オペレーティング・システム 34−2
（１９８７・２・２７）

ＵＮＩＸマシンの
実用的なシステム性能評価法

畠山正行
都立航空高専

コンピュータを道具として使う(Fortran)エンド・ユーザのためのベンチマーク・テスト法及び性能評価法を工夫した。ユーザ・プログラム中にあらわれる基本ルーチンの実行時間を各機種毎にベンチマーク・テストで計測しておく。それと別に対象プログラムの基本ルーチン別の出現総数をカウントし相対的な重みを計算しておく。次に各機種毎に各基本ルーチンの実行時間に対象プログラムの基本ルーチン出現の相対的重みを乗じて、実行予測ｒｅａｌ（ＣＰＵ）時間を出す。この値（又はその逆数としての比性能値）を比較すると、対象プログラムを実際に流したときの各マシンの相対ＣＰＵ時間比が概算できる。必要なら種々重みを変えて試行すればよい。以上のようなテスト及び解析を不完全な形ながら32ビット・エンジニアリング・ワーク・ステーション／ＵＮＩＸの30機種余りについて試みた。その結果、各メーカのワーク・ステーションの処理速度の特徴やその比較が具体的数値で比較され、各ワーク・ステーションの特性の違いから、その相対的重みのかけ方により順位評価が大きく変わること、テスト法や解析法の工夫の仕方を今後改良して行けば、コンピュータの多面的性能とユーザの多様な要求を合理的に評価できるシステムが可能であると結論された。

A Practical Method for Benchmark Test and
Performance Estimation of a Computer System
(in Japanese)

Masayuki HATAKEYAMA
Tokyo Metropolitan College of Aeronautical Engineering
8-53-1, Minamisenju, Arakawa-ku, Tokyo 116, Japan

We propose a practical method for benchmark test and performance estimation for Fortran end users who use the computer as their tools. (1) A benchmark test systrem is performed in various computer system. In this test system, many basic routines which appear in the target user programs are programmed in many small test programs. (2) Frequencies of each basic routine which appears in the target user program are counted. Then, a kind of relative weight of these basic routines in the program can be calculated. (3) A predicted real (CPU) time or specific performance for this user program can be roughly estimated by using $\sum \Delta t_i w_i$. This time shows a relative performance among these tested computer systems for the target program. This method was put in operation to over thirty kinds of engineering workstation/UNIX system, and concluded that since this method can adaptively be applied to users' various requirements, this method is considered to be much more useful than conventional benchmark test methods.

１．はじめに

現代のコンピュータの性能を評価するというのは難しい。その難しさは２つの面の難しさが見出される。それらは

①コンピュータの高性能化・高機能化・種々の分野への応用のため、その仕組はハード・ソフト両面において日を追って複雑になりつつあり、又、各メーカがそれぞれ独自の目標を作り開発しつつあり、それらすべての機能や性能値をすべて客観的に評価できる尺度(measure)を作るのは困難である。

②ユーザの方の性能・機能に対する要求仕様や用途についても従来全く用いられなかった分野への応用時の各機種の性能を知りたいなど個別用途への性能評価を要求するなど、多様化してきている。用途に適したベンチマーク・テストや性能解析は困難である。

このように高級・複雑化したコンピュータ・システムと多様化したユーザの要求・用途に対し、適切なベンチマーク・テスト手法及び性能評価（解析）法は確立されていないように思う。少なくともエンド・ユーザによく知られている手法には適切なものがない。従来からある手法では全く不足という他はない。プロセッサの内部処理性能を表すＭＩＰＳ値はエンド・ユーザが独自に実施するわけにはいかず、又どのような内部仮定（重みなど）をおいた方式で計算したかは不明の場合が殆どであり、又、システムとして常にどのモデルや構成においてもその値が保証されているわけではなかろう。更には１instruction がどのコンピュータでも共通というわけではない。従って信頼性は低い。Whetstone ベンチマーク、Dhrystone ベンチマーク、LINPACK ソフトウェアを用いた MFLOPS 値、バイト・ベンチマークなどが最近よく用いられるようになった。確かにこれらユーザ要求・用途に対し、特定の機能・性能を測るには確かに有効である。これらの欠点は一部の機能・性能しか対象にならないという点にある。又、ここでは専門家にしか必要でないテストや評価法やコンピュータ・システムの内部構造（ソフト、ハード）の一部のみを測るテストや評価法はエンド・ユーザの立場からすると論外になる。

エンド・ユーザの立場からいえば、要するに自分の必要な処理をするのにどの機種が最も適しているかのみを知りたいわけである。しかし厳密に評価したいのなら実際に仕事に本格的に用いてみるしかなく、全ユーザの要求にメーカ（又はベンダー）が応じ切れる筈はない。又、将来的な可能性として要求仕様以外の性能も参考にしたいとなれば話は更に困難である。従って、例えばコンピュータ導入の際の比較評価には充分耐え得るが、例えば10%～30%程度の性能評価誤差は許せる、といった程度の簡易かつ実用的なテスト法や評価法ができれば、ユーザ側としては満足すべきであろう。その場合に満たすべきいくつかの条件を述べると、

１．コンピュータ・システムの内部構造などには触れず、コンピュータ・システム内部は black box とし、ユーザ・プログラム（テスト・プログラム）の処理所要時間（real, usr, sys）の結果のみに注目する。

２．ユーザの個別用途毎にテストを実施しなくてはならない方法であってはならない。あるシステムの全てのモデルについて１回のみ計測しておけば一応可となるようでなくてはならない。従って詳細な構成（ハード、ソフト）による性能差があるものについても一種のテストで代表させることになる。

３．ユーザ側は自己のworkについての各機種の性能（概算処理時間）を評価するため、個々のworkそれぞれについてユーザ・プログラムの分解・解析・再構成等の作業及びその結果を用いての性能概算作業を行う必要がある。つまり個別用途への性能評価はユーザ自身の手で各自実施するわけである。以上の議論からしてエンド・ユーザ側から提案するベンチマーク・テスト法として次のような手法が適切であると考える。

１．ユーザ・プログラム中に現れる全ての基本ルーチンの実行時間を全ての機種にわたり計測しておく。

２．各ユーザ・プログラム中において各基本ルーチンがそれぞれどの位の割合（実行時間の相対的重み）で含まれているかを計測又は計算する。

３．全マシンについて基本ルーチンの実行時間(real time)に対しユーザ要求の重みをかけ、各マシンでの実行予測ＣＰＵ時間を算出・比較する。（又は逆数をとって比性能を比較してもよい）。

２．ベンチマーク・テスト実施要領

基本ルーチンの決定とテスト・プログラムの作成

基本ルーチンを何にするかは原理的には大変困難である。又、すべてをカバーするだけの種類の基本ルーチン群を揃えるのは困難であり実用上も煩雑である。従って、簡易さと精度のバランスを考えていくつかの種類の基本ルーチンを設定すべきである。更にテスト・プログラムを作成した段階で評価精度が落ちることも覚悟すべきである。今回作成し実施したテスト・プ

ログラムを１ａ～１ｈに示す。テスト１は四則演算速度、テスト２は関数呼び出しとその和、テスト３はサブルーチン・コール、テスト４は４重の do ループ、テスト５は再起呼び出しで33の階乗の計算、テスト６は複雑な式計算、テスト７は配列への値代入、テスト８は連立一次代数方程式、をそれぞれテスト内容としたプログラムである。テスト７においては nd の値は 110、510、710、……と次々に変えたテスト（ラージ・メモリー・テスト、仮想記憶テスト）があり、又、テスト８においてはnd＝80、300、500の三種がテストされる。これだけのテスト・プログラムのみではまだ多くのテストすべき項目が残されているのは明白である。しかし今回の実テストは以上のプログラムを用いて行った。

```
test1
      program arith
      a=1.0
      b=2.0
      c=-1.0
      d=-2.0
      e=-3.0
      k=2000

      do 20 m=1,k
      do 10 n=1,k
10  const=a+b*c-d/(e+a)
20  continue

      print *,'1. arith. ite.=
      stop
      end
```

図1a　テスト1

```
test2
      program sincal
      i=100000

      do 10 n=1,i
      e=float(n)
      si=si+sin(e)
10    continue

      print *,'2. sincal. ite.=
      stop
      end
```

図1b　テスト2

```
test3
      program subcal
      n=200000
      do 10 i=1,n
10    call subl(i)
      print *,'3.  subcal. ite=',i
      stop
      end

      subroutine subl(i)
      i=i+1
      return
      end
```

図1c　テスト3

```
test4
      program pow33
      k=30
      do 400 m=1,k
      do 300 j=1,k
        do 200 n=1,33
          ans=1.0
          do 100 i=1,n
            ans=ans*float(i)
100       continue
200     continue
300   continue
400   continue
      print *,'4. 33!. ite',k,
      stop
      end
```

図1d　テスト4

```
test5
      program rec33
      k=500
      do 200 j=1,k
             do 100 n=1,33
                    ans=1.0
                    i=1
                    call subl(ans,i,n)
100          continue
200   continue
      print *,'5.  recur!. ite=',k,'   va
      stop
      end

      subroutine subl(ans,i,n)
      if (i.gt.n) then
             goto 10
      else
             ans=ans*float(i)
      i=i+1
      call subl(ans,i,n)
      end if
10    return
      end
```

図1e　テスト5

```
      test7
         program ran
         integer box(710,710)
         nd=710 - 1

         do 10 i=1,nd
                do 20 j=1,nd
                       box(i,j)=j
20              continue
10       continue

         n=50000
         r1=2.0
         do 30 m=1,n
                i=int(rnd(r1))*nd+1
                j=int(rnd(r1))*nd+1
                k=box(i,j)
                box(i,j)=box(i, nd+1-j)
                box(i, nd+1-j)=k
30       continue

         print *,'vm test.   nd=',nd
         stop
         end

         function rnd(r1)
         ix=int(r1)
         ix=ix*48828125
         if (ix) 100,200,200
100      ix=(ix+2147483647)+1
200      rnd=float(ix)*0.4656613e-9
         r1=float(ix)
         return
         end
```

図1g　テスト7

```
test8
   program ren80
   dimension a(80,81), ans(80)
   n=80

   n1=n
   n2=n1+1
   do 10 j=1, n
   do 10 i=1, n
10 a(i,j)=n+1-i
   do 20 j=1, n
   do 20 i=1, n
20 a(j,i)=a(i,j)
   do 30 i=1, n
30 a(i,n2)=1.0
   a(n1,n2)=1.0

   call lqgs(a,n1,n2,n,ans,isw)
   print *,isw
   print *,'test8. ren80 ite.=',n,
  =        '  answer=',ans(n)
   stop
   end

   subroutine lqgs(a,n1,n2,n,ans,asw)
   dimension a(n1,n2), ans(n)
   klmt=1000
   epsl=1.0e-6
   do 1 i=1, n
 1 ans(i)=0.0
   do 5 k=1,klmt
   e=0.0
   do 4 i=1, n
   delta=a(i,n+1)
   do 3 j=1, n
   delta=delta-a(i,j)*ans(j)
 3 continue

   delta=delta/a(i,i)
   ans(i)=ans(i)+delta
 4 e=e+abs(delta)

   if (e-epsi) 6, 6, 5
 5 continue

   isw=1
   return
 6 isw=0
   return
   end
```

図1h　テスト8

```
      test6
         program power
         a=1
         b=1.001
         c=-1
         d=-1.001
         n=50000

         do 10 i=1,n
         e=(((a+b)*c/d+(a**2-b**3+c**4)*d**4)**5)/a**6*b**7/c**8
        &  /d**9*a**1024-(((a+b**9)/d**2)**7)/(a**4+d**7)
        &  +sqrt(abs(sin(tan(a)/exp(cos(sin(b))))))**128
         f=(sinh(a)*cosh(b)/tanh(-c))**log10(abs(a)**1024)
        &  +sin(cos(tan(c)))**cos(exp(sin(sqrt(abs(atan(sinh(d)))))))
         g=real(int(exp(sin(cos(tan(a))))))**(max(a,b,c,d)**128)**214+abs
        &  (asin(a/b)/acos(c/d))*exp(sqrt(abs(log(abs(exp(max(a,b,c,d)))))))
       10  continue

         print*,'6. power ite.=',n,'  answer e f g=',e,f,g
         stop
         end
```

図1f　テスト6

ベンチマーク・テスト仕様

このベンチマーク・テストは一部を除いて各メーカー、ベンダー、大学や研究所、その他個人などに依頼して実施していただいた。従って口頭での説明なしにテストが実施できるように詳細な仕様書を作り、送って実施してもらい、一定のフォーマットでの回答をいただいて整理した。

テストの実施機種と構成

テスト実施機種としては32ビットのエンジニアリング・ワーク・ステーションを中心とし、16ビット・マシンや汎用中大型、スーパー・ミニコンの一部などを含めた。表1にテスト機種名を揚げてある。ＯＳは大半がＵＮＩＸ又はそれに近いシステムである。各機種の構成については一律に揃えるわけには行かないのでバラバラである。しかしほとんどの機種がメモリー４ＭＢ以上、ＦＰＰボード付、ディスク容量数十ＭＢ以上である。構成上の不利は評価の時に考慮すべきであろう。

Apollo DN560	Pyramid 98xe
Sun-3/160M	Concept 32/9705
HP9000/320	VP-1
MightyFrame	VAX-11/780
MC5400	MicroVAX II
Ustation/E20	VAX 8600
HITAC E-7300	MELCOM70 MX/3000
vsx-800	vsx-758
UNIBOX/3001	Sun-2/120
IX-7	Apollo DN320
DMI-UX/20	MC5500P
TOSBAC UX-400	PC-9801 VM2
Multimax	NEC ACOS 350
Balance 8000	FACOM S-3500
MPZ-2060	HITAC M-280H
Ridge 32	HITAC M-680H
HP9000/550	FPS-264
Apollo DN460	

表1　テスト実施機種

３．性能評価解析

評価と解析は次の手順を踏んで行う。

１．テスト結果の一覧表を作成

２．ユーザ・プログラムの分析から各テストへの相対的重みを決定する。

３．重みを乗じた換算 real time の一覧表を作成する。

４．全テストの real time 合計及び平均値を各機種毎に作成する。次に除くべきデータ、実行不可能テスト、未測定データを考慮した上で比性能値をもとめる。今回用いた比性能値をもとめる式は

$$P = \frac{(\text{重みつきテスト実施数})^2 \times 100}{(\text{評価時間合計}) \times (\text{テスト実施数})} \quad \cdots(1)$$

５．この比性能は対象となるユーザ・プログラムを各機種でテストしたと仮定したときの比性能である。大きい程性能がよい。勿論比性能であるから各機種間の比較は必ず「比」で行わないと意味がなくなる。表２～表４に　結果の一部を示す。 スペースの関係で代表的な６機種のみを示してある。 全機種のデータは参考文献２　p,116～p,123にあるので参照されたい。表２は結果そのままの一覧表、表３は換算表、表４は性能比較表である。　　表３においてはコンパイル、基本処理、大メモリー、の各テストにおいてはＭｉｃｒｏＶＡＸ－Ⅱを、ＶＭ（仮想記憶）テストにおいてはＳｕｎ-３を基準にし、基準機種が各テストを１００秒間かかった時の他機種の値の相当時間を出した。次に各テストの重みとしてコンパイル３、基準処理６、大メモリー６、ＶＭテスト３と決め比性能を求めた。ある意味で数値計算プログラム、それも大型のものに対するオール・ラウンドな性能を比較するという場合の重みに相当している。　本稿ではその目的でないので各機種の性能比較の議論は省略する。

表２　テスト結果（real time）

name	Compile	test1	test2	test3	test4	test5	test6	test7 (110)	test7 (510)	test7 (710)	test8 (80)	test8 (300)	test8 (500)	VMtest start	VMtest end
Apollo DN560	2.9	31.6	5.6	3.5	10.0	9.8	195	9.6	24.1	27.7	26.8	479	1435	153	10000
														1300※	1710※
Sun-3/160M	13.8	(1.6)	4.1	1.2	2.9	7.8	138	8.1	8.9	9.6	16.6	291	870	10.5	10000
														800※	890※
MicroVAX II	13.7	109	27.9	7.8	8.3	14.3	612	10.5	14.1	19.1	31.1	2205	6100	24.0	800
														860※	2000※
Ustation/E20	9.4	135	4.7	1.1	8.2	×	138	6.5	7.6	9.0	29.4	2202	6112	13.5	10000
														1250※	1630※
HITAC E-7300	5.0	136	4.5	0.62	7.2	6.4	131	5.8	7.3	9.1	29.1	2269	6427	11.5	10000
														1200※	1800※
vsx-800	7.1	161	5.1	1.1	9.4	6.9	159	6.9	6.7	7.8	15.4	265	792	9.5	10000
														830※	995※

４．考察と今後の改良案

表２から表４の数値群の特徴を述べるだけで、６機種に特定のwork program（重みつき）　を走らせた場合の性能の優劣を論じることができる。勿論、このベンチマーク・テストはあくまで代表的なルーチンをテスト・ランさせてその結果を以て本来のユーザ・プログラムに対する処理性能を単に予測しているに

表3 テスト結果の換算表

name	Compile	test1	test2	test3	test4	test5	test6	test7 (110)	test7 (510)	test7 (710)	test8 (80)	test8 (300)	test8 (500)	VMtest 指数 1	指数 2	指数 3
Apollo DN560	21.2	29.0	20.1	44.9	120	68.5	31.9	91.4	171	250	86.2	21.7	23.5	12.1	4.91	71.5
Sun-3/160M	101	(1.47)	14.7	15.4	34.9	54.5	22.5	77.1	63.1	50.3	53.4	13.2	14.3	100	100	100
MicroVAX II	100	100	100	100	100	100	100	100	100	100	100	100	100	0.362	0.801	1.83
Ustation/E20	68.6	124	16.8	14.1	98.8	×	22.5	61.9	53.9	47.1	94.5	99.9	100	13.9	56.6	72.8
HITAC E-7300	36.5	125	16.1	7.95	86.7	44.8	21.4	55.2	51.8	47.6	93.6	103	105	8.45	50.2	55.0
vsx-800	51.8	148	18.3	14.1	113	48.3	26.0	65.7	47.5	40.8	49.5	12.0	13.0	50.5	95.2	86.0

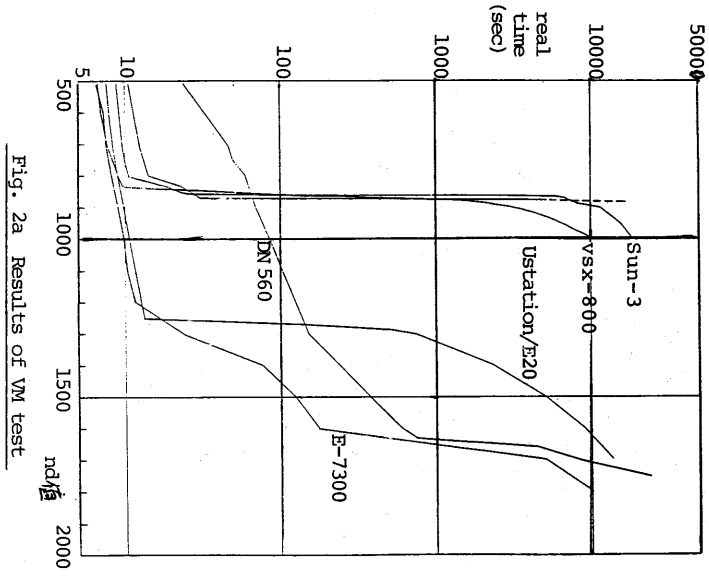

Fig. 2a Results of VM test

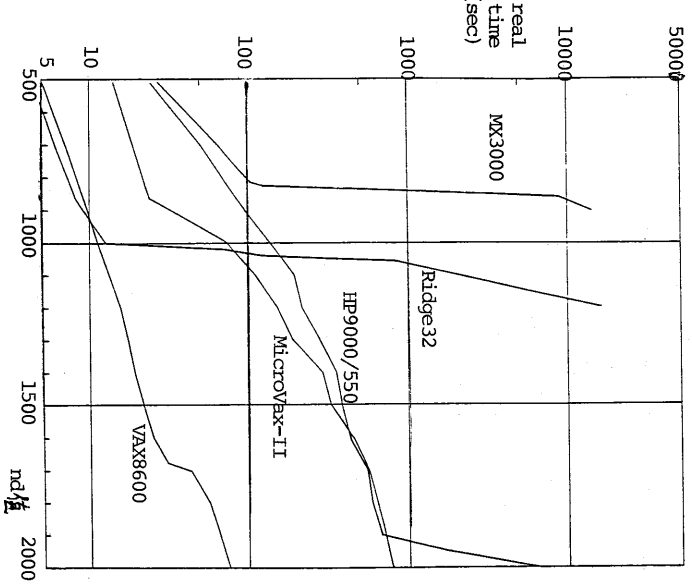

Fig.2b Results of VM test

表４　性能比較表

name	コンパイル	基本処理	大メモリ	ＶＭ	平均	比性能
Apollo DN560	63.5	315	644	88.6	61.7	1.62
Sun-3/160M	302	142	271	300	59.7	1.67
MicroVAX II	300	600	600	2.99	83.5	1.20
Ustation/E20	206	276	458	143	63.7	1.48
HITAC E-7300	109	302	456	114	54.5	1.83
vsx-800	155	368	229	232	54.7	1.83

過ぎず、その点限定的に受け取らなければならない。しかしながら、現状のテスト・プログラムだけではともかく、頻出するかなりの基本ルーチンをテスト・プログラムに加えたシステムを作り表２から表４及び式（１）を用いる手法を使えば、各エンドユーザ向きの性能評価・解析が各エンドユーザ自身の手で充分簡単に出来ると言えるのではなかろうか？しかもかなりの誤差を伴うとはいえ従来のベンチマーク・テストよりは数段はユーザ・プログラムに近い形のシミュレーションを行っているのと同等の手法と言えないであろうか？本手法の特長を箇条書に挙げると、

１．各ユーザは自己の処理内容に応じて個々の解析を行い機種別の評価を行える。従って、その評価による順位付は個々のユーザの用途にとってのみ相関性が高く有意のものとなる。

２．各ユーザは各基本ルーチンの実行時間への重みを変える事により、種々の処理内容に関する実行処理性能を評価する事ができる。

以上の手法には当然多くの欠点や実現上の困難が見出せる。しかしエンド・ユーザの要求に最も近い機種は何かを知りたいという立場に立ったテストを目指すならば、少なくともこの方法は従来あるベンチマーク・テストよりもはるかにユーザにとって便利で信頼性のあるテストになると考えられる。

本手法の欠点を挙げれば次のようになろう。

１．現在のテスト・プログラムでは種類が少なすぎる。

２．あまり高精度の評価は期待できない。

３．解析手順が煩雑である。

これらのうち１．と２．はいわばトレード・オフであると言える。どの程度の内容と種類数を持てばよいかは問題であるが、筆者の経験から推すると現在のテスト１～テスト８までの３倍くらいの内容と種類を持ち、実プログラムをながした時の比性能で10%の精度を保てるようにするのは困難ではないと考える。ベンチマーク・テストそのものにしても、シエル・ファイルを使ってすべて自動的にテスト及び結果の出力を得るようにすることは困難ではない。３．については、本稿冒頭で述べたように、これだけコンピュータ・システムとなれば、それ相応の手続きは必要となって当然であると考える。特に自己のプログラム処理内容別分類はユーザ自身にとっても有益な筈である。

特長は確かにうなずけても、従来のベンチマークより手順が（合理的理由はあっても）面倒な事は確かである。そこで最も効率的実現法を述べたい。

１．適当な機関で基本ルーチン内容と言語を決めテスト・プログラムのシステムを組み実施手順書とともに無料で希望者（メーカ）に配布する。

２．メーカ（又はベンダー）は新機種を出す度に１回だけベンチマーク・テストを実施し、データを機関に提供する。

３．その機関でデータをデータ・ベース化し、適切な価格で販売する。ただしその後有意なデータを提供されれば料金は返却する。こうしてデータを蓄積していく。

以上述べた方法等によりエンド・ユーザにとって有効な性能評価原データが提供されるようにする事を期待したい。

参考文献
(1)畠山政行，「ＤＢＳＳのシステム理論とWorkstation／ＵＮＩＸ， 東京都立航空工業高等専門学校，昭和６０年度研究紀要第２３号，pp.85-136 (1986).
(2)日経データ・プロ編「３２ビット・マイクロプロセサの全容」， pp.100-128，昭和６１年１２月。

（本稿は上記の文献の考え方と手法とを発展させたものである。）