Univerza v Ljubljani
Institut „Jožef Stefan", Ljubljana, Slovenija

IJS Delovno poročilo
DP-6930

Benchmarks I:
Small Memory

R. Krivec
Department of Theoretical Physics

Introduction

(Benchmarks file updated: 93/06/09; additions 94/02/07)

The purpose of this Report is to present benchmarks done on two classes of programs seldom encountered in other benchmarks, *i.e.*, realistic programs with different degrees of vectorizability, and matrix multiplication programs. This combination gives interesting insight into relative efficiencies of different types of computers.

Tests were done using

- two realistic programs (M2MK, with a small degree of vectorizability, and EWF2D, with a large degree of vectorizability;
- various matrix multiplication programs, including those from Veclib;

The benchmarks were performed on the following machines:

- Convex C220 (50 MFLOPS per processor);
- Convex C3860 (120 MFLOPS per processor);
- Vax 8650 FPA (about 1 MFLOPS; this "classic" machine serves as a reference for speedups);
- IBM ES/9021 (138 MFLOPS per processor);
- some HP, SG workstations (of the order of 20 MFLOPS).

Different program sizes were tested. Relative speedups may depend strongly on memory requirements. No workstation with over 100 MB memory was available, therefore the most interesting relative tests, workstations *versus* "supers" at large memory requirements, were unfortunately not possible.

In addition to benchmarks comparing different computers, there are some optimization benchmarks using versions of the same program.

Chapter 1 compares a vector computer to a superscalar workstation, using realistic programs with moderate memory requirements. Chapter 2 compares four different computers using realistic programs with small sizes. Chapter 3 compares matrix multiplications on many computers.

Technical notes:

If not otherwise specified, program EWF2D uses routine RRPSK 1.0 of March, 1992, which is optimized for Convex.

Remarks with unresolved references are purely technical; details are available with the author. Care was taken to document all relevant data to make the tests repeatable.

1. Convex C220, SG Crimson

92/08/11-

Time returned on SG by `etime` agrees with the time returned by command `time` (where available). Seemingly `etime` and `time` return "elapsed time", necessitating careful comparisons with Convex where CPU time is returned by these programs. Uncertainty below is believed to be unimprotant.

Program lengths:
M2MK: 28 MB;
EWF2D: 42 MB.

Table 1-I. CPU time in seconds. R is a relative efficiency, obtained as $const/((CPU\ time) \times (declared\ MFLOPS))$. It is only applicable to comparing speeds of a given program on different computers.

Computer (MFLOPS declared)	M2MK CPU time (opt)	M2MK R	EWF2D CPU time (opt)	EWF2D R
Convex C220 (50)	429 (-O2)	4.7	1874 (-O2)	1.1
SG-Crimson (16)	321 (-O0)	19	7374 (-O2) *1	0.85
	225 (-O1)	28	7041 (-O2) *2	0.89
	168 (-O2)	37	7933 (-O3) *3	0.79

*1 `time` command output not verified (background process); probably "elapsed time" (another application was running).
*2 Foreground process, overnight on empty machine. `time` not known (misdirected `stdout`).
*3 `etime` and `time` give equal times; machine probably empty.

Summary

Relative efficiency R shows that the superscalar architecture of SG RISC processor is much more efficient for "general" unvectorizable programs than the Convex vector architecture. On the other hand, vectorizable programs are able to exploit the potentials of both architectures in an approximately equal measure, Convex being slightly better off. In other words, vectorizable *realistic* programs run at speeds which are very accurately proportional to the respective processor MFLOPS specifications. (This is not the case for matrix multiplication, see Ch. 3.) Note: source programs were identical on both machines but optimized (apart from being properly written in the source) by the respective compilers.

Remark: the final results differ:

```
orig/ewf2d.2ff: DET:     -1.4160351121850609 10**120  (Convex -O2)
sg/ewf2d2.2ff:  DET:     -2.7017608216300068 10**120  (SG -O2)
```

The reasons are: (i) the result is an unphysical quantity, a function value the zero of which the program is to calculate, the zero being typically many orders of magnitude less sensitive to parameter changes than the function itself; (ii) unphysical parameters to make runs short allow random truncation of certain series due to different number ranges; (iii) different orders of execution of arithmetic statements.
Different levels of optimization on Convex have been tested to have similar effects on the above numbers. Also, footnote 6 of Table 2-I shows that Veclib results on IBM and Convex are more similar suggesting (iii) may be the main cause of discrepancies. This kind of instability may also affect operation counts (if (ii) is the main cause of instability) but since most of the time is consumed in matrix manipulations it is assumed that different operation counts have negligible effect on the above CPU times.

M2MK and EWF2D input files

```
92/07/17 mu-d-t SG performance test
T                            LSTORE (UNFORMATTED OUTPUT)
   0                         INUSEL
TT                           PCFC, PVC (PRINT F, V POWER EXPANSIONS)
                 1.0           1.49760154        0.0563329438    M1/2/3 (-1: INF)
  16   0   F                 MUU, IPOS, PSYM
   1  42  -1  -1  40         IPFL, IPFU, IPVL, IPVU, IPWU
   5                                                                 NFPAR
   1     -1.0                -1.0                 1.0                FPAR
   2      0.0                 0.0                -0.290000000000     FPAR
   3      1.0                 1.0                 0.0                FPAR
   4     36.000000000000     38.000000000000     49.000000000000     FPAR
   5      1.0                 1.0                 0.5                FPAR
   5                         MODF (INTERPRETATION OF FPAR BY CFC...)
   1                                                                 NVPAR
   1               -1.0                -1.0                 1.0     VPAR
   0                         MODV (INTERPRETATION OF VPAR BY VC...)
  16  16                                    (NGOA(I), I = 1,2)
   1   1                                    (NSUBA(I), I = 1,2)
  16  16  16  16  16  16  16  16  16  16    (NGOL(I,1), I = 1,10)
   1   1   1   1   1   1   1   1   1   1    (NSUBL(I,1), I = 1,10)
  16  16  16  16  16  16  16  16  16  16    (NGOL(I,2), I = 1,10)
   1   1   1   1   1   1   1   1   1   1    (NSUBL(I,2), I = 1,10)
   0.100   0.030                            DAS, DLS
TT              POMEGA, POTHER (WORKING RUNS: TT)
TFFFF           PCON, PTEST, PTEST1, PRES (WORKING RUNS: TFFF), PCONTR
FFFFFFF         PT00, PT10, PRES0, PTG, PTGA, PTGL, PTJ (WRKNG RUNS: F...F)
   3            NSPR (MAX. NO. ROWS/COLUMS OF MATRICES TO BE PRINTED)

92/07/17 SG performance test
FFFFFFFF        LTDD,LTDDR,LTVICD,LTRRPS,LPRBIN,LPRZOM,LTCPU,LTCRU
FFFFFF          LPRCV0, LPRXX, LPRCV, LPRCVM, LPRMAT, LPRWF
TFT             LTZE, LTZEIF, LXLAST
   8            IINTPR
  20 200        LM, IPROG
  40 153   0                          IPWUU, NSU, INUSEL
  20  16                              NM, JMAX
    0.50    0.50  9999.0              BC0, BC1, RHOUW
0.0303          0.0306                A, B
  +1.0000000D-14  1.00000000D+00  5.00000000D+00    EPSP, TOL, DZU
    1.25   15.00  100.00              ZM, ZMATCH, ZTOP
   0                                  NHH
   2                                  LMOD
.030            .0301          .1                      ENXL, ENXU, ENXS
   1                                  IPIO (0: M1M-, 1: M2M-)
   0.000   0.000   0   0              Z0MTTT, Z0MTTD, IPRTTT, IPCTTT
```

2. Vax 8650, HP 730, IBM ES/9021, Convex C220

(Updated: 92/12/10)

Program lengths are very small:
M2MK: 3.9 MB
EWF2D: 5.8 MB

Table 2-I. R is defined in Table 1-I.

Computer (MFLOPS declared)	Directory		M2MK CPU time	R	EWF2D CPU time	R	Program EWF2D Result (det(0.03))
VAX 8650FPA	[.bm.m2]	*1	168				(overflow)
		*2	168		4071		-2.1778177357242736(49)
HP 730 (32 MB) (KIBK)		*3	29	1.5	291	1.5	-2.2541707156967628(49)
(23)							
IBM ES/9021		*4	23	0.3	198	0.4	-2.2853810805753951(49)
(138)		*5			145	0.5	4.0984477217203359(48)
C220	cfhh/bm/m2		61	0.3	133	1.5	-1.7761131772796315(49)
(50)		*6			133	1.5	4.2701324946203938(48)
	cfhh/bm/m3	*7			140	1.4	-1.7761131772796315(49)
	cfhh/bm/m5	*8			138	1.4	-1.7761131772796315(49)

*1 Default floating point representation.
*2 G_floating, CDX... routines unchanged (as in *1). M2MK output agrees well with *1. Speed ratio C220:Vax is 4071/133 = 31. Results disagree - see discussion in Chapter 1.
*3 Speed ratio C220:HP is 291/133 = 2.2.
*4 FORTVS2 (OPT(3) VECTOR(REPORT(XLIST))), with unchanged RRPSK 1.0. If the loops 72 and 70 are exchanged (K and I) the time is 202 sec. which confirms that IBM prefers K loop inside as it is in RRPSK 1.0 (was written for Convex which interchanges loops anyway).
*5 RRPSK 1.0-DGEMM (92/12/04). This is the RRPSK 1.0 routine with loops 70-72-74 replaced by a call to DGEMM. The result differs and correctness of the call is not tested.
*6 As *5. Note that C220 and IBM Veclib results are similar while different from the respective non-Veclib results. (92/12/07.)
*7 92/12/10. DASUM, DAXPY, etc., all called from Veclib; sources not used. (Names changed DASUMD → DASUM etc.). Slower! (RRPSK 1.0 used, DGEMM NOT called).
*8 92/12/22. DDS 1.4 (hand-distributed loops), RRPSK 1.0. DASUM etc. sources used.

Summary

HP and Convex exhibit behavior seen in Chapter 1. IBM compiler stands out in the sense that it does not optimize a given program automatically as efficient as either HP, SG or Convex. IBM is actually slower than Convex although it should be 1.4 times faster. (Some insight into this is offered by Chapter 3.) Also, footnote 7 shows that on Convex, using Veclib routines tends to be even slightly slower than using source code for the same routines. (Sources were obtained from M. Haftel, NRL, and were used on a Cray X-MP. They are optimized in a general way, not for a specific machine.)

(*Technical remark*: `cfhh/bm/m1`: smaller dimensions because of HP730; `cfhh/bm/m2`: as `.../m1`, but changes for IBM (`FUNCTION`, `1H`).)

M2MK and EWF2D input files

```
92/11/23 mu-d-t HP 730 (32) performance test
T                            LSTORE (UNFORMATTED OUTPUT)
   0                         INUSEL
TT                           PCFC, PVC (PRINT F, V POWER EXPANSIONS)
                1.0            1.49760154        0.0563329438    M1/2/3 (-1: INF)
  10   0   F                 MUU, IPOS, PSYM
   1  22  -1  -1  20         IPFL, IPFU, IPVL, IPVU, IPWU
   5                                                                 NFPAR
   1     -1.0               -1.0                 1.0                 FPAR
   2      0.0                0.0                -0.290000000000      FPAR
   3      1.0                1.0                 0.0                 FPAR
   4     36.000000000000    38.000000000000     49.000000000000      FPAR
   5      1.0                1.0                 0.5                 FPAR
   5                         MODF (INTERPRETATION OF FPAR BY CFC...)
   1                                                                 NVPAR
   1               -1.0                -1.0                 1.0      VPAR
   0                         MODV (INTERPRETATION OF VPAR BY VC...)
  16  16                                    (NGOA(I), I = 1,2)
   1   1                                    (NSUBA(I), I = 1,2)
  16  16  16  16  16  16  16  16  16  16    (NGOL(I,1), I = 1,10)
   1   1   1   1   1   1   1   1   1   1    (NSUBL(I,1), I = 1,10)
  16  16  16  16  16  16  16  16  16  16    (NGOL(I,2), I = 1,10)
   1   1   1   1   1   1   1   1   1   1    (NSUBL(I,2), I = 1,10)
   0.100   0.030                            DAS, DLS
TT              POMEGA, POTHER (WORKING RUNS: TT)
TFFFF           PCON, PTEST, PTEST1, PRES (WORKING RUNS: TFFF), PCONTR
FFFFFFF         PT00, PT10, PRES0, PTG, PTGA, PTGL, PTJ (WRKNG RUNS: F...F)
   3            NSPR (MAX. NO. ROWS/COLUMS OF MATRICES TO BE PRINTED)
```

```
92/11/23 HP 730 (32) performance test
FFFFFFFF        LTDD,LTDDR,LTVICD,LTRRPS,LPRBIN,LPRZOM,LTCPU,LTCRU
FFFFFF          LPRCV0, LPRXX, LPRCV, LPRCVM, LPRMAT, LPRWF
TFT             LTZE, LTZEIF, LXLAST
   8            IINTPR
  20 200        LM, IPROG
  20  66   0                       IPWUU, NSU, INUSEL
  20  16                           NM, JMAX
    0.50    0.50  9999.0           BC0, BC1, RHOUW
0.0303          0.0306             A, B
  +1.0000000D-14  1.00000000D+00  5.00000000D+00    EPSP, TOL, DZU
    1.25   15.00  100.00           ZM, ZMATCH, ZTOP
   0                               NHH
   2                               LMOD
.030            .0301          .1              ENXL, ENXU, ENXS
   1                               IPIO (0: M1M-, 1: M2M-)
   0.000   0.000   0   0           ZOMTTT, ZOMTTD, IPRTTT, IPCTTT
```

Parameter statements:

```
M2MK:
      PARAMETER  (NDFPAR = 5, NDVPAR = 3, NDP0 = 24, NDP = 22)
      PARAMETER  (NDMU = 11)
      PARAMETER  (NDG0 = 16, NNQSI = 16)

EWF2D:
      PARAMETER    (NOUT1 = 0, NIN1 = 3, NOUT4 = 8, NIN20 = 7,
     1              NDFPAR = 20, NDVPAR = 20,
     2              NDP = 22, NDP0 = 24, NDCN = 22,
     3              NDMU = 11, ISYM = 0,
     4              NDINT = 1000,
     5              MACHNO = 2)

DDS:
      PARAMETER    (NCDS = 66, NCDP = 22, NCDCN = 22, NCDML = 21,
     1              NCD1 = 2, NCD2 = 65, NCDINT = 1000)
```

Subroutine versions, output by EWF2D (Convex) for the /m2 test:

```
 -TEMPD- 1.1, 90/01/12. (CONVEX)
1    EWF2D  1.2, 92/08/24 (EWF2C 1.8) (NM, JMAX; MATDN, EWF2ZA, ZERODC)
 -EPIOA - 1.2, 91/10/24.
 -MATDN - 2.0, 92/07/06 (M1PIOH (FANA); MATDM 1.0C)
 -M1PIOH- 1.2, 92/05/26 (MODE = 5 (FANA); CFC004; 5) (M1PIOG 2.1)
 -M1MIOF- 1.0, 92/07/03 (M1MIOD 1.0) (IPXU, NS ADJUSTMENT)
 -M2WRDA- 1.2, 91/12/02.
 -EWF2ZA- 3.1, 92/04/30 (LXLAST) (ZERODB 2.1)
 -DDS   - 1.2, 92/08/24. DDR 1.1 (NM, JMAX ALIGNED; RRPSK)
 -RRLSDD- 1.4, 91/03/13. O: RRLSDC 1.0, 91/02/13.
 -LSELC   - 1.1, 90/04/02. FROM LSELB 1.0, 90/03/25.
 -VICDE - 1.0, 91/02/13. O: VICDD 1.3, 91/01/18.
 -MAINDB- 1.0, 92/03/12 (MATIND 2.1, 91/02/12; MATINV (M. HAFTEL))
 -RRPSK - 1.0, 92/03/26. RRPSJ 1.0 (-OPTJ)
 -BINA    - 1.0, 90/03/31.
 -POWA    - 1.0, 90/03/31.
 -DETDC - 1.0, 92/03/12 (DETDB 2.3, 91/12/30; DETERM (M. HAFTEL))
```

Additional changes in benchmarks are noted in Tables. Changed is RRPSK (Veclib call instead of some loops, or 70, 72, 74 loop order); DDS (optimization); etc.

3. Matrix multiplication benchmark

Updated 93/06/09; HP 710 94/02/07

Table 3-I. `N` $\times$ `N` matrix, dimension `ND`$\times$ `ND`. Program `P25` (Appendix B) and its variations. Initialization time not counted. Operation count used is $2N^3$ (the column before the last). E is the efficiency, given by the ratio of the MFLOPS performed and declared. If operation count is constant, E is proportional to R of Table 1-I.

Machine (declared MFLOPS)	ND	Length (MB)	N	CPU time (sec)	Remarks	MFLOPS	E
Vax 8650FPA	800	15	100	4.9	for/noopt		
			200	40			
			100	2.8	for/opt	0.7	
			200	23		0.7	
			400	188	*0	0.68	
IBM ES/9021 (Wien)							
(138)	800		800	33	*1	31	0.2
				15	*2	68	0.5
				10	*3 DGEMM	100	0.7
	800		200	1.4	*4		
			800	92	*4		
C220	800	16	100	1.9	-O0		
(50)			200	15			
			100	1.1	-O1		
			200	8.4			
			100	.05	-O2	40	0.8
			128	.1		40	
			200	.40		40	
			256	.85		39	
			300	1.3		42	
			400	3.3		39	
			800	27		38	
	1600	62	200	.4		40	
			400	3.3		39	
			800	28		37	0.7
	800		800	29	*5		
	800		400	2.9	*6 DGEMM		
			800	23	*6 DGEMM	45	0.9

C3800	800	16	100	.6	-O0		
(120)			200	5.2			
			100	.3	-O1		
			200	2.4			
			100	.02	-O2	100	0.8
			128	.04		100	
			200	.16		100	
			256	.32		105	
			400	1.25		102	
			800	9.9		100	
HP 730 (32 MB, KIBK)							
(23)	800		200	2.6	*7	6	0.3
			400	21	*7	6	
			800	166	*7	6	
HP 710 (32 MB, IJS 94/02/07) *8							
	400		200	2.1		6	
			400	16		6	
Cray X-MP/216 *9							
			200	.38	(DP: 8.3)	42	
			300	1.196	(DP: 25.7)	45	

*0 If K-loop is inside, the time is 270 sec (270/188 = 1.4) (see Ch. 4, R*8 and R*16 speeds).
*1 Run by P. Marksteiner, 92/11/27. Checked by R. K. 92/12/03.
*2 K-loop inside. This is better for IBM (adapted and run by P. Marksteiner, 92/11/27). Checked by R. K. 92/12/03 (got 16 seconds).
*3 DGEMM (P. Marksteiner, 92/11/27).
*4 FORTVS2 (OPT(3)) (R. Krivec 92/12/03).
*5 Source has K-loop inside, loops interchanged by fc (program P25A).
*6 DGEMM (Veclib).
*7 -O, variables in data area (not in stack).
*8 f77 -O, +O2, +O3 agree; +O1 is 3-4 times slower at N = 200. Values are orientational only because ND = 800 was not tested although the result is probably independent of ND as long as swap is not used. Time is given by the command `time`; the setup time in P25 is negligible.
*9 Unknown program (message of 91/05/15 by M. Haftel). Only slightly faster than C220 for N = 300. DP: Cray double precision.

```
Speedups at N = 200:
--------------------

Vax(/opt):Vax(/noopt)          1.3

C220(-O0):Vax(/noopt)          2.7
C220(-O1):Vax(/opt)            2.8

C220(-O2):Vax(/opt)           59

C220(-O2):C220(-O0)           38
C220(-O2):C220(-O1)           21          vectorization, 1 processor


Speedups at N = 400:
--------------------

C220(-O2):Vax(/opt)           57


Speedups at N = 800:
--------------------

C3800(-O2):C220(-O2)           2.7
```

Summary

IBM shows a large degree of dependence on the source program. Only the Veclib routines exploit the potential of the machine to the full, while the FORTVS2 compiler does not optimize nearly good enough. HP is also below expectations (though not a vector machine, and P25 was perhaps not thoroughly optimized for it). That is, the RISC processor is not as efficient as Convex for very vector-like operations, while being almost as efficient as Convex for the more realistic program EWF2D (Chapter 1).

Convex shows the most consistent efficiency. Note also that on Convex, vectorization can result in a speedup of 21 with respect to scalar optimization which is probably in a large measure due to better memory access. Cray is unexpectedly slow if its declared speed is 235 MFLOPS per processor (unconfirmed).

Note that the preferred loop order is quite different on IBM and Convex. According to P. Marksteiner, this has to do with the structure of the cache. This example shows strikingly how important it is to adjust the source code to different machines before making serious benchmarks.

4. Conclusion

Apart from the conclusions in the separate Chapters, some general conclusions, valid for small memory requirements (under 50 MB) are:

Unvectorizable programs (like M2MK) can run faster (less CPU time) on a superscalar RISC processor like the SG R3000 (16 MFLOPS declared) than on a vector processor like the Convex C220 (50 MFLOPS declared).

Vectorizable realistic programs (like EWF2D) can adapt well to HP and SG RISC and Convex vector processors, CPU time being approximately inversely proportional to the declared MFLOPS. This enables one to predict the speed of EWF2D on an untested machine from its MFLOPS specification.

For matrix multiplications, only Vax and Convex reach efficiencies of about 0.8 (realized *vs.* declared MFLOPS). In addition, it is interesting to note that on the Vax, improper loop order may be 40% slower: it is advisable to write programs in "vector" form even for scalar machines, to facilitate memory access. Workstations did not perform so well here.

The FORTVS2 compiler on IBM produces code strongly dependent on source code (loop order, Veclib routines or not). It is necessary to use Veclib to reach the efficiency of 0.7 for matrix multiplication.

Appendix A

Part of RRPSK 1.0, 92/03/26.

```
C     CALCULATE AND STORE ALL PP, QQ. M1 IS R+2 OF COMMENT.
      DO  68  M1 = 1,JMAX
          JR = M1 - 2
C
C
C         CALCULATE PP(R+1).
          ZZZ51 = BIN(-1+3,JR+1+1) * ZOMPOW(-JR-2+NZOMOF)
          DO  22  I = 1,MM
              DO  20  J = 1,MM
                  PP(J,I,M1) = ZERO
   20         CONTINUE
              PP(I,I,M1) = ZZZ51
   22     CONTINUE
          IF (ZOM .LE. ZUW) THEN
              DO  34  IPX = 1,IPXU
C                 POWER; **** SKIP IF .LT. 0 (BASIC ASSUMPTION). ****
                  IP = IPX - IPOFF
              IF (IP .LT. 0) GO TO  34
              IF (IP .GT. 0  .AND.  JR+1 .GT. IP) GO TO  34
                  ZZZ52 = - BIN(IP+3,JR+1+1) * ZOMPOW(IP-JR-1+NZOMOF) *
     1                    ZZZ4 ** (IP+1)
                  DO  32  I = 1,MM
                      DO  30  J = 1,MM
                          PP(J,I,M1) = PP(J,I,M1) + ZZZ52 * WD(J,I,IPX)
   30                 CONTINUE
   32             CONTINUE
   34         CONTINUE
          ENDIF
C
C
C         CALCULATE QQ(R) (IF R .GE. 0).
          DO  42  I = 1,MM
              DO  40  J = 1,MM
                  QQ(J,I,M1) = ZERO
   40         CONTINUE
   42     CONTINUE
          IF (JR .LT. 0) GO TO  66
C
C
C             ENERGY AND CENTRIFUGAL TERMS.
              ZZZ61 = -0.25D0
              ZZZ62 = -BIN(-2+3,JR+1) * ZOMPOW(-JR-2+NZOMOF)
              DO  50  I = 1,MM
                  IF (JR .EQ. 0) QQ(I,I,M1) = ZZZ61
                  IF (JR .GE. 0) QQ(I,I,M1) = QQ(I,I,M1) +
     1                                        ZZZ62 * MU(I) ** 2
   50         CONTINUE
          IF (ZOM .GT. ZUW) GO TO  66
```

```
            DO  64  IPX = 1,IPXU
C                   POWER; **** SKIP IF .LT. -1 (BASIC ASSUMPTION). ****
                    IP = IPX - IPOFF
                IF (IP .LT. -1) GO TO  64
                IF (IP .GT. 0  .AND.  JR .GT. IP) GO TO  64
                    ZZZ63 = -BIN(IP+3,JR+1) * ZOMPOW(IP-JR+NZOMOF) *
     1                      ZZZ4 ** (IP+2)
                    DO  62  I = 1,MM
                        DO  60  J = 1,MM
                            QQ(J,I,M1) = QQ(J,I,M1) + ZZZ63 * WS(J,I,IPX)
   60                   CONTINUE
   62               CONTINUE
   64           CONTINUE
   66       CONTINUE
   68 CONTINUE
C
C
C     CALCULATE  CNT . N1 IS P+1 OF COMMENT.
      DO  84  N1 = 3,JMAX1
C
C
C         RESET  CNT  (**** DO NOT RESET C(0), C(1) ****).
          DO  12  J = 1,MM
              DO  10  I = 1,MM
                  CNT(I,J,N1) = ZERO
   10         CONTINUE
   12     CONTINUE
          JP = N1 - 1
C
C
C         SUM OVER R (R = -1,..., JP-2).
          DO  76  M1 = 1,JP
              JR    = M1 - 2
              ZZZ72 = JP - JR - 2
              DO  73  K = 1,MM
                  DO  71  J = 1,MM
                      RR(J,K) = ZZZ72 * PP(J,K,M1) + QQ(J,K,M1)
   71             CONTINUE
   73         CONTINUE
              DO  74  J = 1,MM
                  DO  72  I = 1,MM
                      DO  70  K = 1,MM
                          CNT(I,J,N1) = CNT(I,J,N1)
     1                        + CNT(I,K,N1-JR-2) * RR(J,K)
   70                 CONTINUE
   72             CONTINUE
   74         CONTINUE
   76     CONTINUE
C
C
C         FINISH  C(JP) .
          ZZZ71 = - ONE / DFLOAT(JP * (JP - 1))
          DO  82  J = 1,MM
              DO  80  I = 1,MM
                  CNT(I,J,N1) = ZZZ71 * CNT(I,J,N1)
   80         CONTINUE
   82     CONTINUE
   84 CONTINUE
```

Appendix B

```
      PROGRAM  P25
C
C     EACH STEP TIMED SEPARATELY.
C     ONLY ONE LOOP CONFIGURATION TESTED (BECAUSE RRPSK 1.0 PROVED
C     STABILITY ON LOOP REARRANGEMENTS).
C     I-LOOP INSIDE (FOR COMPILERS WHICH DO NOT EXCHANGE LOOPS).
C     V1, 92/11/26.
C
      IMPLICIT    REAL*8  (A-H,O-Z)
      PARAMETER   (ND = 800, NIN = 5, NOUT = 6)
      DIMENSION   A(ND,ND), B(ND,ND), C(ND,ND)
C
      WRITE (NOUT,200)
      READ (NIN,100) N
      WRITE (NOUT,201) ND, N
      CALL TEMPD(T0, T1, TD, NOUT, .TRUE.)
      DO  12  J = 1,N
          DO  10  I = 1,N
              A(I,J) = (J * I)
              B(I,J) = (J + I)
   10     CONTINUE
   12 CONTINUE
      CALL TEMPD(T1, T2, TD, NOUT, .TRUE.)
      DO  24  J = 1,N
          DO  18  I = 1,N
              C(I,J) = 0.D0
   18     CONTINUE
   24 CONTINUE
      CALL TEMPD(T2, T3, TD, NOUT, .TRUE.)
      DO  26  J = 1,N
          DO  22  K = 1,N
              DO  20  I = 1,N
                  C(I,J) = C(I,J) + A(I,K) * B(K,J)
   20         CONTINUE
   22     CONTINUE
   26 CONTINUE
      CALL TEMPD(T3, T4, TD, NOUT, .TRUE.)
      WRITE (NOUT,202) N, ND, ((C(I,J), J = 1,4), I = 1,4)
C
  100 FORMAT (I4)
  200 FORMAT (1H , 8HP25 V1  )
  201 FORMAT (1H , 8HND, N   , 2I8)
  202 FORMAT (1H , 2I8, /, (1H , 4E16.8))
      END
```

— * —