1990

Problems to Test Parallel and Vector Languages -- II

John R. Rice
Purdue University, jrr@cs.purdue.edu

Jin Jing

Report Number:
90-1016

http://docs.lib.purdue.edu/cstech/18

This document has been made available through Purdue e-Pubs, a service of the Purdue University Libraries. Please contact epubs@purdue.edu for additional information.
PROBLEMS TO TEST PARALLEL AND VECTOR LANGUAGES - II

John R. Rice
Jin Jing

CSD-TR-1016
September 1990
Problems to Test Parallel
and Vector Languages – II

John R. Rice**
and
Jin Jing

Computer Sciences Department
Purdue University
Technical Report CSD-TR-1016
CAPO Report CER-90-35
December, 1990

ABSTRACT

This report presents 17 problems selected to test the effectiveness of languages in expressing parallel and vector (and array) computations. Most problems have been extracted from larger computations and thus are somewhat artificial by themselves. However, they do represent a sampling of practical computations. Algorithms for the 17 problems are given in Fortran 77 and they are also designed to be used in timing computations.

This report is based on CSD-TR-516, May 1, 1985 with the same title. The problems are the same except that one has been added. In CDS-TR-516 the algorithms for the problems are expressed in four forms: Fortran 77, Fortran with Array and Parallel Extensions, PROTRAN with Extensions, and Cyber 205 Fortran. In the present report the algorithms are rewritten so as to (a) to correct some errors, (b) to make the coding style more uniform, (c) be parameterized, (d) to move I/O statements to the end, and (e) to improve the placement of runtime measuring statements. The algorithms are presented only in Fortran 77. It is easy to modify the other forms of CSD-TR-516 to conform to the new versions given here.

** This work supported in part by National Science Foundation grant CCR-8619817 and the Strategic Defense Initiative through Army Research Office contract DAAL03-90-G-00107.
INTRODUCTION

We present a set of 17 computations which naturally involve parallel and/or vector (or array) processing. The purpose is to provide a basis for testing the effectiveness of programming languages in expressing the parallel and array processing nature of computations. A secondary purpose is to provide a basis for measuring the execution efficiency (or speed up) of parallel and array processing systems.

We being by describing the changes made to the problems of the previous report [Rice, 1985] and providing some basic data on the new algorithms. Appendix A repeats the text of [Rice, 1985] with minor corrections and updates. Appendix B presents some additional remarks about the new problem 17. Appendix C presents the new algorithms in Fortran 77.

Changes Made in the Algorithms

A) Some of the previous algorithms had minor errors which prevented them from solving the stated problems even though the parallel and vector structures were correct.

B) The coding style has been made uniform.

C) Some of the previous algorithms were parameterized, but not all of them. Now they are all parameterized so that a wide range of computation sizes can be easily selected.

D) All of the input-output statements have been moved to the end of the algorithms.

E) The execution time statements have been placed so as to exclude timing the problem setup and output computations.

Problem 17

Problem 17 is an adaptive metalgorithm for numerical quadrature [Rice, 1975] which is naturally parallel. It is quite appropriate to test the effectiveness of parallel programming languages because of its dynamic, unpredictable behavior. See [Rice, 1976] for a detailed presentation of a specific instantiation of this metalgorithm and a proof of its correctness. It was intended to include it in the previous report [Rice, 1985] but a formulation could not be found which did not bias the implementation in some way. We were finally able to formulate it in a way that does not bias an implementation toward particular data structures and language constructs.
It is difficult to specify this algorithm at the Fortran 77 level without choosing specific data structures and synchronization mechanisms which bias the algorithm toward some programming language styles or toward some parallel architecture. The use of a wide variety of data structures in this metalgorithm is discussed in [Rice, 1975]: the algorithm in [Rice, 1976] uses a queue to hold the intervals being processed and uses critical variables and sections to control access to the queue (synchronize the global information exchange).

The essence of this metalgorithm is that of a collection of items (intervals in this case) which are processed independently by the processors. The result of processing one item is the elimination of the input item and the creation of 0, 1 or 2 new items. New items are placed into the collection and the computation is continued until some objective is reached (the collection becomes empty in this case).

A different instance is presented here which suppresses somewhat the dependence on specific data structures and synchronization mechanisms. The earlier descriptions involve a single collection of unprocessed intervals which all processors had to access to insert and extract items. This is now replaced by having a collection for each processor and having the monitor processor move items from large local collections to small ones in order to keep the load balanced. There is still potential contention for access to the local collections when the monitor moves intervals from one processor to another to achieve load balancing. Some synchronization might be required here, but it is a much less important aspect of this algorithm. This instance of the metalgorithm seems not to bias it unduly toward or away from shared memory or distributed memory architectures.

METALGORITHM. The high level abstraction of the metalgorithm is exactly as in [Rice, 1975], that is:

- CPU1 (HOST) MAIN: Initiates computation
- CPU2 (MONITOR) MAIN: Reads problems. Initialize control.
  Monitor computations for termination
- CPU(IP) MAIN: Processes one interval
  IP = 1 to NCPU
  INIT: Start one interval
  GET: Obtain interval from the collection
  AREAS: Compute numerical values
  PUT: Obtain access to the collection
  INSERT: Insert new intervals into collection.
  Update global control information
MORE SPECIFIC INSTANCE. The above metalgorithm is made more specific by specifying how the collection of intervals is managed. Each of the NCPU processors has its own collection which may be accessed by it and the monitor. The monitor has priority in accessing the collection, but we expect that the data structure provides for no conflict at all (and hence trivial synchronization cost). This can be accomplished using any of a number of data structures (e.g., stack, list, queue). One could consider implementations where efficiency is gained by allowing for conflict, and synchronizations, when the collection is nearly empty.

The collection is managed using the variable SIGNAL(IP) which can have one of three values:

- "run": normal mode of operation, the local collection is not large compared to others and has enough items to proceed.
- "heavy": the local collection is large and items are available to the monitor for redistribution to other processors.
- "starving": the local collection is nearly empty and additional items are requested from the monitor.

These values are computed from four algorithm parameters B_HIGH, B_LOW, C_HIGH, C_LOW and the variables COUNT(IP) = Number of items in processor IP’s collection, T_COUNT = Total of COUNT(IP), IP = 1 to NCPU, BALANCE(IP) = \( \frac{NCPU \times COUNT(IP)}{T_COUNT} \) as follow:

\[
\begin{align*}
\text{IF} ( \text{BALANCE}(IP) \geq B_{\text{HIGH}} \text{ AND } \text{COUNT}(IP) \geq C_{\text{LOW}} ) & \text{ THEN SIGNAL}(IP) = \text{"heavy"} \\
\text{IF} ( \text{BALANCE}(IP) \leq B_{\text{LOW}} \text{ AND } \text{COUNT}(IP) \leq C_{\text{HIGH}} ) & \text{ THEN SIGNAL}(IP) = \text{"starving"} \\
\text{ELSE SIGNAL}(IP) = \text{"run"} \\
\end{align*}
\]

Typical values for the parameters might be 2, 0.5, 3, and 4, respectively. We discuss load balancing in more detail in Appendix B for those who are really interested in adaptive quadrature.

This more specific instance is described in more detail by the actions of the processors as follows:
Hello CPU : 1. Assign value of NCPU
2. Enables the other CPUs
3. Initializes all control variables

MONITOR_CPU : 1. Obtains problem data
2. Divides [A,B] into NCPU equal subintervals and initializes processor collections. Initializes variables (e.g., COUNT(IP), BOUND_ERROR, sets SIGNAL(IP) = "run") and enables all processors.
3. Monitors starving and, when detected, moves some intervals to a starving CPU so that it has BALANCE = i.
4. Monitors BOUND_ERROR and terminates computations (with output) when BOUND_ERROR < EPS or when all interval collections are empty.

PROCESSOR(IP) : Invokes iteratively the sequence
- INIT
- GET
- AREA1
- PUT
- INSERT

In a shared memory architecture, the variables COUNT(IP), SIGNAL(IP), BOUNDA(IP), etc. could be implemented by shared variables. Items could be moved between collections by changing pointers or by copying, depending on which is most efficient or convenient. In a DMMP (Distributed Memory, Message Passing) architecture the management of the collection would be implemented by asynchronous messages. The Fortran 77 program given here makes many specific choices for implementation, but it is not intended to specify the algorithm in further detail so that parallel language descriptions should only conform to the abstract metalgorithm presented here.

Performance Data

We tabulate the sequential execution times of these 17 problems for two sets of parameter values, one to give short execution times and one to give longer times.
Table 1. Execution times for Fortran 77 implementations of the computations on a SUN 3/50. Two selections of the parameters are given for each problem.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Parameter Values and Execution Times</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>WITH A,B,N = 0.000 1.000 10000 GIVES TN = 1.718554 TIME 10.8000 SECONDS, (10.5800 USER, 0.2200 SYSTEM)</td>
</tr>
<tr>
<td></td>
<td>WITH A,B,N = 0.000 1.000 102400 GIVES TN = 1.718307 TIME 100.2400 SECONDS, (100.1800 USER, 0.0600 SYSTEM)</td>
</tr>
<tr>
<td>2</td>
<td>WITH N,M = 80 90 GIVES ESTAR = 446.482 TIME 9.3400 SECONDS, (8.9800 USER, 0.3600 SYSTEM)</td>
</tr>
<tr>
<td></td>
<td>WITH N,M = 320 360 GIVES ESTAR = 1797.89 TIME 105.8200 SECONDS, (103.8600 USER, 1.9600 SYSTEM)</td>
</tr>
<tr>
<td>3</td>
<td>WITH N,M = 90 125 GIVES S = 91.13226 TIME 1.1667 SECONDS, (1.0833 USER, 0.0833 SYSTEM)</td>
</tr>
<tr>
<td></td>
<td>WITH N,M = 270 625 GIVES S = 287.41537 TIME 6.7167 SECONDS, (3.7167 USER, 3.0000 SYSTEM)</td>
</tr>
<tr>
<td>4</td>
<td>WITH N = 64000 METHOD 1 GIVES 2.04807E+09 TIME 7.1200 SECONDS, (7.1000 USER, 0.0200 SYSTEM)</td>
</tr>
<tr>
<td></td>
<td>METHOD 2 GIVES 2.04807E+09 TIME 6.8200 SECONDS, (6.8200 USER, 0.0000 SYSTEM)</td>
</tr>
<tr>
<td></td>
<td>METHOD 3 GIVES 2.04807E+09 TIME 7.6600 SECONDS, (7.6000 USER, 0.0600 SYSTEM)</td>
</tr>
<tr>
<td></td>
<td>METHOD 4 GIVES 2.04807E+09 TIME 6.8000 SECONDS, (6.8000 USER, 0.0000 SYSTEM)</td>
</tr>
<tr>
<td></td>
<td>METHOD 5 GIVES 2.04807E+09 TIME 7.3400 SECONDS, (7.3400 USER, 0.0000 SYSTEM)</td>
</tr>
<tr>
<td></td>
<td>METHOD 6 GIVES 2.04807E+09 TIME 7.2400 SECONDS, (7.2400 USER, 0.0000 SYSTEM)</td>
</tr>
<tr>
<td></td>
<td>METHOD 7 GIVES 2.04807E+09 TIME 5.8600 SECONDS, (5.8600 USER, 0.0000 SYSTEM)</td>
</tr>
<tr>
<td></td>
<td>WITH N = 320000 METHOD 1 GIVES 5.12007E+10 TIME 38.6000 SECONDS, (36.3800 USER, 2.2200 SYSTEM)</td>
</tr>
<tr>
<td></td>
<td>METHOD 2 GIVES 5.12007E+10 TIME 38.2600 SECONDS, (35.9600 USER, 2.3000 SYSTEM)</td>
</tr>
<tr>
<td></td>
<td>METHOD 3 GIVES 5.12007E+10 TIME 40.1000 SECONDS, (38.2400 USER, 1.8600 SYSTEM)</td>
</tr>
<tr>
<td></td>
<td>METHOD 4 GIVES 5.12007E+10</td>
</tr>
<tr>
<td>Method</td>
<td>N</td>
</tr>
<tr>
<td>--------</td>
<td>----</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>36.8800</td>
</tr>
<tr>
<td></td>
<td>36.2200</td>
</tr>
<tr>
<td></td>
<td>28.6200</td>
</tr>
</tbody>
</table>

Method 5 gives 5.12001E+10

Time 37.6000 seconds, (35.2400 user, 2.3600 system)

Method 6 gives 5.12001E+10

Time 36.8800 seconds, (36.1800 user, 0.7000 system)

Method 7 gives 5.12001E+10

Time 36.2200 seconds, (36.0800 user, 0.1400 system)

Method 8 gives 5.12001E+10

Time 28.6200 seconds, (28.6200 user, 0.0000 system)

5

WITH NT, NS = 5 12
GIVES AVERAGE, GENIUS = 64.0874 F
AND LOW-ABOVE = 71.1111

TIME 97.2000 seconds, (96.8600 user, 0.3400 system)

WITH NT, NS = 50 1000
GIVES AVERAGE, GENIUS = 60.0098 F
AND LOW-ABOVE = 66.0109

TIME 107.5200 seconds, (105.6000 user, 1.9200 system)

7

WITH N = 1024
GIVES SOLUTION = 495.352

TIME 11.6600 seconds, (11.6400 user, 0.0200 system)

WITH N = 16384
GIVES SOLUTION = 8165.01

TIME 210.0200 seconds, (208.7800 user, 1.2400 system)

8

WITH N = 16384
GIVES SOLUTION = 8165.01

TIME 210.0200 seconds, (208.7800 user, 1.2400 system)

WITH N = 320
GIVES P(X) VALUES =
0.8225085E-01 0.1517956E-05 0.1138577E+05 NaN NaN

TIME 4.0200 seconds, (3.9800 user, 0.0400 system)

WITH N = 320
GIVES P(X) VALUES =
-NaN -NaN -NaN -NaN -NaN

TIME 4.0200 seconds, (3.9800 user, 0.0400 system)

9

WITH N,M = 781 20
GIVES SUM OF DIFFERENCE TABLE = -4.30577

TIME 5.7600 seconds, (5.6800 user, 0.0800 system)

WITH N,M = 3161 40
GIVES SUM OF DIFFERENCE TABLE = -4.75962

TIME 34.9000 seconds, (34.6600 user, 0.2400 system)

10

WITH N = 40
SUM OF ELEMENTS AND SUM OF DIAGONAL =
WITH N = 100
SUM OF ELEMENTS AND SUM OF DIAGONAL = 0.817687E+38 0.817687E+38
TIME 49.7600 SECONDS. (49.7000 USER, 0.0600 SYSTEM)

WITH N = 1000
GIVEN FMOM(N) (N=1.4) = 0.1903 0.1882 0.1820 0.1720
TIME 5.2800 SECONDS. (5.2200 USER, 0.0600 SYSTEM)

WITH N = 5000
GIVEN FMOM(N) (N=1.4) = 0.0381 0.0381 0.0380 0.0379
TIME 26.6000 SECONDS. (26.4200 USER, 0.1800 SYSTEM)

WITH N,M = 200 200
GIVES CORNER PRODUCTS = 0. -7.99980E+06
TIME 2.0800 SECONDS. (1.8600 USER, 0.2200 SYSTEM)

WITH N,M = 400 400
GIVES CORNER PRODUCTS = 0. -6.39996E+07
TIME 7.6400 SECONDS. (7.1200 USER, 0.5200 SYSTEM)

WITH N = 1000
GIVES E = 5.16012E+06
TIME 14.4000 SECONDS. (14.3000 USER, 0.1000 SYSTEM)

WITH N = 10000
GIVES E = 5.49776E+09
TIME 142.8000 SECONDS. (142.4600 USER, 0.3400 SYSTEM)

FUNCTION 1 ON 0.00 1.00 HAS TRUE = 1.718282
EVALUATION WITH 500 INTERVALS. L = 1
METHOD ANSWER NO OF POINTS ERROR LOG ERROR
1 1.718282341957 501. -0.000000476837 -632163000107E+01
2 1.718281865120 501. 0.000000000000 -310000000000E+02
3 1.301083683968 498. 0.417198181152 -3796575963350E+00

FUNCTION 2 ON 0.00 1.00 HAS TRUE = 0.522210
EVALUATION WITH 500 INTERVALS. L = 1
METHOD ANSWER NO OF POINTS ERROR LOG ERROR
1 0.522214233875 501. -0.000004291534 -536738729477E+01
2 0.522206485271 501. 0.00003457069 -546129179001E+01
3 0.333972334862 498. 0.188237607479 -726293576717E+00

FUNCTION 3 ON -1.00 2.00 HAS TRUE = 6.299197
EVALUATION WITH 500 INTERVALS. L = 1
METHOD ANSWER NO OF POINTS ERROR LOG ERROR
1 6.299215793610 501. -0.000019073486 -471956968307E+01
2 6.29915766449 501. 0.000095353674 -602059984207E+01
3 4.306991100311 498. 1.992205619812 0.299334168434E+00

TIME 2.8400 SECONDS. (2.7800 USER, 0.0600 SYSTEM)
FUNCTION 1 ON 0.00 1.00 HAS TRUE = 1.718282
EVALUATION WITH 2000 INTERVALS, L = 1
METHOD ANSWER NO OF POINTS ERROR LOG ERROR
1 1.718281865120 2001. 0.000000000000 -3.1000000000E+02
2 1.718282580376 2001. -0.000000715256 -6.145538692E+01
3 1.298349142075 1998. 0.419932723045 -3.76820296049E+00

FUNCTION 2 ON 0.00 1.00 HAS TRUE = 0.522210
EVALUATION WITH 2000 INTERVALS, L = 1
METHOD ANSWER NO OF POINTS ERROR LOG ERROR
1 0.522205352783 2001. 0.000004589558 -3.33822917938E+01
2 0.522202551365 2001. 0.000007390976 -5.13129806519E+01
3 0.33862751722 1998. 0.188347190619 -7.25040853024E+00

FUNCTION 3 ON -1.00 2.00 HAS TRUE = 6.299197
EVALUATION WITH 2000 INTERVALS, L = 1
METHOD ANSWER NO OF POINTS ERROR LOG ERROR
1 6.299193859100 2001. 0.000002861023 -5.54347848892E+01
2 6.299199581146 2001. -0.000002861023 -5.54347848892E+01
3 -3.1589987915 1998. 1.983299732208 -2.97388345003E+00

TIME 10.8800 SECONDS, (10.7400 USER, 0.1400 SYSTEM)

15 : IN EQUALLY SPACED POINTS
WITH N = 10
GIVEN MAX ERROR AND DECAY EXPONENT =
0.1490E-07 -2.12

IN CHEBYSHEV SPACED POINTS
WITH N = 10
GIVEN MAX ERROR AND DECAY EXPONENT =
0.4098E-07 -4.13
TIME 13.3000 SECONDS (USER 13.2800 SYSTEM 0.2400)

IN EQUALLY SPACED POINTS
WITH N = 20
GIVEN MAX ERROR AND DECAY EXPONENT =
0.1118E-07 7.90

IN CHEBYSHEV SPACED POINTS
WITH N = 20
GIVEN MAX ERROR AND DECAY EXPONENT =
0.7451E-08 0.00
TIME 79.8000 SECONDS, (79.5000 USER, 0.5800 SYSTEM)

16 : WITH N = 4
RESIDUE 1 := 0.269739825853E-05
RESIDUE 2 := 0.190734863281E-05
RESIDUE 3 := 0.363490607924E-05
RESIDUE 4 := 0.745058059692E-07
TIME 0.0600 SECONDS, (0.0600 USER, 0.0000 SYSTEM)

WITH N = 20
*** THE CONDITION NUMBER IS TOO HIGH  
TIME 0.6800 SECONDS, (0.6600 USER, 0.0200 SYSTEM)

WITH A,B,EPS = 0.00010000 1.00010000 0.01000000  
GIVES THE AREA = 9.21281624 AND THE BOUND = 0.00949743 
TIME 1.8800 SECONDS, (1.8200 USER, 0.0600 SYSTEM)

WITH A,B,EPS = 0.000100001.00010002 0.00010000  
GIVES THE AREA = 9.21067333 AND THE BOUND = 0.00009804  
TIME 16.6600 SECONDS, (16.4800 USER, 0.1800 SYSTEM)

REFERENCES

J.R. Rice. Parallel algorithms for adaptive quadrature III – Program correctness, ACM  

PROBLEMS TO TEST PARALLEL AND VECTOR LANGUAGES
John R. Rice

A-1. INTRODUCTION
A set of 16 problems is presented whose purpose is to test the effectiveness of programming languages in expressing parallel and vector (or array) computations. Most of the problems have been abstracted from common, realistic programs; a few problems are from small, complete programs. These problems are useful to provide an independent and somewhat uniform means to test the many new programming languages being proposed for vector and parallel computation.

A-1.A. The Forms of the Problems
The problems are presented in four forms:

Form A: Fortran 77. This is an ordinary sequential form which serves to define the computation precisely. This form is complete with timing code and output that can be used to check other implementations.

Form B: Fortran with Array and Parallel Extensions. This is an extension which is not precisely defined, but which resembles Fortran 90 in many ways. However, it also differs in some significant ways and it includes constructs for parallel computation.

Form C: PROTRAN with Extensions. PROTRAN is a Fortran extension developed by IMSL which has considered vector/matrix capabilities as well as high level, problem solving statements. The extension used here (but not precisely defined) completes the vector capabilities to a level comparable with Form B (and Fortran 90). It also includes parallel constructs.
Form D: Cyber 205 Fortran. This is CDC's Fortran extension especially targeted for the Cyber 205 vector computer. The Fortran 77 codes were used as a starting point and then a reasonable effort was put into optimizing execution time. In other words, the codes were hand vectorized.

Many of the programs are parameterized in some simple way to allow one to change the problem "size".

A-1.B. Principal Observations

This effort arose in the early 1980's from considering the Fortran 90 proposals and from attempting to evaluate them. The problem set was enlarged to consider also the nature of truly parallel computations expressed in Fortran-like languages.

Studying these 16 problems has led to the following subjective conclusions:

1. **Array Notations.** Science has evolved notation for vectors and arrays which mixes the linear algebra notation (e.g., Solve $Ax = b$ ) with identical notations (e.g., where $A = \{a_{ij}\} = \{1 / (i + j - 1) \text{ for } i \leq j, \ j \leq m \}$ and $b = \{b_j\} = \{1 \text{ for } j = 1, 0 \text{ otherwise} \}$ ). Both forms of the notation are very natural and must be included in some way if the language is to avoid clumsy, error prone constructions.

2. **Array Operations.** A substantial number of arrays operators beyond simple arithmetic are in common scientific use (e.g., sum, product, transpose, inverse, matrix product) and should be included in the language as naturally as possible. Further, new operators to create and manipulate arrays are also needed (e.g., concatenate two vectors, add a new row to an array). It is inevitable that there be numerous operators and that most of them are implemented as procedures (functions or subroutines). The natural use of the SUM, PRODUCT, MAX and MIN operators is difficult to achieve without substantial changes in the usual Fortran syntax.

3. **Language Level.** Programming productivity and execution efficiency both require that the language has the power to express computations at the natural level of scientific notation. It is an inexcusable waste of talent to ask all (or even many) scientists to learn artificial rules about DO-loop organization and
similar machine/compiler effects. The vector languages in current use from Cray Research, CDC and Los Alamos can only be justified as temporary expediences.

4. Parallel Computation. Languages constructs for parallel computation are still very open. We observe:

4a. Control constructs. It is easy to see that only a few control constructs are needed and that there are several reasonable alternatives for them.

4b. Data access and control. A principal difficulty (which does not show up on toy programs of one page) is how to organize and access data. The Fortran COMMON and parameter passing lead to obvious data synchronization problems.

The remainder of this report describes the problem set, first in a summary way using tables of characteristics such as lengths and execution timings. Then there is a set of brief descriptions (only a few lines) followed by four appendices which give all the problems in each of the four forms.

A-2. CHARACTERISTICS OF THE PROBLEMS

This problem set evolved from consideration of the vector/array features that have been proposed for the next Fortran standard. papers and reports that discuss these extensions along with examples are [Smith, 1982], [Wilson, 1982], [Rice, 1981], [Rice, 1984]. A few of these problems were constructed just for this report. A few are taken from the book [Rice. 1983], these are programs that exhibit a high degree of parallelism.

The Fortran 77 form of the programs use the following general design criteria.

1. The computational "size" of the problem can be varied easily. This is implemented with PARAMETER and DATA statements.
2. Timing is included. This involves the availability of a system dependent timing routine.
3. A simple numerical value is printed to provide a check of the accuracy of alternative implementations.
The PROTRAN and extended Fortran forms are the more readable and they also include more general comments describing the problem.

Table A-1 provides a simple summary of the characteristics of the computations of the problems. The following terminology is used:

- **vector:** means one-dimensional array manipulation of normal mathematical vectors (in the linear algebra sense).
- **array:** means manipulation of arrays that are not matrices in the linear algebra sense. The dimension of the arrays involved is given in the table.
- **matrix:** means manipulation of matrices and vectors in the linear algebra sense.
- **parallel:** means independent computations that are not obviously recast in the form of array manipulation.
- **formula:** means the problem involves computations that can be expressed naturally in terms of common mathematical notations.
- **procedures:** means the problem involves a procedure or function that interacts with parallelism or vectorization. Simple procedures like SUM, DOTPRODUCT, COS, etc., are not included.

The problems are relatively small by nature. Table A-2 gives the lengths of each of the forms of the problems. Note that the design of the problems requires several declarations, timing and I/O statements which would not be present in the "production" version of these computations. The counts in Table A-2 do not include these "extra" lines nor comments; they represent the computational kernel of the problem.
Table A-1. Six characteristics of the problems, see text for definitions of the headings.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Vector</th>
<th>Array</th>
<th>Matrix</th>
<th>Parallel</th>
<th>Formula</th>
<th>Procedures</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>12</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13 H</td>
<td>1</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>14</td>
<td>1</td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>15</td>
<td>2</td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>16</td>
<td>x</td>
<td>x</td>
<td></td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>17</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
</tr>
</tbody>
</table>
Table A-2. Counts of the lines of the computational kernel in the four forms of each problem.

<table>
<thead>
<tr>
<th>Form</th>
<th>Problem</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>A. Fortran 77</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B. Extended Fortran</td>
<td></td>
<td>3</td>
<td>7</td>
<td>6</td>
<td>21</td>
<td>20</td>
<td>28</td>
</tr>
<tr>
<td>C. Extended PROTRAN</td>
<td></td>
<td>4</td>
<td>5</td>
<td>4</td>
<td>19</td>
<td>15</td>
<td>24</td>
</tr>
<tr>
<td>D. 205 Fortran (Opt)</td>
<td></td>
<td>7</td>
<td>7</td>
<td>4</td>
<td>24</td>
<td>35</td>
<td>24</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
</tr>
<tr>
<td>A. Fortran 77</td>
<td></td>
<td>28</td>
<td>12</td>
<td>29</td>
<td>28</td>
<td>12</td>
<td>19</td>
</tr>
<tr>
<td>B. Extended Fortran</td>
<td></td>
<td>22</td>
<td>14</td>
<td>29</td>
<td>18</td>
<td>9</td>
<td>19</td>
</tr>
<tr>
<td>C. Extended PROTRAN</td>
<td></td>
<td>11</td>
<td>12</td>
<td>24</td>
<td>21</td>
<td>8</td>
<td>15</td>
</tr>
<tr>
<td>D. 205 Fortran (Opt)</td>
<td></td>
<td>18</td>
<td>12</td>
<td>25</td>
<td>21</td>
<td>20</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>13</td>
<td>13H</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>A. Fortran 77</td>
<td></td>
<td>13</td>
<td>21</td>
<td>72</td>
<td>64</td>
<td>60</td>
<td></td>
</tr>
<tr>
<td>B. Extended Fortran</td>
<td></td>
<td>16</td>
<td>21</td>
<td>64</td>
<td>52</td>
<td>116</td>
<td></td>
</tr>
<tr>
<td>C. Extended PROTRAN</td>
<td></td>
<td>13</td>
<td>17</td>
<td>63</td>
<td>56</td>
<td>27</td>
<td></td>
</tr>
<tr>
<td>D. 205 Fortran (Opt)</td>
<td></td>
<td>13</td>
<td>21</td>
<td>97</td>
<td>112</td>
<td>62</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>13</td>
<td>21</td>
<td>762</td>
<td>67</td>
<td>60</td>
<td></td>
</tr>
</tbody>
</table>

Table A-2 shows two variations of the Cyber 205 Fortran forms. The 205 Fortran (Opt) refers to the hand optimized version, the one expected to be used in a production code. The 205 Fortran (Seq) refers to the Fortran 77 form with minimum modifications needed to make it run on the Cyber 205. The changes made are due to different rules about the use of indexes in arrays, etc.

Table A-3 shows some run time data for the three executable forms of the problems (Fortran 77, 205 Fortran (Opt) and 205 Fortran (Seq)). We do not analyze in any detail the data in this problem; we do note that there is some erratic behavior.
A-3. BRIEF DESCRIPTIONS OF THE PROBLEMS

In this section we give a brief description of the 16 problems. The precise descriptions are the Fortran 77 forms of the problem.

Table A-3. Execution times of three versions of the problems. The Fortran 77 version is run on the VAX 11/780 (single precision, F77 compiler) and the other two on the Cyber 205 (single precision, Fortran 200 compiler). The second column has the ratio of the column 1 to column 2 times under its entries. Similarly, the column 2 to columns 3 ratios are given under the column 3 entries.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Fortran 77</th>
<th>205 Fortran (Seq)</th>
<th>205 Fortran (Opt)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.017</td>
<td>0.0135</td>
<td>0.0127</td>
</tr>
<tr>
<td></td>
<td>1.3</td>
<td></td>
<td>1.06</td>
</tr>
<tr>
<td>2 (5 cases)</td>
<td>1.177</td>
<td>0.0455</td>
<td>0.0459</td>
</tr>
<tr>
<td></td>
<td>25.9</td>
<td></td>
<td>0.99</td>
</tr>
<tr>
<td>3 (3 cases)</td>
<td>0.067</td>
<td>0.0028</td>
<td>0.0131</td>
</tr>
<tr>
<td></td>
<td>23.9</td>
<td></td>
<td>0.21</td>
</tr>
<tr>
<td>4 (7 cases)</td>
<td>0.250</td>
<td>0.032</td>
<td>0.0210</td>
</tr>
<tr>
<td></td>
<td>7.8</td>
<td></td>
<td>1.5</td>
</tr>
<tr>
<td>5 (3 cases)</td>
<td>2.617</td>
<td>0.0758</td>
<td>0.0328</td>
</tr>
<tr>
<td></td>
<td>34.5</td>
<td></td>
<td>2.3</td>
</tr>
<tr>
<td>6 (2 cases)</td>
<td>2.417</td>
<td>0.0411</td>
<td>0.0283</td>
</tr>
<tr>
<td></td>
<td>58.8</td>
<td></td>
<td>1.45</td>
</tr>
<tr>
<td>7 (3 cases)</td>
<td>0.383</td>
<td>0.0345</td>
<td>0.0299</td>
</tr>
<tr>
<td></td>
<td>11.1</td>
<td></td>
<td>1.15</td>
</tr>
<tr>
<td>8 (3 cases)</td>
<td>0.250</td>
<td>0.0220</td>
<td>0.0197</td>
</tr>
<tr>
<td></td>
<td>11.4</td>
<td></td>
<td>1.12</td>
</tr>
<tr>
<td>9 (3 cases)</td>
<td>0.583</td>
<td>0.0254</td>
<td>0.0219</td>
</tr>
<tr>
<td></td>
<td>23.0</td>
<td></td>
<td>1.16</td>
</tr>
<tr>
<td>10 (3 cases)</td>
<td>0.250</td>
<td>0.0192</td>
<td>0.0181</td>
</tr>
<tr>
<td></td>
<td>13.</td>
<td></td>
<td>1.06</td>
</tr>
<tr>
<td>11 (2 cases)</td>
<td>1.033</td>
<td>0.0383</td>
<td>0.0160</td>
</tr>
</tbody>
</table>
Problem 1: evaluate the trapezoidal rule estimate of an integral of \( f(x) \):
\[
T_N = h^*(f(a)/2 + \sum_{i=1}^{N-1} f(a + ih) + f(b)/2)
\]

Problem 2: Compute the value of
\[
e^* = \sum_{i=1}^{n} \prod_{j=1}^{m} (1 + e^{-i-j})
\]

Problem 3: Compute the value of \( S = \sum_{j=1}^{n} \prod_{i=1}^{m} a_{ij} \)

Problem 4: Compute the value of \( R = \sum_{j=1}^{n} \frac{1}{x_j} \)

Problem 5: One has a table of the i-th student’s score on the j-th test. One is to:
(a) list the top score for each student = \textit{top}_i
(b) give the number of scores above the average = \textit{NABOVE}
(c) increase all the above average scores by 10 percent
(d) give the lowest score that is above average = \textit{LOW\_ABOVE}
(e) say whether any student has all scores above average = \textit{GENIUS}

Problem 6: Solve the tridiagonal system \( Tx = y \) by the special, vector oriented algorithm of [Jordan, 1979]. The matrix \( T \) is represented by \( L, D \) and \( U \), its lower diagonal, main diagonal and upper diagonal.
**Problem 7:** Compute polynomial interpolant values of \( f(x) \) at five points using Lagrange interpolation formulas:

\[
p(x) = \sum_{i=1}^{N} f(x_i) l_i(x) \quad l_i(x) = \prod_{\substack{j=1 \atop j \neq i}}^{N} \frac{x-x_j}{x_i-x_j}
\]

**Problem 8:** The divided difference table for a set of data \( x_i, y_i = f(x_i) \) is defined by the formulas

\[
f[x_i] = y_i \\
\frac{f[x_i, x_{i+1}, \ldots, x_{i+k}]}{x_{i+k} - x_i} = \frac{f[x_{i+1}, \ldots, x_{i+k}] - f[x_i, \ldots, x_{i+k-1}]}{x_{i+k} - x_i}
\]

The problem is to compute the first \( M \) columns of the divided difference table

\[
D_{i,k} = \frac{f[x_i, x_{i+1}, \ldots, x_{i+k-1}]}{x_{i+k} - x_i}
\]

**Problem 9:** One has an array \( u_{ij} \) of values on the \( N \) by \( M \) grid and wants to replace each value by the average of its value plus those of all its neighbors. This is expressed by

\[
u_{ij} = \frac{\sum_{Neighbors} u_{ij}}{(Number \ of \ neighbors)}
\]

This computation is typical of what one does in solving partial differential equations, image processing and geometric modeling.

**Problem 10:** Compute the LU factorization of the \( N \) by \( N \) matrix \( A = a_{ij} \) using Gauss elimination with pivoting.

**Problem 11:** Read sets of data \( d_i, i = 1, \ldots, N \), trim the negative values to zero and large values to 1000, do a logarithmic transformation \( d_i = \log(1 + d_i) \) and compute the first four Fourier moments

\[
\sum_{i=1}^{N} d_i \cos(\pi i / (N + 1))
\]

Then save these moments and the data ID in a data base.

**Problem 12:** Give the \( m \) by \( m \) matrix \( A \), the \( 1 \) by \( m \) vector \( R \), the \( m \) by \( 1 \) vector \( C \) and a number \( a \), construct the array

\[
ABIC = \begin{bmatrix} A & C \\ R & a \end{bmatrix}
\]

**Problem 13:** For given vectors \( a, b, c \) and \( d \) of dimension \( N \), compute the new vector

\[
a_i = a_i \sin b^i \\
\]

If \( a_i < \cos(c_i) \) then \( a_i = a_i + c_i \)
else \( a_i = a_i - d_i \)

and then compute
Problem 13H: Modify Problem 13 for a machine (e.g., such as the Denelcor HEP) that wants to have the computation split into groups of 20 processes.

Problem 14: Carry out a test of four methods to integrate three different functions with 10 different levels of accuracy each. Print out a table with all the results including the number of function evaluations used in each integration. This problem comes from [Rice, 1983], page 204.

Problem 15: Carry out a comparison of two types of interpolation points (equispaced and Chebyshev spaced) for Hermite interpolation using piecewise cubic polynomials. The interpolant's value \( v \) at \( y \) can be expressed as

\[
v(y) = \sum_{j=1}^{N} f(x_j) h_{1j}(y) + f'(x_j) h_{2j}(y)
\]

Problem 16: Solve a matrix equation \( Ax = B \) where \( A \) is an \( N \) by \( N \) Hilbert matrix and \( BR \) is an \( N \) by 4 matrix. The matrix order \( N \) takes on the values 4, 8, 12, 16 and 20 and the \( B \) column-vectors are, respectively, the first column of the identity matrix, all 1's, and 0.01 random perturbation of all 1's, and alternating +1, -1.

A-4. BRIEF DESCRIPTIONS OF THE LANGUAGES

The Fortran 77 form is in Fortran as standardized in 1977.

The Extended Fortran form is similar to Fortran 90. The features used here which are taken directly for proposals adopted in Fortran 90 include:

- Array assignments and expressions
- Array oriented functions, e.g., SUM, PRODUCT, COUNT, MASTK_SUM
- The FORALL statement (which was deleted from Fortran 90 in the end) is also used as in

\[
\text{FORALL (I = 1:N) A(I) = A(I - 1) / (I*2 + 2)}
\]

Note that the syntax and form used here is from 1985 documents and the final form of Fortran 90 might be different. Two array statements are used here which are not
included in Fortran 90. They are

1. **Block FORALL.** e.g.,

   ```fortran
   BLOCK FORALL (I = 1:N)
   A(I) = B(I) + C(I)
   CALL GETIT(D,I)
   END ALL
   ```

2. **Range variables.** This allows the working size of arrays to be independent of the storage allocation. See [Rice, 1982a] for more details.

Finally, there are statements which are specifically for parallel computation, namely

3. **DO PARALLEL.** specify a block of independent statements:

   ```fortran
   DO PARALLEL
   CALL SUB1(A,B)
   CALL SUB2(C,D)
   END PARALLEL
   DO PARALLEL(I = 1,N)
   CALL ADDFUNK(F,G,I)
   END PARALLEL
   ```

4. **CRITICAL.** declares a variable which might be accessed simultaneously by different codes and where single access is to be provided.

5. **BEGIN-END with declarations.** This provides two key facilities:

   (A) simple grouping of statements to indicate blocks to be handled in parallel,

   (B) declarations of local (dynamic) arrays and variables within blocks. Consider

   ```fortran
   DO PARALLEL
   BEGIN
   ```
COMMON / SELECT / JFUNK

DO PARALLEL(I - 1,N)

BEGIN

REAL H, TRAP, FI(I)

H = (B - A) / (I + 1)

FOR ALL (J = 1,I) FI(J) = F(A + J*H)

TRAP = F(A) + F(B) + DOTPRODUCT(FI,WT)

END

END PARALLEL

END

more code

END PARALLEL

This code creates a COMMON block SELECT for the first group of statements. Then N groups of statements are created within the first group, each with its own values of H, TRAP and array FI. The statements within a BEGIN-END block are executed sequentially.

For more information use the X3J3 documents describing the Fortran 8X proposals as well as [Smith, 1982], [Wilson, 1982], [Rice, 1982b] and [Rice, 1984].

PROTRAN is described in Chapter 15 of [Rice, 1983], [Aird and Rice, 1983]. PROTRAN is a higher level language for mathematical and scientific computation and, as such, has vector and matrices (not 2-D arrays) as data types. It also has operators like SUM and PRODUCT. The extension used here is in two parts. First, some of the vector and array facilities of Fortran 8X are included so that PROTRAN is as suitable for vector processing as Fortran 8X. This is a small extension. Second, the parallel constructs used for the extension of Fortran are also included, that is:

DO PARALLEL

CRITICAL

BEGIN-END with declaration
The Cyber 205 Fortran, called Fortran 200, is documented in [CDC, 1983]. It has two types of extensions. First are vector processing facilities similar to many of those included in the extended Fortran and extended PROTRAN. They use somewhat different syntax and are more limited due to the specific nature of vectors on the Cyber 205. Second are machine specific subroutines called the Q8 subroutines. These calls allow one to insert specific machine language statements into the object code generated by the Fortran compiler. They are often needed to obtain maximum performance from the Cyber 205.

A-5. REFERENCES


Jordan, Tom (1979), private communication.


APPENDIX B: LOAD BALANCING FOR ADAPTIVE QUADRATURE

The load balancing presented for Problem 17 is simple, but probably not satisfactory for a production quality quadrature code. The objective is to distribute the work evenly over the processors and work is not simply related to the number of intervals. One should consider two extreme cases.

1. The integrand $f(x)$ is uniformly smooth. In this case one expects the intervals to be cut repeatedly until a certain size is reached. At that time all of them are processed. BOUND_ERR is less than EPS, and the computation ceases. The load should remain evenly distributed over all processors provided (a) if was evenly distributed to start, (b) the processors are equally quick in processing an interval.

2. The integrand $f(x)$ has one or a few singularities. In this case there is an initial phase where most of the integration is complete— as measured by the length of the intervals involved. However, the few very short intervals left will continue to require lengthy processing. Thus, each singularity becomes a source of a long sequence of shorter and shorter intervals. Load balancing is critical for good performance.

We suggest two changes to make a program more efficient.

A. Subdivide Intervals into Many Parts. One should divide each interval into a larger and larger number of subintervals as the load starts to become unbalanced. In extreme cases, one probably should have the order of $p$ subintervals when using $p$ processors.

B. Measure Work by Both Intervals Counts and Lengths. Modify the previous collection management by renaming BALANCE(I) to be

$$\text{BALANCE\_COUNT(IP) = NCPU \times COUNT(IP) / TCOUNT.}$$

Then introduce

$$\text{LENGTH(IP) = Length of intervals in IP's collection}$$

$$\text{TLENGTH = Total of LENGTH(IP), IP = 1 to NCPU}$$

$$\text{BALANCE\_LENGTH(IP) = NCPU \times LENGTH(IP) / TLENGTH.}$$
We introduce a weight $\alpha$ as follows

\[
\begin{align*}
AB &= \text{Original interval length} \\
\alpha &= \frac{\text{TLENGTH}}{AB}
\end{align*}
\]

and define $\text{BALANCE}(IP)$ dynamically as

\[
\text{BALANCE}(IP) = \alpha \times \text{BALANCE}_\text{LENGTH} + (1 - \alpha) \times \text{BALANCE}_\text{COUNT}
\]

We believe that these changes would make the algorithm much more practical.
Problem 1

Reference: PROBLEMS TO TEST PARALLEL AND VECTOR LANGUAGES
CSE-TR 516, COMPUTER SCIENCE, PURDUE UNIVERSITY
JOHN R. RICE, MAY 1, 1990

Revised by John R. Rice and J. Jing, Oct. 1, 1990

Parameter (KASES=11)
Dimension IN(KASES)
Dimension TARRAY(2)
Integer N(KASES)
Data A, B / 0.0, 1.0 /
Data N / 100, 200, 400, 800, 1600, 3200,

6400, 12800, 25600, 51200, 102400 /

Loop over cases

Time1 = DTIME(TARRAY)
Do 20 K = 1, KASES

E = (N-N(K)-1)
SUM = 0.0
Do 10 I = 1, N(K)-1
SUM = SUM + A(I)*K(I)
10 continue

TIN(K) = N(K)**((F(A)+F(B))/2.+SUM)
20 continue

Time2 = DTIME(TARRAY)
Do 50 X = 1, KASES
Print 10, A, B, N(K)
10 Format ('Problem 1 with A,B,N = ', F10.1, 3, 'X, I6)
Print 40, TIN(K)
40 Format (' Gives TIN = ', F10.4)
50 continue

Print 60, Time7, TARRAY(1), TARRAY(2)
60 Format ('Time ', F8.4, ' Seconds, ', F8.4, ' User, ', F8.4, ' System' )
End

Function P (X)
F = EXP(X)
Return
End

Probs.Par.Vect.Lang
Prob. 1
PROBLEM 2

REFERENCE: PROBLEMS TO TEST PARALLEL AND VECTOR LANGUAGES
CSC-TR 516, COMPUTER SCIENCE, PURDUE UNIVERSITY
JOHN R. RICE, MAY 1, 1985
REVISED BY JOHN R. RICE AND J. JING, OCT. 1, 1990

PARAMETER (MDIM=320,MDIM=360,ASES=6)
DIMENSION TEMP(MDIM,MDIM),PROD(MDIM),TESTAR,ASES
INTEGER M(ASES),M(ASES)
DIMENSION TARRAY(2)
DATA N / 10,20,40,80,160,320 /
DATA N / 15,30,60,90,180,360 /

LOOP OVER RACES

TIME1 = TIME(TARRAY)
DO 50 K = 1, RACES
  DO 40 J = 1, M(K)
    DO 10 J = 1, M(K)
      Ti = FLOAT(-ABS(1-I))
      TEMP(I,J) = Ti*EXP(Ti)
      10 CONTINUE
    PROD(I,J) = PROD(I,J)*TEMP(I,J)
  20 CONTINUE
  30 CONTINUE
  ESTAR = 0
  DO 40 I = 1, M(K)
    ESTAR = ESTAR+PROD(I)
  40 CONTINUE
  TESTAR(K) = ESTAR
  50 CONTINUE
TIME2 = TIME(TARRAY)
DO 70 K = 1, RACES
  PRINT 60,N(H),K
  FORMAT ('PROBLEM 2 WITH N,H = ','F0.4,F0.4')
  PRINT *, 'GIVEN ESTAR = ',TESTAR(K)
  70 CONTINUE
  PRINT 90,TIME1,TARRAY(1),TARRAY(2)
  FORMAT ('TIME' ,F8.4,' SECONDS, (''F8.4,' USER,'F8.4,' SYSTEM')')
END
PROBLEM 3

REFERENCE: PROBLEMS TO TEST PARALLEL AND VECTOR LANGUAGES
CSD-TR 516, COMPUTER SCIENCE, PURDUE UNIVERSITY
JOHN R. RICE, MAY 1, 1985

REVISED BY JOHN R. RICE AND J. JING, OCT. 1, 1990

INTEGER SIZE
PARAMETER (NDIM=270, NDIH=635, SIZE=NDIM*NDIM, KASES=4)
DIMENSION A(NDIM,NDIM),P(NDIM),N(KASES),H(KASES),TS(KASES)
DIMENSION TARRAY(2)
DATA N / SIZE*1.0001 / M / 10,30,90,270 /
DATA M / 5,25,425,625 /
TIME1 = DTIME(TARRAY)
DO 30 K = 1, KASES
   NH = M(K)
   MD = MDH(K)
   DO 10 I = 1, NH
      P(I) = 0
      DO 10 J = 1, NH
         P(I) = P(I) + A(I,J)
   CONTINUE
   S = 0
   DO 20 I = 1, NH
      S = S + P(I)
   CONTINUE
   TS(K) = S
30 CONTINUE
TIME2 = DTIME(TARRAY)
DO 60 K = 1, KASES
   PRINT 40, N(K),M(K)
40 FORMAT(2,PROBLEM 3 FOR N,M = 16,32,18)
   PRINT 50, TS(K)
50 FORMAT('GIVES S = ',F9.4)
60 CONTINUE
   PRINT 70,TIME2,TARRAY(1),TARRAY(2)
70 FORMAT('TIME ',F8.4,' SECONDS, ',F8.4,' USER, ',F8.4,' SYSTEM')
END
PROBLEM 4

PARAMETER (N=64000, K=200)

METHOD 1
DO 10 I = 1, N
IF (X(I).LE.0) THEN
Y(I) = 1.X(I)
ENDIF
10 CONTINUE

METHOD 2
DO 50 I = 1, N
IF (X(I).LE.0) THEN
Y(I) = 1.X(I)
ENDIF
50 CONTINUE

METHOD 3
DO 100 I = 1, N
IF (X(I).LE.0) THEN
Y(I) = 1.X(I)
ENDIF
100 CONTINUE

METHOD 4
DO 200 I = 1, N
IF (X(I).LE.0) THEN
Y(I) = 1.X(I)
ENDIF
200 CONTINUE

METHOD 5
DO 300 I = 1, N
IF (X(I).LE.0) THEN
Y(I) = 1.X(I)
ENDIF
300 CONTINUE

METHOD 6
DO 400 I = 1, N
IF (X(I).LE.0) THEN
Y(I) = 1.X(I)
ENDIF
400 CONTINUE

PROBS. PAR. VECT. LANG

REVISED BY JOHN R. RICE AND J. JING. OCT. 1, 1980
TOTALTIME = TOTALTIME+TIME
PRINT 40,TIME2,ARRAY(1),ARRAY(2)
130 CONTINUE
PRINT 140,TOTALTIME
140 FORMAT ('TOTAL TIME ',F8.4,' SECONDS')
END
END
**PROBLEM 5**

**REFERENCES:** PROBLEMS TO TEST PARALLEL AND VECTOR LANGUAGES
CSD-TR 516, COMPUTER SCIENCE, PURDUE UNIVERSITY
J. N. R. RICE, Sept 1985
REVISED BY J. N. R. RICE AND J. J. JIN, Occ. 1, 1986

PARAMETER (NDIM=50, NSDIM=100, RASES=5)
DIMENSION SCORES(NDIM,NSDIM),TOP(NDIM),NSDIM]
DIMENSION AVER(60),TAVER(RASES)
DIMENSION TOWABD(RASES)

LOGICAL ABOVE(NDIM,NSDIM),GENIUS,THERE(|RASES,60/NSDIM)
REAL LHABO

DIMENSION TARRAY(2)
DATA H /5,20,30,40,50,M/12,30,200,400,1000
TIME = DTIME(TARRAY)
DO 10 I = 1, NSDIM
DO 10 J = 1, NSDIM
SCORES(I,J) = 60.40.-SIN(I*J*63.21)
10 CONTINUE
DO 100 K = 1, RASES
HABO = 0
LHABO = 9.9K10
NS = N(K)
NS = N(K)
DO 20 I = 1, NT
TOP(I) = XMAX(SCORES,NDIM,NSDIM)
20 CONTINUE
SUM = 0.0
DO 30 I = 1, NT
DO 30 J = 1, NS
SUM = SUM+SCORES(I,J)
30 CONTINUE
AVER = SUM/(N5*NT)
DO 50 I = 1, NT
DO 50 J = 1, NS

INCREASE SCORES ABOVE AVERAGE

IF (SCORES(I,J)>AVER) THEN
SCORES(I,J) = 1.1*SCORES(I,J)
ENDIF

SET SWITCH FOR SCORES ABOVE AVERAGE

IF (SCORES(I,J)>AVER) THEN
HABO = HABO+1
ABOV(1,J) = .TRUE.
ENDIF

40 CONTINUE
50 CONTINUE

LHABO = LHABO+1(SCORES,ABOVE)

DO 60 I = 1, NT
DO 60 J = 1, NS
IF (ABOVE(I,J)) THEN
IF (SCORES(I,J)>LHABO) THEN
LHABO = SCORES(I,J)
ENDIF
ENDIF
60 CONTINUE
Probs.Par.Vect.Lang

prob.6

DO 60 I = 1, K
Y(I) = T(I)
T(I) = 0.
60 CONTINUE

DO 70 I = K+1, N
Y(I) = T(I)
T(I) = -L(I) * L(I-K)
70 CONTINUE

ASSIGN TO L, COMPUTE U

DO 80 I = 1, N-K
L(I) = T(I)
T(I) = U(I) + U(I+K)
80 CONTINUE

DO 90 I = N-K+1, N
L(I) = T(I)
T(I) = 0.
90 CONTINUE

ASSIGN TO U

DO 100 I = 1, N
U(I) = T(I)
100 CONTINUE

K = 2**R

DO 110 I = 1, N
X(I) = Y(I)/D(I)
110 CONTINUE

SUNZ = 0

DO 120 I = 1, N
SUNZ = SUNZ + X(I)
120 CONTINUE

SOLUT(M) = SUNZ

DO 130 I = 1, N
T(I) = X(I) + T(I)
130 CONTINUE

PRINT *, 'GIVES SOLUTION = ', SOLUT(M)

END
Probs.Par.Vect.Lang

prob.7

\[
\begin{align*}
P(\text{num}, x) &= P(\text{num}, x) + \text{TTEMP}(1) \\
70 & \text{continue} \\
80 & \text{continue} \\
90 & \text{continue} \\
90 & \text{continue} \\
90 & \text{continue} \\
90 & \text{continue} \\
90 & \text{continue} \\
90 & \text{continue} \
\end{align*}
\]

\[
\begin{align*}
\text{TIME}1 &= \text{TIME}(\text{ARRAY}) \\
\text{DO } & 90 \text{ num } = 1, \text{ KASES} \\\n& N = \text{KASES}(\text{num}) \\
& \text{FORALL}(i = 1:N) \text{ XI}(i) = 1.08 \\
& \text{DO } 10 i = 1, N \\\n& \text{XI}(i) = 1.08 \\
10 & \text{continue} \\
& \text{DEMONIIATOR} \\
& \text{DO } 10 j = 1, N \\\n& \text{TEMP}(j) = 1.0 \\
& \text{DENOM}(j) = 1.0 \\
& \text{IF } (j \neq i) \text{ THEN} \\
& \text{TEMP}(j) = \text{XI}(j) \cdot \text{XI}(i) \\
& \text{DENOM}(j) = \text{DENOM}(j) \cdot \text{TEMP}(j) \\
& \text{ENDIF} \\
& \text{continue} \\
& \text{NOTE: THE DENOMIIATOR IS INVERTED HERE SO THAT A MULTIPLICATION CAN BE DONE LATER} \\
& \text{DEMON}(i) = 1.0 / \text{DENOM}(i) \\
30 & \text{continue} \\
& \text{DO } 80 k = 1, 5 \\
& \text{DO } 80 j = 1, N \\\n& \text{IF } (j \neq i) \text{ THEN} \\
& \text{TEMP}(j) = \text{XI}(k) \cdot \text{XI}(j) \\
& \text{ENDIF} \\
40 & \text{continue} \\
& \text{TTEMP}(i) = 1.0 \\
& \text{PTM} = 1.0 \\
& \text{DO } 50 j = 1, N \\
& \text{PTM} = \text{PTM} \cdot \text{TTEMP}(j) \\
50 & \text{continue} \\
& \text{XL}(i) = \text{PTM} \cdot \text{DENOM}(i) \\
60 & \text{continue} \\
& \text{FORALL } (i = 1:N) \text{ TTEMP}(1) = P(XI(1)) \cdot XL(1) \\
& P(XI, k) = 0 \\
& \text{DO } 70 j = 1, N \\
& \text{TTEMP}(1) = P(XI(1)) \cdot XL(1) \\
\end{align*}
\]

REFERENCE: PROBLEMS TO TEST PARALLEL AND VECTOR LANGUAGES
CSO-TR 516, COMPUTER SCIENCE, PURDUE UNIVERSITY
JOHN R. RICE, OCT. 1, 1990
REVISED BY JOHN R. RICE AND J. JING, OCT. 1, 1990

PARAMETER (MAXPT=50, KASES=4)
DIMENSION XI(MAXPT), XL(MAXPT)
DIMENSION TEMP(MAXPT), TTEMP(MAXPT), TTEMP(MAXPT)
DIMENSION X(5), KASES, S, TP(5), NR(KASES)
DOUBLE PRECISION DENOM(MAXPT)
DIMENSION ARRAY(2)
DATA X / 1.1, 1.2, 2.1, 2.2 /, NR / 5, 20, 30, 40 /
Probs.Par. Vect.Lang

prob.8

DIM M = 2000, N = 400, L = 5

DO I = 1, M
  DO J = 1, N
    X(I,J) = .2*I + .01*COS(I*FLOAT(J))
  CONTINUE
END

DO I = 1, M
  DO J = 1, N
    DDIFF(I,J) = (DIFF(I+1,J-1) - DIFF(I,J-1) - DIFF(I,J) + DIFF(I-1,J+1)) / (X(I,J+1) - X(I,J-1))
    DSUM = DSUM + DDIFF(I,J)
  CONTINUE
END

WRITE (4,10) M, N
10 FORMAT (2I5)

DO K = 1, L
  WRITE (4,20) K
  CONTINUE
20 FORMAT (1X, 1H01, 3H 0.01* COS(I))

DO I = 1, M
  DO J = 1, N
    X(I,J) = -2 + .01*COS(I) + .01*COS(J)
  CONTINUE
END

DO I = 1, M
  DO J = 1, N
    X(I,J) = (.2*I + .01*COS(I*FLOAT(J)))
  CONTINUE
END

WRITE (4,30) M, N
30 FORMAT (2I5)

DO K = 1, L
  WRITE (4,40) K
  CONTINUE
40 FORMAT (1X, 1H01, 3H 0.01* COS(I))
Problem 9

REFERENCE: PROBLEMS TO TEST PARALLEL AND VECTOR LANGUAGES
CSE-TR 516, COMPUTER SCIENCE, PURDUE UNIVERSITY
JOHN R. RICE, MAY 1, 1985

REVISED BY JOHN R. RICE AND J. JING, OCT. 1, 1990

PARAMETER (NDIM=200, NDIM=200, BASES=6)
DIMENSION U(NDIM,NDIM),T(NDIM,NDIM), TSUM(TBASES)
DIMENSION TARRAY(2)
INTEGER N[BASES], N[BASES]
DATA HR / 4, 10, 20, 200, HR / 4, 20, 60, 180 /

C TIME1 = DTIME(TARRAY)
DO 60 RR = 1, BASES
N = N[RR]
H = HR[RR]

FORALL (I=1:N, J=1:M) U(I,J) = 2*(I+1) + J*(J+1)

DO 20 I = 1, N
DO 20 J = 1, N
U(I,J) = T(I,J) + J*(J+1)
CONTINUE
20 CONTINUE

DO 40 I = 1, N
DO 40 J = 1, N
T(I,J) = (U(I,J)+U(I+1,J)+U(I,J-1)+U(I+1,J-1))/4.
CONTINUE
40 CONTINUE

DO 60 J = 1, M
DO 60 I = 1, N
T(I,J) = (U(I,J)+U(I,J+1)+U(I,J-1)+U(I+1,J))/4.
60 CONTINUE

SUM = SUM(T)
PROD = PRODUCT(T)

SUM = 0.0
DO 70 I = 1, N
DO 70 J = 1, M
SUM = SUM + T(I,J)
CONTINUE
70 CONTINUE

TSUM[RR] = SUM

PRINT 100, HR[RR], HR[RR]
PRINT *, 'GIVES SUM = ', TSMU[RR]
CONTINUE

100 FORMAT ('PROBLEM 5 WITH N,M = ', I6, 2X, I6)
PRINT 110, TIME1, TARRAY(1), TARRAY(2)
110 FORMAT ('TIME ', F8.4, ' SECONDS, (' F8.4, ' USER, ', F8.4, ' SYSTEM)')
* 'PRODUCT CALCULATION CAUSED ARITHMETIC OVERFLOW ON THE VAR' STOP
END
PROBLEM 10

REFERENCE: PROBLEMS TO TEST PARALLEL AND VECTOR LANGUAGES
CSD-TR 516, COMPUTER SCIENCE, PURDUE UNIVERSITY
JOHN R. RICE, MAY 1, 1985

REVISED BY JOHN R. RICE AND J. JIN, OCT. 1, 1990

PARAMETER (N=100, K=10)
DIMENSION X(N), N(X), X(N), N(X)
DIMENSION TSUM(N), SUMD(N)
DIMENSION TARRAY(2)
DATA N / 4, 10, 20, 40, 100 /
TIME = DTIME(TARRAY)

DO 90 NN = 1, N
    K = N(NN)
    FORALL (I=1:N, J=1:N) A(I,J) = SIN(I+J)

DO 10 I = 1, N
    DO 10 J = 1, N
        A(I,J) = SIN(PI*FLOAT(I+1))
    10 CONTINUE

DO 50icol = 1, N
   icolmax = xmax Nicol,icol,icolmax)
    Interchange Rows

DO 20 J = Iocol, N
    TEMP(J) = A(J,ICOL)
    A(I,MAX,J) = A(J,MAX,J)
    A(J,icol,J) = TEMP(J)
    20 CONTINUE

DO 40 JROM = Jicol+1, N
    A(J,icol,JROM) = A(J,icol,JROM)/icolMAX

DO 30 K = Iocol+1, N
    A(JROM,K) = A(J,icol,K)-A(J,icol,K)*A(JROM,icol)

30 CONTINUE

30 CONTINUE

DO 50 coloc = 1, N
    colocmax = xmax coloc,icol,icolmax)
    Interchange Cols

DO 20 J = Iocol, N
    TEMP(J) = A(J,icol)
    A(I,icol,J) = A(J,icol,J)
    A(J,icol,J) = TEMP(J)
20 CONTINUE

DO 40 JROM = Jicol+1, N
    A(J,icol,JROM) = A(J,icol,JROM)/icolMAX

DO 30 K = Iocol+1, N
    A(JROM,K) = A(J,icol,K)-A(J,icol,K)*A(JROM,icol)

30 CONTINUE

DO 90 I = 1, N
    SUMA = SUM(A)
    SUMD = SUMD + SUMA
90 CONTINUE

TIME2 = DTIME(TARRAY)
DO 100 N = 1, K
    PRINT *, 'PROBLEM 10 WITH N = ', N
    SUMA = TSUM(N)
    SUMD = SUMD(N)
    PRINT 90, SUMA, SUMD

100 CONTINUE

90 FORMAT ('SUM OF ELEMENTS AND SUM OF DIAGONAL = ',E15.6)
110 FORMAT ('TIME ',F8.4,' SECONDS, ',F8.4,' USER, ',F8.4,' SYSTEM')
STOP
END

FUNCTION xmax (X, N, M,icol,icolmax)
DIMENSION X(N), M,icol,icolmax
xnamax = 1.0E-10
DO 10 J = Iocol, N
    IF (ABS(X(J,icol)) GT XMAX) THEN
        XMAX = X(J,icol)
        INDEX = J
    ENDIF
10 CONTINUE
10 CONTINUE
RETURN
END

REVISIONS OF PROBLEMS TO TEST PARALLEL AND VECTOR LANGUAGES
CSD-TR 516, COMPUTER SCIENCE, PURDUE UNIVERSITY
JOHN R. RICE, MAY 1, 1995

Probs.Proc.Lang
prob.10

TIME2 = DTIME(TARRAY)
DO 100 N = 1, K
    PRINT *, 'PROBLEM 10 WITH N = ', N
    SUMA = TSUM(N)
    SUMD = SUMD(N)
    PRINT 90, SUMA, SUMD

100 CONTINUE

90 FORMAT ('SUM OF ELEMENTS AND SUM OF DIAGONAL = ',E15.6)
110 FORMAT ('TIME ',F8.4,' SECONDS, ',F8.4,' USER, ',F8.4,' SYSTEM')
STOP
END

FUNCTION xmax (X, N, M,icol,icolmax)
DIMENSION X(N), M,icol,icolmax
xnamax = 1.0E-10
DO 10 J = Iocol, N
    IF (ABS(X(J,icol)) GT XMAX) THEN
        XMAX = X(J,icol)
        INDEX = J
    ENDIF
10 CONTINUE
10 CONTINUE
RETURN
END

PROBS.PAR.VECT.LANG  

Prob.11

PARAMETER (NMAX=5000, RBASE=4)
DIMENSION DATA(NMAX), CONT(NBASE), SNUM(NBASE)
DIMENSION TARRAY(2), TIME(NBASE, 4)
DATA PI / 3.1415926 /, SNUM / 6, 200, 1000, 5000 /
TIME1 = TIME(TARRAY)
DO 50 KK = 1, 4
     N = N(KK)
     DO 10 T = 1, N
       DATA(T) = -1/0.1 * 1080 * SIN(I+10.)
       CONTINUE
     DO 20 T = 1, N
       DATA(T) = ALOG(DATA(T)+1.0)
       CONTINUE
     DO 30 T = 1, N
       DATA(T) = ALOG(DATA(T)+1.0)
       CONTINUE
     TIME2 = TIME(TARRAY)
     DO 80 KK = 1, 4
       PRINT *, 'PROBS. PAR. VECT. LANG', PROBS. PAR. VECT. LANG
       DO 70 T = 1, 4
         SNUM(T) = TIME(T)
       70 CONTINUE
       PRINT 80, KK, SNUM(1), SNUM(2), SNUM(3), SNUM(4)
     80 FORMAT (16, F8.4, F8.4, F8.4, F8.4)
     90 CONTINUE
     PRINT 100, TIME2, TARRAY(1), TARRAY(2)
 100 FORMAT ('TIME', 'S8.4', ' SECONDS, ', 'S8.4', ' USER, ', 'S8.4', ' SYSTEM')
END

SUBROUTINE SAVHOM(I, FVAL, N)
DIMENSION FVAL(N)
PRINT *, I, FVAL
RETURN
END

FUNCTION SUM (X, N)
DIMENSION X(N)

**Probs.Par.Vect.Lang**

**prob.12**

**Problem 12**

**Reference:** PROBLEMS TO TEST PARALLEL AND VECTOR LANGUAGES

CSE-TR 516, COMPUTER SCIENCE, PURDUE UNIVERSITY

JOHN R. RICE, MAY 1, 1985

REVISED BY JOHN R. RICE AND J. JING, OCT. 1, 1990

**Parameter**

\( \text{HD} = 400, \text{MD} = 400, \text{HD} + 1, \text{MD} + 1, \text{HD} - \text{MD} + 1, \text{Bases} - 5 \)

**Dimension**

\( \text{A} (\text{HD}, \text{MD}), \text{R} (\text{HD}), \text{C} (\text{MD}), \text{ABIG} (\text{HD}, \text{MD}) \)

**Dimension Type**

\( \text{TARRAY}[1], \text{TP} 1[\text{Bases}], \text{TP} 2[\text{Bases}] \)

**Inputs**

\( \text{HK} (\text{Bases}), \text{MK} (\text{Bases}) \)

**Data**

\( \text{N} / 4, 30, 50, 100, 400 \), \( \text{MK} / 4, 8, 60, 100, 400 \)

**Time**

\( \text{DTIME} (\text{TARRAY}) \)

**DO 70 K = 1, Bases**

\( \text{N} = \text{MK}(K) \)

\( \text{H} = \text{MK}(K) \)

**DO 20 I = 1, N**

\( \text{R}(I) = 1. - I \)

**DO 10 J = 1, N**

\( \text{A}(I, J) = 1 + J \)

**CONTINUE**

**DO 30 J = 1, N**

\( \text{C}(J) = 1 + J \)

**CONTINUE**

**BUILD ABIG FROM PARTS CREATED**

\( \text{ABIG}(1, J) = \text{A}(1, J) \)

**DO 50 J = 1, N**

\( \text{ABIG}(1, J) = \text{C}(J) \)

**CONTINUE**

**TIME2 = DTIME(TARRAY)**

**DO 90 K = 1, Bases**

**STOP**

**END**
Problem 13

REFERENCE: PROBLEMS TO
CSD-TR 516,
JOHN R. RICE
REVISED BY 
FOR N ="", N(K)
1(K)
PARAMETER (NDIM=2100)
DIMENSION A(NDIM), B(1), TARRAY(2)
DIMENSION TARRAY(2),/
DATA N / 1000, 10000
CRITICAL E
FORALL(I=1) A(I) = 1.
TIME1 = DTIME(TARRAY)
DO 30 K = 1, N(K)
DO 20 I = 1, M(K)
A(I) = I/10.(1.)1/
B(I) = LOG(10*A(I))
C(I) = A(I)+B(I)
D(I) = (A(I)+B(I).
30 CONTINUE
E(K) = 0.0
DO 20 I = 1, M(K)
20 END
**Problem 14**

**Reference:** PROBLEMS TO TEST PARALLEL AND VECTOR LANGUAGES
CED-TR 515, COMPUTER SCIENCE, PURDUE UNIVERSITY
John R. Rice, May 1, 1985

Revised by John H. Rice and J. Jing, Oct. 1, 1992

Page 204 of NUMERICAL METHODS, SOFTWARE AND ANALYSIS

**Parameter:** (NSIMP=3, NFUN=3, HACCUR=1)

REAL SUM1(NACCUR), SUM2(NACCUR), SUM3(NACCUR), SUM22(NACCUR)
REAL (HACCUR), SUM33(NACCUR)

INTEGER NSIMP(NACCUR)
REAL B(NACCUR), BG(NACCUR), BG1(NACCUR), B77(NACCUR)
REAL RESULT(FUN, NACCUR, NACCUR, 2)
REAL ARRAY(2), TIME1, TIME2

INTEGER N(NACCUR)

COMMON /SELECT/ JFUN,
DATA N / 30, 35, 50, 75, 100, 150, 200, 300, 500, 1000, 2000 /

**Print:** 'Problem 14'
TIME1 = DTIME(TARRAY)
DO 100 JFUN = 1, NFUN

**Call Parallel:**
BEGIN 'METHOD 1'

CALL EVALS (A1, B1, TRUE, JFUN)
DO 20 L = 1, NACCUR

BEGIN
B(L) = (B1-A1)/H(L)
SUM1(L) = SUM1(P(A1+B(L))*SEQ(1,N(L)-1))
SUM1(L) = 0
DO 10 J = 2, N(L)-1
SUM1(L) = SUM1(L)*F(A1+B(L)*J)
10 CONTINUE
SUM1(L) = (SUM1(L)+B1+B1)*H(L)/2.
RESULT(12,1,1,1) = SUM1(L)

END

20 CONTINUE

END 'METHOD 1'
BEGIN 'METHOD 2'

CALL EVALS (A2, B2, TRUE, JFUN)
DO 50 L = 1, NACCUR

BEGIN
NSIMP(L) = H(L)
IF (MOD(NSIMP(L),2).EQ.1) NSIMP(L) = NSIMP(L)-1
NSIMP(L) = (B2-A2)/NSIMP(L)

SUM2(L) = SUM2(P(A2+H(L))*SEQ(1,NSIMP(L)-1),7)

SUM22(L) = SUM2(F(A2+B(L))*SEQ(2,NSIMP(L)-2))

SUM22(L) = 0
DO 40 J = 2, NSIMP(L)-2
SUM22(L) = SUM22(L)*F(A2+H(L)*J)
40 CONTINUE
SUM22(L) = SUM22(L)+B(L)*SEQ(2,NSIMP(L)-2)
RESULT(2, JFUN, L, 2) = SUM22(L)+B(L)*SEQ(2,NSIMP(L)-2)

RESULT(1-FUN, L, 2, IN(1)) = NSIMP(L)+1

END

50 CONTINUE

END 'METHOD 2'
BEGIN 'METHOD 3'

CALL EVALS (A3, B3, TRUE, JFUN)
DO 90 L = 1, NACCUR

BEGIN
BG(L) = (B3-A3)/INT(N(L)/3)
B77(L) = 7.7455669249*BG(L)/2.
SUM31(L) = SUM3(F(A3+B77(L)+BG(L))*SEQ(1,N(L)/3))
SUM31(L) = 0
DO 80 J = 1, N(L)/3
SUM31(L) = SUM31(L)*F(A3+B77(L)+BG(L)+J/3)
80 CONTINUE
SUM31(L) = SUM31(L)+B(L)*SEQ(1,N(L)/3))

SUM32(L) = 0
DO 70 J = 1, N(L)/3
SUM32(L) = SUM32(L)*F(A3+B77(L)+BG(L)+J/3)
70 CONTINUE
SUM32(L) = SUM32(L)+B(L)*SEQ(1,N(L)/3))

SUM33(L) = 0
DO 60 J = 1, N(L)/3
SUM33(L) = SUM33(L)*F(A3+B77(L)+BG(L)+J/3)
60 CONTINUE
SUM33(L) = SUM33(L)+B(L)*SEQ(1,N(L)/3))

RESULT(3, JFUN, L, 3) = 1*THT(N(L)/1)

END

90 CONTINUE

END 'METHOD 3'

100 CONTINUE

TIME2 = DTIME(TARRAY)

END
DO 160 JFUNK = 1, NFUNCTION
   CALL FVALS (A, B, TRUE, JFUNK)
   PRINT *, 'FUNCTION', JFUNK
   PRINT 110, JFUNK, A, B, TRUE
110   FORMAT ('FUNCTION', I3, A4, 2X, 'QU', 2X, 'TRUE', 'F10.6)
   DO 150 L = 1, NACCUR, NACCUR-1
   PRINT *, 'EVALUATION WITH', N(L), ' INTERVALS, L = ', L
   PRINT 120
120   FORMAT (2X, 'METHOD', 8X, 'ANSWER', 5X, 'NO OF POINTS', 5X, 'ERROR'
     5X, 'LOG ERROR')
   DO 150 NL = 1, NLIM
   PRINT 130, NL, RESULT(JFUNK, L, NL, 1), RESULT(JFUNK, L, NL, 2)
   , TRUE-RESULT(JFUNK, L, NL, 1), ALOG10(MAX(ABS(TRUE-RESULT
     (JFUNK, L, NL, 1)), 1E-20))
130 CONTINUE
150 CONTINUE
160 CONTINUE
170 PRINT 170, TIME, TARRAY(1), TARRAY(2)
170 FORMAT ('TIME ', F8.4, ' SECONDS, ', F8.4, ' USER, ', F8.4, ' SYSTEM')
STOP
END

FUNCTION F (X)
COMMON /SELECT/JFUNK
   IF (JFUNK.EQ.1) F = EXP(X)
   IF (JFUNK.EQ.2) F = SQRT(ABS(X-.2345))
   IF (JFUNK.EQ.3) F = 1.4*X**1./(1.100*X*X)
RETURN
END

SUBROUTINE FVALS (A, B, TRUE, JFUNK)
   IF (JFUNK.EQ.1) THEN
     A = 0.
     TRUE = 1.7320508254
   ENDIF
   IF (JFUNK.EQ.2) THEN
     A = 0.
     B = 1.
     TRUE = .5222899422093
   ENDIF
   IF (JFUNK.EQ.3) THEN
     A = -1.
     B = 2.
     TRUE = 6.29915656054
   ENDIF
RETURN
END
Probs.Par. Vect.Lang
prob.15

PARAMETER (NPTS=10, NMAX=2, NPTS, NUMOTHS=4, H(2*NPTS)
COMMON /POINTS/ N, RASE
REAL XPTS(NPTS, NPTS, 2), COEF(NMAX, NPTS, 2), H(NPTS)
REAL ERHWMX(NPTS, 2), NTHOTS(KNOTSMAX), TARRAY(2), DECAY(KNOTSMAX)
REAL LOGERR, INTERP, RATIO, TMAX, TMAXI
DATA N, 0, 1 /NPTS /, 1.16592654 / TTIME = TIME(TARRAY)
DO 100 N = 1, 2
BEGIN 'NUMBER OF POINTS USED'
DO 80 N = 2, NPTS
BEGIN 'CASES OF POINT DISTRIBUTION'
IF (NMAX.EQ.1) THEN
B(N) = (B-A)/(N-1)
DO 10 J = 1, N
XPTS(J, N, RASE) = A(N-J)*B(N)
10 CONTINUE
ELSE
DO 20 J = 1, N
XPTS(J, N, RASE) = (A+B)/(N-1)*COEF(N-J)*P(1)/(A-B))
20 CONTINUE
ENDIF
DO PARALLEL OVER J, N, RASE: HERMITIAN CUBIC COEFFICIENTS
FORALL(KASE=1, 2; H=NPTS; J=1, 2N, 2)
COEF(J, N, RASE) = P(XPTS(J, N, RASE))
ENDIF
DO 40 J = 1, N
COEF(2*3+1-N, RASE) = P(XPTS(J, N, RASE))
40 CONTINUE
DO 50 J = 1, N
COEF(2*3-J,N, RASE) = P(XPTS(J, N, RASE))
50 CONTINUE
END PARALLEL OVER J, N, RASE
DO PARALLEL OVER N, RASE: EVALUATE ERRORS OF INTERPOLATION
PUT XPTS POINTS INTO NTHOTS ARRAY OF HERMITIAN CUBIC KNOT
DO 60 J = 1, N
NTHOTS(J+1) = XPTS(J, N, RASE)
60 CONTINUE
ADD THE DUMMY POINTS AT EACH END OF TARRAY

PRINT OUT OF INTERPOLATION POINT ARRAY XPTS(J, N, RASE)
SUPPORTED HERE TO REDUCE AMOUNT OF OUTPUT
STOP
END
REAL FUNCTION BCUBIC(J,X,T,KNOTS)

THE BERNSTEIN CUBIC KNOTS ARE IN THE ARRAY T OF LENGTH KNOTS.
THERE ARE 2 SUCH ARRAYS, 1 FOR EACH CASE OF POINT DIST.
J = INDEX OF THE BASIS FUNCTION
X = POINT OF EVALUATION OF THE BASIS FUNCTION

REAL T(KNOTS)
IKNOT = (J+3)/2
IF (MOD(J,2).EQ.1) THEN
   BCUBIC = HERMC(T,KNOTS,X,IKNOT)
ELSE
   BCUBIC = HERMC1(T,KNOTS,X,IKNOT)
ENDIF
RETURN
END

REAL FUNCTION HERMC(T,KNOTS,X,IKNOT)
REAL T(KNOTS)
IF ((X.LE.T(IKNOT)).AND.(X.GT.T(IKNOT-1))) THEN
   HERMC = 0.0
   IF (X.GT.T(IKNOT)) THEN
      DT = T(IKNOT)-T(IKNOT-1)
      DX = T(IKNOT)-X
   ELSE
      DX = T(IKNOT)-T(IKNOT-1)
   ENDIF
   HERMC = (3-2.*DX/DT)*DX**2/DT**2
ELSE
   HERMC = 0.0
ENDIF
RETURN
END

REAL FUNCTION HERMC1(T,KNOTS,X,IKNOT)
REAL T(KNOTS)
IF ((X.LE.T(IKNOT)).AND.(X.GT.T(IKNOT-1))) THEN
   DX = X-T(IKNOT)
   IF (X.GT.T(IKNOT)) THEN
      DT2 = (T(IKNOT)-T(IKNOT+1))**2
      DXX = (X-T(IKNOT+1))**2
   ELSE
      DT2 = (T(IKNOT)-T(IKNOT-1))**2
      DXX = (X-T(IKNOT-1))**2
   ENDIF
   HERMC1 = DXX*DX/DT2
ELSE
   HERMC1 = 0.0
ENDIF
RETURN
END

REAL FUNCTION F (X)
F = 1./((X**25.0))
RETURN
END

REAL FUNCTION EPRI(l)(X)
EPRI = -2.0*X/(X**25.)**2
RETURN
END
Probs.Par.Vect.Lang

prob.16

\[
\begin{align*}
\text{TT}(N) &= 1 \\
\text{ELSE} \\
\text{BACK SOLVE TO GET FOUR SOLUTIONS} \\
\text{TT}(N) &= 9 \\
\text{DO } K &= 1, \text{RAGES} \\
\text{DO } 56 I &= 1, N \\
\text{ITEMP}(I) &= X(I,K) \\
50 &\text{CONTINUE} \\
\text{CALL } \text{SGEMM} \text{ (HOLD, NOIM, N, IVPT, TEMP, 0)} \\
\text{COMPUTE DOTPRODUCT FOR RESIDUE} \\
\text{DOTPROD} &= 0.0 \\
\text{DO } 70 I &= 1, N \\
\text{DOTPROD} &= \text{DOTPROD} + \text{HILBerti}(1, J) * \text{ITEMP}(J) \\
60 &\text{CONTINUE} \\
\text{DOTPROD} &= \text{DOTPROD} + \text{DOTPROD} * \text{DOTPROD} \\
70 &\text{CONTINUE} \\
\text{TRESID}(N,K) &= \text{SQRT(DOTPROD)} \\
80 &\text{CONTINUE} \\
\text{ENDIF} \\
90 &\text{CONTINUE} \\
\text{C TIME2} &= \text{DTIME} \text{(TARRAY)} \\
\text{DO } 120 N &= \text{RAGES, NOIM, RAGES} \\
\text{IF } \text{TT}(N) \text{EQ.} 1.0 \text{ THEN} \\
\text{PRINT *, 'PROBLEM 16 WITH N=', N} \\
\text{PRINT *, '*** THE CONDITION NUMBER IS TOO HIGH' ELSE} \\
\text{PRINT *, 'PROBLEM 16 WITH N=', N} \\
\text{DO } 100 K &= 1, 4 \\
100 &\text{PRINT } 110, K, \text{TRESID}(N,K) \\
\text{ENDIF} \\
110 &\text{FORMAT} \text{((X, 'RESIDUE', 14, 7, X, ':=', 'X, Z18.12))} \\
110 &\text{CONTINUE} \\
\text{PRINT } 130, \text{TIME2}, \text{TARRAY}(1), \text{TARRAY}(2) \\
130 &\text{FORMAT} \text{((TIME, 'SEC', X, 'SEC', X, 'USER ', X, 'SYS'))} \\
\text{STOP} \\
\text{END}
\end{align*}
\]

\[
\begin{align*}
\text{REFERENCE: PROBLEMS TO TEST PARALLEL AND VECTOR LANGUAGES} \\
\text{CSD-JR 516, COMPUTER SCIENCE, PURDUE UNIVERSITY} \\
\text{JOHN R. RICE, MAY 1, 1983} \\
\text{REVISED BY JOHN R. RICE AND J. JUNG, OCT. 1, 1992} \\
\text{C PAGE 152 OF NUMERICAL METHODS, SOFTWARE AND ANALYSIS} \\
\text{C TEST SOLVING HILBERT MATRIX OF SEVERAL SIZES ON 4 RIGHT SIDES} \\
\text{C} \\
\text{PARAMETER (SIZE=5, RAGES=4, NOIM=SIZE*RAGES)} \\
\text{REAL HILB, HILB, X, T, TEMP, WORK, DOTPROD, DOTPROD} \\
\text{REAL TIME1, TIME2, TIME2} \\
\text{INTEGER I, J, N, IVPT} \\
\text{DIMENSION HILBERT(NOIM, NOIM), HILB(NOIM, NOIM), X(NOIM, RAGES)} \\
\text{DIMENSION B(NOIM, RAGES), WORK(NOIM, IVPT), IVPT(NOIM)} \\
\text{DIMENSION TRESID(NOIM, RAGES), T(NOIM)} \\
\text{DIMENSION TEMP(NOIM), TARRAY(2)} \\
\text{C INITIALIZATION: CREATE HILBERT MATRIX.} \\
\text{TIME1} = \text{DTIME} \text{(TARRAY)} \\
\text{DO } 10 I &= 1, NOIM \\
10 &\text{DO } J &= 1, NOIM \\
\text{HILBERT}(1,J) &= 1.0/(I+J-1.0) \\
10 &\text{CONTINUE} \\
\text{DO } 90 N &= \text{RAGES, NOIM, RAGES} \\
\text{DEFINE HILBERT MATRIX , AND FIRST 1 RIGHT RAND SIDES} \\
\text{DO } 20 J &= 1, N \\
20 &\text{DO } I &= 1, N \\
\text{B}(I,J) &= 0.0 \\
\text{B}(I,J) &= 1.0 \\
20 &\text{CONTINUE} \\
\text{HILB}(I,J) &= \text{HILBERT}(I,J) \\
20 &\text{CONTINUE} \\
\text{B}(I,J) &= 1.0 \\
\text{COMPUTE 4TH SIDE TO MAKE SOLUTION = 1} \\
\text{DO } 30 J &= 1, N \\
30 &\text{DO } I &= 1, N \\
\text{B}(I,J) &= 0.0 \\
\text{B}(I,J) &= \text{B}(I,J) * \text{HILB}(I,J) * (I+J-1.0) \\
30 &\text{CONTINUE} \\
\text{ASSIGN X = B} \\
\text{DO } 40 J &= 1, N \\
40 &\text{DO } I &= 1, RAGES \\
\text{A}(1,3) &= \text{B}(I,J) \\
40 &\text{CONTINUE} \\
\text{USE LINPACK AS PORTFOR 66 ROUTINES} \\
\text{CALL } \text{SGEMM} \text{ (HOLD, NOIM, N, IVPT, MCOND, WORK)} \\
\text{IF } ((1=MCOND).EQ.1.0) \text{ THEN} \\
\text{END}
\end{align*}
\]
PROBLEM 17

REFERENCE: PROBLEMS TO TEST PARALLEL AND VECTOR LANGUAGES
CSE TR 516, COMPUTER SCIENCE, PURDUE UNIVERSITY
REVISED BY JOHN R. RICE AND J. JING, OCT. 1, 1989

PARAMETER (LIQ=100000, LIMCPU=1,1ASES=5)
COMMON /ALGORITHM/ LOOP, FILEM (1ASES), NRESP (1ASES)
COMMON /ALGORITHM/ AREAS (1ASES), XBOUND (1ASES)
COMMON /PROBLEM/ A, B, EPS
COMMON /CONTROL/ ARE, NOUNDA, DISCARD, FINISH
COMMON /QUEUE/ LEADER, HQ, LASTQ, INEXT (LIQ), AEST (LIQ), BOUND (LIQ)
COMMON /QUEUE/ INFLRECT (LIQ), COTAN (LIQ), COTR (LIQ), COTL (LIQ)
COMMON /QUEUE/ FLRECT (LIQ), LEF T (LIQ), IQVIEW (LIQ), IOTHER, IPQ
COMMON /QUEUE/ WAITING, NXTNEXT, IFBETO, IXTAIL, INEXT, COUNT
COMMON /PROCES/ ACSTATE (LIMCPU), CSTATE (LIMCPU), AREA (LIMCPU)
COMMON /PROCES/ AREAS (LIMCPU), BOUN (LIMCPU), BOUND (LIMCPU)
COMMON /PROCES/ ASIZE (LIMCPU), RRETURN (LIMCPU), KSIZE (LIMCPU)
COMMON /PROCES/ DLX (LIMCPU), KMID (LIMCPU), MWID (LIMCPU)
COMMON /PROCES/ COTAN (LIMCPU), COSTAN (LIMCPU), INPL (LIMCPU)

DIMENSION ENERGY (2)
DATA KFU / 10, 50, 100, 500, 1000 /
DATA MESS / 10, 100, 1000, 100000, 1000000 /
DATA LEFT, CENTER, RIGHT / 1, 2, 3 /
DATA A, B / 0.1, 1 /

TIME1 = TIME (ARRAY)

FOR-LOOP FROM 1 TO 1ASES
DO 600 LOOPS = 1, 1ASES

INITIALIZES THE ALGORITHM
AREA = 0
EPS = 1 ./ EPS (LOOP)
DISCARD = EPS (B-A)

FIND THE INITIAL INTERVAL LENGTH
DXI = 1.
HQ = (B-A)/DXI
IF (DXI=MQ) LT. B-A HQ = HQ + 1
DXI = (B-A)/HQ
LASTQ = HQ
LEADER = 1
COUNT = HQ

FIRST SET OF QUANTITIES FOR INITIAL INTERVALS
DO 100 K = 1, HQ

RIGHT (K) = A + K * DXI
LEFT (K) = XRIGHT (K) = DXI
FRIGHT (K) = F (XRIGHT (K))
FLEFT (K) = F (XLEFT (K))
AEST (K) = 0.5 * (XLEFT (K) + XRIGHT (K))
AREA = AREA + AEST (K)
COTAN (K) = DXI / (FRIGHT (K) - FLEFT (K))
TRIGT (K) = A 1
LEFT (K) = K - 1
100 CONTINUE

Determine intervals where cotangent differences change sign
IF (HQ . EQ. 1) GOTO 201
COT (K) = COTAN (K)
COT (K) = COTAN (K) - 1)
IF (HQ . LE. 2) GOTO 201
DO 200 K = 2, HQ-1
COT (K) = COTAN (K)
COT (K) = COTAN (K) + 1)
IF (ABS (COT (K) - COT (K)) . LE. 1) THEN
INFLRECT (K) = 0
ELSE
INFLRECT (K) = CENTER
INFLRECT (K-1) = LEFT
INFLRECT (K+1) = RIGHT
ENDIF
200 CONTINUE
201 CONTINUE

NOW COMPUTE THE INITIAL ERROR BOUND
BOUND (A) = 0.
DO 300 K = 1, HQ
IF (INFLRECT (K) . EQ. CENTER) THEN
COTREC = 1 ./ COTR (K) + 1 ./ COTL (K)
BOUND (K) = DXI * ABS (FLEFT (K) - FRIGHT (K) + COTREC)
ELSE
DF = FRIGHT (K) - FLEFT (K)
IF (INFLRECT (K) . EQ. LEFT) THEN
BOUND (K) = TRIANGLE (COT (K), COTAN (K), 0, DXI, DF)
ELSE
BOUND (K) = TRIANGLE (COT (K), COTAN (K), COTR (K), DXI, DF)
ENDIF
BOUND (K) = BOUND (K) + BOUND (K)
300 CONTINUE

FINALLY FREE ALL INTERVALS AND MARK THEM AS IN THE QUEUE
DO 400 K = 1, HQ
IFREE (K) = . TRUE.
400 CONTINUE
IFREE = .TRUE.
ITREE = .TRUE.
IP = 1

FIX ITEMS FOR END INTERVALS NOT SET CORRECTLY ABOVE
LEFT (1) = LIMIT
RIGHT (HQ) = LIMIT
INFLRECT (HQ) = LIMIT

SECOND SET OF QUANTITIES FOR INITIAL INTERVALS
COT (K) = 0.
COTR (K) = 0.
INFLRECT (K) = 0

EXIT

END
HEXTQ, ITFREE, lOT, TAILINC, NEXT, COl'lfT

PRELIMINARY QUANTITIES

DX(IP) = .5*(RIGHT(IP) - LEFT(IP))
XMID(IP) = RIGHT(IP) - DX(IP)
PHID(IP) = FHMID(IP)
COTAN(IP) = ABS(DX(IP)/(FHID(IP) - FLEFT(IP)))

CHECK INTERVAL SITUATION AND SELECT AREA FORMULAS

IF (INFLECT(IP) .EQ. 0) THEN
BOUNDL(IP) = TRIANGLE(COTAN(IP),COTAN(IP),COTAN(IP),
  * DX(IP),FLEFT(IP) - FHID(IP))
BOUNDR(IP) = TRIANGLE(COTAN(IP),COTAN(IP),COTAN(IP),COTAN(IP),
  * DX(IP),(FHID(IP) - FRIGHT(IP))
INFHL(IP) = 0
ELSE
CALL SPECIAL(COTAN(IP),COTAN(IP),COTAN(IP),COTAN(IP),
  * INFLECT(IP),IP)
ENDIF

CHECK DISCARDING OF INTERVALS

IF (BOUNDL(IP) .LT. DISCARD-1&(IP)) IRETURN(IP) = 1
IF (BOUNDR(IP) .LT. DISCARD-1&(IP)) IRETURN(IP) = 1

SUBROUTINE AREAS(IP)

SUBROUTINE SPECIAL(CLL, CL, CR, CH, INFLECT, IP)

SUBROUTINE AREAL(IP)

SUBROUTINE DX(LIMCPU, XMID(LIMCPU), FHID(LIMCPU)

SUBROUTINE FMID(IP)

SUBROUTINE XBASE(IP), XBASE(IP)
SUBROUTINE PUT(IP)

PARAMETER (LIMCPU=100000, LIMCPU=1, KASES=5)

COMMON /ALGORITHM/ LOOP, NFUN(KASES), NPS(KASES)

COMMON /ALGORITHM/ AREA(KASES), XBOUND(KASES)

COMMON /PROBLEM/ A, B, EPS

COMMON /CONTROL/ AREA, BOUND, DISCARD, FINISH

COMMON /QUEUE/ LEADER, NQ, LASTQ, INEXT(IP), AEST(LIMCPU), BOUND(LIMCPU)

COMMON /QUEUE/ INQUEUE(LIMCPU), LEFT(IP), INEXT(LIMCPU), COUNT

COMMON /PROC/ ACHANCE(LIMCPU), BCHANGE(LIMCPU), AREA(LIMCPU), BOUND(LIMCPU)

COMMON /PROC/ IASSIGN(LIMCPU), IRETURN(LIMCPU), PQUEUE(LIMCPU)

COMMON /PROC/ DX(LIMCPU), XMID(LIMCPU), PHID(LIMCPU)

COMMON /PROC/ COVAR(LIMCPU), COTAN(LIMCPU), INFIL(LIMCPU)

IF (LEADER .EQ. 1) THEN
    RETURN
ELSE
    IASSIGN(IP) = LEADER
    INQUEUE(IP) = .FALSE.
    LEADER = INEXT(LEADER)
ENDIF
RETURN
END

SUBROUTINE GET(IP)

PARAMETER (LIMCPU=100000, LIMCPU=1, KASES=5)

COMMON /ALGORITHM/ LOOP, NFUN(KASES), NPS(KASES)

COMMON /ALGORITHM/ AREA(KASES), XBOUND(KASES)

COMMON /PROBLEM/ A, B, EPS

COMMON /CONTROL/ AREA, BOUND, DISCARD, FINISH

COMMON /QUEUE/ LEADER, NQ, LASTQ, INEXT(IP), AEST(LIMCPU), BOUND(LIMCPU)

COMMON /QUEUE/ INQUEUE(LIMCPU), LEFT(IP), INEXT(LIMCPU), COUNT

COMMON /PROC/ ACHANCE(LIMCPU), BCHANGE(LIMCPU), AREA(LIMCPU), BOUND(LIMCPU)

COMMON /PROC/ IASSIGN(LIMCPU), IRETURN(LIMCPU), PQUEUE(LIMCPU)

COMMON /PROC/ DX(LIMCPU), XMID(LIMCPU), PHID(LIMCPU)

COMMON /PROC/ COVAR(LIMCPU), COTAN(LIMCPU), INFIL(LIMCPU)

IF (INQUEUE(IP) .EQ. 0) THEN
    COUNT = COUNT + 1
    ACHANCE(IP) = AREA(IP)
    BCHANGE(IP) = BOUND(IP)
    IF (COUNT .GT. 1) THEN
        INEXT(IP) = LASTQ
        LASTQ = LASTQ + 1
        INQUEUE(IP) = .FALSE.
        IP = IRETURN(IP)
        ACHANCE(IP) = AREA(IP)
        BCHANGE(IP) = BOUND(IP)
    ELSE
        IP = INEXT(IP)
        COUNT = COUNT - 1
    ENDIF
ELSE IF (RETURN(IP) .EQ. 0) THEN
    IF (RETURN(IP) .EQ. 0) THEN
        IP = IP + 1
        RETURN(IP) = IP
    ELSE
        RETURN(IP) = RETURN(IP)
    ENDIF
ENDIF
RETURN
END

REAL FUNCTION F (X)

PARAMETER (LIMCPU=100000, LIMCPU=1, KASES=5)

COMMON /ALGORITHM/ LOOP, NFUN(KASES), NPS(KASES)

COMMON /ALGORITHM/ AREA(KASES), XBOUND(KASES)

F = 1.0
DO 10 J = 1, NFUN(LOOP)
    F = COS(PI/SIN(X))*SIN(X*SIN(X*X))
10 CONTINUE
RETURN
END