my file.

IN CONFIDENCE

SCIENCE AND ENGINEERING RESEARCH COUNCIL
RUTHERFORD APPLETON LABORATORY

COMPUTING DIVISION

DISTRIBUTED INTERACTIVE COMPUTING NOTE 777

ICL PERQ v APOLLO DOMAIN - NON-ARITHMETIC BENCHMARKS

issued by
I D Benest
4th January 1983

Copies to: K Robinson
C Prosser
J Collis
L Ford
R Witty
I D Benest
DIC Notes file
Apollo/General

Circulate to: F R A Hopgood
All Section

This note documents the results obtained from running a miscellaneous set of non-arithmetic benchmarks on the ICL PERQ and APOLLO DOMAIN computers.

1 INTRODUCTION

A series of benchmarks have been carried out to compare the speed of a number of computer systems sited at the Rutherford Appleton Laboratory. This note documents the results obtained from running a miscellaneous set of non-arithmetic benchmarks on the ICL PERQ and APOLLO DOMAIN machines.

2 COMPUTER DESCRIPTIONS

2.1 ICL Perq

The ICL PERQ used for these tests had the following characteristics:

1. 512kbytes random access memory
2. 24Mbyte Winchester hard disk
3. General Purpose Interface Bus
4. RS232C interface port
5. 768x1024 black and white portrait oriented screen
6. 1Mbyte floppy disk drive
7. Keyboard, tablet
8. PERQ Operating System (POS) D.70
9. PASCAL V6.0
10. FORTRAN V1.3
11. LINK V4.4
12. Microcoded floating point

2.2 Apollo Domain

The APOLLO DOMAIN used for these tests had the following characteristics:

1. Motorola 68000 8MHz CPU
2. 512kbytes random access memory
3. 33Mbyte Winchester hard disk
4. Intel multibus
5. Three RS232C interface ports
6. 800x1024 black and white portrait oriented screen
7. 1Mbyte floppy disk drive
8. Keyboard with touch pad
9. AEGIS operating System 4.0
10. PASCAL 3.00
11. FORTRAN 4.00
12. Software floating point

3 BENCHMARKS

3.1 Introduction

This is a miscellaneous set of benchmarks which include a sorting program, the Eratosthenes prime number generator and disk access tests. All tests were timed using a stop watch from the moment the carriage return was pressed invoking the relevant command, until the operating system prompt re-appeared. Although all compilations, loadings and program runs were timed, only the results of the program runs are included in this note. The tests were performed in the order: compile, load, run, compile, load, run, compile, etc.

3.2 Benchmark Descriptions

NOOPINT2 - A no-operation with 16 bit integer loop counters in both FORTRAN and PASCAL. PERQ FORTRAN does not support single precision integers and therefore could not be performed on the PERQ. The programs are listed in Appendix A.

NOOPINT4 - A no-operation with 32 bit integer loop counters in FORTRAN. The program is listed in Appendix B.

SIEVE - Determines the first 1899 prime numbers using the sieve of Eratosthenes' algorithm and performs this test 10 times. The sieve avoids division and uses prior knowledge about numbers that cannot be prime. The programs for the SIEVE tests are listed in Appendix C and were derived from reference [1]. In order to be compatible with reference [1], single precision (16 bits) integers were declared. Gilbreath's PASCAL coding was modified by replacing the line:

```
fillchar(flags,sizeof(flags),chr(true));
```

with the line:

```
for i:=0 to size do flags[i]:=true;
```

PERQ FORTRAN only provides for 32 bit integers, so that PERQ FORTAN SIEVE could not be performed. As PERQ PASCAL does not provide for 32 bit integer loop controls, a comparison between PASCAL and FORTRAN implementations of SIEVE on the PERQ could not be performed.

WRITETODISK - Writes 1Mbyte of ASCII characters to disk using FORTRAN formatted I/O. The test is listed in Appendix D.

READFROMDISK - Reads 1Myte of ASCII characters from disk using FORTRAN formatted I/O. The test is listed in Appendix D.

QSORT - Generates 15,500 numbers alternately in descending and ascending order and then uses a quick sort procedure to sort them in ascending order. This test is written in PASCAL and is listed in Appendix E.

4 RESULTS

Table 1 gives the results of running the programs listed in Appendices A,B and C. The SIEVE program in ICL PERQ FORTRAN (32 bits) took 23.0 seconds to complete.

TEST	ICL PERQ		APOLLO DOMAIN	
	FORTRAN (secs)	PASCAL (secs)	FORTRAN (secs)	PASCAL (secs)
NOOPINT2	NA	15.6	9.2	9.0
NOOPINT4	26.8	*	10.6	*
SIEVE	NA	11.2	12.4	8.0

Table 1: Miscellaneous Benchmarks (Stopwatch Timing)

Table 2 gives the results of the disk access tests and the sort test. The timings for the disk test represent the actual time to write to, and read from, the hard Winchester disk, the program overheads having been removed.

TEST	ICL PERQ (secs)	APOLLO DOMAIN (secs)
WRITETODISK	1475.0	241.2
READFROMDISK	310.0	205.7
QSORT	31.6	16.9

Table 2: Disk I/O and Sort Benchmark Results (stopwatch Timing)

5 ANALYSIS

Table 3 indicates the relative speed of operation of the two computers. All tests indicate that the Apollo Domain is faster than the ICL PERQ. In particular, writing to the PERQ disk is very much slower.

TEST	ICL PERQ	APOLLO DOMAIN
NOOPINT2 FORTRAN	NA	*
NOOPINT2 PASCAL	1.73	1.0
NOOPINT4 FORTRAN	2.53	1.0
SIEVE FORTRAN	NA	*
SIEVE PASCAL	1.40	1.0
WRITETODISK	6.12	1.0
READFROMDISK	1.51	1.0
QSORT	1.87	1.0

Table 3: Relative speed of Benchmarks

6 CONCLUSIONS

The variety of non-arithmetic tests noted in this document is too small to conclude that the ICL PERQ is slower than the Apollo Domain Computer for this category of benchmark. Nevertheless, the results are not encouraging and the speed of the PERQ's disk write will severely hamper its ability to provide fast virtual memory processing. The Apollo Domain used in these tests could have been enhanced with a 10MHz CPU and a cache/floating point unit, thus enabling it to run significantly faster.

7 REFERENCE

[1] Gilbreath, J., September 1981, "A High Level Language Benchmark", BYTE, page 180-198.

8 APPENDIX A: NOOPINT2 - NO-OPERATION SHORT INTEGERS

```
      program noop
      integer*2 i,j,count
      integer*2 outmax,loopmx

      outmax=100
      loopmx=100

      do 500 count=1,outmax
         do 400 i=1,loopmx
             do 300 j=1,loopmx

  300        continue
  400     continue
  500  continue
       stop
       end
```

```
program noop(input,output);
const loopmx=100;
      outmax=100;
var   i,j,count : integer;

begin
      for count:=1 to outmax do
      begin
        for i:=1 to loopmx do
        for j:=1 to loopmx do
        begin

        end
      end;
end.
```

9 APPENDIX B: NOOPINT4 - NO-OPERATION LONG INTEGERS

```
      program noop
      integer i,j,count
      integer outmax,loopmx

      outmax=100
      loopmx=100

      do 500 count=1,outmax
         do 400 i=1,loopmx
             do 300 j=1,loopmx

  300        continue
  400    continue
  500 continue
      stop
      end
```

10 APPENDIX C: SIEVE PROGRAM

```
{Eratosthenes Sieve Prime Number Program in PASCAL}
Program prime(input,output);

const size = 8190;

var flags : array[0..size] of boolean;
    i,prime,k,count,iter : integer;

begin
writeln('10 iterations');
for iter := 1 to 10 do begin
    count := 0;
    for i:=0 to size do flags[i]:=true;
    for i:= 0 to size do
            if flags[i] then begin
               prime := i+i+3;
               k:= i + prime;
               while k <= size do begin
                     flags[k] := false;
                     k := k + prime;
                     end;
               count := count + 1;
              {writeln(prime)}
               end;
     end;
writeln(count,' primes')
end.
```

```
C Eratosthenes Sieve Prime Number Program in FORTRAN
      LOGICAL FLAGS(8191)
      INTEGER*2 I,PRIME,K,COUNT,ITER

      WRITE(*,50)
50    FORMAT(' 10 iterations')
      DO 92 ITER = 1,10
      COUNT=0
      DO 10 I = 0,8190
10    FLAGS(I)=.TRUE.
      DO 91 I = 0,8190
      IF(FLAGS(I).EQV..FALSE.)GO TO 91
      PRIME=I+I+3
      K=I+PRIME
20    IF(K.GT.8190)GO TO 90
      FLAGS(K) = .FALSE.
      K=K+PRIME
      GOTO 20
90    COUNT=COUNT+1
C     WRITE(*,100)PRIME
91    CONTINUE
92    CONTINUE
      WRITE(*,200) COUNT
200   FORMAT(1X,I6,' primes')
      STOP
100   FORMAT(1X,I6)
      END
```

11 APPENDIX D: DISK I/O TESTS

```
      program wtd

c
c     program to write 1Mbyte to disk
c
c     writes 'nol' lines to disk. (nol=no. of lines)
c     Each line has 9 alphabetic characters followed
c     by an end of line.
c     A similar program (rfd) reads 1Mbyte from disk
c     and a dummy program wrtfdd may be used to
c     subtract overheads to get actual disk read/write
c     times.  Make sure that the file 'temp' is deleted
c     before writing to it.
c


      integer i,unit
      character*9 arr

      data unit,nol,arr/7,100000,'aabbccddz'/

      open(unit,file='temp',status='unknown')
      rewind(unit)

      do 20 i=1,nol
         write(unit,10)arr
10       format(a9)
20    continue

      close(unit)

      stop
      end
```

```
      program rfd

c
c     program to read 1Mbyte from disk
c
c     reads 'nol' lines from disk. (nol=no. of lines)
c     Each line has 9 alphabetic characters followed
c     by an end of line.
c     A similar program (wtd) writes 1Mbyte to disk
c     and a dummy program wrtfdd may be used to
c     subtract overheads to get actual disk read/write
c     times.
c

      integer i,unit
      character*9 arr

      data unit,nol,arr/7,100000,'aabbccddz'/

      open(unit,file='temp',status='unknown')
      rewind(unit)

      do 20 i=1,nol
         read(unit,10)arr
10       format(a9)
20    continue

      close(unit)

      stop
      end
```

```
      program wrtfdd

c
c     program wtd writes 1Mbyte to disk
c     program rfd reads 1Mbyte from disk
c     program wrtfdd is a dummy program used to
c     subtract overheads to get actual disk read/write
c     times.  Becareful to ensure that no optimization
c     is invoked which would remove the do loop.
c


      integer i,unit
      character*9 arr

      data unit,nol,arr/7,100000,'aabbccddz'/

c      open(unit,file='temp',status='unknown')
c      rewind(unit)

      do 20 i=1,nol
c         write(unit,10)arr
c10       format(a9)
20    continue

      close(unit)

      stop
      end
```

12 APPENDIX E: QSORT PROGRAM

```
program qsort( input, output );

const
  max = 15500;   { maximum no of elements sorted }

type
  test = array[ 1..max ] of integer; { storage for elements }

var
  numbers : test;
  W       : integer;
  high    : integer;
  count   : integer;

{                                                                    }
{A recursive quicksort procedure : begins by comparing elements at}
{opposite ends of the array. Therefore, initial interchanges for  }
{quicksort can be made over large distances                         }
{                                                                    }
procedure qsort( var X : test; M, N : integer );

var
  I : integer;
  J : integer;


procedure partition( var X : test;
                     var I, J : integer;
                     left, right : integer );

var
  pivot : integer;


procedure swap( var P, Q : integer );

var
  temp : integer;


begin
temp := P;
P := Q;
Q := temp
end;            { of swap }

begin
pivot := X[ ( left + right ) div 2 ];
I := left;
J := right;
while I <= J do
```

```
  begin
  while X[ I ] < pivot do

     I := I + 1;

  while pivot < X[ J ] do

     J := J - 1;

  if I <= J then

     begin
     swap( X[ I ], X[ J ] );
     I := I + 1;
     J := J - 1;
     end

  end

end;              { of partition }


begin
if M < N then

  begin
  partition( X, I, J, M, N ); {   divide into two    }
  qsort( X, M, J );           { sort left partition  }
  qsort( X, I, N )            { sort right partition }
  end

end; { of quicksort }

{ main program }

begin
W:=1;
count := 1;
high := max * 2;

  while count <= max do
  begin
     high := high - 1;
     numbers[ count ] := high;
     count := count + 1;
     numbers[ count ] := W;
     count := count + 1;
     W := W + 2;
  end;
qsort(numbers,1,max);
end.
```