THE EVALUATION OF A MODEL TO PREDICT THE PERFORMANCE OF AN APPLICATION PROGRAM IN AN UNKNOWN ENVIRONMENT

By

NATHAN O. LANGSTON

Bachelor of Science

Oklahoma Christian College

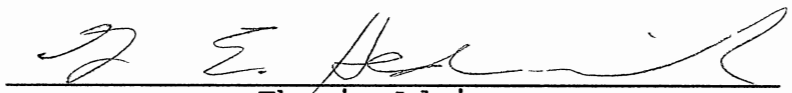
Oklahoma City, Oklahoma

1981

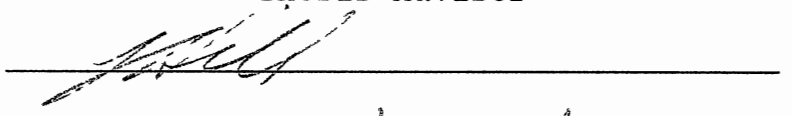
Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
December, 1991

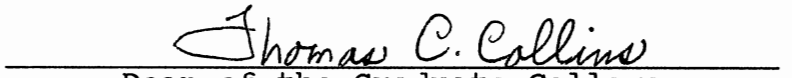
THE EVALUATION OF A MODEL TO PREDICT THE PERFORMANCE OF AN APPLICATION PROGRAM IN AN UNKNOWN ENVIRONMENT

Thesis Approved:



Thesis Advisor





Dean of the Graduate College

ACKNOWLEDGMENTS

I wish to express sincere appreciation to Dr. G. E. Hedrick for his encouragement and advice throughout my graduate program. A special thanks goes to Dr. Steve Hill who provided me with consultation and guidance in writing the thesis. Thanks also go to Dr. Mansur Samadzadeh for serving as a substitute committee member at the last moment.

To Conoco, Inc., I extend sincere thanks. This study would not have been possible without their financial assistance and support.

To all friends and relatives who continuously prompted me for graduation dates, thanks for keeping me focused on the goal. To my children who had many activities and dinners without their dad, thanks for being patient.

Finally, my deepest thanks to my wife who supported me in this effort from the start. Without her, none of this would have been possible.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Figure Page

CHAPTER I

BENCHMARKING TECHNIQUES

Defining The Benchmark

The purpose of a computer benchmark is to determine what computing environment best fits the processing needs of a particular user. To do this three components are needed: a benchmark application, a timing associated with the application, and a set of environments to be tested. Environment in this context refers to a computer's hardware and associated software needed to run the application program. Perhaps the most difficult task in utilizing a benchmark is defining the benchmark program itself [Wilson, 1988]. In general, there are two schools of thought as to what this program should be [Green, 1987; Senson, 1987]. One school of thought is for the user to run a generally accepted synthetic benchmark on various machines to obtain performance characteristics. The second is to run the user's existing applications, or a subset thereof, in the environments. Each technique has specific problems described below.

Synthetic Benchmarks

Synthetic benchmarks are attractive because they test specific areas of the computer, they require little set-up time, and they are ported easily from one machine to another [Green, 1986]. Examples include the Whetstone benchmark which tests a computer's performance of numerical computations and floating-point operations, the Dhrystone benchmark which tests integer computations found in the commercial and system programming environment, the Linpack benchmark which tests mathematical and scientific computations, and the Livermore Loops which tests input/output, graphics, and memory management tasks.

One of the problems with synthetic benchmarks is that the benchmarks are designed to test specific functions. The Linpack benchmark is a subset of the Linpack package which provides tools for the solution of linear equations [Dongarra, 1988]. Consequently, these routines have been optimized to perform a specific task on a variety of machines. In contrast, application programs generally are written and modified over time to function as efficiently as possible within a single processing environment. Expecting the results of the benchmark program to match that of existing applications often proves to be overly optimistic [Schay, 1990].

Another problem associated with a synthetic benchmark is relating the timing obtained to an entire suite of

application programs. The "snapshot" derived from a synthetic benchmark is only indicative of how the benchmark program itself will perform on the machine tested [Lindsay, 1986]. Unless the application programs mirror the snapshot taken, the timings realized by the application programs may differ widely from those of the benchmark [Smith, 1986].

User Applications

The goal of a user developed benchmark is to take multiple snapshots of the environments to be benchmarked in an attempt to determine how a wide variety of the user's applications will perform in the benchmarked environments. This is achieved by defining a suite of dissimilar application programs to be used as the benchmarking set [Borovits, 1984]. The advantage of this method is that the results obtained from the benchmark are more indicative of the entire application program set to be run in the environment [Borovits, 1984]. The problem with this approach is assuring a comprehensive benchmark, given the time necessary to define the benchmark suite and the nature of the computer environments to be tested.

The time necessary to develop a benchmark from existing applications is typically significantly large. Two computer evaluations, one by the Monsanto Oil Company [Shaw, 1987] and one by Conoco Inc.[1] provide examples for this

---

[1]The author was a member of the evaluation team

observation.

In defining the benchmark suite both Monsanto and Conoco took a similar approach which consisted of four steps: accessing the needs of the company, ranking these needs, creating the benchmark, and evaluating the results. The successful completion of each step is necessary to insure a comprehensive benchmark. Monsanto's effort took eight months to complete, while Conoco took ten months to accomplish the task.

The time necessary to insure a complete benchmark suite is in itself a problem. Studies have shown that the life span of an application system is between three and five years [Sensen, 1986]. This would imply that a benchmark suite that takes up to a year to develop would be measuring applications that are nearing maturity and may not reflect the current processing needs.

Another concern of a user-developed benchmark is the complexity of the benchmark itself. The objective of a user- developed benchmark is to measure the performance of an unknown environment in all areas of computing that the existing applications utilize [Senson, 1986]. However, performance measurement tools and techniques are seldom used to ensure some single aspect of the benchmark does not dominate the results [Lindsay, 1987]. Conoco's benchmarking effort serves as an example of this deficiency.

Conoco's benchmark consisted of approximately 10,000

lines of FORTRAN code used in existing seismic application programs. The initial run of this benchmark on the three environments evaluated, indicated that the existing environment was more efficient than two of the test environments. This was surprising since the two environments being evaluated generally were recognized as being more capable of performing the type of processing being evaluated. The problem was resolved when it was discovered that 90% of the benchmark's time was spent executing one subroutine. Only after Conoco decided to utilize vendor supplied vector libraries on the two machines being evaluated was any improvement in timings realized.

This experience highlights another area of concern. It is difficult to determine how much machine-dependent optimization is appropriate for a benchmark suite of programs. One could argue that intensive optimization of the benchmark suite is appropriate because the actual application on the chosen environment will most likely be optimized over its entire life cycle. In fact, some optimization may be necessary to ensure the benchmark will run on the test environments. Unlike synthetic benchmarks which are designed to be portable between environments, application programs generally are written and tuned to take advantage of the architecture where they reside. If a particular application program or a portion of the program is to be used for a benchmark, it must be generalized to run

on other machines. This tuning of the application program for the test environments can affect the predicted results.

These problems were encountered with Conoco's benchmark effort. Each of the new environments evaluated by Conoco had internal vector processing capabilities, however the existing applications ran on a non-vector machine utilizing attached array processors. Calls to the array processors had to be rewritten and the code restructured to run in a vector, rather than a non-vector environment. Because it was difficult to measure the affect the programming changes had on the timings obtained, the results of the benchmark were questionable.

In summary, user developed benchmarks, although generally accepted as more indicative of overall performance than synthetic benchmarks, have inherent problems. The time necessary to identify the key applications, to ensure portability, and to understand the processing characteristics of the benchmark are all factors that must be addressed to ensure a true representation of all programs within an application area.

An Alternative Approach

Due to the problems inherent with the two existing benchmark techniques, this thesis suggests a modified approach for evaluating computers. In essence the method correlates the CPU timings of a user's application program

to a synthetic benchmark and then uses the CPU timing of the benchmark to predict the CPU time of the application program on other machines. The basic hypothesis of the thesis is as follows:

> Given the CPU time of an application program and a standard benchmark program in a particular set of environments, the run time of the application program in an environment outside the set can be predicted by the run time of the benchmark program in the environment outside the set.

The advantage of this approach is that it incorporates the best aspects of both traditional benchmarking techniques. All application programs are considered; yet the evaluation of diverse computing environments is made simpler since the only program being ported to the evaluated computers is a synthetic benchmark. In fact, in some cases the timings of the synthetic benchmark are available in the published literature (see, for examples, [Dongarra, 1988]).

The remainder of this thesis is devoted to defining and testing this "Predictability Model". Chapter II defines the model, Chapter III defines the programs and environments used to test the model, Chapter IV documents the results, and Chapter V details future work in this area.

CHAPTER II

DEFINITION OF THE MODEL

Overview of the Model

The method described in this chapter and used to predict the performance of an application program, is dependent upon the correlation that exists between a benchmark program's CPU time and the desired application program's CPU time. To obtain this correlation, a model is defined consisting of three independent components: an application/benchmark combination, a set of environments and a predictability equation. These three components are considered independent since each can affect the degree of predictability that the model achieves.

The remainder of this chapter describes the prescription of the Predictability Model and gives suggestions on how to improve the model's ability to predict an application's time. The validity of these suggestions is demonstrated in Chapter IV.

Defining the Benchmark and Application Programs

The first step in defining the Predictability Model is

to select the benchmark and the application programs. Two factors should be kept in mind while completing this step: One, the processing profile of the benchmark should match that of the application program and two, the benchmark should be small and self-contained.

Matching the processing profile of a synthetic benchmark with a set of application programs may prove to be difficult but can potentially improve the predictability performance of the model. In a general purpose environment where several diverse types of application programs may run, several benchmarks should be obtained each representing a different set of application programs. For example, the benchmark used in one model might predict the performance of data base applications while another would predict the performance of applications that have a high degree of vector processing. Segregating application programs into groups of similar applications eases the task of selecting the benchmark.

The benchmark selected should be written in the same language as the application program to be tested and should perform similar tasks. This is in recognition of the fact that the compiler is an equal partner with the hardware in the determination of the speed of the total processing environment. For example, if the application programs are written in FORTRAN and perform a high degree of vector processing, then the LINPACK benchmark (written in FORTRAN

and has a high degree of vectorization) would be a better choice than a benchmark that simply measures the performance of variable assignments.

Finally, the benchmark selected should be small, self-contained and easily transportable from one machine to another. Making the program self-contained will ensure that the results obtained are not influenced by programming or data changes that are necessary to make the program run. Keeping the benchmark small makes transporting the program from one machine to another easier.

Selecting the Environments

The environments used in the model are divided into two sets: the available set and the testing set. The available set consists of all the computer environments available on which both the application program and the benchmark have run. The available set should consist of at least three environments; however, these environments can be located on the same machine. For example; running the application and benchmark programs under two different compilers on the same machine would constitute different environments as defined in this work.

The testing set is simply the set of computer environments to be evaluated. As was the case with the selection of the benchmark program however, the degree of predictability can be improved by matching the architecture

of the environments in the testing set to that of the available set. For example, if all of the environments in the available set utilize virtual memory, then in most cases the Predictability Model will perform better if the testing set environments are also virtual memory machines.

The Predictability Equation

The final element of the Predictability Model is the predictability equation. For this work the following predictability equation was used:

$$A = \beta_0 + \beta_1 * B$$

where β_0 and β_1 are constant coefficients defined below and independent of the environment. B is the timing of the benchmark in the testing set and A is the predicted time of the application program in the testing set. In general, we will solve for A, knowing B.

The values of β_0 and β_1 are determined from pairs of A's and B's which we have determined from the available set. Knowing the (A, B) pairs, we use the least squares linear regression algorithm to determine the values of β_0 and β_1. Since the predicted application time A is exclusively dependent on the benchmark CPU time B, β_0 is always set to zero[2]. Inputs to the least squares algorithm are the CPU times A of the application program and the CPU times B of

---

[2]Computing the intercept always to be zero is preferred because it insures that the predicted time will be a positive number Mathematically, it is appropriate to force a zero intercept since one would expect that when the independent variable (the benchmark) is zero, then the dependent variable (the application program) will be zero also.

the benchmark program that reside in the available set.

The least squares method is used to obtain the coefficients for the predictability equation because it is an accepted method for curve fitting that is readily accessible to others who may wish to use the methodology suggested in this work. Furthermore, the method is fairly robust and stable for small fluctuations in the input data. This is not to say that other curve fitting techniques might not be used for the predictability equation, indeed one other method is explored in Chapter IV.

Example

As a contrived example of the Predictability Model, assume we have an available set consisting of three computing environments; X_1, X_2 and X_3 and a testing set consisting of the computing environment Y. The CPU time, A, for an application program and the CPU time, B, for a benchmark program have been obtained for each of the three environments in the available set. In addition, the benchmark program has also been run in the testing set environment, Y, and a CPU time, B, has been obtained. These timings are presented in TABLE I. What then is the predicted CPU time, A, of the application program in the testing environment Y ?

To determine the predicted time, the available set system of equations shown in TABLE II is solved for β_0 and

TABLE I

EXAMPLE PREDICTABILITY MODEL VARIABLES

Sets	Application Program Times (A)	Benchmark Program Times (B)
Available Set		
X_1	100	10
X_2	200	20
X_3	300	30
Testing Set		
Y	???	40

TABLE II

EXAMPLE PREDICTABILITY MODEL EQUATION

$A = \beta_0 + \beta_1 * B$
$100 = \beta_0 + \beta_1 * 10$
$200 = \beta_0 + \beta_1 * 20$
$300 = \beta_0 + \beta_1 * 30$

β_1. In this example, $\beta_0 = 0$ and $\beta_1 = 10$. These values along with CPU time (B = 40) of the benchmark in the testing environment, Y, are input into our predictability equation:

$$A = \beta_0 + \beta_1 * B$$

In this example, the equation would be:

$$A = 0 + 10 * 40$$

and the predicted CPU time, A, of the application program in the testing environment, Y, is determined to be 400.

CHAPTER III

TESTING THE PREDICTABILITY MODEL

Introduction

Chapter II defined the prescription, or makeup, of the Predictability Model. This chapter will define the procedures followed to test the validity of this prescription. In Chapter IV the results of these standards will be reviewed to determine how well the Predictability Model actually performed. The standards described in this chapter include a defined set of computer environments used in this study, a set of application and benchmark programs, an established group of available and testing sets, and a definition of the issues to address to determine the validity of the model. These standards are discussed below.

Defining the Computer Environments

Ten environments consisting of CRAY and CDC CYBER computers were used to test the Predictability Model. These environments are listed in TABLE III.

Defining the Application Programs

Ten application programs were used to test the

TABLE III

COMPUTING ENVIRONMENTS

Environment ID	Machine	Operating System	Compiler	Vector Facility	Memory
C77v	Cray XMP14	COS	CFT77	Yes	Real
CFTv	Cray XMP14	COS	CFT	Yes	Real
C77nv	Cray XMP14	COS	CFT77	No	Real
CFTnv	Cray XMP14	COS	CFT	No	Real
NVEv	Cyber 990	NOS/VE L700	VFTN	Yes	Virtual
NVEnv	Cyber 990	NOS/VE L700	FTN	No	Real
N990	Cyber 990	NOS L688	FTN5	No	Real
N865	Cyber 865	NOS L688	FTN5	No	Real
NV815	Cyber 815	NOS/VE L700	FTN5	No	Real
N815	Cyber 815	NOS L700	FTN	No	Virtual

Predictability Model. One of the programs was written for this work, one was obtained from Cray Research, and eight were obtained from the Seismic Research Department of Conoco Inc. All of the programs are written in FORTRAN. TABLE IV summarizes the major processing of each program.

The CPU time for each application in all of the defined environments was obtained and is listed in TABLE V[3].

The code for the Black Box program and the Subtest program is listed in the Appendix A in its entirety. The

[3]In all cases, the CPU time was obtained by having the application call the system clock immediately upon entering the application and immediately before exiting Subtracting the time at entry from the time at exit eliminated any CPU time that might have been incurred from the initial load All timings were obtained on fully configured systems running production loads, however the clock routines used measured only the CPU time of the application calling the routine Other applications running concurrently with the tested applications should not have affected significantly the timings obtained

TABLE IV

APPLICATION PROGRAM SUMMARY

Program Name	Origin	Predominant Processing
Subtest	User Written	Vector Assignment Vector Multiply
Black Box	Cray Research	Vector Add Vector Multiply
SMMTRIN	Conoco	Vector Assignment
SMMTM	Conoco	Vector Assignment
SMMSTK	Conoco	Vector Gather/Scatter
SMMXP	Conoco	Vector and Scaler Arithmetic
SMMNMO	Conoco	Vector Assignment Vector and Scaler Arithmetic
SMMTDF	Conoco	Multiple Vector Arithmetic
SMDSEQT	Conoco	Multiple Vector Arithmetic Vector Gather/Scatter
SMMBPF	Conoco	Multiple Vector Arithmetic Vector Gather/Scatter

source code for the Conoco written programs in its entirety is considered company confidential; however, the relevant excerpts from the code detailing the major processing is listed also in Appendix A.

Defining the Benchmark Programs

Five benchmark programs were selected to test the Predictability Model. One of the programs was the Linpack benchmark [Dongarra, 1979] and four of the programs were subsets of the Lindsay benchmark [Lindsay, 1987].

TABLE V

CPU TIMES OF THE APPLICATION PROGRAMS
IN THE TESTING ENVIRONMENTS

	SMMBPF	SMDSEQT	SMMTDF	SMMNMO	SMMXP	SMMSTK	SMMTM	SMMTRIN	Black Box	Sub Test
C77v	110 36	71.79	60.12	9.53	9.45	0.75	0.42	0 24	0.72	0 10
CFTv	130 25	95 45	85 09	12 01	8 96	0.84	0.50	0.29	1 31	0 12
C77nv	113 93	192.89	169 03	35.19	10 08	2.39	1.01	0.69	1 70	0 14
CFTnv	150 40	72.68	465.79	50 83	18.03	7.22	3.21	2.47	3 74	0 24
NVEv	176 53	132.38	90.03	39 63	13.27	1.69	0.82	0 43	1.70	0 40
NVEnv	473.98	3259 06	3446.11	199 83	69 05	39.27	18.71	14.92	18 74	1.16
N990	717 29	3439 77	3614 25	210 09	92.28	44.02	20 52	16.99	19.41	0 95
N865	1567 69	7007 14	7344 55	412 65	199.58	88.50	39.59	32.91	44 26	1 63
NV815	7514 30	45900 71	48169 75	3250.20	1114 11	655.05	328.62	270.17	296.08	16.15
N815	13943.12	63797.87	67180 33	4039 14	1539.70	717 77	318.53	257.17	407.12	16.14

The Linpack benchmark was chosen for three reasons: first, it is accepted generally as a measurement for large-scale computers [Green, 1987; Smith, 1987]; second, it is written in FORTRAN and is easily transportable; and finally, it is largely vectorizable, similar to the applications to be measured.

Designed in 1979, the original intent of the Linpack Benchmark was to provide users of the Linpack mathematical package an approximation of the execution times required to solve a system of linear equations using the package. Since then the Linpack benchmark has been run in over 200 different computer environments [Dongarra, 1988].

The Linpack benchmark spends 90% of its processing time solving the vector equation Y=Y+sX for average vector lengths of 66 [Dongarra, 1988]. The benchmark generates its own data and produces a report listing the time spent in each module of the program. The benchmark was modified for this work to report a total CPU execution time.

The Lindsay benchmark was included because of the unique method it uses to evaluate the relative speed of a computer. Developed in 1987 by David S. Lindsay of National Advanced Systems, the Lindsay benchmark attempts to measure the execution time of individual vector instructions rather than measuring the execution of an entire process [Lindsay 1987]. The Lindsay benchmark is written in FORTRAN and is composed of twenty-four different modules that perform atomic vector operations for vector lengths of 1, 50, 200, and 500. An atomic instruction is a single vector operation such as Y=Y+sX.

The benchmark measures the time necessary to perform an atomic instruction for a specified vector length by subtracting the CPU time of a subroutine having 20 atomic units of a given vector length from a subroutine having 40 units of the same vector length. These timings are then calibrated and an estimated time for 100,000 iterations is given. The timing used in this work is the timing for one iteration of an atomic unit based on the 100,000 iteration estimate.

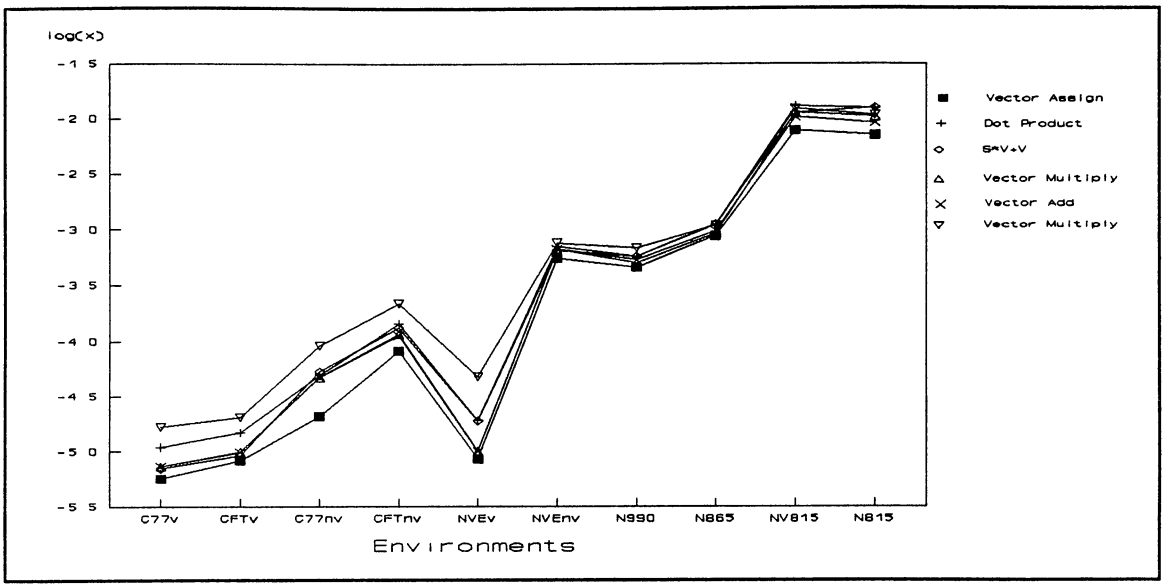


Figure 1. Timing of Selected Lindsay Atomic Units of Vector Length 200 on All Defined Environments

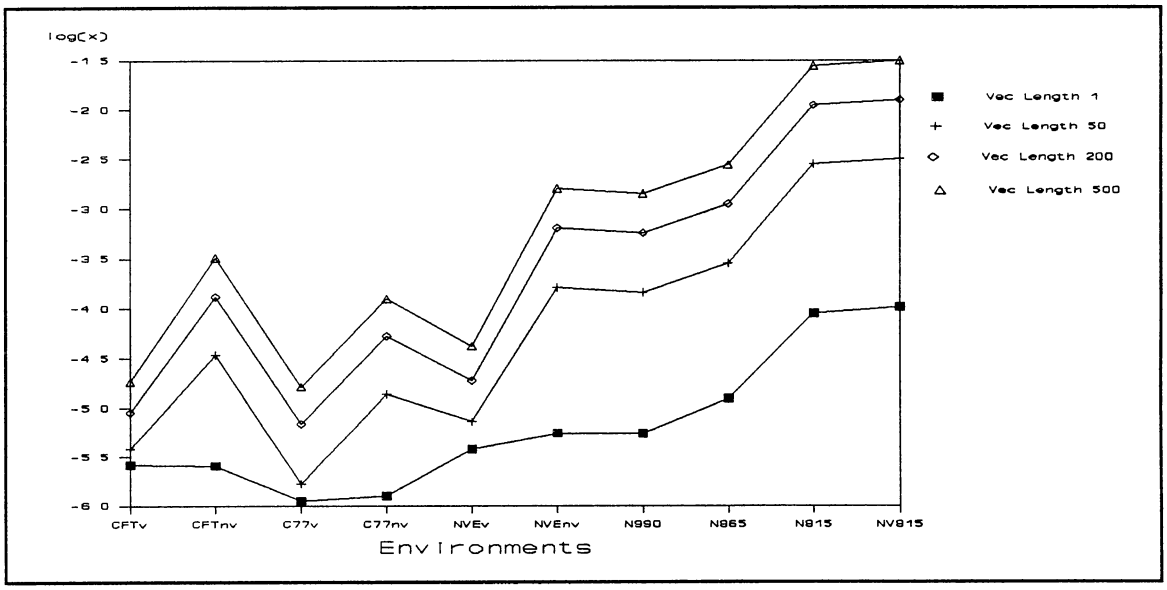


Figure 2. Timing of the Lindsay S*V+V Atomic Unit on All Defined Environments

Two observations were made while evaluating the timings of the Lindsay Benchmark: one, the timings between the various atomic units tracked each other well; and two, the timings obtained for an individual atomic unit were similar with the exception of vectors of length one. These observations are illustrated in Figures 1 and 2. Figure 1 gives the timings of various atomic units for a vector length of 200 while Figure 2 plots the times of the S*V+V atomic unit for a range of vector lengths.

Given the similarity in performance of the Lindsay atomic units, the four Lindsay modules presented in TABLE VI were selected as representative of the entire suite. Their timings in the defined set of environments are listed in TABLE VII. These times are given in seconds per atomic unit per vector length.

Defining the Available and Testing Sets

The available and testing sets were defined for two common situations. The first situation assumes the user of the model has a slow or inadequate computing environment for a particular type of application and desires to know how the program will perform in an environment generally considered better equipped to handle his/her processing needs. The second is the converse, the user has an application that performs extremely well in a given environment but wants to know if performance will remain acceptable when run in an

TABLE VI

LINDSAY BENCHMARKING MODULES

Module Name	Instruction Performed	Vector Length
VA200	Vector Assignment	200
DP1	Dot Product	1
DP200	Dot Product	200
SVV200	S*V+V	200

TABLE VII

CPU TIMES OF BENCHMARK PROGRAMS

	LINPACK	VA200	DP1	DP200	SVV200
C77v	0 86	5 70E-06	6 60E-06	1.10E-05	6.97E-06
CFTv	1 33	8.25E-06	6.10E-06	1.48E-05	9.15E-06
C77nv	2.05	2.07E-05	1 33E-06	4.93E-05	5.31E-05
CFTnv	4 38	8.13E-05	2 11E-06	1.43E-04	1 31E-04
NVEv	2 17	8 55E-06	5 21E-06	1.90E-05	1 88E-05
NVEnv	19 19	5 46E-04	6.68E-06	6 93E-04	6.42E-04
N990	21 11	4 53E-04	6.26E-06	5.68E-04	5.68E-04
N865	51 11	8 77E-04	1 25E-05	1.10E-03	1.12E-03
NV815	315 67	7 09E-03	1 11E-04	1.25E-02	1 26E-02
N815	426 12	7.84E-03	1 10E-04	1.30E-02	1.12E-02

environment generally considered to have less processing power. The decision of which environments to regard as "slow" or which environments to regard as "fast" was made from published Linpack results [Dongarra, 1987] together with assertions from the various vendors [CDC Petroleum Report, 1986, Cray Research, 1982].

From this information, the available and testing sets were defined as presented in TABLE VIII. These numerical criteria were used only to determine which environments should be grouped into which sets for each scenario. These published timings were used only for this partitioning and not for the analysis portion of this thesis.

Although there are several ways to partition the environments in the available and testing sets[4], the two situations defined provide a basis for the most realistic application of this work.

Defining the Test Models

A single test of the Predictability Model consisted of one benchmark program, one application program, one available set, and one environment from the corresponding testing set.

---

[4]Perhaps the most comprehensive test of the Predictability Model would be to evaluate all combinations of benchmarks, application programs, and environments. However, the simplest model consisting of three environments in the available set and one application/benchmark pair results in 720 different predictability models (10 choices for the first environment, 9 for the second, and 8 for the third) With each model predicting the run time of an application program in seven different environments, the amount of data to be analyzed quickly becomes unmanageable. Thus, this thesis considers only two different partitionings.

TABLE VIII

DEFINING THE AVAILABLE AND TESTING SETS

Model Name	Available Set	Testing Set
3Fastobs	C77v CFTv C77nv	N815 NV815 N865 N990 NVEnv NVEv CFTnv
5Fastobs	C77v CFTv C77nv CFTnv NVEv	N815 NV815 N865 N990 NVEnv
9Fastobs	C77v CFTv C77nv CFTnv NVEv NVEnv N990 N865 NV815	N815
3Slowobs	N815 NV815 N865	C77v CFTv C77nv CFTnv NVEv NVEnv N990
5Slowobs	N815 NV815 N865 N990 NVEnv	C77v CFTv C77nv CFTnv NVEv
9Slowobs	N815 NV815 N865 N990 NVEnv NVEv CFTnv C77nv CFTv	C77v

The degree of error for a single test was computed as the percentage difference between the predicted run time of the application program and the actual timing reported. As an example, TABLE IX illustrates how well the Linpack benchmark predicted the Black Box application CPU times using the 3Slowobs model.

TABLE IX

PREDICTABILITY RESULTS OF THE LINPACK/BLACK BOX 3SLOWOBS PREDICTABILITY MODEL

Environment	Actual Time	Predicted Time	%Difference
C77v	0.722	0.362	49.9%
CFTv	1.310	0.806	38.4%
C77nv	1.704	1.493	12.4%
CFTnv	3.738	3.704	0.9%
NVEv	1.703	1.606	5.7%
NVEnv	18.740	17.788	5.1%
N990	19.410	19.604	1.0%

The predicted times presented in TABLE IX were derived as follows. We know the timing A of the application program, Black Box, and the timing, B, of the benchmark program, Linpack, in the Available Set. We also know the timing B of Linpack in all of the environments in the 3Slowobs testing set. These timings are presented in TABLE X.

To determine the predicted time, the available set system of equations shown in TABLE XI is solved for β_1. In this example $\beta_1 = 0.95$. The predicted CPU time of the Black Box application in each environment of the testing set is

TABLE X

DEFINING THE LINPACK/BLACK BOX
3SLOWOBS PREDICTABILITY MODEL

3Slowobs Model	Application Black Box Times (A)	Benchmark Linpack Times (B)
Available Set		
N815	407.12	426.12
NV815	296.08	315.67
N865	44.26	51.11
Testing Set		
N990	???	21.11
NVEnv	???	19.19
NVEv	???	2.17
CFTnv	???	4.38
C77nv	???	2.05
CFTv	???	1.33
C77v	???	0.86

the solution of the following equation:

$$A = \beta_0 + \beta_1 * B$$

where $\beta_0 = 0$, $\beta_1 = 0.95$, and B = the CPU time of the Linpack benchmark for a given testing set environment. These equations are presented in TABLE XII.

TABLE XI

LINPACK/BLACK BOX 3SLOWOBS
PREDICTABILITY EQUATION

Available Set	Predictability Equation
	$A = \beta_0 + \beta_1 * B$
N815	$407.12 = \beta_0 + \beta_1 * 426.12$
NV815	$296.08 = \beta_0 + \beta_1 * 315.67$
N865	$44.26 = \beta_0 + \beta_1 * 51.11$

TABLE XII

DETERMINING THE PREDICTED RESULTS OF THE
LINPACK/BLACK BOX 3SLOWOBS
PREDICTABILITY MODEL

Testing Set	Predictability Equation
	$A = \beta_0 + \beta_1 * B$
C77v	$0.816 = 0 + 0.95 * 0.86$
CFTv	$1.261 = 0 + 0.95 * 1.33$
C77nv	$1.944 = 0 + 0.95 * 2.05$
CFTnv	$4.154 = 0 + 0.95 * 4.38$
NVEv	$2.058 = 0 + 0.95 * 2.17$
NVEnv	$18.201 = 0 + 0.95 * 19.19$
N990	$20.022 = 0 + 0.95 * 21.11$

The percentage difference, or degree of predictability, presented in TABLE IX, is simply the difference of the actual minus the predicted time divided by the actual time (i.e., (a-p)/a where a is the actual time and p is the time predicted by the model).

Defining the Evaluation Procedure

The procedures used in this work for evaluating the results obtained from the models fall into four categories which are summarized in TABLE XIII.

One of the suggestions made in Chapter II, to improve the performance of the Predictability Model, was to ensure that the processing performed by the benchmark matched that of the application program. In keeping with this suggestion, all of the programs use the same language, FORTRAN, and perform some amount of vector processing. However, with the exception of the Subtest program, no attempt was made to match the processing within a benchmark to that of an application program.

Another suggestion from Chapter II to improve the predicted times was to ensure similar environments were used in the models. In Chapter IV this point is confirmed. Also in Chapter IV, a review of the results by environments is made to examine the effect the environments had on the predicted results.

Chapter II also indicated that the predictability

TABLE XIII

SUMMARY OF EVALUATION PROCEDURES

Review by Application/Benchmark
Review by Environments
Review by Number of Environments in Available Set
Review by Predictability Equation

equation could affect the predicted timings. To determine how much of an affect the equation played, a second predictability equation is defined in Chapter IV and the results are evaluated.

Finally, the success of the Predictability Model may be dependent upon the number of environments in the available set. A review of the models by the number of environments in the available set is presented in Chapter IV to determine how many environments need to be in the available set to ensure a consistent predictability error over the entire testing set.

CHAPTER IV

EVALUATING THE RESULTS

Overview

If the thesis presented in this work is correct, then the predicted CPU times generated by the predictability equation for the various models would be similar and within an acceptable range of the actual CPU time. We consider an acceptable range to be within 30% of the actual time[1]. Unfortunately this was not the case as illustrated in Figure 3 which graphs the average performance of all 1,300 predictability models evaluated by benchmark[2].

If all the individual predictability models had results similar to the averages presented in Figure 3, a case could be made that the degree of error is so great that the model is of no value. As illustrated in Figure 4 however, a large number of individual models had predictability errors within the 30% acceptability range.

---

[1]The range of acceptability is of course subjective and very much dependent upon the needs of the person performing the evaluation. However, it seems reasonable to expect that tuning an application for a particular environment could change its execution time by 30%. Thus, it is beyond the scope of this work to expect benchmark accuracies smaller than 30%

[2]Actually Figure 4 1 graphs the averages of averages. The predictability error, or degree of predictability, is defined as the ratio of the actual application time to the predicted time (i.e., (a - p) / a, where a is the actual CPU time of the application program and p is the predicted time obtained from the model) Each bar of this graph represents the average predictability error of a particular benchmark over the entire suite of predictability models tested.

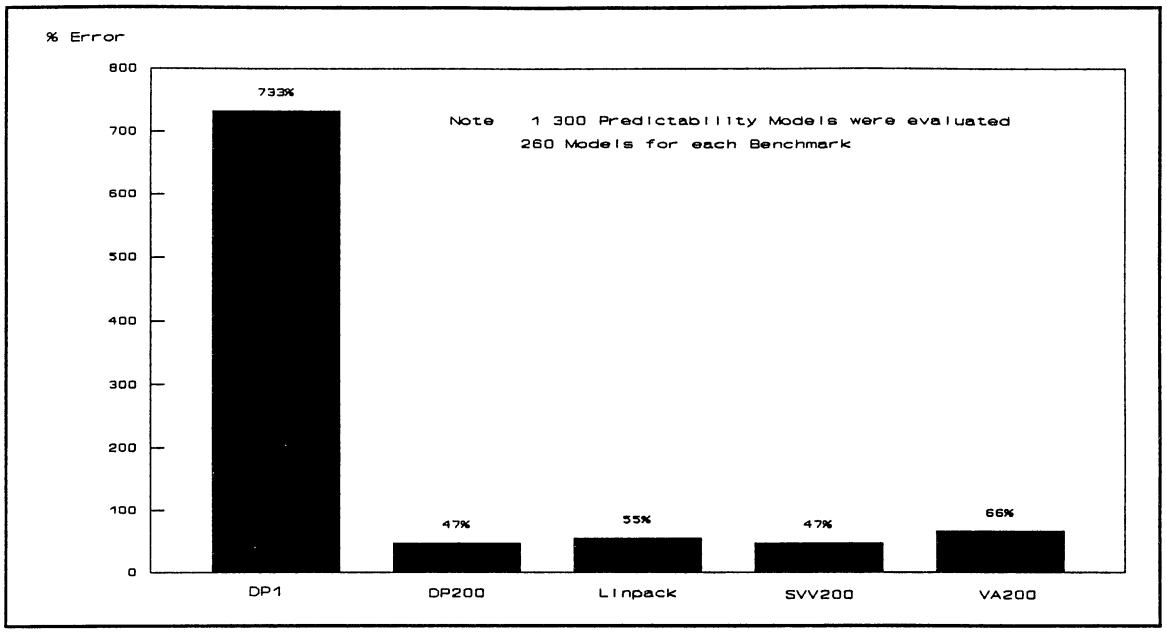


Figure 3. Average Predictability Error of All Models Considered in this Study by Benchmark

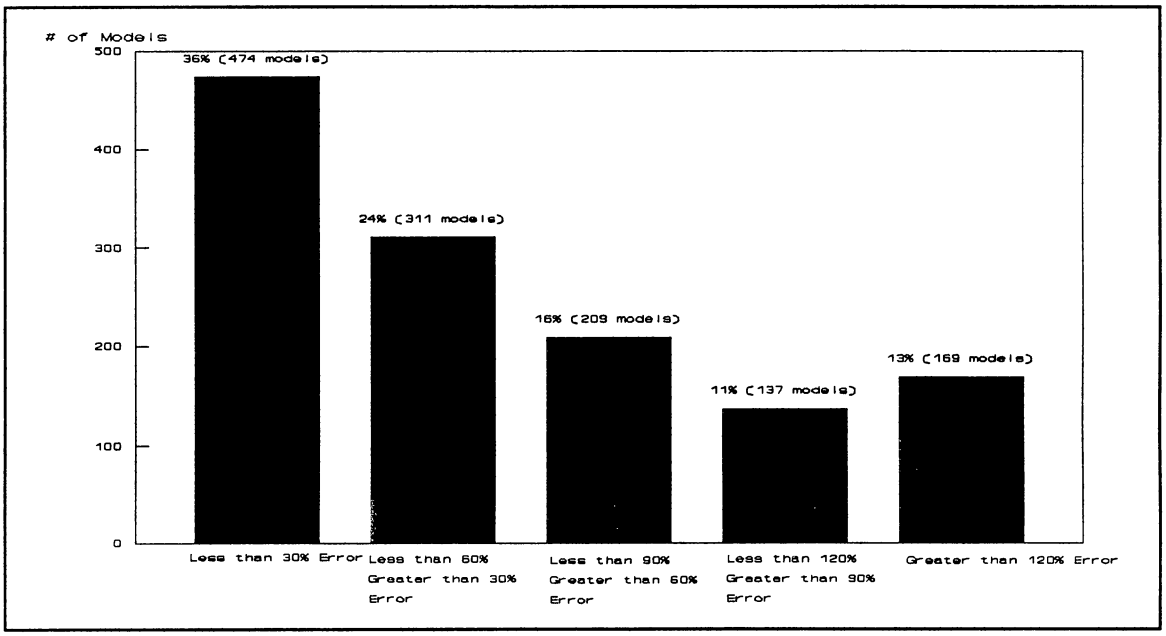


Figure 4. Thirty Percent Interval Breakdown of Individual Predictability Errors

The fact that the Predictability Model did display varying degrees of error for the specific cases evaluated indicates that it can be of value if the factors influencing the error can be identified and controlled. The remainder of this chapter will examine various aspects of the Predictability Model in an effort to identify these factors and discuss methods that can be used to improve the predicted times.

The Benchmark's Influence on The Predictability Model

One aspect of the predictability model that appears to have had a large impact on the degree of error is the relationship that existed between the application being predicted and the benchmark used. An extreme example of this is the differing success the VA200 and DP1 benchmarks had in predicting the time of the SMMTRIN application in the C77v environment; the VA200 benchmark was able to predict the performance of SMMTRIN within 16% of SMMTRIN's actual time CPU time while the DP1 benchmark had a predictability error of over 6,000%. The reason for such a large difference in predictability results is simple to explain: the estimated time generated by the VA200 model was closer to the actual timing of the SMMTRIN application than the time produced by the DP1 model. This is illustrated in Figures 5 and 6 which plot the regression line generated

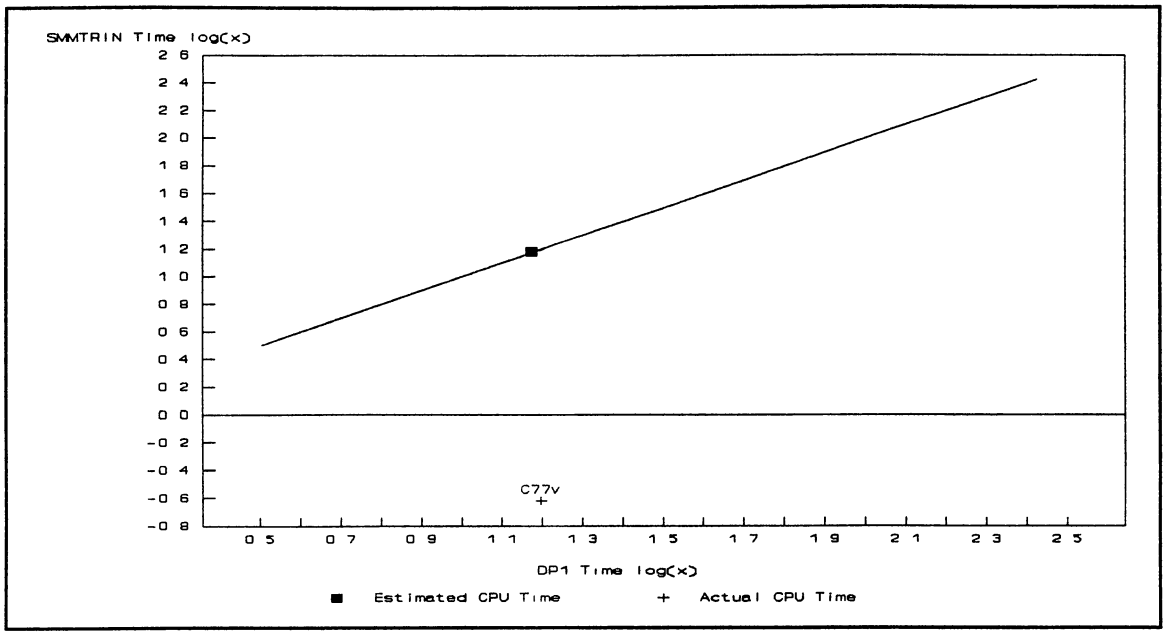


Figure 5. The Predicted CPU Time of SMMTRIN in the C77v Environment using the DP1 Benchmark, 5Slowobs Model

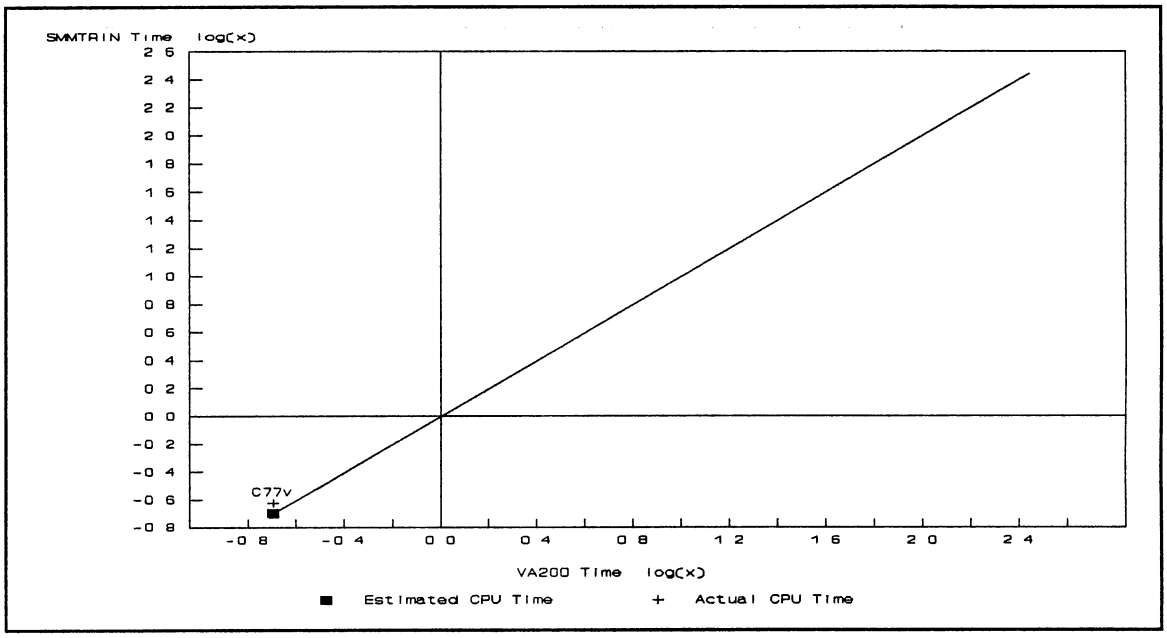


Figure 6. The Predicted CPU Time of SMMTRIN in the C77v Environment using the VA200 Benchmark, 5Slowobs Model

by these two models along with the actual time of the SMMTRIN application.

A more difficult question to answer is how does one determine which benchmark provides the best estimate? One approach might be to use the application/benchmark combination with the smallest RMS [Freedman, 1978] error in the available set. One could assume that the benchmark/application combination with the smallest deviation from the best fit straight line in the available set would also be the one that would provide the best estimate in the tested environment. In the example presented above however, all indications were that DP1 would provide a better estimate than VA200. Figures 7 and 8 illustrate the regression lines generated by the DP1 and VA200 benchmarks within the available set. These figures, coupled with DP1's RMS error of 4.4 and VA200's error of 14 would seem to indicate that DP1 would provide the best estimate of SMMTRIN in the testing set. We have already seen however that this was not the case.

Another method of selecting the benchmark may be to compare the code of the benchmark to that of the application. Since SMMTRIN and VA200 both do simple vector loads, it would seem logical that VA200 would be the better predictor of SMMTRIN. Although this approach works, using this method violates the spirit of the thesis since the processing performed by the benchmark and application have

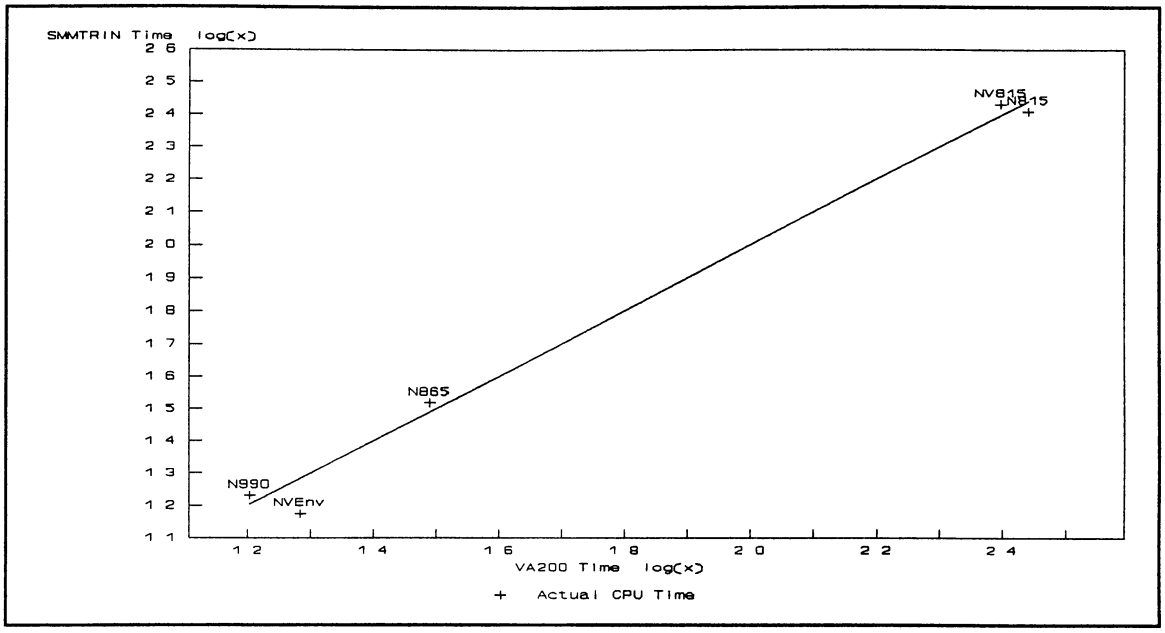


Figure 7. Regression Line of VA200 Benchmark, 5Slowobs Predictability Model Available Set

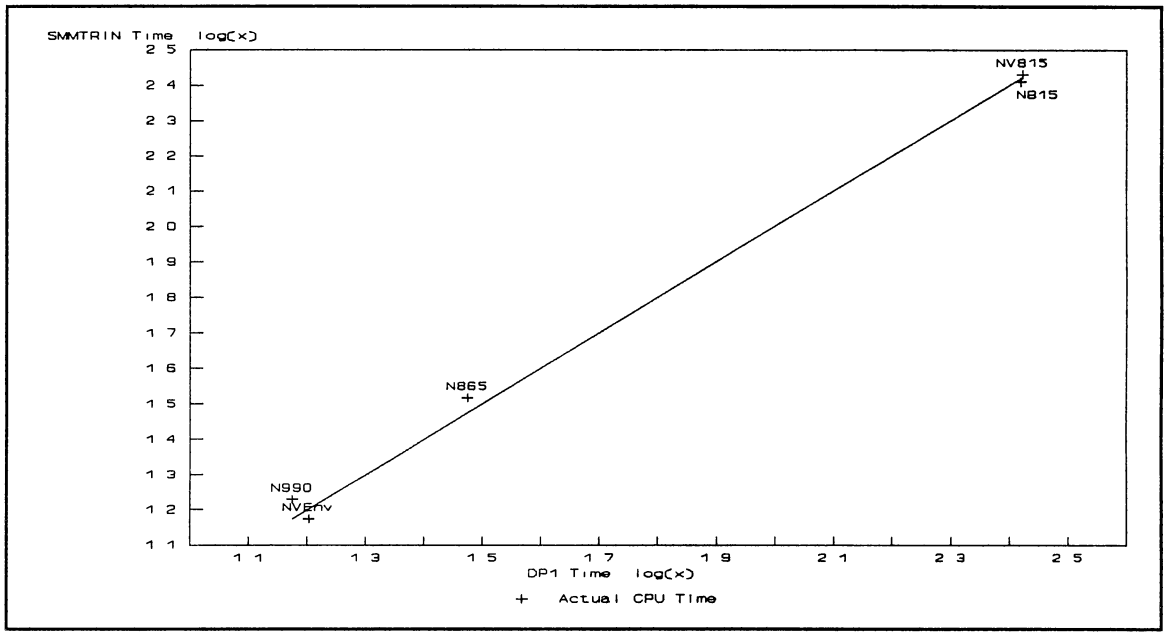


Figure 8. Regression Line of the DP1 Benchmark, 5Slowobs Predictability Model Available Set

to be known.

An approach that works well and is more in keeping with the thesis assertion is to run several models with different benchmarks and compare the predicted times. As an example, Figure 9 details the predictability results of all the benchmarks for the 5Slowobs model, C77 environment, together with application program SMMTRIN's actual CPU time. It is evident from Figure 9 that the DP1 benchmark would be considered an anomaly (or outlier) since it differs so wildly from the others and should not be considered as one of the choices. An argument could also be made to throw out the time predicted by Linpack since it is over twice the time predicted by the other models. Of the three that are left, only SVV200 fails to meet the 30% acceptability criteria with a 34% degree of error.

It is difficult to determine if this approach of always selecting the median predicted time works for all models, but a random selection of environments and applications from the 5Slowobs and 5Fastobs testing sets is presented in Figures 10 through 13 that support the use of this method.

TABLE XIV lists the degree of error for the predicted times presented in Figures 10 through 13 with the shadowed numbers being the median time to select using this method. The method was able to select a predicted time that was well within the 30% acceptability range for two of the examples; Figures 10 and 11. The method failed for Figure 12 with the

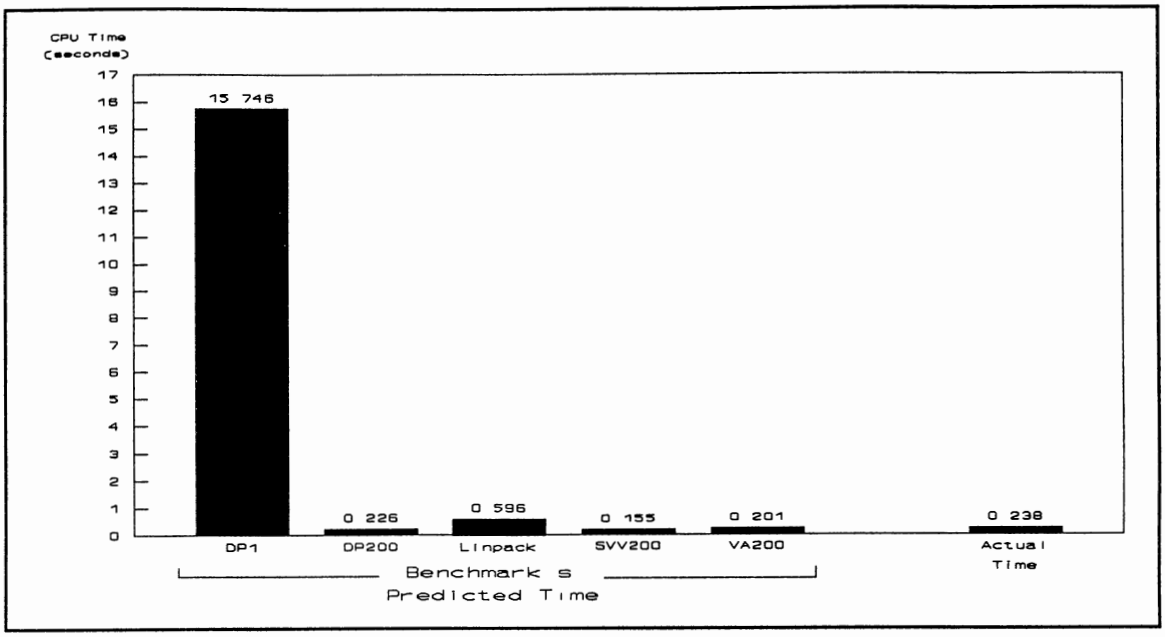


Figure 9. Predicted CPU Times of SMMTRIN in the C77v Environment using the 5Slowobs Model

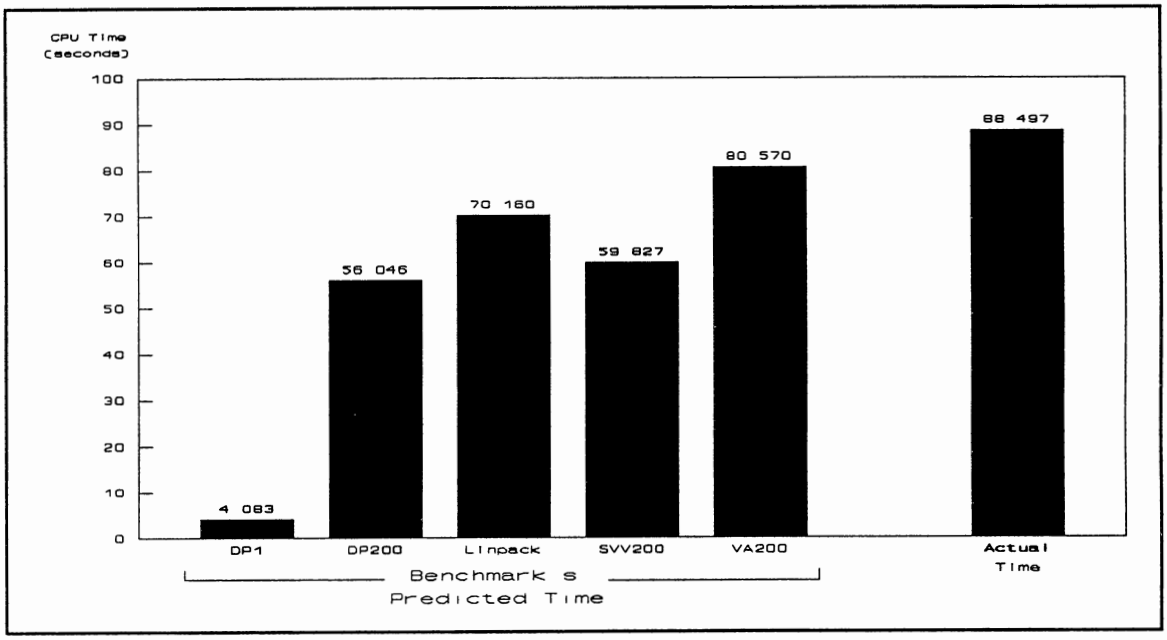


Figure 10. Predicted CPU Times of SMMSTK in the N865 Environment using the 5Fastobs Model

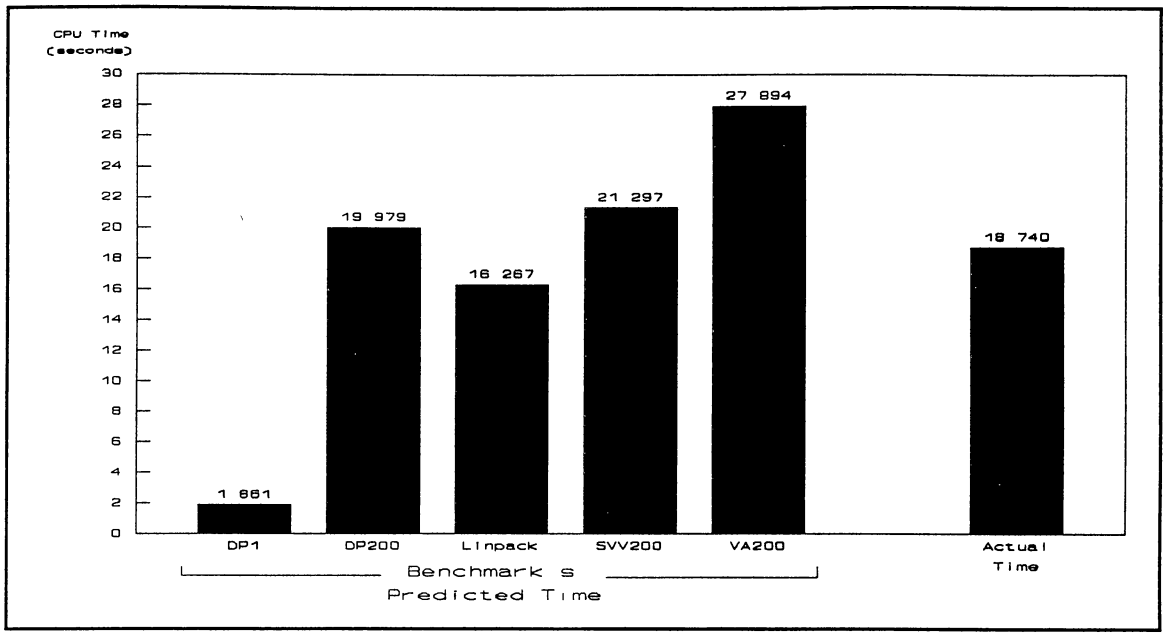


Figure 11 Predicted CPU Times of Black Box in the NVEnv Environment using the 5Fastobs Model

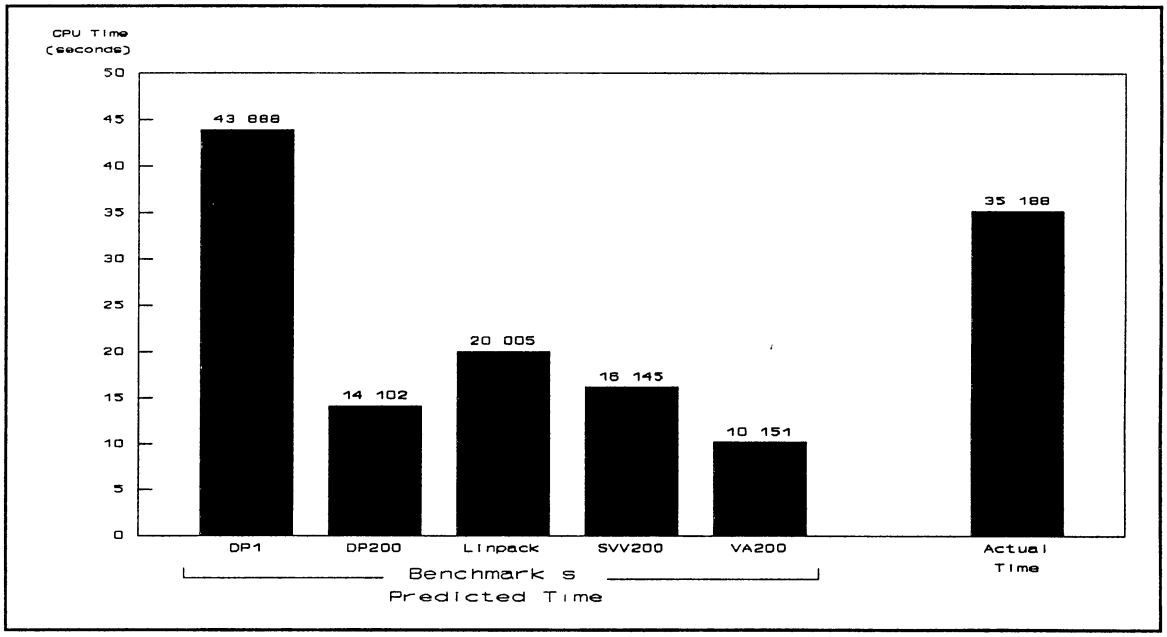


Figure 12. Predicted CPU Times of SMMNMO in the C77nv Environment using the 5Slowobs Model

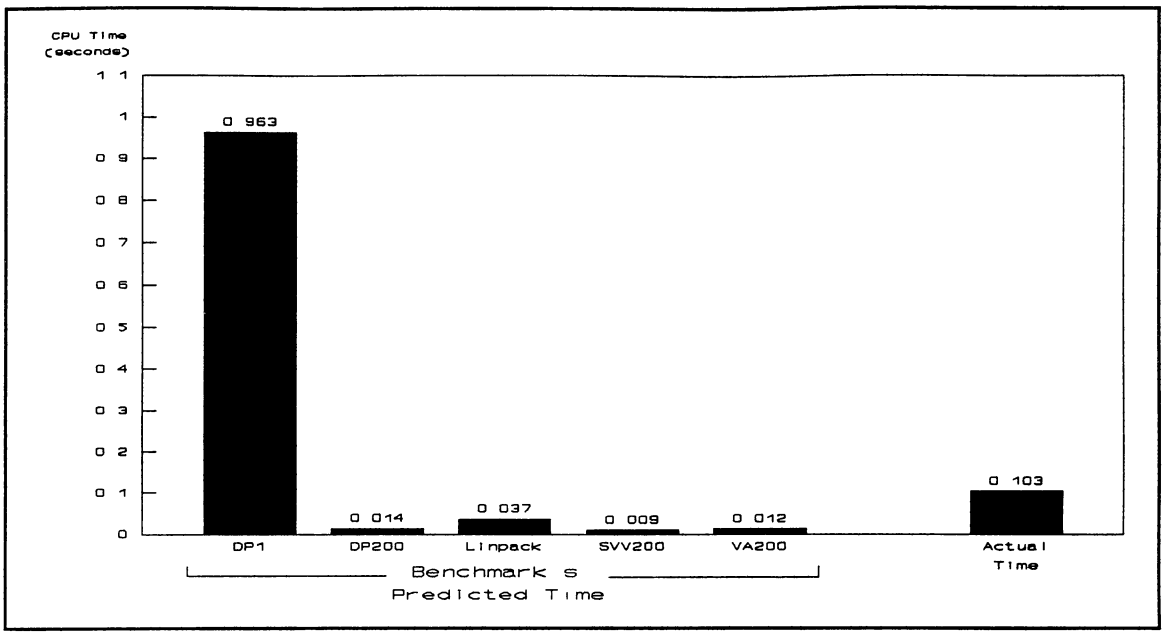


Figure 13. Predicted CPU Times of Subtest in the C77v Environment using the 5Slowobs Model

TABLE XIV

PREDICTABILITY ERRORS OF MODELS PRESENTED IN FIGURES 10 THROUGH 14

	Fig 4.8	Fig 4.9	Fig 4.10	Fig 4.11
DP1	95%	90%	-25%	-835%
DP200	37%	-7%	60%	87%
Linpack	21%	13%	43%	64%
SVV200	32%	-14%	54%	91%
VA200	9%	-49%	71%	88%

anomalous case, dp1, actually being the only estimate with a degree of predictability that was within our 30% error acceptability criteria. For Figure 13, none of the benchmarks were able to predict the application performance to within our desired accuracy. Although these tests do not necessarily indicate that this method will always provide the best estimate, it does guarantee that the time generated by the benchmark selected will not be the worst.

The relationship that exists between the benchmark and the application program is a primary factor influencing the accuracy of the Predictability Model. Unfortunately, no method was found for determining which benchmark provided the best prediction in every situation. Selecting the median time produced from several models using different benchmarks does provide a reasonable guess. However, using this method does not guarantee that the timing selected will be within an acceptable range of the actual CPU time of the application.

The Environment's Influence on The Predictability Model

Figures 14 through 18 demonstrate how the five benchmark/application combinations discussed above and presented in TABLE XV performed in the 5Slowobs and 5Fastobs predictability models. One result these figures illustrate is the poor performance of the model at predicting

TABLE XV

A SAMPLE SELECTION OF FIVE BENCHMARK/APPLICATION COMBINATIONS USED TO TEST THE PREDICTABILITY MODEL

VA200/SMMTRIN
Linpack/SMMSTK
SVV200/Black Box
SVV200/SMMNMO
DP200/Subtest

application CPU times in the vector environments C77v, CFTv, and NVEv. Indeed this tendency to have poor predictions for the vector environments was consistent regardless of which benchmark/application combination was used as seen in Figure 19, which plots the average predictability error of all the five observation models excluding the DP1 models[3].

To examine the influence the vector environments had on the models further, the vector environments were removed from the 5Fastobs available set and NVEnv was included. The five application/benchmark combinations presented in TABLE XVI were then examined again for the N990, N865, NV815 and N815 environments. The results obtained are presented in

[3]It is evident from Figure 4 1 that the DP1 benchmark can be considered an anomaly regardless of which application or environment is being evaluated.

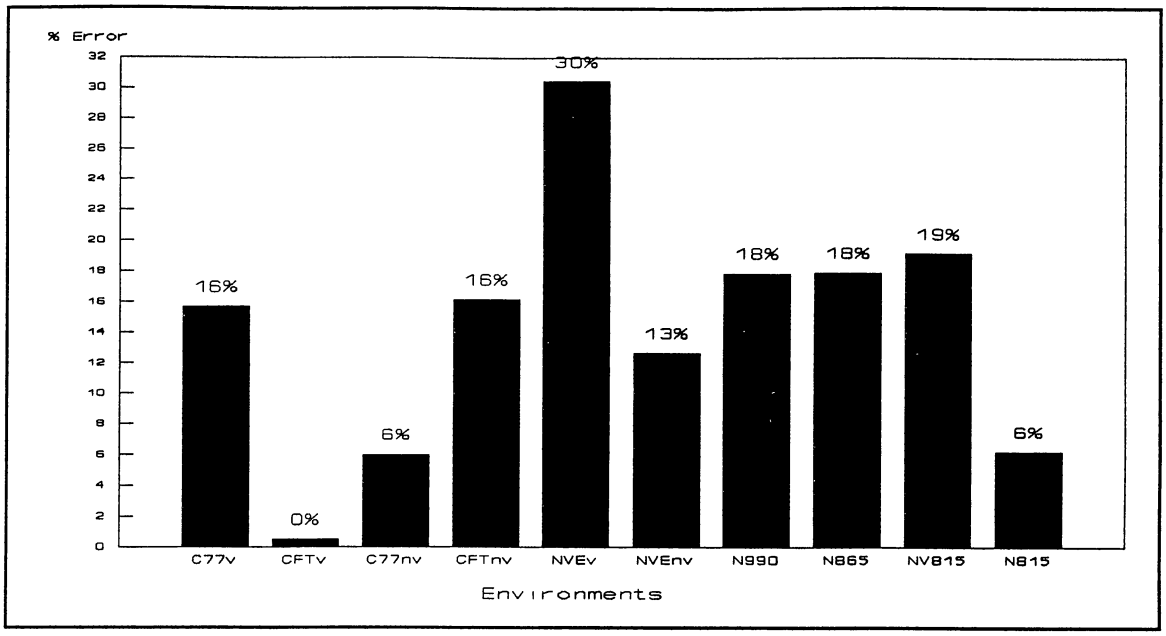


Figure 14. Predictability Error by Environment of the SMMTRIN / VA200 5Fastobs and 5Slowobs Models

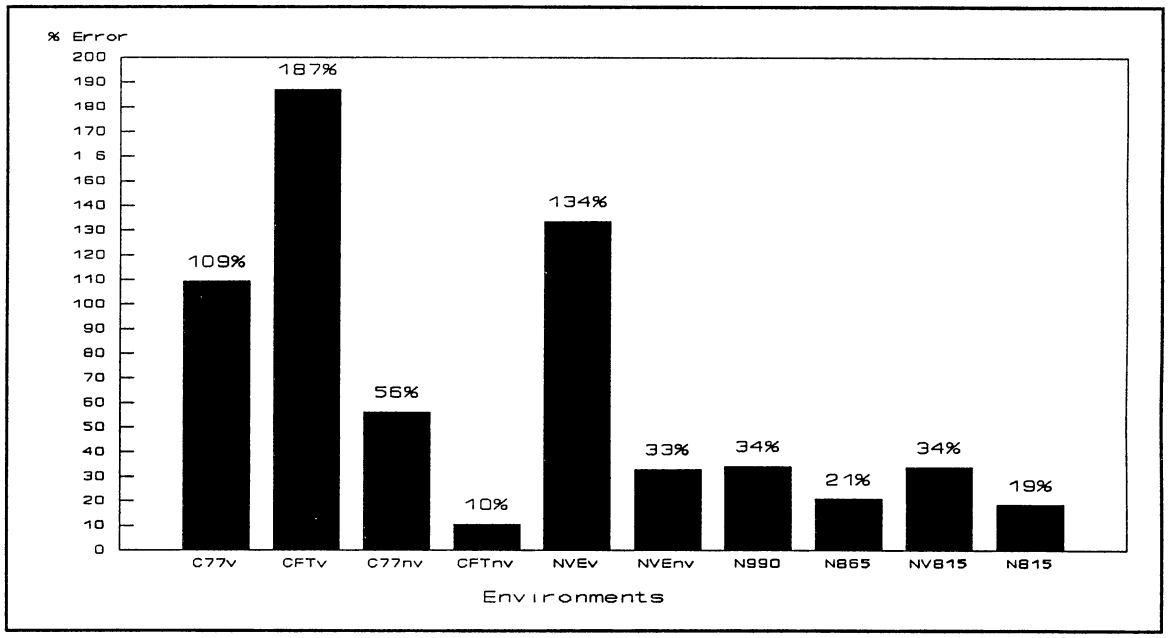


Figure 15. Predictability Error by Environment of the SMMSTK / Linpack 5Fastobs and 5Slowobs Models

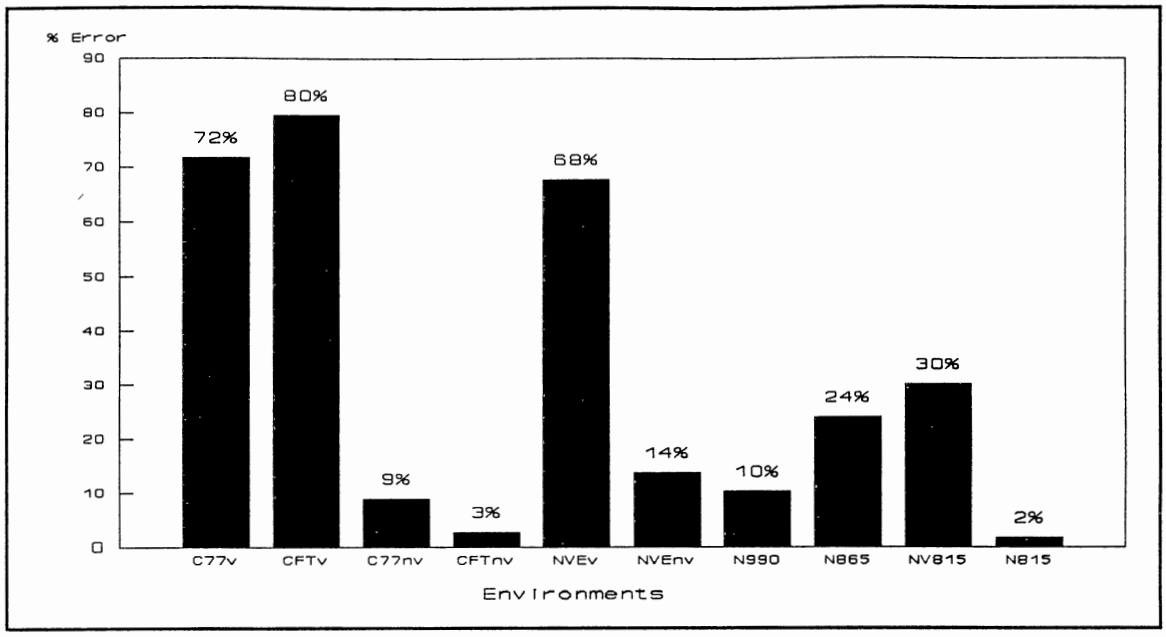


Figure 16. Predictability Error by Environment of the BlackBox / SVV200 5Fastobs and 5Slowobs Models

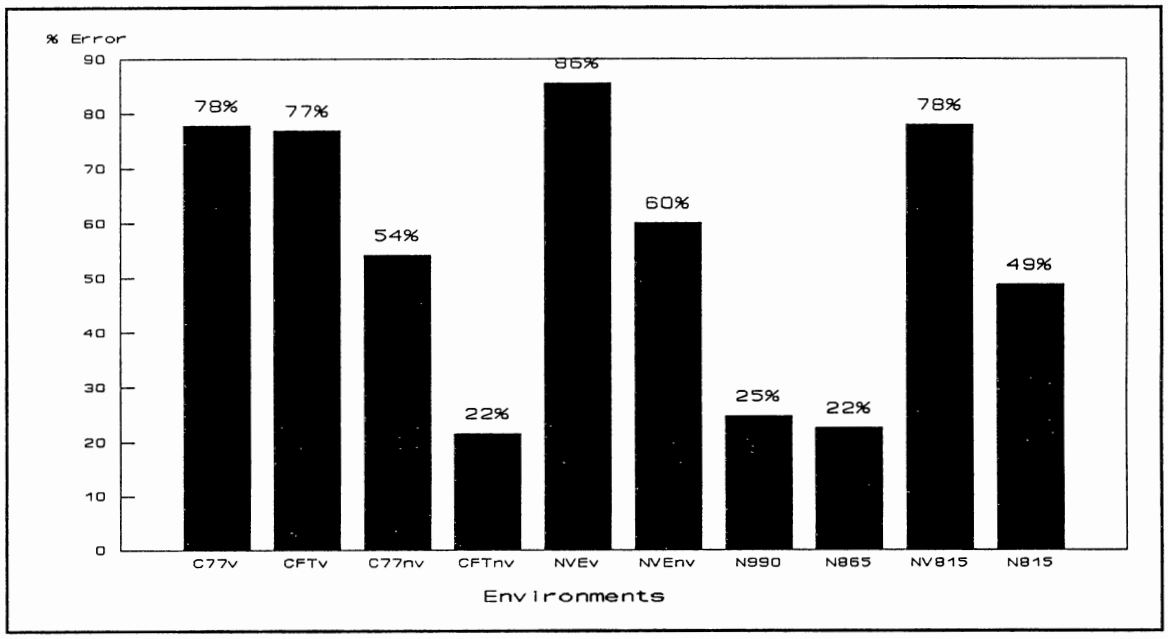


Figure 15. Predictability Error by Environment of the SMMNMO / SVV200 5Fastobs and 5Slowobs Models

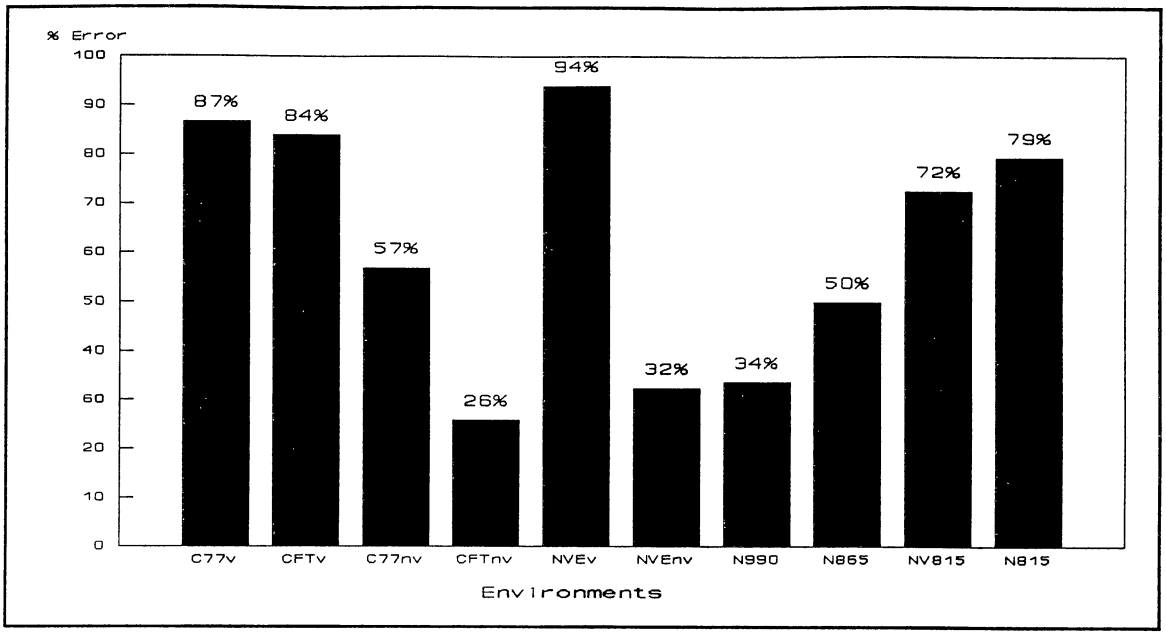


Figure 18. Predictability Error by Environment of the Subtest / DP200 5Fastobs and 5Slowobs Models

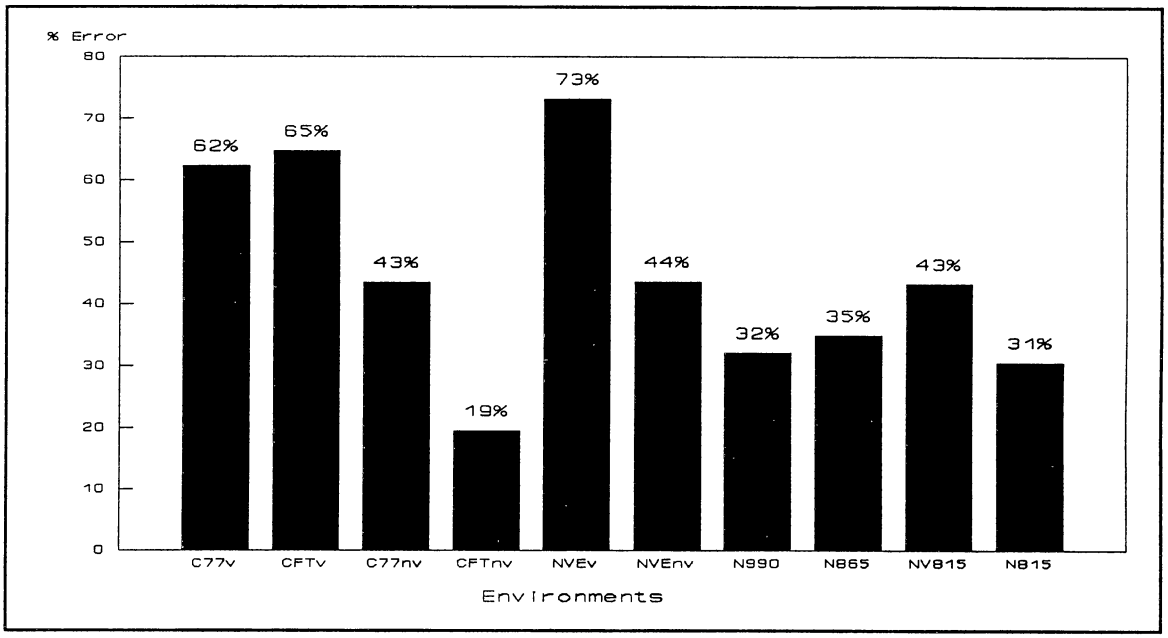


Figure 19. Average Predictability Error by Environment of all 5Fastobs and 5Slowobs Models

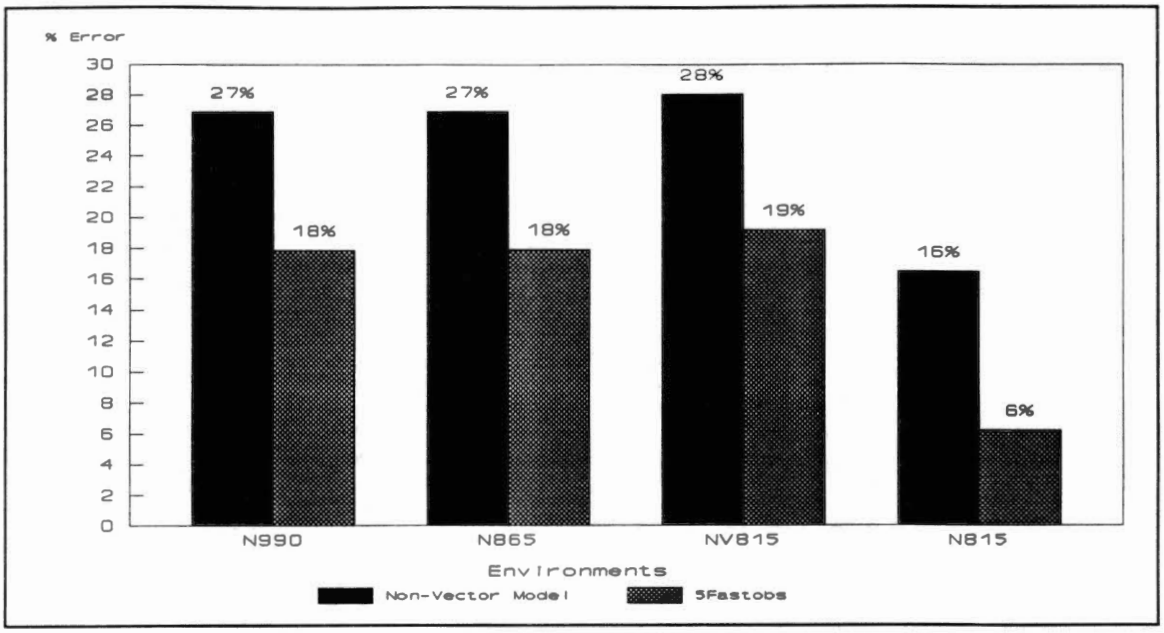


Figure 20. Predictability Errors of the VA200 / SMMTRIN Combination using only Non-Vector Environments

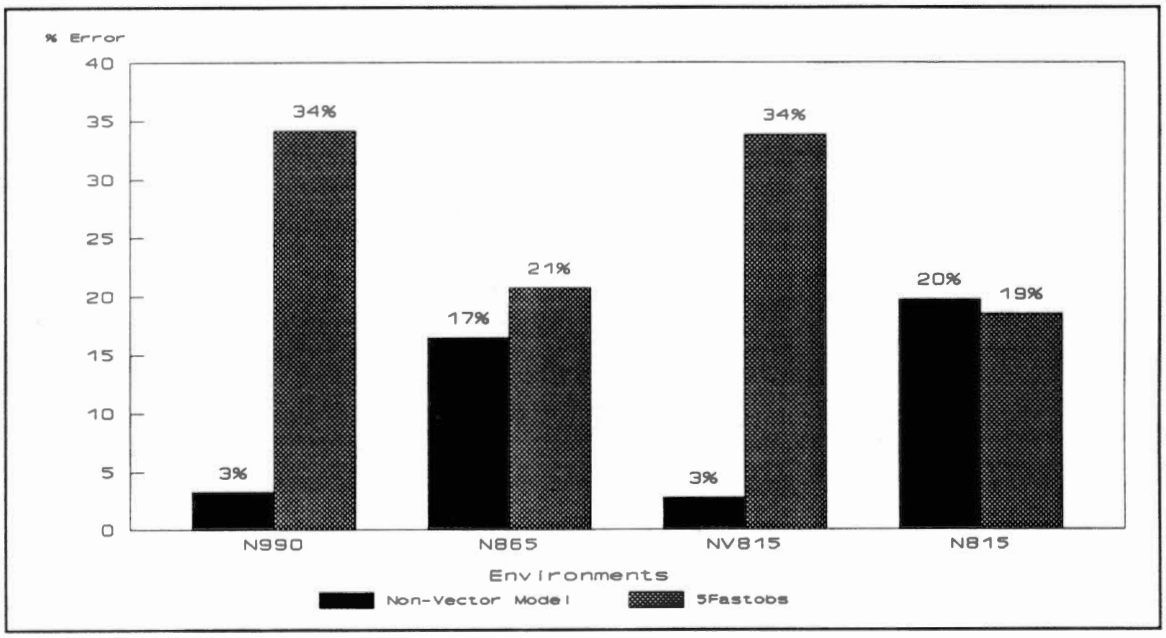


Figure 21. Predictability Errors of the SMMSTK / Linpack Combination using only Non-Vector Environments

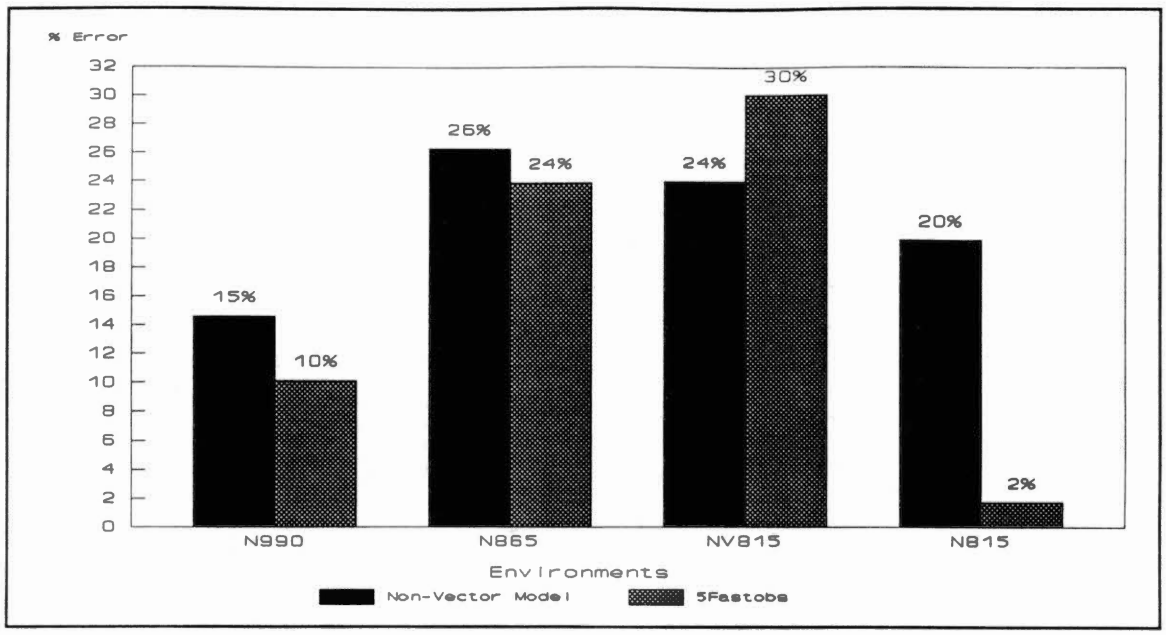


Figure 22. Predictability Errors of the BlackBox / SVV200 Combination using only Non-Vector Environments

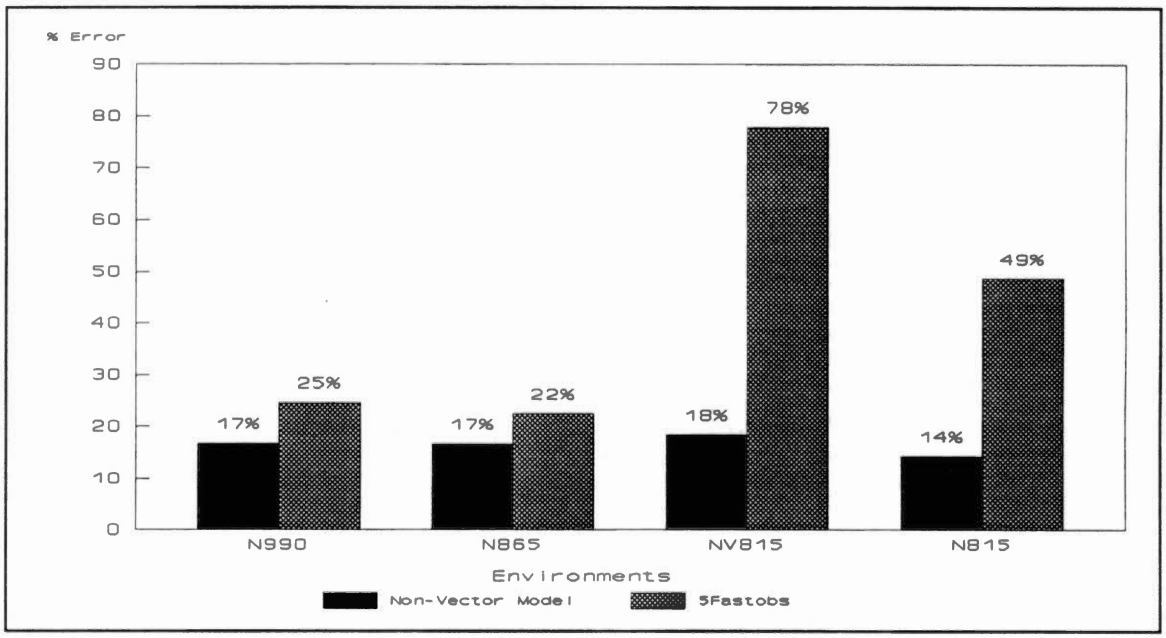


Figure 23. Predictability Errors of the SMMNMO / SVV200 Combination using only Non-Vector Environments

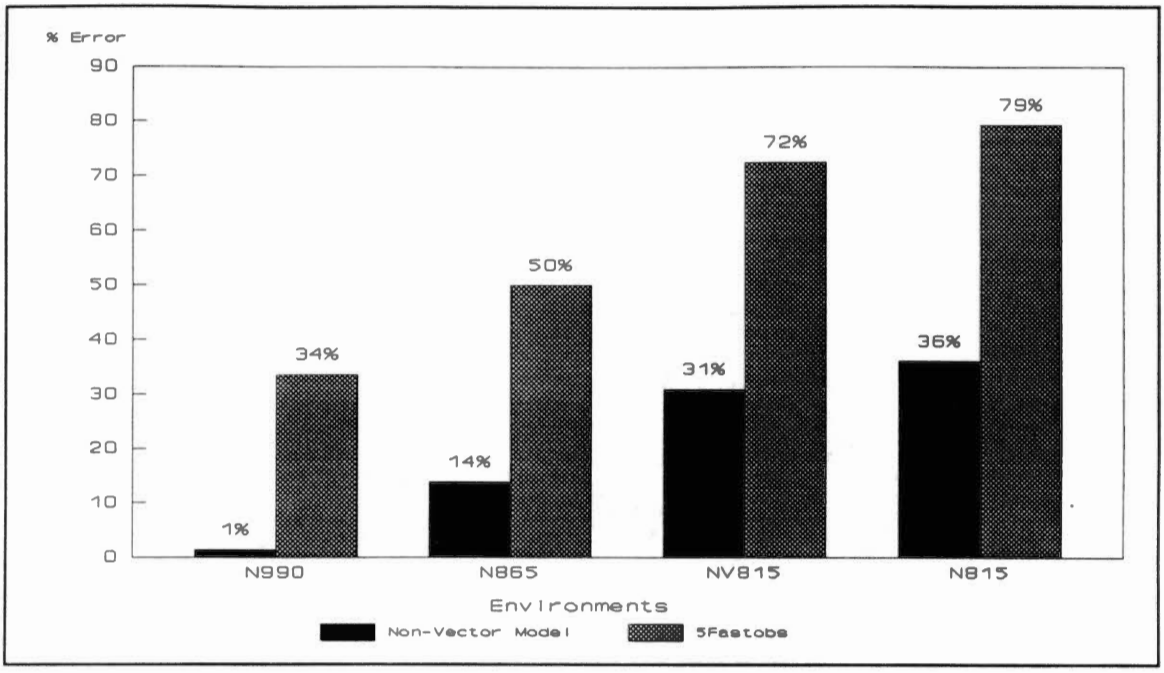


Figure 24. Predictability Errors of the Subtest / DP200 Combination using only Non-Vector Environments

Figures 20 - 24. Although a consistent improvement in the predictability error was not seen over all models, the predictability errors obtained were well within the defined acceptability range with wide deviations from the original 5Fastobs models, hence generally showing an improvement.

These tests should not be considered representative of the entire suite. However, they do indicate that the environments selected for the testing and available sets can influence a Predictability Model's accuracy in predicting error. In general, similar environments tend to have a better degree of predictability than dissimilar environments.

The Impact of the Available Set Size

Another factor influencing the predictability model was the size of the available set. Figure 25 illustrates this by plotting the predictability error of the five test cases presented in TABLE XV for the C77v and N865 environments as more environments were added to the available set.

Striking improvements in the timings generated by the Predictability Model were observed when using the fast environment models to predict the time of the slow environments. Increasing the number of environments in the available set from three to five improved the degree of predictability by 50% for four of the five test cases. Increasing the number of environments again from five to nine actually decreased the accuracy of the predicted time for three of the five test cases; however, with nine environments, all predicted times were within our acceptability range of 30%.

One would expect that using the slow environments to predict the fast environments would demonstrate a similar change in predictability errors. This however was not the case. In fact, the number of environments in the available set had virtually no affect on the predictability error of the slow environment models. The reason for this is that we force the intercept of the predictability equation to be zero ($\beta_0=0$) . When this constraint is removed from the predictability equation, the predicted times do improve as

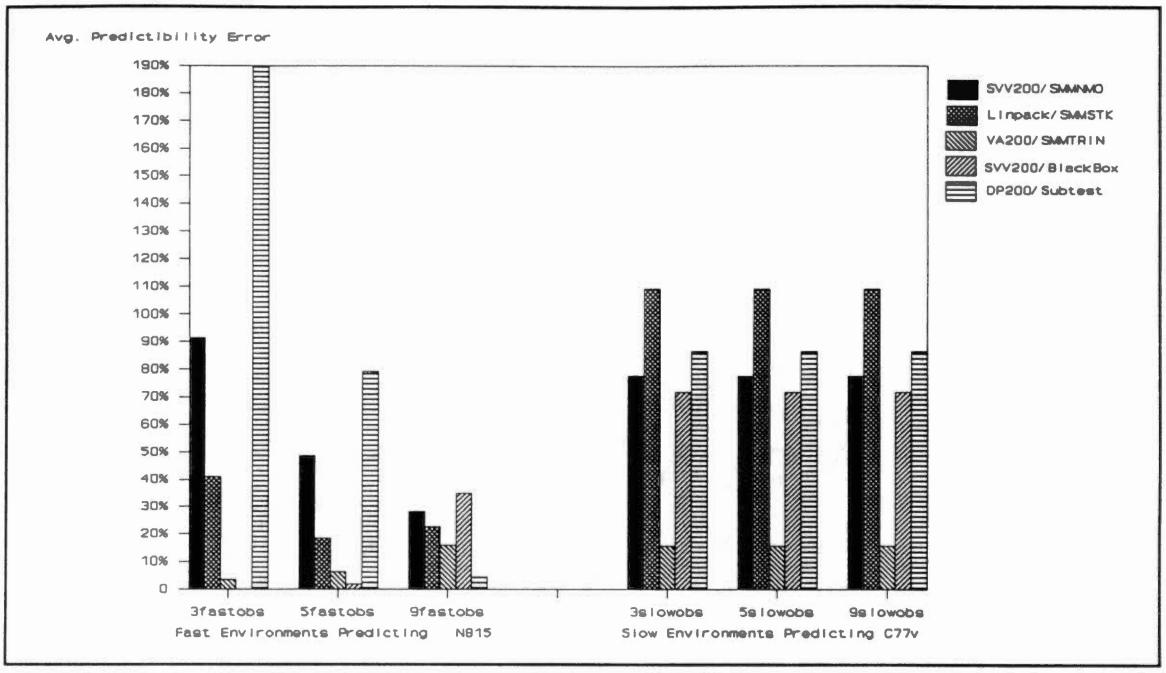


Figure 25. The Effect the Number of Environments in the Available Set had on C77v and N815

more environments are added to the available set. This is illustrated in Figure 26. It should be noted however that the degree of errors for all cases increases dramatically.

Forcing the intercept to be zero constrains the Predictability Model to predict times greater than zero. Consequently, as the CPU times of the faster environments tend to cluster close to zero, they play less of a role in determining the slope of the line generated by the model. Conversely, since the CPU times of the slow environments tend to differ widely, thus having a greater influence on the slope of the line, they exhibit a greater change in

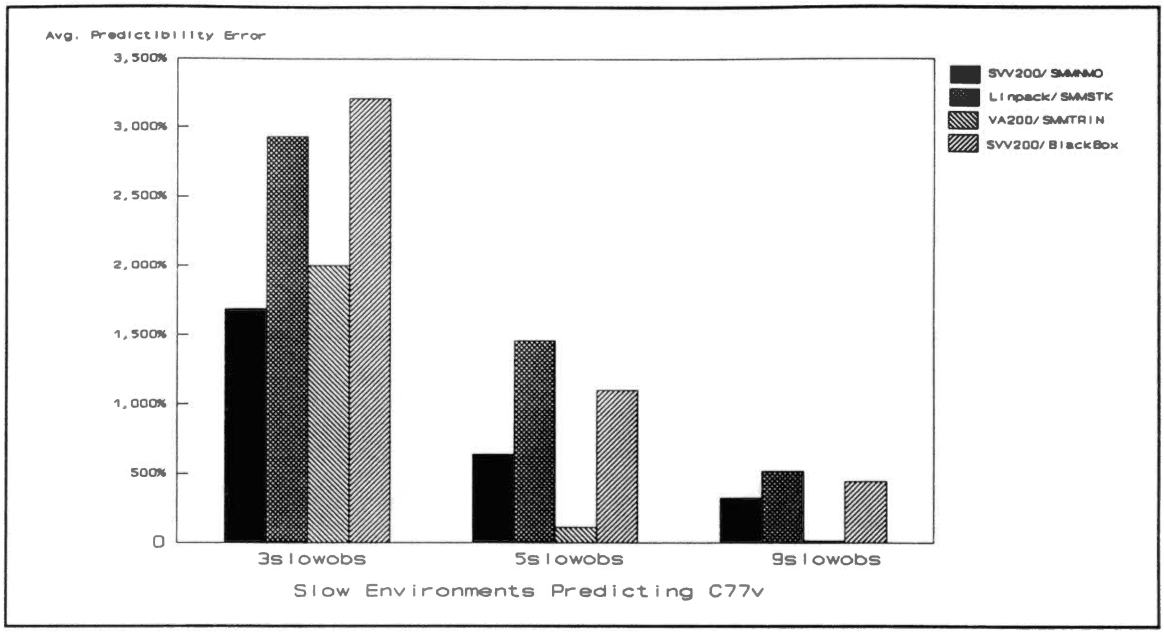


Figure 26. The Effect the Number of Environments in the Available Set had on C77v when the Intercept was Computed

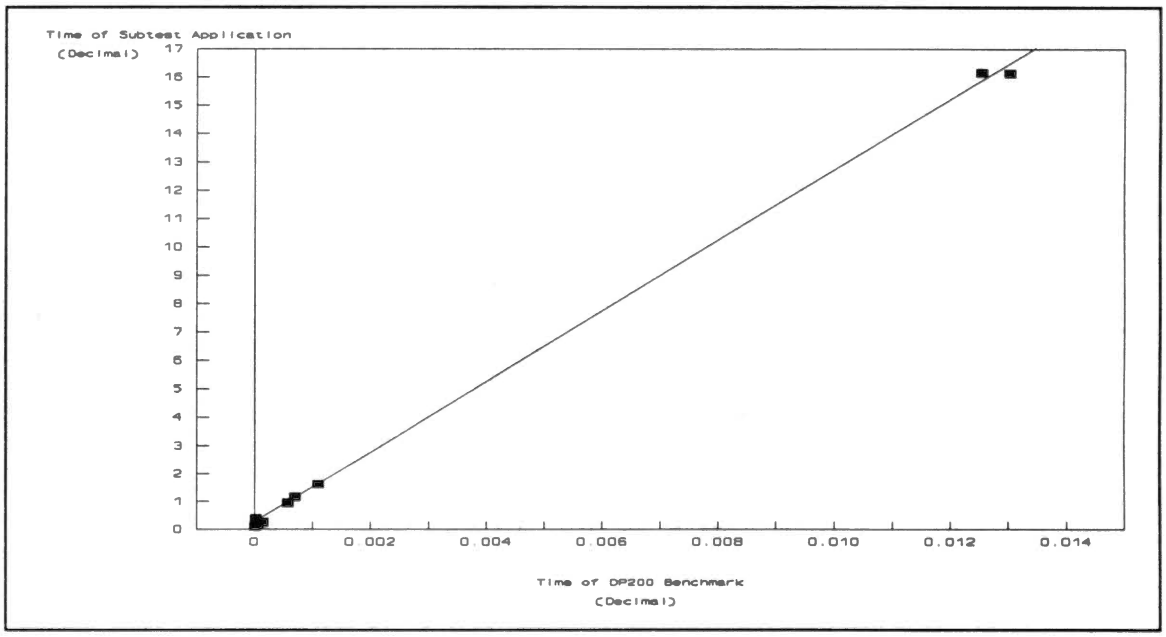


Figure 27. Scatter Diagram of the DP200 / Subtest Predictability Model with Sample Regression Line

predictability errors as more slower environments are added to the available set. This is illustrated in Figure 27 which plots the actual times of the DP200/Subtest model over all environments along with a sample regression line generated by the Model.

In summary, increasing the number of environments in the available set does play a role in the degree of error that is predicted. When using fast environments to predict a slow environment, nine observations in the available set provided an acceptable predictability error for all cases selected. When using a slow environment available set to predict a fast environment, three observations in the available set was sufficient for the cases examined.

Testing the Lindsay Benchmark Hypothesis

The basic theory behind the Lindsay benchmark is that all application programs are made up of atomic units such as adds, deletes, and assignments. Determining how well a computer executes these atomic units gives an indication of how well the computer will execute applications containing them [Lindsay, 1987].

This theory was incorporated in the Predictability Model in an attempt to improve its performance. Defining each benchmark to be an atomic unit, the Predictability Model was modified to allow three benchmark timings to have an influence on the predicted result of one application by

replacing the predictability equation with a multiple linear equation of the form:

$$A = \beta_1 * B_1 + \beta_2 * B_2 + \beta_3 * B_3$$

where B_1, B_2 and B_3 are individual benchmark timings. As before, the values of the β's are determined from pairs of A's and B_n's which have been determined from the available set. Knowing the A, B_n pairs, we use a multiple linear regression algorithm to determine the values of the β_n's. Inputs to the modified Predictability Model were the CPU times of the three Lindsay benchmarks SVV200, VA200, DP200; the CPU times of the application programs SMMNMO, SMMSTK, Black Box, and Subtest; and the environments defined in the 5Fastobs and 5Slowobs testing sets.

A sample Predictability Model is presented in TABLE XVI. The model consists of a 5Fastobs or 5Slowobs available set X; an environment Y, selected from the testing set; the CPU time for an application program, A; and the CPU times, B_1, B_2 and B_3, of the benchmarks, SVV200, VA200, and DP200. The set of equations for the available set is solved for β_1, β_2, and β_3, and these coefficients are in turn used in the predictability equation to solve for A in the testing set.

The results obtained using the modified Predictability Model are presented in Figures 28 and 29 along with the predictability error of the corresponding

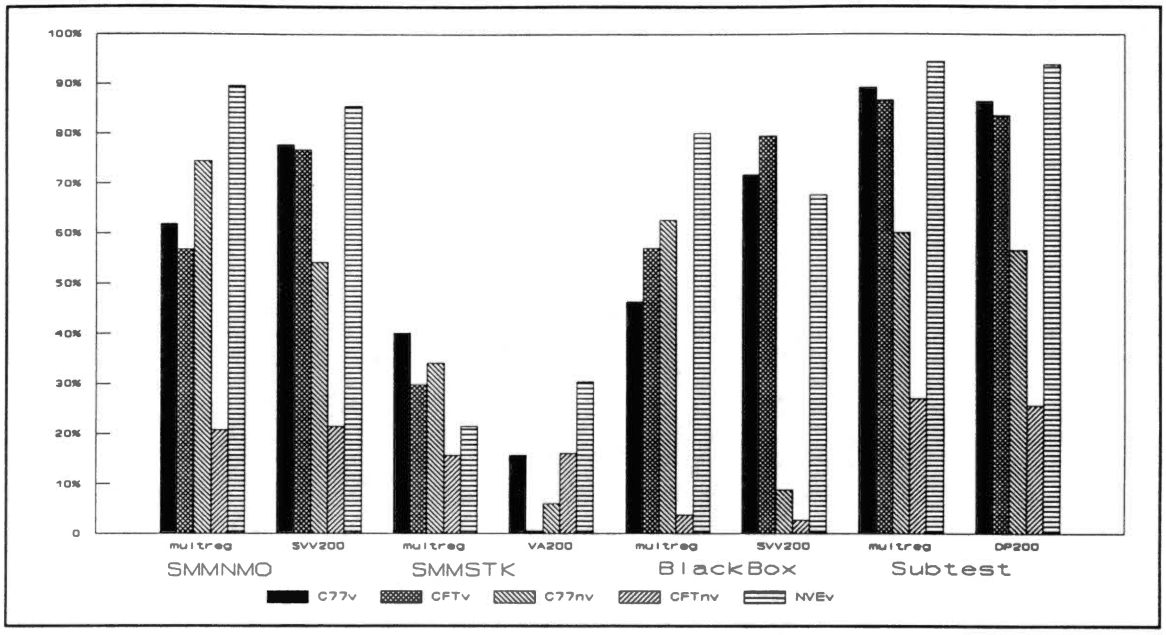


Figure 28. Comparing the Predictability Errors of the Multiple Regression Equation and the Standard Equation using the 5Slowobs Model to Predict the CPU Times of the Fast Environments

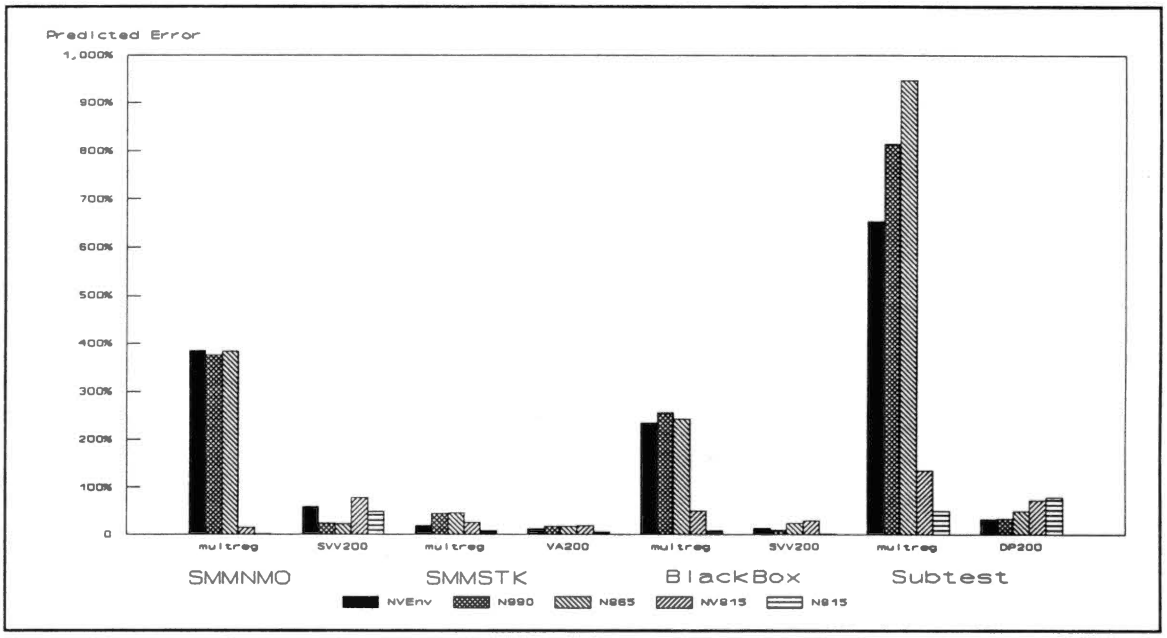


Figure 29. Comparing the Predictability Errors of the Multiple Regression Equation and the Standard Model using the 5Fastobs Model to Predict the CPU Times of the Slow Environments

TABLE XVI

EQUATIONS OF THE MODIFIED PREDICTABILITY MODEL

Environment	Application Times (A) and Benchmark times (B)	Comments
Available Set		
X_1	$A = \beta_1*B_1 + \beta_2*B_2 + \beta_3*B_3$	In the available set we know the values of A and B_n, we solve for β_n.
X_2	$A = \beta_1*B_1 + \beta_2*B_2 + \beta_3*B_3$	
X_3	$A = \beta_1*B_1 + \beta_2*B_2 + \beta_3*B_3$	
X_4	$A = \beta_1*B_1 + \beta_2*B_2 + \beta_3*B_3$	
X_5	$A = \beta_1*B_1 + \beta_2*B_2 + \beta_3*B_3$	
Testing Set		
Y	$A = \beta_1*B_1 + \beta_2*B_2 + \beta_3*B_3$	In the testing set we know the values of B_n and β_n, we solve for A

benchmark/application combinations presented in TABLE XV.

The modified model produced predictability errors similar to the standard model when using the 5Slowobs model to predict the time of the fast environments. This is probably due to the fact that we again forced the intercept to be zero. However, the modified predictability model did not perform as well as the standard model when predicting the CPU times of the slow environments. In fact, several of the predicted times were less than zero. The reason for this is probably due to the three dimensional aspect of the multiple regression algorithm used.

Although this more complex predictability equation does

not appear to improve the predicted time, other equations might be available that provide a smaller error. Care should be taken however in selecting an equation to use. Unless a predictability equation can be defined that provides a substantial improvement in the degree of error, the simplicity of the standard model would tend to encourage its use.

Summary

Figure 4 details the distribution of predictability errors for all models evaluated and introduces a discussion on the methods that could be used to improve this degree of predictability. In summary, four factors were introduced that were thought to have an influence on the predicted results. These factors were the benchmark used, the type of environments to be predicted, the number of environments in the available set, and the nature of the predictability equation. Of the four factors reviewed, the nature of the predictability equation was the only factor that did not provide an improvement in the predicted results.

Figure 30 revisits the data presented in Figure 4 incorporating two of the suggestions presented in this chapter to improve the predicted time. The anomalous benchmark DP1 and the predicted times for the vector environments were removed. As seen in Figure 30, an overall improvement in the predictability errors of the models

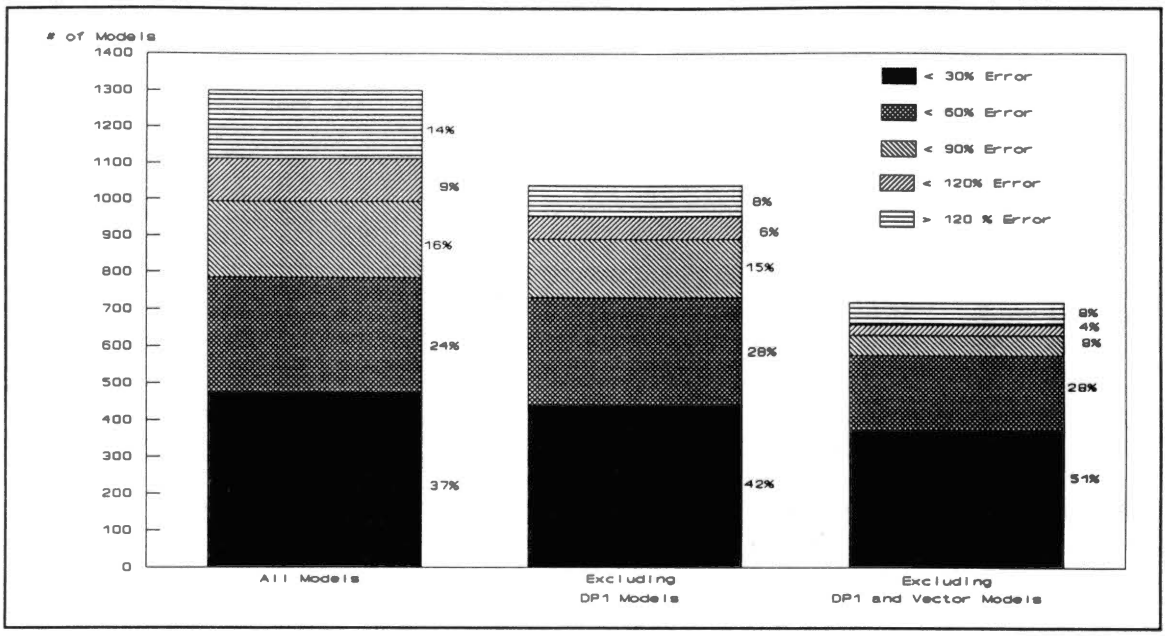


Figure 30. Distribution of Predictability Errors for all Models Evaluated and Selected Subsets

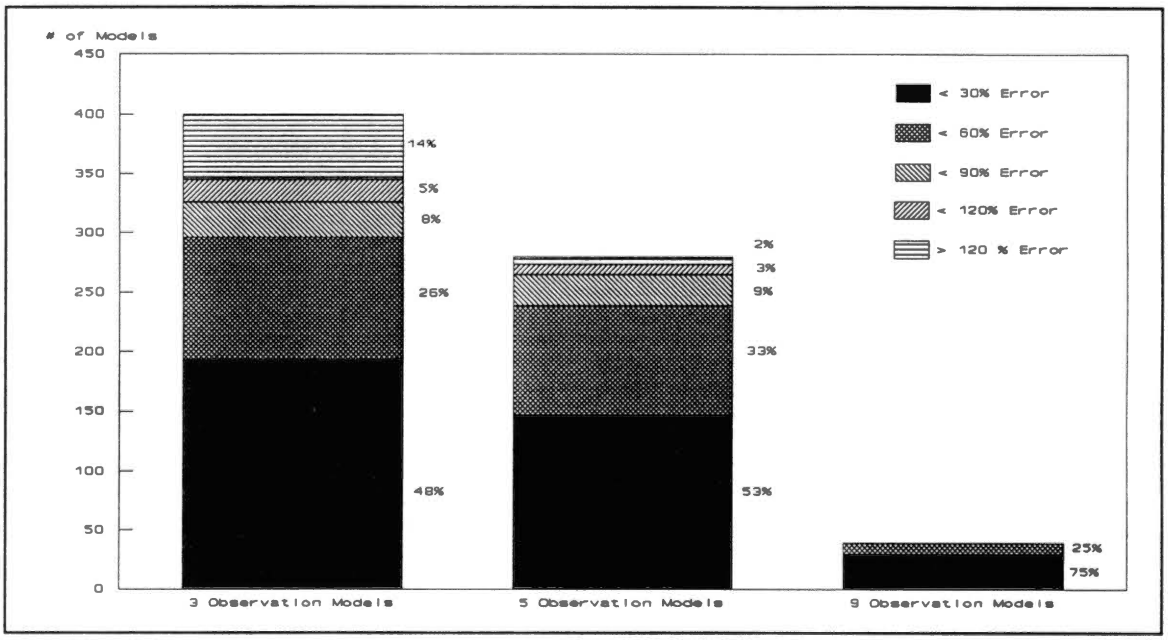


Figure 31. Distribution of Predictability Errors by the Number of Environments in the Available Set Excluding DP1 and Vector Environment Models

tested was realized.

Simply stated, this graph indicates that 50% of the models tested were able to predict the run time of an arbitrary application without knowing the type of processing performed. Furthermore, Figure 31 indicates that this prediction rate can be sustained with as little as three environments in the available set.

At the beginning of this chapter the question was raised as to whether the model was of any value. The results presented in Figure 30 and 31 indicate that it is of value. Work is still needed in several areas however, to ensure that the model will provide consistent results. These areas are identified and discussed in Chapter V.

CHAPTER V

CONCLUSIONS AND SUGGESTIONS FOR FUTURE WORK

The primary objective of this work is to define and test the validity of a model for predicting the performance of an application program within a new environment; an environment in which the application program has never run. This objective has been accomplished. Instructions for establishing a model have been presented along with suggestions to improve the predicted timings. Areas for future work are discussed below.

One area that merits further investigation is determining a way of establishing a confidence level in the Predictability Model. Although a method was presented to select a predicted time, there is no guarantee that this is the correct time to choose or, for that matter, whether or not the predicted time selected will be within a defined confidence range. A method must be established to determine the accuracy of the predicted result without running the application program on the tested machine; otherwise, the model would be of little value.

Another area that should be investigated further is the predictability equation. Although the modified

predictability equation presented in Chapter IV failed to improve the model's degree of predictability, other equations, including nonlinear equations, may be available that provide better estimates.

The model should be evaluated with a larger suite of benchmark and application programs. All the application programs presented in this work were written in FORTRAN and performed some type of vector processing. Consequently, FORTRAN benchmarks were selected that measured vector processing performance. It is unknown if the model's performance would be similar for other computing scenarios. For example, a suite of programs and benchmarks that perform a high degree of data base access could be defined and the predicted results compared to the findings presented in this work to determine if the model is consistent for other computing needs.

Finally, the model needs to be evaluated on a larger set of architectures. Although the model performed well for the architectures presented, the architectures were very similar to each other. More work is needed in this area to determine if predictability errors are similar when other types of architectures are introduced to the set.

In conclusion, the Predictability Model has potential and should be evaluated further. Although large percentage errors occurred for some of the Predictability Models, the ability of the Predictability Model to predict the

performance of an application, that takes four hours to run on an existing machine to within 70 seconds of its actual time of 113 seconds on an evaluated machine, seems appealing.

REFERENCES

Borovits, I. (1984). Management of Computer Operations. Englewood Cliffs, NJ: Prentice-Hall.

Colonna, J. D. (May 4, 1987). General Purpose Supercomputers. Information Week, 43-44.

Dongarra, J. J., Bunch, J. R., Moler, C. B., Stewart, G. W. (1982). Linpack Users' Guide. Philadelphia, PN: Society for Industrial and Applied Mathematics.

Dongarra, J. J. (Spring, 1988). The Linpack Benchmark: An Explanation. Supercomputing Magazine, 10-14.

Dongarra, J. J. (June 1, 1987). Performance of Various Computers Using Standard Linear Equations Software in a Fortran Environment. Argonne National Laboratory.

Freedman, D. (1978). Statistics. New York, NY: W. W. Norton & Company.

Gibson, D. H., Rain, D. W., and Walsh, H. F. (1986). Engineering and Scientific Processing on the IBM 3090. IBM Systems Journal, Vol. 25, No 1, 36-50.

Gilbreath, J. (September, 1981). A High-Level Language Benchmark. Byte, 180-198.

Green, L. (December, 1987). How to Rate Benchmarks. Information Week, 36-40.

Joslin, E. O. (1970). Techniques of Selecting EDP Equipment. Journal of Data Management, Vol. 8, No 2, 73-79.

Kindel, B. (February, 1989). How Fast is Fast? Byte, 251-254.

Ledbetter, C. (1988). How Do You Measure Supercomputer Performance? Control Data World, 16-17.

Lindsay, D. S. (December, 1987). A Technique for Evaluating the Performance of Vector Processors. National Advanced Systems.

Lindsay, D. S., and Bell, T. E. (1986). Directed Benchmarks for CPU Architecture Evaluation. CMG-86 Conference Proceedings, 379-385.

Lindsay, D. S. (December, 1987). Performance Architecture Comparison of Vector Processors on Large Scientific and Commercial Machines. National Advanced Systems.

MSC/NASTRAN Time Estimation and Problem Execution, (February, 1985). MacNeal-Schwendler Corporation, Los Angeles, California.

Senson, G. (1986). How to Assess the Results of Your CPU Benchmark. CMG-86 Conference Proceedings, 658-667.

Shaw, B. R. (September, 1987). Personal Evaluation Key to Technical Computing Selection. World Oil, 39-41.

Smith, R. D. (1986). Vector Processing and the Performance Analyst. CMG-86 Conference Proceedings, 738-743.

Sydow, P. J. (May, 1982). Optimization Guide. Cray Research, Inc.

Wasserman, H. J., Simmons, M. L., and Olaf, M. L. (August, 1987). The Performance of Minisupercomputers: Alliant, Convex, and SCS. Los Alamos National Laboratory.

Wilson, P. (March, 1988). Floating-Point Survival Kit. Byte, 217-226.

APPENDIXES

APPENDIX A

PROGRAM LISTINGS OF BENCHMARKS AND APPLICATIONS

Linpack Benchmark Listing
(with the permission of Dr.Jack J. Dongarra)

```
      PROGRAM LINPACK
      REAL AA(200,200),A(201,200),B(200),X(200)
      REAL TIME(8,6),CRAY,OPS,TOTAL,NORMA,NORMX
      REAL RESID,RESIDN,EPS,EPSLON
      INTEGER IPVT(200)
      LDA = 201
      LDAA = 200
C
      N = 100
      CRAY =  056
      WRITE(6,1)
    1 FORMAT(' PLEASE SEND THE RESULTS OF THIS
RUN TO@D'//
     $       ' JACK J  DONGARRA'/
     $       ' MATHEMATICS AND COMPUTER SCIENCE
DIVISION'/
     $       ' ARGONNE NATIONAL LABORATORY'/
     $       ' ARGONNE, ILLINOIS 60439'//
     $       ' TELEPHONE@D 312-972-7246'//
     $       ' ARPANET@D DONGARRA@AANL-MCS'/)
      OPS = (2 0E0*N**3)/3 0E0 + 2 0E0*N**2
C
         CALL MATGEN(A,LDA,N,B,NORMA)
         T1 = SECOND()
         CALL SGEFA(A,LDA,N,IPVT,INFO)
         TIME(1,1) = SECOND() - T1
         T1 = SECOND()
         CALL SGESL(A,LDA,N,IPVT,B,0)
         TIME(1,2) = SECOND() - T1
         TOTAL = TIME(1,1) + TIME(1,2)
C
C     COMPUTE A RESIDUAL TO VERIFY RESULTS
C
         DO 10 I = 1,N
            X(I) = B(I)
   10    CONTINUE
         CALL MATGEN(A,LDA,N,B,NORMA)
         DO 20 I = 1,N
            B(I) = -B(I)
   20    CONTINUE
         CALL SMXPY(N,B,N,LDA,X,A)
         RESID = 0 0
         NORMX = 0 0
         DO 30 I = 1,N
            RESID = AMAX1( RESID, ABS(B(I)) )
            NORMX = AMAX1( NORMX, ABS(X(I)) )
   30    CONTINUE
         EPS = EPSLON(1 0)
         RESIDN = RESID/( N*NORMA*NORMX*EPS )
         WRITE(6,40)
   40    FORMAT('     NORM  RESID      RESID
     MACHEP',
     $          '         X(1)          X(N)')
         WRITE(6,50) RESIDN,RESID,EPS,X(1),X(N)
   50    FORMAT(1P5E16 8)
C
         WRITE(6,60) N
   60    FORMAT(//'    TIMES ARE REPORTED FOR
MATRICES OF ORDER ',I5)
         WRITE(6,70)
   70
FORMAT(6X,'SGEFA',6X,'SGESL',6X,'TOTAL',5X,'MFLO
PS',7X,'UNIT',
     $        6X,'RATIO')
         TIME(1,3) = TOTAL
         TIME(1,4) = OPS/(1 0E6*TOTAL)
         TIME(1,5) = 2 0E0/TIME(1,4)
         TIME(1,6) = TOTAL/CRAY
         WRITE(6,80) LDA
   80    FORMAT(' TIMES FOR ARRAY WITH LEADING
DIMENSION OF',I4)
         WRITE(6,110) (TIME(1,I),I=1,6)
C
         CALL MATGEN(A,LDA,N,B,NORMA)
         T1 = SECOND()
         CALL SGEFA(A,LDA,N,IPVT,INFO)
         TIME(2,1) = SECOND() - T1
         T1 = SECOND()
         CALL SGESL(A,LDA,N,IPVT,B,0)
         TIME(2,2) = SECOND() - T1
         TOTAL = TIME(2,1) + TIME(2,2)
         TIME(2,3) = TOTAL
         TIME(2,4) = OPS/(1.0E6*TOTAL)
         TIME(2,5) = 2 0E0/TIME(2,4)
         TIME(2,6) = TOTAL/CRAY
C
         CALL MATGEN(A,LDA,N,B,NORMA)
         T1 = SECOND()
         CALL SGEFA(A,LDA,N,IPVT,INFO)
         TIME(3,1) = SECOND() - T1
         T1 = SECOND()
         CALL SGESL(A,LDA,N,IPVT,B,0)
         TIME(3,2) = SECOND() - T1
         TOTAL = TIME(3,1) + TIME(3,2)
         TIME(3,3) = TOTAL
         TIME(3,4) = OPS/(1 0E6*TOTAL)
         TIME(3,5) = 2 0E0/TIME(3,4)
         TIME(3,6) = TOTAL/CRAY
C
         NTIMES = 10
         TM2 = 0
         T1 = SECOND()
         DO 90 I = 1,NTIMES
            TM = SECOND()
            CALL MATGEN(A,LDA,N,B,NORMA)
            TM2 = TM2 + SECOND() - TM
            CALL SGEFA(A,LDA,N,IPVT,INFO)
   90    CONTINUE
         TIME(4,1) = (SECOND() - T1 -
TM2)/NTIMES
         T1 = SECOND()
         DO 100 I = 1,NTIMES
            CALL SGESL(A,LDA,N,IPVT,B,0)
  100    CONTINUE
         TIME(4,2) = (SECOND() - T1)/NTIMES
         TOTAL = TIME(4,1) + TIME(4,2)
         TIME(4.3) = TOTAL
         TIME(4,4) = OPS/(1.0E6*TOTAL)
         TIME(4,5) = 2.0E0/TIME(4,4)
         TIME(4,6) = TOTAL/CRAY
C
         WRITE(6,110) (TIME(2,I),I=1,6)
         WRITE(6,110) (TIME(3,I),I=1,6)
         WRITE(6,110) (TIME(4,I),I=1,6)
  110    FORMAT(6(1PE11.3))
C
         CALL MATGEN(AA,LDAA,N,B,NORMA)
         T1 = SECOND()
         CALL SGEFA(AA,LDAA,N,IPVT,INFO)
```

```
         TIME(5,1) = SECOND() - T1
         T1 = SECOND()
         CALL SGESL(AA,LDAA,N,IPVT,B,0)
         TIME(5,2) = SECOND() - T1
         TOTAL = TIME(5,1) + TIME(5,2)
         TIME(5,3) = TOTAL
         TIME(5,4) = OPS/(1 0E6*TOTAL)
         TIME(5,5) = 2 0E0/TIME(5,4)
         TIME(5,6) = TOTAL/CRAY
C
         CALL MATGEN(AA,LDAA,N,B,NORMA)
         T1 = SECOND()
         CALL SGEFA(AA,LDAA,N,IPVT,INFO)
         TIME(6,1) = SECOND() - T1
         T1 = SECOND()
         CALL SGESL(AA,LDAA,N,IPVT,B,0)
         TIME(6,2) = SECOND() - T1
         TOTAL = TIME(6,1) + TIME(6,2)
         TIME(6,3) = TOTAL
         TIME(6,4) = OPS/(1 0E6*TOTAL)
         TIME(6,5) = 2 0E0/TIME(6,4)
         TIME(6,6) = TOTAL/CRAY
C
         CALL MATGEN(AA,LDAA,N,B,NORMA)
         T1 = SECOND()
         CALL SGEFA(AA,LDAA,N,IPVT,INFO)
         TIME(7,1) = SECOND() - T1
         T1 = SECOND()
         CALL SGESL(AA,LDAA,N,IPVT,B,0)
         TIME(7,2) = SECOND() - T1
         TOTAL = TIME(7,1) + TIME(7,2)
         TIME(7,3) = TOTAL
         TIME(7,4) = OPS/(1 0E6*TOTAL)
         TIME(7,5) = 2 0E0/TIME(7,4)
         TIME(7,6) = TOTAL/CRAY
C
         NTIMES = 10
         TM2 = 0
         T1 = SECOND()
         DO 120 I = 1,NTIMES
            TM = SECOND()
            CALL MATGEN(AA,LDAA,N,B,NORMA)
            TM2 = TM2 + SECOND() - TM
            CALL SGEFA(AA,LDAA,N,IPVT,INFO)
  120    CONTINUE
         TIME(8,1) = (SECOND() - T1 -
TM2)/NTIMES
         T1 = SECOND()
         DO 130 I = 1,NTIMES
            CALL SGESL(AA,LDAA,N,IPVT,B,0)
  130    CONTINUE
         TIME(8,2) = (SECOND() - T1)/NTIMES
         TOTAL = TIME(8,1) + TIME(8,2)
         TIME(8,3) = TOTAL
         TIME(8,4) = OPS/(1 0E6*TOTAL)
         TIME(8,5) = 2 0E0/TIME(8,4)
         TIME(8,6) = TOTAL/CRAY
C
         WRITE(6,140) LDAA
  140    FORMAT(/' TIMES FOR ARRAY WITH LEADING
DIMENSION OF',I4)
         WRITE(6,110) (TIME(5,I),I=1,6)
         WRITE(6,110) (TIME(6,I),I=1,6)
         WRITE(6,110) (TIME(7,I),I=1,6)
         WRITE(6,110) (TIME(8,I),I=1,6)
      STOP
      END
      SUBROUTINE MATGEN(A,LDA,N,B,NORMA)
      REAL A(LDA,1),B(1),NORMA
C
      INIT = 1325
      NORMA = 0 0
      DO 30 J = 1,N
         DO 20 I = 1,N
            INIT = MOD(3125*INIT,65536)
            A(I,J) = (INIT - 32768.0)/16384 0
            NORMA = AMAX1(A(I,J), NORMA)
   20    CONTINUE
   30 CONTINUE
      DO 35 I = 1,N
          B(I) = 0.0
   35 CONTINUE
      DO 50 J = 1,N
         DO 40 I = 1,N
            B(I) = B(I) + A(I,J)
   40    CONTINUE
   50 CONTINUE
      RETURN
      END
      SUBROUTINE SGEFA(A,LDA,N,IPVT,INFO)
      INTEGER LDA,N,IPVT(1),INFO
      REAL A(LDA,1)
C
C     SGEFA FACTORS A REAL MATRIX BY GAUSSIAN
ELIMINATION.
C
C     SGEFA IS USUALLY CALLED BY DGECO, BUT IT
CAN BE CALLED
C     DIRECTLY WITH A SAVING IN TIME IF  RCOND
IS NOT NEEDED.
C     (TIME FOR DGECO) = (1 + 9/N)*(TIME FOR
SGEFA)
C
C     ON ENTRY
C
C        A       REAL(LDA, N)
C                THE MATRIX TO BE FACTORED
C
C        LDA     INTEGER
C                THE LEADING DIMENSION OF THE
ARRAY  A
C
C        N       INTEGER
C                THE ORDER OF THE MATRIX  A
C
C     ON RETURN
C
C        A       AN UPPER TRIANGULAR MATRIX AND
THE MULTIPLIERS
C                WHICH WERE USED TO OBTAIN IT
C                THE FACTORIZATION CAN BE
WRITTEN  A = L*U  WHERE
C                L  IS A PRODUCT OF PERMUTATION
AND UNIT LOWER
C                TRIANGULAR MATRICES AND  U  IS
UPPER TRIANGULAR
C
C        IPVT    INTEGER(N)
C                AN INTEGER VECTOR OF PIVOT
INDICES
C
C        INFO    INTEGER
C                = 0  NORMAL VALUE.
C                = K  IF  U(K,K) .EQ  0 0
THIS IS NOT AN ERROR
C                     CONDITION FOR THIS
SUBROUTINE, BUT IT DOES
C                     INDICATE THAT SGESL OR
```

```
DGEDI WILL DIVIDE BY ZERO
C                       IF CALLED   USE  RCOND  IN
DGECO FOR A RELIABLE
C                       INDICATION OF SINGULARITY
C
C     LINPACK  THIS VERSION DATED 08/14/78
C     CLEVE MOLER, UNIVERSITY OF NEW MEXICO,
ARGONNE NATIONAL LAB
C
C     SUBROUTINES AND FUNCTIONS
C
C     BLAS SAXPY,SSCAL,ISAMAX
C
C     INTERNAL VARIABLES
C
      REAL T
      INTEGER ISAMAX,J,K,KP1,L,NM1
C
C
C     GAUSSIAN ELIMINATION WITH PARTIAL PIVOTING
C
      INFO = 0
      NM1 = N - 1
      IF (NM1  LT  1) GO TO 70
      DO 60 K = 1, NM1
         KP1 = K + 1
C
C        FIND L = PIVOT INDEX
C
         L = ISAMAX(N-K+1,A(K,K),1) + K - 1
         IPVT(K) = L
C
C        ZERO PIVOT IMPLIES THIS COLUMN ALREADY
TRIANGULARIZED
C
         IF (A(L,K)  EQ  0 0E0) GO TO 40
C
C           INTERCHANGE IF NECESSARY
C
            IF (L  EQ  K) GO TO 10
               T = A(L,K)
               A(L,K) = A(K,K)
               A(K,K) = T
   10       CONTINUE
C
C           COMPUTE MULTIPLIERS
C
            T = -1 0E0/A(K,K)
            CALL SSCAL(N-K,T,A(K+1,K),1)
C
C           ROW ELIMINATION WITH COLUMN INDEXING
C
            DO 30 J = KP1, N
               T = A(L,J)
               IF (L  EQ  K) GO TO 20
                  A(L,J) = A(K,J)
                  A(K,J) = T
   20          CONTINUE
               CALL
SAXPY(N-K,T,A(K+1,K),1,A(K+1,J),1)
   30       CONTINUE
         GO TO 50
   40    CONTINUE
            INFO = K
   50    CONTINUE
   60 CONTINUE
   70 CONTINUE
      IPVT(N) = N
      IF (A(N,N)  EQ  0 0E0) INFO = N
      RETURN
      END
      SUBROUTINE SGESL(A,LDA,N,IPVT,B,JOB)
      INTEGER LDA,N,IPVT(1),JOB
      REAL A(LDA,1),B(1)
C
C     SGESL SOLVES THE REAL SYSTEM
C     A * X = B  OR  TRANS(A) * X = B
C     USING THE FACTORS COMPUTED BY DGECO OR
SGEFA
C
C     ON ENTRY
C
C        A       REAL(LDA, N)
C                THE OUTPUT FROM DGECO OR SGEFA
C
C        LDA     INTEGER
C                THE LEADING DIMENSION OF THE
ARRAY  A
C
C        N       INTEGER
C                THE ORDER OF THE MATRIX  A
C
C        IPVT    INTEGER(N)
C                THE PIVOT VECTOR FROM DGECO OR
SGEFA
C
C        B       REAL(N)
C                THE RIGHT HAND SIDE VECTOR
C
C        JOB     INTEGER
C                = 0         TO SOLVE  A*X = B ,
C                = NONZERO   TO SOLVE
TRANS(A)*X = B  WHERE
C                            TRANS(A)  IS THE
TRANSPOSE.
C
C     ON RETURN
C
C        B       THE SOLUTION VECTOR  X
C
C     ERROR CONDITION
C
C        A DIVISION BY ZERO WILL OCCUR IF THE
INPUT FACTOR CONTAINS A
C        ZERO ON THE DIAGONAL   TECHNICALLY THIS
INDICATES SINGULARITY
C        BUT IT IS OFTEN CAUSED BY IMPROPER
ARGUMENTS OR IMPROPER
C        SETTING OF LDA    IT WILL NOT OCCUR IF
THE SUBROUTINES ARE
C        CALLED CORRECTLY AND IF DGECO HAS SET
RCOND  GT  0 0
C        OR SGEFA HAS SET INFO .EQ  0
C
C     TO COMPUTE  INVERSE(A) * C  WHERE  C  IS A
MATRIX
C     WITH  P  COLUMNS
C           CALL DGECO(A,LDA,N,IPVT,RCOND,Z)
C           IF (RCOND IS TOO SMALL) GO TO
C           DO 10 J = 1, P
C              CALL SGESL(A,LDA,N,IPVT,C(1,J),0)
C        10 CONTINUE
C
C     LINPACK. THIS VERSION DATED 08/14/78
C     CLEVE MOLER, UNIVERSITY OF NEW MEXICO,
ARGONNE NATIONAL LAB.
C
C     SUBROUTINES AND FUNCTIONS
```

```
C
C     BLAS SAXPY,SDOT
C
C     INTERNAL VARIABLES
C
      REAL SDOT,T
      INTEGER K,KB,L,NM1
C
      NM1 = N - 1
      IF (JOB  NE  0) GO TO 50
C
C        JOB = 0 , SOLVE  A * X = B
C        FIRST SOLVE  L*Y = B
C
         IF (NM1  LT  1) GO TO 30
         DO 20 K = 1, NM1
            L = IPVT(K)
            T = B(L)
            IF (L  EQ  K) GO TO 10
               B(L) = B(K)
               B(K) = T
   10       CONTINUE
            CALL
SAXPY(N-K,T,A(K+1,K),1,B(K+1),1)
   20    CONTINUE
   30    CONTINUE
C
C        NOW SOLVE  U*X = Y
C
         DO 40 KB = 1, N
            K = N + 1 - KB
            B(K) = B(K)/A(K,K)
            T = -B(K)
            CALL SAXPY(K-1,T,A(1,K),1,B(1),1)
   40    CONTINUE
      GO TO 100
   50 CONTINUE
C
C        JOB = NONZERO, SOLVE  TRANS(A) * X = B
C        FIRST SOLVE  TRANS(U)*Y = B
C
         DO 60 K = 1, N
            T = SDOT(K-1,A(1,K),1,B(1),1)
            B(K) = (B(K) - T)/A(K,K)
   60    CONTINUE
C
C        NOW SOLVE TRANS(L)*X = Y
C
         IF (NM1  LT  1) GO TO 90
         DO 80 KB = 1, NM1
            K = N - KB
            B(K) = B(K) +
SDOT(N-K,A(K+1,K),1,B(K+1),1)
            L = IPVT(K)
            IF (L  EQ  K) GO TO 70
               T = B(L)
               B(L) = B(K)
               B(K) = T
   70       CONTINUE
   80    CONTINUE
   90    CONTINUE
  100 CONTINUE
      RETURN
      END
      SUBROUTINE SAXPY(N,DA,DX,INCX,DY,INCY)
C
C     CONSTANT TIMES A VECTOR PLUS A VECTOR
C     JACK DONGARRA, LINPACK, 3/11/78
C
      REAL DX(1),DY(1),DA
      INTEGER I,INCX,INCY,IX,IY,M,MP1,N
C
      IF(N LE.0)RETURN
      IF (DA  EQ  0 0E0) RETURN
      IF(INCX EQ 1 AND INCY EQ.1)GO TO 20
C
C        CODE FOR UNEQUAL INCREMENTS OR EQUAL
INCREMENTS
C          NOT EQUAL TO 1
C
      IX = 1
      IY = 1
      IF(INCX.LT 0)IX = (-N+1)*INCX + 1
      IF(INCY.LT.0)IY = (-N+1)*INCY + 1
      DO 10 I = 1,N
        DY(IY) = DY(IY) + DA*DX(IX)
        IX = IX + INCX
        IY = IY + INCY
   10 CONTINUE
      RETURN
C
C        CODE FOR BOTH INCREMENTS EQUAL TO 1
C
   20 CONTINUE
      DO 30 I = 1,N
        DY(I) = DY(I) + DA*DX(I)
   30 CONTINUE
      RETURN
      END
      REAL FUNCTION SDOT(N,DX,INCX,DY,INCY)
C
C     FORMS THE DOT PRODUCT OF TWO VECTORS
C     JACK DONGARRA, LINPACK, 3/11/78
C
      REAL DX(1),DY(1),DTEMP
      INTEGER I,INCX,INCY,IX,IY,M,MP1,N
C
      SDOT = 0 0E0
      DTEMP = 0 0E0
      IF(N LE 0)RETURN
      IF(INCX EQ 1.AND.INCY EQ 1)GO TO 20
C
C        CODE FOR UNEQUAL INCREMENTS OR EQUAL
INCREMENTS
C          NOT EQUAL TO 1
C
      IX = 1
      IY = 1
      IF(INCX LT 0)IX = (-N+1)*INCX + 1
      IF(INCY LT.0)IY = (-N+1)*INCY + 1
      DO 10 I = 1,N
        DTEMP = DTEMP + DX(IX)*DY(IY)
        IX = IX + INCX
        IY = IY + INCY
   10 CONTINUE
      SDOT = DTEMP
      RETURN
C
C        CODE FOR BOTH INCREMENTS EQUAL TO 1
C
   20 CONTINUE
      DO 30 I = 1,N
        DTEMP = DTEMP + DX(I)*DY(I)
   30 CONTINUE
      SDOT = DTEMP
      RETURN
      END
      SUBROUTINE  SSCAL(N,DA,DX,INCX)
```

```
C
C     SCALES A VECTOR BY A CONSTANT
C     JACK DONGARRA, LINPACK, 3/11/78
C
      REAL DA,DX(1)
      INTEGER I,INCX,M,MP1,N,NINCX
C
      IF(N.LE 0)RETURN
      IF(INCX EQ 1)GO TO 20
C
C        CODE FOR INCREMENT NOT EQUAL TO 1
C
      NINCX = N*INCX
      DO 10 I = 1,NINCX,INCX
        DX(I) = DA*DX(I)
   10 CONTINUE
      RETURN
C
C        CODE FOR INCREMENT EQUAL TO 1
C
   20 CONTINUE
      DO 30 I = 1,N
        DX(I) = DA*DX(I)
   30 CONTINUE
      RETURN
      END
      INTEGER FUNCTION ISAMAX(N,DX,INCX)
C
C     FINDS THE INDEX OF ELEMENT HAVING MAX
ABSOLUTE VALUE
C     JACK DONGARRA, LINPACK, 3/11/78
C
      REAL DX(1),DMAX
      INTEGER I,INCX,IX,N
C
      ISAMAX = 0
      IF( N  LT  1 ) RETURN
      ISAMAX = 1
      IF(N EQ 1)RETURN
      IF(INCX EQ 1)GO TO 20
C
C        CODE FOR INCREMENT NOT EQUAL TO 1
C
      IX = 1
      DMAX = ABS(DX(1))
      IX = IX + INCX
      DO 10 I = 2,N
         IF(ABS(DX(IX)) LE DMAX) GO TO 5
         ISAMAX = I
         DMAX = ABS(DX(IX))
    5    IX = IX + INCX
   10 CONTINUE
      RETURN
C
C        CODE FOR INCREMENT EQUAL TO 1
C
   20 DMAX = ABS(DX(1))
      DO 30 I = 2,N
         IF(ABS(DX(I)) LE DMAX) GO TO 30
         ISAMAX = I
         DMAX = ABS(DX(I))
   30 CONTINUE
      RETURN
      END
      REAL FUNCTION EPSLON (X)
      REAL X
C
C     ESTIMATE UNIT ROUNDOFF IN QUANTITIES OF
SIZE X
C
      REAL A,B,C,EPS
C
C     THIS PROGRAM SHOULD FUNCTION PROPERLY ON
ALL SYSTEMS
C     SATISFYING THE FOLLOWING TWO ASSUMPTIONS,
C        1.  THE BASE USED IN REPRESENTING
FLOATING POINT
C            NUMBERS IS NOT A POWER OF THREE
C        2.  THE QUANTITY  A  IN STATEMENT 10 IS
REPRESENTED TO
C            THE ACCURACY USED IN FLOATING POINT
VARIABLES
C            THAT ARE STORED IN MEMORY
C     THE STATEMENT NUMBER 10 AND THE GO TO 10
ARE INTENDED TO
C     FORCE OPTIMIZING COMPILERS TO GENERATE
CODE SATISFYING
C     ASSUMPTION 2.
C     UNDER THESE ASSUMPTIONS, IT SHOULD BE TRUE
THAT,
C            A  IS NOT EXACTLY EQUAL TO
FOUR-THIRDS,
C            B  HAS A ZERO FOR ITS LAST BIT OR
DIGIT,
C            C  IS NOT EXACTLY EQUAL TO ONE,
C            EPS  MEASURES THE SEPARATION OF 1 0
FROM
C                 THE NEXT LARGER FLOATING POINT
NUMBER
C     THE DEVELOPERS OF EISPACK WOULD APPRECIATE
BEING INFORMED
C     ABOUT ANY SYSTEMS WHERE THESE ASSUMPTIONS
DO NOT HOLD
C
C
**********************************************
**********
C     THIS ROUTINE IS ONE OF THE AUXILIARY
ROUTINES USED BY EISPACK III
C     TO AVOID MACHINE DEPENDENCIES
C
**********************************************
**********
C
C     THIS VERSION DATED 4/6/83
C
      A = 4 0E0/3 0E0
   10 B = A - 1 0E0
      C = B + B + B
      EPS = ABS(C-1.0E0)
      IF (EPS  EQ. 0 0E0) GO TO 10
      EPSLON = EPS*ABS(X)
      RETURN
      END
      SUBROUTINE SMXPY (N1, Y, N2, LDM, X, M)
      REAL Y(*), X(*), M(LDM,*)
C
C   PURPOSE@D
C     MULTIPLY MATRIX M TIMES VECTOR X AND ADD
THE RESULT TO VECTOR Y.
C
C   PARAMETERS@D
C
C     N1 INTEGER, NUMBER OF ELEMENTS IN VECTOR
Y, AND NUMBER OF ROWS IN
C         MATRIX M
C
C     Y REAL(N1), VECTOR OF LENGTH N1 TO WHICH
```

```
IS ADDED THE PRODUCT M*X
C
C     N2 INTEGER, NUMBER OF ELEMENTS IN VECTOR
X, AND NUMBER OF COLUMNS
C          IN MATRIX M
C
C     LDM INTEGER, LEADING DIMENSION OF ARRAY M
C
C     X REAL(N2), VECTOR OF LENGTH N2
C
C     M REAL(LDM,N2), MATRIX OF N1 ROWS AND N2
COLUMNS
C
C
-----------------------------------------------
---------------------
C
C   CLEANUP ODD VECTOR
C
      J = MOD(N2,2)
      IF (J  GE  1) THEN
         DO 10 I = 1, N1
            Y(I) = (Y(I)) + X(J)*M(I,J)
   10    CONTINUE
      ENDIF
C
C   CLEANUP ODD GROUP OF TWO VECTORS
C
      J = MOD(N2,4)
      IF (J  GE  2) THEN
         DO 20 I = 1, N1
            Y(I) = ( (Y(I))
     $             + X(J-1)*M(I,J-1)) +
X(J)*M(I,J)
   20    CONTINUE
      ENDIF
C
C   CLEANUP ODD GROUP OF FOUR VECTORS
C
      J = MOD(N2,8)
      IF (J  GE  4) THEN
         DO 30 I = 1, N1
            Y(I) = ((( (Y(I))
     $             + X(J-3)*M(I,J-3)) +
X(J-2)*M(I,J-2))
     $             + X(J-1)*M(I,J-1)) + X(J)
*M(I,J)
   30    CONTINUE
      ENDIF
C
C   CLEANUP ODD GROUP OF EIGHT VECTORS
C
      J = MOD(N2,16)
      IF (J  GE  8) THEN
         DO 40 I = 1, N1
            Y(I) = (((((( (Y(I))
     $             + X(J-7)*M(I,J-7)) +
X(J-6)*M(I,J-6))
     $             + X(J-5)*M(I,J-5)) +
X(J-4)*M(I,J-4))
     $             + X(J-3)*M(I,J-3)) +
X(J-2)*M(I,J-2))
     $             + X(J-1)*M(I,J-1)) + X(J)
*M(I,J)
   40    CONTINUE
      ENDIF
C
C   MAIN LOOP - GROUPS OF SIXTEEN VECTORS
C
      JMIN = J+16
      DO 60 J = JMIN, N2, 16
         DO 50 I = 1, N1
            Y(I) = ((((((((((((((( (Y(I))
     $             + X(J-15)*M(I,J-15)) +
X(J-14)*M(I,J-14))
     $             + X(J-13)*M(I,J-13)) +
X(J-12)*M(I,J-12))
     $             + X(J-11)*M(I,J-11)) +
X(J-10)*M(I,J-10))
     $             + X(J- 9)*M(I,J- 9)) + X(J-
8)*M(I,J- 8))
     $             + X(J- 7)*M(I,J- 7)) + X(J-
6)*M(I,J- 6))
     $             + X(J- 5)*M(I,J- 5)) + X(J-
4)*M(I,J- 4))
     $             + X(J- 3)*M(I,J- 3)) + X(J-
2)*M(I,J- 2))
     $             + X(J- 1)*M(I,J- 1)) + X(J)
*M(I,J)
   50    CONTINUE
   60 CONTINUE
      RETURN
      END
```

Lindsay Benchmark Listing
(with permission of Dr. David S. Lindsay)

```
      PROGRAM BENCHV
      IMPLICIT INTEGER (I-N)
      IMPLICIT REAL (V)
************************************************
**********************
*                                              *
*   FORTRAN BENCHMARK PROGRAM VERSION 3 0 8
(08/10/87)                *
*         AUTHOR@D                             *
*                                              *
*     DR  DAVID S  LINDSAY                     *
*                                              *
*  THIS IS AN EXTENSION TO THE BENCH/BENCH1 PAIR
OR TO BENCH2          *
*    IT IS DESIGNED TO WORK WITH OPTIMIZING
COMPILERS, AND            *
*    SPECIFICALLY TO GENERATE MATHEMATICAL CODE
THAT                      *
*    IS VECTORIZABLE   IN THIS WAY, THE SPEED OF
MACHINES                *
*    WITH VECTOR PROCESSORS (E G , CRAY, IBM
3090,                     *
*    NAS 91X0 & XL SERIES) CAN BE COMPARED WITH
NON-VECTOR              *
*    MACHINES (OR WITH THE SAME MNACHINES
EXECUTING SCALAR CODE)     *
*    -- AT FORTRAN SOURCE CODE LEVEL          *
*                                              *
*   COMMON DEFINITIONS@D                       *
*                                              *
*     EACH COMMON BLOCK IN THE TEST IS
REFERENCED HERE, IN CASE      *
*     SPECIAL STORAGE OPTIONS NEED TO BE
SPECIFIED IN THE MAIN         *
*     PROGRAM                                  *
*                                              *
*        THE ARRAYS CONTAINING THE TEST VECTORS
ARE DIMENSIONED      *
*     200,003 RATHER THAN 200,000 TO AVOID
MEMORY BANK CONFLICTS      *
*     IF YOU ARE GOING TO REDIMENSION THEM TO
MAKE THEM SMALLER,     *
*     USE THE FIRST PRIME NUMBER GREATER THAN
THE DESIRED DIMENSION  *
*        A TABLE OF PRIME NUMBERS FOLLOWS@D     *
*                                              *
*     200,003                                  *
*     100,003                                  *
*      50,021                                  *
*      20,011                                  *
*      10,007                                  *
*       5,003                                  *
*       2,003                                  *
*       1,009                                  *
*                                              *
************************************************
**********************
      REAL ETIMES, ETIME, STIME, BGNTIM, ENDTIM

      COMMON /MINTST/ MINTST
      COMMON /MAXTST/ MAXTST

      COMMON /ODATA/ ETIMES (2, 1500)

      COMMON /PARMS/ ITYPE, NREP, NCALIB

      COMMON /RDPARM/ IPARM(3,150),NUMTST,ICURTS

      COMMON /VECTOR/ V00(200003)

      COMMON /VVECT/ VV00(200003)

      COMMON /VRESLT/ VA00(200003)

      COMMON /RESULT/ VANS00


*
*    FIRST FIND STARTING CPU TIME FOR OVERALL
CALCULATION
*
      CALL CPTIME (BGNTIM)
*
*   INITIALIZE THE BIG ARRAYS
*
      DO 1234 I = 1, 200003
         V00(I) = 1
         VV00(I) = 1
         VA00(I) = 1
1234  CONTINUE
*
*    OPEN THE INPUT AND OUTPUT FILES HERE, FIRST
TIME IN.
*
      OPEN (UNIT=15, FILE='BNCHDAT',
STATUS='OLD')
C      OPEN (UNIT=6, FILE='BNCHOUT',
STATUS='NEW')
C      OPEN (UNIT=7, FILE='BNCHSUM',
STATUS='NEW')

*
*    INITILIZE KEY PROGRAM VARIABLES
*
      IEOFF = 0
      ICURTS = 0
```

```
*
*    READ INPUT PARAMETERS FROM FILE (IF IT
EXISTS)
*
      CALL RDCARD
*
*    LOOP ON EACH TEST CASE
*
 5    CONTINUE
*
*    CHECK IF TIME TO TERMINATE (CURRENT TEST
NUMBER  GT. NUMBER
*      OF TESTS TO PERFORM)
*
      IF (ICURTS .GT  NUMTST) GOTO 20300
*
*    SET CONTROL VARIABLES FOR THIS TEST CASE
*
      ITYPE  = IPARM(1,ICURTS)
      NREP   = IPARM(2,ICURTS)
      NCALIB = IPARM(3,ICURTS)
*
*   VALIDATE INPUT PARAMETERS
*
      IF (NCALIB  LE  0) NCALIB=1
      IF (NREP  LE  0) NREP=1
      IF ((ITYPE  GE  MINTST)  AND  (ITYPE  LE
MAXTST)) GO TO 10
        WRITE (6,9001) ITYPE,NREP,NCALIB
        GOTO 20400
  10  CONTINUE
*
*   SET UP TO REPEAT THE SAME TEST
*
      DO 20200 IREP = 1, NREP
*
*   GET INITIAL TIME FOR LOOP
*
        CALL CPTIME(STIME)
*
*   BRANCH TO DESIRED TEST TYPE
*
        NTEST = ITYPE - MINTST + 1
        DO 20100 LOOP = 1, NCALIB
        GOTO (
        100,200,300,400,500,600,700,800,900,

1000,1100,1200,1300,1400,1500,1600,1700,1800,190
0,

2000,2100,2200,2300,2400,2500,2600,2700,2800,290
0

,3000,3100,3200,3300,3400,3500,3600,3700,3800,39
00

,4000,4100,4200,4300,4400,4500,4600,4700,4800,49
00

,5000,5100,5200,5300,5400,5500,5600,5700,5800,59
00

,6000,6100,6200,6300,6400,6500,6600,6700,6800,69
00

,7000,7100,7200,7300,7400,7500,7600,7700,7800,79
00

,8000,8100,8200,8300,8400,8500,8600,8700,8800,89
00

,9000,9100,9200,9300,9400,9500,9600,9700,9800,99
00
      .
,10000,10100,10200,10300,10400,10500,10600,10700
,10800,10900
      .
,11000,11100,11200,11300,11400,11500,11600,11700
,11800,11900
      .
,12000,12100,12200,12300,12400,12500,12600,12700
,12800
     1 ),NTEST
*
100   CALL ASGN20(1)
      GO TO 20100
*
200   CALL ASGN40(1)
      GO TO 20100
*
300   CALL ASGN20(5)
      GO TO 20100
*
400   CALL ASGN40(5)
      GO TO 20100
*
500   CALL ASGN20(10)
      GO TO 20100
*
600   CALL ASGN40(10)
      GO TO 20100
*
700   CALL ASGN20(20)
      GO TO 20100
*
800   CALL ASGN40(20)
      GO TO 20100
*
900   CALL ASGN20(50)
      GO TO 20100
*
1000  CALL ASGN40(50)
      GO TO 20100
*
1100  CALL ASGN20(100)
      GO TO 20100
*
1200  CALL ASGN40(100)
      GO TO 20100
*
1300  CALL ASGN20(200)
      GO TO 20100
*
1400  CALL ASGN40(200)
      GO TO 20100
*
1500  CALL ASGN20(500)
      GO TO 20100
*
1600  CALL ASGN40(500)
      GO TO 20100
*
1700  CALL DOT20(1)
      GO TO 20100
*
1800  CALL DOT40(1)
      GO TO 20100
*
```

```
1900  CALL DOT20(5)
      GO TO 20100
*
2000  CALL DOT40(5)
      GO TO 20100
*
2100  CALL DOT20(10)
      GO TO 20100
*
2200  CALL DOT40(10)
      GO TO 20100
*
2300  CALL DOT20(20)
      GO TO 20100
*
2400  CALL DOT40(20)
      GO TO 20100
*
2500  CALL DOT20(50)
      GO TO 20100
*
2600  CALL DOT40(50)
      GO TO 20100
*
2700  CALL DOT20(100)
      GO TO 20100
*
2800  CALL DOT40(100)
      GO TO 20100
*
2900  CALL DOT20(200)
      GO TO 20100
*
3000  CALL DOT40(200)
      GO TO 20100
*
3100  CALL DOT20(500)
      GO TO 20100
*
3200  CALL DOT40(500)
      GO TO 20100
*
3300  CALL VSVV20(1)
      GO TO 20100
*
3400  CALL VSVV40(1)
      GO TO 20100
*
3500  CALL VSVV20(5)
      GO TO 20100
*
3600  CALL VSVV40(5)
      GO TO 20100
*
3700  CALL VSVV20(10)
      GO TO 20100
*
3800  CALL VSVV40(10)
      GO TO 20100
*
3900  CALL VSVV20(20)
      GO TO 20100
*
4000  CALL VSVV40(20)
      GO TO 20100
*
4100  CALL VSVV20(50)
      GO TO 20100
*
4200  CALL VSVV40(50)
      GO TO 20100
*
4300  CALL VSVV20(100)
      GO TO 20100
*
4400  CALL VSVV40(100)
      GO TO 20100
*
4500  CALL VSVV20(200)
      GO TO 20100
*
4600  CALL VSVV40(200)
      GO TO 20100
*
4700  CALL VSVV20(500)
      GO TO 20100
*
4800  CALL VSVV40(500)
      GO TO 20100
*
4900  CALL VMLT20(1)
      GO TO 20100
*
5000  CALL VMLT40(1)
      GO TO 20100
*
5100  CALL VMLT20(5)
      GO TO 20100
*
5200  CALL VMLT40(5)
      GO TO 20100
*
5300  CALL VMLT20(10)
      GO TO 20100
*
5400  CALL VMLT40(10)
      GO TO 20100
*
5500  CALL VMLT20(20)
      GO TO 20100
*
5600  CALL VMLT40(20)
      GO TO 20100
*
5700  CALL VMLT20(50)
      GO TO 20100
*
5800  CALL VMLT40(50)
      GO TO 20100
*
5900  CALL VMLT20(100)
      GO TO 20100
*
6000  CALL VMLT40(100)
      GO TO 20100
*
6100  CALL VMLT20(200)
      GO TO 20100
*
6200  CALL VMLT40(200)
      GO TO 20100
*
6300  CALL VMLT20(500)
      GO TO 20100
*
6400  CALL VMLT40(500)
      GO TO 20100
*
```

```
36500  CALL VACC20(1)
       GO TO 20100
*
6600   CALL VACC40(1)
       GO TO 20100
*
6700   CALL VACC20(5)
       GO TO 20100
*
6800   CALL VACC40(5)
       GO TO 20100
*
6900   CALL VACC20(10)
       GO TO 20100
*
7000   CALL VACC40(10)
       GO TO 20100
*
7100   CALL VACC20(20)
       GO TO 20100
*
7200   CALL VACC40(20)
       GO TO 20100
*
7300   CALL VACC20(50)
       GO TO 20100
*
7400   CALL VACC40(50)
       GO TO 20100
*
7500   CALL VACC20(100)
       GO TO 20100
*
7600   CALL VACC40(100)
       GO TO 20100
*
7700   CALL VACC20(200)
       GO TO 20100
*
7800   CALL VACC40(200)
       GO TO 20100
*
7900   CALL VACC20(500)
       GO TO 20100
*
8000   CALL VACC40(500)
       GO TO 20100
*
8100   CALL VADD20(1)
       GO TO 20100

8200   CALL VADD40(1)
       GO TO 20100
*
8300   CALL VADD20(5)
       GO TO 20100
*
8400   CALL VADD40(5)
       GO TO 20100
*
8500   CALL VADD20(10)
       GO TO 20100
*
8600   CALL VADD40(10)
       GO TO 20100
*
8700   CALL VADD20(20)
       GO TO 20100
*
8800   CALL VADD40(20)
       GO TO 20100
*
8900   CALL VADD20(50)
       GO TO 20100
*
9000   CALL VADD40(50)
       GO TO 20100
*
9100   CALL VADD20(100)
       GO TO 20100
*
9200   CALL VADD40(100)
       GO TO 20100
*
9300   CALL VADD20(200)
       GO TO 20100
*
9400   CALL VADD40(200)
       GO TO 20100
*
9500   CALL VADD20(500)
       GO TO 20100
*
9600   CALL VADD40(500)
       GO TO 20100
*
9700   CALL DIV20(1)
       GO TO 20100
*
9800   CALL DIV40(1)
       GO TO 20100
*
9900   CALL DIV20(5)
       GO TO 20100
*
10000  CALL DIV40(5)
       GO TO 20100
*
10100  CALL DIV20(10)
       GO TO 20100
*
10200  CALL DIV40(10)
       GO TO 20100
*
10300  CALL DIV20(20)
       GO TO 20100
*
10400  CALL DIV40(20)
       GO TO 20100
*
10500  CALL DIV20(50)
       GO TO 20100
*
10600  CALL DIV40(50)
       GO TO 20100
*
10700  CALL DIV20(100)
       GO TO 20100
*
10800  CALL DIV40(100)
       GO TO 20100
*
10900  CALL DIV20(200)
       GO TO 20100
*
11000  CALL DIV40(200)
       GO TO 20100
*
```

```
11100 CALL DIV20(500)
      GO TO 20100
*
11200 CALL DIV40(500)
      GO TO 20100
*
11300  CALL MLAD20(1)
      GO TO 20100
*
11400  CALL MLAD40(1)
      GO TO 20100
*
11500  CALL MLAD20(5)
      GO TO 20100
*
11600 CALL MLAD40(5)
      GO TO 20100
*
11700 CALL MLAD20(10)
      GO TO 20100
*
11800 CALL MLAD40(10)
      GO TO 20100
*
11900 CALL MLAD20(20)
      GO TO 20100
*
12000 CALL MLAD40(20)
      GO TO 20100
*
12100 CALL MLAD20(50)
      GO TO 20100
*
12200 CALL MLAD40(50)
      GO TO 20100
*
12300 CALL MLAD20(100)
      GO TO 20100
*
12400 CALL MLAD40(100)
      GO TO 20100
*
12500 CALL MLAD20(200)
      GO TO 20100
*
12600 CALL MLAD40(200)
      GO TO 20100
*
12700 CALL MLAD20(500)
      GO TO 20100
*
12800 CALL MLAD40(500)
      GO TO 20100
*
20100  CONTINUE
************************************************
**********************
*    END OF LOOP - GET END TIME,COMPUTE ELAPSED
TIME, AND SAVE       *
************************************************
**********************
        CALL CPTIME(ETIME)
        IDXE = IREP + IEOFF
*
*       MAKE SURE NOT EXCEEDING ETIMES ARRAY
*
        IF (IDXE  LE  1500) GOTO 20110
          WRITE(6,9002)
ITYPE,NREP,NCALIB,IDXE,IREP,IEOFF
*
*         SET NO. REP FOR TEST TO CURRENT (IREP)
AND BRANCH TO
*          OUTPUT AND TERMINATION
*
          IPARM(2,ICURTS) = IREP - 1
          NUMTST = ICURTS + 1
*
*         SEE IF THIS TEST IS ONLY 1 REP
*         IF SO, IGNORE THE WHOLE TEST
*
          IF (IREP  EQ  1) NUMTST = ICURTS
          GOTO 20300

20110   CONTINUE
        ETIMES(1,IDXE) = STIME
        ETIMES(2,IDXE) = ETIME
*
*   END OF COMPUTATIONAL LOOP
*
20200 CONTINUE
*
*    BUMP OFFSET INDEX IN ETIMES ARRAY AND
INCREMENT INDEX TO NEXT
*    TEST IN IPARM ARRAY
*
      IEOFF = IEOFF + NREP
      ICURTS = ICURTS + 1
*
*  GO SEE IF MORE TESTS TO PERFORM
*
      GOTO 5
*
*   END OF PROGRAM
*
20300 CONTINUE
*
*    CALL OUTPUT SUBROUTINE TO WRITE STATISTICS
OF RUN
*
      CALL WRTDAT
*
20400 CONTINUE
*
*  FORMAT STATEMENTS
*
*
 9001 FORMAT(//,' *** ERROR - TEST TYPE UNKNOWN
***',//,
     1       12X,'TEST TYPE (ITYPE)  ',I7,/,
     2       12X,'NO. REPS  (NREP)   ',I7,/,
     3       12X,'CALIB  NO. (NCALIB)',I7 )
*
 9002 FORMAT('1',//,' *** ERROR - ATTEMPT TO
WRITE PAST BOUNDS OF',
     1       ' OUTPUT ARRAY (ETIMES) ***',/,'
FOLLOWING ARE',
     2       ' CONDITIONS OF THE RUN@D',/,'
ITYPE ',I4,/,
     3       '       NREP ',I4,/,'       NCALIB
',I7,//,
     4       '       IDXE ',I4,/,'       IREP
',I4,/,
     5       '      IEOFF ',I4,//,'  ** NO  OF
REPS IN TEST WILL BE',
     6       ' RESET, DATA WILL BE OUTPUT AND
JOB WILL BE TERMINATED')
*
*   NOW FIND FINAL CPU TIME AND PRINT
```

```
*
      CALL CPTIME (ENDTIM)
      ENDTIM = ENDTIM - BGNTIM
      WRITE (6, 1111) ENDTIM
1111  FORMAT ('0TOTAL CPU TIME USED BY ALL PARTS
OF THE BENCHMARK',/,
     ' WAS',F10.3,' SECONDS ')
      STOP
      END

      SUBROUTINE CPTIME (CPUTIM)
************************************************
**********************
*    RETURNS CPUTIM (REAL) AS THE NUMBER OF CPU
SECONDS THAT HAVE  *
*    ELAPSED FOR THIS TASK
                   *
*
                   *
*    THIS ROUTINE IS CODED FOR CYBER NOS/VE
                   *
************************************************
**********************
      REAL CPUTIM

      CPUTIM = SECOND ( )
      RETURN

      END
C     SUBROUTINE CPTIME (CPUTIME)
CC
CC  ELXSI SUPPLIED CPU TIMING ROUTINE
CC
C      INTEGER*8 OS$READCPUTIMER
C      REAL    CPUTIME
C      CPUTIME  =
DFLOAT(OS$READCPUTIMER())/40000000 D0
C      RETURN
C      END

C     SUBROUTINE CPTIME (CPUTIM)
************************************************
**********************
*    RETURNS CPUTIM (REAL) AS THE NUMBER OF CPU
SECONDS THAT HAVE  *
*    ELAPSED FOR THIS TASK
                   *
*
                   *
*    THIS ROUTINE IS CODED FOR CRAY SYSTEMS
                   *
************************************************
**********************
C     REAL CPUTIM
C
C     CPUTIM = SECOND (0 0)
C     RETURN
C
C     END

C     SUBROUTINE CPTIME (CPUTIM)
************************************************
**********************
*    RETURNS CPUTIM (REAL) AS THE NUMBER OF CPU
SECONDS THAT HAVE  *
*    ELAPSED FOR THIS TASK
                   *
*
                   *
*    THIS ROUTINE IS CODED FOR VAX/VMS SYSTEMS
                   *
************************************************
**********************
C     INTEGER IARG, ITIME
C     REAL CPUTIM
C     LOGICAL FIRST
C     DATA FIRST / TRUE /
C
************************************************
**********************
*    THE FIRST CALL TO TIME SHOULD INITIALIZE
THE TASK TIMER         *
*    BY CALLING LIB$INIT_TIMER, WHICH SETS ALL
THE VARIOUS TIMES      *
*    TO ZERO.  OTHERWISE, UNINITIALIZED TIMES
(E.G., NEGATIVE) COULD *
*    BE RETURNED
                   *
************************************************
**********************
C     IF ( NOT  FIRST) GO TO 1
C        CALL LIB$INIT_TIMER
C        FIRST =  FALSE
C
C1    IARG = 2
C     CALL LIB$STAT_TIMER (IARG, ITIME)
************************************************
**********************
*    WITH PARAMETER 2, THIS ROUTINE RETURNS CPU
TIME AS AN INTEGER   *
*    COUNT OF 10 MILLISECOND INTERVALS.
                   *
************************************************
**********************
C     CPUTIM = DBLE(ITIME)/100 D0
C     RETURN
C     END

C     SUBROUTINE CPTIME(CPUTIM)
************************************************
**********************
*    RETURNS CPUTIM (REAL) AS THE NUMBER OF CPU
SECONDS THAT HAVE  *
*    ELAPSED FOR THIS TASK
                   *
*
                   *
*  THIS SUBROUTINE IS WRITTEN FOR THE GOULD
32/87 WITH MPX 3 2       *
*  EXECUTING FORTRAN 4 0
                   *
*
                   *
************************************************
**********************
C     INTEGER NSEC, NCLICK
C     REAL CPUTIM
C
C      CALL M@DCLOCK (NSEC, NCLICK)
C      CPUTIM = FLOAT(NSEC)
C      CPUTIM = CPUTIM + FLOAT(NCLICK)/60
C      RETURN
C      END

C     SUBROUTINE CPTIME(TOTIME)
************************************************
**********************
*    RETURNS CPUTIM (REAL) AS THE NUMBER OF CPU
```

```
SECONDS THAT HAVE  *
*    ELAPSED FOR THIS TASK
                   *
*
                   *
*  THIS SUBROUTINE IS WRITTEN FOR THE HP 1000
RUNNING FORTRAN 77     *
*  NOTE THAT SINCE THE HP OPERATING SYSTEM HAS
NO FACILITY FOR        *
*  RETURNING CPU TIME, THIS ROUTINE ONLY
MEASURES ELAPSED TIME       *
*  THEREFORE, THE TESTS MUST BE RUN STAND-ALONE
IN ORDER TO MAKE      *
*  CPU TIME IDENTICAL TO ELAPSED TIME (THE
BENCHMARK DOES NO I/O     *
*  WHILE EXECUTING TESTS)
                   *
*
                   *
************************************************
**********************
C     REAL TOTIME
C     INTEGER*2 TIMEA(5)
C
C     CALL EXEC (11, TIMEA)
C     TOTIME = DBLE (TIMEA(1))/100
C     TOTIME = TOTIME + DBLE (TIMEA(2))
C     TOTIME = TOTIME + DBLE (TIMEA(3)) * 60
C     TOTIME = TOTIME + DBLE (TIMEA(4)) * 3600
C
C     RETURN
C     END

C     SUBROUTINE CPTIME(TOTIME)
************************************************
**********************
*    RETURNS CPUTIM (REAL) AS THE NUMBER OF CPU
SECONDS THAT HAVE  *
*    ELAPSED FOR THIS TASK
                   *
*
                   *
*  THIS SUBROUTINE IS WRITTEN FOR THE IBM PC
                   *
*  NOTE THAT THE PC'S OPERATING SYSTEM HAS NO
FACILITY FOR           *
*  RETURNING CPU TIME, THIS ROUTINE ONLY
MEASURES ELAPSED TIME       *
*  HOWEVER, THE PC DOES NOT USUALLY EXECUTE IN
MULTI-PROGRAMMING     *
*  MOCE, SO THAT'S OK   (THE BENCHMARK DOES NO
I/O WHILE EXECUTING   *
*  TESTS)
                   *
*  ALSO, THE PC DOES NOT HAVE REAL   THUS ALL
REFERENCES TO REAL *
*  MUST BE REMOVED
                   *
*
                   *
************************************************
**********************
C      IMPLICIT INTEGER (I-N)
C      LOGICAL FIRST
C      DATA FIRST / TRUE /
C
C      CALL GETTIM (IYEAR, IMONTH, IDAY, IHOUR,
IMIN, ISEC, IFRACT)
************************************************
**********************
*
                   *
*  IFRACT IS INTEGER FRACTIONS OF A SECOND
                   *
*  IN UNITS OF 1/32,768 SECONDS
                   *
*
                   *
************************************************
**********************
C      IF (.NOT. FIRST) GO TO 10
C        FIRST =  FALSE.
*
C        LASTHR = IHOUR
C        BASETM = 0.
C10    CONTINUE
*
*  BECAUSE OF LIMITED PRECISION, DO NOT INCLUDE
THE TIME OF DAY
*  IN HOURS IN THE TOTAL TIME   BUT CORRECT FOR
AN HOUR CHANGE.
*
C      IF (LASTHR  EQ. IHOUR) GO TO 20
C        BASETM = BASETM + 3600.
C        LASTHR = IHOUR
C
C20    TOTIME = FLOAT(IMIN) * 60
C    . + FLOAT(ISEC)
C      + FLOAT(IFRACT)/32768.
C      TOTIM = TOTIM + BASETM
C      RETURN
C      END

C     SUBROUTINE CPTIME(CPUTIM)
************************************************
**********************
*    RETURNS CPUTIM (REAL) AS THE NUMBER OF CPU
SECONDS THAT HAVE  *
*    ELAPSED FOR THIS TASK
                   *
*
                   *
*  THIS SUBROUTINE IS WRITTEN FOR THE PR1ME
SYSTEM UNDER PRIMOS      *
*
                   *
************************************************
**********************
C     REAL CPUTIM
C     INTEGER*2 TIMERS (28)
C
C     CALL TMDAT (TIMERS)
C     CPUTIM = DBLE (TIMERS(7))
C     + DBLE(TIMERS(8)) / DBLE(TIMERS(11))
C
C     RETURN
C     END

C      SUBROUTINE CPTIME (TIME)
C      IMPLICIT REAL (T)
C      LOGICAL FIRST
C      DATA FIRST / TRUE /
C*************************************************
**********************
C*
C*     M V S  TIMER ROUTINE (REQUIRES FOLLOWING
ASSEMBLER CODE)
C*
```

```
C*   THE MICROSECOND CLOCK ACCESSED BY THE
ROUTINE MVSTIM COUNTS DOWN
C*   FROM ABOUT 2 BILLION (2**31)   SO WHEN IT
GETS BELOW 1 MINUTE,
C*   RESET IT TO PREVENT UNDERFLOW
C*
C*   A CALL TO MVSTIM WITH AN ARGUMENT OF 0
RESETS THE CLOCK,
C*   ANY OTHER ARGUMENT CAUSES THE COUNTED-DOWN
CLOCK TO BE RETURNED
C*
C***********************************************
**********************
C      IF ( NOT  FIRST) GO TO 5
C        L = 0
C        CALL MVSTIM (L)
C        FIRST =  FALSE
C        TBASE = L
C        TBASE = TBASE/1 D6
C5     CONTINUE
C***********************************************
**********************
C*
C*   ALWAYS GET CURRENT TIME (EVEN IF FIRST
ENTRY)
C*
C***********************************************
**********************
C      CALL MVSTIM (L)
C      TEMP = L
C      TEMP = TEMP/1 D6
C      TIME = TBASE - TEMP
C      IF (L  GT  60000000) RETURN
C***********************************************
**********************
C*
C*   HERE WE RESET THE COUNTER IF LESS THAN ONE
MINUTE LEFT
C*
C***********************************************
**********************
C      L = 0
C      CALL MVSTIM (L)
C      TNEW = L
C      TNEW = TNEW/1 D6
C      TBASE = TBASE + TNEW - TEMP
C      RETURN
C
C      END
C
C FOR MVS SYSTEMS, REMOVE THESE 2 CARDS, MOVE
ALL THE FOLLOWING
C ASSEMBLER CODE 1 SPACE LEFT (TO REMOVE COLUMN
1 STUFF), AND ASSEMBLE
CMVSTIM   CSECT
C         PRINT GEN
C         USING *,12
C***********************************************
**********************
C* STANDARD SUBROUTINE LINKAGE FROM FORTRAN
PROGRAM                    *
C* GPR  1@D ARGUMENT LIST ADDRESS FROM CALLER
                           *
C* GPR 13@D CALLER'S SAVEAREA ADDRESS
                           *
C* GPR 14@D ADDRESS IN CALLER FOR RETURN
                           *
C* GPR 15@D ENTRY ADDRESS OF THIS SUBPROGRAM
                           *
C***********************************************
**********************
C* REGISTER USAGE IN THIS PROGRAM
                           *
C* GPR  0@D RETURN OF ELAPSED MICROSECONDS TO
FORTRAN                    *
C* GPR  1@D ARGUMENT LIST ADDRESS FROM CALLER
                           *
C* GPR  2@D ADDRESS OF PARAMETER PASSED FROM
FORTRAN                    *
C* GPR  3@D WORK REGISTER
                           *
C* GPR  4@D WORK REGISTER
                           *
C* GPR  5@D WORK REGISTER
                           *
C* GPR  6@D NOT USED
                           *
C* GPR  7@D NOT USED
                           *
C* GPR  8@D NOT USED
                           *
C* GPR  9@D NOT USED
                           *
C* GPR 10@D NOT USED
                           *
C* GPR 11@D NOT USED
                           *
C* GPR 12@D BASE REGISTER FOR THIS PROGRAM
                           *
C* GPR 13@D NOT USED
                           *
C* GPR 14@D NOT USED
                           *
C* GPR 15@D RETURN CODE TO FORTRAN IN CASE OF
ERROR IN PROCESSING        *
C***********************************************
**********************
C***********************************************
**********************
C*                      REGISTER EQUATES
                          *
C***********************************************
**********************
CR0       EQU   0
CR1       EQU   1
CR2       EQU   2
CR3       EQU   3
CR4       EQU   4
CR5       EQU   5
CR6       EQU   6
CR7       EQU   7
CR8       EQU   8
CR9       EQU   9
CR10      EQU   10
CR11      EQU   11
CR12      EQU   12
CR13      EQU   13
CR14      EQU   14
CR15      EQU   15
C***********************************************
**********************
C*              END REGISTER EQUATES
                          *
C***********************************************
**********************
C*              BEGIN BODY OF PROGRAM CODE
                          *
C***********************************************
```

```
************************
C         STM   R14,R12,12(R13)        STORE
CALLING PGM'S GPRS
C         LR    R12,R15                OUR
ENTRY POINT INTO BASE REG
C         LA    R15,SAVEAREA           THIS
PGM'S SAVEAREA ADDR IN 15
C         ST    R13,4(R15)             SAVE
CALLER'S SAVEAREA ADDR
C         ST    R15,8(R13)             SAVE OUR
SAVEAREA ADDR IN CALLER
C         LR    R13,R15                CURRENT
SAVEAREA ADDR IN 13
C         L     R2,0(R1)               ADDRESS
OF PASSED VARIABLE
C         SR    R3,R3                  ZERO OUT
R3
C         C     R3,0(R2)               IS
SUBROUTINE INVOKED WITH 0?
C         BE    SETIT                  YES, SET
INTERVAL TIMER
CGETIT    DS    0H                     NO, GET
INTERVAL TIMER VALUE
C         TTIMER ,MIC,TVAL2            GET TIME
INTO TVAL2
C         L     R4,TVAL2               LOAD,
GET READY FOR SHIFT
C         L     R5,TVAL2+4             LOAD,
GET READY FOR SHIFT
C         LA    R3,X'0C'               LOAD 12
TO REG 3
C         SRDL  R4,0(R3)               SHIFT
RIGHT 12 BITS
C         ST    R5,0(R2)               SAVE
ELAPSED TIME FOR CALLER
C         B     GETOUT                 OUR WORK
HERE IS DONE KEMOSABE
CSETIT    DS    0H                     HERE IS
WHERE WE SET THE I T
C         L     R4,HOURS1              PART 1
OF TIME VAL TO R3
C         ST    R4,TVAL                SET TVAL
PART 1
C         L     R5,HOURS2              PART 2
OF TIME VAL TO R3
C         ST    R5,TVAL+4              SET TVAL
PART 2
C         LA    R3,X'0C'               LOAD 12
TO REG 3
C         SRDL  R4,0(R3)               SHIFT
RIGHT 12 BITS
C         ST    R5,0(R2)               SAVE
START TIME FOR CALLER
C         STIMER TASK,MICVL=TVAL       SET
TIMER BASED IN MICROSECS
C         SR    R0,R0                  RETURN
DUMMY VALUE ON SET
C********************************************
************************
C*                    END OF PROGRAM
                    *
C********************************************
************************
CGETOUT   DS    0H
C         L     R13,4(R13)             LOAD
CALLER'S SAVEAREA ADDR TO 13
C         L     R14,12(R13)            LOAD
RETURN ADDR OF CALLER TO 14
C         LM    R1,R12,24(R13)         RESTORE
CALLER'S REGISTERS
C*                                     (BUT
DON'T RELOAD R0)
C         SR    R15,R15                SET
RETURN CODE TO FORTRAN
C         BR    R14                    RETURN TO
CALLER
C********************************************
************************
C*             DATA AREAS REQUIRED FOR
OPERATION                            *
C********************************************
************************
CALGN0    DS    0D           DOUBLEWORD
ALIGNMENT REQUIRED
CTVAL     DC    D'0'         TIMER VALUE
CTVAL2    DC    D'0'         TIMER VALUE 2 -
("NEW" VALUE)
CHOURS1   DC    X'000007FF'
CHOURS2   DC    X'FFFFF000'
CSAVEAREA DS    18F          18 WORD REGISTER
SAVE AREA
C         END

C     SUBROUTINE CPTIME(CPUTIM)
************************************************
**********************
*    RETURNS CPUTIM (REAL) AS THE NUMBER OF CPU
SECONDS THAT HAVE  *
*    ELAPSED FOR THIS TASK.
                    *
*
                    *
*  THIS SUBROUTINE WORKS FOR IBM VM SYSTEMS, AND
CALLS AN ASSEMBLER  *
*  ROUTINE 'VMTIME', WHOSE SOURCE FOLLOWS
                    *
*
                    *
************************************************
**********************
C     IMPLICIT INTEGER (I-N)
C     REAL CPUTIM
C     INTEGER VSEC, VUSEC
C
C     CALL VMTIME (VSEC, VUSEC)
C
C     CPUTIM = VSEC
C     CPUTIM = CPUTIM + VUSEC/1 D6
C     RETURN
C
C     END

************************************************
**********************
*
                    *
*    THIS PROGRAM IS A FORTRAN CALLABLE ROUTINE
THAT RETURNS A      *
*    RESULT OF THE DIAGNOSE X'C' INSTRUCTION.
                    *
*
                    *
*    FORTRAN CALL@D  CALL VMTIME(VSEC, VUSEC)
                     *
*
                    *
*
                    *
```

```
*    WHERE@D
                    *
*           VSEC   THE VIRTUAL CPU TIME, THE
SECONDS PORTION (I*4)   *
*           VUSEC  THE VIRTUAL CPU TIME, THE
MICRO-SECOND PORTION    *
*

^E*NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN**
************************************************
*********************
*
*VMTIME   START 0
*BEGIN    SAVE  (14,12)
*         BALR  3,0
*         USING *,3
*         ST    13,SAVE+4
*         LA    13,SAVE
*
*         LM    4,5,0(1)          LOAD THE
FORTRAN PLIST INTO REGS
*
*         LA    2,PTIMER
*
*         DIAG  2,0,X'C'
*
*         L     6,VCPU
*         L     7,VCPU+4
*         D     6,=F'1000000'     SEPARATE SEC
AND USEC
*         ST    7,0(4)            STORE THE
SECOND PORTION
*         ST    6,0(5)            STORE THE
MICROSECOND PORTION
*
*
*         L     13,SAVE+4
*         RETURN (14,12),RC=0
*
*
*PTIMER   DS    0D
*DATE     DS    D                 CURRENT DATE
*TIME     DS    D                 CURRENT TIME
*VCPU     DS    D                 VIRTUAL CPU
TIME
*TCPU     DS    D                 TOTAL CPUTIME
*SAVE     DS    18F
*
*         LTORG
*         END

      SUBROUTINE WRTDAT
      IMPLICIT INTEGER (I-N)
      REAL ETIMES, ETM, TOTETM, AVGTIM, AVGNRM,
ALLCPU, SUMSQ
*
*    OUTPUT ALL DATA THAT WERE COLLECTED FOR THE
RUN (THE USER
*      SHOULD HAVE BEEN NOTIFIED IN THE MAIN
PROGRAM IF THE
*      NUMBER OF TESTS OVERFLOWED THE ARRAYS)

      COMMON /MAXTST/ MAXTST
      COMMON /MINTST/ MINTST

      COMMON /RDPARM/ IPARM (3,150), NUMTST,
ICURTS

      COMMON /ODATA/ ETIMES (2, 1500)

      DATA ALLCPU /0 /
*
      IEOFF = 0
*
*    RETURN IF NOTHING TO DO
*
      IF (NUMTST .LE  0) RETURN
*
*    LOOP ON EACH TEST CASE EXECUTED
*
      DO 500 IWLA = 1, NUMTST
*
*    SET PARAMETERS OF THIS TEST
*
        ITYPE  = IPARM(1,IWLA)
        NREP   = IPARM(2,IWLA)
        NCALIB = IPARM(3,IWLA)
*
*       WRITE OUT HEADER OF TEST AND INITILIZE
TOTAL SEC. COUNTER
*
        I = ITYPE - MINTST + 1
        GOTO (

1001,1002,1003,1004,1005,1006,1007,1008,1009,
     .
1010,1011,1012,1013,1014,1015,1016,1017,1018,101
9,

1020,1021,1022,1023,1024,1025,1026,1027,1028,102
9

,1030,1031,1032,1033,1034,1035,1036,1037,1038,10
39

,1040,1041,1042,1043,1044,1045,1046,1047,1048,10
49

,1050,1051,1052,1053,1054,1055,1056,1057,1058,10
59

,1060,1061,1062,1063,1064,1065,1066,1067,1068,10
69

,1070,1071,1072,1073,1074,1075,1076,1077,1078,10
79

,1080,1081,1082,1083,1084,1085,1086,1087,1088,10
89

,1090,1091,1092,1093,1094,1095,1096,1097,1098,10
99

,2100,2101,2102,2103,2104,2105,2106,2107,2108,21
09

,2110,2111,2112,2113,2114,2115,2116,2117,2118,21
19
     .
,2120,2121,2122,2123,2124,2125,2126,2127,2128
      ),I

1001    WRITE (6,1101)
1101    FORMAT('120 R*8 VECTOR ASSIGNMENTS OF
LENGTH 1')
        GO TO 2000

1002    WRITE (6,1102)
1102    FORMAT('140 R*8 VECTOR ASSIGNMENTS OF
```

```
LENGTH 1')
        GO TO 2000

1003    WRITE (6,1103)
1103    FORMAT('120 R*8 VECTOR ASSIGNMENTS OF
LENGTH 5')
        GO TO 2000

1004    WRITE (6,1104)
1104    FORMAT('140 R*8 VECTOR ASSIGNMENTS OF
LENGTH 5')
        GO TO 2000

1005    WRITE (6,1105)
1105    FORMAT('120 R*8 VECTOR ASSIGNMENTS OF
LENGTH 10')
        GO TO 2000

1006    WRITE (6,1106)
1106    FORMAT('140 R*8 VECTOR ASSIGNMENTS OF
LENGTH 10')
        GO TO 2000

1007    WRITE (6,1107)
1107    FORMAT('120 R*8 VECTOR ASSIGNMENTS OF
LENGTH 20')
        GO TO 2000

1008    WRITE (6,1108)
1108    FORMAT('140 R*8 VECTOR ASSIGNMENTS OF
LENGTH 20')
        GO TO 2000

1009    WRITE (6,1109)
1109    FORMAT('120 R*8 VECTOR ASSIGNMENTS OF
LENGTH 50')
        GO TO 2000

1010    WRITE (6,1110)
1110    FORMAT('140 R*8 VECTOR ASSIGNMENTS OF
LENGTH 50')
        GO TO 2000

1011    WRITE (6,1111)
1111    FORMAT('120 R*8 VECTOR ASSIGNMENTS OF
LENGTH 100')
        GO TO 2000

1012    WRITE (6,1112)
1112    FORMAT('140 R*8 VECTOR ASSIGNMENTS OF
LENGTH 100')
        GO TO 2000

1013    WRITE (6,1113)
1113    FORMAT('120 R*8 VECTOR ASSIGNMENTS OF
LENGTH 200')
        GO TO 2000

1014    WRITE (6,1114)
1114    FORMAT('140 R*8 VECTOR ASSIGNMENTS OF
LENGTH 200')
        GO TO 2000

1015    WRITE (6,1115)
1115    FORMAT('120 R*8 VECTOR ASSIGNMENTS OF
LENGTH 500')
        GO TO 2000

1016    WRITE (6,1116)
1116    FORMAT('140 R*8 VECTOR ASSIGNMENTS OF
LENGTH 500')
        GO TO 2000

1017    WRITE (6,1117)
1117    FORMAT('120 R*8 DOT PRODUCTS OF LENGTH
1')
        GO TO 2000

1018    WRITE (6,1118)
1118    FORMAT('140 R*8 DOT PRODUCTS OF LENGTH
1')
        GO TO 2000

1019    WRITE (6,1119)
1119    FORMAT('120 R*8 DOT PRODUCTS OF LENGTH
5')
        GO TO 2000

1020    WRITE (6,1120)
1120    FORMAT('140 R*8 DOT PRODUCTS OF LENGTH
5')
        GO TO 2000

1021    WRITE (6,1121)
1121    FORMAT('120 R*8 DOT PRODUCTS OF LENGTH
10')
        GO TO 2000

1022    WRITE (6,1122)
1122    FORMAT('140 R*8 DOT PRODUCTS OF LENGTH
10')
        GO TO 2000

1023    WRITE (6,1123)
1123    FORMAT('120 R*8 DOT PRODUCTS OF LENGTH
20')
        GO TO 2000

1024    WRITE (6,1124)
1124    FORMAT('140 R*8 DOT PRODUCTS OF LENGTH
20')
        GO TO 2000

1025    WRITE (6,1125)
1125    FORMAT('120 R*8 DOT PRODUCTS OF LENGTH
50')
        GO TO 2000

1026    WRITE (6,1126)
1126    FORMAT('140 R*8 DOT PRODUCTS OF LENGTH
50')
        GO TO 2000

1027    WRITE (6,1127)
1127    FORMAT('120 R*8 DOT PRODUCTS OF LENGTH
100')
        GO TO 2000

1028    WRITE (6,1128)
1128    FORMAT('140 R*8 DOT PRODUCTS OF LENGTH
100')
        GO TO 2000

1029    WRITE (6,1129)
1129    FORMAT('120 R*8 DOT PRODUCTS OF LENGTH
200')
        GO TO 2000
```

```
1030    WRITE (6,1130)
1130    FORMAT('140 R*8 DOT PRODUCTS OF LENGTH
200')
        GO TO 2000

1031    WRITE (6,1131)
1131    FORMAT('120 R*8 DOT PRODUCTS OF LENGTH
500')
        GO TO 2000

1032    WRITE (6,1132)
1132    FORMAT('140 R*8 DOT PRODUCTS OF LENGTH
500')
        GO TO 2000

1033    WRITE (6,1133)
1133    FORMAT('120 R*8 S*V + V OPS, LENGTH 1')
        GO TO 2000

1034    WRITE (6,1134)
1134    FORMAT('140 R*8 S*V + V OPS, LENGTH 1')
        GO TO 2000

1035    WRITE (6,1135)
1135    FORMAT('120 R*8 S*V + V OPS, LENGTH 5')
        GO TO 2000

1036    WRITE (6,1136)
1136    FORMAT('140 R*8 S*V + V OPS, LENGTH 5')
        GO TO 2000

1037    WRITE (6,1137)
1137    FORMAT('120 R*8 S*V + V OPS, LENGTH 10')
        GO TO 2000

1038    WRITE (6,1138)
1138    FORMAT('140 R*8 S*V + V OPS, LENGTH 10')
        GO TO 2000

1039    WRITE (6,1139)
1139    FORMAT('120 R*8 S*V + V OPS, LENGTH 20')
        GO TO 2000

1040    WRITE (6,1140)
1140    FORMAT('140 R*8 S*V + V OPS, LENGTH 20')
        GO TO 2000

1041    WRITE (6,1141)
1141    FORMAT('120 R*8 S*V + V OPS, LENGTH 50')
        GO TO 2000

1042    WRITE (6,1142)
1142    FORMAT('140 R*8 S*V + V OPS, LENGTH 50')
        GO TO 2000

1043    WRITE (6,1143)
1143    FORMAT('120 R*8 S*V + V OPS, LENGTH
100')
        GO TO 2000

1044    WRITE (6,1144)
1144    FORMAT('140 R*8 S*V + V OPS, LENGTH
100')
        GO TO 2000

1045    WRITE (6,1145)
1145    FORMAT('120 R*8 S*V + V OPS, LENGTH
200')
        GO TO 2000

1046    WRITE (6,1146)
1146    FORMAT('140 R*8 S*V + V OPS, LENGTH
200')
        GO TO 2000

1047    WRITE (6,1147)
1147    FORMAT('120 R*8 S*V + V OPS, LENGTH
500')
        GO TO 2000

1048    WRITE (6,1148)
1148    FORMAT('140 R*8 S*V + V OPS, LENGTH
500')
        GO TO 2000

1049    WRITE (6,1149)
1149    FORMAT('120 VECTOR MULTIPLIES OF LENGTH
1')
        GO TO 2000

1050    WRITE (6,1150)
1150    FORMAT('140 VECTOR MULTIPLIES OF LENGTH
1')
        GO TO 2000

1051    WRITE (6,1151)
1151    FORMAT('120 VECTOR MULTIPLIES OF LENGTH
5')
        GO TO 2000

1052    WRITE (6,1152)
1152    FORMAT('140 VECTOR MULTIPLIES OF LENGTH
5')
        GO TO 2000

1053    WRITE (6,1153)
1153    FORMAT('120 VECTOR MULTIPLIES OF LENGTH
10')
        GO TO 2000

1054    WRITE (6,1154)
1154    FORMAT('140 VECTOR MULTIPLIES OF LENGTH
10')
        GO TO 2000

1055    WRITE (6,1155)
1155    FORMAT('120 VECTOR MULTIPLIES OF LENGTH
20')
        GO TO 2000

1056    WRITE (6,1156)
1156    FORMAT('140 VECTOR MULTIPLIES OF LENGTH
20')
        GO TO 2000

1057    WRITE (6,1157)
1157    FORMAT('120 VECTOR MULTIPLIES OF LENGTH
50')
        GO TO 2000

1058    WRITE (6,1158)
1158    FORMAT('140 VECTOR MULTIPLIES OF LENGTH
50')
        GO TO 2000

1059    WRITE (6,1159)
1159    FORMAT('120 VECTOR MULTIPLIES OF LENGTH
100')
        GO TO 2000
```

```
1060    WRITE (6,1160)
1160    FORMAT('140 VECTOR MULTIPLIES OF LENGTH
100')
        GO TO 2000

1061    WRITE (6,1161)
1161    FORMAT('120 VECTOR MULTIPLIES OF LENGTH
200')
        GO TO 2000

1062    WRITE (6,1162)
1162    FORMAT('140 VECTOR MULTIPLIES OF LENGTH
200')
        GO TO 2000

1063    WRITE (6,1163)
1163    FORMAT('120 VECTOR MULTIPLIES OF LENGTH
500')
        GO TO 2000

1064    WRITE (6,1164)
1164    FORMAT('140 VECTOR MULTIPLIES OF LENGTH
500')
        GO TO 2000

1065    WRITE (6,1165)
1165    FORMAT('120X(ADD UP A VECTOR OF LENGTH
1)')
        GO TO 2000

1066    WRITE (6,1166)
1166    FORMAT('140X(ADD UP A VECTOR OF LENGTH
1)')
        GO TO 2000

1067    WRITE (6,1167)
1167    FORMAT('120X(ADD UP A VECTOR OF LENGTH
5)')
        GO TO 2000

1068    WRITE (6,1168)
1168    FORMAT('140X(ADD UP A VECTOR OF LENGTH
5)')
        GO TO 2000

1069    WRITE (6,1169)
1169    FORMAT('120X(ADD UP A VECTOR OF LENGTH
10)')
        GO TO 2000

1070    WRITE (6,1170)
1170    FORMAT('140X(ADD UP A VECTOR OF LENGTH
10)')
        GO TO 2000

1071    WRITE (6,1171)
1171    FORMAT('120X(ADD UP A VECTOR OF LENGTH
20)')
        GO TO 2000

1072    WRITE (6,1172)
1172    FORMAT('140X(ADD UP A VECTOR OF LENGTH
20)')
        GO TO 2000

1073    WRITE (6,1173)
1173    FORMAT('120X(ADD UP A VECTOR OF LENGTH
50)')
        GO TO 2000

1074    WRITE (6,1174)
1174    FORMAT('140X(ADD UP A VECTOR OF LENGTH
50)')
        GO TO 2000

1075    WRITE (6,1175)
1175    FORMAT('120X(ADD UP A VECTOR OF LENGTH
100)')
        GO TO 2000

1076    WRITE (6,1176)
1176    FORMAT('140X(ADD UP A VECTOR OF LENGTH
100)')
        GO TO 2000

1077    WRITE (6,1177)
1177    FORMAT('120X(ADD UP A VECTOR OF LENGTH
200)')
        GO TO 2000

1078    WRITE (6,1178)
1178    FORMAT('140X(ADD UP A VECTOR OF LENGTH
200)')
        GO TO 2000

1079    WRITE (6,1179)
1179    FORMAT('120X(ADD UP A VECTOR OF LENGTH
500)')
        GO TO 2000

1080    WRITE (6,1180)
1180    FORMAT('140X(ADD UP A VECTOR OF LENGTH
500)')
        GO TO 2000

1081    WRITE (6,1181)
1181    FORMAT('120 VECTOR ADDS, LENGTH 1')
        GO TO 2000

1082    WRITE (6,1182)
1182    FORMAT('140 VECTOR ADDS, LENGTH 1')
        GO TO 2000

1083    WRITE (6,1183)
1183    FORMAT('120 VECTOR ADDS, LENGTH 5')
        GO TO 2000

1084    WRITE (6,1184)
1184    FORMAT('140 VECTOR ADDS, LENGTH 5')
        GO TO 2000

1085    WRITE (6,1185)
1185    FORMAT('120 VECTOR ADDS, LENGTH 10')
        GO TO 2000

1086    WRITE (6,1186)
1186    FORMAT('140 VECTOR ADDS, LENGTH 10')
        GO TO 2000

1087    WRITE (6,1187)
1187    FORMAT('120 VECTOR ADDS, LENGTH 20')
        GO TO 2000

1088    WRITE (6,1188)
1188    FORMAT('140 VECTOR ADDS, LENGTH 20')
        GO TO 2000

1089    WRITE (6,1189)
1189    FORMAT('120 VECTOR ADDS, LENGTH 50')
```

```
      GO TO 2000

1090  WRITE (6,1190)
1190  FORMAT('140 VECTOR ADDS, LENGTH 50')
      GO TO 2000

1091  WRITE (6,1191)
1191  FORMAT('120 VECTOR ADDS, LENGTH 100')
      GO TO 2000

1092  WRITE (6,1192)
1192  FORMAT('140 VECTOR ADDS, LENGTH 100')
      GO TO 2000

1093  WRITE (6,1193)
1193  FORMAT('120 VECTOR ADDS, LENGTH 200')
      GO TO 2000

1094  WRITE (6,1194)
1194  FORMAT('140 VECTOR ADDS, LENGTH 200')
      GO TO 2000

1095  WRITE (6,1195)
1195  FORMAT('120 VECTOR ADDS, LENGTH 500')
      GO TO 2000

1096  WRITE (6,1196)
1196  FORMAT('140 VECTOR ADDS, LENGTH 500')
      GO TO 2000

1097  WRITE (6,1197)
1197  FORMAT('120 VECTOR DIVIDES, LENGTH 1')
      GO TO 2000

1098  WRITE (6,1198)
1198  FORMAT('140 VECTOR DIVIDES, LENGTH 1')
      GO TO 2000

1099  WRITE (6,1199)
1199  FORMAT('120 VECTOR DIVIDES, LENGTH 5')
      GO TO 2000

2100  WRITE (6,11100)
11100 FORMAT('140 VECTOR DIVIDES, LENGTH 5')
      GO TO 2000

2101  WRITE (6,11101)
11101 FORMAT('120 VECTOR DIVIDES, LENGTH 10')
      GO TO 2000

2102  WRITE (6,11102)
11102 FORMAT('140 VECTOR DIVIDES, LENGTH 10')
      GO TO 2000

2103  WRITE (6,11103)
11103 FORMAT('120 VECTOR DIVIDES, LENGTH 20')
      GO TO 2000

2104  WRITE (6,11104)
11104 FORMAT('140 VECTOR DIVIDES, LENGTH 20')
      GO TO 2000

2105  WRITE (6,11105)
11105 FORMAT('120 VECTOR DIVIDES, LENGTH 50')
      GO TO 2000

2106  WRITE (6,11106)
11106 FORMAT('140 VECTOR DIVIDES, LENGTH 50')
      GO TO 2000

2107  WRITE (6,11107)
11107 FORMAT('120 VECTOR DIVIDES, LENGTH 100')
      GO TO 2000

2108  WRITE (6,11108)
11108 FORMAT('140 VECTOR DIVIDES, LENGTH 100')
      GO TO 2000

2109  WRITE (6,11109)
11109 FORMAT('120 VECTOR DIVIDES, LENGTH 200')
      GO TO 2000

2110  WRITE (6,11110)
11110 FORMAT('140 VECTOR DIVIDES, LENGTH 200')
      GO TO 2000

2111  WRITE (6,11111)
11111 FORMAT('120 VECTOR DIVIDES, LENGTH 500')
      GO TO 2000

2112  WRITE (6,11112)
11112 FORMAT('140 VECTOR DIVIDES, LENGTH 500')
      GO TO 2000

2113  WRITE (6,11113)
11113 FORMAT('120 VECTOR +, *, +, *, LENGTH
1')
      GO TO 2000

2114  WRITE (6,11114)
11114 FORMAT('140 VECTOR +, *, +, *, LENGTH
1')
      GO TO 2000

2115  WRITE (6,11115)
11115 FORMAT('120 VECTOR +, *, +, *, LENGTH
5')
      GO TO 2000

2116  WRITE (6,11116)
11116 FORMAT('140 VECTOR +, *, +, *, LENGTH
5')
      GO TO 2000

2117  WRITE (6,11117)
11117 FORMAT('120 VECTOR +, *, +, *, LENGTH
10')
      GO TO 2000

2118  WRITE (6,11118)
11118 FORMAT('140 VECTOR +, *, +, *, LENGTH
10')
      GO TO 2000

2119  WRITE (6,11119)
11119 FORMAT('120 VECTOR +, *, +, *, LENGTH
20')
      GO TO 2000

2120  WRITE (6,11120)
11120 FORMAT('140 VECTOR +, *, +, *, LENGTH
20')
      GO TO 2000

2121  WRITE (6,11121)
11121 FORMAT('120 VECTOR +, *, +, *, LENGTH
50')
      GO TO 2000
```

```
2122    WRITE (6,11122)
11122   FORMAT('140 VECTOR +, *, +, *, LENGTH
50')
        GO TO 2000

2123    WRITE (6,11123)
11123   FORMAT('120 VECTOR +, *, +, *, LENGTH
100')
        GO TO 2000

2124    WRITE (6,11124)
11124   FORMAT('140 VECTOR +, *, +, *, LENGTH
100')
        GO TO 2000

2125    WRITE (6,11125)
11125   FORMAT('120 VECTOR +, *, +, *, LENGTH
200')
        GO TO 2000

2126    WRITE (6,11126)
11126   FORMAT('140 VECTOR +, *, +, *, LENGTH
200')
        GO TO 2000

2127    WRITE (6,11127)
11127   FORMAT('120 VECTOR +, *, +, *, LENGTH
500')
        GO TO 2000

2128    WRITE (6,11128)
11128   FORMAT('140 VECTOR +, *, +, *, LENGTH
500')
        GO TO 2000

2000    WRITE (6,9001) ITYPE,NREP,NCALIB
 9001 FORMAT(//,
     1        5X,'INPUTS TO RUN - ',/,
     2        7X,'TEST TYPE (ITYPE)  ',I7,/,
     3        7X,'NO  REPS  (NREP)   ',I7,/,
     4        7X,'CALIB  NO  (NCALIB)',I7,
     5       ///,'  REP NO ',5X,'  CPU
TIME',7X,'START TIME',
     6       5X,'  END TIME')
*
*       LOOP ON EACH REPETITION OF THE TEST CASE
AND OUTPUT
*
        TOTETM = 0.
        SUMSQ  = 0

        DO 400 IWLB = 1, NREP
          IDXE = IWLB + IEOFF
          ETM = ETIMES(2,IDXE) - ETIMES(1,IDXE)
          TOTETM = TOTETM + ETM
          SUMSQ  = SUMSQ  + ETM**2
          ALLCPU = ALLCPU + ETM
          WRITE(6,9002)
IWLB,ETM,ETIMES(1,IDXE),ETIMES(2,IDXE)
 9002 FORMAT(3X,I5,5X,F10 3,7X,F10 3,5X,F10 3)
 400    CONTINUE
*
*       OUTPUT TOTAL SECOND COUNTER, BUMP OFFSET
COUNTER, AND LOOP
*         BACK TO OUTPUT NEXT TEST
*
        AVGTIM = TOTETM/NREP
        V      = SUMSQ/NREP - AVGTIM**2
        IF (V  LE  0 ) STDEV = 0
        IF (V .GT. 0.) STDEV = SQRT(V)
        IF (AVGTIM .LE  0) STPCT = 0
        IF (AVGTIM  GT. 0) STPCT = 100  *
STDEV/AVGTIM
        AVGNRM = AVGTIM/NCALIB*100000
        WRITE(6,9003) AVGTIM, STDEV, STPCT,
AVGNRM
9003    FORMAT('0  AVERAGE',3X,F10 3,/,
        ' STANDARD DEVIATION',F10 3,' =',F6 1,'
PER CENT',/,
        '0IF NCALIB WERE 100,000@D ',F15 3)
*
*   WRITE OUT TEST NUMBER AND 100,000 VALUE, AND
% STD  DEV.
*   IN MACHINE-READABLE FORM
*
        WRITE (6, 9004) AVGNRM, ITYPE, STPCT
' WAS WRITE (7
9004    FORMAT (F15 3, I4, F20 3, '%')

        IEOFF = IEOFF + NREP
 500  CONTINUE
      WRITE (6, 9999) ALLCPU
9999  FORMAT ('1SUM OF ALL MEASURED CPU TIMES
WAS',F10 3,' SECONDS ')
      RETURN
      END

      SUBROUTINE RDCARD
      IMPLICIT INTEGER (I-N)
      CHARACTER*71 LABEL
*
*  SUBROUTINE TO READ A SINGLE 'CARD' IMMAGE
THAT SPECIFIES THE
*    CONDITIONS OF THE TEST TO BE EXECUTED@D
*  THE INTERFACE TO THIS PROGRAM IS A DISK FILE
WITH 1
*    TO 'N' TEST CASES SPECIFIED   EACH LINE IN
THE FILE MUST HAVE THE
*    FOLLOWING FORMAT@D
*
C234567890123456789012345678901234567890123456789012345678
901234567890
*ITYPE=III NREP=III NCALIB=IIIIIII
*
      COMMON /PARMS/ ITYPE, NREP, NCALIB
      COMMON /RDPARM/ IPARM(3,150),NUMTST,ICURTS
      COMMON /MINTST/ MINTST
      COMMON /MAXTST/ MAXTST

      INTEGER INTPA(3,100)

      ITMPR = 0
*
*    INITILIZE POINTER TO INPUT PARAMETER ARRAY
*
      IPTR = 0
*
*   FIRST READ A LABEL LINE AND TRANSFER IT TO
THE OUTPUT AND
*   SUMMARY FILES
*
      READ (15, 9000, IOSTAT=IOS) LABEL
9000  FORMAT (A71)
      IF (IOS  EQ. 0) GO TO 5
      WRITE (6, 6)
6     FORMAT(' INPUT FILE EMPTY, RUN ABORTED ')
      STOP
```

```
5     WRITE (6, 8) LABEL
8     FORMAT ('1BENCHMARK TESTS FOR@D',/,1X,A71)
      WRITE (6, 8999) LABEL             'WAS
WRITE (7
8999  FORMAT (' ',A71)
*
*    LOOP ON EACH INPUT CARD IN FILE
*
 10   CONTINUE
*
*    READ IMAGE
*
      READ(15,9001,IOSTAT=IOS) ITMPT,ITMPR,ITMPC
*
*    BRANCH IF NO FILE ERRORS OR END OF FILE
*
      IF ((IOS EQ 0)  AND  (ITMPR GT 0)) GOTO
100
*
*      TEST IF END OF FILE AND NO PARAMETERS IN
*        (NOTE@D  LOGIC WILL TAKE INTO
CONSIDERATION AN INPUT
*        FILE THAT HAS DATA IN WHICH AN I/O
ERROR OCCURS)
*
      IF (IPTR  GT  0) GOTO 200
*
*     NOTIFY USER THAT DISK INPUT FILE NOT USED
*
15    WRITE(6,9003)
9003  FORMAT(' NO TESTS IN INPUT FILE, RUN
TERMINATED ')
      STOP
*
 100  CONTINUE
      IPTR = IPTR + 1
*
*     TEST TO INSURE NOT EXCEEDING CURRENT
MAXIMUM OF IPARM
*     ARRAY (NOTE@D  TEST IS HARDWIRED)
*
      IF (IPTR  LE  150) GOTO 110
*
*       NOTIFY AND CONTINUE WITH THE TESTS
*
        WRITE(6,9002) IPTR
        IPTR = IPTR - 1
        GOTO 200
 110  CONTINUE
      IPARM(1,IPTR) = ITMPT
      IPARM(2,IPTR) = ITMPR
      IPARM(3,IPTR) = ITMPC
      GOTO 10
*
 200  CONTINUE
      ICURTS = 1
      NUMTST = IPTR

 300  CONTINUE
*
*    FORMAT STATEMENTS@D
*
 9001 FORMAT(7X,I3,6X,I3,8X,I7)
*
 9002 FORMAT('1',//,' ** NUMBER OF TESTS EXCEEDS
THE DIMENSION OF',
     1       ' THE IPARM ARRAY  **',/,' POINTER
VALUE IS',I4)
*
      RETURN
      END

      BLOCK DATA BLKDATA

      IMPLICIT REAL (V)

      COMMON /MAXTST/ MAXTST
      COMMON /MINTST/ MINTST
      DATA MINTST /   1 /
      DATA MAXTST / 128 /
      END
      SUBROUTINE ASGN20 (LENGTH)
************************************************
**********************
*     20 VECTOR ASSIGNMENT STATEMENTS
                *
*     VECTORS ARE ALL OF TYPE REAL
              *
*       THIS ROUTINE PERFORMS 20 VECTOR
ASSIGNMENTS OF LENGTH        *
*       'LENGTH'
                *
*       THE MAIN PROGRAM CALLS THIS ROUTINE
SEVERAL TIMES WITH       *
*     DIFFERENT VECTOR LENGTHS.
                *
************************************************
**********************
      IMPLICIT REAL (V)

      COMMON /VECTOR/ V00(200000)

      COMMON /VVECT/ VV00(200000)

      DO 100 LOOP = 1, 20
         DO 1 I = 1, LENGTH
            VV00(I) = V00(I)
1        CONTINUE
100   CONTINUE

      RETURN
      END

      SUBROUTINE ASGN40 (LENGTH)
************************************************
**********************
*     JUST LIKE PREVIOUS SUBROUTINE, BUT WITH 40
                *
*     ASSIGNMENTS
                *
************************************************
**********************
      IMPLICIT REAL (V)

      COMMON /VECTOR/ V00(200000)

      COMMON /VVECT/ VV00(200000)

      DO 100 LOOP = 1, 40
         DO 1 I = 1, LENGTH
            VV00(I) = V00(I)
1        CONTINUE
100   CONTINUE


      RETURN
      END
```

```
      SUBROUTINE DOT20 (LENGTH)
************************************************
**********************
*     20 VECTOR DOT PRODUCTS
                  *
*     VECTORS ARE ALL OF TYPE REAL
                *
*       THIS ROUTINE PERFORMS 20 VECTOR DOT
PRODUCTS OF LENGTH       *
*       'LENGTH'
                  *
*       THE MAIN PROGRAM CALLS THIS ROUTINE
SEVERAL TIMES WITH       *
*     DIFFERENT VECTOR LENGTHS
                  *
************************************************
**********************
      IMPLICIT REAL (V)

      COMMON /VECTOR/ V00(200000)

      COMMON /VVECT/ VV00(200000)

      COMMON /RESULT/ VANS00

      DO 100 LOOP = 1, 20
         VANS00 = 0
         DO 1 I = 1, LENGTH
            VANS00 = VANS00 + V00(I)*VV00(I)
1        CONTINUE
100   CONTINUE

      RETURN
      END

      SUBROUTINE DOT40 (LENGTH)
************************************************
**********************
*     JUST LIKE DOT20, BUT 40 DOT PRODUCTS
                  *
*     VECTORS ARE ALL OF TYPE REAL
                *
************************************************
**********************
      IMPLICIT REAL (V)

      COMMON /VECTOR/ V00(200000)

      COMMON /VVECT/ VV00(200000)

      COMMON /RESULT/ VANS00

      DO 100 LOOP = 1, 40
         VANS00 = 0
         DO 1 I = 1, LENGTH
            VANS00 = VANS00 + V00(I)*VV00(I)
1        CONTINUE
100   CONTINUE

      RETURN
      END

      SUBROUTINE VSVV20 (LENGTH)
************************************************
**********************
*     20 VECTOR ADDS WITH SCALAR MULTIPLY, LIKE
V1 = S*V2 + V1       *
*     VECTORS ARE ALL OF TYPE REAL
                *
*       THIS ROUTINE PERFORMS 20 SUCH
OPERATIONS.                  *
*       THE MAIN PROGRAM CALLS THIS ROUTINE
SEVERAL TIMES WITH       *
*     DIFFERENT VECTOR LENGTHS.
                  *
************************************************
**********************
      IMPLICIT REAL (V)

      COMMON /VECTOR/ V00(200000)

      COMMON /VVECT/ VV00(200000)

      DATA VMULT /1 1 D0/

      DO 100 LOOP = 1, 20
         DO  1 I = 1, LENGTH
             VV00(I) = VMULT*V00(I) + VV00(I)
1        CONTINUE
100   CONTINUE

      RETURN
      END

      SUBROUTINE VSVV40 (LENGTH)
************************************************
**********************
*     JUST LIKE VSVV20, BUT WITH 40 OPERATIONS
                  *
*       THE MAIN PROGRAM CALLS THIS ROUTINE
SEVERAL TIMES WITH       *
*     DIFFERENT VECTOR LENGTHS
                  *
************************************************
**********************
      IMPLICIT REAL (V)

      COMMON /VECTOR/ V00(200000)

      COMMON /VVECT/ VV00(200000)

      DATA VMULT /1 1 D0/

      DO 100 LOOP = 1, 40
         DO  1 I = 1, LENGTH
             VV00(I) = VMULT*V00(I) + VV00(I)
1        CONTINUE
100   CONTINUE

      RETURN
      END

      SUBROUTINE VMLT20 (LENGTH)
************************************************
**********************
*     20 VECTOR MULTIPLIES (LIKE V1 = V2*V3)
                  *
*     VECTORS ARE ALL OF TYPE REAL
                *
*       THE MAIN PROGRAM CALLS THIS ROUTINE
SEVERAL TIMES WITH       *
*     DIFFERENT VECTOR LENGTHS.
                  *
************************************************
**********************
      IMPLICIT REAL (V)

      COMMON /VECTOR/ V00(200000)
```

```
      COMMON /VVECT/ VV00(200000)

      COMMON /VRESLT/ VA00(200000)

      DO 100 LOOP = 1, 20
         DO 1 I = 1, LENGTH
            VA00(I) = V00(I)*VV00(I)
1        CONTINUE
100   CONTINUE

      RETURN
      END

      SUBROUTINE VMLT40 (LENGTH)
************************************************
**********************
*     JUST LIKE VMULT20, BUT 40 MULTIPLIES
                  *
*     VECTORS ARE ALL OF TYPE REAL
                *
************************************************
**********************
      IMPLICIT REAL (V)

      COMMON /VECTOR/ V00(200000)

      COMMON /VVECT/ VV00(200000)

      COMMON /VRESLT/ VA00(200000)

      DO 100 LOOP = 1, 40
         DO 1 I = 1, LENGTH
            VA00(I) = V00(I)*VV00(I)
1        CONTINUE
100   CONTINUE

      RETURN
      END

      SUBROUTINE VACC20 (LENGTH)
************************************************
**********************
*     20 VECTOR ACCUMULATES (I E , ADD UP THE
ELEMENTS OF A VECTOR)  *
*     VECTORS ARE ALL OF TYPE REAL
                *
*       THE MAIN PROGRAM CALLS THIS ROUTINE
SEVERAL TIMES WITH       *
*     DIFFERENT VECTOR LENGTHS
                  *
************************************************
**********************
      IMPLICIT REAL (V)

      COMMON /VVECT/ VV00(200000)

      COMMON /RESULT/ VANS00

      DO 100 LOOP = 1, 20
         VANS00 = 0
         DO 1 I = 1, LENGTH
            VANS00 = VANS00 + VV00(I)
1        CONTINUE
100   CONTINUE

      RETURN
      END

      SUBROUTINE VACC40 (LENGTH)
************************************************
**********************
*     JUST LIKE DOT20, BUT 40 DOT PRODUCTS
                  *
*     VECTORS ARE ALL OF TYPE REAL
                *
************************************************
**********************
      IMPLICIT REAL (V)

      COMMON /VVECT/ VV00(200000)

      COMMON /RESULT/ VANS00

      DO 100 LOOP = 1, 40
         VANS00 = 0
         DO 1 I = 1, LENGTH
            VANS00 = VANS00 + VV00(I)
1        CONTINUE
100   CONTINUE

      RETURN
      END

      SUBROUTINE VADD20 (LENGTH)
************************************************
**********************
*     20 VECTOR ADDS       (LIKE V1 = V2+V3)
                  *
*     VECTORS ARE ALL OF TYPE REAL
                *
*       THE MAIN PROGRAM CALLS THIS ROUTINE
SEVERAL TIMES WITH       *
*     DIFFERENT VECTOR LENGTHS
                  *
************************************************
**********************
      IMPLICIT REAL (V)

      COMMON /VECTOR/ V00(200000)

      COMMON /VVECT/ VV00(200000)

      COMMON /VRESLT/ VA00(200000)

      DO 100 LOOP = 1, 20
         DO 1 I = 1, LENGTH
            VA00(I) = V00(I)+VV00(I)
1        CONTINUE
100   CONTINUE

      RETURN
      END

      SUBROUTINE VADD40 (LENGTH)
************************************************
**********************
*     JUST LIKE VADD20, BUT 40 ADDS
                  *
*     VECTORS ARE ALL OF TYPE REAL
                *
************************************************
**********************
      IMPLICIT REAL (V)

      COMMON /VECTOR/ V00(200000)

      COMMON /VVECT/ VV00(200000)
```

```
      COMMON /VRESLT/ VA00(200000)

      DO 100 LOOP = 1, 40
         DO 1 I = 1, LENGTH
            VA00(I) = V00(I)+VV00(I)
1        CONTINUE
100   CONTINUE

      RETURN
      END

      SUBROUTINE DIV20 (LENGTH)
************************************************
**********************
*     20 VECTOR DIVIDES    (LIKE V1 = V2/V3)
                  *
*     VECTORS ARE ALL OF TYPE REAL
                 *
*       THE MAIN PROGRAM CALLS THIS ROUTINE
SEVERAL TIMES WITH       *
*     DIFFERENT VECTOR LENGTHS
                  *
************************************************
**********************
      IMPLICIT REAL (V)

      COMMON /VECTOR/ V00(200000)

      COMMON /VVECT/ VV00(200000)

      COMMON /VRESLT/ VA00(200000)

      DO 100 LOOP = 1, 20
         DO 1 I = 1, LENGTH
            VA00(I) = V00(I)/VV00(I)
1        CONTINUE
100   CONTINUE

      RETURN
      END

      SUBROUTINE DIV40 (LENGTH)
************************************************
**********************
*     JUST LIKE DIV20, BUT 40 DIVIDES
                  *
*     VECTORS ARE ALL OF TYPE REAL
                 *
************************************************
**********************
      IMPLICIT REAL (V)

      COMMON /VECTOR/ V00(200000)

      COMMON /VVECT/ VV00(200000)

      COMMON /VRESLT/ VA00(200000)

      DO 100 LOOP = 1, 40
         DO 1 I = 1, LENGTH
            VA00(I) = V00(I)/VV00(I)
1        CONTINUE
100   CONTINUE

      RETURN
      END

      SUBROUTINE MLAD20 (LENGTH)
************************************************
**********************
*     20 VECTOR@D  ADD, MULTIPLY, ADD, MULTIPLY
*       (LIKE V1 = V3*(V2 + V3*(V2 + V3))
*     VECTORS ARE ALL OF TYPE REAL
                 *
*       THE MAIN PROGRAM CALLS THIS ROUTINE
SEVERAL TIMES WITH       *
*     DIFFERENT VECTOR LENGTHS.
                  *
************************************************
**********************
      IMPLICIT REAL (V)

      COMMON /VECTOR/ V00(200000)

      COMMON /VVECT/ VV00(200000)

      COMMON /VRESLT/ VA00(200000)

      DO 100 LOOP = 1, 20
         DO 1 I = 1, LENGTH
            VA00(I) = V00(I)*(VV00(I) +
V00(I)*(VV00(I) + V00(I)))
1        CONTINUE
100   CONTINUE

      RETURN
      END

      SUBROUTINE MLAD40 (LENGTH)
************************************************
**********************
*     JUST LIKE MLAD20, BUT 40 LOOPS
                  *
*     VECTORS ARE ALL OF TYPE REAL
                 *
************************************************
**********************
      IMPLICIT REAL (V)

      COMMON /VECTOR/ V00(200000)

      COMMON /VVECT/ VV00(200000)

      COMMON /VRESLT/ VA00(200000)

      DO 100 LOOP = 1, 40
         DO 1 I = 1, LENGTH
            VA00(I) = V00(I)*(VV00(I) +
V00(I)*(VV00(I) + V00(I)))
1        CONTINUE
100   CONTINUE

      RETURN
      END
```

Input Data for Lindsay Benchmark

1	10	10000
2	10	5000
3	10	3000
4	10	2000
5	10	3000
6	10	1500
7	10	1500
8	10	700
9	10	700
10	10	300
11	10	300
12	10	150
13	10	150
14	10	50
15	10	50
16	10	30
17	10	7700
18	10	3000
19	10	3000
20	10	1500
21	10	1500
22	10	700
23	10	700
24	10	400
25	10	400
26	10	150
27	10	150
28	10	70
29	10	70
30	10	30
31	10	30
32	10	15
33	10	7000
34	10	3000
35	10	3000
36	10	1500
37	10	1500
38	10	800
39	10	800
40	10	500
41	10	500
42	10	150
43	10	150
44	10	70
45	10	70
46	10	30
47	10	30
48	10	15
49	10	7000
50	10	3000
51	10	3000
52	10	1000
53	10	1000
54	10	700
55	10	700
56	10	300
57	10	300
58	10	150
59	10	150
60	10	100
61	10	100
62	10	100
63	10	100
64	10	30
65	10	10000
66	10	7000
67	10	7000
68	10	3000
69	10	3000
70	10	1500
71	10	1500
72	10	700
73	10	700
74	10	300
75	10	300
76	10	200
77	10	200
78	10	200
79	10	200
80	10	70
81	10	10000
82	10	5000
83	10	3000
84	10	2000
85	10	2300
86	10	1000
87	10	1000
88	10	700
89	10	700
90	10	300
91	10	300
92	10	100
93	10	100
94	10	70
95	10	50
96	10	30
97	10	7000
98	10	7000
99	10	3000
100	10	1000
101	10	1500
102	10	500
103	10	500
104	10	400
105	10	400
106	10	150
107	10	150
108	10	70
109	10	70
110	10	30
111	10	30
112	10	15
113	10	5000
114	10	2000
115	10	2000
116	10	1000
117	10	1000
118	10	500
119	10	500
120	10	300
121	10	300
122	10	200
123	10	200
124	10	70
125	10	70
126	10	30
127	10	30
128	10	20
0	0	0

Black Box Application Program

```
PRG1(INPUT,OUTPUT,TAPE5=INPUT,TAPE6=OUTPUT)
C  TEST PROGRAM FOR PRODUCT OF MATRIX
C
      DIMENSION U(100,100),V(100,100),W(100,100)
      DIMENSION TIME(5,5)
      REAL MT,KT,MKT
      REAL RSTTIME,RENTIME,RTOTIME
      MULT=20
      RSTTIME=SECOND()
      DO 1000 INDEX=1,5
      N=INDEX*MULT
      L=100
         DO 20 J=1,N
              DO 10 I=1,N
                   IF (I LT J) GO TO 5
                   WW=0 05*(I+J-1)
                   U(I,J)=WW
                   V(I,J)=WW
                   GO TO 10
    5              U(I,J)=U(J,I)
                   V(I,J)=V(J,I)
   10         CONTINUE
   20    CONTINUE
      IF(INDEX NE 1) GO TO 30
      WRITE(6,701)
  701 FORMAT("1",5X,"ORIGINAL MATRIX V",//)
      CALL MPRINT(N,V,L)
      WRITE(6,702)
  702 FORMAT("1",5X,"ORIGINAL MATRIX U",//)
      CALL MPRINT(N,U,L)
   30 CONTINUE
      DO 50 II=1,5
      CALL TIMER(KT)
      GO TO(11,12,13,14,15),II
   11 CALL PRDMX1(U,V,W,N,N,N)
      GO TO 500
   12 CALL PRDMX2(U,V,W,N,N,N)
      GO TO 500
   13 CALL PRDMX3(U,V,W,N,N,N)
      GO TO 500
   14 CALL PRDMX4(U,V,W,N,N,N,10000,10000,10000)
      GO TO 500
   15 CALL PRDMX5(U,V,W,N,N,N,10000,10000,10000)
  500 CALL TIMER(MT)
      MKT = MT - KT
      TIME(II,INDEX)=MKT
      IF(INDEX NE 1) GO TO 50
      WRITE(6,620)
  620
FORMAT(1H1,17X,40H********************************
**********,
     2
30H********************************)
      WRITE(6,610)II,N,MKT
  610
FORMAT(1HO,17X,1H*,1X,5HCASE=,I2,5X,5HSIZE=,I3,5
X,
     1        18HPRODUCT OF MATRIX ,
5X,5HTIME=,F6 4, " SEC ")
      WRITE(6,300)
  300
FORMAT(1HO,17X,40H********************************
**********,
     2
30H********************************)
      CALL MPRINT(N,W,L)
   50 CONTINUE
 1000 CONTINUE
      WRITE(6,2000)
(I*MULT,I=1,5),(I,(TIME(I,J),J=1,5),I=1,5)
 2000 FORMAT("1",T30,"TIME IN SECONDS OF MATRIX
MULTIPLIES",///,
     * T21,"MATRIX
SIZE",/,T5,"ROUTINE",T15,5I10,//,
     * 5(I5,10X,5F10 6,/))

  100 RENTIME=SECOND()
      RTOTIME=RENTIME-RSTTIME
      WRITE (6,3000) RSTTIME,RENTIME,RTOTIME
 3000 FORMAT (1X,'START TIME = ',F10 5,' END
TIME = ',F10.5,
             ' RUN TIME = ',F10.5)
      STOP
      END
C
      SUBROUTINE PRDMX1(A,B,C,L,M,N)
C   PRODUCT OF MATRIX 1
      DIMENSION A(100,100),B(100,100),C(100,100)
      DO 30 I=1,L
         DO 20 K=1,N
              S=0.0
              DO 10 J=1,M
                   S=S+A(I,J)*B(J,K)
   10         CONTINUE
              C(I,K)=S
   20    CONTINUE
   30 CONTINUE
      RETURN
      END
C
      SUBROUTINE PRDMX2(A,B,C,L,M,N)
C   PRODUCT OF MATRIX 2
      DIMENSION A(100,100),B(100,100),C(100,100)
      DO 30 I=1,L
         DO 20 K=1,N
              C(I,K)=0 0
              DO 10 J=1,M
                   C(I,K)=C(I,K)+A(I,J)*B(J,K)
   10         CONTINUE
   20    CONTINUE
   30 CONTINUE
      RETURN
      END
C
      SUBROUTINE PRDMX3(A,B,C,L,M,N)
C   PRODUCT OF MATRIX 3
      DIMENSION A(100,100),B(100,100),C(100,100)
      DIMENSION D(100),E(100)
      KODD=MOD(M,2)
      DO 20 I=1,L
         S=0 0
         DO 10 J=2,M,2
              S=S+A(I,J-1)*A(I,J)
   10    CONTINUE
         D(I)=S
   20 CONTINUE
      DO 40 K=1,N
         S=0 0
         DO 30 J=2,M,2
              S=S+B(J-1,K)*B(J,K)
   30    CONTINUE
         E(K)=S
```

```
   40 CONTINUE
C
      DO 100 I=1,L
         DO 90 K=1,N
              S=-D(I)-E(K)
              DO 80 J=2,M,2

S=S+(A(I,J-1)+B(J,K))*(A(I,J)+B(J-1,K))
   80         CONTINUE
              IF(KODD EQ 0) GO TO 81
              S=S+A(I,M)*B(M,K)
   81         C(I,K)=S
   90    CONTINUE
  100 CONTINUE
      RETURN
      END
C
      SUBROUTINE PRDMX4(A,B,C,L,M,N,NA2,NB2,NC2)
C   PRODUCT OF MATRIX 4
      DIMENSION A(10000),B(10000),C(10000)
      GO TO 2200
 1100 DO 30 I=1,L
         IB=1
         IC=I
         DO 20 K=1,N
              S=0 0
              IA=I
              DO 10 J=1,M
                   S=S+A(IA)*B(IB)
                   IA=IA+NA
                   IB=IB+1
   10         CONTINUE
              C(IC)=S
              IB=IB+NBM
              IC=IC+NC
   20    CONTINUE
   30 CONTINUE
      RETURN
 2200 NA=SQRT(FLOAT(NA2))+0 1
      NB=SQRT(FLOAT(NB2))+0 1
      NC=SQRT(FLOAT(NC2))+0 1
      NBM=NB-M
      GO TO 1100
      END
C
      SUBROUTINE PRDMX5(A,B,C,L,M,N,NA2,NB2,NC2)
C   PRODUCT OF MATRIX 5
      DIMENSION A(10000),B(10000),C(10000)
      DIMENSION D(100),E(100)
      KODD=MOD(M,2)
      GO TO 2200
 1100 DO 20 I=1,L
         S=0 0
         L1=I
         DO 10 J=2,M,2
              L2=L1+NA
              S=S+A(L1)*A(L2)
              L1=L2+NA
   10    CONTINUE
         D(I)=S
   20 CONTINUE
      L1=1
      DO 40 K=1,N
         S=0 0
         DO 30 J=2,M,2
              L2=L1+1
              S=S+B(L1)*B(L2)
              L1=L2+1
   30    CONTINUE
         E(K)=S
         L1=L1+NBM2
   40 CONTINUE
C
      DO 100 I=1,L
         IC=I
         IB=1
         DO 90 K=1,N
              S=-D(I)-E(K)
              IA=I
              DO 80 J=2,M,2
                   IA2=IA+NA

S=S+(A(IA)+B(IB+1))*(A(IA2)+B(IB))
                   IA=IA2+NA
                   IB=IB+2
   80         CONTINUE
              IF(KODD.EQ.0) GO TO 81
              S=S+A(IA)*B(IB)
   81         C(IC)=S
              IC=IC+NC
              IB=IB+NBM2
   90    CONTINUE
  100 CONTINUE
      RETURN
 2200 NA=SQRT(FLOAT(NA2))+0.1
      NB=SQRT(FLOAT(NB2))+0.1
      NC=SQRT(FLOAT(NC2))+0 1
      NBM2=NB-M/2*2
      GO TO 1100
      END
C
      SUBROUTINE MPRINT(N,A,L)
      DIMENSION A(L,L)
      J2=0
    1 J1=J2+1
      J2=J2+10
      IF(J2 GT.N) J2=N
      WRITE(6,61) (J,J=J1,J2)
   61 FORMAT(///,11X,10I10,/)
      DO 10 I=1,N
      WRITE(6,62) I,(A(I,J),J=J1,J2)
   62 FORMAT(1H ,I10,10F10.5)
   10 CONTINUE
      IF(J2.LT.N) GO TO 1
      RETURN
      END
C
      SUBROUTINE TIMER(X)
C     CALL SECOND(S)
C     X=S
      X=SECOND()
      RETURN
```

Subtest Application Program

```
      PROGRAM MAIN
      PARAMETER (N=20011)
      COMMON /XXX/ A(N),B(N),C(N)
      REAL T0,T1,Y
      REAL RSTTIME,RENTIME,RTOTIME
      RSTTIME=SECOND()
      Y=.3
      DO 10 I=1,N
      A(I)=RANDOM(Y)
      B(I)=RANDOM(Y)
      C(I)=RANDOM(Y)
   10 CONTINUE
      T0=SECOND()
* DO 400
      CALL ASSIGNA(500,400)
* DO 300
      CALL ADDA(500,300)
* DO 200
      CALL MULTA(500,200)
* DO 100
      CALL DIVA(500,100)
      T1=SECOND()
      WRITE (6,100) T0,T1
      WRITE (6,101) T1-T0
      PRINT*,'START TIME = ',T0
      PRINT*,'STOP TIME = ',T1
  100 FORMAT(1X,'START TIME = ',F10 6,' STOP
TIME = ',F10 6)
  101 FORMAT(1X,'TOTAL TIME = ',F10 6)
      RENTIME=SECOND()
      RTOTIME=RENTIME-RSTTIME
      WRITE (6,3000) RSTTIME,RENTIME,RTOTIME
 3000 FORMAT (1X,'START TIME = ',F10 5,' END
TIME = ',F10 5,
             ' RUN TIME = ',F10 5)
      STOP
      END
      SUBROUTINE ASSIGNA(VL,REP)
      COMMON /XXX/ A(20011),B(20011),C(20011)
      INTEGER VL,REP
      DO 15 I=1,REP
        DO 10 J=1,VL
        B(J)=A(J)
   10   CONTINUE
   15 CONTINUE
      RETURN
      END
      SUBROUTINE ADDA(VL,REP)
      COMMON /XXX/ A(20011),B(20011),C(20011)
      INTEGER VL,REP
      DO 15 I=1,REP
        DO 10 J=1,VL
        C(J)=A(J)+B(J)
   10   CONTINUE
   15 CONTINUE
      RETURN
      END
      SUBROUTINE MULTA(VL,REP)
      COMMON /XXX/ A(20011),B(20011),C(20011)
      INTEGER VL,REP
      DO 15 I=1,REP
        DO 10 J=1,VL
        C(J)=A(J)*B(J)
   10   CONTINUE
   15 CONTINUE
      RETURN
      END
      SUBROUTINE DIVA(VL,REP)
      COMMON /XXX/ A(20011),B(20011),C(20011)
      INTEGER VL,REP
      DO 15 I=1,REP
        DO 10 J=1,VL
        C(J)=A(J)/B(J)
   10   CONTINUE
   15 CONTINUE
      RETURN
      END
      FUNCTION RANDOM(Y)
      Y=AMOD(16807.*Y,2147483647 )
      RANDOM=Y*4.6566128752458E-10
      RETURN
      END
```

SMMBPF Application Program

```
SMMBPF
      SUBROUTINE BPF(TR)
      REAL TR(*)
C
      COMMON
/SEISCM/NPROC,NSIZE,T,SR,NTV,SR1,NTR,IPARM,
     *
ILX,NTPL,IHX,NTPH,COSPH,SINPH,NFFT,NTAPM,
     *  TAPL(100),TAPH(200)
C
      IF(TR(11) LE.0 ) RETURN
C
      IF(NTAPM.GT 0) CALL
CLEARW(TR(NTV+33),NTAPM)
      CALL RFFT(TR(33),NFFT,1)
C
      IF(NTPL GT 0) THEN
        DO 9 I=1,NTPL
           I1=ILX+2*I+31
           TR(I1)=TAPL(I)*TR(I1)
           I1=I1+1
    9      TR(I1)=TAPL(I)*TR(I1)
        ENDIF
C
      IF(NTPH GT 0) THEN
        DO 19 I=1,NTPH
           I1=IHX-2*I+33
           TR(I1)=TAPH(I)*TR(I1)
           I1=I1+1
   19      TR(I1)=TAPH(I)*TR(I1)
        ENDIF
C
      IF(COSPH NE 1 0 OR SINPH NE 0 0) THEN
        DO 29 I=ILX,IHX,2
          A=TR(33+I)
          B=TR(34+I)
          TR(33+I)=A*COSPH-B*SINPH
   29     TR(34+I)=A*SINPH+B*COSPH

        ENDIF
C

C
      N=TR(5)-1
C
      RETURN
      END
```

SMDSEQT Application Program

```
SMDSEQT
C=============================================
==================
C --- READ PARAMETERS, PREPROCESS AND SAVE
C=============================================
==================
      COMMON /SYSPRM/ IPRINT
      COMMON
/SEISCM/NPROC,NSIZE,T,SR,NTV,SR1,NTR,IPARM,
     *  NCMP,NOFF,X1,XINC,OFF1,OFFINC,NREP,
     *  IREP,IOFF,ICMP,ITRACE,LU,T1,SR2,SR3,NTV1
      COMMON /DUMDAT/ DUMY(8000)
C
      READ(LUN,'(F8 2,F8 3,I5)',END=900)
T,SR,NREP
      IF (NREP.LT 1) NREP=1
C     WRITE(IPRINT,*) ' TIME=',T,' SAMPLE
RATE=',SR
      READ(LUN,15,END=900)
NCMP,NOFF,X1,XINC,OFF1,OFFINC
   15 FORMAT(2I5,4F10 2)
      WRITE(IPRINT,25)
NCMP,NOFF,XINC,OFFINC,NREP
   25 FORMAT(' DTRIN-D PARAMETERS NCMP=',I5,',
NOFF=',I5,
     *  ', XINC=',F7 2,', OFFINC=',F7 2,/,20X,'#
REPS=',I5)
      NTR=NCMP*NOFF
      ITRACE=0
      IOFF=0
      ICMP=1
      IREP=1
      CALL GETLUN(LU)
      SR1=1 /SR
      NTV=T/SR+1 5
      WRITE(IPRINT,835) T,SR,NREP*NTR
  835 FORMAT(' TIME=',F8 2,', SR=',F8 3,', #
TRACES=',I5)
      CALL RANGET(ISEED)
      CALL RANSET(ISEED)
      DO 30 I=1,NTV
        DUMY(I)=0 5-RANF()
   30 CONTINUE
      T1=T
      SR2=SR
      SR3=SR1
      NTV1=NTV
      CALL WRTDB(NPROC,IPARM,10,7)
      RETURN
C
  900 RETURN 1
      END

      SUBROUTINE STK(TRACE,NTRC)

      INTEGER NHW,MAXNS,MAXTL,MAXNO
      PARAMETER
(NHW=32,MAXNS=8050,MAXTL=NHW+MAXNS,MAXNO=1024)

      INTEGER NTRC,LUN
      REAL TRACE(MAXTL)

      INTEGER NPROC,NSIZE,NS,NTR,IPARM
      REAL TMAX,SR,SRI
      COMMON /SEISCM/
NPROC,NSIZE,TMAX,SR,NS,SRI,NTR,IPARM,
     +                FSE,MSF
      REAL FSE,MSF

      INTEGER IPRINT,IREAD
      COMMON /SYSPRM/ IPRINT,IREAD

      INTEGER NSTAK,MUTE(MAXNO)
      REAL STAK(MAXTL)
      LOGICAL START,NEWCMP,DONE
      COMMON /STKTAB/
START,NEWCMP,DONE,NSTAK,STAK,MUTE

      INTEGER I
      REAL SWITCH

      SAVE /STKTAB/

      IF (NTRC EQ.0) THEN
        DONE=.TRUE
      ELSE IF (TRACE(3) NE.STAK(3)) THEN
        NEWCMP=.TRUE.
      END IF

      IF (DONE.OR NEWCMP) THEN
        IF (NSTAK.GT 0) THEN
          CALL
SCALE(STAK,MUTE,NHW,NS,NSTAK,FSE,MSF)
        END IF
        IF ( NOT DONE) THEN
          IF ( NOT START) THEN
            DO 10 I=1,NHW+NS
              SWITCH=TRACE(I)
              TRACE(I)=STAK(I)
              STAK(I)=SWITCH
   10       CONTINUE
            NSTAK=1
            MUTE(1)=STAK(5)
            NTRC=1
            NEWCMP= FALSE.
          ELSE
            DO 20 I=1,NHW+NS
              STAK(I)=TRACE(I)
   20       CONTINUE
            NSTAK=1
            MUTE(1)=STAK(5)
            NTRC=0
            START=.FALSE
            NEWCMP= FALSE
          END IF
        ELSE
          DO 30 I=1,NHW+NS
            TRACE(I)=STAK(I)
   30     CONTINUE
          NTRC=1
        END IF
      ELSE
        CALL STACK(TRACE,MUTE,NHW,NS,NSTAK,STAK)
        NTRC=0
      END IF

      RETURN

      SUBROUTINE
STACK(TRACE,MUTE,NHW,NS,NSTAK,STAK)

      INTEGER NHW,NS,NSTAK,MUTE(*)
```

```
      REAL TRACE(NHW+NS),STAK(NHW+NS)

      INTEGER I

      NSTAK=NSTAK+1
      MUTE(NSTAK)=TRACE(5)

      DO 10 I=NHW+MUTE(NSTAK),NS
        STAK(I)=STAK(I)+TRACE(I)
   10 CONTINUE

      RETURN
      END

      SUBROUTINE TDF(TRACE)
      MUTE=TRACE(5)
      DO 10 I=NHW+MUTE,NHW+NS
        SIGNAL(I-NHW)=TRACE(I)
   10 CONTINUE

      J1=(NFP/2+1)+1
      J2=MIN(NFP,(NS-MUTE+NFP/2+1)+1)
      DO 30 I=MUTE,NS
        TRACE(NHW+I)=0 0
        IF ((I+NFP/2) GT NS) J2=J2-1
        IF ((I-NFP/2) LT MUTE) J1=J1-1
        DO 20 J=J1,J2

TRACE(NHW+I)=TRACE(NHW+I)+FILTER(J)*SIGNAL(I-NFP
/2-1+J)
   20   CONTINUE
   30 CONTINUE

      RETURN

      SUBROUTINE XP(TR)
      IF (NR GT IDIMR) GO TO 200
      IF (INS GT ITM(1)) GO TO 61
      INC(NR) = ISL(1)
      GO TO 66
C
   61 IF (INS LT ITM(NTM)) GO TO 62
      INC(NR) = ISL(NTM)
      GO TO 66
C
   62 DO 64 I=2,NTM
        IF (INS GT ITM(I)) GO TO 64
        FAC = FLOAT(INS-ITM(I-1)) /
FLOAT(ITM(I)-ITM(I-1))
        INC(NR) = FAC*FLOAT(ISL(I)-ISL(I-1)) +
ISL(I-1)
        GO TO 66
   64   CONTINUE
      INC(NR) = ISL(NTM)

   66 LNS = INS + INC(NR) -1

      IF (LNS LT LWIN) GO TO 68
      IF (LNS LT NDUM) NDUM=LNS
C
   68 TPR= 0
      KK =0

      IF (INS GT NDUM) GO TO 80

      IF (LNS LE NDUM) GO TO 70
      LNS = NDUM
      INS = LNS-INC(NR)+1
      IF (INS LT KNT) INS = KNT
C

   70 DO 71 J=INS,LNS
        TPR= TPR + ABS(TR(J))
        IF (TR(J).NE 0.0) KK = KK+1
   71   CONTINUE
      IF(KK.EQ.0) GO TO 60
      PPR = SAMAV * KK

      IF (TPR EQ 0.) GO TO 60

      R(NR) = PPR / TPR

      GO TO 60
C

   80 R(NR) = R(NR-1)
      DO 90 I=1,NR
        IF (R(I) EQ.0.0 .AND. I.GT 1) R(I) =
R(I-1)
        IF (R(I).GT ALMP) R(I)=ALMP
   90   CONTINUE
C
C

      IDIV = INC(NR)-1

      LVAL = NDUM-(INC(NR)/2)

C      INS = 1
      INS=33
      LNS = KNT+(INC(1)/2) -1
      IF (LNS.GT.NDUM1) LNS = NDUM1
      I = 1
      IAMP1 = R(I) * FPC
      IAMP2 = IAMP1
      IAMPAD = 0
      IX = 0

  100 TRMX=0
      INSX=0
      LNSX = 0
  102 DO 101 JJ=INS,LNS
        TR(JJ)=TR(JJ)*IAMP1
        IF (ABS(TR(JJ)) LE.FPLMIT) GO TO 103
        IF(IDEB EQ  0) GO TO 103
        IF(IX.EQ.0) GO TO 300
        IF(JJ-ITHRSH(IX)  LT. I2CYC) GO TO 310
  300   IX=IX+1
        ITHRSH(IX)=JJ
        IX=IX+1
  310   ITHRSH(IX)=JJ
  103   IAMP1 = IAMP1 + IAMPAD
  101   CONTINUE

  115 I = I + 1
      INS = LNS+1
      LNS = INS + ((INC(I)+INC(I-1))/2) -1

      IF (INS GT NDUM1) GO TO 150
      IF (INS.GE LVAL) GO TO 120

      IF (LNS.LE.LVAL) GO TO 130
      LNS = LVAL
      GO TO 130
C
  120 LNS = NDUM1
C
```

```
  130 IDIV = LNS - INS

      IF (IDIV EQ 0) IDIV = 1
C
  140 IAMP1 = IAMP2
      IAMP2 = R(I) * FPC
      IAMPAD = (IAMP2-IAMP1) / IDIV

      GO TO 100
C
C
  150 IF(IDEB EQ. 0 ) GO TO 900
      IF(IX EQ 0) GO TO 900

C


      DO 500 JP=1,IX,2
        INSX = ITHRSH(JP)
        LNSX = ITHRSH(JP+1)

        TRMX = 0
        DO 510 JP3=INSX,LNSX
          ABTR= ABS(TR(JP3))
          IF(ABTR GT TRMX) TRMX=ABTR
  510     CONTINUE
        IF(TRMX EQ  0) GO TO 500
        IAMPX1 = FPC
        IAMPX2 =FPLMIT/ TRMX
        INSY = INSX - ICYC
        IF (INSY LT KNT) INSY = KNT
        LNSY = INSX - 1
        IDIVX = INSY - LNSY - 1
        IF (IDIVX EQ 0) IDIVX = 1
        IADX = (FPC-IAMPX2)/IDIVX
        ASSIGN 106 TO IBR

  104   DO 107 JJ=INSY,LNSY
          TR(JJ)=TR(JJ)* IAMPX1
  107     IAMPX1=IAMPX1 + IADX
        GO TO IBR,(106,108,500)
C

  106   LNSY = LNSX + ICYC
        IF (LNSY GT NDUM) LNSY = NDUM
        INSY = LNSX + 1
        IDIVX = LNSY - INSY + 1
        IF (IDIVX EQ 0) IDIVX = 1
        IADX = (FPC-IAMPX2)/IDIVX
        IAMPX1 = IAMPX2 + IADX
        ASSIGN 108 TO IBR

        GO TO 104
C


  108   LNSY = LNSX
        INSY = INSX
        IAMPX1 = IAMPX2
        IADX= 0
        ASSIGN 500 TO IBR
        GO TO 104
C
  500   CONTINUE
C
  900 RETURN

  200 PRINT 1010
C
C *** CALL EXEXIT (0) *** REPLACED BY 'STOP'
      STOP
C

C
      END

      SUBROUTINE NMO(TRACE)

      IF (DMF.GT.0 0) THEN
        NMO2=(OFFSET/VELX(1))*(OFFSET/VELX(1))
        TOFF2(1)=NMO2
        T0=SQRT(DMF*NMO2)
        I1=INT(SRI*T0)+1
        T0=REAL(I1-1)*SR
        DO 20 I=I1,2,-1
          NMO2=(OFFSET/VELX(I))*(OFFSET/VELX(I))
          TOFF2(I)=T0*T0+NMO2
          IF (T0*T0.LT DMF*NMO2) GO TO 25
          T0=T0-SR
   20   CONTINUE
   25   I1=I
      ELSE
        I1=1
        T0=0 0
      END IF

      DO 30 I=I1,NS
        NMO2=(OFFSET/VELX(I))*(OFFSET/VELX(I))
        TOFF2(I)=T0*T0+NMO2
        T0=T0+SR
   30 CONTINUE
      DO 40 I=I1,NS
        IF (TOFF2(I).GT.TMIN*TMIN) GO TO 45
   40 CONTINUE
   45 I1=I
      DO 50 I=NS-1,I1,-1
        IF (TOFF2(I).LT.TMAX*TMAX) GO TO 55
   50 CONTINUE
   55 I2=I

      IMUTE=TRACE(5)
      MUTE=I1
      IF (MUTE.GT.I2) MUTE=NS+1
      IF (MUTE GT.IMUTE) CALL
FILLW(TRACE(NHW+IMUTE),0 0,MUTE-IMUTE)
      TRACE(5)=MUTE

      IF (MUTE GT NS) THEN
        TRACE(11)=0.0
        RETURN
      END IF

      DO 60 I=I1,I2
        TOFF(I)=SQRT(TOFF2(I))
        IF (TOFF(I) GT.TMAX) TOFF(I)=0 0
   60 CONTINUE
      DO 70 I=I1,I2
        IPS(I)=INT(SRI*TOFF(I))+1
   70 CONTINUE
      TRACE(NHW+1)=0.0
      DO 80 I=I1,I2
        FRAC=SRI*(TOFF(I)-SR*REAL(IPS(I)-1))
        TRACE(NHW+I)=TRACE(NHW+IPS(I))+
     +
FRAC*(TRACE(NHW+IPS(I)+1)-TRACE(NHW+IPS(I)))
   80 CONTINUE
      CALL FILLW(TRACE(NHW+I2+1),0 0,NS-I2)

      RETURN
```

SMMTDF Application Program

```
SMMTDF
      SUBROUTINE TDF(TRACE)
      MUTE=TRACE(5)
      DO 10 I=NHW+MUTE,NHW+NS
        SIGNAL(I-NHW)=TRACE(I)
   10 CONTINUE

      J1=(NFP/2+1)+1
      J2=MIN(NFP,(NS-MUTE+NFP/2+1)+1)
      DO 30 I=MUTE,NS
        TRACE(NHW+I)=0.0
        IF ((I+NFP/2) GT NS) J2=J2-1
        IF ((I-NFP/2) LT MUTE) J1=J1-1
        DO 20 J=J1,J2

TRACE(NHW+I)=TRACE(NHW+I)+FILTER(J)*SIGNAL(I-NFP
/2-1+J)
   20   CONTINUE
   30 CONTINUE

      RETURN
```

SMMNMO Application Program

```
SMMNMO
      SUBROUTINE NMO(TRACE)

      IF (DMF GT 0 0) THEN
        NMO2=(OFFSET/VELX(1))*(OFFSET/VELX(1))
        TOFF2(1)=NMO2
        T0=SQRT(DMF*NMO2)
        I1=INT(SRI*T0)+1
        T0=REAL(I1-1)*SR
        DO 20 I=I1,2,-1
          NMO2=(OFFSET/VELX(I))*(OFFSET/VELX(I))
          TOFF2(I)=T0*T0+NMO2
          IF (T0*T0.LT DMF*NMO2) GO TO 25
          T0=T0-SR
   20   CONTINUE
   25   I1=I
      ELSE
        I1=1
        T0=0 0
      END IF

      DO 30 I=I1,NS
        NMO2=(OFFSET/VELX(I))*(OFFSET/VELX(I))
        TOFF2(I)=T0*T0+NMO2
        T0=T0+SR
   30 CONTINUE
      DO 40 I=I1,NS
        IF (TOFF2(I) GT TMIN*TMIN) GO TO 45
   40 CONTINUE
   45 I1=I
      DO 50 I=NS-1,I1,-1
        IF (TOFF2(I) LT TMAX*TMAX) GO TO 55
   50 CONTINUE
   55 I2=I

      IMUTE=TRACE(5)
      MUTE=I1
      IF (MUTE GT I2) MUTE=NS+1
      IF (MUTE.GT.IMUTE) CALL
FILLW(TRACE(NHW+IMUTE),0 0,MUTE-IMUTE)
      TRACE(5)=MUTE

      IF (MUTE GT NS) THEN
        TRACE(11)=0 0
        RETURN
      END IF

      DO 60 I=I1,I2
        TOFF(I)=SQRT(TOFF2(I))
        IF (TOFF(I).GT TMAX) TOFF(I)=0 0
   60 CONTINUE
      DO 70 I=I1,I2
        IPS(I)=INT(SRI*TOFF(I))+1
   70 CONTINUE
      TRACE(NHW+1)=0.0
      DO 80 I=I1,I2
        FRAC=SRI*(TOFF(I)-SR*REAL(IPS(I)-1))
        TRACE(NHW+I)=TRACE(NHW+IPS(I))+
     +
FRAC*(TRACE(NHW+IPS(I)+1)-TRACE(NHW+IPS(I)))
   80 CONTINUE
      CALL FILLW(TRACE(NHW+I2+1),0 0,NS-I2)

      RETURN
```

SMMXP Application Program

```
SMMXP
      SUBROUTINE XP(TR)
      IF (NR GT IDIMR) GO TO 200
      IF (INS GT ITM(1)) GO TO 61
      INC(NR) = ISL(1)
      GO TO 66
C
   61 IF (INS LT ITM(NTM)) GO TO 62
      INC(NR) = ISL(NTM)
      GO TO 66
C
   62 DO 64 I=2,NTM
        IF (INS GT.ITM(I)) GO TO 64
        FAC = FLOAT(INS-ITM(I-1)) /
FLOAT(ITM(I)-ITM(I-1))
        INC(NR) = FAC*FLOAT(ISL(I)-ISL(I-1)) +
ISL(I-1)
        GO TO 66
   64   CONTINUE
      INC(NR) = ISL(NTM)

   66 LNS = INS + INC(NR) -1

      IF (LNS LT LWIN) GO TO 68
      IF (LNS LT NDUM) NDUM=LNS
C
   68 TPR= 0
      KK =0

      IF (INS GT NDUM) GO TO 80

      IF (LNS LE NDUM) GO TO 70
      LNS = NDUM
      INS = LNS-INC(NR)+1
      IF (INS LT KNT) INS = KNT
C

   70 DO 71 J=INS,LNS
        TPR= TPR + ABS(TR(J))
        IF (TR(J) NE 0 0) KK = KK+1
   71   CONTINUE
      IF(KK EQ 0) GO TO 60
      PPR = SAMAV * KK

      IF (TPR EQ 0 ) GO TO 60

      R(NR) = PPR / TPR

      GO TO 60
C

   80 R(NR) = R(NR-1)
      DO 90 I=1,NR
        IF (R(I) EQ 0 0  AND  I GT 1) R(I) =
R(I-1)
        IF (R(I) GT ALMP) R(I)=ALMP
   90   CONTINUE
C
C


      IDIV = INC(NR)-1

      LVAL = NDUM-(INC(NR)/2)

C      INS = 1
      INS=33
      LNS = KNT+(INC(1)/2) -1
      IF (LNS GT NDUM1) LNS = NDUM1
      I = 1
      IAMP1 = R(I) * FPC
      IAMP2 = IAMP1
      IAMPAD = 0.
      IX = 0


  100 TRMX=0.
      INSX=0
      LNSX = 0
  102 DO 101 JJ=INS,LNS
        TR(JJ)=TR(JJ)*IAMP1
        IF (ABS(TR(JJ)).LE.FPLMIT) GO TO 103
        IF(IDEB.EQ. 0) GO TO 103
        IF(IX.EQ.0) GO TO 300
        IF(JJ-ITHRSH(IX) .LT. I2CYC) GO TO 310
  300   IX=IX+1
        ITHRSH(IX)=JJ
        IX=IX+1
  310   ITHRSH(IX)=JJ
  103   IAMP1 = IAMP1 + IAMPAD
  101   CONTINUE

  115 I = I + 1
      INS = LNS+1
      LNS = INS + ((INC(I)+INC(I-1))/2) -1

      IF (INS GT NDUM1) GO TO 150
      IF (INS GE LVAL) GO TO 120

      IF (LNS LE LVAL) GO TO 130
      LNS = LVAL
      GO TO 130
C
  120 LNS = NDUM1
C
  130 IDIV = LNS - INS

      IF (IDIV EQ.0) IDIV = 1
C
  140 IAMP1 = IAMP2
      IAMP2 = R(I) * FPC
      IAMPAD = (IAMP2-IAMP1) / IDIV

      GO TO 100
C
C
  150 IF(IDEB EQ. 0 ) GO TO 900
      IF(IX.EQ 0) GO TO 900


C

      DO 500 JP=1,IX,2
        INSX = ITHRSH(JP)
        LNSX = ITHRSH(JP+1)

        TRMX = 0.
        DO 510 JP3=INSX,LNSX
          ABTR= ABS(TR(JP3))
          IF(ABTR.GT TRMX) TRMX=ABTR
  510     CONTINUE
        IF(TRMX.EQ. 0) GO TO 500
        IAMPX1 = FPC
        IAMPX2 =FPLMIT/ TRMX
        INSY = INSX - ICYC
        IF (INSY LT KNT) INSY = KNT
```

```
      LNSY = INSX - 1
      IDIVX = INSY - LNSY - 1
      IF (IDIVX EQ.0) IDIVX = 1
      IADX = (FPC-IAMPX2)/IDIVX
      ASSIGN 106 TO IBR

  104 DO 107 JJ=INSY,LNSY
        TR(JJ)=TR(JJ)* IAMPX1
  107   IAMPX1=IAMPX1 + IADX
      GO TO IBR,(106,108,500)
C

  106 LNSY = LNSX + ICYC
      IF (LNSY GT NDUM) LNSY = NDUM
      INSY = LNSX + 1
      IDIVX = LNSY - INSY + 1
      IF (IDIVX EQ 0) IDIVX = 1
      IADX = (FPC-IAMPX2)/IDIVX
      IAMPX1 = IAMPX2 + IADX
      ASSIGN 108 TO IBR

      GO TO 104
C

  108 LNSY = LNSX
      INSY = INSX
      IAMPX1 = IAMPX2
      IADX= 0
      ASSIGN 500 TO IBR
      GO TO 104
C
  500 CONTINUE
C
  900 RETURN

  200 PRINT 1010
C
C *** CALL EXEXIT (0) *** REPLACED BY 'STOP'
      STOP
C

C
      END
```

SMMSTK Application Program

```
SMMSTK
      SUBROUTINE STK(TRACE,NTRC)

      INTEGER NHW,MAXNS,MAXTL,MAXNO
      PARAMETER
(NHW=32,MAXNS=8050,MAXTL=NHW+MAXNS,MAXNO=1024)

      INTEGER NTRC,LUN
      REAL TRACE(MAXTL)

      INTEGER NPROC,NSIZE,NS,NTR,IPARM
      REAL TMAX,SR,SRI
      COMMON /SEISCM/
NPROC,NSIZE,TMAX,SR,NS,SRI,NTR,IPARM,
     +                FSE,MSF
      REAL FSE,MSF

      INTEGER IPRINT,IREAD
      COMMON /SYSPRM/ IPRINT,IREAD

      INTEGER NSTAK,MUTE(MAXNO)
      REAL STAK(MAXTL)
      LOGICAL START,NEWCMP,DONE
      COMMON /STKTAB/
START,NEWCMP,DONE,NSTAK,STAK,MUTE

      INTEGER I
      REAL SWITCH

      SAVE /STKTAB/

      IF (NTRC EQ 0) THEN
        DONE= TRUE.
      ELSE IF (TRACE(3).NE STAK(3)) THEN
        NEWCMP=.TRUE
      END IF

      IF (DONE.OR NEWCMP) THEN
        IF (NSTAK GT 0) THEN
          CALL
SCALE(STAK,MUTE,NHW,NS,NSTAK,FSE,MSF)
        END IF
        IF ( NOT DONE) THEN
          IF ( NOT START) THEN
            DO 10 I=1,NHW+NS
              SWITCH=TRACE(I)
              TRACE(I)=STAK(I)
              STAK(I)=SWITCH
   10       CONTINUE
            NSTAK=1
            MUTE(1)=STAK(5)
            NTRC=1
            NEWCMP= FALSE
          ELSE
            DO 20 I=1,NHW+NS
              STAK(I)=TRACE(I)
   20       CONTINUE
            NSTAK=1
            MUTE(1)=STAK(5)
            NTRC=0
            START= FALSE
            NEWCMP= FALSE
          END IF
        ELSE
          DO 30 I=1,NHW+NS
            TRACE(I)=STAK(I)
   30     CONTINUE
            NTRC=1
          END IF
        ELSE
          CALL STACK(TRACE,MUTE,NHW,NS,NSTAK,STAK)
          NTRC=0
        END IF

        RETURN

      SUBROUTINE
STACK(TRACE,MUTE,NHW,NS,NSTAK,STAK)

      INTEGER NHW,NS,NSTAK,MUTE(*)
      REAL TRACE(NHW+NS),STAK(NHW+NS)

      INTEGER I

      NSTAK=NSTAK+1
      MUTE(NSTAK)=TRACE(5)

      DO 10 I=NHW+MUTE(NSTAK),NS
        STAK(I)=STAK(I)+TRACE(I)
   10 CONTINUE

      RETURN
      END
```

SMMTM Application Program

```
SMMTM
      SUBROUTINE TMUTE (TR)
      IX = T3*SR1+1
      IF (IX GT NTV) IX=NTV+1
      N = NTAP
      IF(IX+N  GT  NTV)  N = NTV-IX + 1
      IF(IX GT 1) THEN
        IX1=IX-1
        CALL CLEARW (TR(33),IX1)
        IF(N GT 0) THEN
          MX = IX + 32
          CALL MULT (TAP,TR(MX),TR(MX),N)
          ENDIF
        ENDIF
      TR(5) = IX
      RETURN
C
      END

      SUBROUTINE MULT(A,B,C,N)
      REAL A(N),B(N),C(N)
C
      DO 9 I=1,N
    9   C(I)=B(I)*A(I)
      RETURN
      END
```

SMMTRIN Application Program

```
SMMTRIN
C================================================
==================
C --- READ PARAMETERS, PREPROCESS AND SAVE
C================================================
==================
      COMMON /SYSPRM/ IPRINT
      COMMON
/SEISCM/NPROC,NSIZE,T,SR,NTV,SR1,NTR,IPARM,
     *  NCMP,NOFF,X1,XINC,OFF1,OFFINC,NREP,
     *  IREP,IOFF,ICMP,ITRACE,LU,T1,SR2,SR3,NTV1
      COMMON /DUMDAT/ DUMY(8000)
C
      READ(LUN,'(F8 2,F8 3,I5)',END=900)
T,SR,NREP
      IF (NREP LT 1) NREP=1
C     WRITE(IPRINT,*) ' TIME=',T,' SAMPLE
RATE=',SR
      READ(LUN,15,END=900)
NCMP,NOFF,X1,XINC,OFF1,OFFINC
   15 FORMAT(2I5,4F10 2)
      WRITE(IPRINT,25)
NCMP,NOFF,XINC,OFFINC,NREP
   25 FORMAT(' DTRIN-D PARAMETERS NCMP=',I5,',
NOFF=',I5,
     *  ', XINC=',F7 2,', OFFINC=',F7 2,/,20X,'#
REPS=',I5)
      NTR=NCMP*NOFF
      ITRACE=0
      IOFF=0
      ICMP=1
      IREP=1
      CALL GETLUN(LU)
      SR1=1 /SR
      NTV=T/SR+1 5
      WRITE(IPRINT,835) T,SR,NREP*NTR
  835 FORMAT(' TIME=',F8 2,', SR=',F8 3,', #
TRACES=',I5)
      CALL RANGET(ISEED)
      CALL RANSET(ISEED)
      DO 30 I=1,NTV
        DUMY(I)=0 5-RANF()
   30 CONTINUE
      T1=T
      SR2=SR
      SR3=SR1
      NTV1=NTV
      CALL WRTDB(NPROC,IPARM,10,7)
      RETURN
C
  900 RETURN 1
      END
```

APPENDIX B

CPU TIMES OF APPLICATION PROGRAMS AND BENCHMARKS IN ALL ENVIRONMENTS

Applications										
Environment	SMMBPF	SMDSEQT	SMMTDF	SMMNMO	SMMXP	SMMSTK	SMMTM	SMMTRIN	Black Box	Subtest
C77v	110 363	71 791	60 116	9 527	9 455	0 750	0 418	0 238	0.722	0 103
CFTv	130.248	95 452	85 093	12 011	8 960	0 844	0 497	0 289	1 310	0 116
C77nv	113 930	192 885	169 027	35 188	10 080	2 394	1 011	0 689	1 704	0 144
CFTnv	150 403	472 678	465 785	50 826	18 030	7 224	3 213	2 466	3 738	0 243
NVEv	176.529	132 375	90 034	39 632	13 266	1 693	0 820	0 432	1 703	0 399
NVEnv	473 980	3259 064	3446 108	199 830	69 045	39 267	18 705	14 924	18 740	1 164
N990	717 285	3439 770	3614 249	210 092	92 276	44 017	20 517	16 985	19 410	0 945
N865	1567 691	7007 143	7344 551	412 646	199 578	88 497	39 585	32 909	44 262	1 625
NV815	7514 300	45900.706	48169 750	3250 200	1114 113	655 053	328 622	270 166	296 081	16 153
N815	13943.122	63797.869	67180.326	4039.139	1539.703	717.766	318.533	257.168	407.123	16.136

Benchmarks										
Environment	LINPACK	VA200	DP1	DP200	SVV200					
C77v	0 861	5 70e-06	6 60e-06	1 10e-05	6 97e-06					
CFTv	1 329	8 25e-06	6 10e-06	1 48e-05	9 15e-06					
C77nv	2 051	2 07e-05	1 33e-06	4 93e-05	5 31e-05					
CFTnv	4 376	8 13e-05	2 11e-06	1 43e-04	1 31e-04					
NVEv	2 170	8 55e-06	5 21e-06	1 90e-05	1 88e-05					
NVEnv	19 194	5 46e-04	6 68e-06	6 93e-04	6 42e-04					
N990	21 105	4 53e-04	6 26e-06	5 68e-04	5 68e-04					
N865	51 114	8 77e-04	1 25e-05	1 10e-03	1 12e-03					
NV815	315 669	7 09e-03	1.11e-04	1 25e-02	1 26e-02					
N815	426.115	7.84e-03	1.10e-04	1.30e-02	1.12e-02					

APPENDIX C

PREDICTED TIMES OF APPLICATION PROGRAMS IN ALL ENVIRONMENTS

Predicted Environment	Available Set	SMMBPF	SMDSEQT	SMMTDF	SMMNMO	SMMXP	SMMSTK	SMMTM	SMMTRIN	Black Box	Subtest
DP1 Benchmark											
C77nv	3slowobs	129.363	660 981	695 014	43 894	16 011	8 276	3 901	3 179	4 237	0 194
C77nv	5slowobs	129 337	661 073	695 117	43 888	16 013	8 277	3 901	3 179	4 236	0 194
C77v	3slowobs	640 695	3273 642	3442 196	217 393	79 297	40 987	19 318	15 744	20 983	0 963
C77v	5slowobs	640.566	3274 100	3442 708	217 364	79 305	40 992	19 321	15 746	20 978	0 963
C77v	9slowobs	639 323	3265 452	3433 492	216 860	79 126	40 884	19 270	15 704	20 925	0 961
CFTnv	3fastobs	42.788	33.549	29 154	4 676	3 334	0 340	0 182	0 109	0 384	0 040
CFTnv	3slowobs	204.727	1046 057	1099 916	69 466	25 338	13 097	6 173	5 031	6 705	0 308
CFTnv	5slowobs	204.686	1046 203	1100 080	69 456	25 341	13 098	6 174	5 031	6 703	0 308
CFTv	3slowobs	592 135	3025 526	3181 305	200 917	73 287	37 881	17 854	14 551	19 393	0 890
CFTv	5slowobs	592.016	3025.950	3181 779	200 889	73 295	37 885	17 856	14 553	19.388	0 890
N815	3fastobs	2236.420	1753.525	1523 789	244 426	174 256	17 748	9 536	5.678	20 067	2 109
N815	5fastobs	2812.204	2897.339	2504 117	479 722	229 519	36.072	17.569	11 307	30 698	4 028
N815	9fastobs	7516.371	45475.860	47719 846	3209 903	1108 329	646.997	323.911	266.284	293 063	15.926
N865	3fastobs	253 133	198.476	172 473	27 666	19 723	2 009	1 079	0 643	2.271	0.239

Predicted Environment	Available Set	SMMBPF	SMDSEQT	SMMTDF	SMMNMO	SMMXP	SMMSTK	SMMTM	SMMTRIN	Black Box	Subtest
N865	5fastobs	318 304	327 940	283 432	54 298	25 978	4 083	1 989	1 280	3 475	0 456
N990	3fastobs	127 019	99 593	86 545	13 882	9 897	1 008	0 542	0 323	1 140	0 120
N990	3slowobs	607 741	3105 262	3265 146	206 212	75 218	38 879	18 325	14 935	19 904	0 913
N990	5fastobs	159.721	164 556	142 223	27 246	13 036	2.049	0 998	0.642	1 744	0 229
NV815	3fastobs	2251.928	1765 684	1534 356	246 121	175 464	17.871	9 602	5.718	20.206	2.124
NV815	5fastobs	2831 704	2917.430	2521 481	483 049	231.110	36.322	17 690	11.385	30.911	4.056
NVEnv	3fastobs	135.611	106 330	92 399	14 821	10 566	1 076	0 578	0.344	1 217	0.128
NVEnv	3slowobs	648.853	3315.325	3486.026	220 162	80 307	41.509	19 564	15.945	21.250	0.975
NVEnv	5fastobs	170 526	175.688	151 844	29 089	13 917	2.187	1 065	0.686	1.861	0 244
NVEv	3fastobs	105.651	82.839	71 986	11 547	8 232	0 838	0 451	0 268	0 948	0.100
NVEv	3slowobs	505.505	2582 887	2715.875	171.522	62 565	32.339	15 242	12.422	16 556	0.760
NVEv	5slowobs	505.403	2583.249	2716 280	171.499	62 571	32 342	15 244	12.423	16.552	0.760

Predicted Environment	Available Set	SMMBPF	SMDSEQT	SMMTDF	SMMNMO	SMMXP	SMMSTK	SMMTM	SMMTRIN	Black Box	Subtest
DP200 Benchmark											
C77nv	3slowobs	41 707	212 511	223 462	14 098	5 146	2 654	1 249	1 018	1 362	0 062
C77nv	5slowobs	41 716	212 624	223 582	14 102	5 148	2 655	1 250	1 018	1 362	0 062
C77v	3slowobs	9 269	47 228	49 661	3 133	1 144	0 590	0 278	0 226	0 303	0 014
C77v	5slowobs	9 271	47 253	49 688	3 134	1 144	0 590	0 278	0 226	0 303	0 014
C77v	9slowobs	9 271	47 252	49 687	3 134	1 144	0 590	0 278	0 226	0 303	0 014
CFTnv	3fastobs	451 189	603 372	528 269	103 930	37 786	7 149	3 184	2 106	5 737	0 513
CFTnv	3slowobs	120 752	615 269	646 975	40 816	14 898	7 683	3 617	2 947	3 943	0.180
CFTnv	5slowobs	120.777	615 594	647 323	40 829	14 905	7 687	3 619	2 948	3.944	0 180
CFTv	3slowobs	12 546	63.926	67.220	4 241	1.548	0 798	0 376	0 306	0 410	0.019
CFTv	5slowobs	12 549	63.960	67 256	4 242	1 549	0.799	0 376	0 306	0 410	0 019
N815	3fastobs	41171.935	55059 032	48205 639	9483 838	3448 011	652 405	290 579	192.149	523 471	46 769
N815	5fastobs	18608.320	45269.868	43470.851	5555 330	1972 211	666 103	297 014	222 248	375.291	28.935
N815	9fastobs	7909.553	48051.620	50427 267	3392 056	1168 931	684.075	342 616	281 679	309.625	16 825
N865	3fastobs	3464.193	4632.649	4056 007	797 967	290 115	54 893	24 449	16.167	44.045	3 935
N865	5fastobs	1565.698	3808 992	3657 624	467 424	165 941	56 046	24 991	18 700	31 577	2 435

Predicted Environment	Available Set	SMMBPF	SMDSEQT	SMMTDF	SMMNMO	SMMXP	SMMSTK	SMMTM	SMMTRIN	Black Box	Subtest
N990	3fastobs	1795 264	2400 798	2101 962	413 534	150 347	28 448	12 670	8 378	22 825	2.039
N990	3slowobs	480 469	2448 133	2574 293	162 407	59 279	30 571	14 391	11 725	15 691	0.718
N990	5fastobs	811 399	1973 951	1895 506	242 235	85 996	29 045	12 951	9 691	16 364	1.262
NV815	3fastobs	39635 995	53005 026	46407 303	9130 039	3319 381	628 067	279 739	184 980	503 942	45.024
NV815	5fastobs	17914 127	43581 052	41849 149	5348 086	1898 637	641 254	285 934	213.957	361 291	27 856
NVEnv	3fastobs	2191 856	2931 159	2566 307	504 888	183 561	34 732	15 469	10 229	27 868	2 490
NVEnv	3slowobs	586 609	2988 951	3142 981	198 284	72 374	37 324	17 570	14.316	19 157	0.876
NVEnv	5fastobs	990.645	2410.016	2314 243	295 747	104 994	35 461	15 812	11.832	19 979	1 540
NVEv	3fastobs	60.094	80 363	70 360	13 842	5 033	0 952	0 424	0 280	0 764	0 068
NVEv	3slowobs	16.083	81 948	86.171	5 436	1 984	1 023	0 482	0.392	0 525	0.024
NVEv	5slowobs	16.086	81.991	86 217	5 438	1 985	1 024	0 482	0.393	0.525	0 024

Predicted Environment	Available Set	SMMBPF	SMDSEQT	SMMTDF	SMMNMO	SMMXP	SMMSTK	SMMTM	SMMTRIN	Black Box	Subtest
Linpack Benchmark											
C77nv	3slowobs	60 649	303 717	319 432	20 002	7 356	3 737	1 745	1 420	1 945	0 087
C77nv	5slowobs	60 651	303 823	319 545	20 005	7 358	3 738	1 746	1 421	1 945	0 087
C77v	3slowobs	25 452	127 459	134 054	8 394	3 087	1 568	0 732	0 596	0 816	0 037
C77v	5slowobs	25 453	127.504	134.102	8 395	3.088	1 569	0 733	0 596	0 816	0 037
C77v	9slowobs	25 455	127 499	134 096	8 396	3 088	1 569	0 733	0 596	0 816	0 037
CFTnv	3fastobs	327 107	380 889	333 462	62 802	26 546	4 353	2 017	1 305	3.819	0 351
CFTnv	3slowobs	129.422	648 113	681 647	42 684	15.696	7 974	3 724	3 031	4 151	0.186
CFTnv	5slowobs	129 424	648 338	681 888	42 689	15 702	7 977	3 725	3 032	4 151	0.186
CFTv	3slowobs	39.296	196.786	206 968	12 960	4 766	2.421	1 131	0 920	1.260	0 056
CFTv	5slowobs	39.297	196.855	207 042	12 962	4.768	2.422	1 131	0 921	1.260	0.056
N815	3fastobs	31848 613	37085 086	32467 378	6114 672	2584 647	423 875	196 433	127 106	371 787	34 159
N815	5fastobs	21504 704	40976.574	38261 894	5641 135	2068 392	584 894	263 901	191 399	361.128	34 387
N815	9fastobs	10241.014	61933.911	64994 677	4363 295	1509 793	880 418	440 441	362 138	398 910	21 602
N865	3fastobs	3820 348	4448 480	3894 571	733 475	310 037	50.845	23 563	15 247	44.597	4.098
N865	5fastobs	2579 561	4915 277	4589 642	676 673	248 111	70 160	31 656	22 959	43 319	4.125

Predicted Environment	Available Set	SMMBPF	SMDSEQT	SMMTDF	SMMNMO	SMMXP	SMMSTK	SMMTM	SMMTRIN	Black Box	Subtest
N990	3fastobs	1577 426	1836 783	1608 073	302 853	128 015	20 994	9 729	6 295	18 414	1 692
N990	3slowobs	624 118	3125 434	3287 148	205 836	75 694	38 454	17 957	14 615	20.017	0.897
N990	5fastobs	1065 104	2029 524	1895 069	279 399	102 445	28 969	13 071	9 480	17 886	1.703
NV815	3fastobs	23593 652	27472 865	24052 037	4529 787	1914 723	314-009	145 518	94 161	275 422	25.305
NV815	5fastobs	15930.820	30355 704	28344 652	4178 988	1532 278	433 293	195 499	141 789	267.526	25.474
NVEnv	3fastobs	1434 579	1670 449	1462 450	275 427	116 422	19 093	8 848	5 725	16.747	1.539
NVEnv	3slowobs	567 600	2842 403	2989 473	187 196	68 839	34 972	16 330	13 291	18 205	0.815
NVEnv	5fastobs	968.651	1845 736	1723 457	254 098	93 168	26 346	11 887	8 621	16 267	1.549
NVEv	3fastobs	162.169	188 832	165.319	31 135	13 161	2.158	1 000	0 647	1.893	0.174
NVEv	3slowobs	64 163	321.313	337 938	21 161	7 782	3 953	1 846	1 502	2.058	0.092
NVEv	5slowobs	64.164	321 425	338 058	21 164	7 785	3.955	1 847	1 503	2.058	0.092

Predicted Environment	Available Set	SMMBPF	SMDSEQT	SMMTDF	SMMNMO	SMMXP	SMMSTK	SMMTM	SMMTRIN	Black Box	Subtest
SVV200 Benchmark											
C77nv	3slowobs	47 042	242 264	254 715	16 140	5 868	3 053	1 444	1 178	1 553	0 072
C77nv	5slowobs	47 053	242 393	254 853	16 145	5 871	3 055	1 445	1 178	1 554	0 072
C77v	3slowobs	6 176	31 806	33 441	2 119	0 770	0 401	0 190	0 155	0 204	0.009
C77v	5slowobs	6 177	31 823	33 459	2 120	0 771	0 401	0 190	0 155	0 204	0.009
C77v	9slowobs	6 178	31 822	33 458	2 120	0 771	0 401	0 190	0 155	0 204	0 009
CFTnv	3fastobs	356.010	516 201	452 074	90 866	30.359	6.225	2 718	1 818	4 777	0 419
CFTnv	3slowobs	116 211	598 487	629 246	39 872	14 496	7 543	3 568	2 909	3 837	0 178
CFTnv	5slowobs	116 239	598 805	629 585	39 883	14 504	7 546	3 569	2 910	3 838	0 178
CFTv	3slowobs	8.113	41 779	43 927	2 783	1 012	0 527	0 249	0.203	0.268	0.012
CFTv	5slowobs	8 114	41.802	43 950	2 784	1 012	0.527	0 249	0.203	0 268	0.012
N815	3fastobs	30324.965	43970.006	38507 691	7739 961	2585 984	530.250	231 506	154 817	406 937	35 681
N815	5fastobs	19728 029	48339 365	46342 624	6006 685	2094 542	711 048	316 324	236 592	400 051	30 999
N815	9fastobs	6758.817	41057.049	43086 808	2898 299	998 841	584 512	292 748	240 684	264 562	14 375
N865	3fastobs	3034.571	4400.008	3853 403	774 526	258 775	53.061	23 166	15 492	40.722	3.570
N865	5fastobs	1659.910	4067 259	3899 254	505 401	176 234	59 827	26 615	19 907	33 660	2 608

Predicted Environment	Available Set	SMMBPF	SMDSEQT	SMMTDF	SMMNMO	SMMXP	SMMSTK	SMMTM	SMMTRIN	Black Box	Subtest
N990	3fastobs	1542 763	2236 946	1959 054	393 766	131 560	26 976	11 778	7 876	20 703	1 815
N990	3slowobs	503.600	2593 531	2726 822	172 785	62 820	32 687	15 460	12 606	16 629	0.770
N990	5fastobs	860.222	2107 793	2020 727	261 916	91 331	31 005	13 793	10 316	17.444	1 352
NV815	3fastobs	34171 595	49547 468	43392 275	8721 752	2914 008	597 511	260 872	174.455	458 556	40 206
NV815	5fastobs	18992.065	46536.040	44613 789	5782 602	2016 404	684 522	304 523	227.765	385.127	29.843
NVEnv	3fastobs	1743 571	2528 109	2214 046	445 018	148 684	30 487	13 311	8.901	23 397	2 051
NVEnv	3slowobs	569 149	2931 107	3081 748	195 275	70 997	36 941	17 472	14.247	18 794	0 870
NVEnv	5fastobs	1050.254	2573 426	2467 126	319 776	111 506	37 854	16 840	12 595	21.297	1.650
NVEv	3fastobs	51.060	74 035	64 838	13 032	4 354	0 893	0.390	0 261	0 685	0 060
NVEv	3slowobs	16 667	85 837	90 248	5 719	2 079	1 082	0.512	0.417	0.550	0.025
NVEv	5slowobs	16.671	85.882	90 297	5.720	2 080	1.082	0.512	0.417	0.550	0.025

Predicted Environment	Available Set	SMMBPF	SMDSEQT	SMMTDF	SMMNMO	SMMXP	SMMSTK	SMMTM	SMMTRIN	Black Box	Subtest
VA200 Benchmark											
C77nv	3slowobs	30 243	153 424	161 338	10 159	3 716	1 909	0 897	0 731	0 983	0 045
C77nv	5slowobs	30.216	153 354	161 265	10 151	3 714	1 908	0 896	0.730	0 982	0 045
C77v	3slowobs	8 312	42 166	44 341	2 792	1 021	0 525	0 246	0 201	0 270	0 012
C77v	5slowobs	8 304	42 147	44 321	2 790	1 021	0 524	0 246	0.201	0 270	0 012
C77v	9slowobs	8 305	42 146	44 321	2 790	1 021	0 524	0 246	0 201	0.270	0 012
CFTnv	3fastobs	622.962	796 181	697 088	135 303	51 607	9 329	4.206	2 763	7.700	0.694
CFTnv	3slowobs	118 552	601 414	632 434	39 822	14 566	7 483	3 516	2 864	3.854	0 175
CFTnv	5slowobs	118 444	601 137	632 150	39 792	14 559	7 480	3 514	2.863	3 850	0 175
CFTv	3slowobs	12 028	61 016	64 163	4 040	1 478	0.759	0 357	0.291	0.391	0 018
CFTv	5slowobs	12.017	60.988	64 134	4.037	1 477	0 759	0 357	0.290	0.391	0 018
N815	3fastobs	60015 978	76703 809	67157 267	13035 015	4971 786	898 748	405 249	266 176	741 829	66 867
N815	5fastobs	19330.026	48597 721	46894 125	5813 392	2080 470	719 540	321 054	241.285	400 354	30.072
N815	9fastobs	8397 065	50906 143	53423 308	3588 957	1239 435	724 007	362 377	297 932	327 875	17 787
N865	3fastobs	6720 242	8588 849	7519 882	1459 586	556 712	100 637	45 377	29 805	83.066	7 487
N865	5fastobs	2164.464	5441 692	5250 933	650 950	232 959	80 570	35 950	27 018	44 829	3 367

Predicted Environment	Available Set	SMMBPF	SMDSEQT	SMMTDF	SMMNMO	SMMXP	SMMSTK	SMMTM	SMMTRIN	Black Box	Subtest
N990	3fastobs	3471.396	4436 640	3884 457	753 961	287 574	51 985	23 440	15.396	42 908	3 868
N990	3slowobs	660 618	3351 321	3524 179	221 904	81 170	41 701	19 591	15 959	21 475	0 977
N990	5fastobs	1118 072	2810 950	2712 412	336 254	120 337	41 619	18 570	13 956	23 157	1 739
NV815	3fastobs	54316 298	69419 296	60779 383	11797.088	4499 619	813 395	366 763	240 897	671 378	60 516
NV815	5fastobs	17494.265	43982 426	42440 619	5261 298	1882 889	651 206	290 564	218 371	362 332	27 216
NVEnv	3fastobs	4181 452	5344 132	4679 002	908 180	346 396	62 618	28 235	18 545	51.685	4.659
NVEnv	3slowobs	795 743	4036 816	4245 032	267 293	97 773	50 231	23 598	19 223	25 868	1.177
NVEnv	5fastobs	1346.768	3385 916	3267 222	405 033	144 951	50 132	22 369	16 811	27 894	2 095
NVEv	3fastobs	65 462	83 664	73 251	14 218	5 423	0 980	0.442	0 290	0 809	0 073
NVEv	3slowobs	12.458	63 198	66 457	4 185	1 531	0 786	0.369	0 301	0.405	0.018
NVEv	5slowobs	12.446	63 169	66 428	4 181	1 530	0 786	0 369	0.301	0.405	0 018

APPENDIX D

DEGREE OF PREDICTABILITY OF THE PREDICTABILITY MODEL FOR ALL APPLICATIONS

Environment Predicted	Available Set	SMMBPF	SMDSEQT	SMMTDF	SMMNMO	SMMXP	SMMSTK	SMMTM	SMMTRIN	Black Box	Subtest
DP1 Benchmark											
C77nv	3slowobs	14%	243%	311%	25%	59%	246%	286%	361%	149%	35%
C77nv	5slowobs	14%	243%	311%	25%	59%	246%	286%	361%	149%	35%
C77v	5slowobs	480%	4461%	5627%	2182%	739%	5364%	4517%	6516%	2806%	835%
C77v	9slowobs	479%	4449%	5611%	2176%	737%	5350%	4505%	6498%	2798%	833%
C77v	3slowobs	481%	4460%	5626%	2182%	739%	5363%	4516%	6515%	2806%	835%
CFTnv	3slowobs	36%	121%	136%	37%	41%	81%	92%	104%	79%	27%
CFTnv	5slowobs	36%	121%	136%	37%	41%	81%	92%	104%	79%	27%
CFTnv	3fastobs	72%	93%	94%	91%	82%	95%	94%	96%	90%	83%
CFTv	3slowobs	355%	3070%	3639%	1573%	718%	4388%	3492%	4935%	1380%	667%
CFTv	5slowobs	355%	3070%	3639%	1573%	718%	4389%	3493%	4935%	1380%	667%
N815	9fastobs	46%	29%	29%	21%	28%	10%	2%	4%	28%	1%
N815	3fastobs	84%	97%	98%	94%	89%	98%	97%	98%	95%	87%
N815	5fastobs	80%	95%	96%	88%	85%	95%	94%	96%	92%	75%
N865	5fastobs	80%	95%	96%	87%	87%	95%	95%	96%	92%	72%
N865	3fastobs	84%	97%	98%	93%	90%	98%	97%	98%	95%	85%

Environment Predicted	Available Set	SMMBPF	SMDSEQT	SMMTDF	SMMNMO	SMMXP	SMMSTK	SMMTM	SMMTRIN	Black Box	Subtest
N990	5fastobs	78%	95%	96%	87%	86%	95%	95%	96%	91%	76%
N990	3slowobs	15%	10%	10%	2%	18%	12%	11%	12%	3%	3%
N990	3fastobs	82%	97%	98%	93%	89%	98%	97%	98%	94%	87%
NV815	5fastobs	62%	94%	95%	85%	79%	94%	95%	96%	90%	75%
NV815	3fastobs	70%	96%	97%	92%	84%	97%	97%	98%	93%	87%
NVEnv	3slowobs	37%	2%	1%	10%	16%	6%	5%	7%	13%	16%
NVEnv	3fastobs	71%	97%	97%	93%	85%	97%	97%	98%	94%	89%
NVEnv	5fastobs	64%	95%	96%	85%	80%	94%	94%	95%	90%	79%
NVEv	3fastobs	40%	37%	20%	71%	38%	50%	45%	38%	44%	75%
NVEv	5slowobs	186%	1851%	2917%	333%	372%	1810%	1759%	2773%	872%	90%
NVEv	3slowobs	186%	1851%	2917%	333%	372%	1810%	1759%	2773%	872%	90%

Environment Predicted	Available Set	SMMBPF	SMDSEQT	SMMTDF	SMMNMO	SMMXP	SMMSTK	SMMTM	SMMTRIN	Black Box	Subtest
DP200 Benchmark											
C77nv	3slowobs	63%	10%	32%	60%	49%	11%	24%	48%	20%	57%
C77nv	5slowobs	63%	10%	32%	60%	49%	11%	24%	48%	20%	57%
C77v	3slowobs	92%	34%	17%	67%	88%	21%	34%	5%	58%	87%
C77v	5slowobs	92%	34%	17%	67%	88%	21%	34%	5%	58%	87%
C77v	9slowobs	92%	34%	17%	67%	88%	21%	34%	5%	58%	87%
CFTnv	3fastobs	200%	28%	13%	104%	110%	1%	1%	15%	53%	111%
CFTnv	3slowobs	20%	30%	39%	20%	17%	6%	13%	19%	5%	26%
CFTnv	5slowobs	20%	30%	39%	20%	17%	6%	13%	20%	6%	26%
CFTv	3slowobs	90%	33%	21%	65%	83%	5%	24%	6%	69%	84%
CFTv	5slowobs	90%	33%	21%	65%	83%	5%	24%	6%	69%	84%
N815	3fastobs	195%	14%	28%	135%	124%	9%	9%	25%	29%	190%
N815	5fastobs	33%	29%	35%	38%	28%	7%	7%	14%	8%	79%
N815	9fastobs	43%	25%	25%	16%	24%	5%	8%	10%	24%	4%
N865	3fastobs	121%	34%	45%	93%	45%	38%	38%	51%	0%	142%
N865	5fastobs	0%	46%	50%	13%	17%	37%	37%	43%	29%	50%

Environment Predicted	Available Set	SMMBPF	SMDSEQT	SMMTDF	SMMNMO	SMMXP	SMMSTK	SMMTM	SMMTRIN	Black Box	Subtest
N990	3fastobs	150%	30%	42%	97%	63%	35%	38%	51%	18%	116%
N990	3slowobs	33%	29%	29%	23%	36%	31%	30%	31%	19%	24%
N990	5fastobs	13%	43%	48%	15%	7%	34%	37%	43%	16%	34%
NV815	3fastobs	427%	15%	4%	181%	198%	4%	15%	32%	70%	179%
NV815	5fastobs	138%	5%	13%	65%	70%	2%	13%	21%	22%	72%
NVEnv	3fastobs	362%	10%	26%	153%	166%	12%	17%	31%	49%	114%
NVEnv	3slowobs	24%	8%	9%	1%	5%	5%	6%	4%	2%	25%
NVEnv	5fastobs	109%	26%	33%	48%	52%	10%	15%	21%	7%	32%
NVEv	3fastobs	66%	39%	22%	65%	62%	44%	48%	35%	55%	83%
NVEv	3slowobs	91%	38%	4%	86%	85%	40%	41%	9%	69%	94%
NVEv	5slowobs	91%	38%	4%	86%	85%	40%	41%	9%	69%	94%

Environment Predicted	Available Set	SMMBPF	SMDSEQT	SMMTDF	SMMNMO	SMMXP	SMMSTK	SMMTM	SMMTRIN	Black Box	Subtest
Linpack Benchmark											
C77nv	3slowobs	47%	57%	89%	43%	27%	56%	73%	106%	14%	39%
C77nv	5slowobs	47%	58%	89%	43%	27%	56%	73%	106%	14%	39%
C77v	3slowobs	77%	78%	123%	12%	67%	109%	75%	150%	13%	65%
C77v	5slowobs	77%	78%	123%	12%	67%	109%	75%	151%	13%	64%
C77v	9slowobs	77%	78%	123%	12%	67%	109%	75%	151%	13%	64%
CFTnv	3fastobs	117%	19%	28%	24%	47%	40%	37%	47%	2%	44%
CFTnv	3slowobs	14%	37%	46%	16%	13%	10%	16%	23%	11%	23%
CFTnv	5slowobs	14%	37%	46%	16%	13%	10%	16%	23%	11%	23%
CFTv	3slowobs	70%	106%	143%	8%	47%	187%	127%	218%	4%	51%
CFTv	5slowobs	70%	106%	143%	8%	47%	187%	128%	219%	4%	51%
N815	3fastobs	128%	42%	52%	51%	68%	41%	38%	51%	9%	112%
N815	5fastobs	54%	36%	43%	40%	34%	19%	17%	26%	11%	113%
N815	9fastobs	27%	3%	3%	8%	2%	23%	38%	41%	2%	34%
N865	3fastobs	144%	37%	47%	78%	55%	43%	40%	54%	1%	152%
N865	5fastobs	65%	30%	38%	64%	24%	21%	20%	30%	2%	154%

Environment Predicted	Available Set	SMMBPF	SMDSEQT	SMMTDF	SMMNMO	SMMXP	SMMSTK	SMMTM	SMMTRIN	Black Box	Subtest
N990	3fastobs	120%	47%	56%	44%	39%	52%	53%	63%	5%	79%
N990	3slowobs	13%	9%	9%	2%	18%	13%	12%	14%	3%	5%
N990	5fastobs	48%	41%	48%	33%	11%	34%	36%	44%	8%	80%
NV815	3fastobs	214%	40%	50%	39%	72%	52%	56%	65%	7%	57%
NV815	5fastobs	112%	34%	41%	29%	38%	34%	41%	48%	10%	58%
NVEnv	3fastobs	203%	49%	58%	38%	69%	51%	53%	62%	11%	32%
NVEnv	3slowobs	20%	13%	13%	6%	0%	11%	13%	11%	3%	30%
NVEnv	5fastobs	104%	43%	50%	27%	35%	33%	36%	42%	13%	33%
NVEv	3fastobs	8%	43%	84%	21%	1%	27%	22%	50%	11%	56%
NVEv	3slowobs	64%	143%	275%	47%	41%	134%	125%	247%	21%	77%
NVEv	5slowobs	64%	143%	275%	47%	41%	134%	125%	248%	21%	77%

Environment Predicted	Available Set	SMMBPF	SMDSEQT	SMMTDF	SMMNMO	SMMXP	SMMSTK	SMMTM	SMMTRIN	Black Box	Subtest
SVV200 Benchmark											
C77nv	3slowobs	59%	26%	51%	54%	42%	28%	43%	71%	9%	50%
C77nv	5slowobs	59%	26%	51%	54%	42%	28%	43%	71%	9%	50%
C77v	3slowobs	94%	56%	44%	78%	92%	47%	55%	35%	72%	91%
C77v	5slowobs	94%	56%	44%	78%	92%	47%	55%	35%	72%	91%
C77v	9slowobs	94%	56%	44%	78%	92%	47%	55%	35%	72%	91%
CFTnv	3fastobs	137%	9%	3%	79%	68%	14%	15%	26%	28%	72%
CFTnv	3slowobs	23%	27%	35%	22%	20%	4%	11%	18%	3%	27%
CFTnv	5slowobs	23%	27%	35%	22%	20%	4%	11%	18%	3%	27%
CFTv	3slowobs	94%	56%	48%	77%	89%	38%	50%	30%	80%	89%
CFTv	5slowobs	94%	56%	48%	77%	89%	38%	50%	30%	80%	89%
N815	3fastobs	117%	31%	43%	92%	68%	26%	27%	40%	0%	121%
N815	5fastobs	41%	24%	31%	49%	36%	1%	1%	8%	2%	92%
N815	9fastobs	52%	36%	36%	28%	35%	19%	8%	6%	35%	11%
N865	3fastobs	94%	37%	48%	88%	30%	40%	41%	53%	8%	120%
N865	5fastobs	6%	42%	47%	22%	12%	32%	33%	40%	24%	61%

Environment Predicted	Available Set	SMMBPF	SMDSEQT	SMMTDF	SMMNMO	SMMXP	SMMSTK	SMMTM	SMMTRIN	Black Box	Subtest
N990	3fastobs	115%	35%	46%	87%	43%	39%	43%	54%	7%	92%
N990	3slowobs	30%	25%	25%	18%	32%	26%	25%	26%	14%	19%
N990	5fastobs	20%	39%	44%	25%	1%	30%	33%	39%	10%	43%
NV815	3fastobs	355%	8%	10%	168%	162%	9%	21%	35%	55%	149%
NV815	5fastobs	153%	1%	7%	78%	81%	4%	7%	16%	30%	85%
NVEnv	3fastobs	268%	22%	36%	123%	115%	22%	29%	40%	25%	76%
NVEnv	3slowobs	20%	10%	11%	2%	3%	6%	7%	5%	0%	25%
NVEnv	5fastobs	122%	21%	28%	60%	61%	4%	10%	16%	14%	42%
NVEv	3fastobs	71%	44%	28%	67%	67%	47%	52%	40%	60%	85%
NVEv	3slowobs	91%	35%	0%	86%	84%	36%	38%	4%	68%	94%
NVEv	5slowobs	91%	35%	0%	86%	84%	36%	38%	3%	68%	94%

Environment Predicted	Available Set	SMMBPF	SMDSEQT	SMMTDF	SMMNMO	SMMXP	SMMSTK	SMMTM	SMMTRIN	Black Box	Subtest
VA200 Benchmark											
C77nv	3slowobs	73%	20%	5%	71%	63%	20%	11%	6%	42%	69%
C77nv	5slowobs	73%	20%	5%	71%	63%	20%	11%	6%	42%	69%
C77v	3slowobs	92%	41%	26%	71%	89%	30%	41%	16%	63%	88%
C77v	5slowobs	92%	41%	26%	71%	89%	30%	41%	16%	63%	88%
C77v	9slowobs	92%	41%	26%	71%	89%	30%	41%	16%	63%	88%
CFTnv	3fastobs	314%	68%	50%	166%	186%	29%	31%	12%	106%	186%
CFTnv	3slowobs	21%	27%	36%	22%	19%	4%	9%	16%	3%	28%
CFTnv	5slowobs	21%	27%	36%	22%	19%	4%	9%	16%	3%	28%
CFTv	3slowobs	91%	36%	25%	66%	84%	10%	28%	1%	70%	85%
CFTv	5slowobs	91%	36%	25%	66%	84%	10%	28%	0%	70%	85%
N815	3fastobs	330%	20%	0%	223%	223%	25%	27%	4%	82%	314%
N815	5fastobs	39%	24%	30%	44%	35%	0%	1%	6%	2%	86%
N815	9fastobs	40%	20%	20%	11%	20%	1%	14%	16%	19%	10%
N865	3fastobs	329%	23%	2%	254%	179%	14%	15%	9%	88%	361%
N865	5fastobs	38%	22%	29%	58%	17%	9%	9%	18%	1%	107%

Environment Predicted	Available Set	SMMBPF	SMDSEQT	SMMTDF	SMMNMO	SMMXP	SMMSTK	SMMTM	SMMTRIN	Black Box	Subtest
N990	3fastobs	384%	29%	7%	259%	212%	18%	14%	9%	121%	309%
N990	3slowobs	8%	3%	2%	6%	12%	5%	5%	6%	11%	3%
N990	5fastobs	56%	18%	25%	60%	30%	5%	9%	18%	19%	84%
NV815	3fastobs	623%	51%	26%	263%	304%	24%	12%	11%	127%	275%
NV815	5fastobs	133%	4%	12%	62%	69%	1%	12%	19%	22%	68%
NVEnv	3fastobs	782%	64%	36%	354%	402%	59%	51%	24%	176%	300%
NVEnv	3slowobs	68%	24%	23%	34%	42%	28%	26%	29%	38%	1%
NVEnv	5fastobs	184%	4%	5%	103%	110%	28%	20%	13%	49%	80%
NVEv	3fastobs	63%	37%	19%	64%	59%	42%	46%	33%	52%	82%
NVEv	3slowobs	93%	52%	26%	89%	88%	54%	55%	30%	76%	95%
NVEv	5slowobs	93%	52%	26%	89%	88%	54%	55%	30%	76%	95%

VITA 2

Nathan O. Langston

Candidate for the Degree of

Master of Science

Thesis: THE EVALUATION OF A MODEL TO PREDICT THE PERFORMANCE OF AN APPLICATION PROGRAM IN AN UNKNOWN ENVIRONMENT

Major Field: Computer Science

Biographical:

Personal Data: Born in Carriboo, Maine, June 14, 1958, the son of Bill and Anne Langston.

Education: Received Bachelor of Science Degrees in Mathematics and History from Oklahoma Christian College in May, 1981; Completed requirements for the Master of Science degree at Oklahoma State University in December, 1991.

Professional Experience: Supervisor, Application Technology Division, Computer Information Systems, Conoco Inc., Ponca City, Oklahoma, May 1981 to present, supporting PC and mini-computer operating systems.