Technical Document 1111
July 1987

UNIX Benchmark System

T. Brewer
Integrated Systems Analysts, Inc.
This work was performed for Code 914, NAVOCEANSYSCEN. Contract N66001-84-D-0053 was carried out by Integrated Systems Analysts, Inc., Marina Gateway, 740 Bay Boulevard, Chula Vista, California 92010-5254.

Released by
R.L. Broersma, Head
System Software Branch

Under authority of
J. Gilbreath, Head
Computer Sciences and Simulation Division
UNIX Benchmark System

A benchmark suite has been consolidated to test and evaluate a variety of computer systems and compare the results. VAX 11/780, under UNIX 4.3 BSD, has been selected as the baseline system to which each target system (procurement candidates) would be compared. The program, BENCH, collects and stores test results from all the target systems, and produces two reports. The first report compares any two systems that the user selects. The second report summarizes all the test data into a single report.

The suite presently has 18 tests and the user can specify which test may be used by modifying an ASCII file called BENCHLIST. Additional tests may be added to the suite by modifying the BENCHLIST and supplying the appropriate code.
## CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>1</td>
</tr>
<tr>
<td>Descriptive Summary</td>
<td>2</td>
</tr>
<tr>
<td>Fortran Tests</td>
<td>2</td>
</tr>
<tr>
<td>Sieve Tests</td>
<td>3</td>
</tr>
<tr>
<td>General Tests</td>
<td>3</td>
</tr>
<tr>
<td>Pre-Award Tests</td>
<td>4</td>
</tr>
<tr>
<td>Database Format</td>
<td>5</td>
</tr>
<tr>
<td>Reports</td>
<td>7</td>
</tr>
<tr>
<td>Comparison Report</td>
<td>9</td>
</tr>
<tr>
<td>UNIX Programmer's Manual</td>
<td>13</td>
</tr>
<tr>
<td>Instructions for Data Collection Using the Bench Program</td>
<td>15</td>
</tr>
<tr>
<td>Prime Numbers</td>
<td>17</td>
</tr>
<tr>
<td>Calling Sequence and Arguments Passing</td>
<td>18</td>
</tr>
<tr>
<td>Random Numbers</td>
<td>19</td>
</tr>
<tr>
<td>Fast Fourier Transform</td>
<td>21</td>
</tr>
<tr>
<td>Matrix Inversion</td>
<td>23</td>
</tr>
<tr>
<td>Polynomial Roots</td>
<td>25</td>
</tr>
<tr>
<td>C Sieve</td>
<td>28</td>
</tr>
<tr>
<td>Fortran Sieve</td>
<td>29</td>
</tr>
<tr>
<td>Pascal Sieve</td>
<td>30</td>
</tr>
<tr>
<td>Whetstone</td>
<td>31</td>
</tr>
<tr>
<td>Dhrystone</td>
<td>36</td>
</tr>
<tr>
<td>Block Write</td>
<td>45</td>
</tr>
<tr>
<td>Block Read</td>
<td>46</td>
</tr>
<tr>
<td>Sort (Part I)</td>
<td>47</td>
</tr>
<tr>
<td>Sort (Part II)</td>
<td>48</td>
</tr>
<tr>
<td>Sort (Part III)</td>
<td>50</td>
</tr>
<tr>
<td>Integer Arithmetric</td>
<td>52</td>
</tr>
<tr>
<td>Real Arithmetric</td>
<td>53</td>
</tr>
<tr>
<td>Large Data Space</td>
<td>54</td>
</tr>
<tr>
<td>Compile</td>
<td>55</td>
</tr>
</tbody>
</table>
Introduction

In March 1986, NAVOCEANSYSCEN tasked ISA to prepare a comprehensive test suite to systematically test and evaluate a variety of computer systems and compare the results. NAVOCEANSYSCEN supplied ISA with their Pre-Award test suite, with instructions to also include certain published tests, such as the Dhrystone and Whetstone benchmarks. It was ISA's task to consolidate all the designated tests into one easily-runnable program which can be used by either Government or Contractor personnel to test all computer systems that are candidates for procurement.

NAVOCEANSYSCEN selected the VAX 11/780 under UNIX 4.3 BSD as the baseline system to which each target system (procurement candidates) would be compared. Our program, bench, collects and stores test results from the baseline system, collects and stores results from all the target systems, and produces two reports. The first report compares any two systems which the user selects, and the second report summarizes all the test data in one report. The user tells the program which of these tasks he wishes performed through the use of options on the input line. All input is in standard UNIX format. For example,

% bench -b

would execute the test suite and store the results as the baseline data. See the enclosed man page for a list of all available options.

The suite includes eighteen tests at present. The user is able to specify which of the eighteen tests will be used by modifying an ASCII file called benchlist. Benchlist includes the names of all the tests. The user adds or deletes the comment indicator to tell the program to include or exclude that particular test. Additional tests may be added to the suite by modifying the benchlist and supplying the appropriate code.
Descriptive Summary
The following is a very brief description of the tests in the benchmark suite.

Fortran Tests

1. Prime Numbers
   This program generates the prime numbers from 0 to 8192 (optionally printing out the results).

2. Calling Sequence and Argument Passing
   This program initializes nine variables, passes them to a subroutine, which in turn has four assignments. This sequence is repeated one million times.

3. Random Numbers
   This program tests the random number generator by generating 12,800 random numbers and checking the randomness.

4. Fast Fourier Transform
   This program performs fast fourier transform using the decimation-in-time method (optionally printing out the results).

5. Matrix Inversion
   This program performs matrix inversion using the Gauss-Jordan Reduction (optionally printing out the results).

6. Polynomial Roots
   Roots of polynomials are calculated using the Bairstow's method (optionally printing out the results).
Sieve Tests

7. C Sieve
C version of the Sieve of Eratosthenes prime number program.

8. Fortran Sieve
Fortran version of the Sieve of Eratosthenes prime number program.

9. Pascal Sieve
Pascal version of the Sieve of Eratosthenes prime number program.

General Tests

10. Whetstone
A C version of the original Algol benchmark, "A Synthetic Benchmark" by H. J. Curnow and B. A. Wichman. Compiler optimization and floating point performance are tested.

11. Dhrystone
This program contains a distribution of statements which are considered to be representative: 53% assignment, 32% control statements, and 15% procedure and function calls.
Pre-Award Tests

12. Block Write
This program creates a very large file by writing 8K byte blocks one after the other.

13. Block Read
This program reads the file created by the block write program. The reads are executed in 8K byte blocks.

14. Sort
A shell script to test the section 1 sort call. A file is sorted on a particular column and the result is compared to a presorted file to test the results of the sort.

15. Integer Arithmetic
Addition, subtraction, multiplication, and division are performed on integer variables. The group of operations is executed 2.9 million times.

16. Real Arithmetic
Similar to the integer arithmetic, this program performs addition, subtraction, multiplication, and division on real variables. This group of operations is executed 600,000 times.

17. Large Data Space
This program references a data area larger than real memory making 20,000 references to random locations.

18. Compile
This script compiles two C code files and loads the two object files into a single output file.
Database Format

The benchmark test data is stored in a series of files which reside in the current working directory. The file containing the baseline data is called baseline. A file is created for each system tested and is called targetXXX, where XXX is a 3-digit number assigned by the benchmark program and which is unique to each system. The format for both the baseline file and all the target files is identical.

The format of the database files illustrated on the following page.
DEC SYSTEM MANUFACTURER
VAX 11/780 SYSTEM MODEL NUMBER
4 3 BSD OPERATING SYSTEM VERSION
5 NUMBER OF TESTS, INCLUDING ONE FOR THE COMPOSITE

THREE FLAGS (0 OFF, 1 ON) FOR THE THREE WAYS TO EXECUTE THE SUITE: SINGLY, ALL AT ONCE, ALL AT ONCE WITH A LOAD

EACH LINE CONTAINS THREE NUMBERS:
1. REAL TIME
2. USER TIME
3. SYSTEM TIME

EACH GROUP OF THREE LINES REPRESENTS THE RESULTS FROM ONE TEST AND EACH LINE REPRESENTS ONE REPETITION OF THE TEST THIS IS SET IN THE C CODE (CURRENTLY REP = 3)
Reports

Bench produces two reports: a comparison report based on two systems of the user's choice, and a summary report which includes all systems tested.

The comparison report is invoked when the user specifies the -p option on the command line. Bench displays a list of those systems in its database, and prompts the user to choose two systems from the list. The comparison report displays elapsed time, user time, system time, and percent usage for each test and each system chosen. A composite is also displayed. The composite is a sum of all systems chosen and represented as if it were a separate test. The elapsed time is the total amount of time that is consumed; the "clock" time. The user time is the amount of time the process spent executing nonprivileged instructions (e.g., arithmetic calculations, sorting, searching, etc.). System time is the time the process spent executing privileged (kernel) commands, such as system calls, plus some system-level overhead. The percent usage is that portion of the elapsed time that is actually spent executing the command. It is calculated thusly:

\[
\text{percent usage} = \left( \frac{\text{system time} + \text{user time}}{\text{elapsed time}} \right) \times 100
\]

The lowest elapsed time for each test for each system is indicated on the report by an asterisk (*). A separate column is displayed for the elapsed ratio. The first figure in the elapsed ratio column is the lowest time ratio, which is the ratio of the lowest elapsed time of the second system to the lowest elapsed time of the first system, or

\[
\text{lowest time ratio} = \frac{\text{lowest elapsed time of second system}}{\text{lowest elapsed time of first system}}
\]
The second figure of the elapsed ratio is the average time ratio. The average time ratio is the average elapsed time of the second system divided by the average elapsed time of the first system, or

\[
\text{average time ratio} = \frac{\text{average elapsed time of second system}}{\text{average elapsed time of first system}}
\]

At the bottom of the report, an average elapsed ratio is given, based on all tests except the composite. The average elapsed ratio is the average of each lowest time ratio (first figure) and the average of each average time ratio (second figure).

The summary report displays elapsed time averages for all systems tested. The test results are normalized to the baseline system. The summary report is invoked when the user specifies the -e option on the command line.

Sample output for both comparison report and the summary report follow.
Comparison Report

Test executed one at a time with no extra load (Expressed in seconds)

<table>
<thead>
<tr>
<th>Type</th>
<th>Test Number</th>
<th>Elapsed Time</th>
<th>User Time</th>
<th>System Time</th>
<th>Percent Usage</th>
<th>Test Number</th>
<th>Elapsed Time</th>
<th>Elapsed Ratio</th>
<th>User Time</th>
<th>System Time</th>
<th>Percent Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>composite</td>
<td>1</td>
<td>20.08</td>
<td>5.18</td>
<td>0.36</td>
<td>28</td>
<td>1</td>
<td>20.07</td>
<td>1</td>
<td>5.26</td>
<td>0.42</td>
<td>28</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>20.04</td>
<td>5.09</td>
<td>0.25</td>
<td>27</td>
<td>2</td>
<td>20.08</td>
<td>1</td>
<td>5.18</td>
<td>0.60</td>
<td>29</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>20.04*</td>
<td>5.06</td>
<td>0.30</td>
<td>27</td>
<td>3</td>
<td>20.00</td>
<td>1</td>
<td>5.20</td>
<td>0.58</td>
<td>29</td>
</tr>
<tr>
<td>avg</td>
<td></td>
<td>20.05</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>20.08</td>
<td>[1.00, 1.00]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>uptime</td>
<td>1</td>
<td>0.51</td>
<td>0.05</td>
<td>0.10</td>
<td>29</td>
<td>1</td>
<td>0.78*</td>
<td>0.04</td>
<td>0.11</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>0.91</td>
<td>0.03</td>
<td>0.08</td>
<td>12</td>
<td>2</td>
<td>0.81</td>
<td>0.05</td>
<td>0.11</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.50*</td>
<td>0.05</td>
<td>0.07</td>
<td>24</td>
<td>3</td>
<td>1.42</td>
<td>0.02</td>
<td>0.09</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>avg</td>
<td></td>
<td>0.64</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1.00</td>
<td>[1.52, 1.56]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>whetstone</td>
<td>1</td>
<td>2.18</td>
<td>1.71</td>
<td>0.07</td>
<td>82</td>
<td>1</td>
<td>2.66*</td>
<td>1.67</td>
<td>0.09</td>
<td>86</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>1.99</td>
<td>1.67</td>
<td>0.06</td>
<td>87</td>
<td>2</td>
<td>3.24</td>
<td>1.71</td>
<td>0.16</td>
<td>58</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>1.97*</td>
<td>1.69</td>
<td>0.08</td>
<td>89</td>
<td>3</td>
<td>3.07</td>
<td>1.77</td>
<td>0.15</td>
<td>63</td>
<td></td>
</tr>
<tr>
<td>avg</td>
<td></td>
<td>2.05</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2.99</td>
<td>[1.35, 1.46]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>integer arith</td>
<td>1</td>
<td>1.71</td>
<td>1.33</td>
<td>0.05</td>
<td>81</td>
<td>1</td>
<td>2.97</td>
<td>1.44</td>
<td>0.09</td>
<td>52</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>1.21</td>
<td>1.30</td>
<td>0.04</td>
<td>83</td>
<td>2</td>
<td>2.62</td>
<td>1.33</td>
<td>0.14</td>
<td>56</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>1.81*</td>
<td>1.30</td>
<td>0.07</td>
<td>85</td>
<td>3</td>
<td>2.41*</td>
<td>1.38</td>
<td>0.11</td>
<td>62</td>
<td></td>
</tr>
<tr>
<td>avg</td>
<td></td>
<td>1.64</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2.67</td>
<td>[1.50, 1.62]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>real arith</td>
<td>1</td>
<td>2.46</td>
<td>2.09</td>
<td>0.08</td>
<td>88</td>
<td>1</td>
<td>2.75*</td>
<td>2.10</td>
<td>0.11</td>
<td>80</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>2.38*</td>
<td>2.08</td>
<td>0.05</td>
<td>90</td>
<td>2</td>
<td>4.01</td>
<td>2.09</td>
<td>0.15</td>
<td>56</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>2.43</td>
<td>2.02</td>
<td>0.09</td>
<td>87</td>
<td>3</td>
<td>3.58</td>
<td>2.12</td>
<td>0.18</td>
<td>64</td>
<td></td>
</tr>
<tr>
<td>avg</td>
<td></td>
<td>2.42</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3.45</td>
<td>[1.17, 1.43]</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* Marks lowest elapsed time for the particular test
** Averages of all the test ratios
* Ratios are displayed as the [lowest time ratio, average time ratio]
Comparison Report

Test executed all at once with no extra load
(Expressed in seconds)

<table>
<thead>
<tr>
<th>Type</th>
<th>Test Number</th>
<th>Elapsed Time</th>
<th>User Time</th>
<th>System Time</th>
<th>Percent Usage</th>
<th>Test Elapsed Time</th>
<th>User Time</th>
<th>System Time</th>
<th>Percent Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>composite</td>
<td>1</td>
<td>10.05</td>
<td>5.26</td>
<td>0.38</td>
<td>56.1</td>
<td>10.09</td>
<td>5.40</td>
<td>0.57</td>
<td>59.7</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>10.07</td>
<td>5.13</td>
<td>0.27</td>
<td>54.2</td>
<td>10.08</td>
<td>5.31</td>
<td>0.44</td>
<td>57.1</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>10.09</td>
<td>5.24</td>
<td>0.47</td>
<td>57.3</td>
<td>10.07*</td>
<td>5.12</td>
<td>0.42</td>
<td>55.5</td>
</tr>
<tr>
<td>avg</td>
<td></td>
<td>10.07</td>
<td></td>
<td></td>
<td></td>
<td>10.08</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>uptime</td>
<td>1</td>
<td>1.02</td>
<td>0.02</td>
<td>0.09</td>
<td>11.0</td>
<td>2.57</td>
<td>0.08</td>
<td>0.07</td>
<td>6.0</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>0.58*</td>
<td>0.05</td>
<td>0.10</td>
<td>26.2</td>
<td>1.67</td>
<td>0.02</td>
<td>0.11</td>
<td>8.8</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>1.28</td>
<td>0.01</td>
<td>0.14</td>
<td>12.3</td>
<td>1.01*</td>
<td>0.02</td>
<td>0.11</td>
<td>13.3</td>
</tr>
<tr>
<td>avg</td>
<td></td>
<td>0.98</td>
<td></td>
<td></td>
<td></td>
<td>1.75</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>whetstone</td>
<td>1</td>
<td>7.41</td>
<td>1.78</td>
<td>0.10</td>
<td>25.0</td>
<td>6.80</td>
<td>1.72</td>
<td>0.19</td>
<td>28.0</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>5.20*</td>
<td>1.69</td>
<td>0.04</td>
<td>33.1</td>
<td>7.05</td>
<td>1.70</td>
<td>0.13</td>
<td>27.0</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>7.78</td>
<td>1.70</td>
<td>0.09</td>
<td>23.3</td>
<td>6.20*</td>
<td>1.70</td>
<td>0.08</td>
<td>29.0</td>
</tr>
<tr>
<td>avg</td>
<td></td>
<td>6.80</td>
<td></td>
<td></td>
<td></td>
<td>6.71</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>integer arith</td>
<td>1</td>
<td>6.25</td>
<td>1.35</td>
<td>0.08</td>
<td>23.0</td>
<td>6.26</td>
<td>1.46</td>
<td>0.08</td>
<td>25.0</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>4.32*</td>
<td>1.29</td>
<td>0.03</td>
<td>31.2</td>
<td>5.85</td>
<td>1.36</td>
<td>0.05</td>
<td>24.0</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>8.58</td>
<td>1.37</td>
<td>0.08</td>
<td>22.3</td>
<td>5.34*</td>
<td>1.30</td>
<td>0.10</td>
<td>26.0</td>
</tr>
<tr>
<td>avg</td>
<td></td>
<td>5.71</td>
<td></td>
<td></td>
<td></td>
<td>5.82</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>real arith</td>
<td>1</td>
<td>7.88</td>
<td>2.12</td>
<td>0.08</td>
<td>28.0</td>
<td>7.78</td>
<td>2.13</td>
<td>0.17</td>
<td>30.0</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>5.73*</td>
<td>2.10</td>
<td>0.05</td>
<td>38.2</td>
<td>7.35*</td>
<td>2.17</td>
<td>0.08</td>
<td>31.0</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>8.66</td>
<td>2.16</td>
<td>0.07</td>
<td>26.3</td>
<td>6.73*</td>
<td>2.09</td>
<td>0.10</td>
<td>33.0</td>
</tr>
<tr>
<td>avg</td>
<td></td>
<td>7.42</td>
<td></td>
<td></td>
<td></td>
<td>7.29</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Average Elapsed Ratios ** [1.07, 0.98]

* Marks lowest elapsed time for the particular test
** Averages of all the test ratios
+ Ratios are displayed as the [lowest time ratio, average time ratio]
Comparison Report

Test executed all at once with an extra load (expressed in seconds)

<table>
<thead>
<tr>
<th>Type</th>
<th>Test Number</th>
<th>Elapsed Time</th>
<th>User Time</th>
<th>System Time</th>
<th>Percent Usage</th>
<th>Test Number</th>
<th>Elapsed Time</th>
<th>Elapsed Ratio*</th>
<th>User Time</th>
<th>System Time</th>
<th>Percent Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>composite</td>
<td>1</td>
<td>10.04</td>
<td>5.11</td>
<td>0.30</td>
<td>54</td>
<td>1</td>
<td>15.11</td>
<td></td>
<td>5.20</td>
<td>0.42</td>
<td>37</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>10.08</td>
<td>5.09</td>
<td>0.28</td>
<td>53</td>
<td>2</td>
<td>10.11</td>
<td></td>
<td>5.39</td>
<td>0.47</td>
<td>58</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>10.04*</td>
<td>5.13</td>
<td>0.23</td>
<td>53</td>
<td>3</td>
<td>10.11*</td>
<td></td>
<td>5.21</td>
<td>0.51</td>
<td>57</td>
</tr>
<tr>
<td></td>
<td>avg</td>
<td>10.05</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>11.78</td>
<td>[1.01, 1.17]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>uptime</td>
<td>1</td>
<td>0.71*</td>
<td>0.02</td>
<td>0.09</td>
<td>15</td>
<td>1</td>
<td>0.80*</td>
<td></td>
<td>0.08</td>
<td>0.05</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>0.78</td>
<td>0.01</td>
<td>0.11</td>
<td>16</td>
<td>2</td>
<td>1.03</td>
<td></td>
<td>0.04</td>
<td>0.10</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.76</td>
<td>0.07</td>
<td>0.08</td>
<td>17</td>
<td>3</td>
<td>1.01</td>
<td></td>
<td>0.07</td>
<td>0.08</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td>avg</td>
<td>0.74</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.95</td>
<td>[1.13, 1.27]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>whetstone</td>
<td>1</td>
<td>7.45</td>
<td>1.70</td>
<td>0.05</td>
<td>23</td>
<td>1</td>
<td>9.83</td>
<td></td>
<td>1.72</td>
<td>0.06</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>6.49*</td>
<td>1.69</td>
<td>0.03</td>
<td>27</td>
<td>2</td>
<td>7.92</td>
<td></td>
<td>1.80</td>
<td>0.07</td>
<td>24</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>6.70</td>
<td>1.75</td>
<td>0.01</td>
<td>26</td>
<td>3</td>
<td>7.43*</td>
<td></td>
<td>1.76</td>
<td>0.10</td>
<td>25</td>
</tr>
<tr>
<td></td>
<td>avg</td>
<td>6.88</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>8.39</td>
<td>[1.14, 1.22]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>integer arith</td>
<td>1</td>
<td>5.49</td>
<td>1.35</td>
<td>0.03</td>
<td>25</td>
<td>1</td>
<td>7.02</td>
<td></td>
<td>1.29</td>
<td>0.11</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>4.05*</td>
<td>1.27</td>
<td>0.03</td>
<td>27</td>
<td>2</td>
<td>6.45</td>
<td></td>
<td>1.36</td>
<td>0.12</td>
<td>23</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>5.03</td>
<td>1.30</td>
<td>0.04</td>
<td>27</td>
<td>3</td>
<td>5.43*</td>
<td></td>
<td>1.34</td>
<td>0.09</td>
<td>26</td>
</tr>
<tr>
<td></td>
<td>avg</td>
<td>5.12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>6.60</td>
<td>[1.12, 1.29]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>real arith</td>
<td>1</td>
<td>9.09</td>
<td>2.04</td>
<td>0.07</td>
<td>23</td>
<td>1</td>
<td>10.04</td>
<td></td>
<td>2.11</td>
<td>0.14</td>
<td>22</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>8.40</td>
<td>2.12</td>
<td>0.08</td>
<td>26</td>
<td>2</td>
<td>9.23</td>
<td></td>
<td>2.16</td>
<td>0.12</td>
<td>25</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>8.21*</td>
<td>2.01</td>
<td>0.08</td>
<td>25</td>
<td>3</td>
<td>8.52*</td>
<td></td>
<td>2.04</td>
<td>0.17</td>
<td>26</td>
</tr>
<tr>
<td></td>
<td>avg</td>
<td>8.57</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>9.26</td>
<td>[1.04, 1.08]</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* Marks lowest elapsed time for the particular test
** Averages of all the test ratios
+ Ratios are displayed as the [lowest time ratio, average time ratio]
Summary Report

Elapsed time averages normalized to the baseline

<table>
<thead>
<tr>
<th></th>
<th>(baseline)</th>
<th>DIGITAL</th>
<th>DIGITAL</th>
<th>DEC</th>
<th>DEC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>VAX 11/780</td>
<td>VAX 11/730</td>
<td>VAX 8600</td>
<td>VAX 11/750</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4.3 BSD</td>
<td>4.3 BSD</td>
<td>4.3 BSD</td>
<td>4.1 BSD</td>
<td></td>
</tr>
<tr>
<td>composite</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>one at a time</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>all at once, no load</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>all at once, w/ load</td>
<td>1.0</td>
<td>1.2</td>
<td>1.0</td>
<td>1.3</td>
<td></td>
</tr>
<tr>
<td>uptime</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>one at a time</td>
<td>1.0</td>
<td>1.6</td>
<td>1.1</td>
<td>3.2</td>
<td></td>
</tr>
<tr>
<td>all at once, no load</td>
<td>1.0</td>
<td>1.8</td>
<td>0.7</td>
<td>2.5</td>
<td></td>
</tr>
<tr>
<td>all at once, w/ load</td>
<td>1.0</td>
<td>1.3</td>
<td>1.5</td>
<td>6.4</td>
<td></td>
</tr>
<tr>
<td>wheatsone</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>one at a time</td>
<td>1.0</td>
<td>1.5</td>
<td>1.1</td>
<td>1.4</td>
<td></td>
</tr>
<tr>
<td>all at once, no load</td>
<td>1.0</td>
<td>1.0</td>
<td>0.8</td>
<td>1.8</td>
<td></td>
</tr>
<tr>
<td>all at once, w/ load</td>
<td>1.0</td>
<td>1.2</td>
<td>1.2</td>
<td>1.3</td>
<td></td>
</tr>
<tr>
<td>integer arith</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>one at a time</td>
<td>1.0</td>
<td>1.0</td>
<td>1.1</td>
<td>1.5</td>
<td></td>
</tr>
<tr>
<td>all at once, no load</td>
<td>1.0</td>
<td>1.0</td>
<td>0.8</td>
<td>1.6</td>
<td></td>
</tr>
<tr>
<td>all at once, w/ load</td>
<td>1.0</td>
<td>1.3</td>
<td>1.3</td>
<td>1.5</td>
<td></td>
</tr>
<tr>
<td>real arith</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>one at a time</td>
<td>1.0</td>
<td>1.4</td>
<td>1.3</td>
<td>1.6</td>
<td></td>
</tr>
<tr>
<td>all at once, no load</td>
<td>1.0</td>
<td>1.0</td>
<td>0.8</td>
<td>1.8</td>
<td></td>
</tr>
<tr>
<td>all at once, w/ load</td>
<td>1.0</td>
<td>1.1</td>
<td>1.0</td>
<td>1.2</td>
<td></td>
</tr>
</tbody>
</table>
NAME

bench - benchmark driver and result comparison generator

SYNOPSIS

bench -b [-v] [-s] [-l] [-n "make, model, version"]
bench -p [-s] [-f] [-l] [-o outfile]
bench -e [-s] [-f] [-l] [-o outfile]

DESCRIPTION

Bench is a benchmark driver program to time the execution of a suite of tests specified in the ASCII file benchlist. The defaults to bench are designed to allow a user with little or no understanding of the options to establish a baseline system and create comparisons of other systems with the baseline.

The user can override the defaults by using the options. For example: data can be collected without generating a comparison report; an output filename can be specified for the comparison report; a summary table of all the system results can be generated; and selected groups of tests can be executed without running the complete suite.

The available flags are:

- **-b** Execute the test suite and add the results to the database as the baseline from which comparisons will be produced.

- **-a** Execute the test suite and add the results to the database. Unless used with the -o option, no comparison report will be generated.

- **-p** Prepare a comparison report between two systems of the user's choice. May be used with the -o option; default is to the line printer.

- **-e** Generate a summary table containing normalized elapsed times for all systems in the database. If no system has been assigned as the baseline, the user will be prompted for a system to use as a baseline. Default is to the line printer.

- **-v** Verbose: causes output to be generated to standard output. This information is helpful when trying to follow the progress of the driver. Default is off.

- **-n "make, model, version"**
  Use the make, model, and version of the system to identify the results. This option is useful when executing the driver in a batch mode. If this is not specified on the command line, the user will be prompted for make, model, and version.

- **-o outfile**
  Name the formatted output file outfile. By default the output file is created by adding the last three digits of the process id to /tmp/bench.
The presence of any of the -s, -f, and -l flags cause the execution to be limited to only what is specified. (If -s, -f, or -l are not specified, the default sets all three flags.)

-s  This flag causes tests to be executed one after the other with no extra load added to the system.

-f  This flag causes simultaneous execution of the tests with no extra load added to the system.

-l  This flag causes simultaneous execution of the tests with extra load added to the system at the same time.

EXAMPLES

Execute the test suite on the current machine and store the results as the baseline.

% bench -b

Execute the test suite on the current machine and store the results. Also input the make, model, and version from the command line.

% bench -a -n "DEC, VAX 8600, 4.3 BSD"

Print a summary of current database into outfile.

% bench -e -o outfile

FILES

/tmp/benchXXX  formatted output of the comparison
baseline      result data of baseline system
benchlist     the path of test and the printable name
targetXXX     result data of system to be compared to baseline

BUGS
Instructions for Data Collection Using the Bench Program

Before bench can be used on any system, instructions 1 through 4 must be completed. All tar instructions are assuming 1600 bpi on drive 0.

1. Mount tape on drive 0 at 1600 bpi.
2. Change to a working directory with at least 3,000 blocks free.
3. To unload the tape, type:
   \% tar xv
4. To compile the driver program and test suite, type:
   \% make
   It may be necessary to edit the bench.mk file to alter the names for the different compilers with optimizers on.

After completing steps 1-4 above, any of the remaining sections can be followed to collect data or display previously collected data.

To establish a baseline and store the results on the tape:
1. Type:
   \% bench -b
   a. Enter the make, model, and version of the system when prompted by the program.
   b. Sit back and relax.
2. To store the baseline file on the tape, type:
   \% tar u baseline
3. Remove all working files and directories from the disk if desired.

To add a target system to the database:
1. Type:
   \% bench -a
   a. Enter the system description when prompted by the program.
   b. Sit back and relax.
2. To store target system results on tape, type:
   \% tar u target*
   If a print-out is desired, skip to one of the last two sections.
3. Remove all working files and directories if desired.
To print a comparison report between two systems in the database:

1. Type:

   \% bench -p

   A list of systems in the database will appear preceded by a number. The system will prompt you for two numbers to indicate the two systems to be compared. The output will be sent to the line printer.

2. Remove all working files and directories if desired.

To print a summary report of all systems in the database:

1. Type:

   \% bench -e

   The output will be sent to the line printer.

2. Remove all working files and directories if desired.
Prime Numbers

C prime.f
C C PROGRAM TO GENERATE PRIME NUMBERS
C C Compile by: fort -O prime.f -o prime
C
C PROGRAM PRIME
C COMMON/DAT/VALUE(8192)
C INITILIZE DATA STRUCTURES
C
ILUM=6
IPRT=0
ICRT=512
CUR=2.
TOP=3.
I=1
C CHECK REMAINDER
C 5 IF(AMOD(TOP,CUR).EQ.0.)GO TO 10
           CUR=CUR+1.
           IF(CUR.LT.TOP)GO TO 5
C IF WE SCAN FROM 2 THRU TOP, THEN TOP IS A PRIME NUMBER
C
VALUE(I)=TOP
I=I+1
C SET UP FOR NEXT PRIME NUMBER
C 10 TOP=TOP+2.
    CUR=2.
    IF(I.LE.ICRT)GO TO 5
C PRINT THE PRIME NUMBERS WEVE GENERATED
C
IF(IPRT.EQ.0)STOP
DO 15 I=1,ICRT,8
    WRITE(ILUM,9010)(VALUE(J),J=I,I+7)
9010 FORMAT(8F10.0)
15 CONTINUE
STOP
END
Calling Sequence and Arguments Passing

C calseq.f
C
C PROGRAM TO TEST CALLING SEQUENCE AND ARGUMENT PASSING
C
C Compile by: fort -O calseq -o calseq
C
C
PROGRAM CALSEQ
C
Z=0.
10 I=0
J=1
K=2
L=3
A=0.
B=1.
C=2.
D=3.
CALL CALSEQ1(A, I, B, J, C, K, D, L)
Z=Z+1.
IF(Z.LT.1.E6)GO TO 10
STOP
END
C
SUBROUTINE CALSEQ1(A, I, B, J, C, K, D, L)
A=D
B=C
I=J
K=L
RETURN
END
Random Numbers

C rndsk.f

C PROGRAM TO PERFORM A CHECK OF THE RANDOM NUMBER GENERATOR
C BY PERFORMING DIRECT ACCESS TO A DISK FILE.
C THE SUBROUTINE WILL USE A RANDOM NUMBER FROM 1 TO 256
C AS THE KEY TO READ A RECORD, INCREMENT THE VALUE READ,
C AND WRITE THE NEW VALUE.
C
C Compile by: fort -O rndsk.f -o rndsk
C
C PROGRAM RNDISK
C
C    ILUN=6
C    IPRT=0
C    ICNT=128
C    FCNT=FLOAT(ICNT)
C    FCHK=FCNT*100.
C    IRAN=0
C    isize = 4
C    B=rand(IRAN)
C    OPEN(ACCESS='DIRECT',
C    1FILE='TEST',
C    2FORM='UNFORMATTED',
C    3MAXREC=ICNT+1,
C    4RECL=isize*2,
C    5STATUS='UNKNOWN',
C    6UNIT=4)
C
C CREATE FILE WITH EACH RECORD CONTAINING ALL ZEROS
C
DO 10 I=1,ICNT
C    IREC=I
C    WRITE( 4,rec=IREC)FLOAT( IREC),0.
C :CONTINUE
C
C GENERATE ICNT=100 RANDOM NUMBERS
C
C    A=0.
20    IREC=IFIX(FCNT*rand(IRAN))+1
C    IF(IREC.GE.1.AND.IREC.LE.ICNT)GO TO 25
C    WRITE(ILUN,9010)IREC
9010 FORMAT(' RANDOM NUMBER OUT OF RANGE',I6)
25    I=IREC
C    READ(4,rec=I)RNUM,COUNT
C    COUNT=COUNT+1.
C    I=IREC
C    WRITE(4,rec=I)RNUM,COUNT
C    A=A+1.
C    IF(A.LT.FCHK)GO TO 20
C
C READ FILE, GET MIN, MAX AND AVERAGE OF RANDOM NUMBER GENERATOR
C
C    AMIN=9999.
C    AMAX=0.

19
Random Numbers

AVE=0.
DO 30 I=1,ICN I
IREC=I
READ(4,rec=IREC) RNUM,COUNT
IF(COUNT.GT.AMAX) AMAX=COUNT
IF(COUNT.LT.AMIN) AMIN=COUNT
AVE=AVE+COUN T
30 CONTINUE
CLOSE(UNIT=4)
AVE=AVE/PCNT
IF(IPRT.EQ.0) STOP
WRITE(ILUN,9000)AMIN,AMAX,AVE
9000 FORMAT(3F15.0)
STOP
END
Fast Fourier Transform

C fft.f
C
C PROGRAM TO PERFORM A FAST FOURIER TRANSFORM USING THE
C DECIMATION-IN-TIME METHOD.
C
C Compile by: fort -O fft.f -o fft
C
PROGRAM FFT
C
COMMON/DAT/A(4096)
COMPLEX A,U,W,T
C
INITIALIZE
C
ILUM=6
IPRT=0
DO 25 LOOP=1,10
M=12
ICNT=2**M
PER=FLOAT(ICNT/16)
PI=3.141592653589793
DO 1 I=1,ICNT
B=SIN(2.*PI*FLOAT(I)/PER)
A(I)=CMPLX(B,0.)
1 CONTINUE
NWV?-N/2
NM'=N-1
J=1
DO 7 I=1,NM1
IF(I.GE.J)GO TO 5
T=A(J)
A(J)=A(I)
A(I)=T
5 K=NW2
IF(K.GE.J)GO TO 7
J=J-K
K=K/2
GO TO 6
7 J=J+K
PI=3.141592653589793
DO 20 L=1,M
LE=2**L
LE1=LE/2
U=(1.,0.)
W=CMPLX(COS(PI/FLOAT(LE1)),SIN(PI/FLOAT(LE1)))
DO 20 J=1,LE1
DO 10 I=J,M,LE
IP=I+LE1
T=A(IP)*U
A(IP)=A(I)-T
A(I)=A(I)+T
10 T=I+LE1
U=U*W
25 CONTINUE
IF(IPRT.EQ.0)STOP
Fast Fourier Transform

DO 30 I=1,128,4
WRITE(ILUN,9000)(A(J),J=I,I+3)
9000 FORMAT(4G15.6)
30 CONTINUE
STOP
END
Matrix Inversion

C matrix.f

C MATRIX INVERSION USING GAUSS-JORDAN REDUCTION
C INVERTED MATRIX OVERLAYS ORIGINAL MATRIX IN MEMORY
C PARITAL PIVOTING IS NOT USED

C Compile by; fort -O matrix -o matrix

PROGRAM MATRIX

COMMON/DAT/A(15,15)
DOUBLE PRECISION A
ILUM=6
IPRT=0
DO 10 LOOP=1,10000
N=4
A(1,1)=1.
A(1,2)=1.
A(1,3)=1.
A(1,4)=1.
A(2,1)=4.
A(2,2)=5.
A(2,3)=6.
A(2,4)=7.
A(3,1)=6.
A(3,2)=10.
A(3,3)=15.
A(3,4)=21.
A(4,1)=12.
A(4,2)=30.
A(4,3)=60.
A(4,4)=105.

C CALCULATE ELEMENTS OF REDUCED MATRIX
C DO 6 K=1,N
C CALCULATE NEW ELEMENTS OF PIVOT ROW
C DO 4 J=1,N
IF(J.EQ.K)GO TO 4
A(K,J)=A(K,J)/A(K,K)
4 CONTINUE
C CALCULATE ELEMENT REPLACING PIVOT ELEMENT
C A(K,K)=1./A(K,K)
C CALCULATE NEW ELEMENTS NOT IN PIVOT ROW OR PIVOT COLUMN
C DO 5 I=1,N
IF(I.EQ.K)GO TO 5
DO 5 J=1,N
IF(J.EQ.K)GO TO 5
Matrix Inversion

\[ A(I,J) = A(I,J) - A(K,J)*A(I,K) \]

CONTINUE

CALCULATE REPLACEMENT ELEMENTS FOR PIVOT COLUMN-EXCEPT PIVOT ELEMENT

DO 6 I=1,N
IF(I.EQ.K)GO TO 6
A(I,K) = -A(I,K) * A(K,K)
6 CONTINUE

CONTINUE

OUTPUT INVERTED MATRIX

IF(IPRT.EQ.0)STOP
WRITE(ILUN,8)((A(I,J),J=1,N),I=1,N)
8 FORMAT(4F16.4)
STOP
END
Polynomial Roots

```
C roots.f
C
C ROOTS OF POLYNOMIAL BY BAIRSTOWS METHOD
C Compile by: fort -O roots.f -o roots
C
PROGRAM ROOTS
C
DIMENSION A(30),B(30),C(30)
ILUN=6
IPRT=0
IF(IPRT.EQ.0)GO TO 200
IPRT=0
JPRT=1
200 DO 100 LOOP=1,10000
IF(LOOP.NE.10000)GO TO 220
IF(JPRT.EQ.0)GO TO 220
IPRT=1
220 UI=0.
VI=0.
EPSI=1.E-6
N=5
A(1)=-3.
A(2)=-10.
A(3)=10.
A(4)=44.
A(5)=48.

C SEE IF N=0,1, OR GREATER THAN 1
C
40 IF(N-1).LE.0,5,7
5 P=A(1)
Q=0.
IT=1
IF(IPRT.NE.0)WRITE(ILUN,6)N,P,Q,IT
6 FORMAT(' X(','12,') =','2X,F8.4,6X,F8.4,10X,I3)
GO TO 100
C
C SEE IF N=2 OR IF N IS GREATER THAN 2
C
7 IF(N.EQ.2).GO TO 8
GO TO 13
8 U=A(1)
V=A(2)
IT=1
9 P=-U/2.
RAD=U**2-4.*V
C
C CHECK THE SIGN OF U**2-4.*V
C
IF(RAD.GT.0.)GO TO 12
RAD=-RAD
Q=SQRT(RAD)/2.
IF(IPRT.NE.0)WRITE(ILUN,6)N,P,Q,IT
```
Polyomial Roots

M=N-1
Q=Q
90 IF(IPRT.NE.0)WRITE(ILUN,6)N,P,Q,IT
10 N=M-1
C CHECK TO SEE IF N IS GREATER THEN ZERO
C IF(N.LE.0)GO TO 100
DO 11 I=1,N
11 A(I)=B(I)
GO TO 40
12 Q=SQRT(RAD)/2.
W=P
Z=Q
P=P+Q
Q=0.
IF(IPRT.NE.0)WRITE(ILUN,6)N,P,Q,IT
N=M-1
P=W-2
GO TO 90
13 U=U
V=V
IT=1
C C CALCULATE THE B VALUES
C 50 B(1)=A(1)-U
B(2)=A(2)-B(1)*U-V
DO 14 K=3,N
14 B(K)=A(K)-B(K-1)*U-B(K-2)*V
C C CALCULATE THE C VALUES
C 50 C(1)=B(1)-U
C(2)=B(2)-C(1)*U-V
M=M-1
DO 15 K=3,N
15 C(K)=B(K)-C(K-1)*U-C(K-2)*V
C C CALCULATE DELU AND DELV
C IF(N.GT.3)GO TO 17
DENOM=C(N-1)-C(N-2)**2
IF(DENOM.EQ.0.)GO TO 30
DELU=(B(N)-B(N-1)*C(N-2))/DENOM
16 DELV=(C(N-1)*B(N-1)-C(N-2)*B(N))/DENOM
GO TO 18
17 DENOM=C(N-1)*C(N-3)-C(N-2)**2
IF(DENOM.EQ.0.)GO TO 30
DELU=(B(N)*C(N-3)-B(N-1)*C(N-2))/DENOM
GO TO 16
C C CALCULATE NEW U AND V VALUES
C
Polynomial Roots

18  U = U + DELU
    V = V + DELV
    SUM = ABS(DELU) + ABS(DELV)

C  STORE THE FIRST SUM CALCULATED
C
    IF(IT.EQ.1) GO TO 19
    GO TO 20
19  STORE = SUM
    GO TO 21
20  IF(IT.EQ.50) GO TO 28
    IF(IT.GE.200) GO TO 26
21  IF(SUM.LE.EPSI) GO TO 9
    IF(IT.EQ.100) GO TO 23
    IT = IT + 1
    GO TO 50
22  IF(IPRT.NE.0) WRITE(ILUN,24)
23  FORMAT(' CONVERGENCE IS SLOW')
    IF(IPRT.NE.0) WRITE(ILUN,25) U, V
24  FORMAT(' U=',E14.7,' V=',E14.7)
    GO TO 22
25  IF(IPRT.NE.0) WRITE(ILUN,27)
26  FORMAT(' STOPPED AFTER 200 ITERATIONS')
    IF(IPRT.NE.0) WRITE(ILUN,25) U, V
    GO TO 100
C
C  SEE IF SUM AFTER 50 ITERATIONS EXCEEDS FIRST SUM STORED
C
28  IF(SUM.LT.STORE) GO TO 21
    IF(IPRT.NE.0) WRITE(ILUN,29)
29  FORMAT(' DIVERGENCE OCCURRING')
    IF(IPRT.NE.0) WRITE(ILUN,25) U, V
    GO TO 100
30  IF(IPRT.NE.0) WRITE(ILUN,31)
31  FORMAT(' DENOMINATOR IS ZERO')
    GO TO 100
100  CONTINUE
STOP
END
C Sieve

/*
 * sieve.c
 *
 * Eratosthenes Sieve Prime Number Program in C */
 * Compile by: cc -O sieve.c -o csieve
 * */

#define true 1
#define false 0
#define size 8190

char flags[size + 1];

main() {
    int i, prime, k, count, iter;

    printf("100 iterations\n");
    for(iter = 1; iter <= 100; iter++) {
        count = 0;
        for(i = 0; i <= size; i++)
            flags[i] = true;
        for(i = 0; i <= size; i++) {
            if(flags[i]) {
                prime = i + i + 3;
                for(k = i + prime; k <= size; k += prime)
                    flags[k] = false;
                count++;
            }
        }
    }
    printf("%d is largest of %d primes.\n", prime, count);
}

Fortran Sieve

c sieve.f

eratosthenes sieve with Knuth's optimization

c Compile by: fort -O sieve.f -o fsieve

c
integer i,j,k,iter,prime,count
logical flags(8191),last

write(6,10)
10 format ('100 iterations')
do 20 iter = 1, 100
    count = 0
    do 30 i = 1, 8191
        flags(i) = .true.
        last = .false.
        do 40 i = 1, 8191
            if (.not. flags(i)) go to 50
            prime = i + i + 1
            count = count + 1
            write(6,11) prime
11 format (lx,i6)
        if (last) go to 50
            k = (prime*prime - 1) / 2
            k = i + prime
            do 60 j = k, 8191, prime
                flags(j) = .false.
                if (prime .ge. 127) last = .true.
50 continue
40 continue
20 continue
write(6,12) count
12 format (lx, i6, ' primes')
end
Pascal Sieve

(* sieve.p *)

(* Eratosthenes Sieve Prime Number Program in Pascal *)

(* Compile by: pi sieve.p *)

program prime(output);

const
  size = 8190;

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

begin
  writeln('100 iterations');
  for iter := 1 to 100 do begin
    cnt := 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;
        cnt := cnt + 1
      end;
  end;
  writeln(cnt,' primes')
end.


Whetstone

/*
* Whetstone benchmark in C. This program is a translation of the
* original Algol version in "A Synthetic Benchmark" by H.J. Curnow
* Used to test compiler optimization and floating point performance.
* Compile by: cc -O -s -o whet whet.c
* or: cc -O -DPOUT -s -o whet whet.c
* if output is desired.
*/

#define ITERATIONS 10 /* 1 Million Whetstone instructions */

#include "math.h"

double x1, x2, x3, x4, x, y, z, t, t1, t2;
double el[4];
int i, j, k, l, n1, n2, n3, n4, n6, n7, n8, n9, n10, n11;

main()

    /* initialize constants */
    t = 0.499975;
    t1 = 0.50025;
    t2 = 2.0;

    /* set values of module weights */
    n1 = 0 * ITERATIONS;
    n2 = 12 * ITERATIONS;
    n3 = 14 * ITERATIONS;
    n4 = 345 * ITERATIONS;
    n6 = 210 * ITERATIONS;
    n7 = 32 * ITERATIONS;
    n8 = 899 * ITERATIONS;
    n9 = 616 * ITERATIONS;
    n10 = 0 * ITERATIONS;
    n11 = 93 * ITERATIONS;

    /* MODULE 1: simple identifiers */
    x1 = 1.0;
    x2 = x3 = x4 = -1.0;

    for(i = 1; i <= n1; i *= 1) /
        x1 = (x1 + x2 + x3 + x4) * t;
        x2 = (x1 + x2 - x3 - x4) * t;
        x3 = (x1 - x2 + x3 + x4) * t;
        x4 = (-x1 + x2 + x3 + x4) * t;

}
Whetstone

ifdef POUT
    pout(n1, n1, n1, x1, x2, x3, x4);
endif

/* MODULE 2: array elements */

e1[0] = 1.0;

for (i = 1; i <= n2; i *= 1) {
}

ifdef POUT
    pout(n2, n3, n2, e1[0], e1[1], e1[2], e1[3]);
endif

/* MODULE 3: array as parameter */

for (i = 1; i <= n3; i += 1)
    pa(e1);

ifdef POUT
    pout(n3, n2, n2, e1[0], e1[1], e1[2], e1[3]);
endif

/* MODULE 4: conditional jumps */

j = 1;
for (i = 1; i <= n4; i += 1) {
    if (j == 1)
        j = 2;
    else
        j = 3;
    if (j > 2)
        j = 0;
    else
        j = 1;
    if (j < 1)
        j = 1;
    else
        j = 0;
}

ifdef POUT
    pout(n4, j, j, x1, x2, x3, x4);
endif

/* MODULE 5: omitted */

/* MODULE 6: integer arithmetic */
Whetstone

j = 1;
k = 2;
l = 3;

for (i = 1; i <= n6; i += 1) {
  j = j * (k - j) * (l - k);
  k = l * k - (l - j) * k;
  l = (l - k) * (k + j);
  el[i - 2] = j + k + l;
  el[k - 2] = j * k + l;
  /* C arrays are zero based */
}

#ifdef POUT
  pout(n6, j, k, el[0], el[1], el[2], el[3]);
#endif

/* MODULE 7: trig. functions */

x = y = 0.5;

for (i = 1; i <= n7; i += 1) {
  x = t * atan(t2*sin(x)*cos(x)/(cos(x+y)+cos(x-y)-1.0));
  y = t * atan(t2*sin(y)*cos(y)/(cos(x+y)+cos(x-y)-1.0));
}

#ifdef POUT
  pout(n7, j, k, x, x, y, y);
#endif

/* MODULE 8: procedure calls */

x = y = z = 1.0;

for (i = 1; i <= n8; i += 1)
  p3(x, y, &z);

#ifdef POUT
  pout(n8, j, k, x, y, z, z);
#endif

/* MODULE 9: array references */

j = 1;
k = 2;
l = 3;

el[0] = 1.0;
el[1] = 2.0;
el[2] = 3.0;

for (i = 1; i <= n9; i += 1)
  p0();

#ifdef POUT
  pout(n9, j, k, el[0], el[1], el[2], el[3]);
#endif
/* MODULE10: integer arithmetic */

    j = 2;
    k = 3;
    for(i = 1; i <= n10; i++) {
        j = j + k;
        k = j + k;
        j = k - j;
        k = k - j - j;
    }

#ifdef POUT
    pout(n10, j, k, x1, x2, x3, x4);
#endif

/* MODULE11: standard functions */

    x = 0.75;
    for(i = 1; i <= n11; i++)
        x = sqrt( exp( log(x) / t1));

#ifdef POUT
    pout(n11, j, k, x, x, x, x);
#endif

exit(0);

pa(e)
double e[4]:
{
    register int j;
    j = 0;
    lab:
        j += 1;
        if (j < 6)
            goto lab;
}

p3(x, y, z)
double x, y, *z;
{
    x = t * (x + y);
    y = t * (x + y);
    *z = (x + y) / t2;
}
p0()
{
    e1[j] = e1[k];
    e1[k] = e1[l];
    e1[l] = e1[j];
}

#define POUT
pout(n, j, k, x1, x2, x3, x4)
int n, j, k;
double x1, x2, x3, x4;
{
    printf("%6d%6d%6d %5e %5e %5e\n", n, j, k, x1, x2, x3, x4);
}
#endif
Dhrystone

dry.c

"DHRYSTONE" Benchmark Program

Version: C/1.1, 12/01/84

Date: PROGRAM updated 01/06/86, RESULTS updated 02/17/86

Author: Reinhold P. Weicker, CACM Vol 27, No 10, 10/84 pg. 1013
Translated from ADA by Rick Richardson
Every method to preserve ADA-likeness has been used, at the expense of C-ness.

Compile:
cc -O dry.c -o drynr : No registers
cc -O -DREG=register dry.c -o dryr : Registers

Run:
dyrnr; dryr

The following program contains statements of a high-level programming language (C) in a distribution considered representative:

assignments 53%
control statements 32%
procedure, function calls 15%

100 statements are dynamically executed. The program is balanced with respect to the three aspects:
- statement type
- operand type (for simple data types)
- operand access
  operand global, local, parameter, or constant.

The combination of these three aspects is balanced only approximately.

The program does not compute anything meaningful, but it is syntactically and semantically correct.

/*
/* Accuracy of timings and human fatigue controlled by next two lines */
#define LOOPS 50000 /* Use this for slow or 16 bit machines */
#define LOOPS 500000 /* Use this for faster machines */

/* Compiler dependent options */
#undef NOENUM /* Define if compiler has no enum's */
#undef NOSTRUCTASSIGN /* Define if compiler can't assign structures */

/* define only one of the next two defines */
#define TIMES /* Use times(2) time function */
#undef TIMES /* Use time(2) time function */

/* define the granularity of your times(2) function (when used) */
#define HZ 60 /* times(2) returns 1/60 second (most) */
#undef HZ 100 /* times(2) returns 1/100 second (WECO) */

/* for compatibility with goofed up version */
#undef GOOF /* Define if you want the goofed up version */

#define GOOF char Version[] = "1.0";
#undef GOOF char Version[] = "1.1";

#define structassign(d, s) memcpy(&(d), &(s), sizeof(d))
#undef structassign(d, s) d = s

#define NOENUM
#define Ident1 1
#define Ident2 2
#define Ident3 3
#define Ident4 4
#define Ident5 5
typedef int Enumeration;
#undef NOENUM
typedef enum {Ident1, Ident2, Ident3, Ident4, Ident5} Enumeration;
typedef int OneToThirty;
typedef int OneToFifty;
typedef char CapitalLetter;
typedef char String30[31];
typedef int Array1Dim[51];
typedef int Array2Dim[51][51];

struct Record {
    struct Record *PtrComp;
    Enumeration Discr;
    Enumeration EnumComp;
}
Dhrystone

OneToFifty
String30

};

typedef struct Record RecordType;
typedef RecordType * RecordPtr;
typedef int boolean;

#define NULL 0
#define TRUE 1
#define FALSE 0

#endif REG
#define REG
#endif

extern Enumeration Func1();
extern boolean Func2();

#ifndef TIMES
#include <sys/types.h>
#include <sys/times.h>
#endif

main()
{
    ProcO();
    exit(0);
}

/*
 * Package 1
 */

int IntGlob;
boolean BoolGlob;
char Char1Glob;
char Char2Glob;
Array1Dim Array1Glob;
Array2Dim Array2Glob;
RecordPtr PtrGlb;
RecordPtr PtrGlbNext;

ProcO()
{
    OneToFifty IntLoc1;
    OneToFifty IntLoc2;
    OneToFifty IntLoc3;
    REG char CharLoc;
    REG char CharIndex;
    Enumeration EnumLoc;
    String30 String1Loc;
    String30 String2Loc;
    extern char *malloc();
Dhrystone

#ifdef TIME
long time();
long starttime;
long benchtime;
long nulltime;
register unsigned int i;

starttime = time( (long *) 0);
for (i = 0; i < LOOPS; ++i);
nulltime = time( (long *) 0) - starttime; /* Computes o'head of loop */
#endif

#ifdef TIMES

time_t starttiine;
time_t benchtime;
time_t nulltime;
struct tms tms;
register unsigned int i;

times(&tms); starttiine = tms.tms_utime;
for (i = 0; i < LOOPS; ++i);
times(&tms);
nulltime = tms.tms_utime - starttime; /* Computes overhead of looping */
#endif

PtrGlbNext = (RecordPtr) malloc(sizeof(RecordType));
PtrGlb = (RecordPtr) malloc(sizeof(RecordType));
PtrGlb->PtrComp = PtrGlbNext;
PtrGlb->Discr = Ident1;
PtrGlb->EnumComp = Ident3;
PtrGlb->IntComp = 40;
strcpy(PtrGlb->StringComp, "DHRYSTONE PROGRAM, SOME STRING");
#ifndef GOOF
strcpy(String1Loc, "DHRYSTONE PROGRAM, 1' ST STRING"); /*GOOF*/
#endif

Array2Glob[8][7] = 10; /* Was missing in published program */

raries
-- Start Timer --
# ifdef TIME
starttiine = time( (long *) 0);
#endif

#include TIMES
#include TIMES

for (i = 0; i < LOOPS; ++i) {

Proc5();
Proc4();
IntLoc1 = 2;
IntLoc2 = 3;
strcpy(String2Loc, "DHRYSTONE PROGRAM, 2' ND STRING");
EnumLoc = Ident2;

}
Dhrystone

BoolGlob = ! Func2(String1Loc, String2Loc);
while (IntLoc1 < IntLoc2)
{
    IntLoc3 = 5 * IntLoc1 - IntLoc2;
    Proc7(IntLoc1, IntLoc2, &IntLoc3);
    ++IntLoc1;
}
Proc8(Array1Glob, Array2Glob, IntLoc1, IntLoc3);
Proc1(PtrGlb);
for (CharIndex = 'A'; CharIndex <= Char2Glob; ++CharIndex)
    if (EnumLoc == Func1(CharIndex, 'C'))
        Proc6(Ident1, &EnumLoc);
IntLoc3 = IntLoc2 * IntLoc1;
IntLoc2 = IntLoc3 / IntLoc1;
IntLoc2 = 7 * (IntLoc3 - IntLoc2) - IntLoc1;
Proc2(&IntLoc1);

/***************
-- Stop Timer --
*************** /

#ifndef TIME
    benchtime = time( (long *) 0) - starttime - nulltime;
    printf("Dhrystone(%s) time for %ld passes = %ld
", 
        Version,
        (long) LOOPS, benchtime);
    printf("This machine benchmarks at %ld dhrystones/second
", 
        ((long) LOOPS) / benchtime);
#endif

#ifndef TIMES
    times(&tms);
    benchtime = tms.tm_utime - starttime - nulltime;
    printf("Dhrystone(%s) time for %ld passes = %ld
", 
        Version,
        (long) LOOPS, benchtime/HZ);
    printf("This machine benchmarks at %ld dhrystones/second
", 
        ((long) LOOPS) * Hz / benchtime);
#endif

Proc1(PtrParIn)
REG RecordPtr  PtrParIn;
{
#define NextRecord  (*(PtrParIn->PtrComp))
struct assign(NextRecord, *PtrGlb);
    PtrParIn->IntComp = 5;
    NextRecord.IntComp = PtrParIn->IntComp;
    NextRecord.PtrComp = PtrParIn->PtrComp;
    Proc1(NextRecord.PtrComp);
    if (NextRecord.Discr == Ident1)
    {
        // More code here
    }
}
Dhrystone

NextRecord.IntComp = 6;
Proc6(PtrParIn - EnumComp, &NextRecord.EnumComp);
NextRecord.PtrComp = PtrGlb - PtrComp;
Proc7(NextRecord.IntComp, 10, &NextRecord.IntComp);
}
else
    struct assign(*PtrParIn, NextRecord);
}

Proc2(IntPar10)
OnsToFifty    *IntPar10;
{
    REG OneToFifty    IntLoc;
    REG Enumeration   EnumLoc;
    IntLoc = *IntPar10 + 10;
    for(;;)
    {
        if (CharGlob == 'A')
        {
            IntLoc;
            *IntPar10 = IntLoc - IntGlob;
            EnumLoc = Ident;
        }
        if (EnumLoc == Ident1)
            break;
    }
}

Proc3(PtrParOut)
RecordPtr    *PtrParOut;
{
    if (PtrGlb != NULL)
        *PtrParOut = PtrGlb - PtrComp;
    else
        IntGlob = 100;
    Proc1(10, IntGlob, &PtrGlb - IntComp);
}

Proc4()
{
    REG boolean    BoolLoc;
    BoolLoc = CharGlob == 'A';
    BoolLoc |= BoolGlob;
    CharGlob = 'A';
}

Proc5()
{
    CharGlob = 'A';
    BoolGlob = FALSE;
Dhrystone

extern boolean Func3();

Proc6(EnumParIn, EnumParOut)
REG Enumeration EnumParIn;
REG Enumeration *EnumParOut;
{
    *EnumParOut = EnumParIn;
    if (! Func3(EnumParIn))
        *EnumParOut = Ident4;
    switch (EnumParIn)
    {
    case Ident1:  *EnumParOut = Ident1; break;
    case Ident2:  if (IntGlob > 100) *EnumParOut = Ident1;
                  else *EnumParOut = Ident4;
                  break;
    case Ident3:  *EnumParOut = Ident2; break;
    case Ident4:  break;
    case Ident5:  *EnumParOut = Ident3;
    }
}

Proc7(IntParI1, IntParI2, IntParOut)
OneToFifty IntParI1;
OneToFifty IntParI2;
OneToFifty *IntParOut;
{
    REG OneToFifty IntLoc;
    IntLoc = IntParI1 + 2;
    *IntParOut = IntParI2 + IntLoc;
}

Proc8(Array1Par, Array2Par, IntParI1, IntParI2)
Array1Dim Array1Par;
Array2Dim Array2Par;
OneToFifty IntParI1;
OneToFifty IntParI2;
{
    REG OneToFifty IntLoc;
    REG OneToFifty IntIndex;
    IntLoc = IntParI1 + 5;
    Array1Par[IntLoc] = IntParI2;
    Array1Par[IntLoc+1] = Array1Par[IntLoc];
    Array1Par[IntLoc+30] = IntLoc;
    for (IntIndex = IntLoc; IntIndex <= (IntLoc+1); ++IntIndex)
        Array2Par[IntLoc][IntIndex] = IntLoc;
    ++Array2Par[IntLoc][IntLoc-1];
    Array2Par[IntLoc+20][IntLoc] = Array1Par[IntLoc];
    IntGlob = 5;
}
Dhrystone

```c
Enumeration Func1(CharPar1, CharPar2)
CapitalLetter CharPar1;
CapitalLetter CharPar2;
{
    REG CapitalLetter CharLoc1;
    REG CapitalLetter CharLoc2;
    CharLoc1 = CharPar1;
    CharLoc2 = CharLoc1;
    if (CharLoc2 != CharPar2)
        return (Ident1);
    else
        return (Ident2);
}

boolean Func2(StrParIl, StrPar12)
String30 StrParIl;
String30 StrPar12;
{
    REG OneToThirty IntLoc;
    REG CapitalLetter CharLoc;
    IntLoc = 1;
    while (IntLoc <= 1)
        if (Func1(StrParIl[IntLoc], StrPar12[IntLoc+1]) == Ident1)
            CharLoc = 'A';
            ++IntLoc;
        if (CharLoc == 'W' && CharLoc <= 'Z')
            IntLoc = 7;
        if (CharLoc == 'X')
            return(TRUE);
        else
            if (strcmp(StrParIl, StrPar12) > 0)
                IntLoc += 7;
                return (TRUE);
            else
                return (FALSE);
    }
}

boolean Func3(EnumParln)
REG Enumeration EnumParln;
{
    REG Enumeration EnumLoc;
    EnumLoc = EnumParln;
    if (EnumLoc == Ident3) return (TRUE);
    return (FALSE);
}
```
Dhrystone

#ifdef NOSTRUCTASSIGN
memcpy(d, s, l)
register char *d;
register char *s;
register int l;
{
    while (l--) *d++ = *s++;
}
#endif
Block Write

/*
 * blockwrite.c
 * This program creates a very large file.
 * Compile by: cc -O blockwrite.c -o blockwrite
 */
#define NAME "BLOCKWRITE"
#define FName "bigfile"
#define BSIZE 8096 /* 8K block */
#define BLOCKS 128 /* number of 8K blocks in file (total 1 Mbyte) */

#include <sys/file.h>

int main() {
    int fileflags = O_CREAT O_TRUNC O_APPEND O_WRONLY;
    int filemode = 0777;
    int f;
    int icount = 0;
    char buffer[BSIZE];
    int i;

    printf("%s: beginning (%d bytes in file)");
    fflush(1);
    for (icount = 0; icount < 23; icount++) {
        if ((f = open(FNAME, fileflags, filemode)) < 0) {
            printf("%s: unable to create '%s'", FILENAME);
            exit(1);
        }
        for (i = 1; i <= BLOCKS; i++) write(f, buffer, BSIZE);
        close(f);
    }
    printf("%s: complete (%d bytes in file)");
    fflush(1);
}


Block Read

/*
 * blockread.c
 * This program reads a very large file.
 * Compile by: cc -O blockread.c -o blockread
 *
 */
#define NAME "BLOCKREAD"
#define FILENAME "bigfile"
#define BSIZE 8096 /* 8K block */
#define BLOCKS 128 /* number of 8K blocks in file (total 1 Mbyte) */

#include <sys/file.h>

int main() {

  int fileflags = O_RDWR;
  int filemode = 0444;
  int f;
  char buffer[BSIZE];
  int i = 0;
  int icount;

  printf("%s: beginning (%d bytes in file))\n", NAME, BSIZE * BLOCKS);
  fflush(1);
  for (icount = 0; icount < 39; icount++) {
    if ((f = open(FILENAME, fileflags, filemode)) <= 0) {
      printf("%s: unable to open '%s'\n", NAME, FILENAME);
      exit(1);
    }

    for (i=1; i<=BLOCKS; i++) read(f, buffer, BSIZE);
    close(f);
  }

  printf("%s: complete (%d bytes in file))\n", NAME, BSIZE * BLOCKS);
  fflush(1);
}

46
#csh script to run timing on sort
# sorttest
# sortfile - input to sort
# sortstandard - presorted output for checking
# echo Start of sort
sort -4 +5 ./preaward/sortfile > sortout
echo End of sort. Start of compare.
diff ./preaward/sortstandard sortout
echo End of compare.
rm -f sortout
csh is a shell (command interpreter) with C-like syntax

**SYNOPSIS**

```csh
[ B csh

B -cefs inst vVxx
] [ arg ...]
```

**DESCRIPTION**

`Csh` is a first implementation of a command language interpreter incorporating a history mechanism (see `csh: History Substitutions`), job control facilities (see `csh: Jobs`), and a C-like syntax. So as to be able to use its job control facilities, users of `csh` must (and automatically) use the new tty driver fully described in `csh: tty (4)`. This new tty driver allows generation of interrupt characters from the keyboard to tell jobs to stop. See `csh: tty (1)` for details on setting options in the new tty driver.

An instance of `csh` begins by executing commands from the file `.cshrc` in the home directory of the invoker. If this is a login shell then it also executes commands from the file `.login` there. It is typical for users on crt's to put the command `stty crt` in their `.login` file, and to also invoke `csh: tset (1)` there.

**See Also**

`csh: History Substitutions`, `csh: Jobs`, `tty (4)`, `tset (1)`
In the normal case, the shell will then begin reading commands from the terminal, prompting with `%'.
Processing of arguments and the use of the shell to process files containing command scripts will be described later.

The shell then repeatedly performs the following actions:

- A line of command input is read and broken into words.
- This sequence of words is placed on the command history list and then parsed.
- Finally, each command in the current line is executed.

When a login shell terminates, it executes commands from the file `.logout' in the user's home directory.

The shell splits input lines into words at blanks and tabs with the following exceptions.

The characters `&' `%' `;' `<' `>' `( ' `)' form separate words.
If doubled in `&&', `&&', `<<' or `'>' these pairs form single words.
These parser metacharacters may be made part of other words, or prevented their special meaning, by preceding them with `*'.
A newline preceded by `*e' is equivalent to a blank.

In addition, strings enclosed in matched pairs of quotations, `'aa', `'ga' or `'s`, form parts of a word; metacharacters in these strings, including blanks and tabs, do not form separate words.
These quotations have semantics to be described subsequently.

`etc.'
Sort (Part 111)

```bash
$ last argument
& Repeat the previous substitution.
0 first (command) word
10 ex write.c
11 cat oldwrite.c
12 diff *write.c
[1][1234
**(ua first argument, i.e. 'l'
*-sflsfr abbreviates 'O*-sflsfr'
#09 write michael
#flsfr #flsfr '"th argument
#fisfr '* abbreviates #fisfr #-$'
#fisfr.s-sflsfr range of words
d directory
e existence
f plain file
o ownership
r read access
s/#flsfr/#flsfr# Substitue #fisfr for *fisfr
w write access
x execute access
z zero size

S$#name
SS
S+0
S<
S?0
S?name
Sname
Sname[selector]
Snumber
S{#name}
S{?name}
S{name[selector]}
S{name}
S{number}
(As in
Both
(See the description of
The
(The words
(as in
(e.g. '$shell').
(second form).
.. 
.B -v 
.B -x 
.B -c 
.B -e 
.B -f 
.B -i 
.B -n 
.B -s 
.B -t 

S0
```
Sort (Part 111)

B 1-v
B 1-x
B alias
B alloc
B break
B breaksw
B breaksw
B cd
B chdir
B continue
B default:
B default:
B else
B else
B end
B end
B endif
B endif
B endsw
B endsw
B exit
B history
B login
B logout
B nice

etc.
Integer Arithmetic

/*
 * integer.c
 * This program does integer arithmetic
 * Compile by: cc -O integer.c -o integer
 */

#define NAME "INTEGER"
#define COUNT 2900000 /* number of iterations */

main()
    long i;    /* iteration counter */
    long a, b, c, d;    /* integer variables for arithmetic */

    a = 1234; b = 2345; c = 3456; d = 4567;
    printf("%s: beginning (%d iterations)\n", NAME, COUNT);
    for ( i = 0; i < COUNT; i++ )    /* do some arithmetic */
        a = b + c - d;
        b = a * b / d;
    }
    printf("%s: complete (%d iterations)\n", NAME, COUNT);
}
Real Arithmetic

/*
 * real.c
 *
 * This program does real arithmetic.
 *
 * Compile by: cc -O real.c -o real
 *
 */

#define NAME "REAL"
#define COUNT 600000 /* number of iterations */

float aa, bb, cc, dd;

int ii, jj, kk;

main()
{
    printf("%s: beginning (%d iterations)\n", NAME, COUNT);
    for(ii = 1; ii < COUNT; ii++)
    {
        aa = ii;
        bb = aa * aa;
        cc = (bb - aa - .137526)/aa;
    }
    printf("%s: complete (%d iterations)\n", NAME, COUNT);
}
/*
 * largedata.c
 *
 * This program has a data space larger than real memory.
 * Compile by: cc -O largedata.c -o largedata
 */

#define NAME "LARGEDATA"
#define COUNT 20000 /* number of iterations */
#define BLOCK 1024 /* 1K block */
#define BSIZE 4000 /* large buffer size (blocks) */
#define ADDR 0xe000 /* base address of array */

main() {
    register char *curptr; /* current pointer */
    register long i; /* iteration counter */
    register long pagecount; /* number of new pages */
    register long limit; /* number of references */
    register long size; /* size of array */

    limit = COUNT;
    size = BSIZE;

    sbrk(ADDR + BSIZE * BLOCK); /* increase data space */
    srand(1); /* init random generator */
    i = 0;
    pagecount = 0;
    printf("%s: beginning (%d iterations, size %d)\n", NAME, COUNT, size * BLOCK);
    while ( ++i < limit ) { /* make COUNT memory references */
        curptr = (char *) (ADDR + (rand() % size) * BLOCK);
        if ( *curptr == 0 ) {
            pagecount++; /* increase new page count */
            curptr = 1;
        }
    }
    printf("%s: complete (%d pages referenced, %d for the first time)\n", NAME, COUNT, pagecount);
}

Large Data Space
Compile

Compile and load of to routine

compiletest
to.c - C source
subs.c - C source

echo cc -O -c preaward/to.c
c c -O -c preaward/to.c
echo cc -O -c preaward/subs.c
c c -O -c preaward/subs.c
echo cc -O -o to.o subs.o
c c -O -o to.o subs.o
rm -f to.o subs.o
END DATE FILMED

8-12-87