LA-6456-MS
Informal Report

C.3

UC-32

Issued: December 1976

CRAY-1 Evaluation

Final Report

by

T. W. Keller

los alamos
scientific laboratory
of the University of California
LOS ALAMOS, NEW MEXICO 87545

An Affirmative Action/Equal Opportunity Employer

UNITED STATES
ENERGY RESEARCH AND DEVELOPMENT ADMINISTRATION
CONTRACT W-7405-ENG. 36

Printed in the United States of America. Available from
National Technical Information Service
U.S. Department of Commerce
5285 Port Royal Road
Springfield, VA 22161
Price: Printed Copy $5.50 Microfiche $3.00

ACKNOWLEDGEMENTS

Much of Chapter III is drawn from reports by P. Iwanchuk and L. Rudsinski, who performed the scalar performance evaluation with the able assistance of T. Montoya and J. Miller. J. Moore performed the disk studies and much of Chapter VI is extracted from his reports. The discussion on Computation and I/O Interference in the same chapter closely follows a report by J. C. Browne (Visiting Staff Member). R. Tanner edited the manuscript with contributions from many readers and with the aid of the word processing section of group C-4. The vector performance studies (Chapter IV) were conducted by A. McKnight. Data reported in Chapter V was collected by N. Nagy and D. Buckner. T. Jordan, K. Witte, and A. Marusak performed the preliminary scalar performance test referenced in Chapter VIII. Many people reviewed the coding of the scalar performance modules (Chapter III), with special acknowledgement due D. Lindsay of FEDSIM. Discussions with B. Buzbee, J. Worlton, and F. Dorr helped define the scope of the evaluation under the time constraints involved.

(This page was not bound with the report as the result of an error on the author's part. He apologizes to all the above for this oversight.)

TABLE OF CONTENTS

SIGNATURES

[illegible]

Ronald S. Schwartz, Director
Office of ADP Management
Energy Research and Development Administration

[illegible]

Ronald Bartell, Deputy Assistant Director for
Program Analysis and Budget
Division of Military Application
Energy Research and Development Administration

Fred W. Dorr

Fred W. Dorr, Division Leader
Computer Science and Services Division
Los Alamos Scientific Laboratory

SECTION I

EXECUTIVE SUMMARY

The performance evaluation of the CRAY-1 computer was structured to determine if the CRAY-1 meets the minimum performance standards set forth by the Los Alamos Scientific Laboratory (LASL) and the Energy Research and Development Administration (ERDA) to qualify the machine for further consideration for procurement.

The performance standards are divided into specific qualification criteria in three main areas: scalar performance, vector performance and reliability. The qualification criteria are summarized in Table I-1. The final Evaluation Plan, including precise definitions of the qualification criteria, is presented in Appendix A of this document.

It was impossible to convert large segments of the LASL computing workload to the CRAY-1 because programs to be run on the machine would require assembly language coding. Thus, for the scalar test, a sampling scheme was adopted that selected small computational kernels to be run on both the CDC 7600 and the CRAY-1. Kernels were drawn from a program by a method that weighted the probability of drawing a specific kernel by its contribution to the total execution time of the program. The sampling process was defined to a level of detail that eliminated the chances of biasing the selection toward either machine. By statistical methods it was possible to establish a test of the hypothesis that the CRAY-1 (in scalar mode) is greater than two times faster than the CDC 7600 with 90 percent confidence for any sampled program. To assure that the code kernels were representative of the potential LASL workload for the CRAY-1, the code kernels were drawn from the actual programs expected to comprise the eventual Class VI workload. Only programs consuming greater than one CDC 7600 hour per run and requiring more than one run per week were considered as potential workload candidates.

A second workload for the CRAY-1 was established from a sample that included frequently run codes not expected to be included in the immediate LASL workload for the machine. This was done as an attempt to establish a performance index of the machine based upon a more general workload, and one that might be more representative of computing throughout ERDA.

In order to eliminate the impact of compiler efficiency, kernels were coded as efficiently as was feasible in both CRAY-1 assembly language and CDC 7600 assembly language.

It was not possible to rigorously establish "representative" kernels for testing the vector performance of the CRAY-1. This was because none of the existing codes comprising the workload had been converted for vector operations, and such conversion efforts were outside the evaluation's time frame.

At the risk of oversimplifying, the vector computational speed of the CRAY-1 relative to the CDC 7600 is a function of both vector length and complexity of the vectorized arithmetic function. Relative performance of the CRAY-1 increases with vector length and complexity of operation. Performance criteria using vector lengths of 20, 100 and 500 were established, and the complexity of vector operations remained to be chosen. The simplest expressions, such as R=A+B, result in the lowest relative performance. Relative performance increases with greater numbers of different operators and operands (increasing complexity), as this allows increased overlap of functional units and chaining to occur. Chaining refers to an increase in parallelism resulting from the ability of the machine to store the result of a computation in a vector register while the result is re-entered as an operand to another vector computation in the same clock period. Thus, two or more primitive vector operations may be "chained" together. The more complex the evaluated expression, the greater the likelihood that chaining can occur. The Applications Support and Research Group of the Computer Science and Services Division at LASL was asked to furnish vector kernels that, in their judgement, represented common vector operations that user codes would perform on the CRAY-1. Five expressions of medium complexity, such as R=A*B+C, were chosen to evaluate the machine's vector performance as a function of vector length. The average of the five performance ratios was chosen to compare against the qualification criteria. Each vector kernel was coded as efficiently as was feasible in assembly language for each machine. In particular, the CDC 7600 kernels were coded as "in-stack" loops--a scheme that pushes the CDC 7600 toward its theoretical maximum performance for these algorithms.

The reliability of the CRAY-1 was evaluated by establishing a reliability test code to run on the machine for long periods of time. This "exerciser" was designed to utilize as many different hardware units of the machine as possible, at as rapid a rate as possible, for extended periods. The exerciser underwent several evolutionary stages toward this goal. The latest version of the exerciser accesses the machine's memory at a sustained rate several times greater than that plausible for a production workload. The reliability figures were determined for contiguous 20-workday periods in order to "smooth" short-term fluctuations in reliability.

In addition to the tests against the qualification criteria outlined above, the evaluation also investigated the CRAY-1 disk system performance and made other miscellaneous studies. For sake of brevity these studies will not be discussed in the Executive Summary.

Clearly, many aspects of the CRAY-1's performance were evaluated, some in considerable detail. An impartial study was accomplished despite the constraints of a primitive CRAY-1 operating system, the absence of a Fortran compiler, the relatively short evaluation period, and the necessity of agreement by LASL, ERDA, and the Federal Computer Performance Evaluation and Simulation Center (FEDSIM) upon an evaluation plan. Rigorously defensible results were obtained by adopting methods of known accuracy wherever possible. Although the constraints confined the scope of the evaluation, they did not hinder the objectivity nor accuracy of its results.

Results

A brief summary of the evaluation results is presented in Table I-1.

Scalar. Timing results for the scalar kernels are summarized in Table III-6 of Section III. On the basis of these results the hypothesis that the CRAY-1 is at least two times faster than the CDC 7600 for scalar kernels was satisfied in all tests.

Vector. Results of the vector kernel timings are summarized in Table I-1. The machine met the vector performance qualification criteria for the three vector lengths.

Reliability. The machine met the reliability criteria for many reported 20-day periods. The Mean-Time-To-Failure (MTTF) fluctuated from a low of approximately 2.5 hours to a high of approximately 7.5 hours during the six-month period. No trend was observed.

Approximately 89 percent of all machine failures during the evaluation were memory parity errors. If one assumes that all memory errors were correctable single-bit errors, then installation of single-bit memory error correction would have resulted in an increase in MTTF by a factor of nine. Such an increase would result in extremely good reliability for a machine of this complexity.

The conclusion of the evaluation is that the CRAY-1 satisfies the threshold performance criteria in all categories. FEDSIM, in a separate report to be issued, concurs in this conclusion.

Table I-1. Summary of Qualification Criteria and Evaluation Results.

A. Qualification Criteria

Scalar Performance.	
Scalar operations	$\geq$2 x CDC 7600 in speed
Vector Performance.	
Short vector operations	$\geq$3 x CDC 7600 in speed
Medium vector operations	$\geq$4 x CDC 7600 in speed
Long vector operations	$\geq$5 x CDC 7600 in speed
Reliability.	
Mean-Time-To-Failure	$\geq$4 hrs.
Mean-Time-To-Repair	$\leq$1 hr.
System availability	$\geq$0.8

B. Results

Scalar. The hypothesis that the CRAY-1 is $\geq$2 x the CDC 7600 in speed tests as true for each of the three codes comprising the LASL "applications" workload. The hypothesis tests as true for a more general workload consisting of five codes.

Vector. Average CRAY-1/CDC 7600 speed ratio of five vector functions by vector length:

Vector length 20 (short)	3.30
Vector length 100 (medium)	4.50
Vector length 500 (long)	5.12

Reliability. For the period 5-14-76 to 6-11-76:

Mean-Time-To-Failure	7.08 hrs.
Mean-Time-To-Repair	0.38 hr.
System Availability	0.950

SECTION II

INTRODUCTION

The growing complexity of the programmatic efforts at the Los Alamos Scientific Laboratory (LASL) in weapons design, laser fusion and isotope separation, magnetic fusion energy research, reactor safety, and other basic and exploratory research requires calculations that are beyond the capability of the fastest computer currently installed in the LASL Central Computing Facility--the CDC 7600. Thus, LASL needs a computer that can meet the growing requirements for computational speed. For this reason, LASL, in conjunction with the Energy Research and Development Administration (ERDA) and the Federal Computer Performance Evaluation and Simulation Center (FEDSIM), undertook to evaluate a new class of scientific computer system. This class has been identified by ERDA as "Class VI"--one that is capable of executing 20 to 60 million floating point operations per second.

Context of Evaluation

Because computers of this class are necessarily pushing the state of the art of both computer architecture and electronic technology, there have been worries expressed in ERDA and Congress that the usual procurement procedures did not afford adequate protection to the Government against failure of these machines to fulfill their specifications for reliability and performance. The basic reason for this is that Class VI computers are very complex, and considerable time is required to convert previous application programs so that an adequate evaluation of the product can be assured. By the time reliability and performance evaluations can be conducted, the usual protections for the Government have elapsed.

To avoid a repetition of previous ERDA experiences with similar computers, a plan was designed that offered vendors an equal opportunity to compete and yet provided a large measure of protection for the Government.

The plan to implement these desired policies of competition and protection consisted of the following steps.

a. Offer all potential vendors an opportunity to have their computers evaluated with respect to stated performance criteria within approximately a six-month time scale. A letter was sent on 2-26-76 by ERDA to a number of vendors.

b. Determine the set of computers to be evaluated. In the words of the letter to the vendors, "....a demonstration and evaluation of machine capabilities must precede a decision to acquire a new class of computers." Thus, all vendors were given an opportunity to have their products evaluated; however, only those with a product available for immediate evaluation which met the minimum evaluation criteria would qualify to provide the equipment for later acquisition at LASL.

Responses were received from two vendors--Control Data Corporation (CDC) and Cray Research, Incorporated (CRI). The CDC response did not offer a computer for immediate evaluation; the proposed product was not scheduled for availability until about eighteen months after the beginning of the evaluation period. Thus, the policy of "evaluation before acquisition" could not be implemented with respect to the CDC proposal, and the CDC proposal was judged non-responsive to LASL's requirements.

c. Evaluate the products proposed. The proposal by CRI was consistent with the specified criteria and that computer was evaluated, beginning April 1, 1976. The evaluation was a cooperative effort between LASL, ERDA and FEDSIM and conformed to the performance criteria specified to vendors.

d. On the basis of evaluation results (c), determine whether to pursue a procurement action.

In summary, a plan was implemented that offered vendors an equal opportunity to compete, yet afforded the Government extra protection in an area of high technological risk.

Qualification Criteria

The performance standards necessary to qualify a computer for procurement were established in the ERDA letter to vendors. Qualification criteria for the CRAY-1, incorporating these standards, are formalized in the Evaluation Plan (Appendix A of this document). An informal description of the criteria follows. The criteria are divided into three general areas: scalar operations, vector operations and reliability.

The scalar operations criterion is that by methods discussed in Section III and formalized in the Evaluation Plan, the hypothesis that the CRAY-1 is at least two times faster than the CDC 7600 for two specified workloads must be satisfied.

The vector operations criteria are defined for three vector lengths. The criteria are that the CRAY-1 be capable of executing vector operations at a rate at least 3, 4 and 5 times that at which the CDC 7600 can perform the equivalent functions for vectors of lengths 20, 100 and 500, respectively.

Reliability is determined by three metrics: System Availability (SA), Mean-Time-to-Failure (MTTF) and Mean-Time-to-Repair (MTTR). Informally, SA is the fraction of time the system is available for use, MTTF is the expected time to system failure, and MTTR is the expected "down-time" interval. The reliability criteria are that MTTF be at least 4 hours, MTTR be at most 1 hour, and SA be at least 0.8. Furthermore, the three criteria must collectively be met during any contiguous 20-workday period.

These criteria are summarized in Table I-1 of the Executive Summary (Section I) of this report.

It now appears that there is an existing computer system that meets LASL Class VI computing requirements -- the Cray Research, CRAY-1. The CRAY-1 was installed at LASL for six months for evaluation against the threshold performance criteria specified by LASL and ERDA to qualify the machine for procurement. The purpose of this document is to describe the approach taken by LASL, ERDA and FEDSIM in evaluating the CRAY-1 and the results of the evaluation. This document was preceded by a preliminary version of this report ("Interim Report of the CRAY-1 Evaluation") issued July 1, which consisted of the CRAY-1 Evaluation Plan and gave results of the evaluation to that date.

SECTION III

SCALAR PERFORMANCE

Introduction and Approach

The scalar operations performance criterion is that the CRAY-1 must perform scalar operations at a rate at least twice that of the CDC 7600 for each of two workloads. The approach to testing the machine against this criterion is specified in detail in the Evaluation Plan (Appendix A). Some of the constraints and requirements which shaped this approach are described here.

Since no compiler was available on the CRAY-1, any code implemented on the machine was written in assembly language. The decision was made to implement "optimal" code on both machines, instead of attempting to emulate compiler-generated code. The fundamental reason for this approach was that the CRAY-1 and CDC 7600 have different architectures. For example, the B and T programmer-controlled register caches on the CRAY-1 have no equivalent on the CDC 7600. Attempts to emulate compilers would raise such questions as: "How well will a (hypothetical) CRAY-1 compiler take advantage of this cache for the storage of temporaries?" It soon became obvious that no satisfactory solution to the problem of functional differences existed where the problem was compounded by questions of compiler "intelligence" and convention.

Another reason that code optimization was undertaken was because "optimal" is a yardstick for comparison. For very short modules it is reasonable to expect that careful coding, cross-checking by other programmers, the use of software tools that test efficiencies, etc., will result in nearly optimal code. The metric for optimality is simple. If two approaches are considered, the approach resulting in the least execution time is more optimal. When emulating compiler-generated code, any question of approach becomes a debate between compiler implementation philosophies.

The next decision to be made was whether Input/Output operations (I/O) should be included in the analysis or whether computational kernels should be compared alone. A complete analysis would require inclusion of the properties of CPU/IO overlap. This would necessitate a complete analysis of the program's CPU burst, I/O burst profile. Unfortunately, tools did not exist for such complete analysis, nor could they be constructed within the time frame of the evaluation. Furthermore, at the time the evaluation was started, the complete operating characteristics of the eventual CRAY-1 I/O subsystem were not well-determined. The only manner in which to gather such statistics would be extensive modifications in the running codes and operating system. This approach was not considered tractable. A modeling approach was suggested, but since only a trivial I/O system was available, it was not thought that models could be validated with the necessary rigor. Thus the decision was made to limit the comparison to the machines' mainframe hardware. An analysis of the CRAY-1 I/O subsystem was formulated to determine if any pathologies existed in the subsystem.

The next decisions facing the evaluation design committee were 1) what codes would be analyzed and 2) how would kernels be selected from these codes to run on the machine? A workload determination was considered essential for selecting "representative" codes. Since the workload of the Class VI computer would consist of long-running codes, the manner of the selection of criteria defining the potential workload for the machine was relatively straightforward. These criteria are put forth in the Evaluation Plan. A more ticklish question was how to draw "representative" kernels from the codes. Since kernels would have to be written in assembly language for both machines and "optimized", the amount of code that could be implemented was obviously small. The general question of "representativeness" is still a very active area of research in computer performance prediction. Many opinions were put forth by the evaluation design committee. The constraint that blocked most approaches was the desire to attain rigorously defensible estimates. There is no metric of "representativeness" that can be applied. Error bounds for performance estimates for workloads attainable from kernels are usually impossible to attain. Thus a new approach was considered necessary.

It was decided to adopt a sampling scheme to which proven statistical methods could be applied in order to attain error bounds for the performance estimates. The sequential sampling scheme that was formulated is described in the Statistics Addendum. Informally, the approach assumes that any code can be partitioned into small, nonoverlapping code segments. If we ignore I/O, then the total execution time for any code will be the sum of the total execution times for the segments. What we desire to estimate is the ratio of the CDC 7600 execution time to CRAY-1 execution time for the code. If this speed ratio is at least two then the CRAY-1 meets the scalar qualification criterion for that code. The sampling scheme draws a number of these segments from the code. These computational kernels are then implemented in assembly language on each machine and timed. The speed ratios of these kernels are thus related to the actual speed ratio for the code. The larger the sample size from a particular code, the more confident we are that the sampled speed ratio is close to the code's actual speed ratio.

Unfortunately, it became apparent that the sample size necessary to estimate a code's speed ratio with adequate confidence was too large to be tractable. However, the hypothesis that the CRAY-1 is greater than twice as fast as the CDC 7600 (the qualification criterion) can be tested with a tractable sample size. A procedure can be constructed so that the number of kernels drawn is not fixed, but is a function of the fraction of kernels that have speed ratios in excess of two. The hypothesis is tested each time the sample size is increased by one. The number of samples drawn must be at least six if we wish the probabilities of accepting the hypothesis when it is false or rejecting it when it is true to each be less than 0.1. The practical significance of this scheme is that no more kernels than necessary need be programmed, thus avoiding the wasted effort of unnecessarily large sample sizes. Again, the reader is referred to Appendix A for a more thorough explanation of the scheme.

An estimate can be made of the actual speed ratio for a code, although the small sample size precludes determining the error bounds on the estimate. Such estimates are included in the Informal Estimates Section of this report.

A final complication for the workload determination was the concern that the LASL workload would not be "representative" of the broader ERDA-wide workload for the machine. LASL, of course, was primarily interested in the performance of a small set of codes that, for programmatic reasons, would be candidates for conversion to the Class VI computer. Thus a second set of codes, including the LASL set, was also tested. The first set was designated the Class VI applications workload, while the extended set was labeled the Class VI workload. It was decided to test the hypothesis for each of the applications codes and to test the hypothesis for the Class VI workload as a whole. At that time the problem of determining a rule attaching a weighting to the importance of each code in the applications workload became apparent. It was decided to use a decision rule that is applied against the results of the hypothesis test for each applications code and which determines whether or not CRAY-1 performance over the entire workload is satisfactory. The decision rule was formulated by LASL and approved by the ERDA Division of Military Application before the testing was initiated. The rule adopted was that the hypothesis must test as true for one or more of the three applications codes in order for the CRAY-1 to meet the applications workload threshold criterion. An indeterminate test result was to be considered a negative result when applying the decision rule.

Preliminary Scalar Performance Test

A preliminary test of the CRAY-1 scalar performance was desired in order to estimate if the CRAY-1 might meet the scalar performance criteria prior to beginning the extensive sampling procedure outlined above. Since the kernels selected by the sampling method would by necessity be very small, on the order of three to ten Fortran statements, a preliminary test program an order of magnitude larger was desired in order to provide a rough check on the sensitivity to kernel size of the CRAY-1/CDC 7600 speed ratio when determined by the sampling method.

The test routine chosen is a kernel of the Equation of State (EOS) algorithm. EOS is common to many LASL production codes. The algorithm consumes approximately 3.4% and 1.2%, respectively, of the CPU time for representative runs of two of the production codes comprising the Class VI workload.

The kernel TLU extracted from EOS is given in Figure B-1 and consists of over 70 lines of Fortran code. The kernel was hand translated into CDC 7600 assembly language and CRAY-1 assembly language as efficiently as was feasible for each machine. The assembly language code is available in a separate document. In order to factor out the impact of possible differences in efficiency between the CRAY-1 and CDC 7600 library versions of the functions ALOG and EXP, the functions were re-defined as ALOG(x) = x and EXP(x) = x.

Naturally, the CRAY-1 version of TLU consisted of purely scalar calculations. It can be noted from Figure B-1 that the test routine is characterized by much address computation, numerous tests and branches, and a moderate amount of floating-point arithmetic.

For each machine, the test routine was timed for 5400 calls with a different parameter set for each call in order to fully exploit all paths in the routine. This was accomplished by using a driver. A Fortran listing of the CDC 7600 driver is given in Figure B-2. The equivalent CRAY-1 driver was written in PASCAL and CRAY-1 assembly language.

Results of these timings are given in Table III-1. The timing ratio of 2.55, greater than the criterion of 2.0, indicated that the more extensive sampling procedure was worthwhile.

Table III-1. Scalar Test Routine Timings.

A. CDC 7600 Execution Time	.0877 sec.
B. CRAY-1 Execution Time	.0344 sec.
C. Ratio A/B	2.55

Workload Determination

For a code to be included in the Class VI workload, the following criteria were established:

1. The code consumes enough CDC 7600 CPU time to merit conversion to a Class VI computer.

2. The code will continue to be used at current or increased levels.

3. Support for the code's conversion to a Class VI computer exists within the user group.

A measure of at least one run per week greater than one hour on the CDC 7600 was chosen as the level of significance to merit conversion (Criterion 1). To accomplish this, a detailed analysis of LASL's production workload was undertaken utilizing historical accounting data provided by LASL's computer operations group.

Statistics are collected and maintained on the two CDC 7600 operating systems utilized for production work: CROS and LTSS. CROS is a batch operating system suited to a production environment, while LTSS is a timesharing operating system suited to an interactive environment. Both systems run on the CDC 7600. Accounting methodologies between these two systems differ. Statistics on the CROS system are detailed and flexible enough to summarize and tabulate requested data to the level of a single run. LTSS statistics, however, are logged by account number, the accounting philosophy being that an account number is associated with a "family" of codes programmatically related. Thus, many codes may charge to a single account and the charged time cannot be easily apportioned among individual codes, as in the CROS system.

A summary of compiled results is presented in Tables III-2 and III-3. The CROS data is straightforward. The criterion of "greater than one hour" could not be as easily applied to the LTSS workload because of the previously discussed accounting methodology. The approach taken on LTSS was to determine usage by account number. The codes associated with an account number were then determined by interviewing the holder of the account. Since it was not possible to break down the run times by code, short runs within a "family" are lumped together with production runs. Thus, the method of ascertaining the production workload on LTSS was to conduct a user survey to discover the account numbers and associated codes active during the production shifts. Fortunately, LTSS production codes are few in number and highly visible, since they were only recently transferred from CROS to LTSS.

The total LTSS charged time, by family of codes, is presented in Table III-3 for three months of usage. The reader will note that the values of charged time are low when compared to the CROS figures, especially when one considers that the CPU time includes all runs charged to an account, not merely runs of one-hour duration or more. This is explained by 1) at that time, there was only one CDC 7600 running LTSS in a production environment while three CDC 7600s were running CROS, and 2) LTSS currently has a smaller population of production codes.

The complete set of codes enumerated in Tables III-2 and III-3 comprise the potential Class VI workload. The significance criterion (Criterion 1) determines that a code must have more than 12 one-hour or greater runs in three months to be considered in the Class VI workload. On the basis of Criterion 1 all codes consuming less than 12 hours of CDC 7600 time in greater than one-hour runs were eliminated from consideration.

Additionally, programmatic requirements dictated that certain codes would receive little or no support for conversion to the CRAY-1. A user review process eliminated many codes from consideration on the basis of Criterion 2; namely, ZORK, TD4TWOD, TIGER, and the MAGEE and MCN families. MCRAD, EOTOTO, GUANO, TORUS-M and SIMMER were left to comprise the Class VI workload.

Of these five codes, MCRAD, GUANO, and EOTOTO were selected to comprise the Class VI applications workload. Table III-4 tabulates the final results of the workload study.

Table III-2.
Potential Class VI Workload from CROS.

System: CROS

Period Evaluated: 1/1/76 to 3/31/76

Source of Data: CROS HISTORICAL DAYFILE DATABASE

Code	User Supplied Description	Total CPU Time used (hrs.) for Jobs >1 hr.	% of Total CPU time for Jobs >1 hr.
GUANO	2-D Lagrangian explosion code	271.28	31.06
ZORK	1-D explosion code	92.75	10.62
EOTOTO	2D Monte Carlo particle-in-cell code	79.27	9.08
TD4TWOD	Controller for Magee family of codes	72.98	8.36
TIGER	2-D particle-in-cell S/N code	64.57	7.40
TORUS-M	3-D toroidal geometry fluid code; magnetic fusion research	48.58	5.56
SIMMER	Hydrodynamic-neutronics-equation of state code; reactor safety	39.00	4.47
MAGEE	2-D hydrodynamic code	36.51	4.18
MCNGACE	3-D Monte Carlo neutron and gamma transport code	17.05	1.95
MCNG	3-D Monte Carlo neutron and gamma transport code	10.03	1.15
MCRAD	Monte Carlo radiation output code	11.02	1.15
DITTO	1-D Lagrangian explosion code	8.67	0.99
MCMD	Monte Carlo molecular dynamics for hard spheres and hard disks	7.59	0.87
MOOP	Weapons diagnostics	6.81	0.78
DYNSTAR	Weapons diagnostics	5.89	0.68
LONGWALK	Weapons diagnostics	4.47	0.51

SPECTOR	2-D radiation transfer code; astrophysics research	3.31	0.38
YAQM	2-D radiation transfer code	3.04	0.35
UNDETER-MINED	No identification--unable to infer from other dayfile data the code ID.	91.22	10.45
TOTAL		873.04	100.00

Table III-3. Potential Class VI Workload from LTSS.

System: LTSS

Period Evaluated: 1/1/76 to 3/31/76

Source Data: Account Number Data Base and User Survey

Code	User Supplied Description	Total Time (hrs.) Charged for all Runs	Percent
MCRAD	Monte Carlo radiation output code	463	85.42
MAGCON	Controller for MAGEE family of codes	49	9.04
LASNIX	2-D hydrodynamic code for laser fusion pellet design	14	2.58
WAVE	2-D electrodynamic relativistic particle simulation code; laser fusion	11	2.03
SIMMER	Hydrodynamic-neutronics-equation of state code; reactor safety	5	0.92
TOTAL		542	99.99

Table III-4. LASL Class VI Workload and
Class VI Applications Workload.

Class VI Workload

Code ID	User Supplied Description
MCRAD	Monte Carlo radiation output code
GUANO	2-D Lagrangian explosion code
EOTOTO	2-D Monte Carlo particle-in-cell code
TORUS-M	3-D toroidal geometry fluid code; magnetic fusion energy research
SIMMER	Hydrodynamic-neutronics-equation of state code; reactor safety

Class VI Applications Workload

Code ID	User Supplied Description
MCRAD	Monte Carlo radiation output code
GUANO	2-D Lagrangian explosion code
EOTOTO	2-D Monte Carlo particle-in-cell code.

Execution and Monitoring

After the LASL Class VI workload and its subset, the Class VI applications workload, had been determined, each user of a selected code was asked to provide files of the code's Fortran source and all data and control card files necessary for a typical execution of the program. The source code and associated data files were copied in order to insulate the "evaluation" code from the normal evolutionary changes of a production environment. Isolation was desired in order that monitoring results be reproducible during the entire evaluation period.

Monitoring proceeded utilizing the LASL-written software monitor STAT. For a given code, the entire field length of a particular overlay and its suboverlays was divided into non-overlapping address ranges labeled "buckets." STAT samples the program address counter and determines which bucket encompasses the address. STAT then increments a count for that bucket and subsequently summarizes these results into a cumulative distribution that gives the fraction of time spent by the sampled overlay in execution in or below the address range for each bucket. Implementation does not allow the generation of a cumulative distribution over more than one overlay. Additionally, the bucket size was chosen to be 16 words--a size large enough to encompass a meaningful Fortran segment yet small enough to result in assembly language segments of tractable size.

The subject codes were run and monitored over a sufficiently long period to exercise the normally used features of each code--typically five to fifteen minutes of CDC 7600 time. Monitoring was done on those overlays that encompassed the main iterative loop. The five to fifteen minutes includes pre- and post-processing which were ignored because this processing consumes insignificant portions of a code's production execution time.

Kernel Selection

In order to achieve a single cumulative distribution for an entire code execution, a separate program was written to integrate STAT-generated overlay cumulative distributions. This integration is accomplished by weighting the individual overlay distributions by the percent of time spent in each overlay. From this data a single cumulative distribution can be inferred.

The same program then generates a random number, evenly distributed from 0 to 1, utilizing the Fortran library function RANF. The generated random number is applied to the cutoff points of the cumulative distribution associated with each overlay. For example, assume three overlays (A, B and C) were sampled by STAT and consumed 10 sec., 20 sec., and 35 sec., respectively, of run time. The fraction of the total time spent by each overlay would be 10/65, 20/65 and 35/65, respectively. This would imply cutoff points of 10/65 and 30/65 for overlays A and B, respectively. Thus, if the random number .31417 was obtained, it would be associated with overlay B. Furthermore, the bucket associated with .31417 would be associated with the B overlay cumulative distribution value .31417 - 10/65.

An initial sample size of six from each of the three Class VI applications workload codes was drawn for the Class VI applications workload sample. This is because the minimum sample size for testing the speed ratio hypothesis is six (see the Statistics Addendum). Ten kernels were drawn for the test of the Class VI workload, two from each of the five codes comprising the workload. The initial sample size of ten allows each code to be equally weighted in the test. The random number which determined the location of each kernel (the "bucket") in the code and the overlay block in which the kernel resides is presented in Table III-5. Each kernel is identified by a unique label. Kernel labels begin with the first letter of the code from which they were drawn, and are numbered in the order in which they were drawn. Kernels drawn for the Class VI workload test are specified by an "H" in their label.

In this initial drawing of 28 kernels, four were duplicates, namely G2/G3. E3/E6, M2/MH1 and SH1/SH2. Duplication occurs when two random numbers are so close in value that they "fall" into the same bucket.

Table III-5.
Random Numbers Associated with Kernels.

Code Name	Kernel ID	Random Number	Overlay Block
EOTOTO	E1	.0678473	(1,3)
	E2	.3986746	(2,2)
	E3	.9058777	(2,2)
	E4	.1780081	(2,2)
	E5	.8107489	(2,2)
	E6	.8764135	(2,2)
	EH1	.6488114	(2,2)
	EH2	.6430924	(2,2)
GUANO	G1	.0063286	(2,0)
	G2	.8719697	(7,0)
	G3	.9059678	(7,0)
	G4	.9562930	(8,0)
	G5	.3848960	(2,0)
	G6	.2780379	(2,0)
	GH1	.2046974	(2,0)
	GH2	.4215584	(2,0)
MCRAD	M1	.3130292	(2,0)
	M2	.0574823	(0,0)
	M3	.0409673	(0,0)
	M4	.6274798	(2,1)
	M5	.6945399	(2,1)
	M6	.7010670	(2,1)
	MH1	.0706280	(0,0)
	MH2	.6218473	(2,1)
TORUS-M	TH1	.2906896	(0,0)
	TH2	.5704439	(0,0)
SIMMER	SH1	.1115372	(3,5)
	SH2	.2733280	(3,5)

A portion of the segment of Fortran statements associated with the bucket was selected as the Fortran kernel. This was accomplished according to the following procedure.

a. If the address boundaries of the bucket resulted in incomplete Fortran statements, the segment was expanded to include entire statements.

b. Starting at the selected address one went "backward in execution" to either the segment boundary or a labeled statement actively accessed from outside the segment. "Active" execution paths are those taken during execution of the code. Whether an execution path was active or inactive was inferred from the STAT histogram of execution within the segment. The selected statement formed the first statement of the Fortran kernel.

c. From the first statement of the Fortran kernel one went "forward in execution" to either the segment boundary or an active jump outside the segment. The subsegment encountered in this action comprises the Fortran kernel.

If at any time in this selection process additional knowledge of program flow was required, STAT was rerun over the kernel in question with a one-word bucket size. It should be noted that seldom were two possible program flow paths active in the same segment. Thus steps b and c usually consisted of merely identifying the active portion of the segment. The active portion then comprised the kernel.

In summary, kernel selection was accomplished in a straightforward and unbiased manner. The methodology yielded representative samples weighted by frequency of execution in a code.

Kernel Implementation

The code selection yielded an initial 28 kernels to be coded both in COMPASS (CDC 7600 assembly language) and in CAL (CRAY-1 assembly language). A careful examination of the segments, from which each kernel was selected, established coding assumptions for each kernel. Some examples of coding assumptions follow. By examining the section of code from which the kernel was extracted one could determine variables local to the kernel. These variables were left in registers after calculation but not stored to memory. If a variable was a DO loop index, it was assumed to be in a register and not fetched from memory every time it was used. On the CRAY-1, B and T registers were used for temporary storage whenever feasible. Both the Fortran statements and the coding assumptions were incorporated as comments at the beginning of each kernel.

The evaluation team was assigned two full-time programmers and the services of two data analysts. One programmer was the primary author of the majority of the COMPASS kernels, while the other was the primary author of the CAL kernels. The data analysts took the handwritten code, punched the decks, and assembled and executed the kernels. They corrected some assembly errors and kept a file of all the decks and listings. This technique let the programmers concentrate their efforts on writing correct and efficient kernels. The success of this technique is indicated by the fact that it took only five weeks to completely write, assemble, and make the first debug pass over 52 assembly language kernels--26 COMPASS and 26 CAL. Two of the 28 kernels were nothing more than a block transfer of data from Large Core Memory (LCM) to Small Core Memory (SCM) on the CDC 7600. Since no such transfer is necessary on the CRAY-1, the kernels "exceeded" the speed ratio criterion and no coding was necessary.

Following the programming of the kernels the programmers spent four weeks debugging the kernels, "tuning" the code for efficiency, and verifying that the kernels satisfied the execution assumptions. Both programmers verified all kernels for accuracy and efficiency. The CAL and COMPASS versions of each kernel were checked against each other, instruction by instruction, by both programmers to verify their functional equivalence. The data analysts continued making all changes, executing the kernels, and updating the files.

A further pass over all the kernels was made by other experienced assembly language programmers.

These people looked for errors in the documentation and the coding and checked to see if any improvement could be made in the efficiency of the kernels. This pass revealed few substantial errors. Most corrections were aimed toward improvement in documentation. Once this process was complete the suggested changes were incorporated, and the kernels sent to FEDSIM for final review. Changes suggested by FEDSIM were then incorporated to produce the finished kernels.

The following steps were taken to guarantee that the implemented kernels are efficient and accurate.

1. The authors of the kernels were chosen for their experience in efficient coding. They were extremely knowledgeable about the architecture of the two machines that were programmed.

2. REGREF, a LASL-developed software tool that statically analyzes COMPASS code for efficiency, was used as an aid in optimizing the COMPASS kernels.

3. Both the two-man checkout and the cross-checking by other programmers were done to correct inefficiencies and to minimize any differences in efficiency attributable to individual programmers.

The assembly language listings of the kernels, timing information and driver programs comprise about 350 pages of output. In addition, from one to ten pages of Fortran text were extracted for each kernel in order to define the kernel and establish the coding assumptions relevant to the kernel. The STAT-generated output used to locate and define the kernels is approximately 500 pages long for the five monitored codes. Source listings of the monitored codes were so lengthy that they were tractable only in microfiche format.

Because of the length of this material, no listings can be reproduced in this report. However, listings of kernel E1 are presented as an example in Figure B-3.

Timing Results

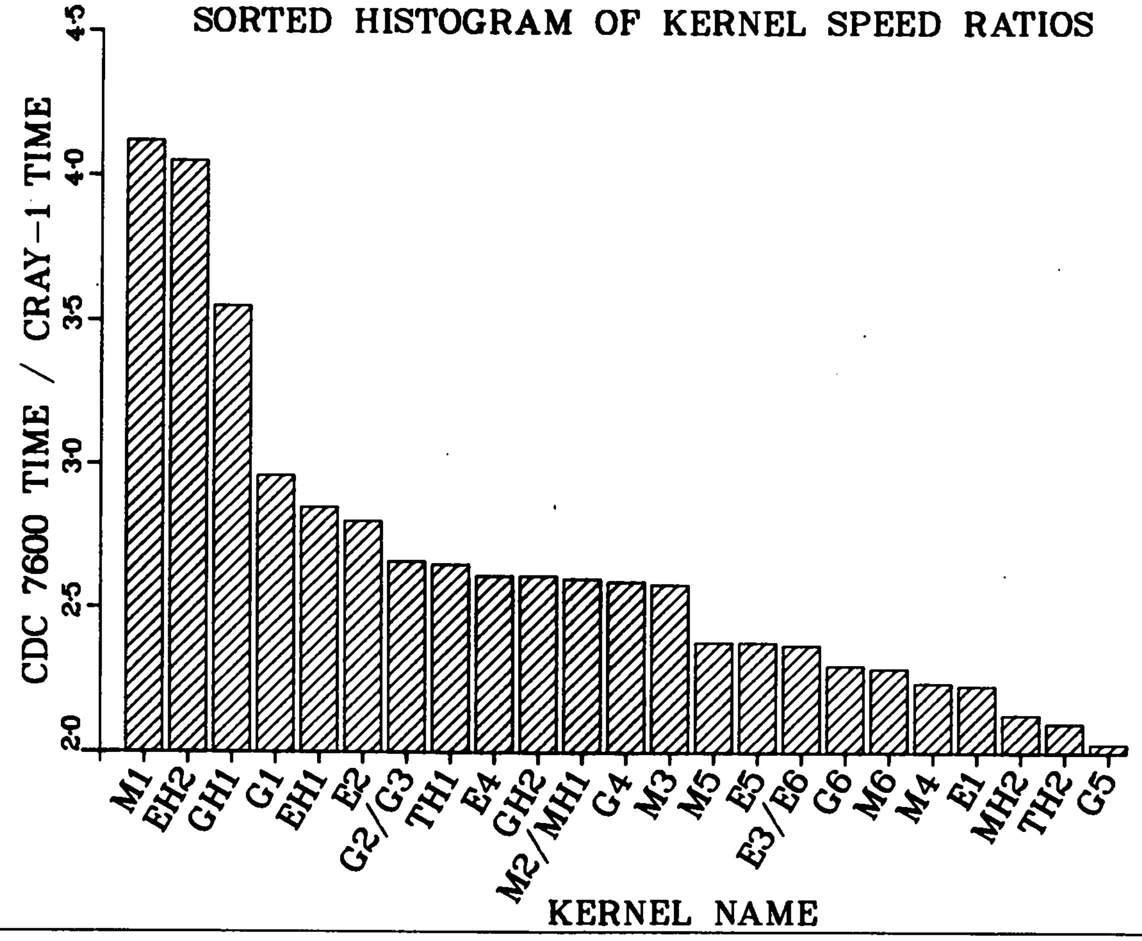
Results of the kernel timings are displayed in Table III-6. A sorted histogram of the kernel speed ratios is displayed in Figure III-1.

Table III-6. Timing Results.

Code Name	Kernel ID	Subprogram Name	Absolute Address Octal	CRAY-1 Cycles	CDC 7600 Cycles	CDC 7600 Time/ CRAY-1 Time
EOTOTO	E1	HYDRO	130020-130040	80	81	2.23
	E2	FLYPART	112760-113000	92	117	2.80
	E3	ALNLOG	134120-134140	93	100	2.37
	E4	FLYPART	110140-110160	75	89	2.61
	E5	ACECASE	131660-131700	100	108	2.38
	E6	ALNLOG	134120-134140	93	100	2.37
	EH1	ACETOTN	127070-127100	64	83	2.85
	EH2	ACETOTN	127020-127040	51	94	4.05
GUANO	G1	EOSE	61200-61220	121	163	2.96
	G2	NSOLV	123220-123240	130	157	2.66
	G3	NSOLV	123220-123240	130	157	2.66
	G4	UDDER	101020-101040	117	138	2.59
	G5	CSOLV	115440-115460	117	108	2.03
	G6	CSOLV	115320-115340	44	46	2.30
	GH1	VEPQT	107760-110000	121	195	3.55
	GH2	CSOLV	115460-115500	70	83	2.61
MCRAD	M1	RADSWP	71640-71660	47	88	4.12
	M2	TEXP	44463-44512	49	58	2.60
	M3	TLOG	44463-44512	46	54	2.58
	M4	H2200	103620-103640	51	52	2.24
	M5	H2200	105120-105160	72	78	2.38
	M6	H2200	105200-105220	72	75	2.29
	MH1	TEXP	44463-44512	49	58	2.60
	MH2	H2100	103600-103620	67	65	2.13
TORUS-M	TH1	BAAL	31460-31500	64	77	2.65
	TH2	BAAL3	44660-44700	111	106	2.10
SIMMER	SH1*	ECSRW	15520-15540	--	--	--
	SH2*	ECSRW	15520-15540	--	--	--

*These two kernels perform large core memory/small core memory transfers on the CDC 7600. No such transfers are necessary on the CRAY-1. Hence, these kernels "exceed" the speed ratio criterion and did not require implementation.

FIGURE III-1.
SORTED HISTOGRAM OF KERNEL SPEED RATIOS



All kernels drawn displayed a speed ratio that was in excess of 2.0. Thus it was not necessary to draw more than six kernels per test (with the exception of the Class VI workload test in which two kernels from each of the five codes were chosen). This result was unexpected because of the following simple analysis. The ratio of CPU cycle times for the two machines is 2.2. The ratio of memory access times is 1.8. If one assumes the cycle counts for instructions to be roughly equivalent, then one would expect the speed ratio to bracket this range. A careful analysis of the coded kernels revealed that certain hardware features of the CRAY-1 resulted in the kernels being coded in fewer CRAY-1 instructions than CDC 7600 instructions, resulting in speed ratios greater than 2.2 in many cases. A summary of these features follows.

Impact of Hardware Features

Out of the 26 kernels that were coded, 23 took fewer cycles to execute on the CRAY-1 than on the CDC 7600. In the opinion of the programmers, the CRAY-1 was also easier to program. Both of these facts can be attributed to hardware features that exist on the CRAY-1 and not on the CDC 7600. Some of these features are given here.

1. The B and T registers are available for temporary storage. These registers eliminate the use of memory for partial results that are calculated in a fairly elaborate Fortran expression.

2. Data can be fetched and stored out of all eight S registers. Given the Fortran expression I = J, it is necessary to fetch J and store it to location I. Since the fetch times are 10 and 8 cycles for the CRAY-1 and the CDC 7600, respectively, it would appear at first glance that the CDC 7600 should take fewer cycles than the CRAY-1. This is not true because the CDC 7600 fetches data into X registers 1-5 and stores out of X registers 6 and 7. So after J is fetched into X1, for example, it must be moved to X6 before being stored. Thus in this case the operation takes the same number of cycles on both machines.

3. Data can be fetched and stored directly to all A registers. This allows index calculations to be completely independent of the S registers and other floating

point calculations.

4. Results from the floating add unit are normalized before being returned to the S register. The floating add unit takes 6 cycles on the CRAY-1 and only 4 cycles on the CDC 7600. But if one adds the cost of normalizing, which is 3 cycles, the add and subtract times are better on the CRAY-1 than on the CDC 7600. Furthermore, because it takes two instructions to complete the add on the CDC 7600 there is always the chance of adding extra cycles due to access path delays.

5. All functional units are fully segmented. On the CDC 7600 the multiply and divide functional units are only partially segmented. The cycle times for the multiply units are 7 and 5 for the CRAY-1 and the CDC 7600, respectively. The full segmentation for the CRAY-1 overcomes this cycle count disadvantage at three successive multiplies. That is, if three successive multiplies are issued on both machines, the last result will be available on both machines at the same cycle.

6. The instruction buffer on the CRAY-1 is much larger than that of the CDC 7600. Forward branching on the CRAY-1 does not necessarily clear the instruction buffer, and in fact most of the forward branches in the kernels were to instructions contained in the instruction buffer.

Conclusions

The hypothesis that the CRAY-1 in scalar mode is greater than two times faster than the CDC 7600 was tested individually for three codes (the Class VI applications workload) and for a group of five codes (the Class VI workload). The hypothesis tested as true in all cases. Thus the CRAY-1 meets the scalar performance criterion for each of the three Class VI applications workload codes and for the Class VI workload.

The reader is cautioned that a careful interpretation of these results is necessary. First, the manner in which the kernels were implemented upon each machine was designed to be a test of the hardware, not software, of the system. The performance effects of compilers were not considered. If one assumes that the ability of a compiler to produce efficient code increases with the man-years invested in the compiler, then one must assume that any CRAY-1 compiler when implemented because of its relative immaturity will produce less efficient code than the very mature optimizing compiler available on the CDC 7600. Second, the tests were made with the CRAY-1 in scalar mode. Since CRAY-1 vector operations are faster than the equivalent scalar operations, vectorization will normally result in higher speed ratios. Third, input/output was not considered in this test. Performance of input/output-bound codes will be determined by the relative performances of the two machines' I/O subsystems. However, it should be noted that on the basis of statistics generated by the STAT sampling routine, the Class VI applications workload codes are heavily CPU-bound on the CDC 7600. Finally, the sample size drawn, while statistically valid for the hypothesis test, is too small to provide us with statistically valid estimates of the total code's speed ratio. The CRAY-1 was tested against a performance threshold for efficient machine language computational kernels, and exceeded this threshold.

SECTION IV

VECTOR PERFORMANCE

CRAY-1 vector algorithms are typically two to five times faster than equivalent scalar algorithms, depending upon vector length and the algorithm's complexity. It is then obvious that the performance of the CRAY-1 relative to the CDC 7600 for a particular code will depend upon the vector operations that can be exploited by that code. Although it would be very desirable to characterize precisely the type and proportion of vector operations each code would employ on the CRAY-1, this is presently an unmanageable task. Converting codes to employ a significant amount of vector operations inevitably requires restructuring of the algorithms. This restructuring may be so trivial that it can be done during compilation or, more typically, may require the efforts of an experienced programmer. Such restructuring was beyond the resources, both time and human, of this evaluation.

The approach of categorizing the "vectorizability" and type of vector operations by automatic examination of source code is meaningless. Any such examination results in a "vectorized" proportion that inevitably increases, possibly by an order of magnitude, when the code is restructured for a vector machine. See Ref. 1 for a more detailed discussion of automatic "vectorization" of programs.

Thus, at this time it is not possible to obtain rigorously "representative" vector functions in order to test the CRAY-1 against the vector criteria. "Vectorized" codes exist at other installations, but none duplicate the LASL workload, and few have been written for the CRAY-1. The approach adopted was to test the CRAY-1 against vector algorithms submitted by the Applications Support and Research Group of the Computer Science and Services Division of LASL that, in their judgement, represented typical vector operations that user codes would perform on the machine. Five operations were chosen to evaluate the machine's vector performance versus vector length. The average of the five speed ratios was chosen to compare against the qualification criteria. Each vector kernel was coded as efficiently as was feasible in assembly language for each machine. In particular, the CDC 7600 kernels were coded so that all instructions were maintained in the instruction stack of the machine during the operation loop, with functional units overlapped to the fullest extent--a scheme representing very efficient utilization of the CDC 7600.

The five functions were timed on both machines and speed ratios calculated. Timings are presented in Table IV-1. Fortran versions of the five functions are presented in Figure B-4. The average speed ratios of the CDC 7600 execution times to the CRAY-1 execution times for the five functions are 3.39, 4.50 and 5.12 for vector lengths of 20, 100 and 500, respectively. Thus the CRAY-1 meets the vector operations performance criteria as specified in Section III.A.2 of the Evaluation Plan (Appendix A).

1. Optimal Utilization of Supercomputers, Volume I - The Control Data 7600, April 1976, R&D Associates, P. O. Box 9695, Marina Del Rey, California.

Table IV-1. Vector Performance Summary.

A. Execution times of the algorithms (milliseconds):

Vector Length:	20		100		500	
Machine:	CRAY-1	7600	CRAY-1	7600	CRAY-1	7600
VVSPVS	.00190	.00732	.00537	.02712	.02288	.12612
VVVPVV	.00235	.00861	.00788	.03509	.03603	.16701
VVVPV	.00213	.00726	.00660	.02816	.02945	.13266
DOTPRO	.00312	.00649	.00551	.02189	.01650	.09889
ADDVEC	.00180	.00715	.00523	.02489	.02278	.11275

B. Ratios of CDC 7600 execution times to CRAY-1 execution times:

VVSPVS	3.85	5.05	5.51
VVVPVV	3.66	4.45	4.64
VVVPV	3.41	4.27	4.50
DOTPRO	2.08	3.97	5.99
ADDVEC	3.97	4.76	4.95
C. Average Ratio:	3.39	4.50	5.12

SECTION V

RELIABILITY

Approach

It was considered crucial tnat any Class VI computer considered for purchase be very reliable. Extended periods of downtime would be intolerable since the programmatic functions served by the computer could not be absorbed by other machines at the Laboratory. Thus rigorous reliability standards were formulated. In addition to "system availability," the single measure that is commonly used to define reliability, two additional reliability criteria were specified. Threshold reliability criteria of at least 80 percent system availability, at least four hours Mean-Time-to-Failure (MTTF) and at most one hour Mean-Time-to-Repair (MTTR) were established by LASL and ERDA as defining an acceptable level of reliability. The reader is referred to the Evaluation Plan (Appendix A) for a precise definition of these measures.

The reliability criteria would have to be met for a contiguous twenty-workday period for the machine to be considered for further procurement. The twenty-day period was established in order to smooth daily fluctuations expected for the measures; the period was considered long enough to prevent a machine meeting the criteria during a "fluke" period of good behavior. In order not to unfairly penalize newly constructed machines, the measures from the best twenty-workday period would be applied against the criteria.

Commonly adopted logging procedures for determining machine reliability were deemed inappropriate due to the special evaluation environment of the CRAY-1. This conclusion is the result of the two observations that:

1. The burden of logging machine reliability falls upon a large number of operators and programmers, and is vulnerable to human error; and

2. The complex environment and primitive operating system of the CRAY-1 make it difficult to isolate CRAY-1 hardware failures from a) operator errors, b) ECLIPSE hardware/software failures, and c) CRAY-1 benchmark operating system software errors (unless a hardware error

interrupt occurs and is handled correctly by the system).

The evaluation environment of the CRAY-1 resulted in the CPU being idle the greatest fraction of time. The mode of operation for running on the machine typically consists of a programmer performing nearly all tasks on the ECLIPSE and running a program on the CRAY-1 for only brief intervals. The CRAY-1 benchmark operating system presently does not have implemented the capability of running a "background" job to keep the machine busy.

The EXERCISER Program

The EXERCISER program was written in an attempt to overcome these limitations. The objective of the program is to approximate a production environment on the CRAY-1 by utilizing as many hardware features of the machine as possible for substantial periods of time. The EXERCISER program is self-verifying so that all machine failures, detected at the hardware level or not, will be noted. The program keeps a printed log of the time of each failure. Reliability statistics may be gathered from the printed log, thus minimizing the possibility of human error.

Hardware components specifically being tested are the vector and scalar functional units of the CPU, the memory, and input/output (I/O) components. Unfortunately, I/O was not available on the machine during the first two months of the evaluation.

EXERCISER was adapted from one of the first programs written for the CRAY-1 at LASL. Coded in CAL with PASCAL drivers, it solves for the vector x the matrix equation Ax=y by LU decomposition.

For each NXN matrix, N+1 systems of equations are solved. The A and y are parameterized such that every element of x in the N+1th system is near unity. Before N is incremented, every element of x is tested against unity. If a "near" unity test fails for any component of x, a message is dispatched to the operator and the event is logged by the program to a print file. The program then restarts the cycle until it is terminated by the operator. Most failures, such as parity errors, cause termination of the program and a message is dispatched to the operator by the operating system.

After June 15 substantial changes were made in the program in order for it to more fully meet its objective. First, N was fixed at 705, so as to "exercise" almost all of available memory. The

EXERCISER code and data require approximately 501 000 words.

The second substantial change was the addition of I/O to a DD-19 disk. The control flow of the EXERCISER program remains unchanged with the following exception. At the completion of solving the Nth system, the LU matrix is written to disk and then read from disk. The N+1th system is then solved and the test against unity is made. Any disk errors will result in this test failing. As before, upon detection of a failure, the program logs the failure and transmits a message to the operator.

EXERCISER was run from five to eight hours per day, five days a week throughout the entire six-month period. A sample of the daily log is shown in Figure V-1.

Results and Conclusions

Results, by twenty-workday interval, calculated weekly, are displayed in Table V-1. The MTTF measures are also displayed graphically in Figure V-2. Failures are categorized in Table V-2. One sees that the CRAY-1 meets the threshold criteria for numerous periods.

An analysis of failures occuring during EXERCISER runs revealed that "intermittent" memory parity errors dominate. Hardware detection of a memory parity error prompts an interrupt which idles the machine and saves the program counter. EXERCISER is so constructed that from the program counter value and a memory dump, the exact memory bit causing the failure can be determined. According to Cray Research, Inc. (CRI), roughly 60 percent of the intermittent failures were diagnosed to this level of detail, of which all were caused by failure of a single bit. In all but 12 of the memory parity error failures the memory module incorporating that bit could not be made to fail again, neither during EXERCISER nor during various CRI memory diagnostic routines. In the 12 cases of reproducible failure, the memory module was judged defective and replaced. In the remaining cases, tests of the modules by CRI could determine no differences in operating characteristics between "once failing" and "never failing" modules. Once reinstalled in the machine, no module failed again.

CRI made various hardware changes in an attempt to eliminate the failure mode or cause it to replicate at a more rapid rate. None of the hardware changes appeared to affect the failure rate. An obstacle to diagnosing this problem was the relatively long average interval between failures (4 hours), which implied very long running times during a change to determine if the failure rate had been changed.

Since the "once failing" module was seldom replaced, the MTTR measure should be more accurately labeled "mean-time-to-return," since hardware repairs were infrequent. In order to obtain a more accurate estimate of repair times, the mean-time-to-repair for all EXERCISER failures during which a repair was effected was calculated. For these 20 failures the total down-time was 8.37 hours, resulting in a MTTR of 0.41 hours. From this measure we conclude that only a minor portion of repair time was consumed by the actual hardware change, the major portion being consumed by alerting the engineer and running diagnostic routines.

Since memory parity errors so clearly dominate the statistics, an analysis of EXERCISER was made in an effort to relate its memory access rate to that of a production environment. Memory utilization by EXERCISER is divided into two phases. In the I/O phase, 497 025 memory locations are accessed twice during a 3.10-second period. This results in a memory access rate of approximately 0.0062 times the theoretical maximum of 80 million accesses per second (80 MAPS). In the computation phase of the program, a 705 by 705 matrix is initialized and decomposed and a system of 705 linear equations is solved. This phase lasts 14.11 seconds. The use of vector instructions in the inner computational loop results in 64 memory accesses every 78 clock cycles. The inner loop consumes roughly 90% of the time for this phase. Thus memory is driven at approximately 0.9 x 64/78 = .74 of its maximum of 80 MAPS. Ignoring the memory accesses during the I/O phase, EXERCISER drives the CRAY-1 memory at .74 of its capacity for 14.11/(14.11 + 3.10) = .82 of the program's execution, for an overall memory access rate that is 0.61 of its maximum.

Predicting a typical memory access rate for a production environment with the above accuracy is not possible. However, it is plausible that the EXERCISER bandwidth is at least several times higher than that expected for a production workload. Thus we would expect the MTTF for a production workload on the CRAY-1 to be significantly higher than that observed under EXERCISER.

Figure V-1. Daily EXERCISER Log.

EXERCISER LOG

EXERCISER VERSION:

SYSTEM VERSION:

COMMENTS:

DATE:

EXERCISER BEGIN TIME:

EXERCISER END TIME:

OPERATOR'S INITIALS:

CE'S INITIALS:

EXERCISER INITIATED	EXERCISER DETECTED ERROR	SYSTEM DETECTED ERROR	EXERCISER TURNED OFF	SYSTEM CRASHED	CE REPAIR INITIATED	CE RETURNED MACHINE	CE REPLACED HARDWARE MODULE				

SUMMARY

Time between EXERCISER start and failure/between failures (minutes).

1.______ 2.______ 3.______ 4.______ 5.______

6.______ 7.______ 8.______ 9.______ 10.______

Time CE has machine for repair (minutes)

1.______ 2.______ 3.______ 4.______

5.______ 6.______ 7.______ 8.______

Time from last failure to shut off

1.______

No time intervals

1.______

Figure V-2. MTTF Grapn.

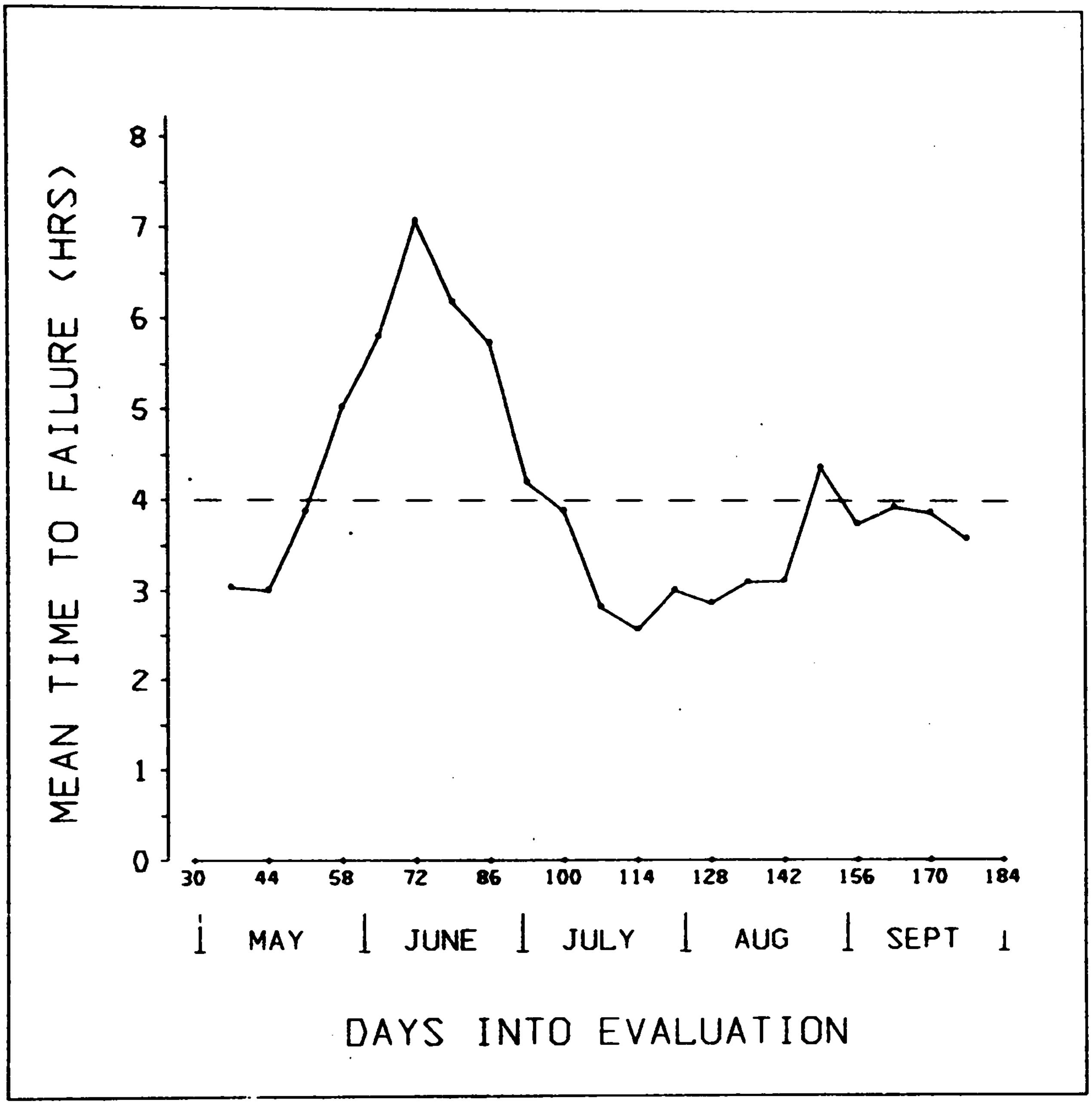


Table V-1.

Reliability Statistics Summarized by Week.

Twenty-Workday Period	Operational Use time (Hrs.)	Remedial Maintenance Time (Hrs.)	No. of Failures	MTTF (Hrs.)	MTTR (Hrs.)	SA
4/5 -5/7	115.15	17.72	40	2.88	.44	.87
4/14-5/14	113.50	16.35	38	2.99	.43	.87
4/23-5/21	112.05	11.83	29	3.86	.41	.90
4/30-5/28	115.37	9.25	23	5.02	.40	.93
5/6 -6/4	110.13	7.08	19	5.80	.37	.94
5/14-6/11	106.25	5.65	15	7.08	.38	.95
5/21-6/18	104.98	6.33	17	6.18	.37	.94
5/28-6/25	109.02	7.07	19	5.74	.37	.94
6/8 -7/2	100.78	10.40	24	4.20	.43	.91
6/15-7/9	97.12	9.55	25	3.88	.38	.91
6/22-7/16	87.42	16.67	31	2.82	.54	.84
6/29-7/23	88.57	18.12	35	2.53	.52	.83
7/6 -7/30	99.45	15.67	33	3.01	.47	.86
7/13-8/6	97.55	17.68	34	2.87	.52	.85
7/19-8/13	108.45	13.25	35	3.10	.38	.89
7/27-8/20	102.77	12.47	33	3.11	.38	.89
8/4 -8/27	104.60	8.77	24	4.36	.37	.92
8/10-9/3	104.73	10.85	28	3.74	.39	.91
8/12-9/10	105.85	9.82	27	3.92	.36	.92
8/17-9/17	108.45	10.02	28	3.87	.36	.92
8/23-9/24	114.25	10.43	31	3.69	.34	.92

Table V-2.

Failures Classified by Type.

Memory parity errors*	152
Disk	1
Vector modules	12
Instruction buffer module	1
Floating add module	4
Total	170

*Of the 152 memory parity errors, 12 were found to be reproducible.

NOTE: Of the 18 non-memory related failures, 8 repairs were effected at the time of failure.

If the cause of the intermittent memory failures cannot be diagnosed and eliminated, the advantages of a machine with memory error correction are obvious. If one assumes all intermittent memory failures (139) could have been avoided by a single bit correction technique, then the MTTF for such a machine over the entire evaluation period would have increased by a factor of 170/(170-139) = 5.5.

This estimate assumes that a reproducible memory parity error, of which 12 were observed, would result in the machine failing. CRI reported that all 12 memory modules replaced were single bit failures. If one assumes that error correction would allow deferring module replacement into the scheduled maintenance period, then it is possible that single bit correction would have resulted in a MTTF increase by a factor of 170/(170-152) = 9.4 over the entire evaluation period.

SECTION VI

INPUT/OUTPUT PERFORMANCE

Disk Performance Tests

Four tests were designed to test the capability of the CRAY-1 DD-19 disk system.

Routine DISK1 measures the instantaneous data rate for the DD-19 disk system. One track (8192 64-bit words) is written to disk and then read back 1000 times, with the writes and reads timed. No arm movement or head switching is required, since the same track is repeatedly read and written.

Routine DISK2 determines the maximum data rate the DD-19 will sustain. A large file is written sequentially on the disk and read back several times, with the read and write operations timed. This test is designed to determine the maximum data transfer rate that can be expected for large files.

Disk arm positioning times are measured by routine DISK3. A file is written to the disk across many cylinder boundaries, then read back in one sector (512-word) blocks. Read requests are arranged to force arm movement across a fixed number of cylinders. The time to position across a minimum of 2^i cylinders, $i=0,\ldots,8$, is recorded.

DISK4 measures the data error rate for the disk. A large file of approximately two million sequential integers (256 tracks) is written to the disk, then read back and verified.

Table VI-1 summarizes the test results and Table VI-2 summarizes expected disk characteristics. All test routines were written in CRAY-1 assembly language and run under the existing benchmark operating system.

The benchmark operating system imposed constraints that are discussed later. All timings were made using the CRAY-1 real-time CPU clock. Tests were run on both DD-19 disk units available and no significant differences between units were detected.

Table VI-1. Disk Test Results.

	Total No. Words	Time (sec) Write	Time (sec) Read	Rate (1000 words/sec) Write	Rate (1000 words/sec) Read
DISK1:					
16 sectors/track (1 track buffer)	8 192 000	33.04	33.03	248	248
16 sectors/track (1 cylinder buffer)	8 192 000	33.04	14.37	248	446
15 sectors/track	7 680 000	16.52	16.52	465	465
DISK2:					
16 sectors/track	8 192 000	33.06	18.20	248	450
15 sectors/track	7 680 000	18.20	18.20	422	422

Positioning Time

	Cylinders Moved	Time (Sec)
DISK3:		
	369	.088
	256	.067
	128	.050
	64	.050
	32	.033
	16	.033
	8	.033
	4	.017
	2	.017
	1	.017

	No. Words	Errors	Error Rate
DISK4:	4.23×10^9	0	0

Table VI-2.

Summary of Expected Disk Characteristics.

(from Interim Disk Storage Subsystem, publication 2240005 and Mass Storage Subsystem Reference Manual, publication 2240630, Cray Research, Inc.)

Available cylinders:	411 (404 plus 7 spare)
Number of head addresses:	10 (10 groups of 4 physical heads each)
Sectors/head address/cylinder:	16
CPU (64-bit) words/sector:	512
CPU (64-bit) words/disk:	33 669 120
Rotation:	3 600 RPM (±2%)
Seek timing.	
410 cylinder move:	80 ms maximum
1 cylinder move:	15 ms maximum
average:	50 ms
Latency.	
maximum:	16.7 ms

DISK1 uses a 8192-word (one-track) buffer. A one track block is transferred to the disk and then back 1000 times. Results are given in Table VI-1.

This test revealed tnat a disk revolution was being missed on every read or write attempt, since the overall transfer rate was half that expected. The buffer size was increased by a factor of 10 and the test run again. Results revealed that reads were proceeding at the expected rate but writes proceeded at the old rate. Analysis revealed that this anomaly was caused by a combination of hardware and software factors. The interim disk controller allows 16 sectors per track. In the case of disk reads, after the sixteenth sector is read the controller's one-sector buffer must be transmitted into central memory (CM) and another read request for the track of data initiated. The time required to empty the buffer and for the CPU to manage the new read request prevents issuing a controller read before the intersector gap has passed, thus missing a disk revolution. The larger buffer eliminates 9 of the 10 occurrences of the missed revolution as only 1 request is made by DISK1.

An analysis of the anomaly affecting writes in the 16 sectors/track case revealed the following. After the sixteenth sector is written, the CPU must initiate a head switch (not to be confused with a seek). In addition, the data for the first sector of the next track must be transferred from the CPU to the disk controller. The transfer of data to the disk controller cannot occur until the CPU has completed the head switching operation. By the time both have been completed, the disk has revolved too far for the transfer of data from the controller to the first sector of the new track to take place on the current revolution. Thus data can be written only every other revolution of the disk. This situation is caused by both command information and data being transferred on the same channel for writes.

In the case of reads, data transfer and commands are on different channels and can proceed simultaneously. When only 15 sectors are written there is sufficient time during the intersector gap for the head switch and data transfer to take place.

The maximum expected transfer rate of the disk with 16 sector tracks is 491 520 words/sec (±2%). For the large buffer case the results are within this two percent range.

DISK2 is implemented by reading or writing a 1000-track file to disk sequentially across track and cylinder boundaries. A ten-track (one cylinder) central memory buffer is used. The software/hardware anomaly that plagued DISK1 was observed. For this reason the test was also run transferring 15 sectors per track. Results are summarized in Table VI-1. Note that the 15 sector/track results are identical and close to optimum.

The missed revolution problem discovered by DISK1 and DISK2 should not exist for systems utilizing the product-line controller. Electronic head switching for a contiguous block of data is handled automatically by the controller.

Due to system limitations, DISK3 could only measure positioning timings to the resolution of one disk revolution (17 ms). This is not judged a serious problem because it corresponds to actual usage in most cases. However, the timings recorded are conservative because disk flaws, while present, are not included in the distances recorded. Since the system records as flawed a full cylinder at a time, this represents an extra cylinder to traverse per flaw. Thus, the timing for 369 cylinders actually represents about 390. Also, the full 411-cylinder seek could not be tested due to a software limitation in the operating system. Timing results are presented in Table VI-1. The timings are close to the expected capabilities of the disk unit.

DISK4 was implemented so that it could be run in a piecemeal fashion whenever time was available on the machine. The file was written and read 2,017 times during the course of a month. No errors occurred.

On the basis of these results we conclude that the performance of the disk system should be as expected under the new controller. Several deficiencies in the interim controller were discovered which should be corrected by the product-line controller. The practice of recording the disk space as flawed by cylinder is wasteful of space; production software should allow "flawing" by track or sector. Reliability of the disk system appears to be excellent.

Computation and I/O Interference

The bandwidths of the computation and I/O sections of the CRAY-1 may be degraded from nominal values through conflict for memory access. A detailed analysis of the cases of memory conflict is possible but laborious, and results would be difficult to validate. Tests on the current configuration (which has only a single disk controller) suffer because the single controller cannot support the different levels of traffic intensity necessary to measure bandwidth degradation versus traffic rate.

Cray Research, Inc., (CRI) estimates that the memory access bandwidth is capable of sustaining up to four disk and three interface channel transfers without degrading the compute rate of a mix of scalar and vector instructions.[1] A cursory analysis sustains this claim.

The disk controller can deliver four words per microsecond (µs). However, four disks, the maximum number a single controller will support, can deliver a maximum of only 2.4 words/µs. Thus, four simultaneous controller transfers can deliver a maximum of 9.6 words/µs. This is approximately 1/8 of the maximum memory bandwidth. Actual operating conditions will seldom generate such high degrees of I/O overlap for extensive time periods. Note that bank conflicts are ignored in the above analysis. This is because the nature of I/O transfers themselves will minimize bank conflicts. Thus, it is plausible that "normal" I/O activity will cause insignificant degradation of CPU performance.

Worst case degradation will occur for a program requiring continuous memory accesses during long vector transfers if I/O transfers are being attempted at the maximum rate. This degradation will result from the delays imposed upon the issue of vector instructions accessing central memory due to memory cycle stealing by the I/O transfers. CRI estimates that such degradation would be less than 10 percent over the same program running without I/O interference.[1]

The analysis of degradation of I/O system performance through lost data, missed disk revolution and repeated transfers is not quite so straightforward. The disk system operates through a double buffering of 512 word sectors. It is only necessary for the I/O system to obtain enough memory accesses (512) to fill a sector buffer in the approximately 847 µs taken for the disk controller to fill or empty the alternate sector buffer. The I/O system must, therefore, obtain memory accesses at a minimum average rate of .604 words per µs per controller or approximately 0.76 percent of the maximum memory bandwidth. This suggests a very low performance loss due to word transfers in ordinary circumstances. A worst case would be continuing execution of memory-to-vector register transfers. Each such transfer could in principle block I/O memory access for approximately .85 μs.[1]

A test was devised to measure the degradation to computation and to I/O performance for an I/O process and a computational process competing for memory. The computational kernel is characterized by sustained 64 element vector transfers to and from memory. The I/O kernel is characterized by block transfers between memory and disk proceeding at the maximum rate the existing operating system will support. The routine embodying both kernels is labeled IOTEST.

In IOTEST, an inner loop loads two vectors, performs an integer add, then stores the result. The operation is simply C=A+B, where A, B, and C are vectors of logical length 7,680. Since the vector registers of the machine are 64 elements long, the loop performs the operation in multiples of 64 words. An outer loop performs the vector add 262 144 times. These values were chosen to be large in order to result in sustained memory accesses via vector instructions.

The I/O kernel of IOTEST continuously overwrites 15 sector blocks to a single disk track. This results in the maximum transfer rate the current configuration will sustain by avoiding missed disk revolutions. Because of interim disk controller limitations, two I/O interrupts occur per sector transferred, requiring processing by the CPU monitor. An I/O interrupt is not honored while a vector instruction is being executed. There is no limit to the number of blocks written. The number of written blocks is determined upon completion of the vector additions.

IOTEST was also implemented with a vector kernel that performs no memory references in either loop. This was done in an attempt to factor out the effect of the overhead of two I/O interrupts per sector transferred from the missed chaining due to memory cycle stealing by I/O transfers. This version, labeled IOTEST', is identical to IOTEST with the exception that no vectors are fetched from or stored to memory. Only register-to-register operations are performed. The I/O kernels in both IOTEST and IOTEST' are identical.

1. CRAY-1 Computer System Reference Manual, publication number 2240004, Cray Research, Inc., p. 5-1.

Table VI-3.

Results of IOTEST and IOTEST'.

	Time (sec.) without I/O	Time (sec.) with I/O	Vector kernel run time degradation
IOTEST	84.30	90.38	7.2%
IOTEST'	27.60	28.62	3.7%

	Blocks written with I/O	I/O Interrupts	Transfer rate (words/sec)
IOTEST	5,456	163,680	463,621
IOTEST'	1,727	51,810	462,621

NOTE: IOTEST, without I/O, makes roughly 6 x 10^9 memory accesses in 84.3 seconds, which is 89 percent of the memory bandwidth.

Table VI-3 shows the results of IOTEST and IOTEST'.

The I/O transfer rate for both tests is nearly identical to the maximum rate determined earlier. Differences are well within the two percent range in disk spindle speed. We conclude that an I/O transfer rate of .463 megawords/sec is not impacted by a memory-bound computation. Since .463 megawords/sec is only 0.58 percent of the memory bandwidth, no degradation to the I/O process would be expected. An analysis of degradation to the computational process is not so straightforward.

Complicating matters are the two degradation effects contributing to the decrease in the vector kernel performance: first, overhead of system I/O routines and second, the I/O access to memory that inhibits vector chaining. There can be no degradation to vector chaining caused by the execution of the I/O routines. IOTEST', which performs essentially no memory references, experiences a degradation of 3.7 percent. This is assigned to the overhead of the I/O routines, since no memory conflicts are likely. IOTEST experiences a degradation of 7.2 percent, which includes the effect of memory conflict. A casual analysis allows assigning an upper bound of 3.5 percent degradation to the vector kernel due to memory conflicts. Four simultaneous disk transfers would drive this upper bound to 14 percent, which is near Cray Research, Inc.'s claim of a 10 percent degradation.

The product-line performance of the CRAY-1 I/O system should be significantly better than that reported in these tests because of improved performance and features in the product-line controller.[2] Improvements impacting these tests are summarized.

The new controller will support 18-sector tracks on the disks as opposed to the current 16-sector tracks, thus leading to a higher I/O transfer rate. The new controller will also eliminate the timing/control problem responsible for the "half-speed" writes. Finally, the product-line controller will relieve the CPU monitor from the responsibility of handling I/O transfers sector by sector. Monitor calls will be reduced to two per block (a block may be an arbitrary number of sectors) from the present system's two per sector. Thus the software overhead recorded in these tests should be significantly reduced.

These test results support our cursory analysis of degradation due to computation and I/O interference for memory access. We conclude that degradation should not be a serious concern, excepting the unlikely case of long periods of bank conflicts.

2. _CRAY-1 Computer System Mass Storage Subsystem Reference Manual_, Publication 2240630, Cray Research, Inc., 1976.

SECTION VII

INFORMAL ESTIMATES

In accordance with Section V of the Evaluation Plan (Appendix A), an informal estimate of the CRAY-1 scalar speed relative to the CDC 7600 is provided for the Class VI workload. Referring to Table III-6, the sum of the CRAY-1 CPU cycles for eight modules comprising the sample is 597, while for the CDC 7600 the sum of the CPU cycles is 761. This provides a preliminary speed ratio of 2.80. The two kernels that performed LCM (large core memory) to SCM (small core memory) transfers were ignored.

Adding a zero cycle count to the CRAY-1 total for these two kernels and the appropriate cycle count to the CDC 7600 total would result in a much larger ratio. However, considering the very small sample size, the value of this estimate did not justify implementing the kernels on the CDC 7600.

In accordance with Section V of the Evaluation Plan an informal estimate of the CRAY-1 scalar speed ratio relative to the CDC 7600 is shown for each of the Class VI applications workload codes in Table VII-1.

Table VII-1.
Informal estimates of Speed Ratios for the Three Applications Workload Codes.

CODE	Sum of CRAY-1 Cycles	Sum of CDC 7600 Cycles	CDC 7600 Time/ CRAY-1 Time
EOTOTO	533	595	2.46
GUANO	659	769	2.57
MCRAD	337	405	2.64

Because of the very small sample size for each code, the reader is advised to make assumptions based on these values with caution.

An informal estimate of the percentage of code "vectorizable" from inspection of the sample was to be provided in this section as per Section V of the Evaluation Plan. However, the small sample size (six) for each code limits the resolution of such an estimate to 16 percent. The reader may inspect

Table VII-2 and draw his own conclusions. The vectorization criteria referenced in the table are those established in the Evaluation Plan. The true vectorization potential of each code can be achieved only by the restructuring of its algorithms.

Table VII-2. Module "Vectorizability."

KERNEL ID	VECTORIZABLE? (IF YES IDENTIFY CRITERION)	
E1	NO	
E2	NO	
E3	NO	
E4	NO	
E5	NO	
E6	NO	
EH1	NO	
EH2	NO	
G1	NO	
G2	YES	1, 2 AND 3
G3	YES	1, 2 AND 3
G4	YES	1, 2 AND 3
G5	NO	
G6	YES	1, 2 AND 3
GH1	YES	1
GH2	YES	1, 2 AND 3
M1	NO	
M2	NO	
M3	NO	
M4	NO	
M5	NO	
M6	NO	
MH1	NO	
MH2	NO	
TH1	NO	
TH2	NO	
SH1	-	-
SH2	-	-

SECTION VIII

CONCLUSIONS

The purpose of this evaluation was to determine if the CRAY-1 computer meets the minimum performance standards set forth by LASL and ERDA to qualify the machine for further consideration for procurement. These standards are divided into specific qualification criteria in three main areas: scalar performance, vector performance and reliability.

Scalar Performance

The hypothesis that the CRAY-1 in scalar mode is at least two times faster than the CDC 7600 was tested on samples drawn from each of the three codes comprising the Class VI applications workload, and from a sample drawn equally from the five codes comprising the Class VI workload. The hypothesis test was structured so that the probability of a wrong result is less than 0.1. The hypothesis tested as true in all cases, with the minimum number of kernels necessary to test the hypothesis. Thus the CRAY-1 meets both the Class VI workload scalar performance criterion and the Class VI applications workload scalar performance criterion.

Kernel timings also provided estimates of the CDC 7600/CRAY-1 execution time ratios (speed ratios) for scalar computation. The speed ratios for all four workload samples ranged from about 2.5 to greater than 2.8. In addition, the speed ratio for the preliminary scalar test code was 2.5. From these results we conclude that the CRAY-1 in scalar mode has the potential for executing CPU-bound codes 2.5 times faster than the CDC 7600.

Vector Performance

Five operations were chosen to evaluate the CRAY-1's vector performance versus vector length. Each operation was coded as efficiently as was feasible for both the CDC 7600 and the CRAY-1. Each operation yielded a CDC 7600/CRAY-1 speed ratio for three vector lengths. The average of the five speed ratios was compared against the qualification criterion for each vector length. The average speed ratios were 3.39, 4.50 and 5.12 for vector lengths of 20, 100 and 500, respectively. Thus, speed ratio criteria of 3, 4 and 5 for vector lengths of 20, 100 and 500, respectively, were met.

Reliability

Reliability of the CRAY-1 was evaluated by running a reliability test code on the machine for long periods of time. The EXERCISER program was designed to access as many different hardware units of the machine as possible, at a rapid rate, for extended periods. The test program is characterized by access rates significantly higher than those of an initial production workload, and above those of longer term workloads. Mean-time-to-failure ranged from 2.53 hrs. to 7.08 hrs. for the reported periods. The criterion of at least four hours was exceeded for 7 of the 21 overlapping periods. Mean-time-to-repair ranged from 0.34 hrs. to 0.52 hrs., exceeding the criterion of no more than 1 hour for all reported periods. Likewise, system availability ranged from 0.85 to 0.95, exceeding the criterion of at least 0.80 for all reported periods. Reliability was good for a serial number one machine of this size and speed.

An analysis of machine failures revealed that memory parity errors dominated the statistics. If one assumes that single-bit memory error correction would have eliminated all memory failures, then a CRAY-1 with this feature would have resulted in a mean-time-to-failure measure approximately nine times greater than that observed over the entire six-month evaluation period, with a less dramatic increase in system availability. The indications are that the CRAY-1 with error correcting memory would be an exceptionally reliable machine.

Input/ Output Studies

Although no qualification criteria were established in the area of input/output (I/O) operations, a study of the CRAY-1 I/O subsystem was undertaken to determine if any pathologies existed. Performance tests uncovered the fact that disk revolutions were being missed during disk writes, due primarily to limitations in the interim disk controller. These limitations should not exist for the product-line controller. With this exception, expected transfer rates for the disk were observed. Head positioning times were close to expected values. An error detection test wrote and read over four billion words to disk with no errors.

The degradation to vector computation due to I/O interference by memory cycle stealing and I/O interrupts was measured to determine if this might pose a serious problem to CRAY-1 vector performance. Execution time degradations to vector computation in a worst case test with the single disk yielded degradations of 3.5 percent and 3.7 percent, which were assigned to memory cycle stealing and I/O interrupts, respectively. These low values, and a memory bandwidth analysis of the system, indicate that I/O interference to vector computation should not be a matter of serious concern.

In summary, disk performance for a production system should be close to that expected. Early detection of the write problem has prompted Cray Research, Inc., to attempt a hardware modification to the interim disk controller in order to bring write transfer rates up to expected values.

Conclusion

The conclusion of this evaluation is that the CRAY-1 meets every performance criterion established by LASL and ERDA to qualify the machine for further consideration for procurement.

APPENDIX A

The computer configuration being evaluated includes the following units.

- The CRAY-1 central processing unit, with 524 288 words (64 bits plus one parity bit) of memory arranged in 16 banks;
- One disk control unit (DCU);
- Two 819 disk units;
- Twelve independent channels (asynchronous, full duplex);
- One Data General Eclipse station, with the following input-output units:
    - One TEC 455 display,
    - One TEC 1440 display,
    - One Gould 5000 printer,
    - One Documation M1000 card reader,
    - One Century Data disk, and
    - One Data General nine-track tape.

THE CRAY - 1 COMPUTER

The Cray Research, Inc. CRAY-1 Computer System is a large-scale, general-purpose digital computer featuring vector as well as scalar processing, a 12.5 nanosecond clock period, and a 50 nanosecond memory cycle time. The CRAY-1 is capable of executing over 80 million floating point operations per second. Even higher rates are possible with programs that take advantage of the vector features of the computer.

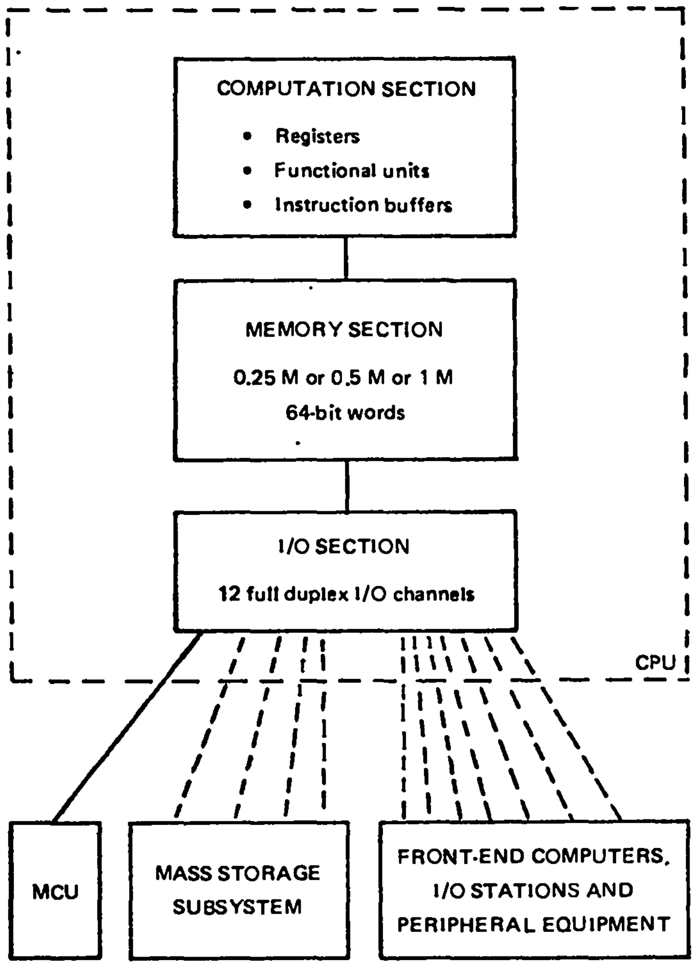
The CRAY-1 is particularly adapted to the needs of the scientific community and is especially useful in solving problems requiring the analysis and prediction of the behavior of physical phenomena through computer simulation. The fields of weather forecasting, aircraft design, nuclear research, geophysical research, and seismic analysis involve this process. For example, the movements of global air masses for weather forecasting, air flows over wing and airframe surfaces for aircraft design, and the movements of particles for nuclear research, all lend themselves to such simulations. In each scientific field, the equations are known but the solutions require extensive computations involving large quantities of data. The quality of a solution depends heavily on the number of data points that can be considered and the number of computations that can be performed. The CRAY-1 provides substantial increases with respect to both the number of data points and computations so that researchers can apply the CRAY-1 to problems not feasibly solvable in the past.

CONFIGURATION

The basic configuration of the CRAY-1 consists of the central processor unit (CPU), power and cooling equipment, one or more minicomputer consoles, and a mass storage (disk) subsystem. The CPU holds the computation, memory, and I/O sections of the computer. A minicomputer serves either as a maintenance control unit or a job entry station.

INPUT/OUTPUT

Input/output is via twenty-four I/O channels, twelve of which are input and twelve output. Any number of channels may be active at a given time. The channel transfer rate is based on the channel width (currently 8 or 16 bits). For a 16 bit channel, maximum rates of 160 million bits per second are attainable. Higher rates are possible with wider channels. In practice, this theoretical transfer rate is limited by the speed of peripheral devices and by memory reference activity of the CPU.



BASIC COMPUTER SYSTEM

MEMORY

The CRAY-1 memory is constructed of 1024-bit LSI chips. Up to 1,048,576 (generally referred to as one million) 64-bit words are arranged in 16 banks. The bank cycle time, that is, the time required to remove or insert an element of data in memory, is 50 nanoseconds. This short cycle time provides an extremely efficient random-access memory. One parity bit per word is maintained in 16 modules of the central processor. There is no inherent memory degradation for machines with less than one million words of memory.

FACTS AND FIGURES

CPU	
Instruction size	16 or 32 bits
Clock period	12.5 nsec
Instruction stack/buffers	64 words (4096 bits)
Functional units	twelve: 3 integer add; 1 integer multiply; 2 shift; 2 logical; 1 floating add; 1 floating multiply; 1 reciprocal approx.; 1 population count
Programmable registers	8x64 64-bit; 73 64-bit; 72 24 bit; 1 7-bit
Max. vector result rate	12.5 nsec / unit
FLOATING POINT COMPUTATION RATES (results per second)	
Addition	80×10^6 / sec
Multiplication	80×10^6 / sec
Division	25×10^6 / sec
MEMORY	
Technology	bipolar semiconductor
Word length	64 bits
Address space	4M words
Data path width (bits)	64 (1 word)
Cycle time	50 nsec.
Size	262,144 words or 524,288 words or 1,048,576 words
Organization / interleave	16 banks
Maximum band width	80×10^6 words / sec (5.1×10^9 bits / sec)
Error checking	1 parity bit / word
PHYSICAL CHARACTERISTICS / ELECTRONIC TECHNOLOGY	
Size of CPU cabinet	9 ft diameter base; 4.5 ft diameter center; 6 ft height
Weight of mainframe	5 tons
Cooling	Freon
Plug-in modules	1506
Module types	109
PC boards	5 layer
Circuitry (equivalent no. of transistors)	2.5M
Logic	ECL, 1 nsec.
High-density logic	SSI

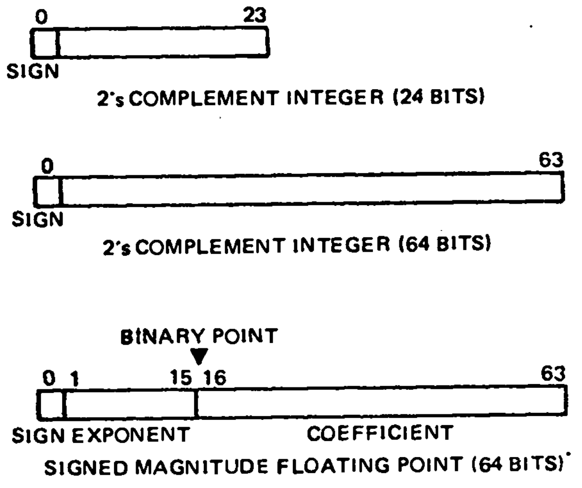
COMPUTATION SECTION

The computation section as illustrated on page 4 is composed of instruction buffers, registers, and functional units which operate together to execute sequences of instructions.

Data structure

Internal character representation in the CRAY-1 is in ASCII with each 64-bit word able to accommodate eight characters.

Numeric representation is either in two's complement form (24-bit or 64-bit) or in 64-bit floating point form using a signed magnitude binary coefficient and a biased exponent. Exponent overflow and underflow is caused if the exponent is greater than 57777_8 or less than 20000_8. For scalar operations, either of these conditions causes an interrupt except where the interrupt has been inhibited. For vector operations, these conditions do not cause an interrupt.


0
23
SIGN
2's COMPLEMENT INTEGER (24 BITS)
0
63
SIGN
2's COMPLEMENT INTEGER (64 BITS)
BINARY POINT
0 1
15 16
63
SIGN EXPONENT
COEFFICIENT
SIGNED MAGNITUDE FLOATING POINT (64 BITS)

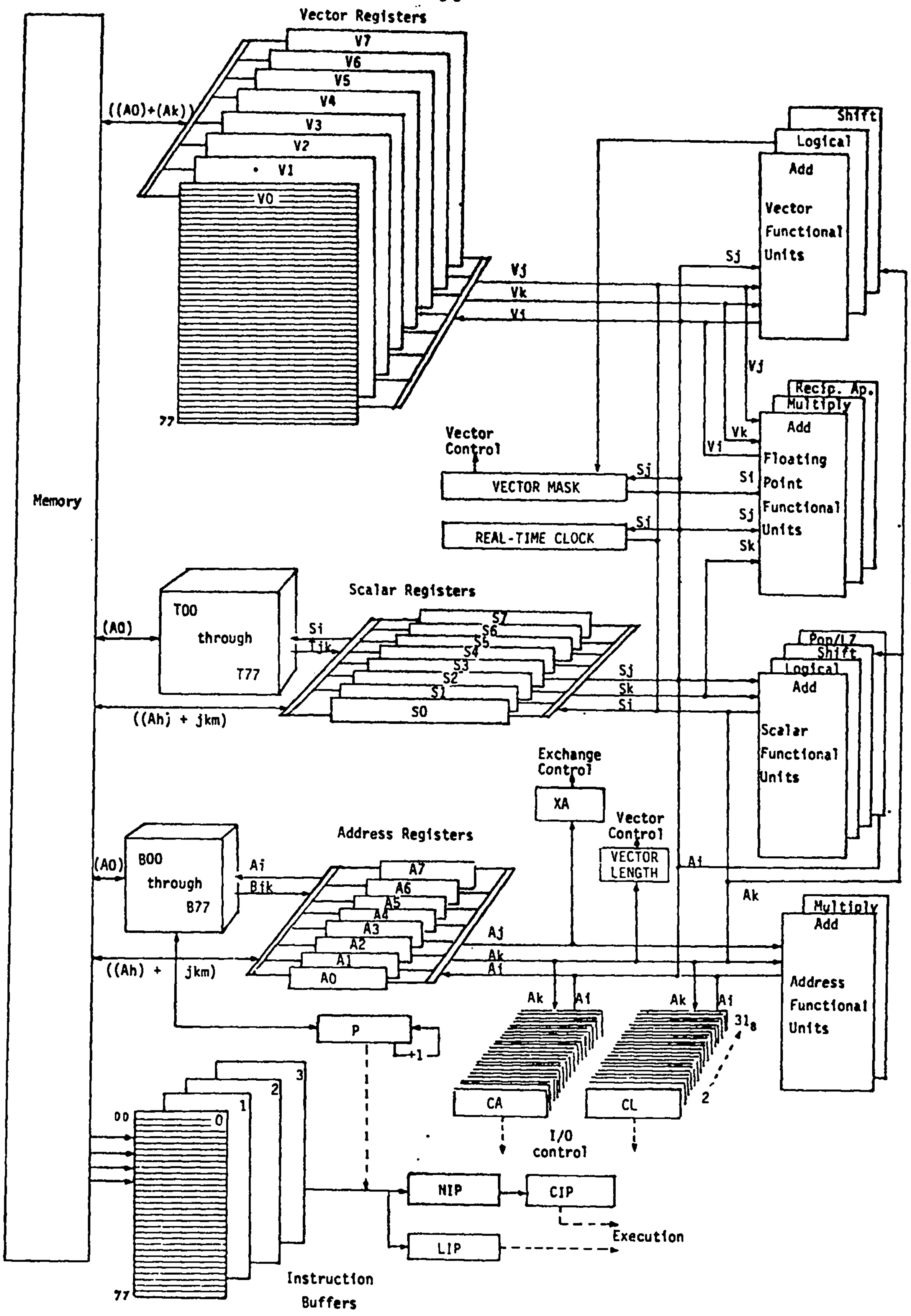
DATA FORMATS

Instruction set

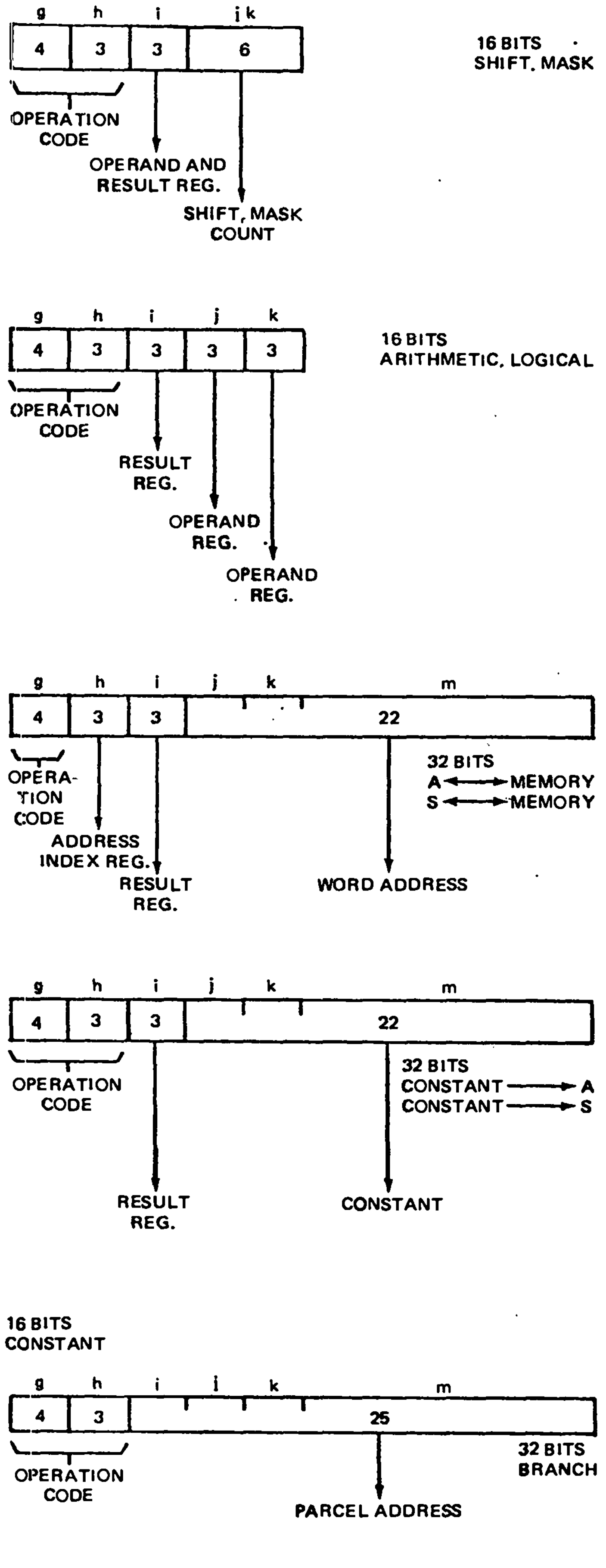
The CRAY-1 executes 128 operation codes as either, 16-bit (one parcel) or 32-bit (two-parcel) instructions. Operation codes provide for both scalar and vector processing.

In general, an instruction that references registers occupies one parcel: an instruction that references memory occupies two parcels. All of the arithmetic and logical instructions reference registers.

Floating point instructions provide for addition, subtraction, multiplication, and reciprocal approximation. The reciprocal approximation instruction allows for the computation of a floating point divide operation using a multiple instruction sequence.



COMPUTATION SECTION



INSTRUCTION FORMATS

Integer or fixed point operations are provided for as follows: integer addition, integer subtraction, and integer multiplication. An integer multiply operation produces a 24-bit result; additions and subtractions produce either 24-bit or 64-bit results. No integer divide instruction is provided. The operation can be accomplished through a software algorithm using floating point hardware.

The instruction set includes Boolean operations for OR, AND, and exclusive OR and for a mask-controlled merge operation. Shift operations allow the manipulation of 64- or 128-bit operands to produce a 64-bit result. Similar 64-bit arithmetic capability is provided for both scalar and vector processing. Full indexing capability allows the programmer to index throughout memory in either scalar or vector modes of processing. This allows matrix operations in vector mode to be performed on rows, on columns, or on the diagonal.

Addressing

Instructions that reference data do so on a word basis. Instructions that alter the sequence of instructions being executed, that is, the branch instructions, reference parcels of words. In this case, the lower two bits of an address identify the location of an instruction parcel in a word.

Instruction buffers

All instructions are executed from four instruction buffers, each consisting of 64 16-bit registers. Associated with each instruction buffer is a base address register that is used to determine if the current instruction resides in a buffer. Since the four instruction buffers are large, substantial program segments can reside in them. Forward and backward branching within the buffers is possible and the program segments may be noncontiguous. When the current instruction does not reside in a buffer, one of the instruction buffers is filled from memory. Four memory words are transferred per clock period. The buffer that is filled is the one least recently filled, that is, the buffers are filled in rotation. To allow the current instruction to issue as soon as possible, the memory word containing the current instruction is among the first four transferred. A parcel counter register (P) points to the next parcel to exit from the buffers. Prior to issue, instruction parcels may be held in the next instruction parcel (NIP), lower instruction parcel (LIP) and current instruction parcel (CIP) registers.

Operating registers

The CRAY-1 has five sets of registers, three primary and two intermediate. Primary registers can be accessed directly by functional units. Intermediate registers are not accessible by functional units but act as buffers between primary registers and memory.

The figure on page 4 represents the CRAY-1 registers and functional units. The 64 address and 64 scalar intermediate registers can be filled by block transfers from memory. Their purpose is to reduce memory references made by the scalar and address registers.

The eight address registers are each 24 bits and can be used to count loops. provide shift counts. and act as index registers in addition to their main use for memory references.

The eight 64-bit scalar registers in addition to contributing operands and receiving results for scalar operations can provide one operand for vector operations.

Each of the eight vector (V) registers is actually a set of 64 64-bit registers. called elements. The number of vector operations to be performed (that is. the vector length) determines how many of the elements of a register are used to supply operands in a vector set or receive results of the vector operation. The hardware accommodates vectors with lengths up to 64; longer vectors are handled by the software dividing the vector into 64-element segments and a remainder.

Associated with the vector registers are a 7-bit vector length register and a 64-bit vector mask register. The vector length register, as its name implies, determines the number of operations performed by a vector instruction. Each bit of the vector mask register corresponds to an element of a V register. The mask is used with vector merge and test instructions to allow operations to be performed on individual vector elements.

Supporting registers

In addition to the operating registers, the CPU contains a variety of auxiliary and control registers. For example, there is a channel address (CA) register and a channel limit register (CL) for each I/O channel.

Functional units

Instructions other than simple transmits or control operations are performed by hardware organizations known as functional units. Each of the twelve units in the CRAY-1 executes an algorithm or a portion of the instruction set. Units are independent. A number of functional units can be in operation

A functional unit receives operands from registers and delivers the result to a register when the function has been performed. The units operate essentially in three-address mode with source and destination addressing limited to register designators.

All functional units perform their algorithms in a fixed amount of time. No delays are possible once the operands have been delivered to the unit. The amount of time required from delivery of the operands to the unit to the completion of the calculation is termed the "functional unit time" and is measured in 12.5 nsec clock periods.

The functional units are all fully segmented. This means that a new set of operands for unrelated computation may enter a functional unit each clock period even though the functional unit time may be more than one clock period. This segmentation is made possible by capturing and holding the information arriving at the unit or moving within the unit at the end of every clock period.

The twelve functional units can be arbitrarily assigned to four groups: address, scalar, vector, and floating point. The first three groups each acts in conjunction with one of the three primary register types, to support address, scalar, and vector modes of processing. The fourth group, floating point, can support either scalar or vector operations and will accept operands from or deliver results to scalar or vector registers accordingly.

FUNCTIONAL UNITS

Functional Unit	Unit Time (Clock Periods)	Instructions
Address integer add	2	030, 031
Address multiply	6	032
Scalar integer add	3	060, 061
Scalar logical	1	042 - 051
Scalar shift	2	052 - 055
	3	056, 057
Scalar leading zero/pop count	4	026
	3	027
Vector integer add	3	154 - 157
Vector logical	2	140 - 147, 175
Vector shift	4	150 - 153
Floating point add	6	062, 063, 170 - 173
Floating point multiply	7	060 - 067, 160 - 167
Floating point reciprocal	14	070, 174

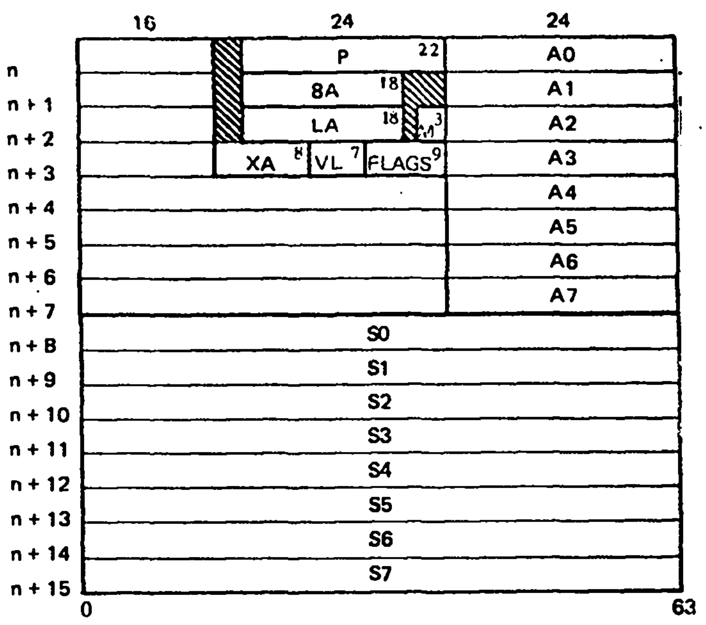
Memory field protection

Each object program has a designated field of memory. Field limits are defined by a base address register and a limit address register. Any attempt to reference instructions or data beyond these limits results in a range error.

Exchange mechanism

The technique employed in the CRAY-1 to switch execution from one program to another is termed the exchange mechanism. A 16-word block of program parameters is maintained for each program. When another program is to begin execution, an operation known as an exchange sequence is initiated. This sequence causes the program parameters for the next program to be executed to be exchanged with the information in the operating registers to be saved. The operating register contents are thus saved for the terminating program and entered with data for the new program.

Exchange sequences may be initiated automatically upon occurrence of an interrupt condition or may be voluntarily initiated by the user or by the operating system through normal and error exit instructions.



Flags*		Modes*	
31	Console Interrupt	37	Interrupt on Floating Point
32	RTC Interrupt	38	Interrupt on Storage Parity
33	Floating Point Error (Scalar Reference Only)	39	Monitor Mode
34	Operand Range		
35	Program Range		
36	Storage Parity		
37	I/O Interrupt		
38	Error Exit		
39	Normal Exit		

P = Program Address
BA = Base Address
LA = Limit Address
XA = Exchange Address
VL = Vector Length

* Bit position from left of word

EXCHANGE PACKAGE

ARCHITECTURE

Construction

The CRAY-1 is modularly constructed of 1506 modules held by 24 chassis. Each module contains two 6 in. by 8 in. printed circuit boards on which are mounted a maximum of 144 integrated circuit packages per board. Emitter coupled logic (ECL) is used throughout. Four basic chip types are used: a high-speed 5/4 NAND gate, a slow-speed 5/4 NAND gate, a 16x1 register chip, and a 1024x1 memory chip.

Appearance

The esthetics of the machine have not been neglected. The CPU is attractively housed in a cylindrical cabinet. The chassis are arranged two per each of the twelve wedge-shaped columns. At the base are the twelve power supplies. The power supply cabinets. which extend outward from the base are vinyl padded to provide seating for computer personnel.

The compact mainframe occupies a mere 70 sq. ft. of floor space.

Cooling

The speed of the CPU is derived largely by keeping wire lengths extremely short in the mainframe. This. in turn. necessitates a dense concentration of components with an accompanying problem of heat dissipation. The Freon cooling system used in the CRAY-1 employs the latest in refrigeration technology to maintain a column temperature of about 68° in the unit.

MAINTENANCE CONTROL UNIT (MCU)

A 16-bit minicomputer system serves as a maintenance control unit. The MCU performs system initialization and basic recovery for the operating system. Included in the MCU system is a software package that enables the minicomputer to monitor CRAY-1 performance during production hours.

STATIONS

The CRAY-1 computer system may be equipped with one or more 16-bit minicomputer systems that provide input data to the CRAY-1 and receive output from the CRAY-1 for distribution to a variety of slow-speed peripheral equipment. A station consists of a Data General S-200 minicomputer or equivalent. Peripherals attached to the station vary depending on whether the station is a local or remote job entry station or a data concentrator used for multiplexing several remote stations.

EXTERNAL INTERFACE

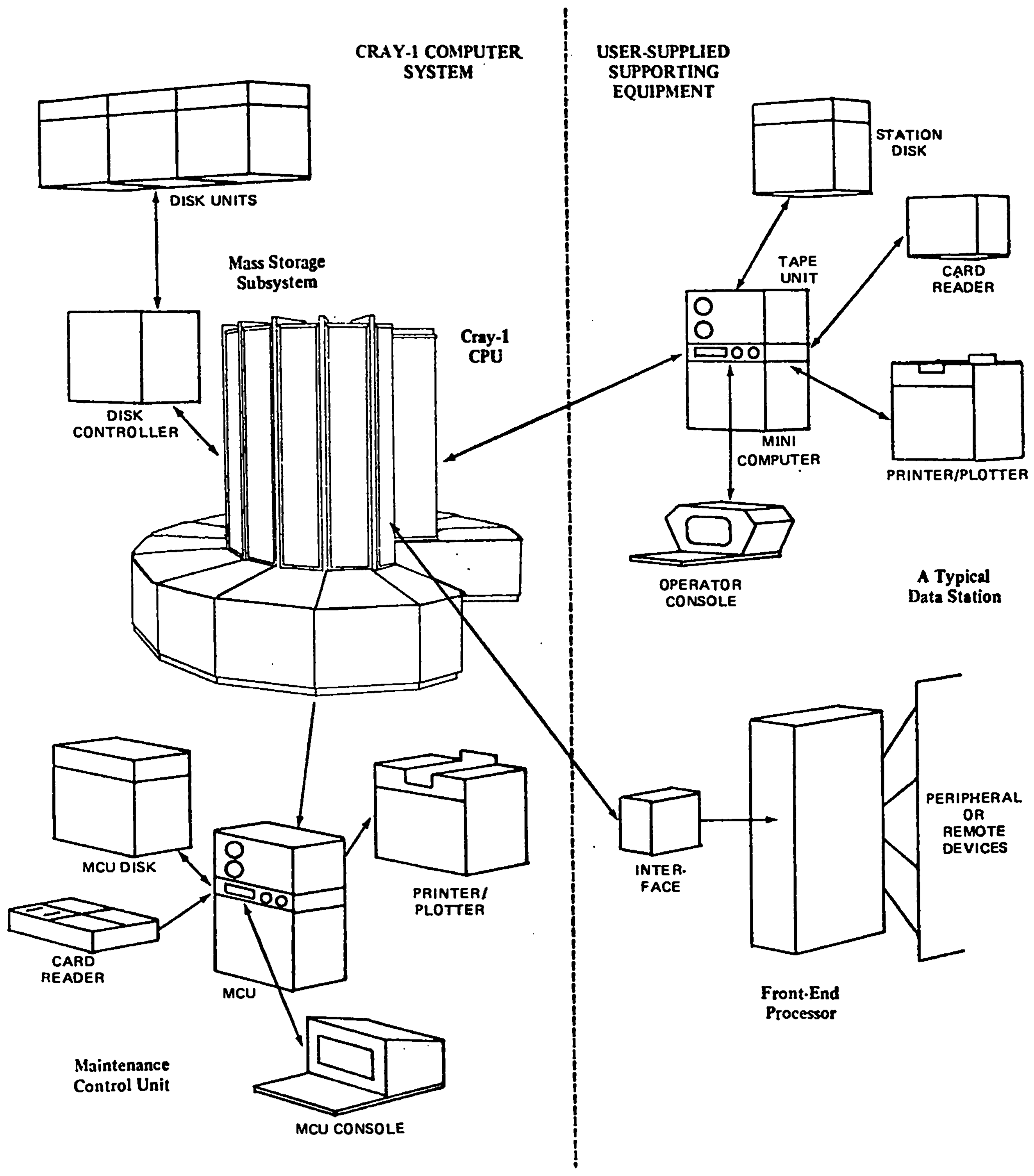
The CRAY-I may be interfaced to front-end host systems through special controllers that compensate for differences in channel widths, machine word size, electrical logic levels, and control protocols. The interface is a Cray Research, Inc. product implemented in logic compatible with the host system.

SYSTEM MASS STORAGE

System mass storage consists of two or more Cray Research, Inc. DCU-2 Disk Controllers and multiple DD-19 Disk Storage Units. The disk controller is a Cray Research. Inc. product and is implemented in ECL logic similar to that used in the mainframe. Each controller may have four DD-19 disk storage units attached to it. Operational characteristics of the DD-19 units are summarized in the accompanying table.

CHARACTERISTICS
OF DD-19 DISK STORAGE UNIT

Bit capacity per drive	2.424×10^9
Tracks per surface	411
Sectors per track	18
Bits per sector	32,768
Number of head groups	10
Recording surfaces per drive	40
Latency	16.6 msec
Access time	15 - 80 msec
Data transfer rate (average bits per sec.)	35.4×10^6
Total bits that can be streamed to a unit (disk cylinder capacity)	5.9×10^6



DATA FLOW THROUGH SYSTEM

MAINTENANCE SERVICES

Cray Research, Inc. provides resident maintenance engineers on a contractual basis.

Evaluation Plan

I. INTRODUCTION

The growing complexity of the programmatic efforts at the Los Alamos Scientific Laboratory (LASL) in weapons design, laser fusion and isotope separation, controlled thermo-nuclear research, reactor safety, and other basic and exploratory research requires calculations that are beyond the capability of the fastest computer currently installed in the LASL Central Computing Facility—the CDC 7600. For this reason, LASL, in conjunction with the Energy Research and Development Administration (ERDA) is evaluating a new class of scientific computer system. This class has been identified by ERDA as "Class VI."

A Class VI computer system is one that is capable of executing 20 to 60 million floating point operations per second (MFLOPS). Table 1 lists the various classes of computer systems currently identified by ERDA.

CLASS	MFLOPS	REPRESENTATIVE COMPUTERS
III	.5 - 2	CDC 6600, IBM 370/158
IV	2 - 6	CDC 7600, IBM 370/195
V	6 - 20	CDC STAR 100, TI-ASC4X
VI	20 - 60	(NEW)

TABLE 1. ERDA COMPUTER SYSTEM CLASS DEFINITIONS.

It now appears that there is at least one existing computer system that may meet LASL Class VI computing requirements—the Cray Research CRAY-1. The purpose of this document is to specify the approach to be taken by LASL and ERDA in evaluating the CRAY-1 computer system.

The CRAY-1 computer is installed at LASL for approximately six months, during which time the evaluation described in this document is taking place. The CRAY-1 configuration being evaluated is described in Appendix A. A summary of CRAY-1 characteristics is shown in Appendix A.

II. OBJECTIVES OF THE CRAY-1 EVALUATION

This evaluation has two main objectives. The first is to assess the hardware performance of the CRAY-1 system. The second is to determine the performance characteristics of the CRAY-1 system as it applies to a specifically chosen LASL workload. To meet these two major objectives, the CRAY-1 evaluation has been divided into two closely related phases: hardware performance and applications performance.

III. MANDATORY REQUIREMENT AND QUALIFICATION CRITERIA

The mandatory requirement is that the CRAY-1 must substantially meet the qualification criteria described in this section. Details of the evaluation methodology are given in section IV.

A. Hardware Performance Qualification Criteria

1. Scalar Operations.

The CRAY-1 must be capable of executing scalar operations at a rate at least twice that of the CDC 7600, while operating upon floating point data at least 48 bits in length.

2. Vector Operations.

The CRAY-1 is capable of performing vector operations; the CDC 7600 does not have this capability. Therefore, the following vector hardware qualification criteria for the CRAY-1 are specified with respect to the rate at which a CDC 7600 can perform the equivalent function. Both machines must operate upon floating point data at least 48 bits in length.

a. Short Vectors. Short vectors are defined to have a vector length of 20. The CRAY-1 must be capable of executing vector operations of this length at a rate at least three times that at which the CDC 7600 can perform the equivalent function.

b. Medium Vectors. Medium vectors are defined to have a vector length of 100. The CRAY-1 must be capable of executing vector operations of this length at a rate at least four times that at which the CDC 7600 can perform the equivalent function.

c. Long Vectors. Long vectors are defined to have a vector length of 500. The CRAY-1 must be capable of executing vector operations of this length at a rate at least five times that at which the CDC 7600 can perform the equivalent function.

3. Reliability.

There are three evaluation areas with respect to CRAY-1 reliability characteristics. These areas are System Availability (SA), Mean Time to Failure (MTTF) and Mean Time to Repair (MTTR). The criteria specified for all three areas must be met for any single contiguous period of 20 workdays during the six-month evaluation period. The following measures will be used in determining the CRAY-1 reliability characteristics:

- Wall Clock Time (WC) - the elapsed time of the measured period.

- Remedial Maintenance (RM) - that portion of WC during which the CRAY-1 is in use by Cray Research, Inc., to rectify a problem.

- No Time (NT) - that portion of WC that is nonproductive because of such things as no work available, building facilities failure, preventive maintenance, Cray Research software development or Cray Research engineering modifications.

- Operational Use Time (OUT) - that portion of WC during which the CRAY-1 is performing productive work. It is specifically defined as:

$$OUT = WC-RM-NT.$$

The qualification criteria in each reliability area are identified below.

a. System Availability. System Availability (SA) is defined as follows:

$$SA = OUT/(OUT + RM).$$

System availability on the CRAY-1 must equal or exceed 0.8 for a contiguous 20-workday period during the evaluation.

b. Mean Time To Failure (MTTF). Mean Time to Failure is defined as the ratio of OUT to the number of OUT periods (OUT periods not terminated by a machine failure are concatenated with succeeding OUT periods). MTTF for the CRAY-1 system must equal or exceed four hours for a contiguous 20-workday period during the evaluation.

c. Mean Time To Repair (MTTR). Mean Time to Repair is the ratio of RM to the number of RM periods. MTTR must be less than or equal to one hour for a contiguous 20-workday period during the evaluation.

Table 2 summarizes the hardware qualification criteria for the CRAY-1 evaluation.

FUNCTIONS	QUALIFICATION CRITERIA
Scalar Operations	$\geq$ 2 X (CDC 7600)
Short Vector Operations	$\geq$ 3 X (CDC 7600)
Medium Vector Operations	$\geq$ 4 X (CDC 7600)
Long Vector Operations	$\geq$ 5 X (CDC 7600)
MTTF	$\geq$ 4 hours*
MTTR	$\leq$ 1 hour*
System Availability	$\geq$ 0.8*

*These three criteria must collectively be met during any contiguous 20 workday period.

TABLE 2. Hardware Performance Qualification Criteria.

B. Applications Performance Qualification Criterion

Using the procedures defined in section IV.B, the hypothesis that the CRAY-1 is two times faster than the CDC 7600 upon the specified LASL Class VI applications workload must be satisfied.

IV. EVALUATION METHODOLOGY

As stated earlier, the CRAY-1 evaluation will be accomplished in two closely related phases: the hardware performance evaluation and the application performance evaluation. Each is discussed separately.

Much of the evaluation methodology uses statistical sampling methods. Some specific techniques of sampling and evaluation to be used are currently being determined and will be attached as an addendum to this plan. If during the course of the evaluation it is evident that a change in methods would result in more accurate results, this plan will be modified accordingly. Any changes in the methodology will be thoroughly documented.

A. Hardware Performance

1. Preliminary Scalar Performance Estimate

Since the scalar performance of the CRAY-1 on the LASL Class VI workload cannot be reliably estimated until the evaluation is complete, a preliminary test of the CRAY-1 scalar performance is being conducted for the interim report. A test routine extracted from LASL production codes will be executed on both the CDC 7600 and the CRAY-1. This routine is documented in Appendix B. The routine is coded in assembly language as efficiently as is feasible for each machine. A preliminary estimate of CRAY-1 scalar performance is the ratio of execution times for the two machines. This estimate will be included in the final report for informative purposes only.

In addition, a summary of the results obtained from the module timings (Steps 3 and 5 below) completed by June 15 will be included in the interim report.

2. Scalar Operations

Step 1. Workload Analysis. To determine which LASL codes are potential candidates for execution on the Class VI computer system, it is necessary to investigate the codes currently active at LASL. This will be accomplished for codes executing under the CROS operating system by examining the historical data base maintained by LASL. All codes requiring more than one CDC 7600 CPU hour per execution will be identified for further individual analysis. Similarly, the LTSS historical data base will be examined for those accounts that consume more than one CDC 7600 CPU hour in one 24-hour day. These codes are labeled the "potential Class VI workload."

A review process with the users will then be conducted in order to finalize and establish the set of LASL codes that will comprise the LASL Class VI workload. These codes will be documented on forms shown in Figures 1 and 2.

Step 2. Code Execution. Each code selected will be executed in a typical manner on the CDC 7600. This execution will be monitored by the LASL-written software monitor STAT. The entire program field length (of size FL) will be divided into b "buckets", each of size ℓ words, such that $b*\ell \equiv FL$. STAT samples the P-counter value (the execution address) every 3.6 ms, determines which bucket encompasses the address, and increments a count for that bucket. At the completion of the program execution, one obtains the frequency f_i that the program was found executing within the ℓ words of bucket i. A cumulative distribution is then obtained for the code by bucket i, giving the fraction of time spent by the code in execution in or below the addresses of bucket i.

The measurements will be conducted during a sufficiently long period to exercise the normally used features of each code during a representative run. The object of this step is the acquisition of representative cumulative distributions, as described above, for each code.

Step 3. Module Selection. A module is a segment of Fortran statements residing in a Class VI workload code. A number of modules (the sample size) will be drawn from each code. The sample size required to provide a statistically valid result is currently being determined and will be documented before Step 3 is completed and will be part of the statistics addendum to this plan.

The procedure for drawing a module will be to obtain a random number, evenly distributed from 0 to 1, from the Fortran library function RANF. The generated random number, x, will be applied to the inverse of the cumulative distribution of Step 2 above to identify the bucket associated with x. An additional STAT run with one-word resolution will be made to identify the address associated with x and to further characterize program behavior within the selected bucket.

A portion of the segment of Fortran statements associated with the bucket will be selected as the Fortran module. This will be accomplished according to the following procedure.

a. If the address boundaries of the bucket result in incomplete Fortran statements, expand the segments to include entire statements.

b. Starting at the selected address go "backward in execution" to either the segment boundary or a labeled statement actively accessed from outside the segment. "Active" execution paths are those taken during execution of the code in the previous step. Whether an execution path is active or inactive may be inferred from the STAT histogram of execution within the segment. The selected statement forms the first statement of the Fortran module.

c. From the first statement of the Fortran module go "forward in execution" to either the segment boundary or an active jump outside the segment. The subsegment encountered in this action comprises the Fortran module. Those modules selected will be listed on the form shown in Figure 3.

The bucket size is presently being defined and will be included in the statistics addendum to this plan.

Step 4. Module Optimization. Each module chosen from Step 3 will be coded as efficiently as is feasible in assembly language for each machine. Only scalar operations will be used for the CRAY-1. The source language statements will be consulted as an aid in determining the algorithm a selected code segment represents.

Step 5. Module Timings. The assembly language versions of the modules will be executed and timed on the two machines. Timings will be listed on the form shown in Figure 3.

Step 6. Evaluation Code Scalar Speed Ratio Hypothesis Test. The hypothesis that the CRAY-1, in scalar mode, is two times faster than the CDC 7600 will be tested using the module timings obtained in Step 5. The specific procedure to perform this test is currently being determined. It will be part of the statistics addendum to this plan.

3. Vector Operations

a. A parameter-driven test routine will be written as efficiently as is feasible for each machine in assembly language to determine the vector performance of the CRAY-1. The test routine will consist of vector algorithms extracted from LASL production codes. The algorithms comprising the test routine are presented in Appendix B.

b. The vector length for the test routine will be an input parameter modified according to the vector length being tested (short, medium, or long).

c. The test routine will be executed on both the CDC 7600 and the CRAY-1 systems, and a ratio of execution times will be obtained.

d. The qualification criterion, as stated in section III for each vector length, must be met by the test routine timings. The timing results will be documented on the form shown in Figure 4.

4. Reliability

The reliability evaluation will be accomplished using the EXERCISER program currently in existence at LASL. Figure 5 shows the form to be used to document the operation of the CRAY-1 when running the EXERCISER program.

Data collected on the form shown in Figure 5 will then be used as the basis for calculating reliability statistics for a 20-workday sliding window as follows:

a. System Availability. System availability (SA) will be calculated weekly from Figure 5 data as follows:

$$SA = OUT/(OUT+RM)$$

$$OUT = \sum_{i=1}^{N} OUT_i \quad \text{and} \quad RM = \sum_{j=1}^{M} RM_j$$

where OUT_i is an "OUT" period, RM_j is an "RM" period, N is the number of machine failures (the number of "OUT" periods) and M is the number of remedial maintenance (RM) periods.

b. Mean Time To Failure (MTTF). The MTTF will be calculated weekly from data recorded on the form shown in Figure 5 as:

$$MTTF = OUT/N.$$

c. Mean Time To Repair (MTTR). The MTTR will be calculated weekly from data recorded on the form shown in Figure 5 as:

$$MTTR = RM/M.$$

The statistics described above in sections 3a-c will each be calculated weekly during the evaluation period. The qualification criteria for all three must be satisfied during the same 20-contiguous-workday period. The form shown in Figure 6 will be used to document the reliability phase of the CRAY-1 evaluation.

B. Applications Performance

Step 1. Workload Analysis. To determine which LASL codes are potential candidates for the Class VI computer system it is necessary to investigate the codes currently active at LASL. This will be accomplished for those codes executing under the CROS operating system by examining the historical data base on CROS code execution maintained by LASL. All codes requiring more than one CDC 7600 CPU hour per execution will be identified for further individual analysis. Similarly, the LTSS historical data base will be examined for those executions consuming more than one CDC 7600 CPU hour in a 24-hour day under the LTSS operating system. These codes are labeled the "potential Class VI workload."

A review process with the users will then be conducted in order to finalize and establish the set of LASL codes that will comprise the LASL Class VI workload. These codes will be documented on forms shown in Figures 1 and 2.

Step 2. Applications Workload Performance Specifications. LASL will specify a subset of the Class VI workload to be used for the applications performance phase of the evaluation. This list of codes is termed the Class VI applications workload. LASL will also provide a decision rule to determine if the CRAY-1 meets the applications performance criterion. This specification must be provided by LASL before Step 3 is begun, and it must then be approved by the Division of Military Application (DMA). This list and the associated decision rule will be documented as an addendum to this plan.

Step 3. Code Execution. Each code selected will be executed in a typical manner on the CDC 7600. This execution will be monitored by the LASL-written software monitor STAT. The entire field length of the program will be divided into buckets. STAT samples the P-counter, determines which bucket encompasses the address, and increments a count for that bucket. A cumulative distribution is then obtained which gives the fraction of time spent by the code in execution in or below the address range for each bucket.

The measurements will be conducted during a sufficiently long period to exercise the normally used features of each code during a representative run. The object of this step is the acquisition of representative cumulative distributions, as described above, for each code.

Step 4. Module Selection. A module is a segment of Fortran statements residing in a Class VI workload code. A number of modules (the sample size) will be drawn from each code. The sample size required to provide a statistically valid result is currently being determined and will be documented before Step 4 is completed and will be part of the statistics addendum to this plan.

The procedure for drawing a module will be to obtain a random number, evenly distributed from 0 to 1, from the Fortran library function RANF. The generated random number, x, will be applied to the inverse of the cumulative distribution of Step 3 above to identify the bucket associated with x. An additional STAT run with one-word resolution will be made to identify the address associated with x and to further characterize program behavior within the selected bucket.

A portion of the segment of Fortran statements associated with the bucket will be selected as the Fortran module. This will be accomplished according to the following procedure.

a. If the address boundaries of the bucket result in incomplete Fortran statements, the segment will be expanded to include entire statements.

b. Starting at the selected address go "backward in execution" to either the segment boundary or a labeled statement actively accessed from outside the segment. "Active" execution paths are those taken during execution of the code in the previous step. Whether an execution path is active or inactive may be inferred from the STAT histogram of execution within the segment. The selected statement forms the first statement of the Fortran module.

c. From the first statement of the Fortran module go "forward in execution" to either the segment boundary or an active jump outside the segment. The subsegment encountered in this action comprises the Fortran module. Those modules selected will be listed on the form shown in Figure 3.

<u>Step 5</u>. <u>Module Optimization</u>. Each module chosen from Step 4 will be coded as efficiently as is feasible in assembly language for each machine. Only scalar operations will be used for the CRAY-1. The source language statements will be consulted as an aid in determining the algorithm a selected code segment represents.

<u>Step 6</u>. <u>Module Timings</u>. The assembly language versions of the modules will be executed and timed on the two machines. Timings will be listed on the form shown in Figure 3.

<u>Step 7</u>. <u>Evaluation Code Scalar Speed Ratio Hypothesis Test</u>. The hypothesis that the CRAY-1, in scalar mode, is two times faster than the CDC 7600 will be tested for each Class VI applications workload code using the module timings obtained in Step 6. (The specific procedure to perform this test is currently being determined and will be part of the statistics addendum to this plan.) The decision rule specified in Step 2 will then be applied.

C. <u>Input/Output Studies</u>

Although the configuration delivered with the CRAY-1 precludes a complete evaluation of the I/O capabilities of the system, a series of disk studies will be performed. At this time, the specific disk requirements and evaluations are undetermined. It is, however, anticipated that tests in the following areas will be accomplished:

- data rate - sustained and instantaneous
- positioning times
- data integrity
- error recovery
- impact of I/O on the CPU and memory.

Specific tests will be developed by the project team and documented as an addendum to this plan. This documentation will be accomplished by approximately June 30.

V. <u>INFORMAL ESTIMATES</u>

A. <u>Scalar Performance Estimate</u>

The CRAY-1/CDC 7600 scalar speed ratio will also be estimated based on the module timings obtained during the scalar performance evaluation.

Since the accuracy of this estimate cannot be verified statistically within the time constraints of this evaluation, it will be included in the final report for informative purposes only.

B. <u>Applications Performance Estimate</u>

The CRAY-1/CDC 7600 scalar applications speed ratio will be estimated based on the module timings obtained during the applications performance evaluation. Since the accuracy of this estimate cannot be verified statistically within the time constraints of this evaluation, it will be included in the final report for informative purposes only.

C. <u>Class VI Applications Workload Vectorization Estimate</u>

An estimate of the percentage of vectorization immediately obtainable for the Class VI applicaticns workload will be provided in the final report for informative purposes only.

This estimate will be obtained by examining the source code surrounding each selected module of the Class VI applications workload and determining if it is vectorizable. Vectorization criteria will be established and documented before the modules are examined for vectorizability. Module vectorizability will be summarized on the form shown in Figure 7.

An attempt will also be made to provide an estimate of a representative vector length for vectorizable sections of code.

VI. REPORTS

A. Interim Report

An interim evaluation report will be issued on approximately July 1. It will describe the preliminary results of the hardware performance phase of the evaluation. The results presented in this report will, of necessity, be subject to modification in the final report. This report is intended to provide an early summary of the CRAY-1 evaluation.

B. Final Report

A final evaluation report will be available on approximately October 1. This report will describe the objectives, methodology, findings, results and conclusions made as a result of this evaluation.

STATISTICS ADDENDUM

I. HYPOTHESIS

Let a code be partitioned into m modules. Let s_i be the CRAY-1 running time for module i, let t_i be the CDC 7600 running time for module i and let n_i be the number of times module i is executed in the complete code. It is the intent to test that the CDC 7600 run time for the entire code is greater than two times the CRAY-1 run time for the entire code, or

$$\sum_{M}(n_i t_i - 2n_i s_i) > 0$$

and hence that the CRAY-1 is at least two times faster than the CDC 7600 for the code.

II. SAMPLING

Sampling to test the hypothesis for the Class VI workload (section IV.A) will consist of the following procedure.

Assume that ℓ codes comprise the Class VI workload. ℓ modules will initially be drawn, one from each code. If more than ℓ modules are required initially, then 2ℓ modules, two from each code, will be drawn. The hypothesis will be tested at $k=i\cdot\ell$, for the i^{th} batch of modules.

The procedure for drawing a module from a Class VI workload code is specified in section IV.A., Step 3 of the Evaluation Plan.

Sampling to test the hypothesis for each Class VI applications workload code (section IV.B) will consist of the procedure outlined in section IV.B., Step 4 of the Evaluation Plan.

Samples drawn for applications performance testing will not be used for hardware performance testing, and samples drawn for hardware performance testing will not be used for applications performance testing.

The bucket size is initially 16 words. From preliminary studies this size has resulted in tractable module sizes. The bucket size will be changed if the resulting code segments are trivial or of intractable length.

III. TESTING

It is clear that we would like to test the hypothesis that the mean of the population of the $n_i t_i - 2n_i s_i$ for $i=1,2,\ldots,m$ is greater than zero. However, every statistical test of hypothesis on the mean of a population requires that an assumption be made about the distribution of the population. In the case of the $n_i t_i - 2n_i s_i$, we have no idea of the general shape of the distribution, let alone the type. Therefore the statistical testing will be done on the median of the distribution.

In testing the hypothesis that the median of the population of $n_it_i-2n_is_i$ is zero, we are in effect asking if 50% of the values of the $n_it_i-2n_is_i$ are greater than zero and 50% are less than zero. If we denote the true median by u_m, then we can formulate the above in the two statements

$$H_o\colon \mu_m \geq 0 \text{ vs. } H_a\colon \mu_m < 0 \text{ or}$$

$$H_o\colon \mu_m \leq 0 \text{ vs. } H_a\colon \mu_m > 0.$$

It is obvious that we could make either of two types of errors. We could accept H_o when H_a is true, or we could accept H_a when H_o is true. One usually can control only the probability of accepting H_a when H_o is true. The probability of making the other type of error can be as large as 1-Pr{accepting H_a when H_o is true}. Usual statistical tests of hypotheses require almost conclusive "proof" to accept H_a and require very little "proof" to accept H_o. In the case of testing the performance of the CRAY-1, the usual procedure of hypothesis tests will be unfair to one of the parties involved.

Therefore, the following set of hypotheses will be tested using sequential sampling [1].

H_1: At least 60% of the population of $n_it_i-2n_is_i$ is greater than zero. (The CRAY-1 is greater than two times faster than the CDC 7600.)

H_2: No more than 40% of the population of $n_it_i-2n_is_i$ is greater than zero. (The CRAY-1 is two or less times faster than the CDC 7600.)

The probability of accepting H_1 when H_2 is true will be fixed at .10, and the probability of accepting H_2 when H_1 is true will be fixed at .10. For tractability the maximum number of samples to be taken on any one code will be set at 50 from the Class VI applications workload and 50 total from the Class VI workload. The procedure is:

(1) Take the kth sample and determine s_i and t_i, the CRAY-1 run time and the CDC 7600 run time, and let

$Z_k=1$ if $n_it_i-2n_is_i>0$, or $t_i-2s_i>0$

$=0$ otherwise;

(2) Let $T_k=Z_k+T_{k-1}$, where $T_0=0$,

(3) If $T_k \geq U_k$ accept H_1,

If $T_k \leq L_k$ accept H_2,

If $L_k < T_k < U_k$ and $k<50$, increment k by 1 and go to Step 1.

The values of U_k and L_k are calculated from [1] and given by:

k	U_k	L_k	k	U_k	L_k	k	U_k	L_k
1	-	-	21	14	7	41	24	17
2	-	-	22	14	8	42	24	18
3	-	-	23	15	8	43	25	18
4	-	-	24	15	9	44	25	19
5	-	-	25	16	9	45	26	19
6	6	0	26	16	10	46	26	20
7	7	0	27	17	10	47	27	20
8	7	1	28	17	11	48	27	21
9	8	1	29	18	11	49	28	21
10	8	2	30	18	12	50	28	22
11	9	2	31	19	12			
12	9	3	32	19	13			
13	10	3	33	20	13			
14	10	4	34	20	14			
15	11	4	35	21	14			
16	11	5	36	21	15			
17	12	5	37	22	15			
18	12	6	38	22	16			
19	13	6	39	23	16			
20	13	7	40	23	17			

Therefore, after 50 samples, H_1 will be accepted, or H_2 will be accepted or no decision will be made. When no decision is made, it means that there is not enough information available to choose either H_1 or H_2, and probably indicates that the proportion of the $n_i t_i - 2n_i s_i$ which is greater than zero is between .4 and .6. In order for the CRAY-1 to meet the performance criteria specified in the Evaluation Plan, H_1 must be accepted.

The following simulation study of 1 000 tests of maximum sample size of 50 demonstrates this point. P is the true proportion of the population greater than zero, A.samp is the average sample size needed to make a decision, Pass is the number of samples in which H_1 (CRAY-1 passes) is accepted, Fail is the number of samples in which H_2 (CRAY-1 fails) is accepted, and Ques is the number of samples in which no decision is made after 50 samples.

P	A.Samp	Pass	Fail	Ques
.10	7.62	0	1000	0
.12	7.75	0	1000	0
.14	8.32	0	1000	0
.16	8.85	0	1000	0
.18	9.32	0	1000	0
.20	9.96	0	1000	0
.22	10.53	0	1000	0
.24	11.44	0	1000	0
.26	12.27	1	999	0
.28	13.50	3	996	1
.30	15.18	7	988	5
.32	16.11	8	976	16
.34	17.94	22	949	29
.36	19.15	23	945	32
.38	21.85	53	879	68
.40	23.71	89	814	97
.42	25.39	97	776	127
.44	27.72	171	659	170
.46	28.72	220	571	209
.48	28.64	305	482	213
.50	29.94	358	409	233
.52	29.37	481	293	226
.54	27.21	602	234	164
.56	26.81	692	153	155
.58	24.88	762	113	125
.60	23.57	830	71	99
.62	21.64	888	55	57
.64	19.59	937	25	38
.66	17.87	950	21	29
.68	15.97	981	11	8
.70	14.62	992	5	3
.72	13.26	996	3	1
.74	12.46	999	1	0
.76	11.59	998	1	1
.78	10.81	999	1	0
.80	10.02	999	1	0
.82	9.48	1000	0	0
.84	8.72	1000	0	0
.86	8.20	1000	0	0
.88	7.81	1000	0	0
.90	7.46	1000	0	0

Reference

[1] Wald, A., Sequential Analysis, John Wiley and Sons, 1947.

I/O STUDIES

Five tests have been devised to test the capability of the CRAY-1 I/O System.

DISK1. This test is designed to measure the instantaneous data rate for the DD-19 disk system. One track (8192 words) will be written to the disk and read back 1 000 times and timed. No arm movement or head switching will be required.

DISK2. This test is designed to determine the maximum data rate the DD-19 disk will sustain. A file of approximately two million words will be written sequentially on the disk then read back. The operation will be timed.

DISK3. This test is designed to measure arm positioning times. A large file will be written to the disk across many cylinder boundaries, then read back in one sector (512 word) blocks. Read requests will be arranged to force arm movement across a fixed number of cylinders in order to satisfy the request. The time required to position across 2^i cylinders, $i=0,\ldots,8$, will be recorded.

DISK4. This test is designed to gauge data integrity. A large file of approximately two million words of sequential integers will be written to the disk. The file will then be read back and verified.

DISK5. This test is designed to measure the impact of I/O upon the CPU and memory. Three timings will be made: 1) The time required to execute a program characterized by memory references. The program will consist of one million passes through a loop which adds two vectors of length 1 000 (A and B) and stores the result in C, of length 1 000. 2) The time to execute the above program when one disk is kept busy at its maximum sustained rate writing a 10 000 word block. Upon disk completion, the CPU will immediately issue another write request for the block. Disk positioning times will be minimized by issuing the write request at the next address from where the disk left off. Hence many copies of the blocks will be written into consecutive disk addresses. 3) If possible, the above test will be conducted with both disks active.

VECTORIZATION CRITERIA

(This appendix fulfills the requirements of section V.C. of the Evaluation Plan by establishing vectorization criteria to be used in the informal estimates of vectorizability.)

Mathematically, a vector is defined as an n-tuple of numbers that can be thought of as a function on the first n positive integers. Conceptually, components of a vector may have some natural ordering, for example, 3-dimensional Cartesian coordinates. On the other hand, the components of a vector often do not have any natural ordering, for example, each component may denote the velocity of a particle whose position is random in space. The important point is that when a scientist conceives of a vector, he is at liberty to specify its components and their ordering in whatever fashion he desires. Of course, convenience in algebraic manipulation may force him to order them in such a fashion that he can formulate the associated function on the first n positive integers.

Computer designers have incorporated an increasing amount of parallelism into general purpose computers in order to meet the constant demand for greater speed. The introduction of this parallelism has naturally led to the concept of vectors and vector operations in a computer with the following constraints.

1. Vectors must be stored such that the memory spacing between successive elements is constant.

2. Components of vector operations must be mutually independent, i.e., the i^{th} component of the result must be independent of all other components of the result.

Thus the concept of a vector in a computer is somewhat less general than that normally encountered in mathematics. In addition, the CRAY-1 computer only performs vector operations on the contents of vector registers, and these registers can contain at most 64 elements. These registers offer high arithmetic efficiency in situations where much arithmetic is performed on a few operand vectors. However, because of the fixed register size, a "pure vector" operation on the CRAY-1 will usually incorporate some small amount of scalar overhead associated with partitioning a vector into segments that are compatible with the length of vector registers.

These considerations lead to the following criteria for vectorization on the CRAY-1 computer.

1. The computation must be expressible as a repeated calculation. For example, a computation expressible as a Fortran DO loop may be vectorizable.

2. With few exceptions, elements of source operands must be independent of the elements of result operands. For example:

```
     DO 1 I = 1,N
   1 R(I) = R(I) + B(I) is vectorizable,

     DO 2 I = 1,N
   2 R(I) = R(I) + R(I-1) is not vectorizable.
```

3. Elements of operand vectors and result vectors must be stored with constant memory spacing, respectively. For example:

```
     DO 1 I=1,N,M
   1 V(I) = S1*U(I) + T(I) is vectorizable,

     DO 2 I=1,N
     M=J(I)
   2 R(M) = R(M) + A(I) is not vectorizable.
```

These three criteria will be applied to samples drawn from the Class VI applications workload codes in order to determine the proportion of the code that is immediately vectorizable as per section V.C of the Evaluation Plan. If the function performed by a sample can be re-written so as to satisfy the above criteria, that sample will be considered vectorizable.

DECISION RULE

In accordance with section IV.B of the Evaluation Plan the following decision rule was provided by LASL and approved by the Division of Military Application as the applications workload performance specifications.

> The applications performance specification is that by the techniques specified in the statistics addendum, the hypothesis H_1 that the CRAY-1 is greater than two times faster than the CDC 7600 must be accepted for one or more of the three Class VI applications workload codes.

Evaluation Plan Figures

Since Figures 1 thru 7 are presented in the body of the report, they are not repeated.

APPENDIX B

FIGURES

Figure B-1. Preliminary Scalar Test Routine Algorithms.

```
      subroutine tlu
      common/rmat/matid(12),at(12),rho(12),rhom(12),bep(12),psp(12),
     1polg(8,12),plg(16,16,12),eolg(8,12),elg(16,16,12),pelg(11,28,12)
      common/meos/arg1,arg2,func1,func2,maex,ifx,arg3(20),t6,t7
      dimension temp(40)
c
c
c
c                                     t(i)is t**4 coming in
c       ifx    func1    func2         ifx
c        1     p                       8  arg3(1)=t(1)     arg3(2)=t(2)
c        2     e                       8  arg3(3)=rho(1)   arg3(4)=rho(2)
c        3     op                      8  arg3(5)=rho(3)   arg3(6)=m(1)
c        4     p        dp/dr          8  arg3(7)=m(2)     arg3(8)=m(3)
c        5     dp/dt                   8  arg3(9)=lam(1)   arg3(10)=lam(2)
c        6     de/dt    de/dr          8  arg3(11)=lam(3)  arg3(12)=lam(4)
c        7     p        t
c        9     arg3(1)=e(1)      arg3(2)=e(2)      arg3(3)=e(3)
c        9     arg3(4)=m(1)      arg3(5)=m(2)      arg3(6)=m(3)
c        9     arg3(7)=mass(1)   arg3(8)=mass(2)   arg3(9)=mass(3)
c        9     arg3(10)=1/v
c        9     func1=p           func2=t
c        9     arg3(11)=dp/dt1   arg3(12)=dp/dt2   arg3(13)=dp/dt3
c        9     arg3(14)=dp/dr1   arg3(15)=dp/dr2   arg3(16)=dp/dr3
c        9     arg3(17)=di/dt1   arg3(18)=di/dt2   arg3(19)=di/dt3
c        9     temp(15)=di/dr1   temp(16)=di/dr2   temp(17)=di/dr3
c        9     temp(18)=p1       temp(19)=p2       temp(20)=p3
c        9     temp(21)=rho1     temp(22)=rho2     temp(23)=rho3
c        9     temp(24)=alpha    temp(25)=beta     temp(26)=gamma
c       10     temp(27)=t1       temp(28)=t2       temp(29)=t3
c
      ixf=ifx
      r=arg1
      t=arg2
      l=maex
      if(ixf.eq.7)er=arg2
      c=r/rho(l)
    1 cl=tlog(c)+9.
      cc=amax1(amin1(aint(cl),16.),2.)
      fc=cl-cc
      ic=cc
      goto(5,5,50,5,5,5,300,400,500,300,5,5,5)ixf
    5 if(t.gt..0009765625)goto20
      t4=ft=0.
      t3=1.
      t5=1024.*t
      goto(10,15,50,10,10,15,10,400,10,10,10,10,15)ixf
   10 t1=plg(ic-1,1,l)
      t2=plg(ic,1,l)-t1
      go to 45
   15 t1=elg(ic-1,1,l)
      t2=elg(ic,1,l)-t1
      go to 35
   20 tl=tlog(t)+12.
```

```
      tt=amin1(aint(tl),16.)
      ft=tl-tt
      it=tt
      goto(40,30,50,40,40,30,40,400,40,40,40,40,30)ixf
   30 t1=elg(ic-1,it-1,l)
      t2=elg(ic,it-1,l)-t1
      t3=elg(ic-1,it,l)-t1
      t4=t2-elg(ic,it,l)+elg(ic-1,it,l)
      t5=1.
   35 t6=t7=0.
      if(ic-9)100,36,37
   36 t6=(c-1.)**2*(psp(l)+bep(l)*(c-1.))
      t7=(c-1.)*(2.*psp(l)+3.*(c-1.)*bep(l))
      go to 100
   37 t6=texp(eolg(ic-9,l)+fc*(eolg(ic-8,l)-eolg(ic-9,l)))
      t7=t6*(eolg(ic-8,l)-eolg(ic-9,l))/c
      goto100
   40 t1=plg(ic-1,it-1,l)
      t2=plg(ic,it-1,l)-t1
      t3=plg(ic-1,it,l)-t1
      t4=t2-plg(ic,it,l)+plg(ic-1,it,l)
      t5=1.
   45 goto(48,48,48,48,100,100,48,400,48,48,48,100,100)ixf
   48 t6=t7=0
      if(ic-9)100,47,46
   46 t6=texp(polg(ic-9,l)+fc*(polg(ic-8,l)-polg(ic-9,l)))
      t7=t6*(polg(ic-8,l)-polg(ic-9,l))/c
      goto100
   47 t6=c**2*(c-1.)*(2.*psp(l)+3.*bep(l)*(c-1.))
      t7=psp(l)*(6.*c**2-4.*c)+bep(l)*(6.*c-18.*c**2+12.*c**3)
      goto100
   50 cl=.60206*tlog(c)+8.
      cc=amax1(amin1(aint(cl),11.),2.)
      fc=cl-cc
      ic=cc
      tl=3.*tlog(t)+14.
      tt=amax1(amin1(aint(tl),28.),2.)
      ft=tl-tt
      it=tt
      t1=pelg(ic-1,it-1,l)
      t2=pelg(ic,it-1,l)-t1
      t3=pelg(ic-1,it,l)-t1
      t4=t2-pelg(ic,it,l)+pelg(ic-1,it,l)
      t5=1.
  100 func1=texp(t1+fc*t2+ft*(t3-fc*t4))*t5
      return
  300 return
  400 return
  500 return
      end
      function texp(x)
      texp = exp(x*.69314718055995)
      return
      end
      function tlog(x)
      tlog=alog(x)*1.44269504
      return
      end
```

Figure B-2. CDC 7600 Driver Routine.

```
      PROGRAM DRIVER(INP,OUT)
C THIS IS THE CDC 7600 DRIVER PROGRAM FOR TIMING THE
C PRELIMINARY SCALAR TEST ROUTINE TLU. ALL ARGUMENTS
C TO TLU ARE PASSED IN COMMON.
C
      EXTERNAL TLU
      COMMON/RMAT/MATID(12),AT(12),RHO(12),RHOM(12),BEP(12),PSP(12),
     1POLG(8,12),PLG(16,16,12),EOLG(8,12),ELG(16,16,12),PELG(11,28,12)
      COMMON/MEOS/ARG1,ARG2,FUNC1,FUNC2,MAEX,IFX,ARG3(20),T6,T7
      DIMENSION T(28),C(16)
      ILOOP = 30
      JLOOP = 30
      NLOOP = 1
      FJLOOP = JLOOP
      FILOOP = ILOOP
           DO 10 I=1,16
           A1=I-11
           T(I)=2.**A1
                DO 10 J=1,16
                A1=J-8
                C(J)=2.**A1
                     DO 10 K=1,12
                     PLG(J,I,K)=ALOG(C(J)**2*T(I)**2)*1.44269504
                     ELG(J,I,K)=PLG(J,I,K)
   10                CONTINUE
           DO 20 I=1,28
           A1=I
           T(I)=2.**((A1-13.)/3.)
                DO 20 J=1,11
                A1=J
                C(J)=10.**((A1-7.)/2.)
                     DO 20 K=1,12
                     PELG(J,I,K)= ALOG(C(J)**2*T(I)**2)*1.44269504
   20                CONTINUE
           DO 25 I=1,8
                DO 25 K=1,12
                POLG(I,K)=EOLG(I,K)=ALOG(1.E-20)*1.44
                BEP(K)=PSP(K)=0
                RHO(K)=1.
   25           CONTINUE
  100 CONTINUE
      NTIME = 0
           DO 260 NN = 1,NLOOP
           DR=(C(16)-C(1))/FJLOOP
           DT=(T(16)-T(1))/FILOOP
                     DO 200 J=1,JLOOP
                     ARG1=C(1)+DR*(J-1)
                          DO 200 I=1,ILOOP
                          ARG2=T(1)+DT*(I-1)
                          DO 200 L = 1,6
```

```
                              IFX=MAEX=L
                              Y1=ARG1**2
                              Y2=ARG2**2
                              Y3=Y1*Y2
                              IF(IFX.EQ.7)ARG2=Y3+.013718*Y2**2/ARG1
C
C                             CALL TLU FROM TIMING FUNCTION TFN
C
                              CALL TFN (NCTFN,TLU,FUNC1)
                              NTIME = NTIME + NCTFN
  200                         CONTINUE
  260                    CONTINUE
      TIME = NTIME*.0275E-06
      PRINT 2002, NTIME, TIME
2002 FORMAT(I10,6HCYCLES,F15.10,7HSECONDS)
      STOP
      END
```

Figure B-3. Listings of Kernel E1.

CRAY-1 Assembly Language.

```
                                           BASE      0
                                           IDENT     E1
                                  *
                                  *
                                  * FORTRAN MODULE
                                  *     AR=1.0/AR
                                  *     UEFF=SUMU*AR
                                  *     VEFF=SUMV*AR
                                  *     XN=XP(IP)+UEFF*DTOS
                                  *     YN=XP(IP+1)+VEFF*DTOSS
                                  *     AR=ABS(XN-INT(XN))
                                  * SELECTION ASSUMPTION - STATEMENT FOLLOWING TEST NOT TAKEN
                                  *     IF (AR.LT.EM6.OR.AR.GT.0.99999) XN=XN+EM4
                                  *
                                  * EXECUTION ASSUMPTIONS
                                  *     NONE
                                  *
                                  *
                                           ENTRY     E1
                                           ABS
                                           ORG       200
200a 072100                       E1       S1        RT
   b 075177                                T77       S1
   c 120100000220                          S1        AR,A0          S1=AR
201a 120200000222                          S2        SUMU,A0        S2=SUMU
   c 120300000223                          S3        SUMV,A0        S3=SUMV
202a 122400000224                          S4        DTOS,A0        S4=DTOS
   c 120500000225                          S5        DTOSS,A0       S5=DTOSS
203a 070710                                S7        /HS1           S7=31 BIT APPROX 1./AR
   b 100100000226                          A1        IP,A0          A1=IP
   d 064442                                S4        S4*FS2         S4=SUMU*DTOS
204a 064553                                S5        S5*FS3         S5=SUMV*DTOSS
   b 121600000233                          S6        XP-1,A1        S6=XP(IP)
   d 067117                                S1        S1*IS7         S1=2ND STEP DIVISION DENOMINATOR
205a 064227                                S2        S2*FS7
   b 064337                                S3        S3*FS7
   c 064447                                S4        S4*FS7
   d 064557                                S5        S5*FS7
206a 064221                                S2        S2*FS1         S2=UEFF
   b 064331                                S3        S3*FS1         S3=VEFF
   c 064441                                S4        S4*FS1         S4=UEFF*DTOS
   d 064551                                S5        S5*FS1         S5=VEFF*DTOSS
207a 121100000234                          S1        XP.A1          S1=XP(IP+1)
   c 130200000227                          UEFF.A0   S2             STORE UEFF
```

```
210a 130300000230              VEFF.A0   S3                    STORE VEFF
   c 062664                    S6        S6+FS4                S6=XN
   d 044200040060              S2        040060
211b 042220                    S2        <60                   S2=040060.....0
   c 062115                    S1        S1+FS5                S1=YN
   d 062262                    S2        S6+FS2                S2=AINT(XN)
212a 130600000231              XN,A0     S6                    STORE XN
   c 120500000254              S5        EM6.A0                S5=EM6
213a 120400000221              S4        E99.A0                S4=.99999
   c 063262                    S2        S6-FS2                S2=XN-INT(XN)=FRACTIONAL PART OF XN
   d 130100000232              YN,A0     S1                    STORE YN
214b 045220                    S2        #S0&S2                S2=ABS(XN-INT(XN))=AR
   c 063525                    S5        S2-FS5                S5=AR-EM6
   d 063442                    S4        S4-FS2                S4=.99999-AR
215a 130200000220               AR,A0    S2                    STORE AR
   c 051054                    S0        S5!S4
   d 017 000002178             JSM       LAB1
216b 072600                    S6        RT
   c 074777                    S7        T77
   d 061567                    S5        S6-S7
217a 006 000002178        LAB1 J         *
   c 043000                    S0        0
   d 004000                    EX
220  0400024000000000000000  AR      CON       0400024000000000000000
221  0400074000000000000000  E99     CON       0400074000000000000000
222  0400014000300000000000  SUMJ    CON       0400014000300000000000
223  0400014000300000000000  SUMV    CON       0400014000300000000000
224  0400014000000000000000  DTOS    CON       0400014000000000000000
225  0400014000000000000000  DTOSS   CON       0400014000000000000000
226  0000000000000000000001  IP      CON       1
227  0000000000000000000000  UEFF    CON       0
230  0000000000000000000000  VEFF    CON       0
231  0000000000000000000000  XN      CON       0
232  0000000000000000000000  YN      CON       0
233  0000000000000000000000  ARR     CON       0
234                          XP      BSSZ      20
254  1400007777700000000000  EM6     CON       1400007777700000000000
                                     END
```

CDC 7600 Assembly Language.

```
                                                       IDENT  E1
                                          *
                                          *
                                          * FORTRAN MODULE
                                          *      AR=1.0/AR
                                          *      UEFF=SUMU*AR
                                          *      VEFF=SUMV*AR
                                          *      XN=XP(IP)+UEFF*DTOS
                                          *      YN=XP(IP+1)+VEFF*DTOSS
                                          *      AR=ABS(XN-INT(XN))
                                          * SELECTION ASSUMPTION - STATEMENT FOLLOWING TEST NOT TAKEN
                                          *      IF (AR.LT.EM6.OR.AR.GT.0.99999) XN=XN+EM4
                                          *
                                          * EXECUTION ASSUMPTIONS
                                          *      NO LCM VARIABLES
                                          *      VALUE FOR VARIABLE AR STORED INTO DUMMY VARIABLE ARR TO
                                          *      EXPEDITE LOOPING AND MULTIPLE CALLS (AVOIDING DIVIDE BY ZERO IN
                                          *      FIRST FORTRAN STATEMENT)
                                          *      STORAGE INTO ARR HAS NO IMPACT UPON TIMING.
                                          *
                                          *
                                                       ENTRY  E1
                                                       USE    //
 0                                        ITIME        BSS    1
                                                       USE    *
 0  00000000000000000000                  E1           DATA   0
 1  01550                                 CLOCK        TB5
          5110000024 +                                 SA1    AR          X1=AR
 2  7120017204                                         SX2    172048      X2=17204
          5130000064 +                                 SA3    IP          X3=IP
 3  5140000026 +                                       SA4    SUMU        X4=SUMU
          20255                                        LX2    45          X2=1.
 4  5150000031 +                                       SA5    SUMV        X5=SUMV
          44721                                        FX7    X2/X1       X7=1.0/AR
 5  5110000065 +                                       SA1    DTOS        X1=DTOS
          5120000067 +                                 SA2    DTOSS       X2=DTOSS
 6  5233000032 +                                       SA3    XP-1+X3     X3=XP(IP)
          40647                                        FX6    X4*X7       X6=SUMU*AR=UEFF
                40757                                  FX7    X5*X7       X7=SUMV*AR=VEFF
 7  5140000070 +                                       SA4    EM6         X4=EM6
          5053000001                                   SA5    A3+1        X5=XP(IP+1)
10  40161                                              FX1    X6*X1       X1=UEFF*DTOS
       40272                                           FX2    X7*X2       X2=VEFF*DTOSS
          5160000027 +                                 SA6    UEFF        STORE TO UEFF
11  5170000030 +                                       SA7    VEFF        STORE TO VEFF
          30631                                        FX6    X3+X1       X6=XN
                30752                                  FX7    X5+X2       X7=YN
12  5110000023 +                                       SA1    C99         X1=.99999
          43201                                        MX2    1
                24606                                  NX6    X6          X6=XN
13  20273                                              LX2    59          X2=2000...08
       24707                                           NX7    X7          X7=YN
          32226                                        DX2    X2+X6       X2=FRACTIONAL PART XN
14  5160000032 +                                       SA6    XN          STORE TO XN
          21673                                        AX6    59          X6=SIGN EXTENSION OF XN
                24202                                  NX2    X2          X2=FRACTIONAL PART XN
```

```
15  5170000060 +                      SA7    YN            STORE TO YN
              13662                   BX6    X6-X2         X6=ABS( )=AR
                    31264             FX2    X6-X4         X2=AR-EM6
16  31316                             FX3    X1-X6         X3=.9999-AR
          5160000025 +                SA6    ARR           STORE TO ARR
                    12223             BX2    X2+X3         X2=AR-EM6 .OR. 9999-AR
17  0332000021 +                      NG     X2,LXXX       TEST RIGGED TO DROP THROUGH
                                * END TIMING SECTION
              01660                   TB0
                    77665             SX6    B6-B5
20  0336000001 +                      NG     X6,CLOCK
              5160000000 C            SA6    ITIME
21  0400000000 +              LXXX    EQ     E1
22  5160000024 +                      SA6    AR            STORE TO AR
23  17177777753016451671      C99     DATA   .99999
24  17214000000000000000      AR      DATA   2.
                                * ARR DUMMY VARIABLE AS STATED IN ASSUMPTIONS.
25  17214000000000000000      ARR     DATA   2.
26  17204000000000000000      SUMU    DATA   1.
27  17204000000000000000      UEFF    DATA   1.
30  00000000000000000000      VEFF    DATA   0.
31  17214000000000000000      SUMV    DATA   2.
32  00000000000000000000      XN      DATA   0.
33                            XP      BSSZ   25
64  00000000000000000001      IP      DATA   1
65  17216000000000000000      DTUS    DATA   3.
66  00000000000000000000      YN      DATA   0.
67  17224000000000000000      DTUSS   DATA   4.
70  00000000000000000000      EM6     DATA   0.
71                                    END
```

Figure B-4. Vector Test Routine Algorithms.

```
      SUBROUTINE VVSPVS(R,A,S1,B,S2,N)
      DIMENSION A(1),B(1),R(1)
C
C     CALCULATE R=A*S1+B*S2 WHERE A,B ARE
C     VECTORS AND S1,S2 ARE SCALARS.
C     INPUT
C     N=VECTOR LENGTH
C     A, B AND R ARE VECTORS OF LENGTH N
C
      DO 10 I=1,N
      R(I)=A(I)*S1+B(I)*S2
   10 CONTINUE
      RETURN
      END
      SUBROUTINE VVVPVV(R,A,B,C,D,N)
      DIMENSION A(1),B(1),C(1),D(1),R(1)
C
C     CALCULATE R=A*B+C*D WHERE A,B,C AND D ARE VECTORS
C     INPUT
C     N=VECTOR LENGTH
C     A,B,C,D AND R ARE VECTORS OF LENGTH N
C
      DO 10 I=1,N
      R(I)=A(I)*B(I)+C(I)*D(I)
   10 CONTINUE
      RETURN
      END
      SUBROUTINE VVVPV(R,A,B,C,N)
      DIMENSION A(1),B(1),C(1),R(1)
C
C     CALCULATE R=A*B+C WHERE A,B AND C ARE VECTORS
C     INPUT
C     N=VECTOR LENGTH
C     A,B,C AND R ARE VECTORS OF LENGTH N
C
      DO 10 I=1,N
      R(I)=A(I)*B(I)+C(I)
   10 CONTINUE
      RETURN
      END
      SUBROUTINE DOTPRO(N,X,IX,Y,IY,R)
      DIMENSION X(1),Y(1)
C
C     CALCULATE THE SUMMATION OF THE PRODUCT OF X*Y
C     WHERE X AND Y ARE VECTORS.
C     N IS THE NUMBER OF VALUES TO BE SUMMED; IX AND IY
C     ARE THE SPACING IN THE X AND Y VECTORS, RESPECTIVELY;
C     AND R IS THE RESULT.
C
      IXX=1
      IYY=1
      R=0
      DO 10 I=1,N
      R=R+X(IXX)*Y(IYY)
      IXX=IXX+IX
   10 IYY=IYY+IY
      RETURN
      END
      SUBROUTINE ADDVEC(A,B,C,S,N)
      DIMENSION A(1),B(1),C(1)
C
C     CALCULATE THE EXPRESSION A=S*B+C
C     WHERE S IS A SCALAR VALUE AND B,C,A ARE
C     VECTORS. N IS THE NUMBER OF VALUES TO
C     BE CALCULATED.
C
      DO 10 I=1,N
   10 A(I)=S*B(I)+C(I)
      RETURN
      END
```