することとともに，この種の工具の設計の一つの範例を示すことをもねらいとしている．

1. 序

仮想記憶方式の普及によって，プログラムの主記憶占有量の1語2語のちがいはほとんど問題にならなくなったが，大きいプログラムには大きい(実装)主記憶が必要，という一般的関係には厳としてかわりがない．プログラムのワーキングセット(頻繁に使われる部分)の大きさ（動的記憶占有量）がある限度を超えれば，仮想記憶方式のシステムといえどももはや満足には動作しなくなる．すなわち，いたずらに記憶ページの取り込みと追い出しを繰り返して事実上何ら有効な計算をしない状態（スラッシング状態）においちる[1]．

したがって，与えられた仮想記憶システムにおいて利用者プログラムの動的記憶占有量が何程までであればスラッシング状態におちいらないですむかは，利用者にとって大いに知りたいところである．この情報(以下**有効実記憶容量**と呼ぶ）は，たとえば新しい計算機を導入しようとするときなどに大変役立つ．

この有効実記憶容量とは，実装主記憶容量からシステムプログラム(主としてオペレーティングシステム)に食われる部分を差し引いた残りである．その値は，たとえばシステムジェネレーションの際のパラメタの選びかたによって大幅に変わるし，またシステムプログラムと利用者プログラムの間で動的なページの奪い合いが起るようなことも考えられるので，技術資料にもとづいて机上で予測することは，できたにしてもなかなかめんどうである．

そこで本文では，有効実記憶容量の概略値を実験的に求めるための簡便な道具（プログラム）を示すことにする．原理は簡単である．指定された大きさの（仮想）記憶領域の上を「あばれまわる」（まんべんなく参照する）ようなプログラムを走らせることによってその分のワーキングセットを発生させ，実経過時間を測る．記憶領域の大きさを増して行くと，どこかでスラッシング状態が発生して実経過時間が急激にふえはじめる．そのようすをグラフにえがく．グラフの立ち上りばなが有効実記憶容量に対応すると考えられる．スラッシングの発生状況を見るにはページの出し入れの回数を調べることができればより直接的であるが，このように実経過時間に着目する方が簡便である．

なお，この種の小道具では，仕立てかた（パッケージング）がきわめて重要である．そこで本文では，ねらいを単に便利な道具を提供することのみに置かず，道具のよい仕立てかたに関する一つの範例を示すことにも置く．次の第2節では道具そのものについて，第3節では主として仕立てかたについて述べる．

2. プログラム

プログラムを**図1**に示す．使用言語は前処理向き言語 Ratfor[2] であり，しかるべき前処理プログラムで Fortran に変換できるが，なお人手による翻訳の便をはかるため，入出力に関数 getc, putc をもちいず Fortran 流の read 文，write 文をそのままもちい，また相等を示す＝＝を使わず，.eq. とする，などの配慮がしてある．Ratfor については付録で，最小限の説明を与えておく．

このプログラムの先頭には，本来ならばプログラム

† A Tool for Estimating Maximal Working-Set Size in a Virtual Memory System with Discussions on Packaging by IZUMI KIMURA, SHIGEYUKI TAKAGI (Department of Information Science, Tokyo Institute of Technology), and JIRO SHIMIZU (Computer Center, Tokyo Institute of Technology).

†† 東京工業大学理学部情報科学科
††† 現在，日立製作所
†††† 東京工業大学総合情報処理センター

```
define(CR,5)              # nyuuryoku kiban
define(LP,6)              # shutsuryoku kiban
define(WORDSIZE,4)        # 1go no byte suu
define(PAGESIZE,1024)     # 1pe-ji no byte suu
define(STORAGE,1024000)   # kasou ryouiki no gosuu
define(INITIALIZE, )      # abare no maeshori
define(TABSIZE,100)       # kekka no hyou no ookisa

#  COMMON BLOCKS
#  jikken jouken wo oboeru basho
define(AREABLOCK,
          integer from,to,step,times,delta,
                  hajime,owari,kizami,kaisuu,geta;
          common  /area/ from,to,step,times,delta,
                  hajime,owari,kizami,kaisuu,geta)
#  jikken kekka wo tameru basho
define(COSTBLOCK,
          integer atai(2,TABSIZE),komoku,afure;
          common  /cost/ atai,komoku,afure)

#***  main program  ***
  AREABLOCK; COSTBLOCK   # common no kotei
    call yomu
    call hakaru
    call gurafu
    stop
    end

#*** yomu .. data no yomikomi ***
  subroutine yomu
    integer pmax
    AREABLOCK             # han'i no jouhou
    read(CR,10) from,to,step,times,delta
     10 format(5I10)
                          # nyuuryoku no hosei
    hajime = from; owari  = to
      kizami = step; kaisuu = times; geta = delta
    pmax   = (STORAGE*WORDSIZE)/PAGESIZE
    hajime = max0(min0(hajime,pmax),1)
    owari  = max0(hajime,min0(owari,pmax))
    kizami = max0(1,kizami)
    kaisuu = max0(1,kaisuu)
    geta   = max0(geta,0)
    return
    end

#*** hyodai .. jikken jouken no uchidashi ***
  subroutine hyodai
    AREABLOCK
    write(LP,20)
     20 format('1'/' VS TEST          ',
          'HAJIME     OWARI      ',
          'KIZAMI   KAISUU      GETA'/)
    write(LP,30) from,to,step,times,delta
     30 format('   YOMI       ',5I10)
    write(LP,40) hajime,owari,kizami,kaisuu,geta
     40 format('   HOSEI      ',5I10)
    write(LP,50)
     50 format(//'          AREA    TIME(MS)',
               '  I',10('.....I')//)
    return
    end

#*** hakaru .. abareta jikan wo hakaru ***
  subroutine hakaru
    integer toki1,toki2,hirosa
    AREABLOCK             # han'i no jouhou
    COSTBLOCK             # jikan wo tameru basho
    call hyodai
    komoku = 0
    afure = 0
    do hirosa = hajime,owari,kizami $(
      INITIALIZE          # abare no maeshori
      call tokei(toki1)
      call abare(hirosa,kaisuu)
      call tokei(toki2)
      call tameru(hirosa,toki2-toki1)
      $)
    return
    end

#*** abare .. shitei no hirosa de
#                abareru (fibonacci houshiki) ***
  subroutine abare(hirosa,kaisuu)
    integer han i,motmae,mae,ima,hirosa,kaisuu,i
    integer ryoiki(STORAGE); common ryoiki
                          # fibonacci no shokichi
    motmae = 0; mae = 1
                          # gosuu ni naosu
    han i = (hirosa*PAGESIZE)/WORDSIZE
    do i = 1,kaisuu $(
      ima = mod(motmae+mae,han i)
      ryoiki(ima+1) = ima
      motmae = mae; mae = ima
      $)
    return
    end

#*** tameru .. kakatta jikan wo tamete oku ***
  subroutine tameru(hirosa,jikan)
    integer hirosa,jikan
    COSTBLOCK             # jikan wo tameru basho
                          # jikangire no tatsaku
    write(LP,60) hirosa,jikan
     60 format(' ',I10,I12)
    if(komoku .lt. TABSIZE) $(
      komoku = komoku + 1
      atai(1,komoku) = hirosa
      atai(2,komoku) = jikan
      $)
    else if(afure .eq. 0) $(
      afure = 1
      write(LP,70)
       70 format('  KOUMOKU NO KAZU GA OOSUGIRU')
      $)
    return
    end

#*** gurafu .. kakatta jikan no gurafu wo egaku ***
  subroutine gurafu
    integer gyou(101),hoshi,saidai,kosuu,i,j,k
    real tan i
    AREABLOCK             # han'i no jouhou
    COSTBLOCK             # jikan wo tameru basho
    data  gyou  /'I',9*'+',*'*',9*'+',*'*',9*'+',*'*',
                 9*'+',*'*',9*'+',*'*',9*'+',*'*',
                 9*'+',*'*',9*'+',*'*',9*'+',*'*',
                 9*'+',*'*'/
    data  hoshi /'*'/
    call hyodai
    saidai = 0
    do i = 1,komoku
      saidai = max0(saidai,atai(2,i))
    if(saidai .le. 0) $(
      write(LP,80)
       80 format(' JIKAN GA MIJIKAKUTE HAKARENAI')
      $)
    else $(
      tan i = float(saidai) / 100.0
      do i = 1,komoku $(
        kosuu = max0(0,
                  int(float(atai(2,i))/tan i + 0.5))
        k = atai(1,i) + geta
        if(kosuu .eq. 0)
          write(LP,90) k,atai(2,i),hoshi
        else $(
          write(LP,90) k,atai(2,i),
                       (gyou(j),j=1,kosuu),hoshi
           90 format(' ',I10,I12,3X,101A1)
          $)
        $)
      $)
    return
    end

#*** tokei .. jikoku wo watasu ***
  subroutine tokei(toki)
    integer toki
    call clock(toki,2)
    toki = toki * 10
    return
    end
```

図 1 有効実記憶容量推定プログラム

Fig. 1 A tool for estimating the maximal working-set size.

の表題，著者名，作成日付，使いかた，および全体の構成に関する注釈文をつけたいところであったが，紙面節約のためさしひかえた．表題はつけるとすればVS TEST となろう．使いかた，および構成については下で手短かに説明をする．上記の諸点を除けば，プログラムは自己解説的 (self-documenting) であると信ずる．

2.1 使用方法

次の5個の整数値を Fortran の書式 5I10 にしたがって1行に並べ，入力させる．

a. hajime——あばれまわる領域の大きさの初期値．

b. owari——同，最終値．

```
VS TEST          HAJIME     OWARI     KIZAMI    KAISUU      GETA

  YOMI              220       270          1      1000         0
  HOSEI             220       270          1      1000         0

      AREA     TIME(MS)  |.....|.....|.....|.....|.....|.....|.....|.....|.....|.....|
       220          290  |*************************
       221           30  |+**
       222           20  |+*
       223           30  |+**
       224           30  |+**
       225           30  |+**
       226           30  |+**
       227           30  |+**
       228           30  |+**
       229           30  |+**
       230           20  |+*
       231           30  |+**
       232           30  |+**
       233           30  |+**
       234           30  |+**
       235           20  |+*
       236           20  |+*
       237           30  |+**
       238           30  |+**
       239           30  |+**
       240           30  |+**
       241           30  |+**
       242           30  |+**
       243           30  |+**
       244           20  |+*
       245           60  |+*****
       246          180  |+****************
       247          110  |+*********
       248          150  |+*************
       249          170  |+***************
       250          260  |+************************
       251          270  |+*************************
       252          420  |+***************************************
       253          230  |+*********************
       254          240  |+**********************
       255          300  |+****************************
       256          440  |+****************************************
       257          460  |+*****************************************
       258          430  |+****************************************
       259          680  |+***************************************************************
       260          520  |+************************************************
       261          780  |+*************************************************************************
       262          800  |+***************************************************************************
       263          630  |+**********************************************************
       264          880  |+*****************************************************************************
       265          720  |+****************************************************************
       266          980  |+*********************************************************************************
       267          860  |+*******************************************************************************
       268          740  |+*****************************************************************
       269          930  |+**************************************************************************
       270         1070  |+*************************************************************************************
```

図 2 出 力 例

Fig. 2 A sample output.

c. kizami ——同，増分.

d. kaisuu ——領域内参照の繰り返し回数.

e. geta ——プログラム本体の大きさに相当するゲタ.

d 以外はページ単位で与える．1 ページの大きさはたとえば 1024 バイトであるが，機種ごとに変更できる (**2.2** 節)．ゲタの選びかたについては **2.4** 節参照．a—e の値が筋の通らないものであるときは安全をむねとした補正がほどこされる．

出力としてはたとえば**図 2** のようなものが得られる．図 2 からは，たとえば 253 ページ分のワーキングセットを発生させたとき，実経過時間は 230 ms であり，これは実験の全過程で出会った最大の経過時間 (270 ページに対する 1070 ms) の約 21% にあたる，ということがわかる．出力ページの頭には与えられた入力パラメタとその補正値が示されている．今の場合パラメタの補正は起っていない．

なお図 2 のものが出力される前に，同じものが右側のグラフを欠いた形でもう 1 ページ出力されるが，これは実験中に時間打ち切りが起ったときにも有効なデータが得られるようにするための予備的出力である．そのほか，経過時間の測定結果が全部 0 (または負) と出たときには警告

JIKAN GA MIJIKASUGITE HAKARENAI

が，また実験項目数が結果を記録するための表の収容能力を超えたときには警告

KOUMOKU NO KAZU GA OOSUGIRU

が出される．あとの場合，表の容量一杯までの測定結果のみが記録されグラフ化されるが，測定結果の，表の容量を超える部分も，予備的出力の方には打ち出される．

2.2 機種ごとに要する変更

図 1 末尾のサブルーチン tokei は，引数として与えられた整数型の変数の中に実時刻をミリ秒単位で入れて戻るサブルーチンであって，ここには HITAC 8700 用のものが示してある (**3.1** 節参照)．これは機種ごとに用意しなければならない．

図 1 冒頭の “define” にはじまる 7 個の行は機種に

よっては変更する必要があるかも知れないものである．各行のコンマのあとの数字 5, 6, 1024, 1024000, 100, および空所は，# の右側に記されているような意味をもつ．そこを適宜変更する．1024000 は最初に確保しておく仮想記憶領域の語数である．

サブルーチン abare はワーキングセット発生アルゴリズムを与える．ごく常識的に各ページの頭を読みながらループするようなプログラムで間に合うことも多いが，Fortran 処理系の中にはそういうプログラムに出会ったときワーキングセットを小さくするような最適化をするものもあるので，ここではフィボナッチ数列に基づく方法が使ってある．この部分もまれには機種いかんによる変更を要する可能性がある．(abare の中の motmae, mae の初期値は 0, 1 でなく，もっと大きな値をとった方がよかったかも知れない．)

なおプログラム冒頭6行目の空所（前述）は abare に入る前におこないたい初期設定があればそこに指定する，というためのものである．その分は経過時間の計算から除外される．

2.3 全体の構成

全体は二つの名前つき共通領域 area および cost と，それらをとりまくサブルーチン群から組み立てられている．area には実験条件を示すパラメタが入る．area は yomu によって設定され，hyodai, hakaru および gurafu によって読み出される．cost には実験結果が表形式でしまわれる．cost は hakaru によって初期設定され，tameru から値を受け取る．たまった値は gurafu によって利用される．標準 Fortran の文法上の制約を考え，これらの共通領域は主プログラムでも言及してある（common no kotei とあるところ）が，これはたいていの処理系で，なくても間に合う．

ワーキングセットを発生させるための領域は無名共通領域内（abare の中の配列 ryoiki 上）に取ってあるが，これはこうするとページ境界が半端なところにくることを防止できることが多いことによる．

2.4 ゲタについて

geta なるもの（2.1 節，e）が用意してある理由は次の通りである．geta が0のときは，このプログラムの出力グラフ（図2）のたて軸の目盛には無名共通領域上のページ数だけが表示される．しかるに，abare サブルーチンの中心部分は実験中実記憶上になければならないので，これでは完全に正確とはいえない．1～5ページ程度かさあげして考えなければならない．(最悪の場合 abare の中のループが2ページにまたがり，mod がサブルーチンとして別のページに取られ，さらに変数 mae, motmae などが別の2ページにまたがって割り付けられる，というような可能性がある．)また処理系によっては無名共通領域のはじまりがページ境界に合っていないかも知れず，そのときは実記憶占有量がさらにもう1ページふえる．

そこで geta を1ないし6に設定すればこのような事情を出力グラフのたて軸の目盛に反映させることができる．もっとも geta をいくつに取るのがもっとも正当かは記憶地図（メモリマップ）でも作ってみなければわからない．ふつうは上のような事情があることを心得た上で0と取っておけば十分であろう．

3. 検　討

この節では，3.1 で本文のプログラムの測定性能について，3.2 でそのユーザ界面，使い勝手について，3.3 ではプログラム内部の作成の手ぎわについて論ずる．

3.1 測定性能について

図2の出力は東京工業大学総合情報処理センターにかって設置されていた HITAC 8700（主記憶 0.5 M バイト，OS 7 使用）における実測結果に基づいている．図1冒頭の7行も HITAC 8700 に合わせて作られている．実験はこのほか 2, 3 の機種についてもおこなったが，さまざまな考慮からここではこの現役を退いた機種についてのデータを示した．

ただし上記 HITAC 8700 は図1のプログラムの最終版完成以前に撤去されてしまったので，図2は HITAC 以外の機種で，呼ばれるたびに HITAC での測定結果（図1の原型となったある Fortran プログラムによる）を順に返すような，にせの tokei を使って打ち出した．

図2からわかるように，実時間を測定するという簡便な方法でも，おそらくは外部からの擾乱（たとえばコンソールへの定期的なメッセージの出力）によると思われる凹凸はあるにせよ，スラッシング状態の発生とともにグラフが急速に立ち上って行くようすは，はっきり見てとれる．もっとも図2のデータはシステムを独占しておこなった実験によるものであって，そうでない場合はこうは行かない．

図2のグラフの頭の部分（記憶占有量 220 ページの行）に見られるピークは，他の機種でも見られるもの

で，実験の開始時にパラメタ hajime (2.1 節) に相当する数の記憶ページを実記憶に持ち込んでくるための時間と解釈される．一度 hajime ページの実記憶が確保されてしまえば，あとは記憶容量をふやすたびに kizami ページ取り込んでくるだけですむので，グラフはほぼ CPU 時間だけから成る低い値に落ちる．

ワーキングセットをどんどん大きくして行くとグラフはやがて飽和状態に達するが，これは記憶領域を1回参照するごとに必ずページの入れ換えが起る状態に対応するものと解釈される．

グラフはある程度だらだらと立ち上る．これはフィボナッチ数列による abare のアルゴリズムが，事実上等確率で無作為にページを拾い出していると見てよい，ということから一応説明がつく．システムが一つのページを取り込むために他のどのページを追い出したかを知っていて，追い出されたページをねらい打ちするようなアルゴリズムをもし使ったとすれば，グラフは階段関数的に飽和値まで，一気にはね上るはずである．

もっとも以上の解釈，説明はシステムの内部構造に立ち入って確認したものではないので，ここではあくまで一つの可能性として記しておくにとどめる．

なおこのような測定結果は，計算機システムに関する総合判断の一環としておこなうのでなければならないことはもちろんである．オペレーティングシステムの小ささのみに気をとられて，処理速度の検討を忘れてはならない．いずれにせよ本文の方法は簡便な割には敏感であり，測りたい対象を直接測っていることが特徴である，といえよう．

3.2 ユーザ界面について

この種の実験は，たとえばマシンタイムを1時間だけ予約して種々テストをおこなう中の1項目であるようなことが多い．したがってプログラムの処理環境にはあまり注文がつけられない．このことは本文のプログラムの設計上大きな考慮点をなしている．

本文のプログラムを二つにわけて，まず実験結果を対応表の形でファイルに書き出し，次にそのファイルを読んでグラフを編集する，というように構成することも考えられる．Kernighan らのソフトウェア工具の思想[2]にしたがえば，当然そうあらねばならないところである．ここでそういう構成がとられていないのは，はじめての計算機に対面する上でのトラブルをなるべく減らしたいからである．

多くの機種で，ファイルを作るにはジョブ制御文を書く必要がある．その際のまちがいでマシンタイムを浪費することは避けなければならない．ジョブ制御文の代りに TSS のコマンドを使うときも同様である．

Ratfor を使った理由もこれと関連する．こういう道具はできるだけ平凡な言語を使って作るのがよい．その意味で Fortran (あるいはたぶん Cobol) が好適である．だが Fortran で書法的にすっきりしたプログラム[3]を書こうとすると，大変長く複雑になる．こういうプログラムは前もってよく理解しておかないと現場でとっさの処置ができないので，それではこまる．そこで Fortran に直訳可能で，もっとわかりやすい言語として，Ratfor を選んだ．冒頭の define 文の編集，tokei サブルーチンの組み込み，および Ratfor から Forfran への翻訳は事前におこなっておくのを建前とする．入力行は現場で様子を見ながら入れ換えて使うことになろう．入力行のチェックを厳重におこない，親切に警告を出し，時間打ち切りについての手当てをしているのは，すべて現場でのトラブルを減らすことをねらうものである．事前にやっておけることに関する部分は，やや楽に作ってある．

3.3 内部の作りについて

本文のプログラムは書法[3]についても十分注意して作ったつもりであるが，プログラムをあまり長くしないためにいくつかの妥協をした．考慮点を以下に列記する．

a. hajime, owari, ……の補正前の値を from, to, ……であらわす，というのは，hajime, owari などを要素数2の配列とする方がよいが，そうすると do 文の機能とのからみなどが出てめんどうである．

b. hajime, owari, ……は自由形式で読む方がよい．またはせめて書式 5F 10.0 で読みたいところであるが，プログラムがその分複雑化する．

c. 本来ならばモジュール分割をもっとあらわに書きあらわしたいところである．たとえば hakaru の中で komoku と afure に0を入れているのは，いわば cost モジュールの初期設定だから，じかに komoku や afure を引用せずに call cost 0 とでも書く方がよいが，プログラムが長くなる．

d. たとえば冒頭で PAGESIZE を 0 と define するとあとで0による割り算が起る．そのチェックもやりたいが手がまわらない．

e. gurafu の中の format 文は技巧的すぎる．もっとすっきり書けるが長くなる．

f. afure は本質的には真理値であり，それを整数

値であらわしているのは気持ちが悪いが，どうもこの方がわかりやすい．

g. motmae, hyodai など，だいぶ苦しい名前があるが，Fortran の名前の長さに関する制限上やむを得ない．

謝辞 この研究を公表するようすすめてくださった市川惇信教授，実験をおこなう上で便宜をはかってくださった小林啓美教授，前野年紀助教授，および本文の仕上げ作業を手伝ってくださった角田博保氏に感謝する．

参考文献

1) Denning, Peter J.: Virtual Memory, Comput. Surveys, Vol. 2, No. 3, pp. 153–189 (1970).
2) Kernighan, Brian W. and Plauger, P. J.: Software Tools, 338 p, Addison-Wesley, Reading, Mass., (1976).
3) Kernighan, Brian W. and Plauger, P. J.: The Elements of Programming Style, Second Edition, 168 p, McGraw-Hill, New York (1978 and 1974).
(第1版の和訳——木村：プログラム書法，198 p., 共立出版，東京 (1976).)

付　録　Ratfor のまとめ

図1では大文字と小文字は同じ意味をもつと思ってよい．いずれも Fortran の大文字に翻訳される．各行の # 以下は注釈である．define (a, b) は，以下 a という名前が出てきたらそれを文字列 b でおきかえよ，という意味をもつ．b にはセミコロンや改行が入っていてもよい．セミコロンは1行の中に二つ以上の文を入れるのに使用できる．$(, $) は Algol 60 の **begin**, **end** に相当する．do i=k,l は do のあとに適当な文番号をおぎなって，Fortran の DO 文に翻訳される．以上の点を除き，Algol 風（または PL/1 風）の Fortran と思って読めば，図1はたやすく理解できると信ずる．

（昭和53年7月25日受付）
（昭和54年8月23日採録）