MPL-43

To: Multics Performance Log

From: J. H. Saltzer

Subject: Comparison of Compile Time, Run Time and Size of a Small
          Program Using BASIC, FORTRAN, and PL/I.

          A short FORTRAN subroutine which reads a number, ℓ , and then
computes and prints the $(\ell + 2)$'nd prime was borrowed from the Multics
standard certifier script, and recoded in BASIC and PL/I. The three
versions were then compared on several points; the results are presented
here.

I. Source Programs

     The two new language versions were coded to be as similar as possible
in algorithm to the original FORTRAN routine. The lack of a modulo function
in BASIC was bypassed by a direct computation using the integer function;
this change being in the innermost loop may have affected execution time
performance described in Section II.

The three source programs are listed for comparison in Figure I.

II. Execution Time

     Each program was executed several times each, with input values of
5, 20, 40, 90, and 175. The input lines to both the command interpreter
and the program itself were queued. In order to distinguish page fault
time from pure execution time, each command was queued two or more times,
in an attempt to drive the number of page-faults to zero on second and
later executions. Each experiment was repeated several times, and the
smallest cpu time observed was recorded, in an attempt to minimize the
effect of interrupts whose execution time is currently charged to the
executing process. The results are shown in Figure III. In general, this
graph suggests that the execution time of the three object programs is
in the ratio 1:2:3 for FORTRAN, PL/I, and BASIC, respectively. Note that
with input value 175, the required execution time of all three programs
solidly swamps out the end effects of program starting and input/output
statements, and even of BASIC compilation time.

     The difference in execution time of FORTRAN and PL/I appeared worth
further study, so the object programs were compared in detail. Figure IV
and V exhibit the two object programs. In terms of the physical program

I. Comparison of source program listings:

fpm.fortran

11/24/69

```
      subroutine fpm
      read(5,70) l
70 format(i3)
      m = 1
      do 10 i = 3,100000
      k = i-1
      do 20 j = 2,k
      if(mod(i,j)) 20,10,20
20 continue
      m = m+1
      if(m-1) 10,40,40
10 continue
40 write(6,60) m,i
60 format(7h Prime ,i4,3h is,i6)
      return
      end
```

pm.pl1

```
pm:           procedure;
declare (m,i,k,j,l) fixed binary;
              call read_list_(l);
              m = 1;
              do i = 3 to 100000;
              k = i - 1;
              do j = 2 to k;
              if mod(i,j) = 0 then go to 1190;
1160:         end;
              m = m+1;
              if m>= 1 then go to 1200;
1190:         end;
1200:         call ioa_("prime ¬d is ¬d", m, i);
              end;
```

pm.basic

```
100 input l
110 let m = 1
120 for i = 3 to 100000
130 let k = i - 1
140 for j = 2 to k
150 if (i = int(i/j)*j) then 190
160 next j
170 let m = m + 1
180 if m >= 1 then 200
190 next i
200 print "prime";m;"is";i
210 end
```

II. Queuing of command input to obtain zero-page-fault case:

```
basic pm
20
basic pm
20
basic pm

Compile time in ms. = 368, Page waits = 21
            11/21/69              23:07

? PRIME 20    IS 71

r 2307   1.178   67


Compile time in ms. = 193, Page waits = 0
            11/21/69              23:07

? PRIME 20    IS 71

r 2307   .611   0
```
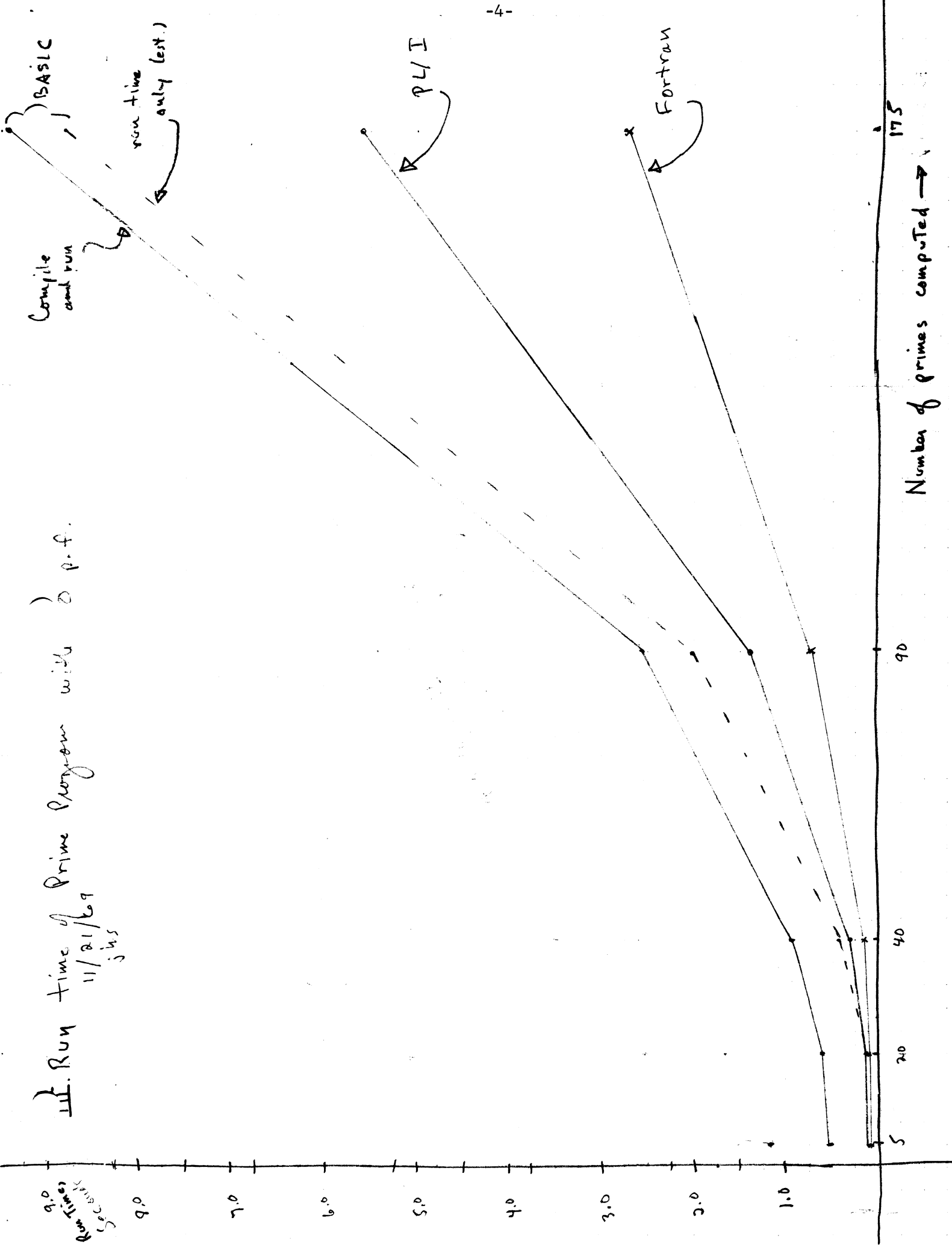
II. Run time of Prime Program with 3 P.f.
11/21/69
jhs

Run Time
(seconds)

9.0

8.0

7.0

6.0

5.0

4.0

3.0

2.0

1.0

5   20   40   90   175

Number of primes computed →

BASIC

Compile and run

run time only (est.)

PL/I

Fortran

sizes, the reason for the difference in execution time is clear. The inner and outer program loops measure out as follows:

	FORTRAN	PL/I
Inner loop	11 instructions	14 inst. + 5 in subroutine = 19
Outer loop	22 instructions	39 inst. (including subroutine

Upon inspecting the compiled code, the difference seems to be primarily that FORTRAN is very good on register optimization, while PL/I largely ignores the subject. Compare also the FORTRAN compiled "AOS" on line 152, compared with the PL/I sequence starting at 73, both for the statement "m = m + 1". It appears that FORTRAN misses few tricks.

III. Compile Time

The compile times were compared, using a similar command stacking technique to get the compiler "in core" and minimize the effect of missing page faults. It was found to be impossible to bring the number of page faults to zero when using PL/I as the compiler apparently does not fit into the available (~160K) core. Results were as follows, in both cases after linking had been accomplished by an earlier command.

	Compile Time/sec.	Page Faults
FORTRAN	1.720	0
PL/I	4.698	170
BASIC	.500(est.)	0

The BASIC compile time is estimated from the lower asymptote of its compile-and-execute curve in Figure III.

IV. Working Set Size

Each program was compiled following a "flush" command, and then executed (with input value 5) following a "flush" command. The experiment was repeated several times to insure that other users had not distorted the result. The following table indicates the number of missing-page faults observed.

IV. Compilation of source lines 4-12 by FORTRAN:

11/24/69

```
STATEMENT BEGINNING ON LINE #    4
   000131  000001236007     000000          LDQ      =0000001.DL
   000132  000001756100     000000          STQ      AP/m
   000133  000003236007     000000          LDQ       000003.DL
           000134                    A00037  NULL
   000134  000002756100     000000          STQ      AP/i
STATEMENT BEGINNING ON LINE #    6
   000135  000001176007     000000          SBQ      =0000001.DL
   000136  000004756100     000000          STQ      AP/k
   000137  000002236007     000000          LDQ       000002.DL



           000140                    A00063  NULL
   000140  000005756100     000000          STQ      AP/j
STATEMENT BEGINNING ON LINE #    8
   000141  000002236100     000000          LDQ      AP/i
   000142  000005506100     000000          DIV      AP/j
   000143  000044773000     000000          LRL       000044
   000144  000000600000     200000          TZE      A00144
           000145                    A00105  NULL
   000145  000005236100     000000          LDQ      AP/j
   000146  000001036007     000000          ADLQ      000001.DL
   000147  000004116100     000000          CMPQ     AP/k
   000150  000140600000     200000          TZE      A00063
   000151  000140602000     200000          TNC      A00063
STATEMENT BEGINNING ON LINE #   10
   000152  000001054100     000000          AOS      AP/m
STATEMENT BEGINNING ON LINE #   11
   000153  000001236100     000000          LDQ      AP/m
   000154  000003176100     000000          SBQ      AP/l
   000155  000000605000     200000          TPL      A00156
           000156                    A00144  NULL
   000156  000002236100     000000          LDQ      AP/i
   000157  000001036007     000000          ADLQ      000001.DL
   000160  303241116007     000000          CMPQ      303241.DL
   000161  000134602000     200000          TNC      A00037
           000162                    A00155  NULL
```

Outer Loop

Inner Loop

V. Compilation of source lines 4-12 by PL/I:

11/24/69

000036	aa	777745 2360 04	llq	-27,ic	
000037	aa	6 00124 7561 00	stq	sp|84	
000040	aa	777747 2360 04	llq	-25,ic	
000041	aa	6 00125 7561 00	stq	sp|85	
000042	aa	6 00125 2361 00	llq	sp|85	
000043	aa	777745 1160 04	cmpq	-27,ic	
000044	aa	000002 6000 04	tze	2,ic	
000045	aa	000041 6050 04	tpl	33,ic	
000046	aa	6 00125 2361 00	llq	sp|85	
000047	aa	777734 1760 04	sbq	-36,ic	
000050	aa	6 00126 7561 00	stq	sp|86	

STATEMENT 1 ON LINE 4
000003 = 000000000001

STATEMENT 1 ON LINE 5
000007 = 000000000003

000010 = 000000303240
000046
000106

STATEMENT 1 ON LINE 6

000003 = 000000000001

OUTER LOOP

000051	aa	6 00126 2361 00	llq	sp|86	
000052	aa	6 00131 7561 00	stq	sp|89	
000053	aa	777736 2360 04	llq	-34,ic	
000054	aa	6 00127 7561 00	stq	sp|87	
000055	aa	6 00127 2361 00	llq	sp|87	
000056	aa	6 00131 1161 00	cmpq	sp|89	
000057	aa	000002 6000 04	tze	2,ic	
000050	aa	000013 6050 04	tpl	11,ic	
000051	aa	6 00125 2361 00	llq	sp|85	
000052	aa	6 00127 3521 00	eapbp	sp|87	
000053	aa	0 00704 6701 00	tsblp	ap|452	
000054	aa	777716 1160 04	cmpq	-50,ic	
000055	aa	000002 6010 04	tnz	2,ic	
000056	aa	000014 7100 04	tra	12,ic	
000057	aa	6 00127 2361 00	llq	sp|87	
000070	aa	777713 0760 04	adq	-53,ic	
000071	aa	6 00127 7561 00	stq	sp|87	
000072	aa	777763 7100 04	tra	-13,ic	
000073	aa	6 00124 2361 00	llq	sp|84	
000074	aa	777707 0760 04	adq	-57,ic	
000075	aa	6 00124 7561 00	stq	sp|84	
000076	aa	6 00124 2361 00	llq	sp|84	
000077	aa	6 00130 1161 00	cmpq	sp|88	
000100	aa	000002 6040 04	tmi	2,ic	
000101	aa	000005 7100 04	tra	5,ic	
000102	aa	6 00125 2361 00	llq	sp|85	
000103	aa	777700 0760 04	adq	-64,ic	
000104	aa	6 00125 7561 00	stq	sp|85	
000105	aa	777735 7100 04	tra	-35,ic	

INNER LOOP

STATEMENT 1 ON LINE 7

000011 = 000000000002

000061
000073
STATEMENT 1 ON LINE 8

mod_fx1 — 5 instruction subroutine

000002 = 000000000000
000067
000102
STATEMENT 1 ON LINE 9

000003 = 000000000001

000055
STATEMENT 1 ON LINE 10

000003 = 000000000001

STATEMENT 1 ON LINE 11

000102
000106
STATEMENT 1 ON LINE 12

000003 = 000000000001

000042

	Compile	Run
BASIC	74	
FORTRAN	105	15
PL/I	225	27

Note that the PL/I compiler taken only a few more page faults when it follows "flush" than when it follows itself. The unusually large number of pages required to execute the PL/I object program suggested further analysis. Figure VI shows a page trace of the PL/I program. Most of the trouble clearly arose from the use of subroutine "read_list_" for input, since that subroutine call resulted in 12 distinct page faults. It also appears that in system 4.7f, segment "pl1-operators is not yet wired-down.

A second fact uncovered by the page trace is that segment 110, the Teletype DIM (bound-tty-active) is organized so that all six of its pages are touched by a write and read call. In earlier systems, this segment was "optimally bound" such that only three pages were touched. A review of the contents of bound-tty-active suggests that such a reordering ot its components could again reduce the number of pages touched to three.

VI. Page fault trace of PL/I object program execution following a "flush" command:

19/23/69

```
meter_start;flush
measurement started
pm 5
5
meter_stop
r 819   2.564   293

Ready for 1 value
prime 5 is 11
r 819   .322  (27

measurement stopped
r 819   .098   6
```

Page #
segment # or name if known

```
print_pages 35
  18696668625   0  000124
  18696643303   0  phcs_.link       } meter-stop
  18696508150  16  meter_start          command
  18696489710  31  !BBBHwnPcGZQDDx
  18696402098   5  !BBBHwnPcGZQDDx
  18696359160   3  read_list_
  18696328938   0  free_
  18696286736   0  bin_oct              12 pages
  18696240114   0  read_list_           touched
  18696192060   4  read_list_           by
  18696169407   2  read_list_           read_list_
  18696031631   6  write_out
  18696006734  11  !BBBHwnPcGZQDDx
  18695970285   2  stack_01
  18695959267   1  stack_01
  18695923550   3  stack_01
  18695839472   1  read_list_
  18695803045   1  pl1_operators  } not wired yet !
  18695770729   0  pl1_operators  }
  18695621300   0  pm
  18695598870  32  !BBBHwnPcGZQDDx
  18695567624   6  000110
  18695546720   0  000110
  18695516733   3  000110
  18695475981   1  000053       — TTY DIM
  18695432839   2  000110
  18695384450   0  000112
  18695345641   5  000110
  18695319802   4  000110
  18695288055   5  000016
  18695266610   1  listen_
  18695217248   1  hcs_.link
  18695180876   0  000120
  18695155433   0  process_info
  18695135919   0  hcs_.link
  r 820  3.227  63
```

27 pages touched

tail end of flush command

↑ Time in μsec.