SOFTWARE METRICS FOR PARALLEL PROGRAMS

By

IMTIAZ AHMAD

Bachelor of Science

University of the Punjab

Lahore, Pakistan

1981

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

SOFTWARE METRICS FOR PARALLEL PROGRAMS

Thesis Approved:

M. Samadzadeh
Thesis Adviser

J. Chandler

D. E. Hensley

Norman N. Durham
Dean of the Graduate College

ACKNOWLEDGEMENTS

First I would like to thank my thesis advisor, Dr. Mansur H. Samadzadeh for his continuous guidance, dedication, and valuable instruction throughout my research work. Without his encouragement and motivation, completion of this thesis would not have been possible.

Special thanks are due to Dr. Keith A. Teague for providing access to the hardware necessary for this work, and to Dr. Gary B. Lamont for providing his compendium of parallel programs used as a testbed in this thesis. I also wish to thank Drs. P. Larry Claypool and William L. Woodall for helping me in the selection of statistical tests.

I would also like to thank Drs. John P. Chandler and George E. Hedrick for their suggestions and advice while serving on my thesis committee. In addition, I would like to thank my supervisor at the University Computer Center, Larry P. Watkins, for allowing me to follow a flexible work schedule during my thesis research.

Finally, I wish to thank my family. It was their continuous support that gave me the motivation and inspiration to complete my graduate studies.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

We have all encountered various types of failures or collapses. For instance, generally speaking, a building can collapse because of engineering defects and an automobile can collapse because of engine failure. In computer science, software fails not just because of hardware failure, but most of the time because of the lack of due consideration to the field of software engineering while developing that particular software.

The field of software engineering has existed for the past three decades [Goldberg86] and has been defined by Boehm [Boehm81] as,

> ... the application of science and mathematics by which the capabilities of computer equipment are made useful to man via computer programs, procedures, and associated documentation.

Conte *et al.* [Conte86] summarized the goal of software engineering as: "... to produce higher quality software at lower cost."

Software engineering has several fields. One of these fields is the study of static measurements of programs as indicators of repairability, clarity, complexity, reliability, number of faults, productivity, quality, *etc.* These measurements are expressed by using the concept of metrics. In general, metrics can be applied to many different levels of a computer system in both software and hardware areas.

Generally speaking, software complexity can be perceived as clarity, understandability, or ease of modifying and debugging programs. Software complexity metrics thus attempt to objectively measure the difficulty involved in

developing and maintaining programs. Several metrics have been proposed and evaluated for sequential programs. Parallel software, despite its rapid growth, currently lacks software complexity metrics.

One way to begin to address the question of the complexity of parallel programs is by exploring the parallel aspects of the software complexity issue. The purpose of this thesis is to apply the existing software metrics to parallel programs and construct new metrics specifically suited for parallel programs.

Several parallel machines are available commercially with different types of architectures. Each machine has a different operating system and hence different applications software. The programs that were analyzed as part of this thesis were made available, upon a request by the author, by Dr. G. B. Lamont of Air Force Institute of Technology (AFIT), Dayton, Ohio. These programs had been written specifically for the Intel's iPSC family of supercomputers [Intel88]. TABLE I gives a general overview of the parallel programs in the compendium in terms of their size, hardware used, Lines of Code, and McCabe's V(G) (discussed in Section 3.1.3). Intel's iPSC was chosen for this study because of its existence on Oklahoma State University (OSU) campus and availability to graduate students.

Chapter 2 of this thesis gives a brief review of the history of parallelism and software metrics used in this study. Chapter 3 includes a discussion of the metrics considered for this study. Chapter 4 discusses the experimental methodology utilized. Analysis of the measurements is included in Chapter 5. Chapter 6 summarizes the main conclusion and elucidate some possible areas of future work.

TABLE I

A GENERAL CATEGORIZATION OF THE PARALLEL PROGRAMS USED IN THE STUDY

Application Type	Hardware 1=iPSC/1 2=iPSC/2	Number of Programs	LOC* Range	McCabe's V(G)
Ring Simulation	1 & 2	2	225 - 275	10 -12
Mesh Network Simulation	1	4	150 - 300	10 - 30
Sorts (Bitonic Odd-Even,Radix)	1 & 2	7	600 - 1200	60 - 110
Heap Sort	1 & 2	6	225 - 500	20 - 60
Neural Network	1	2	500 - 1000	35 - 130
Travelling Salesman	1	2	1150 - 1250	130 -135
Set Covering Problem	2	3	4000 - 14000	250-1200
Partial Differential	1	1	700	70
Graph Search	1	1	800	50
Assignment Problem**	1	1	1300	160
Matrix Multiplication & Inversion	1	1	525	40
Dining Philosophers	1	2	600 - 725	30 - 60

* Here LOC represents the total length of a parallel program in terms of lines of code including blank and comment lines

** A parallel implementation of the Assignment Problem using the Hungarian Method [Compendium90]

CHAPTER II

PARALLELISM: HISTORY AND HARDWARE

Concurrent or parallel architectures are not new. Even John von Neumann, whose ideas lead to the development of the sequential architecture that is used in most computers today, preferred the parallel approach [Rattner85]. However, the technological barriers of the time, such as unreliability of vacuum tubes, distanced the idea from its practical implementation.

In late 1960s several parallel machines were introduced, including the 64-processor ILLIAC IV [Hayes88] at the University of Illinois. The ILLIAC IV's limited memory and expensive hardware kept it away from commercial use. The evolution of the cost-effective Very Large Scale Integrated (VLSI) technology in 1970s [Hayes88] stimulated interest in developing parallel computers for commercial use. In 1980s several vendors introduced parallel machines with different architectures and topologies. Each of them has some advantages and disadvantages over others. Some of the better known parallel computers are: Butterfly GP1000 of BBN Advanced Computers Incorporation with local or shared memory and switch interconnection topology [BBN89], Multimax 520 of ENCORE Computers with shared memory and bus topology [Encore89], and iPSC/2 of Intel Scientific Computers with local (distributed) memory and hypercube interconnection topology [iPSC88 and Intel88]. Programs chosen to study in this thesis were exclusively written for Intel's iPSC family of computers.

2.1 Definition of Parallel Processing

In most of the available literature, the terms concurrent and parallel have generally been used interchangeably. Fox [Fox88] defined concurrent processing as:

> ... the use of several working entities (either identical or heterogeneous), working together toward a common goal.

In concurrent computation, Fox considers the working entities as computers and the goal as a large computation problem, such as weather prediction.

2.2 Types of Parallelism

Stone [Stone87] discusses two types of parallelism: coarse-grain and fine-grain. He states that in general the performance benefits of a multiprocessor machine strongly depend on the ratio R/C, where R is the length of a run-time quantum and C is the length of the communication overhead incurred during that quantum. Stone uses the ratio R/C as a measure of task granularity and states that in coarse-grain parallelism R/C is relatively high, such that each run-time quantum generates a relatively small amount of communication overhead. On the other hand, in fine-grain parallelism, R/C is very low, hence it causes a relatively large amount of communication overhead during each run-time quantum.

In this thesis we are not putting any emphasis on the dynamic aspects of application programs. Instead we are taking the source code that is running correctly, and analyzing it in a number of different ways that are described later.

2.3 Intel's iPSC Concurrent Supercomputers

The first version of Intel's iPSC Concurrent Supercomputer was introduced in mid 1980s and was named iPSC. This first version (later known as iPSC/1) was

based on the hypercube architecture [Seitz85] and had Intel's 80286 microprocessor with the XENIX[1] operating system. In the late 1980s, Intel introduced its new version and named it iPSC/2. The new version has a 80386 microprocessor and a number of added features over iPSC/1, such as SX scalar processor, UNIX V.3 operating system [UNIX86], and the Direct-Connect Module (DCM) [Nugent88]. DCM is a specialized hardware that controls the message passing system and is attached to each hypercube node.

The iPSC/2 system consists of compute nodes, I/O nodes, and a front-end processor called host. Each node is a processor-memory pair, with distinct memory from host and other nodes. Each node runs its own copy of NX/2 [Pierce88] operating system. The NX/2 operating system is written almost entirely in C and can manage up to 20 processes per node. It also manages the numeric coprocessors for each process on a node. The front-end processor is called the System Resource Manager (SRM) or the local host. SRM runs the UNIX operating system. The host program executes in the UNIX environment and provides the user interface and, if needed, loads the node program to each node. It also provides true 32-bit node architecture performance [Close88].

In both iPSC/1 and iPSC/2, concurrency is achieved by grouping loosely-coupled independent processing elements executing portions of a larger computational problem simultaneously. All parallel application programs run over an iPSC consist of at least two modules. One module runs on the front-end SRM and the other runs on each participating node. In general, a module running on a local host is known as a host module and a module running on nodes is known as a node module.

The following is an example of solving a sequential problem with a parallel algorithm on an iPSC machine. Example 1 shows how developing a parallel

1 XENIX is a registered trademark of Microsoft Corporation.

algorithm for a problem is in general more complex in terms of logic design and program writing than developing a sequential algorithm for the same problem. The program for Example 1 appears in Appendix A.

Example 1: Given a parallel machine such as iPSC/2, we need to sum the numbers from 1 to 100 on its p processors. Suppose p=1, that is, we have a one-processor machine to sum the integers from 1 to 100. Assuming that adding two integers takes one time unit, to sum the integers from 1 to 100 would take 99 time units. On a uniprocessor platform, a parallel algorithm to sum the integers from 1 to 100 would take almost the same number of time units as a sequential algorithm.

Now suppose we have two processors (p=2) to sum the integers from 1 to 100. We can assign half of all the integers involved in the summation to each processor, that is, in the case of the integers from 1 to 100, each processor will get 50 integers to sum. In this way, generally speaking, integers from 1 to 100 can be summed in approximately half of the time that it would take to sum them on one processor. Hence, in terms of processing time or speed-up, we can achieve almost a 100% gain. But to achieve this gain we have to change our parallel algorithm such that the range can be divided by the number of processors available. This change in the algorithm may increase its complexity.

Now consider p=3. Obviously in this case we cannot divide 100 integers (from 1 to 100) into 3 equal ranges. We have to change our parallel summation algorithm again to get the optimal processing time. In the case of three processors, our algorithm should be able to assign 33 integers to two of the three processors, and 34 integers to the third processor. Developing an algorithm that could handle this uneven assignment of the ranges to processors may take some extra effort and may also make the programming task more difficult and complex in terms of the lines of code and time spent in writing it.

From the above example, the following observation can be made: the static complexity of software involved in parallel processing probably has more dimensions or aspects than the static complexity of software on conventional (sequential) computers, even though the basic issues (i.e., understandability, quality, maintainability, *etc.*) are the same.

CHAPTER III

SOFTWARE METRICS

Because of the intuitive relationship between conceptual complexity and software quality, several studies have focused attention on the development and validation of a set of quantitative metrics to measure the complexity of software. Intuitively speaking, parallel software is more difficult to understand than sequential software. This is in general true because of the differences in the programming languages, programming environments, and specially the architectures of sequential and parallel machines, and the difficulty of visualizing parallel execution. Also, since sequential and parallel programs can be considered different as far as understandability is concerned, it is not advisable to use metrics developed for sequential programs on parallel programs without first validating them for suitability. This spawned the need to develop parallel software metrics.

3.1 Types of Software Metrics

Several metrics have been developed and studied for measuring the complexity of sequential software as well as hardware for different machines and languages. Some of the metrics developed for software are cyclomatic complexity [McCabe76], software science metrics [Halstead77], and information flow metrics [Henry79]. However, none of the software metrics or analyses have been carried out for the C programming language on iPSC machines. The performance evaluation of parallel systems from the hardware aspect has also been the subject of

several studies [Zuberek85]. Work done by Haban and Wybranietz [Haban89] on monitoring and measuring parallel systems can be noted here.

Conte *et al.* [Conte86] discussed several types of metrics, such as Size, Data structure, Logic structure, Software Science, Effort, and Cost. They indicated that program complexity increases with size and that large programs are generally more difficult to understand and write, contain relatively more errors, and are more difficult to debug. To reduce this complexity, software designers have increasingly turned to program modularization and structured design methodologies. The advantages of program modularization are typically expressed in terms of comprehensibility, manageability, efficiency, error reduction, and reduced maintenance effort. Conte *et al.* state that not all computer scientists agree on these advantages and some consider program modularization a disadvantage because the need for proper interfacing among the modules increases as the number of modules grows.

3.1.1 Size Metrics

The size of a program is a well-known and widely-accepted measure and is still considered a basic measure for some models of software development, and cost and schedule estimation. The size metric can be calculated in several ways. One way is by counting the number of lines of code and another is by counting the tokens.

Size metric measured in terms of lines of code may not be satisfactory for modern programming languages because not all lines in a program may have the same level of difficulty in their production. Some of the lines in a program may have fewer tokens and hence be in general less difficult to produce than other lines in the same program. However, lines of code (LOC) is still the most widely-used

size metric. Lines of code is defined by Conte *et al.*, as the sum of all non-commented and non-blank lines. This definition is used in this thesis for size metric.

3.1.2 Token Count Metrics

Halstead [Halstead77] viewed a program as a sequence of tokens, which could be either operands representing data or operators manipulating the operands. Halstead's four basic counts are as follows.

n_1 : Number of unique operators

n_2 : Number of unique operands

N_1 : Total occurrences of operators

N_2 : Total occurrences of operands

There is no general agreement among researchers regarding exactly which tokens in a given language are operators and which are operands. This makes a general consensus regarding token counting hard to reach. Conte *et al.* suggested that the classification of a token as operators or operands should be determined by the programmer who is developing the counting tool.

Halstead defined various metrics based on these four basic counts, some of them are listed below.

The vocabulary n is defined as $n = n_1 + n_2$

The program length N is given by $N = N_1 + N_2$

The estimated program length is defined as

$$N_{est} = n_1 * \log(n_1) + n_2 * \log(n_2)$$

(All logarithms are base 2 unless explicitly stated otherwise.)

3.1.3 McCabe's Cyclomatic Complexity Metric

McCabe [McCabe76] suggested a metric to measure the maintenance difficulty of a program based on the number of different independent paths through it. These independent paths through a program add to the complexity of testing a program, as experienced by programmers. He suggested a control flow metric, based on the number of conditions (such as the "if" statements in a program) which he called the cyclomatic complexity. The cyclomatic complexity is defined as

$$V(G) = e - n + 2p$$

where e is the number of edges, n is the number of nodes, and p is the number of connected components in the control flow graph of a program. An alternative formulation of the cyclomatic complexity is

$$V(G) = Pr + 1$$

where Pr is the number of predicates in the program. V(G) can be easily calculated using this alternate form.

3.1.4 Residual Complexity Metrics

Samadzadeh and Edwards [Samadzadeh88] proposed a metric called the residual complexity which measures the remaining complexity in a software document after some attempt has been made to understand it by conceptually subdividing or chunking it. They argue that a software document can be thought of as a set of tokens of different types. In an abstract view of the classification part of the comprehension process, a user trying to understand a software document examines individual tokens and finds the class to which each token belongs. Each classification represents a level of understanding and refinement of a classification or partition signifies an improvement in understanding. After all the tokens have

been classified at a certain level of comprehension, the as yet uncovered portion of the software complexity can be represented as

$$R = N_1 * \log(N_1) + N_2 * \log(N_2) + ... + N_q * \log(N_q)$$

where R is the residual complexity metric and N_i, i = 1, 2, ..., q, is the number of tokens in the ith class or block of the current partition.

3.1.5 Proposed Metrics

In an attempt to modify the cyclomatic complexity for a parallel algorithm, we can consider all message passing/receiving commands as virtual conditional statements. This assumption can be intuitively supported by the argument that for any message passing/receiving command the program control jumps to another location, thus increasing the difficulty of comprehending the program.

The following are some of the metrics proposed specifically for parallel programs.

3.1.5.1 Message Send Metrics

Three types of "Send" metrics can be identified:

i. Host Send metric;

ii. Node Send metric; and

iii. Total Send metric.

Host Send metric (H_S) is the sum of all message send commands appearing in the host program of an application, which may or may not have a corresponding message receive command in the same application. Node Send metric (N_S) is the sum of all message send commands appearing in a node program of an application, which may or may not have a corresponding message receive command in the same

application. Finally, Total Send metric (T_S) is the sum of all message send commands in an application, $T_S = H_S + N_S$.

3.1.5.2 Message Receive Metrics

The definitions of Message Receive Metrics are analogous to those of the Message Send Metrics. Three types of "Receive" metrics can be identified:

i. Host Receive metric;

ii. Node Receive metric; and

iii. Total Receive metric.

Host Receive metric (H_r) is the sum of all message receive commands appearing in the host program of an application, which may or may not have a corresponding message send command in the same application. The node Receive metric (N_r) is the sum of all message receive commands appearing in a node program of an application, which may or may not have a corresponding message send command in the same application. Total Receive Metric (T_r) is the sum of all message receive commands in an application, $T_r = H_r + N_r$.

CHAPTER IV

EXPERIMENTAL PROCEDURE

This chapter describes the experimental methodology used in the design of this study including the framework adopted for experimentation, data collection methodology, and the static metrics gathered and their analyses. Models derived based on both the experts' perceptions and the static metrics are also discussed.

The framework of experimentation defined by Basili *et al.* [Basili86] was used in the design of this study. This choice was made because of the wide acceptance of their framework in the field of software engineering and related research areas. According to Basili *et al.'s* framework, there are four stages in the experimentation process: 1) definition, 2) planning, 3) operation, and 4) interpretation. The definition, planning, and operation stages are described below. The fourth category, interpretation, will be discussed in the following chapter.

4.1 Experiment Definition

This study was devised to understand (motivation) the parallel aspects of software complexity. The purpose of the study was to conduct an exploratory empirical study of academic programs (domain) written in iPSC/2-C [Green89] on Intel's iPSC family of concurrent supercomputers. Initially, over 35 programs written by eight graduate students of a particular graduate level class were evaluated, both by using metrics and from the perspective of a number of experts in parallel programming. Nineteen parallel programs written by eight graduate students were chosen for final evaluation. The reasons for the exclusion of some

programs and the criteria for the inclusion of the programs selected for final analysis is discussed in Section 4.3.1. There were four subjects who ranked the programs based on the questions asked, according to their best judgement. The subjects had considerable experience in the field of parallel programming specially on the iPSC family of concurrent computers.

4.2 Experiment Planning

Only the syntactically correct and properly running programs were included in the study. An objective as well as subjective assessment of static measurements in a multivariate design was proposed. The programs were to be evaluated based on the metrics described in Chapter 3. A non-parametric test was chosen for correlation analysis in consultation with Dr. P. Larry Claypool, Professor of Statistics, at Oklahoma State University, and Dr. William L. Woodall, Professor of Statistics, at the University of Alabama. This choice was made because limited data were available and distributional assumptions could not be met. The data consisted of both objective as well as subjective measurements. The correlation analysis was used to study the possible relationships between static metrics and the experts' ratings of the complexity of programs.

4.3 Experiment Operation

The next two subsections explain the design of a questionnaire to glean and compile the experts' judgements and the software tools that were developed and/or used to collect the metrics. Some of the problems encountered are also described briefly.

4.3.1 Design of the Questionnaire

A questionnaire (Appendix B.1) was devised to capture the experts' subjective perception of the relative complexity of the 37 programs included in the study. In the questionnaire, each application was mentioned with its complete directory path in the compendium. The questionnaire had 10 questions. The first four questions were included to judge the participants' expertise and experience level. This was necessary to make sure that the participants had enough experience in the field of parallel processing to judge parallel programs. Questions 5 through 10 were designed to elicit the judgement of the participants regarding the programs used in the study. In fact Questions 1 through 4 serve as a pretest and Questions 5 through 10 serve as a posttest [Conte86]. Some of the questions asked in the questionnaire were redundant. This was done intentionally to compare the consistency in the participants' replies. For instance, question 9 asks to rate the overall complexity of an application whereas in questions 5 and 6 the understandability of the host and the node programs were requested to be rated. Intuitively speaking, the replies should be in the form of opposite ranking, e.g., if three applications A, B, and C are ranked as 1, 2, and 3 by questions 5 and 6. then the same applications should be ranked as 3, 2, and 1 by question 9. The above expectation was met when correlation analysis was done on the experts' replies (see Section 5.3, TABLE IXa).

It was expected that not all the participants in the study would be familiar with each of the application included in the study. Hence, in the questionnaire, the participants were requested to record their judgement only for those applications with which they were familiar. This instruction was to give the participants a feeling that they were not obliged to rate each application for every question. As a general

rule, this also helps restrict the outliers. Thus the ratings across the application names that were left blank, were assumed to be unanswered.

Participation in the study was voluntary. Out of the original 10 experts targeted in this study, four replied. All four participants had adequate experience and education. Their education and experience in the field of parallel processing is given in TABLE II. Among the 37 application originally included in the study, 19 were judged by all the participants. One application, i.e., "*project/beard/src/thesis/parallel*", was found to be an outlier even though it was rated by all the participants. This particular application has approximately 14,000 lines of code. Although the Spearman Rank Correlation Coefficient test (discussed in Section 5.1) takes care of bad outliers, it was dropped from the study because the above application was affecting the mean values drastically.

TABLE II

EXPERTS' EDUCATION AND EXPERIENCE LEVEL

Number of Participants	Highest Academic Degree	Parallel Processing Experience
1	Ph. D.	5 Years
2	M.S.	2 Years
1	B.S.	2 Years

4.3.2 Software Used to Gather the Data

Among the three pre-written software packages [Bishop87, Graham83, and PCMETRIC90] initially thought to be useful in collecting various static measurements, the commercially available tool PC-METRIC was chosen to collect

some of the data because of its availability on campus. The rest of the data collection was achieved through the programs developed on the iPSC/2 concurrent supercomputer.

PC-METRIC is a microcomputer-based software tool that runs under the Disk Operating System (DOS) [DOS87]. It expects as input any syntactically correct and compilable C program and generates a report. The report contains a set of two complexity metrics: the Software Science family of metrics and the cyclomatic complexity metrics.

An advantage of using PC-METRIC was that it considers a C source code file as a series of tokens. All of the reserved and non-executable words used in an input C source file can be defined in an external file which is used by the tool at run time. Thus it does not matter which flavor of the C programming language is used for the analyses. The author of this study took advantage of this facility and used PC-METRIC to extract static measures from the parallel programs written in the iPSC/2-C programming language [Green89].

There is an exception to the flexibility of PC-METRIC. Programs written in the Pascal programming language style cannot be analyzed using PC-METRIC even if they are syntactically correct and properly compilable programs (see Example 2).

Example 2:

```
#define     BEGIN     {
#define     END       }
.
.
.
main()
BEGIN
        .
        .
        .
END
```

(Source: User's Guide for C, PC-METRIC, ver. 1.0, *Set Laboratories, Inc.*, Mulino, Oregon, pp 3-12, 1990.)

This is because PC-METRIC uses braces or curly brackets (i.e., "{" and "}") as delimiters of the body of the executable code.

If most of the source code is written in the Pascal style, the PC-METRIC's *User's Guide* recommends that the source file should be run through a preprocessor, which is a utility program available on the iPSC/2, to avoid spurious results. Empty procedures such as

```
procedureX()
{

}
```

may also produce spurious results, and should be taken out prior to the final analysis of an input source code file.

Before using PC-METRIC, some issues had to be resolved. As mentioned above, the tool is a PC-based software, thus all the programs in the compendium had to be down-loaded onto a floppy disk in ASCII (American Standard Code for Information Interchange) format prior to evaluation. Each application was given the same path and name as it had in the compendium except when it was prohibited by the DOS naming conventions.

PC-METRIC can be executed in four modes: Interactive, Command Line, Indirect, and Batch [PCMETRIC90]. The interactive mode was found to be the most convenient and was chosen for the source code analyses.

The normal output of PC-METRIC consists of two files <*filename*>.RPT and <*filename*>.EXP. The file with extension .RPT contains a complexity analysis report on a procedure by procedure basis and a complexity summary for the entire file. The file with extension .EXP contains the listing of all procedures which

exceed predefined complexity standards [PCMETRIC90]. Another file with the extension .ERR is created if errors are encountered. Figures C.2 and C.3, in Appendices C.1 and C.2, depict a sample output report generated by this software tool.

A few other points are also worth to mentioning here about PC-METRIC. For instance, if a statement such as

#define symb

encountered, then *symb* will be defined through the analysis of all the files or until a *#undef symb* statement is encountered. Contrary to this, a statement such as

#define FALSE 0

is not considered as a definition of a symbol (in this case *FALSE*) that is, any time the symbol is assigned a value, it is ignored. The header (.h) file or files must be entered or selected first among the source files to be analyzed so that all the definitions can be picked up.

Another point found interesting was the way PC-METRIC handles the occurrences of parentheses. In C, parentheses are used for three purposes: after a control statement, after a procedure call, or to change the default ordering of arithmetic operations. To differentiate between these uses, three different types of parenthesis have been defined in the reserved-word file (Appendix C.3). These three types were represented as '(c', '(p', and '(', respectively. Similarly, asterisk '*' and ampersand '&' each have two uses and hence are defined separately in the reserved-word file. Asterisk '*' is used to indicate the multiplication sign and '*p' to

indicate a pointer. Similarly, ampersand '&' is used to indicate the unary AND and '&p' to indicate the address operator.

The cyclomatic complexity was considered for this study because it counts the operators '&&' and '||', as well as the regular decision operators such as 'if', and 'while'. The counting strategy adopted in PC-METRIC was a modified form of the counting strategy discussed by Conte *et al.* [Conte86] for Pascal programs and implemented by Moll and Samadzadeh [Moll89] (refer to [PCMETRIC90] for a complete counting strategy used to collect the measurements).

The following two paragraphs describes the data collection procedure used to extract static measures from the parallel programs.

As mentioned above, that all the parallel programs included in the final analyses were ported to a microcomputer. Subsequently, the available tool was used to collect some of the Software Science metrics and McCabe's cyclomatic complexity metrics. However, before generating any report, the files for reserved-words and non-executable words were checked to make sure that all reserved and non-executable words are defined in the appropriate files.

As explained in Section 2.3, parallel programs for the iPSC/2 are each divided into two modules. One which runs on the host processor and the other which runs on the node processors. Each module may or may not contain more than one file. Halstead's Software Science metrics and McCabe's cyclomatic metrics were extracted from the programs. For Halstead's metrics, n_1, n_2, N_1, N_2, and the Effort E were collected for the host as well as for the node programs by feeding the files related to each module to the available tool as input. Token count for the whole application was measured by adding the operators and operands of each module instead of inputting all the files in the host and the node modules to the available tool. This was because each module was a separate entity, and a variable

used in a program related to the host module had no relation with a variable used in the node module with the same name as shown in Example 3.

Example 3:

Host Program (host.c)

```
.
.
.
main()
{
        .
        .
        .
        result = 1;
        if (result = =1)
                .....
        .
        .
        .
}
```

Node Program (node.c)

```
        .
        .
        .
main()
{
        .
        .
        .
        result = 2;
        if (result = = 2)
                .....
        .
        .
        .
}
```

In the above example, suppose the programs host.c and node.c were input together to PC-METRIC, then it would consider the variable *'result'* as a single unique operand. Intuitively speaking, the variable *'result'* in the programs host.c and node.c constitutes two separate operands, and hence should be counted as two unique operands.

The above approach was adopted in measuring McCabe's cyclomatic complexity metrics also, even though for this metric the files involved in both modules could be input together to the tool. TABLES III and IV contain the static measurements for selected McCabe's cyclomatic metrics and Halstead's Software Science metrics, respectively.

TABLE III

CYCLOMATIC COMPLEXITY MEASUREMENTS

Apl#	Host V(G)	Node V(G)	Total V(G)
1	4	6	10
2	2	12	14
3	16	65	81
4	47	62	109
5	46	107	153
6	28	30	58
7	33	40	73
8	4	17	21
9	4	17	21
10	11	25	36
11	8	15	23
12	12	138	150
13	4	29	33
14	6	127	133
15	6	127	133
16	5	42	47
17	6	152	158
18	18	27	45

For application names see Appendix B 2

TABLE IV

SOFTWARE SCIENCE MEASUREMENTS

	Host Metrics					Node Metrics					Overall Application Metrics				
Apl#	n_1	n_2	N_1	N_2	Effort	n_1	n_2	N_1	N_2	Effort	n_1	n_2	N_1	N_2	Effort
1	25	33	123	62	25451 0	30	29	151	80	56230 0	55	62	274	142	180012 4
2	23	19	69	36	12337 0	27	37	230	144	117901 0	50	56	299	180	258964 8
3	38	31	223	136	182794 0	51	77	1049	645	2532915 0	89	108	1272	781	5035564 1
4	60	106	701	359	794292 0	45	90	1068	647	1963117 0	105	196	1769	1006	6156799 3
5	61	113	827	446	1140590 0	73	190	1993	1169	5708385 0	134	303	2820	1615	13892220 6
6	50	68	583	316	718841 0	47	60	532	294	641207 0	97	128	1115	610	3115386 6
7	54	88	737	416	1052191 0	49	75	781	445	1239374 0	103	163	1518	861	5213123 2
8	28	34	201	105	78774 0	29	26	274	166	235497 0	57	60	475	271	659753 2
9	29	40	218	121	90830 0	30	31	359	217	358691 0	59	71	577	338	902370 8
10	30	38	210	108	82527 0	44	64	483	249	423225 0	74	102	693	357	1014296 2
11	30	37	195	100	72547 0	28	33	285	157	174600 0	58	70	480	257	549286 1
12	45	59	327	189	249198 0	62	124	1539	1010	4852372 0	107	183	1866	1199	8788218 1
13	29	42	138	86	40900 0	39	49	540	355	816741 0	68	91	678	441	1348325 7
14	36	57	279	151	134080 0	56	102	1847	1278	8007307 0	92	159	2126	1429	11715881 3

TABLE IV (*continued*)

SOFTWARE SCIENCE MEASUREMENTS

	Host Metrics					Node Metrics					Overall Application Metrics				
Apl#	n_1	n_2	N_1	N_2	Effort	n_1	n_2	N_1	N_2	Effort	n_1	n_2	N_1	N_2	Effort
15	35	56	239	131	98573 0	53	97	1714	1188	6808561 0	88	153	1953	1319	9820970 9
16	35	60	334	199	203246 0	49	78	957	622	2155954 0	84	138	1291	821	4113304 3
17	32	53	259	156	125266 0	50	87	2565	1608	13686523 0	82	140	2824	1764	18474018 9
18	36	72	438	281	341188 0	39	54	593	379	869898 0	75	126	1031	660	2541378 7

For application names see Appendix B 2

Since the available tool did not produce the extended size metrics (i.e., the number of blank and commented lines) and the communication metrics, two separate programs were developed to measure these metrics. For size metrics, a parallel program was developed (Appendix D) on the iPSC/2. The pseudocode of the program to collect size metrics is depicted in Appendix E.1. (because the same algorithm also was used in the program that collects the communication metrics).

The program collecting the size measures expects a syntactically correct parallel program as an input and produces as output four measures: the number of executable lines; the number of blank lines; the number of commented lines; and number of total lines in the input file. The size metric was divided into the above four categories so that analyses could be made to find out which metric or combination of metrics had more influence in terms of the comprehensibility of the parallel programs.

The program to extract the size measures is itself a parallel program. It has the capability to accept any number of files as input as there are nodes (processors) available on the system. The program processes all the input files at the same time (in a parallel fashion). However, since the iPSC/2 available on campus has 32 nodes, the program accepts a maximum of 32 files as input at one time and process them in parallel. The host module acts as a driver of the application and does the job of allocating a source code file to each node to extract the measurements. The host module is given access to a file that has a complete path listing of all the files that need to be processed (Appendix E.2). As soon as a node finishes extracting measurements from the file it was working on, it sends a message to the host with its node number and the collected metrics. The host receives the packet, saves the message and sends a new source file path to the same node. This process continues until all source files are processed.

Another program was developed to collect the proposed communication metrics (Appendix F) on the iPSC/2. This program uses the same algorithm (Appendix E.1) as developed for the program collecting several size metrics. Again, the host module works as a driver and allocates files to each node whenever the nodes are free and in return collects communication measurements.

A *makefile* [Green89] was written to compile the newly developed tools (Appendix E.3). The outputs generated by the parallel programs used for gathering the size and the communication metrics were then manually added for the host and the node modules and are depicted in TABLES V and VI, respectively.

TABLE V

SIZE MEASUREMENTS

	Host # of Lines			Node # of Lines			Total # of Lines			
Apl#	Exec	Blank	Comnt	Exec	Blank	Comnt	Exec	Blank	Comnt	Appl Length
1	41	12	73	58	29	64	99	41	137	277
2	29	8	22	79	19	74	108	27	96	231
3	159	42	168	439	114	600	566	140	721	1427
4	244	48	169	454	79	321	687	122	479	1288
5	253	50	182	448	87	356	691	131	517	1339
6	196	38	212	175	36	328	371	74	540	985
7	244	56	233	235	43	378	479	99	611	1189
8	66	25	56	102	31	71	168	56	127	351
9	70	26	65	102	32	84	172	58	149	379
10	45	9	60	116	16	113	161	25	193	379
11	52	14	57	76	12	71	128	26	128	282
12	138	48	151	349	110	442	487	158	593	1238
13	62	26	22	150	55	246	212	81	268	561
14	95	29	43	501	146	348	596	175	391	1162
15	100	35	60	497	151	364	589	182	411	1182
16	102	39	69	250	128	433	352	167	502	1021
17	89	84	20	795	228	73	884	312	93	1289
18	147	37	45	177	43	75	324	80	120	524

For application names see Appendix B 2

TABLE VI

COMMUNICATION COMPLEXITY MEASUREMENTS

	Host Communication Mesgs		Node Communication Mesgs		Total Communication Mesgs		
Apl#	Send	Receive	Send	Receive	Send	Receive	Total
1	1	2	4	3	5	5	10
2	0	1	3	2	3	3	6
3	3	2	2	2	5	4	9
4	9	3	10	16	19	19	38
5	9	3	10	16	19	19	38
6	6	3	9	12	15	15	30
7	6	3	9	12	15	15	30
8	5	2	7	10	12	12	24
9	5	2	7	10	12	12	24
10	5	2	10	10	15	12	27
11	5	2	5	8	10	10	20
12	2	3	10	9	12	12	24
13	1	1	3	5	4	6	10
14	2	5	12	10	14	15	29
15	2	5	11	10	13	15	28
16	4	4	12	10	16	14	30
17	3	2	32	33	35	35	70
18	9	2	4	12	13	14	27

For application names see Appendix B 2

As defined in Section 3.1.5, residual complexity is based on the notion that the understandability of a software document that can be modeled by a token

categorization process. In this study the classification schemes considered were based on Halstead's operator-operand token classification. The following three residual complexity classifications schemes were considered for this study:

1. Operator and Operand tokens;
2. Host and Node tokens; and
3. Host Operator, Host Operand, Node Operator, and Node Operand tokens.

Also, as mentioned in Section 3.1.5, after classifying the tokens into 'q' equivalence classes, the residual complexity R, is computed as

$$R = N_1 * Log(N_1) + N_2 * Log(N_2) + \ldots + N_q * Log(N_q)$$

where N_j is the number of tokens in the jth set for $1 <= j <= q$. Two definitions used in this study for N_j were:

i) count of the number of unique token in equivalence class j; and

ii) count of the total occurrences of tokens in equivalence class j.

This spawned 6 sets of measures, two for each of the three classification schemes defined above. Thus, for the kth ($1 <= k <= 3$) classification scheme two definitions namely R_k and R_{kuniq} were defined, where R_k was defined in terms of the total occurrences of tokens, and R_{kuniq} was defined in terms of the unique occurrences of tokens.

Since Halstead's basic token counts (n_1, n_2, N_1, N_2) were already measured for each of the application used for the final analysis, they were ported to another directory on a microcomputer, and LOTUS 1-2-3 [LOTUS83] was used to compute the residual complexity by simply embedding the formulas in LOTUS. The resulting residual complexity measures are depicted in TABLE VII.

TABLE VII

RESIDUAL COMPLEXITY MEASUREMENTS

	Classification I		Classification II		Classification III	
	R1U	R1T	R2U	R2T	R3U	R3T
1	687 1	3234 1	686 8	3207 0	570 6	2821 8
2	607 4	3807 5	610 4	3901 5	505 8	3444 5
3	1305 8	20622 7	1317 4	21217 3	1124 8	19249 8
4	2197 4	29119 5	2179 6	29078 7	1898 9	26460 1
5	3444 5	49532 9	3409 3	49893 1	3022 5	45698 3
6	1536 1	16931 0	1533 4	16825 0	1311 6	15208 3
7	1886 5	24436 7	1877 5	24305 8	1621 4	22059 4
8	686 8	6413 8	687 1	6390 5	570 6	5685 9
9	783 7	8131 9	783 2	8131 2	654 5	7262 0
10	1140 0	9566 9	1143 4	9608 9	970 8	8637 9
11	768 8	6332 7	768 2	6304 6	641 0	5617 1
12	296 7	32538 3	2099 1	33493 5	1825 6	30535 2
13	1006 1	10250 7	1005 0	10524 9	848 6	9442 5
14	1762 9	38477 6	1762 1	40041 8	1524 3	36589 9
15	1678 8	35020 9	1676 5	36537 8	1448 5	33358 0
16	1517 9	21289 8	1511 7	21604 4	1299 3	19569 0
17	1519 4	51397 0	1517 2	53797 3	1306 3	49387 7
18	1346 3	16501 9	1337 6	16470 1	1147 2	14838 3

For application names see Appendix B 2

Legend R1U, R1T stands for residual complexity measurements calculated for the two cases of Unique and Total occurrences of tokens in the Operator/Operand classification

R2U, R2T same as above except for the tokens in the Host/Node classification

R3U, R3T same as above except for the tokens in the Host Operators, Host Operands, Node Operators and Node Operands, respectively

Other techniques for collecting static measurement specifically for parallel programs, as discussed in the literature [Zuberek85 and Haban89], were also considered. But those techniques were more detailed and hardware oriented than what was needed for this study, hence they were dropped from further consideration.

CHAPTER V

ANALYSIS OF THE MEASUREMENTS

This chapter describes the methodology used in the analysis of static code measurements including a discussion on inter-metric correlations and correlations among static metrics and the experts' judgements. Also, six models derived based on the experts' perceptions and static metrics are discussed. Tables are used generously to elucidate the discussion. All data analyses were done on the IBM[1] mainframe (IBM 3090/200S) [IBM3090-89] using the SAS statistical package [SAS90a]. Standard statistical methods were used (e.g., as described by Conte *et al.* [Conte86]).

5.1 Choice of a Statistical Test

In comparison studies especially for small samples and whenever there is any doubt about assumptions, a nonparametric test is found to be more powerful and desirable than a parametric test [Gibbons71]. Conte *et al.* support Gibbons' statement and add that most nonparametric tests can be applied to data from ordinal scale effectively. Generally, speaking nonparametric statistics require fewer assumptions than their counterpart parametric tests where more restrictions are applied, because nonparametric statistics use the ranks of the observations in the sample and ignore the actual data. One important point to mention is that nonparametric statistics are a kind of transformation, since each measure is transformed into its own rank and hence helps eliminate undesirable outliers.

1 IBM is a registered trademark of International Business Machine Corporation.

In this study each software complexity metric measures complexity on a potentially different scale and the best way to compare them is by using their ranking in the sample data rather than their actual values.

Selection of a statistical test was not an easy job for this study, as is the case for similar studies such as [Moll89] and [Nandakumar89]. Conover [Conover71] describes it as frustrating, since the process of experimentation does not always lay bare the "truth". He adds that: "One experiment, with one set of observations, may lead two scientists to two different conclusions".

Several nonparametric test such as Friedman, Spearman, and Kendall can be found in the literature (see, e.g., [Conover71], [Daniel78], and [Gibbons71]). The nonparametric statistical test chosen for this study for correlation analysis is the Spearman Rank Correlation Coefficient test. This was done after consulting with Dr. P. Larry Claypool, Professor of Statistics, at Oklahoma State University, and Dr. William L. Woodall, Professor of Statistics, at the University of Alabama. As mentioned in Section 4.2, the choice of this nonparametric test was made especially because of two reasons: first, limited data was available and distributional assumptions, e.g., that a distribution is normal could not be made, for a parametric test; second, the author was interested in checking the monotonicity among the observations and hence among the selected metrics rather than just in checking their linear correlations.

5.2 Inter-metric Correlations

Inter-metric correlations using the Spearman Rank Correlation Coefficient statistical test [SAS90b] are included in Appendix G. Variable names used in the correlation coefficient analysis and the variable names used in the regression analysis are included in the Appendix H with their short descriptions. Some of the

important correlations within a metric type and among the metrics are discussed below. As a result of these correlations analyses, several interesting points came to surface which are discussed in the following sections.

As explained in Section 3.1.1, the size metric is widely accepted and considered as a basic measure for some models of software development. TABLE VIII represents some of the interesting figures from among the static metric correlations included in Appendix G.

TABLE VIII

EXTRACTED SIZE MEASUREMENTS

Metric	Mean	Approximate Percentage	Std Deviation
Host Executable Lines	118 44	30	78.80
Host Commented Lines	94 83	28	69.95
Node Executable Lines	277 94	70	204 74
Node Commented Lines	246 72	72	170 11
Total Executable Lines	393.00	-	239 23
Total Commented Lines	337.55	-	212 93
Host Cyclomatic Complexity	14 44	20	14 45
Node Cyclomatic Complexity	57 66	80	49 41
Total Cyclomatic Complexity (Host and Node)	72 11	-	53 19
Total Host Communication Complexity (Message Sends and Receives)	-	27	-
Total Node Communication Complexity (Message Sends and Receives)	-	73	-

Generally speaking, a host module on a loosely-coupled distributed memory parallel machine (such as the iPSC/2), acts as a driver for an application. It manages the distribution of work for the nodes participating in the execution of an

application. This fact was supported by this study too. Notice that in the Mean column of TABLE VIII only 30% of the total executable code belongs to the host modules and the rest belongs to the node modules. Both the host and the node modules were found to be comparably proportional as far as the executable lines of code and the documentation lines are concerned.

An interesting point to mention here is that a host module runs on a single processor whereas a node module runs on several processors. Is it then appropriate to divide 70% of the code by the total number of nodes that participated in the application execution in order to find out the Mean of the executable lines of code for each node (or processor)? Intuitively speaking, the answer is negative, because a node may be executing only 10% of the executable code which may consist of a loop statement (such as a 'for' loop that counts numbers from 1 to 1 million), whereas its neighboring node could be doing a relatively simple work such as assigning and initializing a number of variables and that code may be 20% of the executable code.

Steep standard deviation values of executable code for the host and the node modules are an evidence of variation in their sizes. Correlations between the executable lines of code and the documentation (that is, commented lines) of the host modules were better than the correlations between the executable lines of code and the commented lines of code of the node modules. This could be interpreted as more consistency in the proportion of the executable code and the documentation in the host modules than in the node modules. Strong, positive correlations were found between the host and the node executable lines of code, and Halstead's E of the overall application and the node efforts, respectively, even though the sizes of the modules vary considerably. The significance levels in the above two correlations were less than 0.01

Another interesting point was that even though the Mean executable lines of code of the host was 30%, it contained only 20% of the cyclomatic complexity. Does

it mean that the host modules are less complex than the node modules? The answer is affirmative as far as the parallel programs used in this study are concerned. However, more research needs to be done to support the above answer. Strong, positive correlation exist between the executable lines of code of the node modules and the cyclomatic complexity; more so than between the executable lines of code and the cyclomatic complexity of the host modules.

Approximately 27% of message send or receive statements were found in the host modules and 73% were found in the node modules. This again supports the general fact regarding distributed-memory parallel machines such as the iPSC/2, that the host module acts as a driver of an application and the node module do all the complex computations. Another observation is that there was more communication going on among the nodes than between the host and the nodes or *vice versa*. Relatively weak correlation exist between communication statements and the executable lines of code (mostly at significance levels of 0.05 or less). This was expected, because communication in parallel programs is, in general, independent of the size of a program.

Residual complexity schemes, in which total occurrences of tokens were considered, correlated better with the executable lines of code of the node modules and the overall application's lines of code metrics than when unique occurrences of tokens were considered. On the other hand correlation between the lines of code of the host modules and the residual complexity metrics was higher when unique occurrence of tokens were considered. The significance levels in both cases (unique or total occurrences) was much less than 0.01.

In Halstead's token count no matter what counting strategy is used, the number of unique operators (n_1) should be less than or at most equal to the number of unique operands (n_2) in a source code file (because there can be no operators without at least one operand). This was verified by the operator and operand counts

for both the host and the node modules. It was noticed that the correlation between the node effort and the total effort was significantly higher than the correlation between the host effort and the total effort, at approximately the same significance levels (less than 0.01). Within the host as well as the node modules, total operators (N_1) and operands (N_2) correlate slightly better than the unique operators and operands with the respective efforts of the host or the node modules.

Both the cyclomatic complexity of the node and the host modules correlate positively with their respective effort measurements at significance levels of less than 0.01. However, correlation between the cyclomatic complexity and the effort of the node modules was stronger than the correlation between the cyclomatic complexity and effort of the host modules.

Residual complexity metrics correlated with Halstead's metrics in the same manner as they correlated with the size metrics described above. Residual complexity schemes, in which total occurrences of tokens were considered, correlated better with the node's and each overall application's Software Science metrics than when unique occurrences of tokens were considered. But the Software Science metrics for the host modules correlated better with the residual complexity when unique occurrences of tokens were considered. The significance levels in both cases (unique or total occurrences) was much less than 0.01.

Weaker correlations were found between the cyclomatic complexity and the communication metrics, suggesting that the two are not dependent on each other (that they measure different dimensions of software complexity). Correlations between the cyclomatic complexity and residual complexity metrics of the host modules were weaker than the correlations between the cyclomatic complexity and residual complexity metrics of the node modules at significance levels of less than 0.05. This was due to the fact that the node modules have more operators and operands than the host modules.

The proposed communication metrics were not found to correlate significantly with residual metrics at significance levels of 0.05, suggesting that the communication metrics are independent of the six token classification measures considered in this study. Residual metric may need to be divided into further classifications, such as the host and the node communication statements, to find better correlations with the communication metrics.

Finally, the six measurements considered for residual complexity metrics were found to correlate to each other strongly and positively at significance levels much less than 0.01.

5.3 Analysis of the Subjective Ratings

As discussed in Section 4.3.1, a questionnaire (Appendix B.1) was devised to correlate the perceived complexity of a number of experts to the five metrics considered in this study. The questionnaire was mailed electronically to the original compiler of the compendium of parallel programs used in this study [Compendium90], Dr. G. B. Lamont of the Air Force Institute of Technology (AFIT). Prior to the design of the questionnaire, the author of this thesis made a personal trip to AFIT in Dayton, Ohio, to discuss several aspects of this study with Dr. Lamont [Lamont90].

A problem had to be resolved after receiving the replies to the questionnaires but prior to the use of the Spearman Rank Correlation Coefficient test. This problem was how to merge the experts' rating of each application in the questionnaire. This issue was solved after consultation with Dr. P. Larry Claypool, Professor of Statistics at Oklahoma State University, by adding the individual ratings for questions 5 through 9 (see Figure 1) in the questionnaire separately. For instance, if the experts' replies to question 5 were 4, 4, 3, and 4, then the total for

question 5 would be the sum of the above four ratings, that is, 15. For question 10 (see Figure 1), the ratings were converted into numeric ratings and then added together. Appendix B.2 includes the individual ratings and their sums. As a different approach, accumulated rating could have been divided by the total number of experts who participated in the study in order to normalize the results. But this was avoided because it would not have helped in the analyses and was considered just an extra unnecessary step.

Q #5 How would you rate the UNDERSTANDABILITY of the HOST program(s) of the following applications on a scale of 1 to 5?

(Assume 1 indicates the poorest level and 5 the highest level for questions 5 through 9)

Q #6 How would you rate the UNDERSTANDABILITY of the NODE program(s) of the following applications on a scale of 1 to 5?

Q #7 How would you rate the documentation of the HOST program(s) of the following applications on a scale of 1 to 5?

Q #8. How would you rate the documentation of the NODE program(s) of the following applications on a scale of 1 to 5?

Q #9 How would you rate the overall perceived or conceptual COMPLEXITY (different from computational complexity) of the following applications on a scale of 1 to 5?

Q #10. If the following applications had been developed as sequential programs, do you think they would have taken less/more/same amount of time and effort?

Figure 1 QUESTIONS 5 THROUGH 10 OF THE QUESTIONNAIRE

Once the above problem was resolved, the Spearman test was applied to find out the correlations between the experts' perceived complexity ratings and the static measurements (Appendix G).

Some of the more important correlations at significance levels of less than or equal to 0.05 are depicted in TABLES IXa through IXf and are discussed below. As mentioned earlier, short descriptions of all the variable names used in the correlations are included in Appendix H.

TABLE IXa

IMPORTANT CORRELATIONS AMONG THE EXPERTS' REPLIES

	Q5	Q6	Q7	Q8	Q10
Q5	1 00	**0.87**	0 88	0 91	0 70
Q7	0 88	0 99	1 00	**0.96**	0 91
Q9	**-0.83**	**-0.87**	**-0.89**	**-0.94**	**-0.91**

This paragraph interprets the correlations shown in TABLE IXa. Strong, positive correlations between questions 5 and 6, and also between questions 7 and 8, suggest that the replies were consistent with respect to the understandability and documentation of the host and the node modules. Negative correlations between question 9 and questions 5, 6, 7, and 8 were expected. The reason for the anticipated negative correlations, as discussed in Section 4.3.1, was the nature of the redundancy embedded in the questions asked to compare the consistency in the participants' replies. Strong, negative correlation between questions 9 and 10 suggested that parallel programs with relatively less conceptual complexity might

have taken relatively more effort if they had been rewritten as sequential programs. This was a surprise to the author too.

TABLES IXb through IXf depict the correlations between the subjective ratings of the perceived complexity of the applications with the five static measurements considered in this study. Weak correlations (weaker than expected) were found at the significance levels of 0.05. Experts' judgements regarding the perceived complexity of the applications correlates better with the executable lines of code of each application than with the total lines of code in the applications. However, the executable lines of code of the node modules correlated better among the three measures shown in TABLE IXb.

TABLE IXb

Q9 VS THE SIZE METRICS

	Node Executable Lines	Application Executable Lines	Total Lines
Q9	0 62	0 55	0 50

With Halstead's measurements, question 9 (that is, the experts' perceived complexity rating) correlated with the effort of the overall application quite satisfactorily (TABLE IXc). However, a higher correlation was found between question 9 and the effort of the node modules than the effort of the overall application. Notice that none of the host metrics correlated with the experts' perceived complexity ratings at the significance levels of 0.05 or less (See Appendix G). This was a little unusual and unexpected.

TABLE IXc

Q9 VS THE SOFTWARE SCIENCE METRICS

	Node n1	Node n2	Node N1	Node N2	Node Effort
Q9	0.56	0.53	0.67	0.70	0.72
	Application n2	Application N1	Application N2	Application N1+N2	Application Effort
Q9	0.46	0.63	0.65	0.65	0.63

The cyclomatic complexity of the node modules correlated better with question 9 than each overall application's cyclomatic complexity at the significance level of much lesser than 0.05 (TABLE IXd). With the communication metrics, only the message sends metric of the node modules correlated, although weekly, with question 9 at the significance level of less than 0.02 (TABLE IXe). It was also observed that the residual complexity metrics correlated to a fair degree with the perceived complexity. Notice that, at the significance levels of 0.05 or less, only those residual metrics which were based on total occurrences of tokens were adequately correlated to the experts' perceived complexity (TABLE IXf).

TABLE IXd

Q9 VS THE CYCLOMATIC COMPLEXITY METRICS

	Node V(G)	Application V(G)
Q9	0 73	0.61

TABLE IXe

Q9 VS THE COMMUNICATION METRICS

	Node Message Sends
Q9	0.57

TABLE IXf

Q9 VS THE RESIDUAL METRICS

	R1T	R2T	R3T
Q9	0 65	0 65	0 65

5.4 Proposed Models

To study the relationships between the chosen metrics and the experts' perception of relative comprehensibility of parallel programs (question 9), the Stepwise Linear Regression Analysis [SAS90b] was used. For this purpose, six Stepwise Linear Regressions, one against each metric and one by including all possible combinations of five static measures, were run. The following are the resulting six models. For each model presented below, the submetrics chosen were based on the author's intuition and best judgement. Since question 9 represents the perceived complexity by the experts, the acronym PC is used in the following models. In each model, first full model is presented followed by the proposed model. The standard error for each independent variable and residuals for each observation are included in Appendix I. Detailed descriptions of variable names used in the following models are included in Appendix H.

In the proposed models, all variables left in the models are significant at the 0.15 level, which is also a default level for Stepwise Linear Regressions analysis used in the SAS package. The coefficient values with one standard error are presented in the following format in each model:

(parameter value $^{+}$ one standard error)

1. A model considering the size measurements:

Full model:

$$PC = a_0 + a_1 * HEXELNS + a_2 * HCMTLNS + a_3 * NEXELNS + a_4 * NCMTLNS + a_5 * TEXELNS + a_6 * TCMTLNS + e$$

where e (read as epsilon) stands for residual error.

Proposed model:

$$PC = (8.49 \pm 1.02) + (0.0086 \pm 0.0030) * NEXELNS + e$$

where NEXELNS stands for the executable lines of code of the node modules. The sum of squared residuals is 103.10 and the R-square is equal to 0.33.

2. A model considering the Software Science measurements:

Full model:

$$PC = a_0 + a_1 * HUN1 + a_2 * HUN2 + a_3 * + a_4 HCAPN1 + HCAPN2 + a_5 * HEFRT + a_6 * NUN1 + a_7 * NUN2 + a_8 * NCAPN1 + a_9 * NCAPN2 + a_{10} * NEFRT + a_{11} * TUN1 + a_{12} * TUN2 + a_{13} * TUN1N2 + a_{14} * TCAPN1 + a_{15} * TCAPN2 + a_{16} * TCAPN1N2 + a_{17} * TEFRT + e$$

where *e* (read as epsilon) stands for residual error.

Proposed model:

$$PC = (5.8 \pm 1.4) + (0.0132 \pm 0.0048) * NCAPN1 + (0.00000141 \pm 0.00000064) * TEFRT + e$$

where NCAPN1 and TEFRT stand for the total operators of the node module and the overall effort, respectively, of each application. The coefficient of TEFRT has six significant digits after the decimal point, this is because the data was not been normalized (see TABLE IV). The sum of squared residuals is 69.59 and the R-square is equal to 0.55.

3. A model considering cyclomatic complexity measurements:

Full model:

$$PC = a_0 + a_1 * HOSTVG + a_2 * NODEVG + a_3 * TOTVG + e$$

where *e* (read as epsilon) stands for residual error.

Proposed model:

$$PC = (8.58 \pm 0.86) + (0.04 \pm 0.01) * NODEVG + e$$

where NODEVG stands for the cyclomatic complexity of the node module of each application. The sum of squared residuals is 89.04 and R-square is equal to 0.42.

4. A model considering the communication complexity measurements:

Full model:

$$PC = a_0 + a_1 * HMSGSND + a_2 * HMSGREC + a_3 * NMSGSND + a_4 * NMSGREC + a_5 * TMSGSND + a_6 * TMSGREC + a_7 * TCOMMSG + e$$

where e (read as epsilon) stands for residual error.

Proposed model:

$$PC = (8.56 \pm 1.02) + (0.262 \pm 0.093) * NMSGSND + e$$

where NMSGSND stands for the message send statements of the node module of each application. The sum of squared residuals is 104.13 and R-square is equal to 0.33.

5. A model considering the residual complexity measurements:

Full model:

$$PC = a_0 + a_1 * R1U + a_2 * R1T + a_3 * R2U + a_4 * R2U + a_5 * R3U + a_6 * R3T + e$$

where e (read as epsilon) stands for residual error.

Proposed model:

$$PC = (4.52 \pm 2.07) - (0.0033 \pm 0.0012) * R1T + (0.0238 \pm 0.0090) * R2T - (0.0223 \pm 0.0091) * R3T + e$$

where R1T, R2T, and R3T stand for the sizes of classes (in the case where the total occurrences of tokens are considered) in the classifications schemes defined in Section 3.1.4. The sum of squared residual is 61.26 and the R-square is equal to 0.60.

6. A model considering selected submetrics among the five static measurements:

Full model:

$$PC = a_0 + a_1 * TEXELNS + a_2 * TCMTLNS + a_3 * TOTLNS + a_4 * TUN1 + a_5 * TUN2 + a_6 * TUN1N2 + a_7 * TCAPN1 + a_8 * TCAPN2 + a_9 * TEFRT + a_{10} * TOTVG + a_{11} * TMSGSND + a_{12} * TMSGREC + a_{13} * TCOMMSG + a_{14} * R1U + a_{15} * R1T + a_{16} * R2U + a_{17} * R2T + a_{18} * R3U + a_{19} * R3T + e$$

where e (read as epsilon) stands for residual error.

Proposed model:

$$PC = (4.9 \pm 1.4) + (0.037 \pm 0.0114) * TCAPN2 + (0.182 \pm 0.105) + * TMSGSND - (0.00118 \pm 0.00396) * R1T + e$$

where TCAPN2 stands for the total operands in each application, TMSGSND stands for the total message send statements in each application, and R1T stands for the residual complexity calculated for the case of total occurrences of token when tokens were classified as operators and operands. The sum of squared residuals is 58.37 and the R-square is equal to 0.62.

The R-square value (Appendix I), also called the coefficient of determination, is the square of the correlation between dependent variables and the predicted values. The significance probability, Prob>F (Appendix I), is the probability of getting a greater F statistic [SAS90b] than that observed if the hypothesis is true. The steady increase of the R-square value in the Stepwise Regression Analysis indicates the appropriateness of the models presented.

Another sign of the appropriateness of the above six models is the significance probability, i.e., the Prob>F levels, which in these cases are much less than 0.01. Notice that almost all the models are heavily dependent on the nodes' tokens. The intercept and other coefficients' values are given in Appendix I.

Other statistical methods such as Nonlinear Regression Analysis [SAS90b] were considered for this study. However because of the small sample size it was decided that the results of these methods would not be very reliable.

CHAPTER VI

EPILOGUE AND FUTURE WORK

There are many ways to measure the performance of a parallel system. Several studies conducted by the researchers [Zuberek85, Haban89, and Karp90] are mostly from the hardware point of view measuring, among other things, the inter-processor communication or parallel processors performance. This author found a lack of literature discussing the relationship between conceptual complexity and structural complexity of parallel programs and hence decided to explore this area.

Before discussing the conclusions, two points need to be mentioned: 1) since this was the first study of its kind, the conclusions of this study should be interpreted as observations, and 2) the final analysis and the proposed models should be construed as general templates for hypotheses in future studies.

On the average, 20% of the cyclomatic complexity and 27% of the communication complexity were found in the host module that had an average 30% of the executable lines of code of the applications considered in this study. This was expected, as the host modules are generally considered as drivers of applications and are relatively less complex than their counterpart, the node modules. Another reason why the host module carries less percentage of the code is the fact that the host program runs on a single processor whereas the node module splits the code among several processors on a parallel machine such as iPSC/2.

In this study five sets of metrics were investigated including the proposed communication metrics. Almost all five static metrics, at different significance

levels, were found to be strongly correlated to each other. This supports the use of the metrics, which were originally proposed for sequential programs, to measure the structural complexity of parallel programs.

Residual complexity, that attempts to quantify the understanding process of a software document by dividing it into different classes of tokens, correlates strongly and positively with the size metric. However, residual complexity correlates better when the classifications are based on the total occurrences of tokens as opposed to the classifications based on the set of unique tokens. To get higher correlations with residual complexity, further token classifications need to be described.

Weaker correlation between the cyclomatic complexity and the communication metrics suggested that the two are not directly dependent on each other and perhaps they measure different dimensions of the structural complexity of software. The same situation was found in the cases of correlations between the cyclomatic complexity and the communication complexity with residual complexity metrics.

It is evident from the data that there is very little discrimination among the experts' replies. For instance, consider the replies number 1 through 11 (Appendix B.2), the total number of participants who replied to question 5 (column labeled "Q5") is 14. This predicts that either the question was too general, i.e., it was not specific enough so that a participant could reply differently or the sample data was too little to obtain some reliable Stepwise Regression analysis and hence present a meaningful model.

The R-square (or coefficient of determination) and the significance probability (Prob > F levels) are the two major values to be considered to probe the healthiness of a model. The six models presented in Section 5.4 showed the significance probabilities less than 0.01. The R-square values for the models varied between good to moderate, as the models accounted for the variation from

approximately 62% to 32%. As explained earlier, since the sample size was small, these models may not represent truly their respective populations. The models given in Section 5.4 provide a reasonable approximation that could be considered as hypotheses for future research and tested empirically on a larger set of programs and/or with a larger population of participants with varying levels of expertise.

The compendium of parallel programs used in this study has lot to be explored (either as future work related to this thesis or unrelated to this thesis).

This study was specific to parallel programs written on Intel iPSC family of concurrent supercomputers. Future studies may address some of the issues that were not discussed in this study. An issue that can be investigated is to find out the distribution of the code that resides on the node modules among the nodes participating in the execution of an application. Another topic for future study is to consider the programs that belong to some specific categories, such as sorting or simulation programs, and find out which program is an optimal solution to the problem (in terms of being least complex) and why, or what is the optimal size of a sorting program. Other future work may involve the comparison or correlation of the growth of the host modules and/or node modules with respect to complexity metrics. Also more refined and/or different classification schemes for residual complexity metrics could be defined to find better correlations between residual complexity metrics and perceived complexity. Programs in the compendium could also be used to evaluate the effort needed to write the same application on other parallel machines such as the Sequent [Sequent89], a tightly-coupled, shared-memory parallel machine. Control flow in the parallel program using graph theory could also be constructed and quantified to be compared with other structural metrics.

REFERENCES

[Basili86]
V. R. Basili, R. W. Selby, and D. H. Hutchens, "Experimentation in Software Engineering," *IEEE Trans. Software Eng.*, vol. SE-12, pp. 733-743, July 1986.

[BBN89]
BBN Advanced Computers Inc., Cambridge, MA, 1989.

[Bishop87]
M. Bishop, "Profiling under UNIX by Patching," *Software--Practice & Experience*, vol. 17, pp. 729-739, Oct. 1987.

[Boehm81]
B. W. Boehm, *Software Engineering Economics*, Prentice-Hall, Englewood Cliffs, NJ, 1981.

[Close88]
Paul Close, "The iPSC/2 Node Architecture," *The Third Conference on Hypercube Concurrent Computers and Applications*, Pasadena, California, vol. I, pp. 43-50, January 1988.

[Compendium90]
G. B. Lamont and R. A. Beard, *Compendium of Parallel Programs for the Intel iPSC Computers*, vol. 1,2,3, ver. 1.4, Dept. of Electrical and Comp. Eng., School of Eng., Air Force Inst. of Tech., Wright-Patterson AFB, Dayton, OH, October 1990.

[Conover71]
W. J. Conover, *Practical Nonparametric Statistics*, John Wiley & Sons Inc., New York, NY, 1971.

[Conte86]
S. D. Conte, H. E. Dunsmore, and V. Y. Shen, *Software Engineering Metrics and Models*, Bengamin/Cummings, Menlo Park, CA, 1986.

[Daniel78]
W. W. Daniel, *Applied Nonparametric Statistics*, Houghton Mifflin Company, Boston, MA, 1978.

[DOS87]
International Business Machines Corporation, P.O. Box 1328-W, Boca Raton, FL, 1987.

[Encore89]
Encore Computer, Marlborough, MA, 1989.

[Fox88]
G. Fox, M. Johnson, G. Lyzenga, S. Otto, J. Salmon, and D. Walker, *Solving Problems on Concurrent Processors,* vol. I, Prentice-Hall, Englewood Cliffs, NJ, 1988.

[Gibbons71]
J. D. Gibbons, *Nonparametric Statistical Inference*, McGraw-Hill Book Company, New York, NY, 1971.

[Goldberg86]
R. Goldberg, "Software Engineering: An Emerging Discipline," *IBM Syst. J.*, vol. 25, nos. 3 & 4, pp. 334-353, 1986.

[Graham83]
S. L. Graham, P. B. Kessler, and M. K. McKusick, "An Execution Profiler for Modular Programs," *Software--Practice & Experience*, vol. 13, pp. 671-685, 1983.

[Green89]
Green Hills Software, Inc., *iPSC/2 - C Language Reference Manual*, Green Hills Software, Inc., CA, 1989.

[Haban89]
D. Haban and D. Wybranietz, "Monitoring and Measuring Parallel Systems Using a Non-Intrusive, Rule-Based Evaluation System," *Technical Report* TR-88-007, ICSI, Berkeley, CA, March 1989.

[Halstead77]
M. H. Halstead, *Elements of Software Science*, Elsevier Nort-Holland, Inc., New York, NY, 1977.

[Hayes88]
J. P. Hayes, *Computer Architecture and Organization*, McGraw-Hill Inc., New York, NY, 1988.

[Henry79]
S. M. Henry, "Information Flow Metrics for the Evaluation of Operating Systems' Structure," *Ph.D Dissertation*, Iowa State Univ., Ames, IA, 1979.

[IBM3090-89]
"*3090 Processors Complex - Functional Characteristics*," International Business Machine Corporation, Publication number SA22-7121-8, Seventh Edition, Poughkeepsie, NY, 1989.

[Intel88]
Intel Scientific Computers, Beaverton, Oregon, 1988.

[iPSC88]
The iPSC/2 User's Guide, Intel Scientific Computers, Beaverton, OR, 1988.

[Karp90]
A. H. Karp and H. P. Flatt, "Measuring Parallel Processor Performance," *Communications of the ACM*, vol. 33, no. 5, pp. 539-543, May 1990.

[Kernighan78]
B. W. Kernighan and D. M. Ritchie, *The C Programming Language*, Prentice-Hall, Englewood Cliffs, NJ, 1978.

[Lamont90]
G. B. Lamont, *Private Communication*, Dept. of Electrical and Comp. Eng., School of Eng., Air Force Inst. of Tech., Wright-Patterson AFB, Dayton, OH, October 1990.

[LOTUS83]
Lotus Development Corporation, *User's Manual*, Release 2, 161 First Street, MA, 1983.

[McCabe76]
T. J. McCabe, "A Complexity Measure," *IEEE Trans. Software Eng.*, vol. SE-2, pp. 308-320, December 1976.

[Moll89]
K. E. Moll and M. H. Samadzadeh, "An Empirical Study of the Relationship Between Static Software Complexity Metrics and Dynamic Measurements of Pascal and C Programs," *Proceedings of the 1989 ACM South Central Regional Conference*, Tulsa, OK, pp. 150-157, November 1989.

[Nandakumar89]
C. K. Nandakumar, "Quantifying the Software Maintenance Task: An Empirical Study of Complexity Metrics Across Versions," *Masters Thesis*, Computer Science Department, Oklahoma State University, Stillwater, OK, May 1989.

[Nugent88]
S. F. Nugent, "The iPSC/2 Direct-Connect Communications Technology," *The Third Conference on Hypercube Concurrent Computers and Applications*, Pasadena, CA, vol. I, pp. 51-60, January 1988.

[PCMETRIC90]
Set Laboratories, Inc., PC-METRIC, ver. 1.0, Mulino, OR, 1990.

[Pierce88]
Paul Pierce, "The NX/2 Operating System," *The Third Conference on Hypercube Concurrent Computers and Applications*, Pasadena, CA, vol. I, pp. 384-390, January 1988.

[Rattner85]
J. Rattner, "Concurrent Processing: A New Direction in Scientific Computing," *AFIPS Conference Proceedings*, Chicago, IL, vol. 54, pp. 157-166, July 1985.

[Samadzadeh88]
M. H. Samadzadeh and W. R. Edwards, Jr., "A Classification Model of Software Comprehension," *21st Hawaii Int. Conf. on System Sciences (HICSS21)*, HI, 1988.

[Sequent89]
Sequent Computer System, Inc., "*Guide to Parallel Programming - On Sequent Computer Systems*," Editor: Anita Osterhaug, Printice-Hall, Englewood Cliffs, NJ, 1989.

[SAS90a]
SAS/STAT User's Guide, ver. 6, Fourth Edition, vol. 1, SAS Inst., Cary, NC, 1990.

[SAS90b]
SAS/STAT User's Guide, ver. 6, Fourth Edition, vol. 2, SAS Inst., Cary, NC, 1990.

[Seitz85]
C. L. Seitz, "The Cosmic Cube," *Communications of the ACM*, vol. 28, no. 1, pp. 22-33, January 1985.

[Stone87]
H. S. Stone, *High-Performance Computer Architecture*, Addison-Wesley Publishing Company, Reading, MA, 1987.

[UNIX86]
The UNIX System V User's Manual, AT&T, Prentice-Hall, Englewood Cliffs, NJ, 1986.

[Zuberek85]
W. M. Zuberek, "Performance Evaluation of Concurrent Systems Using Timed Petri Nets," *Proc. ACM Computer Science Conference*, Denver, CO, pp. 326-329, March 1985.

APPENDICES

APPENDIX A

A SAMPLE PROGRAM

```
/*------------------------------------------------------------------------------------------------------------------------*/
/*                                                                                                                        */
/*      File            host c   (Host program)                                                                           */
/*      Author          Intel Corporation, June 1989                                                                      */
/*      Modified by     Imtiaz Ahmad, February 10, 1990                                                                   */
/*      Class           ECEN 5303      (Parallel Processing)                                                              */
/*      Assignment #    1                                                                                                 */
/*      Purpose         To learn about programming on the iPSC/2 concurrent                                               */
/*                      computer                                                                                          */
/*                                                                                                                        */
/*------------------------------------------------------------------------------------------------------------------------*/
/*                                                                                                                        */
/*      Problem                                                                                                           */
/*              Write a program on the iPSC/2 to sum integers from integer a to b                                         */
/*                                                                                                                        */
/*                      ------------------------------------------------------                                            */
/*                                                                                                                        */
/*      This program sums the numbers that exist between two limits  The user provides                                   */
/*      input in the form of lower and upper limits, with the number of processors to be                                  */
/*      used  To calculate sum, the program uses parallel processing and clocks the                                       */
/*      solution time                                                                                                     */
/*                                                                                                                        */
/*      The performance results are stored in an output file named OUTPUT  Input                                          */
/*      data comes from the INPUT file in the form of the number of processors needed                                     */
/*      and the lower and upper limits of the range                                                                       */
/*                                                                                                                        */
/*      Called by       None                                                                                              */
/*      Calling         print_header()              (internal function)                                                   */
/*                      user_input()                (external function)                                                   */
/*                      print_report()              (internal function)                                                   */
/*                                                                                                                        */
/*      Message sending to      user_input()        (external function)                                                   */
/*      Message receiving from user_input()         (external function)                                                   */
/*                                                                                                                        */
/*      MAIN LOGIC AND SOURCE CODE WAS COPIED FROM THE DIRECTORY                                                          */
/*      /usr/ipsc/examples/c                                                                                              */
/*                                                                                                                        */
/*------------------------------------------------------------------------------------------------------------------------*/

#include <stdio h>
#include <cube h>

#define INPFILE         "input"         /* input data file name */
#define OUTFILE         "output"        /* input data file name */
#define HOST_PID        100             /* process id of the host process */
#define NODE_PID        0               /* process id for node processes*/
#define INIT_TYPE       0               /* type of initialization message*/
#define SIZE_TYPE       2               /* type of size message */
#define PART_TYPE       10              /* type of partial summation message */
#define ALL_NODES       -1              /* symbol for all nodes */
#define ALL_PIDS        -1              /* symbol for all processes */

struct msg_type {                       /* structure for parameters of summation */
        double  a,                      /* lower limit of summation */
```

```
        b,                          /* upper limit of summation */
    long  points,                   /* number of points in quadrature rule*/
},

struct msg_type msg,                /* pointer to summation structure */

int     size,                       /* number of working nodes */
double  llimit, ulimit,             /* stores lower and upper limits */

long    tms, ms, tsec,              /* time calculation variables */
        sec, min,

FILE    *inp, *out, *fopen(),       /* pointer to input and output files */

main()
{       /* Host main */
        /* open input and output file and print header */
        inp = fopen(INPFILE, "r"),
        out = fopen(OUTFILE, "w"),
        getcube ("", "32", "", 0),          /* allocate given number of nodes */

        setpid(HOST_PID),
        print_header(),                     /* print report header */

        /* Load all nodes with pid NODE_PID  */
        load ("node", ALL_NODES, NODE_PID),

        for (,,) {                          /* Infinite loop */

                /* Get user input from a file  */
                if (!user_input(&msg, &size)) break,

                llimit = msg a,             /* saving lower limit */
                ulimit = msg b,             /* saving upper limit */

                /*
                 * Send message containing number of working nodes to all nodes
                 */
                csend(SIZE_TYPE, &size, sizeof(size), ALL_NODES, NODE_PID),

                /*
                 * Send message containing the integration parameters to all nodes  */
                 */
                csend(INIT_TYPE, &msg, sizeof(msg), ALL_NODES, NODE_PID),

                /*
                 * Wait to receive message containing the summation result and */
                 * process execution time
                 */
                crecv(PART_TYPE, &msg, sizeof(msg)),


                                            /* Calculate the time interval */
                tms  = msg points,
```

```
                ms   = tms % 1000,
                tsec = (tms - ms) / 1000,
                sec  = tsec % 60,
                min  = (tsec - sec) / 60,

                print_report(size, llimit, ulimit, msg.b, min, sec, ms, msg a),
        } /* End infinite loop */

        killcube(ALL_NODES, ALL_PIDS),
        relcube(),                          /* release attached cube*/
        close (inp, out),

        printf("Normal termination of the program \n"),
} /* End host main */

/*-------------------------------------------------------------------------------------------------------------------------*/
/*      This function prints the performance report header                                                                 */
/*                                                                                                                         */
/*      Called by       main()          (internal function)                                                                */
/*      Calling         None                                                                                               */
/*-------------------------------------------------------------------------------------------------------------------------*/
print_header()
{
        fprintf(out, "\t   Following is the performance report for the given data\n"),
        fprintf(out, "\t   ----------------------------------------------------\n"),
        fprintf(out, "# of  Lower     Upper              Basic  Elapsed-Time\n"),
        fprintf(out, "prcs  limit     limit     Range    slices min sec   ms              SUM\n"),
        fprintf(out, "-------------------------------------------------------------------------------\n"),
}

/*-------------------------------------------------------------------------------------------------------------------------*/
/*      This function prints the performance report                                                                        */
/*                                                                                                                         */
/*      Called by       main()          (internal function)                                                                */
/*      Calling         None                                                                                               */
/*-------------------------------------------------------------------------------------------------------------------------*/
print_report(siz, llim, ulim, bs, m, se, ms, tot)
int *siz,
double llim, ulim, bs, tot,
long m, se, ms,
{
double lim,
        lim = ulim - llim + 1;
        fprintf(out, "%3d %6 0f %11 0f %11 0f %11 0f %3ld %3ld %4ld %20 0f\n", siz, llim, ulim, lim,
bs, m, se, ms, tot);
}
```

```
/*--------------------------------------------------------------------------------------------------------------*/
/*                                                                                                              */
/*      File            node c  (Node program)                                                                  */
/*      Author          Intel Corporation, June 1990                                                            */
/*      Modified by     Imtiaz Ahmad, February 10, 1990                                                         */
/*                                                                                                              */
/*      This program sums the numbers within a given range with parallel processing,                            */
/*      and clocks the solution time                                                                            */
/*                                                                                                              */
/*      The user selects the number of processors and the number of points to be                                */
/*      summed, and put them in an input file  By selecting and timing different                                */
/*       cube sizes, a measure of the  speedup for completely perfectly parallel programs                       */
/*      can be obtained                                                                                         */
/*                                                                                                              */
/*      All nodes                                                                                               */
/*      1) Receive the message specifying the number of working nodes                                           */
/*      2) Receive the message containing the integration parameters                                            */
/*      3) Participate in the global sum operation (gdsum) which sums                                           */
/*              the partial integrals  Non-working nodes contribute a 0 value                                   */
/*                                                                                                              */
/*      Each working node calculates a partial integral                                                         */
/*                                                                                                              */
/*      Root node                                                                                               */
/*      1) Calculates elapsed execution time                                                                    */
/*      2) Sends the summation result and execution time back to host                                           */
/*                                                                                                              */
/*      MAIN LOGIC AND SOURCE CODE WAS COPIED FROM THE DIRECTORY                                                */
/*      /usr/ipsc/examples/c  directory                                                                         */
/*                                                                                                              */
/*      Called by       main()                  (external function)                                             */
/*      Calling         f()                     (internal function)                                             */
/*                                                                                                              */
/*      Message sending to      main()          (ext  func -- host c)                                           */
/*      Message receiving from  user_input()    (external function)                                             */
/*                                                                                                              */
/*--------------------------------------------------------------------------------------------------------------*/

#include <cube h>

#define  HOST_PID       100             /* process id of the host process */
#define  INIT_TYPE      0               /* type of initialization message */
#define  SIZE_TYPE      2               /* type of size message */
#define  PART_TYPE      10              /* type of partial sum message */
#define  ROOT           0               /* root node id */

int work_nodes,                         /* number of nodes which will work on problem */
   my_pid,                              /* process id of the nodes */
   my_node,                             /* node id of each node */

long m,                                 /* minimum number of slices to be given each node */
    extra_slices,                       /* remainder of the range after even distribution */
    starttime,                          /* start time of calculation */

double partial_sum,                     /* holds partial sum */
```

```
    work,
    my_a,                          /* local lower limit of summation range */
    my_b,                          /* local upper limit of summation range */

struct msg_type {                  /* structure for parameters of summation */
        double  a,                 /* lower limit of summation */
                b,                 /* upper limit of summation */
        long  points;              /* number of rounded points in the range */
},

struct msg_type sum,

main()
{       /* node main */

long f();
int j,
        my_pid = mypid(),          /* get process id */
        my_node = mynode(),        /* get node number*/

        for (,,) {                 /* Infinite loop */

                partial_sum = 0 0,

                /* receive message containing number of working * nodes  */
                crecv(SIZE_TYPE, &work_nodes, sizeof(work_nodes)),

                /* receive message containing the summation * parameters  */
                crecv(INIT_TYPE, &sum, sizeof(sum)),

                if (my_node < work_nodes) {    /* If I am a working node    */

                        starttime = mclock(),    /* Get initial clock value  */

                        /* calculate size of summation slice. */
                        m = f(sum points , work_nodes),
                        extra_slices = sum points - (m * work_nodes),

                        /* calculate lower and upper limits for each node */
                        my_a = sum a + m * my_node,
                        if (my_node = = (work_nodes -1))
                          my_b = my_a + m -1 + extra_slices;
                        else
                          my_b = my_a + m -1,

                        /* calculate partial sum on the sub-interval */
                        /* by using the formula  (b^2 - a^2 + b + a)  */

                        partial_sum = ((my_b * my_b)-(my_a * my_a) + my_b + my_a)/2,

                }       /* end if I am working node */

                gdsum(&partial_sum, 1, &work), /* Sum the partial-sum */
```

```
                /* If I am the root node, calculate the elapsed time and
                 * send the summed partial sum and the time to the host  */
                if (mynode() = = ROOT) {
                        sum a = partial_sum,
                        sum.b = m,
                        sum points = mclock() - starttime,
                        csend(PART_TYPE, &sum, sizeof(sum), myhost(), HOST_PID),
                }
        } /* End infinite loop */
} /* End node main */

/*-------------------------------------------------------------------------------------------------------------------*/
/*      This function calculates and returns the range of integers to be summed on      */
/*      each processor                                                                   */
/*                                                                                       */
/*      Called by       main()          (internal function)                              */
/*      Calling  None                                                                    */
/*-------------------------------------------------------------------------------------------------------------------*/
long f(x, y)
long x,
int y,
{
long z = 0,
        for (, y< =x, z+ +) x = x -y,

        if (z= =0) return (1),
        else return(z),
}
```

Example· Source code of the example given in Section 2 3

APPENDIX B

THE QUESTIONNAIRE AND THE EXPERTS' REPLIES

APPENDIX B.1: DETAILED QUESTIONNAIRE

MASTERS THESIS RESEARCH
QUESTIONNAIRE

Dear Participant

I am a Masters student at Oklahoma State University (OSU) and I am currently doing my thesis on "Software Metrics for Parallel Programs". Would you please take a few minutes to help me with my research by filling out and returning the following questionnaire Your participation is voluntary.

Thank you

SUBJECTIVE EVALUATION OF PARALLEL PROGRAMS
DEVELOPED ON THE iPSC FAMILY OF COMPUTERS

This questionnaire will take about 10 minutes to complete No detailed answers are required. After completing the questionnaire, kindly mail it to me (e-mail or US mail) Please try to fill out and return the questionnaire to me within one week Your help is extremely appreciated

Graduate Student Name: Imtiaz Ahmad
Office Address. 113 Math Sciences Building
University Computer Center
Oklahoma State University
Stillwater, OK 74078

Phone# Home (405) 744-2648
Office (405) 744-6701

E-mail Address: ahmad@d cs okstate.edu

This questionnaire is designed for experts who are well versed in parallel programming theoretically or have had sufficient hands-on experience in parallel programming, so that they can subjectively evaluate parallel programs developed for Intel's iPSC/1 or iPSC/2 concurrent computer. While answering the questions, please feel free to add any comments that you might have Also, if you do not wish to answer a question, please leave it blank or, if possible, contact me by telephone or through e-mail for clarification All questions could be answered by marking the given spaces (dashed lines) by any character such as "x"

The term UNDERSTANDABILITY, which is used in this questionnaire is defined below

Code possesses the characteristic understandability to the extent that its purpose is clear to the inspector This implies that variable names or symbols are used consistently, modules of code are self-descriptive, and the control structure is simple or in accordance with a prescribed standard

****

Questions 1 through 4 assess the expertise level of the person who is evaluating the applications (i e , programs) considered in this study

Q #1 Highest academic degree ----------------------------------

Q #2 Experience in computer programming

--------	--------	--------
less than 2 years	2-5 years	more than 5 years

Q #3: Experience (hardware and/or software) in parallel processing:

--------	--------	--------
less than 2 years	2-5 years	more than 5 years

Q #4 Experience (hardware and/or software) with Intel's iPSC family of computers (iPSC/1 and iPSC/2)

--------	--------	--------
less than 1 year	1-2 years	more than 2 years

<= =>

In questions 5 through 9 below please rate the 37 applications (or THE ONES THAT YOU ARE FAMILIAR WITH) used in this study. IF YOU ARE NOT FAMILIAR WITH AN APPLICATION, PLEASE LEAVE IT BLANK

All the applications reside under the "tex/programs" directory of the compendium except the last one which resides in the "tex/compendium" directory. This collection of application programs were made available to me by Dr Lamont of the Air Force Institute of Technology, Dayton, Ohio The directory paths shown below represent the application category, author(s) name, etc. The appropriate machine name is also given in front of each application's path

<= =>

Q #5 How would you rate the UNDERSTANDABILITY of the HOST program(s) of the following applications on a scale of 1 to 5?
Assume 1 indicates the poorest level and 5 the highest level of understandability

rings/beard	(iPSC/1)	1---	---	---	---	---5
rings/huson/c .	(iPSC/1)	1---	---	---	---	---5
rings/proicou	(iPSC/1)	1---	---	---	---	---5
meshs/beard .	(iPSC/1)	1---	---	---	---	---5
meshs/fife	(iPSC/1)	1---	---	---	---	---5
meshs/harding .	(iPSC/1)	1---	---	---	---	---5
meshs/huson	(iPSC/1)	1---	---	---	---	---5
meshs/proicou	(iPSC/1)	1---	---	---	---	---5
sorts/Beard	(iPSC/1)	1---	---	---	---	---5
sorts/Beard_Koch/cube386 .	(iPSC/2)	1---	---	---	---	---5
sorts/Beard_Koch/mem_cube	(iPSC/1)	1---	---	---	---	---5
sorts/Fife_Proicou/cube386	(iPSC/2)	1---	---	---	---	---5

sorts/Fife_Proicou/mem_cube	(iPSC/1)	1---	---	---	---	---5
sorts/Harding_Rottman/cube386	(iPSC/2)	1---	---	---	---	---5
sorts/Harding_Rottman/mem_cube	(iPSC/1)	1---	---	---	---	---5
sorts/Huson	(iPSC/1)	1---	---	---	---	---5
heaps/Beard_Koch/cube386	(iPSC/2)	1---	---	---	---	---5
heaps/Beard_Koch/mem_cube	(iPSC/1)	1---	---	---	---	---5
heaps/Fife_Proicou/cube386	(iPSC/2)	1---	---	---	---	---5
heaps/Fife_Proicou/mem_cube	(iPSC/1)	1---	---	---	---	---5
heaps/Harding_Rottman/cube386	(iPSC/2)	1---	---	---	---	---5
heaps/Harding_Rottman/mem_cube	(iPSC/1)	1---	---	---	---	---5
heaps/Huson	(iPSC/1)	1---	---	---	---	---5
projects/NeuralNets/conway	(iPSC/1)	1---	---	---	---	---5
projects/NeuralNets/simmers	(iPSC/1)	1---	---	---	---	---5
projects/TSP/rottman	(iPSC/1)	1---	---	---	---	---5
projects/TSP/sawyer	(iPSC/1)	1---	---	---	---	---5
projects/beard/src/Thesis/parallel	(iPSC/2)	1---	---	---	---	---5
projects/beard/src/Thesis/serial	(iPSC/2)	1---	---	---	---	---5
projects/beard/src/Version1	(iPSC/2)	1---	---	---	---	---5
projects/fife/src	(iPSC/1)	1---	---	---	---	---5
projects/harding	(iPSC/1)	1---	---	---	---	---5
projects/huson/src/new_stuff	(iPSC/1)	1---	---	---	---	---5
projects/koch/src	(iPSC/1)	1---	---	---	---	---5
projects/proicou/dinephil1	(iPSC/1)	1---	---	---	---	---5
projects/proicou/dinephil2	(iPSC/1)	1---	---	---	---	---5
tex/compendium/projects/SCPArchive	(iPSC/2)	1---	---	---	---	---5

<= =>

Q #6 How would you rate the UNDERSTANDABILITY of the NODE program(s) of the following applications on a scale of 1 to 5?
Assume 1 indicates the poorest level and 5 the highest level of understandability

rings/beard	(iPSC/1)	1---	---	---	---	---5
rings/huson/c	(iPSC/1)	1---	---	---	---	---5
rings/proicou	(iPSC/1)	1---	---	---	---	---5
meshs/beard	(iPSC/1)	1---	---	---	---	---5
meshs/fife	(iPSC/1)	1---	---	---	---	---5
meshs/harding	(iPSC/1)	1---	---	---	---	---5
meshs/huson	(iPSC/1)	1---	---	---	---	---5
meshs/proicou	(iPSC/1)	1---	---	---	---	---5
sorts/Beard	(iPSC/1)	1---	---	---	---	---5
sorts/Beard_Koch/cube386	(iPSC/2)	1---	---	---	---	---5
sorts/Beard_Koch/mem_cube	(iPSC/1)	1---	---	---	---	---5
sorts/Fife_Proicou/cube386	(iPSC/2)	1---	---	---	---	---5
sorts/Fife_Proicou/mem_cube	(iPSC/1)	1---	---	---	---	---5
sorts/Harding_Rottman/cube386	(iPSC/2)	1---	---	---	---	---5
sorts/Harding_Rottman/mem_cube	(iPSC/1)	1---	---	---	---	---5
sorts/Huson	(iPSC/1)	1---	---	---	---	---5
heaps/Beard_Koch/cube386	(iPSC/2)	1---	---	---	---	---5
heaps/Beard_Koch/mem_cube	(iPSC/1)	1---	---	---	---	---5
heaps/Fife_Proicou/cube386	(iPSC/2)	1---	---	---	---	---5
heaps/Fife_Proicou/mem_cube	(iPSC/1)	1---	---	---	---	---5
heaps/Harding_Rottman/cube386	(iPSC/2)	1---	---	---	---	---5

heaps/Harding_Rottman/mem_cube	(iPSC/1)	1---	---	---	---	---5
heaps/Huson	(iPSC/1)	1---	---	---	---	---5
projects/NeuralNets/conway	(iPSC/1)	1---	---	---	---	---5
projects/NeuralNets/simmers	(iPSC/1)	1---	---	---	---	---5
projects/TSP/rottman	(iPSC/1)	1---	---	---	---	---5
projects/TSP/sawyer	(iPSC/1)	1---	---	---	---	---5
projects/beard/src/Thesis/parallel	(iPSC/2)	1---	---	---	---	---5
projects/beard/src/Thesis/serial	(iPSC/2)	1---	---	---	---	---5
projects/beard/src/Version1	(iPSC/2)	1---	---	---	---	---5
projects/fife/src	(iPSC/1)	1---	---	---	---	---5
projects/harding	(iPSC/1)	1---	---	---	---	---5
projects/huson/src/new_stuff	(iPSC/1)	1---	---	---	---	---5
projects/koch/src	(iPSC/1)	1---	---	---	---	---5
projects/proicou/dinephil1	(iPSC/1)	1---	---	---	---	---5
projects/proicou/dinephil2	(iPSC/1)	1---	---	---	---	---5
tex/compendium/projects/SCPArchive	(iPSC/2)	1---	---	---	---	---5

<===>

Q #7: How would you rate the documentation of the HOST program(s) of the following applications on a scale of 1 to 5?
Assume 1 indicates the poorest level and 5 the best and most informative level of documentation.

rings/beard	(iPSC/1)	1---	---	---	---	---5
rings/huson/c	(iPSC/1)	1---	---	---	---	---5
rings/proicou	(iPSC/1)	1---	---	---	---	---5
meshs/beard	(iPSC/1)	1---	---	---	---	---5
meshs/fife	(iPSC/1)	1---	---	---	---	---5
meshs/harding	(iPSC/1)	1---	---	---	---	---5
meshs/huson	(iPSC/1)	1---	---	---	---	---5
meshs/proicou	(iPSC/1)	1---	---	---	---	---5
sorts/Beard	(iPSC/1)	1---	---	---	---	---5
sorts/Beard_Koch/cube386	(iPSC/2)	1---	---	---	---	---5
sorts/Beard_Koch/mem_cube	(iPSC/1)	1---	---	---	---	---5
sorts/Fife_Proicou/cube386	(iPSC/2)	1---	---	---	---	---5
sorts/Fife_Proicou/mem_cube	(iPSC/1)	1---	---	---	---	---5
sorts/Harding_Rottman/cube386	(iPSC/2)	1---	---	---	---	---5
sorts/Harding_Rottman/mem_cube	(iPSC/1)	1---	---	---	---	---5
sorts/Huson	(iPSC/1)	1---	---	---	---	---5
heaps/Beard_Koch/cube386	(iPSC/2)	1---	---	---	---	---5
heaps/Beard_Koch/mem_cube	(iPSC/1)	1---	---	---	---	---5
heaps/Fife_Proicou/cube386	(iPSC/2)	1---	---	---	---	---5
heaps/Fife_Proicou/mem_cube	(iPSC/1)	1---	---	---	---	---5
heaps/Harding_Rottman/cube386	(iPSC/2)	1---	---	---	---	---5
heaps/Harding_Rottman/mem_cube	(iPSC/1)	1---	---	---	---	---5
heaps/Huson	(iPSC/1)	1---	---	---	---	---5
projects/NeuralNets/conway	(iPSC/1)	1---	---	---	---	---5
projects/NeuralNets/simmers	(iPSC/1)	1---	---	---	---	---5
projects/TSP/rottman	(iPSC/1)	1---	---	---	---	---5
projects/TSP/sawyer	(iPSC/1)	1---	---	---	---	---5
projects/beard/src/Thesis/parallel	(iPSC/2)	1---	---	---	---	---5
projects/beard/src/Thesis/serial	(iPSC/2)	1---	---	---	---	---5

projects/beard/src/Version1	(iPSC/2)	1---	---	---	---	---5
projects/fife/src	(iPSC/1)	1---	---	---	---	---5
projects/harding	(iPSC/1)	1---	---	---	---	---5
projects/huson/src/new_stuff	(iPSC/1)	1---	---	---	---	---5
projects/koch/src	(iPSC/1)	1---	---	---	---	---5
projects/proicou/dinephil1	(iPSC/1)	1---	---	---	---	---5
projects/proicou/dinephil2	(iPSC/1)	1---	---	---	---	---5
tex/compendium/projects/SCPArchive	(iPSC/2)	1---	---	---	---	---5

<==>

Q #8: How would you rate the documentation of the NODE program(s) of the following applications on a scale of 1 to 5?
Assume 1 indicates the poorest level and 5 the best and most informative level of documentation

rings/beard	(iPSC/1)	1---	---	---	---	---5
rings/huson/c	(iPSC/1)	1---	---	---	---	---5
rings/proicou	(iPSC/1)	1---	---	---	---	---5
meshs/beard	(iPSC/1)	1---	---	---	---	---5
meshs/fife	(iPSC/1)	1---	---	---	---	---5
meshs/harding	(iPSC/1)	1---	---	---	---	---5
meshs/huson	(iPSC/1)	1---	---	---	---	---5
meshs/proicou	(iPSC/1)	1---	---	---	---	---5
sorts/Beard	(iPSC/1)	1---	---	---	---	---5
sorts/Beard_Koch/cube386	(iPSC/2)	1---	---	---	---	---5
sorts/Beard_Koch/mem_cube	(iPSC/1)	1---	---	---	---	---5
sorts/Fife_Proicou/cube386	(iPSC/2)	1---	---	---	---	---5
sorts/Fife_Proicou/mem_cube	(iPSC/1)	1---	---	---	---	---5
sorts/Harding_Rottman/cube386	(iPSC/2)	1---	---	---	---	---5
sorts/Harding_Rottman/mem_cube	(iPSC/1)	1---	---	---	---	---5
sorts/Huson	(iPSC/1)	1---	---	---	---	---5
heaps/Beard_Koch/cube386	(iPSC/2)	1---	---	---	---	---5
heaps/Beard_Koch/mem_cube	(iPSC/1)	1---	---	---	---	---5
heaps/Fife_Proicou/cube386	(iPSC/2)	1---	---	---	---	---5
heaps/Fife_Proicou/mem_cube	(iPSC/1)	1---	---	---	---	---5
heaps/Harding_Rottman/cube386	(iPSC/2)	1---	---	---	---	---5
heaps/Harding_Rottman/mem_cube	(iPSC/1)	1---	---	---	---	---5
heaps/Huson	(iPSC/1)	1---	---	---	---	---5
projects/NeuralNets/conway	(iPSC/1)	1---	---	---	---	---5
projects/NeuralNets/simmers	(iPSC/1)	1---	---	---	---	---5
projects/TSP/rottman	(iPSC/1)	1---	---	---	---	---5
projects/TSP/sawyer	(iPSC/1)	1---	---	---	---	---5
projects/beard/src/Thesis/parallel	(iPSC/2)	1---	---	---	---	---5
projects/beard/src/Thesis/serial	(iPSC/2)	1---	---	---	---	---5
projects/beard/src/Version1	(iPSC/2)	1---	---	---	---	---5
projects/fife/src	(iPSC/1)	1---	---	---	---	---5
projects/harding	(iPSC/1)	1---	---	---	---	---5
projects/huson/src/new_stuff	(iPSC/1)	1---	---	---	---	---5
projects/koch/src	(iPSC/1)	1---	---	---	---	---5
projects/proicou/dinephil1	(iPSC/1)	1---	---	---	---	---5
projects/proicou/dinephil2	(iPSC/1)	1---	---	---	---	---5
tex/compendium/projects/SCPArchive	(iPSC/2)	1---	---	---	---	---5

<==========>

Q #9: How would you rate the overall perceived or conceptual COMPLEXITY (different frorn computational complexity) of the following applications on a scale of 1 to 5? Assume 1 indicates a lowest level and 5 the highest level of complexity

rings/beard	(iPSC/1)	1---	---	---	---	---5
rings/huson/c	(iPSC/1)	1---	---	---	---	---5
rings/proicou	(iPSC/1)	1---	---	---	---	---5
meshs/beard	(iPSC/1)	1---	---	---	---	---5
meshs/fife	(iPSC/1)	1---	---	---	---	---5
meshs/harding .	(iPSC/1)	1---	---	---	---	---5
meshs/huson	(iPSC/1)	1---	---	---	---	---5
meshs/proicou .	(iPSC/1)	1---	---	---	---	---5
sorts/Beard	(iPSC/1)	1---	---	---	---	---5
sorts/Beard_Koch/cube386	(iPSC/2)	1---	---	---	---	---5
sorts/Beard_Koch/mem_cube	(iPSC/1)	1---	---	---	---	---5
sorts/Fife_Proicou/cube386	(iPSC/2)	1---	---	---	---	---5
sorts/Fife_Proicou/mem_cube	(iPSC/1)	1---	---	---	---	---5
sorts/Harding_Rottman/cube386	(iPSC/2)	1---	---	---	---	---5
sorts/Harding_Rottman/mem_cube	(iPSC/1)	1---	---	---	---	---5
sorts/Huson	(iPSC/1)	1---	---	---	---	---5
heaps/Beard_Koch/cube386	(iPSC/2)	1---	---	---	---	---5
heaps/Beard_Koch/mem_cube	(iPSC/1)	1---	---	---	---	---5
heaps/Fife_Proicou/cube386	(iPSC/2)	1---	---	---	---	---5
heaps/Fife_Proicou/mem_cube .	(iPSC/1)	1---	---	---	---	---5
heaps/Harding_Rottman/cube386	(iPSC/2)	1---	---	---	---	---5
heaps/Harding_Rottman/mem_cube	(iPSC/1)	1---	---	---	---	---5
heaps/Huson	(iPSC/1)	1---	---	---	---	---5
projects/NeuralNets/conway	(iPSC/1)	1---	---	---	---	---5
projects/NeuralNets/simmers	(iPSC/1)	1---	---	---	---	---5
projects/TSP/rottman	(iPSC/1)	1---	---	---	---	---5
projects/TSP/sawyer	(iPSC/1)	1---	---	---	---	---5
projects/beard/src/Thesis/parallel	(iPSC/2)	1---	---	---	---	---5
projects/beard/src/Thesis/serial	(iPSC/2)	1---	---	---	---	---5
projects/beard/src/Version1 .	(iPSC/2)	1---	---	---	---	---5
projects/fife/src	(iPSC/1)	1---	---	---	---	---5
projects/harding . .	(iPSC/1)	1---	---	---	---	---5
projects/huson/src/new_stuff	(iPSC/1)	1---	---	---	---	---5
projects/koch/src . . .	(iPSC/1)	1---	---	---	---	---5
projects/proicou/dinephil1	(iPSC/1)	1---	---	---	---	---5
projects/proicou/dinephil2	(iPSC/1)	1---	---	---	---	---5
tex/compendium/projects/SCPArchive	(iPSC/2)	1---	---	---	---	---5

<==========>

Q #10· If the following applications had been developed as sequential programs, do you think they would have taken less/more/same amount of time and effort?

		Less	More	Same	Don't Know
rings/beard	(iPSC/1)	---	---	---	---
rings/huson/c .	(iPSC/1)	---	---	---	---
rings/proicou	(iPSC/1)	---	---	---	---
meshs/beard	(iPSC/1)	---	---	---	---
meshs/fife	(iPSC/1)	---	---	---	---
meshs/harding	(iPSC/1)	---	---	---	---
meshs/huson	(iPSC/1)	---	---	---	---
meshs/proicou	(iPSC/1)	---	---	---	---
sorts/Beard	(iPSC/1)	---	---	---	---
sorts/Beard_Koch/cube386	(iPSC/2)	---	---	---	---
sorts/Beard_Koch/mem_cube	(iPSC/1)	---	---	---	---
sorts/Fife_Proicou/cube386	(iPSC/2)	---	---	---	---
sorts/Fife_Proicou/mem_cube	(iPSC/1)	---	---	---	---
sorts/Harding_Rottman/cube386	(iPSC/2)	---	---	---	---
sorts/Harding_Rottman/mem_cube	(iPSC/1)	---	---	---	---
sorts/Huson	(iPSC/1)	---	---	---	---
heaps/Beard_Koch/cube386	(iPSC/2)	---	---	---	---
heaps/Beard_Koch/mem_cube	(iPSC/1)	---	---	---	---
heaps/Fife_Proicou/cube386	(iPSC/2)	---	---	---	---
heaps/Fife_Proicou/mem_cube	(iPSC/1)	---	---	---	---
heaps/Harding_Rottman/cube386	(iPSC/2)	---	---	---	---
heaps/Harding_Rottman/mem_cube	(iPSC/1)	---	---	---	---
heaps/Huson	(iPSC/1)	---	---	---	---
projects/NeuralNets/conway .	(iPSC/1)	---	---	---	---
projects/NeuralNets/simmers	(iPSC/1)	---	---	---	---
projects/TSP/rottman	(iPSC/1)	---	---	---	---
projects/TSP/sawyer	(iPSC/1)	---	---	---	---
projects/beard/src/Thesis/parallel	(iPSC/2)	---	---	---	---
projects/beard/src/Thesis/serial	(iPSC/2)	---	---	---	---
projects/beard/src/Version1	(iPSC/2)	---	---	---	---
projects/fife/src	(iPSC/1)	---	---	---	---
projects/harding	(iPSC/1)	---	---	---	---
projects/huson/src/new_stuff	(iPSC/1)	---	---	---	---
projects/koch/src .	(iPSC/1)	---	---	---	---
projects/proicou/dinephil1	(iPSC/1)	---	---	---	---
projects/proicou/dinephil2	(iPSC/1)	---	---	---	---
tex/compendium/projects/SCPArchive	(iPSC/2)	---	---	---	---

< = End of Questionnaire = = = = = = = = = = = = = = = = = = = >

APPENDIX B.2: APPLICATION NAMES AND ACCUMULATED TOTAL NUMBER OF EXPERTS' REPLIES TO THE QUESTIONS IN THE QUESTIONNAIRE.

Apl#	APLNAME	Q5	Q6	Q7	Q8	Q9	Q10
1	rings/beard	14	15	16	16	5	8
2	meshs/beard	14	15	16	16	5	8
3	sorts/Beard	14	14	15	15	10	4
4	sorts/Beard_Koch/cube386	14	14	15	15	10	4
5	sorts/Beard_Koch/mem_cube	14	14	15	15	10	4
6	sorts/Fife_Proicou/cube386	14	14	15	15	10	4
7	sorts/Fife_Proicou/mem_cube	14	14	15	15	10	4
8	heaps/Beard_Koch/cube386	14	14	15	15	10	4
9	heaps/Beard_Koch/mem_cube	14	14	15	15	10	4
10	heaps/Fife_Proicou/cube386	14	14	15	15	10	4
11	heaps/Fife_Proicou/mem_cube	14	14	15	15	10	4
12	projects/NeuralNets/conway	10	10	12	12	15	3
13	projects/NeuralNets/simmers	10	10	12	12	15	3
14	projects/TSP/rottman	11	12	14	13	13	4
15	projects/TSP/sawyer	13	14	15	14	13	4
16	projects/harding	12	11	13	12	14	3
17	projects/huson/src/new_stuff	12	11	13	12	16	3
18	projects/koch/src	12	11	14	13	10	4

"Q5" to "Q10" represents the Question 5 through Question 10 in the Questionnaire (Appendix B 1)

APPENDIX C

PC-METRIC REPORTS AND THE LISTING OF RESERVED AND NON-EXECUTABLE WORDS

APPENDIX C.1: SAMPLE PC-METRIC REPORT BY PROCEDURE

```
10/22/1990
Page:  1
PC-METRIC (C) Version 2.4
Complexity Report by Procedure for: C:\SAMPLE.C
-----------------------------------------------
```

Procedure	n1	n2	N1	N2	N	N^	P/R	V	E
main	30	19	106	44	150	228	1.5	842	29256
get_tok	26	32	122	66	188	282	1.5	1101	29529
token_type	9	13	76	33	109	77	0.7	486	5553
print_stable	11	8	26	15	41	62	1.5	174	1796
print_ctable	13	8	27	16	43	72	1.7	189	2455

VG1	VG2	LOC	<;>	SP
6	6	59	19	8
15	15	76	26	7
9	9	22	9	3
3	3	10	6	1
3	3	13	7	1

Figure C.2

APPENDIX C.2: SAMPLE PC-METRIC REPORT BY COMPLEXITY

```
10/22/1990
PC-METRIC (C) Version 2.4
Summary Complexity Report for: C:\SAMPLE.RPT
-------------------------------------------

Unique Operators (n1):       39
Unique Operands (n2):        64
Total Operators (N1):       357
Total Operands (N2):        174

Software Science Length (N):                    531
Estimated Software Science Length (N^):         590
Purity Ratio (P/R):                            1.11

Software Science Volume (V):          3551
Software Science Effort (E):        188234

Estimated Errors using Software Science (B^):      1
Estimated Time to Develop, in hours (T^):          3


Cyclomatic Complexity (VG1):                   32
Extended Cyclomatic Complexity (VG2):          32
Average Cyclomatic Complexity:                  6
Average Extended Cyclomatic Complexity:         6

Lines of Code (LOC):                          282
Number of Procedures/Functions:                 5
Number of Executable Semi-colons (<;>):        67
```

Figure C.3

APPENDIX C.3: iPSC/2-C RESERVED AND NON-EXECUTABLE WORDS

RESERVED WORDS

!	!=	"	%	%=	&
&&	&=	&p	'	(	(c
(p	)	*	*=	*p	+
++	+=	,	-	--	-=
->	.	/	/=	:	;
<	<<	<<=	<=	=	==
>	>=	>>	>>=	?	[
]	^	^=	break	case	continue
default	do	else	entry	for	goto
if	return	sizeof	switch	while	{
|	|=	||	}	~	

NON-EXECUTABLE WORDS

auto	char	const	double	enum	extern
FILE	float	int	long	register	short
signedstatic	struct	typedef	union	unsigned	void
volatile					

APPENDIX D

PARALLEL PROGRAM TO COLLECT SIZE MEASUREMENTS

```
/*-------------------------------------------------------------------------------------------------------------*/
/*                                                                                                             */
/*      File            host c   (Host program)                                                                */
/*                                                                                                             */
/*      Author.         Imtiaz Ahmad, November 1990                                                            */
/*                                                                                                             */
/*      Purpose         To collect Size metrics                                                                */
/*                                                                                                             */
/*      Description     Counts the number of lines of code in a C source file                                  */
/*                      Generates a report with number of executable lines, number                             */
/*                      number of blank lines, number of comment lines, and number                             */
/*                      of total lines in a source file                                                        */
/*                                                                                                             */
/*      Caution         Program assumes that the input file is syntax error free                               */
/*                                                                                                             */
/*-------------------------------------------------------------------------------------------------------------*/

#include <stdio.h>
#include <cube.h>

#define TOTNODES    32          /* total node - must be < or = to alloc  nodes */
#define HOST_PID    100         /* process id of the host process */
#define NODE_PID    0           /* process id for node processes */
#define INIT_TYPE   0           /* type of initialization message */
#define RSLT_TYPE   10          /* type of partial summation message */
#define ALL_NODES   -1          /* symbol for all nodes */
#define ALL_PIDS    -1          /* symbol for all processes */
#define MAX_FILES   200         /* max  files that can be evaluated */
#define PATHLEN     81          /* max  characters in a file path */

struct stat {                   /* saves frequency of different type of LOCs */
        int n_bl_lines,
          n_com_lines,
          loc,
},

struct info {                   /* structure used to send and recv messages */
        int nodenum,
        int pathnum,
        char filepath[PATHLEN],
        struct stat LOC,
},

struct info initinfo,
struct info initinfo1[MAX_FILES],       /* store collected metric */

char in_file[PATHLEN],
char f_name[PATHLEN],
FILE *fp,

/*-------------------------------------------------------------------------------------------------------------*/
/*      main()                                                                                                 */
/*                                                                                                             */
/*      Logic           Allocate cube, load node programs to all nodes  Send a packet                          */
```

```
/*                      to each node with file for which metrics needs to be collected      */
/*                      Receives a packet from each node with computed metrics and          */
/*                      stores it in an array of structures for later printing              */
/*                                                                                          */
/*      Caution.        Make sure that full pathname of a file has been passed to nodes     */
/*------------------------------------------------------------------------------------------*/
main()
{       /* Host main */

        int i, j, k,                    /* temporary variables */
        fileread,                       /* number of files read from input file */
        filecomp,                       /* number of files received by host after
                                                collecting the metric */
        node_avail,                     /* node number to process next data file */

        getcube ("", "32", "", 0),      /* getcube with given number of nodes */
        setpid(HOST_PID);               /* set the pid of host process */
        load ("node", ALL_NODES, NODE_PID),  /* load nodes with node progs */

        open_file(),

        filecomp = fileread = node_avail = 0,

        /* reads file names from user's given input file until end of file */
        while(fgets(f_name, PATHLEN, fp) != NULL)
        {
                fileread++,

                /* initialize structure */
                init_msg(f_name, strlen(f_name), node_avail, fileread),

                /* This if statment will be true for the first n file paths, where n is the      */
                /* number of nodes available in the cube through TOTNODES variable */
                if (fileread <= TOTNODES) {

                        csend(INIT_TYPE, &initinfo, sizeof(initinfo), node_avail, NODE_PID),
                        node_avail++,
                }
                else {
                        crecv (RSLT_TYPE, &initinfo, sizeof(initinfo)),

                        filecomp++,
                        store_result(initinfo pathnum),
                        node_avail = initinfo nodenum,
                        init_msg(f_name, strlen(f_name), node_avail, fileread),

                        csend(INIT_TYPE, &initinfo, sizeof(initinfo), node_avail, NODE_PID),
                }

        } /*while fgets*/

        /* This if statment checks whether all activated files are received by the host or node */
        if (fileread != filecomp) {
                for (j=filecomp, j< fileread, j++) {
```

```
                        crecv (RSLT_TYPE, &initinfo, sizeof(initinfo)),

                        /* store result in an array of stuctures */
                        store_result(initinfo pathnum),
                }
        }

        /* print results on screen */
        print_result(fileread),

        fclose(fp),                             /* close input file */
        killcube(ALL_NODES, ALL_PIDS),          /* kill cube */
        relcube(),                              /* release cube */
        printf("Normal termination of the program \n"),
} /* End host main */


/*-------------------------------------------------------------------------------------------------------------------*/
/*      open_file()                                                                                                  */
/*                                                                                                                   */
/*      Purpose         Prompts for user input  User must enter the file name that                                   */
/*                      contains the complete pathname of the files for which metrics                                */
/*                      needs to be calculated                                                                       */
/*-------------------------------------------------------------------------------------------------------------------*/
open_file()
{
        printf("Enter input file name  \n\n"),
        gets(in_file),

        if ((fp=fopen(in_file, "r")) == NULL)
        {
                printf("Can not open file containing path names in host  \n"),
                exit(0),
        }

        printf("                                                Blnk  Com  Tot\n"),
        printf("Input File Name (with complete path)                    LOC  Lns  Lns  Lns\n"),
        printf("-------------------------------------------------------------------------\n"),
}

/*-------------------------------------------------------------------------------------------------------------------*/
/*      init_msg()                                                                                                   */
/*                                                                                                                   */
/*      Purpose         Initialize the structure before sending it to a node                                         */
/*-------------------------------------------------------------------------------------------------------------------*/
init_msg(fn, i, n, f)
char fn[],
int i, n, f,
{
        int j,
        for(j=0,j<80,j++)
          initinfo filepath[j] = '\0',
```

```
        initinfo nodenum = n,                   /* node number to send */
        initinfo pathnum = f,                   /* file number read */
        strncpy(initinfo filepath, fn, i-1),    /* i=length of file name */
        initinfo.LOC n_bl_lines = -1,           /* init LOC vars*/
        initinfo.LOC.n_com_lines = -1,
        initinfo LOC.loc = -1,
}

/*-------------------------------------------------------------------------------------------------------------------*/
/*      store_result()                                                                                               */
/*                                                                                                                   */
/*      Purpose         To store computed metric in an array at subscript i                                          */
/*-------------------------------------------------------------------------------------------------------------------*/
store_result(i)
int i,
{
        initinfo1[i] nodenum = initinfo nodenum,
        initinfo1[i] pathnum = initinfo pathnum,
        strcpy(initinfo1[i] filepath, initinfo filepath),
        initinfo1[i] LOC n_bl_lines = initinfo LOC n_bl_lines,
        initinfo1[i] LOC n_com_lines = initinfo LOC n_com_lines,
        initinfo1[i] LOC loc = initinfo LOC loc,
}

/*-------------------------------------------------------------------------------------------------------------------*/
/*      print_result()                                                                                               */
/*                                                                                                                   */
/*      Purpose         To print the resultant array                                                                 */
/*-------------------------------------------------------------------------------------------------------------------*/
print_result(f)
int f,
{
        int i,

        for (i=0,i< f, i++)     {
                printf("%-58s %4d %4d %4d %4d\n\n", initinfo1[i+1] filepath,
initinfo1[i+1] LOC.n_bl_lines, initinfo1[i+1] LOC n_com_lines, initinfo1[i+1] LOC loc,
initinfo1[i+1] LOC.n_bl_lines + initinfo1[i+1] LOC n_com_lines + initinfo1[i+1] LOC.loc),
        }
}

/*---------------------------------------- End of host c Module ----------------------------------------*/
```

```
/*-------------------------------------------------------------------------------------------------------------*/
/*                                                                                                             */
/*      File:           node.c  (Node program)                                                                 */
/*                                                                                                             */
/*      Author:         Imtiaz Ahmad, November 1990                                                            */
/*                                                                                                             */
/*      Purpose         To collect Size metrics                                                                */
/*                                                                                                             */
/*      Description     Counts the number of lines of code in a C source file                                  */
/*                      Sends collected metrics to host for final printing                                     */
/*                                                                                                             */
/*      Caution         Program assumes that the input file is syntax error free                               */
/*                                                                                                             */
/*-------------------------------------------------------------------------------------------------------------*/

#include <stdio.h>
#include <cube.h>

#define TRUE            1               /* assigning symbolic names to program constants */
#define FALSE           0
#define MAXLINE         150
#define BLANK           ' '
#define SLASH           '/'
#define STAR            '*'

#define TOTNODES        32              /* total nodes - must be < or = to alloc  nodes */
#define HOST_PID        100             /* process id of the host process */
#define NODE_PID        0               /* process id for node processes */
#define INIT_TYPE       0               /* type of initialization message */
#define RSLT_TYPE       10              /* type of partial summation message */
#define ALL_NODES       -1              /* symbol for all nodes */
#define ALL_PIDS        -1              /* symbol for all processes */
#define MAX_FILES       200             /* max  files that can be evaluated */
#define PATHLEN         81              /* max  characters in a file path */

struct stat {                           /* saves frequency of LOCs */
        int n_bl_lines,
          n_com_lines,
          loc,
},

struct info {                           /* structure used to send and recv messages */
        int nodenum,
        int pathnum,
        char filepath[PATHLEN],
        struct stat LOC,
},

struct stat temp,
struct info initinfo,
struct info initinfo1[MAX_FILES],       /* store computed metric */

char in_file[PATHLEN],
char f_name[PATHLEN],
```

```
FILE *fp,

/*-------------------------------------------------------------------------------------------------------------------*/
/*      main()                                                                                                       */
/*                                                                                                                   */
/*      Logic           Runs an infinite loop . Receives a message from the host with file                           */
/*                      name to be processed  Returns a structure to the host with                                   */
/*                      collected metrics.                                                                           */
/*                                                                                                                   */
/*      Caution         Program assumes that there is no syntax error in the input file                              */
/*-------------------------------------------------------------------------------------------------------------------*/
main()
{       /* Node main */

        char line[MAXLINE],             /* buffer to hold a single line */
        char *temp,                     /* temporary pointer */
        FILE *fp,                               /* pointer to input file */
        char in_file[PATHLEN],          /* input file name */
        char f_name[PATHLEN],           /* file name with complete path */
        int in_fd,                      /* input file descriptor */
        int i,

        for (,,) {                      /* infinite loop */
                                        /* wait to receive a message from host */
                crecv(INIT_TYPE, &initinfo, sizeof(initinfo)),

                strcpy(in_file, initinfo filepath),

                in_fd = open(in_file,0),

                if (in_fd < = 0) {
                        close(in_fd),

                        /* sends a message to host without collecting any measure */
                        csend(RSLT_TYPE, &initinfo, sizeof(initinfo), myhost(), HOST_PID),
                } /* end if */
                else {
                        temp n_bl_lines = 0,
                        temp n_com_lines = 0,
                        temp loc = 0,

                        while( readline(in_fd, line) ) {

                                temp = line,

                                while( *temp = = BLANK )
                                        temp + +,

                                if ( *temp ! = '\0' ) {

                                        if ( prec_com_match( temp ) )
                                                find_end_comment( in_fd, temp),
                                        else
```

```
                                        temp loc++,
                                } /* end if */
                                else    {
                                        temp n_bl_lines++,
                                } /* end else */

                        } /* End WHILE GETLINE */

                        initinfo nodenum = mynode(),
                        initinfo LOC n_bl_lines = temp n_bl_lines,
                        initinfo LOC n_com_lines = temp n_com_lines,
                        initinfo LOC loc = temp loc,

                        csend(RSLT_TYPE, &initinfo, sizeof(initinfo),myhost(), HOST_PID),

                        close(in_fd),
                } /*end else */
        } /* end infinite loop */
} /* end Node main */


/*-------------------------------------------------------------------------------------------------------------------*/
/*      readline()                                                                                                   */
/*                                                                                                                   */
/*      Description  To read the next line from the input file C source file                                         */
/*-------------------------------------------------------------------------------------------------------------------*/
readline(fd, buffer)
int fd,

char *buffer,
{
        int i, end_of_line, rd_flag,
        char c,
        static j,

        i = 0,
        end_of_line = FALSE,
        rd_flag = read(fd,&c,1),

        if ( rd_flag != 1 ){
                return(FALSE),
        }

        if ( c == '\n' )   {
                buffer[i] = '\0',

                return(TRUE),
        }

        while( rd_flag == 1 && !end_of_line ) {
                if ( c == '\n')    {
                        end_of_line = TRUE,
                        j++,
```

```
                }
                else
                        buffer[i++] = c,

                if ( !end_of_line )          {
                        rd_flag = read(fd,&c,1),
                }
        } /* end while */

        buffer[i] = '\0',
        return(TRUE),
}

/*-------------------------------------------------------------------------------------------------------------------*/
/*      prec_com_match()                                                                                             */
/*      Description     To check whether the current line has any comment lines                                      */
/*                      beginning in it                                                                              */
/*-------------------------------------------------------------------------------------------------------------------*/
prec_com_match( line )
char *line,
{
        int matched,

        matched = FALSE,

        while ( *line != '\0' && !matched )          {

                if ( *line == SLASH && *(line+1) == STAR )
                        matched = TRUE,
                        line++,
                } /* end if */
        return(matched),
}

/*-------------------------------------------------------------------------------------------------------------------*/
/*      find_end_comment()                                                                                           */
/*      Description     To find and stop at the position in the file where the                                        */
/*                       current comment ends                                                                        */
/*-------------------------------------------------------------------------------------------------------------------*/
find_end_comment(fd,line)
int fd,
char *line,
{
        char c,
        int matched,
        int rd_flag;
        int end_of_line,
        int code_line,              /* turn flag on if line has code also */

        matched = FALSE,
        code_line = FALSE,


        if ( !open_comment ( line ) )       {
```

```
        temp loc++,
        code_line = TRUE,
}

while ( *line != '\0' )     {

        if ( *line == STAR && *(line+1) == SLASH )
                matched = TRUE,

        line++,
} /* end while */

if ( matched )    {
        if ( !code_line )  {
                temp n_com_lines++,
        }
return,
}

if (!matched && !code_line)       {
        temp n_com_lines++,
}

rd_flag = read(fd,&c,1),

end_of_line = FALSE,

while( rd_flag == 1 && !matched )        {

        if ( c == '\n' )   {
                temp n_com_lines++,
                end_of_line = TRUE,
        } /* end if */

if ( end_of_line ) {

        do      {
                rd_flag = read( fd, &c,1),
        } while ( rd_flag == 1 && ( c == ' ' || c == '\t' ) ),

        if ( rd_flag == 1 && c == '\n' )  {
                temp n_bl_lines++,
        }

        end_of_line = FALSE,
} /* end if end-of-line */

        if ( c == STAR ) {
                rd_flag = read( fd, &c,1),
                if ( rd_flag == 1 && c == SLASH )       {
                        matched = TRUE,
                        rd_flag = read( fd, &c,1),
                        rd_flag = read( fd, &c,-2),
                }
```

```
                        if ( c == '\n' && !matched)       {
                                temp n_com_lines++,
                        }
                } /* end if c==STAR */

                if ( !matched )   {
                        rd_flag = read(fd,&c,1),
                }

        } /* end while */

        temp n_com_lines++,
}

/*-------------------------------------------------------------------------------------------------------------------*/
/*      open_comment()                                                                                               */
/*                                                                                                                   */
/*      Description      To check if the current line has a begin comments                                           */
/*-------------------------------------------------------------------------------------------------------------------*/
open_comment(line)
char *line,
{
        if ( *line == SLASH )
                if ( *(line+1) == STAR )
                        return(TRUE),

        return(FALSE),
}


/*---------------------------------------- End of node c Module ----------------------------------------*/
```

APPENDIX E

PSEUDO CODE, FILE LISTING, AND MAKEFILE

APPENDIX E.1: PSEUDO CODE FOR THE PARALLEL PROGRAMS DEVELOPED TO COLLECT THE SIZE AND THE COMMUNICATION MEASUREMENTS.

```
h[1]1:  (Host) Getcube with n number of nodes (n=1,2,...32);

h2:     (Host) Load node program to each node in the allocated cube;

h3:     (Host) Prompt for input file that has data file names with their complete
               paths;

h4:     (Host) Initialize flags and counters;

h5:     (Host) Initialize filesread = filescompleted = 0;

h6:     (Host) While (!EOF) {
                       read a file name to be processed;

h7:     (Host)         filesread++;

h8:     (Host)         initialize message packet;

h9:     (Host)         if (filesread <= Totalnodes)        {
                              send packet (message) to node_available;
                              node_available++;

n[2]1:  (Node)         for (;;) { /* infinite loop */
                              receives a packet with filename to be processed;

n2:     (Node)                 Initialize metric counters to zero;

n3:     (Node)                 While (Valid Token) {
                                     detect token type and increment the
                                     appropriate counter;
                              } /* end of While loop started at n3: */

n4:     (Node)                 send a packet back to host with collected metrics;
                       } /* end of infinite loop on node started at n1: */
                       else   {
                              receive a message from node that has just completed
                              the metrics from the file it was processing;
                              filescompleted++;
                              store_result;
                              update node_available;
                              send new file_name to node_available;
                       } /* end of else statement */

h10:    (Host) } /* end of while started at h6: */
```

---

1 "h" represents code running on the Host processor.
2 "n" represents code running on the Node processors.

```
h11:  (Host)        if (filesread != filescompleted) {
                        for (j=filescompleted; j<filesread; j++)  {
                            receive computed metrics from nodes;
                            store_result;
                        } /* end of for loop */
                    } /* end if statement */
h12:  (Host) end of the Host program;
```

APPENDIX E.2: THE COMPLETE PATH OF SOURCE CODE FILES USED IN THIS STUDY.

tex/programs/heaps/Beard_Koch/cube386/heap.c tex/programs/heaps/Beard_Koch/cube386/host.c
tex/programs/heaps/Beard_Koch/mem_cube/heap.c tex/programs/heaps/Beard_Koch/mem_cube/host.c
tex/programs/heaps/Fife_Proicou/cube386/HeapSort.c tex/programs/heaps/Fife_Proicou/cube386/NodeHeap.c
tex/programs/heaps/Fife_Proicou/mem_cube/HeapSort.c tex/programs/heaps/Fife_Proicou/mem_cube/NodeHeap.c
tex/programs/meshs/beard/h.c tex/programs/meshs/beard/n.c
tex/programs/projects/NeuralNets/conway/host.c tex/programs/projects/NeuralNets/conway/node.c
tex/programs/projects/NeuralNets/simmers/host.c tex/programs/projects/NeuralNets/simmers/node.c
tex/programs/projects/TSP/rottman/control.c tex/programs/projects/TSP/rottman/host.c tex/programs/projects/TSP/rottman/worker.c
tex/programs/projects/TSP/sawyer/control.c tex/programs/projects/TSP/sawyer/host.c tex/programs/projects/TSP/sawyer/node.h tex/programs/projects/TSP/sawyer/worker.c
tex/programs/projects/harding/ben.c tex/programs/projects/harding/host.c tex/programs/projects/harding/q.h tex/programs/projects/harding/queue.c
tex/programs/projects/huson/src/new_stuff/host.c tex/programs/projects/huson/src/new_stuff/parallel.c tex/programs/projects/huson/src/new_stuff/serial.c
tex/programs/projects/koch/src/host.c tex/programs/projects/koch/src/node.c
tex/programs/rings/beard/host.c tex/programs/rings/beard/node.c

tex/programs/sorts/Beard/cmpf.c tex/programs/sorts/Beard/local.h tex/programs/sorts/Beard/msgio.c tex/programs/sorts/Beard/msgtypes.h tex/programs/sorts/Beard/parsorts.c tex/programs/sorts/Beard/scpgbl.h tex/programs/sorts/Beard/srlsorts.c
tex/programs/sorts/Beard_Koch/cube386/bmerge.c tex/programs/sorts/Beard_Koch/cube386/host.c tex/programs/sorts/Beard_Koch/cube386/local.h tex/programs/sorts/Beard_Koch/cube386/merge.c tex/programs/sorts/Beard_Koch/cube386/oddeven.c
tex/programs/sorts/Beard_Koch/mem_cube/bmerge.c tex/programs/sorts/Beard_Koch/mem_cube/host.c tex/programs/sorts/Beard_Koch/mem_cube/local.h tex/programs/sorts/Beard_Koch/mem_cube/merge.c tex/programs/sorts/Beard_Koch/mem_cube/oddeven.c
tex/programs/sorts/Fife_Proicou/cube386/Bitonic.c tex/programs/sorts/Fife_Proicou/cube386/OddEven.c tex/programs/sorts/Fife_Proicou/cube386/Radix.c tex/programs/sorts/Fife_Proicou/cube386/msort.c tex/programs/sorts/Fife_Proicou/cube386/sort.c
tex/programs/sorts/Fife_Proicou/mem_cube/Bitonic.c tex/programs/sorts/Fife_Proicou/mem_cube/OddEven.c tex/programs/sorts/Fife_Proicou/mem_cube/Radix.c tex/programs/sorts/Fife_Proicou/mem_cube/msort.c tex/programs/sorts/Fife_Proicou/mem_cube/sort.c

APPENDIX E.3: THE MAKE FILE USED TO COMPILE PROGRAMS DEVELOPED FOR COLLECTING THE SIZE AND THE COMMUNICATION MEASUREMENTS.

```
/*
 * makefile
 *
 *
 * This file is used to compile and link the host.c and node.c files for the parallel
 * programs developed to measure size and communication metrics.
 *
 * The command "make all" causes compilation and linking.
*/

all:            host node

sx:             host nodesx

host:  host.o
                cc -o host host.o -host

nodesx:         node.c
                cc node.c -o node -sx -node

node:  node.c
                cc -o node node.c -node -sx

clean:
                rm host node host.o
```

APPENDIX F

PARALLEL PROGRAM TO COLLECT COMMUNICATION MEASUREMENTS

```
/*-------------------------------------------------------------------------------------------------------------*/
/*                                                                                                             */
/*      File            host c   (Host program)                                                                */
/*                                                                                                             */
/*      Author          Imtiaz Ahmad, December 1990                                                            */
/*                                                                                                             */
/*      Purpose         To collect the proposed Communication metrics                                          */
/*                                                                                                             */
/*      Rule:           The Communication metrics are collected by counting the                                */
/*                      number of message-sent and message-receive statements in                               */
/*                      the host and node programs                                                             */
/*                                                                                                             */
/*      Caution:        Program does not distinguish between different types or names                          */
/*                      of send or receive statments                                                           */
/*                                                                                                             */
/*-------------------------------------------------------------------------------------------------------------*/

#include <stdio.h>
#include <cube.h>

#define TOTNODES    32          /* total node - must be < or = alloc  nodes */
#define HOST_PID    100         /* process id of the host process */
#define NODE_PID    0           /* process id for node processes */
#define INIT_TYPE   0           /* type of initialization message */
#define RSLT_TYPE   10          /* type of partial summation message */
#define ALL_NODES   -1          /* symbol for all nodes */
#define ALL_PIDS    -1          /* symbol for all processes */
#define MAX_FILES   200         /* max  files that can be evaluated */
#define PATHLEN     81          /* max  characters in a file path */

struct stat {                   /* saves frequency of communication messages */
        int msgsnd1,
        int msgrcv1,
};

struct info {                   /* structure used to send and recv messages */
        int nodenum,
        int pathnum,
        char filepath[PATHLEN],
        struct stat commsg1,
};

struct stat commsg,
struct info initinfo,
struct info initinfo1[MAX_FILES],       /* store collected metric */

char in_file[PATHLEN],
char f_name[PATHLEN],
FILE *fp,

/*-------------------------------------------------------------------------------------------------------------*/
/*      main()                                                                                                 */
/*                                                                                                             */
/*      Logic           Allocate cube, load node programs to all nodes   Send a packet                         */
```

```
/*                      to each node with file for which metrics needs to be collected      */
/*                      Receives a packet from each node with computed metrics and          */
/*                      stores it in an array of structures for later printing              */
/*                                                                                          */
/*      Caution         Make sure that full pathname of a file has been passed to nodes     */
/*------------------------------------------------------------------------------------------*/
main()
{       /* Host main */

        int i, j, k,                    /* temporary variables */
        fileread,                       /* number of files read from input file */
        filecomp,                       /* number of files received by host after
                                                collecting the metric */
        node_avail,                     /* node number to process next data file */

        getcube ("", "32", "", 0),      /* getcube with given number of nodes */
        setpid(HOST_PID),               /* set the pid of host process */
        load ("node", ALL_NODES, NODE_PID), /* load nodes with node progs */

        open_file(),

        filecomp = fileread = node_avail = 0,

        /* reads file names from user's given input file until end of file */
        while(fgets(f_name, PATHLEN, fp) != NULL)
        {
                fileread++,

                /* initialize structure */
                init_msg(f_name, strlen(f_name), node_avail, fileread),

                /* This if statment will be true for the first n file paths, where n is the      */
                /* number of nodes available in the cube through TOTNODES variable */
                if (fileread <= TOTNODES) {

                        csend(INIT_TYPE, &initinfo, sizeof(initinfo), node_avail, NODE_PID),
                        node_avail++,
                }
                else {
                        crecv (RSLT_TYPE, &initinfo, sizeof(initinfo)),

                        filecomp++,
                        store_result(initinfo.pathnum),
                        node_avail = initinfo.nodenum,
                        init_msg(f_name, strlen(f_name), node_avail, fileread),

                        csend(INIT_TYPE, &initinfo, sizeof(initinfo), node_avail, NODE_PID),
                }

        } /*while fgets*/

        /* This if statment checks whether all activated files are received by the host or node */
        if (fileread != filecomp) {
                for (j=filecomp, j< fileread, j++) {
```

```
                        crecv (RSLT_TYPE, &initinfo, sizeof(initinfo)),

                        /* store result in an array of stuctures */
                        store_result(initinfo pathnum),
                }
        }

        /* print results on screen */
        print_result(fileread),

        fclose(fp),                     /* close input file */
        killcube(ALL_NODES, ALL_PIDS),  /* kill cube */
        relcube(),                      /* release cube */
        printf("Normal termination of the program \n"),
} /* End host main */

/*-------------------------------------------------------------------------------------------------------------------*/
/*      print_info()                                                                                                 */
/*                                                                                                                   */
/*      Purpose         Used for debugging only  It has no contribution in the actual                                */
/*                      execution of the program                                                                     */
/*-------------------------------------------------------------------------------------------------------------------*/
print_info(i)
int i,
{
        printf("%4d %4d %4d %4d %s\n", initinfo1[i+1] nodenum, initinfo1[i+1] pathnum,
initinfo1[i+1] commsg1 msgsnd1, initinfo1[i+1] commsg1 msgrcv1, initinfo1[i+1] filepath),
}

/*-------------------------------------------------------------------------------------------------------------------*/
/*      open_file()                                                                                                  */
/*                                                                                                                   */
/*      Purpose.        Prompts for user input  User must enter the file name that                                   */
/*                      contains the complete pathname of the files for which metrics                                */
/*                      needs to be calculated                                                                       */
/*-------------------------------------------------------------------------------------------------------------------*/
open_file()
{
        printf("Enter input file name  \n\n"),
        /* strcpy(in_file, "aa"), */
        gets(in_file),

        if ((fp=fopen(in_file, "r")) == NULL)
        {
                printf("Can not open file containing path names in host  \n"),
                exit(0),
        }

        /* strcpy(in_file, in_file), */

        printf("                                        Msg  Msg  Tot\n"),
        printf("Input File Name (with complete path)                  send recv  Msg\n"),
        printf("-------------------------------------------------------------------\n"),
```

```
}

/*-------------------------------------------------------------------------------------------------------------------*/
/*      init_msg()                                                                                                   */
/*                                                                                                                   */
/*      Purpose         Initialize the structure before sending it to a node                                         */
/*-------------------------------------------------------------------------------------------------------------------*/
init_msg(fn, i, n, f)
char fn[],
int i, n, f,
{
        int j,
        for(j=0,j<80,j++)
          initinfo filepath[j] = '\0',

        initinfo nodenum = n;                   /* node number to send */
        initinfo pathnum = f,                   /* file number read */
        strncpy(initinfo filepath, fn, i-1),    /* i=length of file name */
        initinfo commsg1 msgsnd1 = -1,          /* init msg vars*/
        initinfo commsg1 msgrcv1 = -1,
}

/*-------------------------------------------------------------------------------------------------------------------*/
/*      store_result()                                                                                               */
/*                                                                                                                   */
/*      Purpose         To store computed metric in an array at subscript i                                          */
/*-------------------------------------------------------------------------------------------------------------------*/
store_result(i)
int i,
{
        initinfo1[i] nodenum = initinfo nodenum,
        initinfo1[i] pathnum = initinfo pathnum,
        strcpy(initinfo1[i] filepath, initinfo filepath),
        initinfo1[i] commsg1 msgrcv1 = initinfo commsg1.msgrcv1,
        initinfo1[i].commsg1 msgsnd1 = initinfo commsg1 msgsnd1,
}

/*-------------------------------------------------------------------------------------------------------------------*/
/*      print_result()                                                                                               */
/*                                                                                                                   */
/*      Purpose         To print the resultant array                                                                 */
/*-------------------------------------------------------------------------------------------------------------------*/
print_result(f)
int f,
{
        int i,
        for (i=0,i< f, i++)
                printf("%-63s %4d %4d  %4d\n\n", initinfo1[i+1] filepath,
                 initinfo1[i+1] commsg1 msgsnd1, initinfo1[i+1] commsg1 msgrcv1,
                 initinfo1[i+1].commsg1 msgsnd1 + initinfo1[i+1] commsg1 msgrcv1),
}

/*---------------------------------------- End of host c Module ----------------------------------------*/
```

```
/*-------------------------------------------------------------------------------------------------------------------*/
/*                                                                                                                   */
/*      File            node c  (Node program)                                                                       */
/*                                                                                                                   */
/*      Author          Imtiaz Ahmad, December 1990                                                                  */
/*                                                                                                                   */
/*      Purpose         To collect Communication measures.                                                           */
/*                                                                                                                   */
/*      Caution         Program does not distinguish between different types or names                                */
/*                      of send or receive statements                                                                */
/*                                                                                                                   */
/*-------------------------------------------------------------------------------------------------------------------*/

#include <stdio.h>
#include <cube.h>
#include "node h"                       /* defines states, classes and state tables */

/*-------------------------------------------------------------------------------------------------------------------*/
/*      main()                                                                                                       */
/*                                                                                                                   */
/*      Logic           Runs an infinite loop  Receives a message from the host with file                            */
/*                      name to be processed  Returns a structure to the host with                                   */
/*                      collected metrics                                                                            */
/*                                                                                                                   */
/*      Caution         It is assumed that there is no syntax error in the input file                                */
/*-------------------------------------------------------------------------------------------------------------------*/
main()
{       /* Node main */

        FILE  *fp,                      /* file pointer */
        char  in_file[PATHLEN],         /* input file name */
              f_name[PATHLEN],          /* file name */
              token[200],               /* token collected */
        int   i, kk,                    /* temporary counters */
              in_fd,                    /* input file descriptor */
              msgsnd,                   /* total message send */
              msgrcv,                   /* total message recv */

        for (;;) {                      /* beginning of infinite loop */

                crecv(INIT_TYPE, &initinfo, sizeof(initinfo)),

                strcpy (in_file, initinfo filepath),

                in_fd = open(in_file,0),

                if (in_fd < = 0) {
                        close(in_fd),
                        csend(RSLT_TYPE, &initinfo, sizeof(initinfo), myhost(), HOST_PID),
                }
                else {
                        msgsnd = 0,
                        msgrcv = 0,
```

```
                        while ( get_tok( in_fd, token ) = = VALID_TOKEN ) {

                                switch( kk=token_type (token) ) {

                                case CSEND
                                        msgsnd+ +,
                                        break,
                                case CRECV
                                        msgrcv+ +,
                                        break,
                                default
                                        break,

                                } /* end switch */
                        } /* end while get_tok() */

                        /* save necessary info before sending it back to host */
                        initinfo nodenum = mynode(),
                        initinfo commsg1 msgsnd1 = msgsnd,
                        initinfo commsg1 msgrcv1 = msgrcv;

                        csend(RSLT_TYPE, &initinfo, sizeof(initinfo), myhost(), HOST_PID),
                        close(in_fd),
                } /* end else */
        } /* end infinite for loop */
} /* end Node main */

/*-------------------------------------------------------------------------------------------------------------------*/
/*      get_tok()                                                                                                    */
/*                                                                                                                   */
/*      Purpose         To collect a basic token according to the rules set by state table                           */
/*-------------------------------------------------------------------------------------------------------------------*/
get_tok(fd,token)
int fd,
char *token,
{
        char c,
        int i,
        int curr_state,
        int nxt_state,
        int char_class,
        int read_flag,

        i = 0,
        token[i] = '\0',
        nxt_state = START,

        while (nxt_state < ENDWORD) {

                read_flag = read(fd,&c,1),

                if (read_flag != 1)
                        char_class = EOF1,
                else
```

```
                char_class = class_tbl[c],

        curr_state = nxt_state,
        nxt_state = state_tbl[curr_state][char_class],

        switch ( nxt_state ) {

        case WORD
                token[i++] = c,
                break,

        case START
        case FIRST_SLASH
        case BEG_COM
        case FIRST_ST.
                break,

        case FIRST_OR
        case FIRST_AND
                token[i++] = c,
                break,

        case ENDWORD
                token[i] = '\0',
                break,

        case ENDWORD_UG

                token[i] = '\0',
                lseek(fd, -1L, 1),
                break,

        case ENDWORD_CC
                token[i++] = c,
                token[i] = '\0',
                break,

        case ERR
                printf("\nError in state table - Metric calculations may be
effected ..node=%d, pathnum=%d\n", mynode(), initinfo pathnum),
                break,

        default
                break,

        }  /* End SWITCH */
    }  /* End WHILE next stae < ENDWORD */

    if ( nxt_state != STOP )
        return( VALID_TOKEN ),
    else
        return( TOKEN_OVER ),
}
```

```
/*-------------------------------------------------------------------------------------------------------------------*/
/*      token_type()                                                                                                 */
/*                                                                                                                   */
/*      Purpose         To classify the input token in one of the two types                                          */
/*                                                                                                                   */
/*      Caution         Assumed that only the following types of send or receive                                     */
/*                      message statements are used in the input file                                                */
/*-------------------------------------------------------------------------------------------------------------------*/
token_type(token)
char *token;
{
        if ( strcmp(token,"csend") = = 0 )
                return( CSEND ),
        if ( strcmp(token,"crecv") = = 0 )
                return( CRECV ),
        if ( strcmp(token,"sendmsg") = = 0 )
                return( CSEND ),
        if ( strcmp(token,"recvmsg") = = 0 )
                return( CRECV ),
        if ( strcmp(token,"send") = = 0 )
                return( CSEND ),
        if ( strcmp(token,"recv") = = 0 )
                return( CRECV ),
        if ( strcmp(token,"sendw") = = 0 )
                return( CSEND ),
        if ( strcmp(token,"recvw") = = 0 )
                return( CRECV ),

        return(NONE),
}

/*-------------------------------------------------------------------------------------------------------------------*/
/*      print_stable()                                                                                               */
/*                                                                                                                   */
/*      Purpose         Prints the state table and is used only for debugging purposses                              */
/*-------------------------------------------------------------------------------------------------------------------*/
print_stable()
{
        int i,j,
        for (i=0, i<MAX_STATES, i++){
                for (j=0, j<MAX_CLASSES, j++) {
                        printf("%3d ", state_tbl[i][j]),
                }
                printf("\n"),
        }
}

/*-------------------------------------------------------------------------------------------------------------------*/
/*      print_ctable()                                                                                               */
/*                                                                                                                   */
/*      Purpose         Prints the class table and is used only for debugging purposses                              */
/*-------------------------------------------------------------------------------------------------------------------*/
print_ctable()
{
```

```
        int i, j,
        j = 0,
        for (i=0, i<150, i++) {
                j++,
                printf("%3d,", class_tbl[i]),
                if (j == 10) {
                        j = 0,
                        printf("\n"),
                }
        }
}

/*---------------------------------------- End of node c Module ----------------------------------------*/
```

```
/*-------------------------------------------------------------------------------------------------------------*/
/*                                                                                                             */
/*      File            node h  (Node program)                                                                 */
/*                                                                                                             */
/*      Author          Imtiaz Ahmad, December 1990                                                            */
/*                                                                                                             */
/*      Purpose         To define all common global variables                                                  */
/*                                                                                                             */
/*      Caution         Must be included in the node c file                                                    */
/*                                                                                                             */
/*-------------------------------------------------------------------------------------------------------------*/

#define HOST_PID        100             /* Process id of the host process */
#define INIT_TYPE       0               /* Type of initialization message */
#define RSLT_TYPE       10              /* Type of message that stores result */
#define MAX_FILES       200             /* Files that can be analyzed */
#define PATHLEN         81              /* Complete file path length */

struct stat {                           /* Structure that saves the metric */
        int msgsnd1,
        int msgrcv1,
},

struct info {                           /* Structure that saves necessary info */
        int nodenum,                    /* regarding metric */
        int pathnum,
        char filepath[PATHLEN],
        struct stat commsg1,
},

struct stat commsg,
struct info initinfo,

#define VALID_TOKEN  1
#define TOKEN_OVER  -1
#define MAX_STATES   12
#define MAX_CLASSES 9


/*------------------------------------------------ Token Types ------------------------------------------------*/
#define CSEND           0
#define CRECV           1
#define NONE            2


/*---------------------------------------------- Character Classes ----------------------------------------------*/
#define AL              0               /* Alphanumeric Characters */
#define EOF1            1               /* End of file */
#define SL              2               /* Slash character */
#define ST              3               /* Star Character */
#define OR              4               /* Bitwise OR operator */
#define AND             5               /* Bitwise AND operator */
#define WH              6               /* Equivalent white space characters (For this program)
                                                like '+','-','\n','[','<',etc , */
```

```
#define QN          7          /* Question mark */
#define ILL         8          /* Illegal characters */


/*---------------------------------------- States ----------------------------------------*/
#define START        0          /* Start state, Begin to collect a token */
#define WORD         1          /* Collecting a token */
#define FIRST_SLASH  2          /* There is a slash, maybe this is a begin
                                        comment mark */
#define BEG_COM      3          /* Yes   This is a begin comment mark */
#define FIRST_ST     4          /* There is a star  Maybe this is the end
                                        comment mark */
#define FIRST_OR     5          /* There is an OR operator. See if there is
                                        one more OR operator to make it a token */
#define FIRST_AND    6          /* There is an AND operator. See if there is
                                        one more AND operator to make it a token */
#define ENDWORD      7          /* Collected a token */
#define ENDWORD_UG8             /* Collected a token, Unget last char */
#define ENDWORD_CC9             /* Collected a token, Add the last character
                                        read, to the token */
#define STOP         10         /* End of file reached */
#define ERR          11         /* Error in state table */


/*------------------------------ Character Class Table ------------------------------*/
int class_tbl[] = {

        ILL,    ILL,    ILL,    ILL,    ILL,    ILL,    ILL,    ILL,    ILL,    WH,
        WH,     ILL,    ILL,    WH,     ILL,    ILL,    ILL,    ILL,    ILL,    ILL,
        ILL,    ILL,    ILL,    ILL,    ILL,    ILL,    ILL,    ILL,    ILL,    ILL,
        ILL,    ILL,    WH,     WH,     WH,     WH,     WH,     WH,     AND,    WH,
        WH,     WH,     ST,     WH,     WH,     WH,     WH,     SL,     AL,     AL,
        AL,     AL,     AL,     AL,     AL,     AL,     AL,     AL,     WH,     WH,
        WH,     WH,     WH,     QN,     WH,     AL,     AL,     AL,     AL,     AL,
        AL,     AL,     AL,     AL,     AL,     AL,     AL,     AL,     AL,     AL,
        AL,     AL,     AL,     AL,     AL,     AL,     AL,     AL,     AL,     AL,
        AL,     WH,     WH,     WH,     WH,     WH,     WH,     AL,     AL,     AL,
        AL,     AL,     AL,     AL,     AL,     AL,     AL,     AL,     AL,     AL,
        AL,     AL,     AL,     AL,     AL,     AL,     AL,     AL,     AL,     AL,
        AL,     AL,     AL,     WH,     OR,     WH,     WH,     ILL,    ILL,    ILL,
        ILL,    ILL,    ILL,    ILL,    ILL,    ILL,    ILL,    ILL,    ILL,    ILL,
        ILL,    ILL,    ILL,    ILL,    ILL,    ILL,    ILL,    ILL,    ILL,    ILL,
        ILL,    ILL,    ILL,    ILL,    ILL,    ILL,    ILL,    ILL,    ILL,    ILL,
        ILL,    ILL,    ILL,    ILL,    ILL,    ILL,    ILL,    ILL,    ILL,    ILL,
        ILL,    ILL,    ILL,    ILL,    ILL,    ILL,    ILL,    ILL,    ILL,    ILL,
        ILL,    ILL,    ILL,    ILL,    ILL,    ILL,    ILL,    ILL,    ILL,    ILL,
        ILL,    ILL,    ILL,    ILL,    ILL,    ILL,    ILL,    ILL,    ILL,    ILL,
        ILL,    ILL,    ILL,    ILL,    ILL,    ILL,    ILL,    ILL,    ILL,    ILL,
        ILL,    ILL,    ILL,    ILL,    ILL,    ILL,    ILL,    ILL,    ILL,    ILL,
        ILL,    ILL,    ILL,    ILL,    ILL,    ILL,    ILL,    ILL,    ILL,    ILL,
        ILL,    ILL,    ILL,    ILL,    ILL,    ILL,    ILL,    ILL,    ILL,    ILL,
        ILL,    ILL,    ILL,    ILL,    ILL,    ILL,    ILL,    ILL,    ILL,    ILL,
        ILL,    ILL,    ILL,    ILL,    ILL
```

```
};

/* ---------------------------------------------- State Table ----------------------------------------------*/
int state_tbl[MAX_STATES][MAX_CLASSES] = {

        /* Class        AL          EOF1          SL          ST
        OR      AND         WH              QN          ILL */

        /*Token*/

        /*START*/   WORD,       STOP,       FIRST_SLASH,        START,
        FIRST_OR,   FIRST_AND,  START,      ENDWORD_CC,         ERR,

        /*WORD*/    WORD,       ENDWORD,    FIRST_SLASH,        ENDWORD,
        ENDWORD_UG, ENDWORD_UG, ENDWORD,    ENDWORD_UG,         ERR,

        /*FIRSTSLASH*/ ENDWORD_UG, ENDWORD,   ENDWORD_UG,     BEG_COM,
        ENDWORD_UG, ENDWORD_UG, ENDWORD_UG, ENDWORD_UG,     ERR,

        /*BEG COM*/ BEG_COM,    ERR,        BEG_COM,        FIRST_ST,
        BEG_COM,    BEG_COM,    BEG_COM,    BEG_COM,        ERR,

        /*FIRST STAR*/ BEG_COM, ERR,        ENDWORD,        FIRST_ST,
        BEG_COM,    BEG_COM,    BEG_COM,    BEG_COM,        ERR,

        /*FIRST OR*/ ENDWORD_UG, ERR,       ENDWORD_UG,     ENDWORD_UG,
        ENDWORD_CC, ENDWORD_UG, ENDWORD_UG, ENDWORD_UG,     ERR,

        /*FIRST AND*/ ENDWORD_UG, ERR,      ENDWORD_UG,     ENDWORD_UG,
        ENDWORD_UG, ENDWORD_CC, ENDWORD_UG, ENDWORD_UG,     ERR

},

/*---------------------------------------- End of node h Module ----------------------------------------*/
```

APPENDIX G

INTER-METRIC CORRELATION ANALYSES

Correlation Analysis

10 'WITH' Variables: HEXELNS HBLKLNS HCMTLNS NEXELNS NBLKLNS NCMTLNS TEXELNS TBLKLNS TCMTLNS TOTLNS
6 'VAR' Variables: Q5 Q6 Q7 Q8 Q9 Q10

Simple Statistics

Variable	N	Mean	Std Dev	Median	Minimum	Maximum
HEXELNS	18	118 444444	73 807558	97 500000	29 000000	253 000000
HBLKLNS	18	34 777778	19 028015	36 000000	8 000000	84 000000
HCMTLNS	18	94 833333	69 959862	62 500000	20 000000	233 000000
NEXELNS	18	277 944444	204 744453	206 000000	58 000000	795 000000
NBLKLNS	18	75 500000	59 602852	49 000000	12 000000	228 000000
NCMTLNS	18	246 722222	170 113044	283 500000	64 000000	600 000000
TEXELNS	18	393 000000	239 235301	361 500000	99 000000	884 000000
TBLKLNS	18	108 555556	73 533684	90 000000	25 000000	312 000000
TCMTLNS	18	337 555556	212 931975	329 500000	93 000000	721 000000
TOTLNS	18	839 111111	446 659087	1003.000000	231 000000	1427 000000
Q5	18	13 000000	1 455214	14 000000	10 000000	14 000000
Q6	18	13 055556	1 696787	14 000000	10 000000	15 000000
Q7	18	14 444444	1 199128	15 000000	12 000000	16 000000
Q8	18	14 166667	1 424574	15 000000	12 000000	16 000000
Q9	18	10 888889	3 027111	10 000000	5 000000	16 000000
Q10	18	4 222222	1 437136	4 000000	3 000000	8.000000

Spearman Correlation Coefficients / Prob > |R| under Ho Rho=0 / N = 18

	Q5	Q6	Q7	Q8	Q9	Q10
HEXELNS	-0 00707 0 9778	-0.22642 0 3663	-0.21364 0 3947	-0 20220 0 4210	0 16694 0 5079	-0 23806 0 3415
HBLKLNS	-0 22872 0.3613	-0 42803 0 0764	-0 43659 0 0701	-0 43795 0 0691	0 46359 0 0527	-0 51854 0 0275
HCMTLNS	0 45567 0 0574	0 28346 0 2544	0 28196 0 2570	0 32235 0 1920	-0 27611 0 2674	0 13025 0 6064
NEXELNS	-0 44777 0 0624	-0 46080 0 0543	-0 48182 0 0429	-0 56770 0 0140	0 62347 0 0057	-0 45875 0 0555
NBLKLNS	-0 61274 0 0069	-0 55808 0 0161	-0 58409 0.0109	-0 68935 0 0016	0 73250 0 0005	-0 56538 0 0145
NCMTLNS	-0 26513 0 2877	-0 32541 0 1876	-0 34091 0 1662	-0 36328 0 1384	0 39521 0 1045	-0 38436 0.1153
TEXELNS	-0 31564 0 2020	-0 39802 0 1019	-0 41797 0 0843	-0 47192 0 0480	0 54824 0 0185	-0 43621 0 0703
TBLKLNS	-0 60301 0 0081	-0 54927 0 0182	-0 57697 0 0122	-0 68178 0 0018	0 73212 0 0006	-0 56509 0 0145
TCMTLNS	0 02120 0 9335	-0 09666 0 7028	-0 11358 0 6536	-0 08550 0 7359	0 15664 0 5348	-0 21067 0 4014

Correlation Analysis

6 'WITH' Variables: Q5 Q6 Q7 Q8 Q9 Q10
6 'VAR' Variables: Q5 Q6 Q7 Q8 Q9 Q10

Simple Statistics

Variable	N	Mean	Std Dev	Median	Minimum	Maximum
Q5	18	13 000000	1 455214	14 000000	10 000000	14 000000
Q6	18	13 055556	1 696787	14 000000	10.000000	15 000000
Q7	18	14 444444	1 199128	15 000000	12 000000	16 000000
Q8	18	14 166667	1 424574	15 000000	12 000000	16 000000
Q9	18	10.888889	3 027111	10 000000	5.000000	16 000000
Q10	18	4.222222	1 437136	4.000000	3 000000	8 000000

Spearman Correlation Coefficients / Prob > |R| under Ho: Rho=0 / N = 18

	Q5	Q6	Q7	Q8	Q9	Q10
Q5	1.00000 0.0	0 87992 0 0001	0.88141 0 0001	0 91240 0 0001	-0 83163 0 0001	0 70430 0.0011
Q6	0.87992 0 0001	1.00000 0.0	0 99625 0.0001	0 96174 0.0001	-0 87055 0.0001	0 89581 0 0001
Q7	0.88141 0.0001	0.99625 0.0001	1.00000 0.0	0.96787 0.0001	-0 89944 0 0001	0.91652 0.0001
Q8	0.91240 0.0001	0.96174 0.0001	0.96787 0 0001	1 00000 0.0	-0 94528 0.0001	0 89603 0.0001
Q9	-0 83163 0 0001	-0.87055 0.0001	-0 89944 0.0001	-0.94528 0.0001	1 00000 0 0	-0 91594 0 0001
Q10	0 70430 0 0011	0 89581 0 0001	0.91652 0.0001	0 89603 0 0001	-0 91594 0.0001	1.00000 0.0

Correlation Analysis

Spearman Correlation Coefficients / Prob > |R| under Ho: Rho=0 / N = 18

	Q5	Q6	Q7	Q8	Q9	Q10
TOTLNS	-0.23685 0.3440	-0.38230 0.1174	-0.40000 0.1000	-0.41994 0.0827	0.49742 0.0357	-0.47611 0.0458

Correlation Analysis

17 'WITH' Variables: HUN1 HUN2 HCAPN1 HCAPN2 HEFRT NUN1 NUN2 NCAPN1 NCAPN2 NEFRT TUN1 TUN2 TUN1N2 TCAPN1 TCAPN2 TCAPN1N2 TEFRT

6 'VAR' Variables: Q5 Q6 Q7 Q8 Q9 Q10

Simple Statistics

Variable	N	Mean	Std Dev	Median	Minimum	Maximum
HUN1	18	37.555556	11.642833	35.000000	23.000000	61.000000
HUN2	18	55.888889	25.840891	54.500000	19.000000	113.000000
HCAPN1	18	338.944444	225.123880	249.000000	69.000000	827.000000
HCAPN2	18	188.777778	122.758757	143.500000	36.000000	446.000000
HEFRT	18	302424	362591	129673	12337	1140590
NUN1	18	44.500000	12.701320	46.000000	27.000000	73.000000
NUN2	18	72.388889	40.744742	69.500000	26.000000	190.000000
NCAPN1	18	942.222222	710.268844	687.000000	151.000000	2565.000000
NCAPN2	18	591.833333	465.025521	412.000000	80.000000	1608.000000
NEFRT	18	2813805	3679114	1054636	56230	13686523
TUN1	18	82.055556	22.367182	83.000000	50.000000	134.000000
TUN2	18	128.277778	61.622003	127.000000	56.000000	303.000000
TUN1N2	18	210.333333	83.223300	211.500000	106.000000	437.000000
TCAPN1	18	1281.166667	808.642017	1193.500000	274.000000	2824.000000
TCAPN2	18	780.611111	511.298478	720.500000	142.000000	1764.000000
TCAPN1N2	18	2061.777778	1317.423937	1889.000000	416.000000	4588.000000
TEFRT	18	5209993	5339407	3614345	180012	18474019
Q5	18	13.000000	1.455214	14.000000	10.000000	14.000000
Q6	18	13.055556	1.696787	14.000000	10.000000	15.000000
Q7	18	14.444444	1.199128	15.000000	12.000000	16.000000
Q8	18	14.166667	1.424574	15.000000	12.000000	16.000000
Q9	18	10.888889	3.027111	10.000000	5.000000	16.000000
Q10	18	4.222222	1.437136	4.000000	3.000000	8.000000

Spearman Correlation Coefficients / Prob > |R| under Ho: Rho=0 / N = 18

	Q5	Q6	Q7	Q8	Q9	Q10
HUN1	-0.04308 0.8652	-0.24102 0.3353	-0.23445 0.3490	-0.21141 0.3997	0.19962 0.4271	-0.23843 0.3407
HUN2	-0.22966 0.3593	-0.39120 0.1084	-0.37708 0.1229	-0.37420 0.1261	0.29966 0.2270	-0.33955 0.1680
HCAPN1	-0.08009 0.7521	-0.29112 0.2412	-0.28167 0.2575	-0.27649 0.2667	0.22134 0.3774	-0.27759 0.2647
HCAPN2	-0.06124 0.8092	-0.29567 0.2336	-0.28622 0.2496	-0.27205 0.2748	0.21793 0.3850	-0.29742 0.2307
HEFRT	-0.03180 0.9003	-0.26724 0.2837	-0.25441 0.3083	-0.22985 0.3589	0.17367 0.4907	-0.25776 0.3018
NUN1	-0.40047 0.0996	-0.41117 0.0901	-0.43454 0.0715	-0.48822 0.0398	0.55761 0.0162	-0.43316 0.0725

Correlation Analysis

Spearman Correlation Coefficients / Prob > |R| under Ho: Rho=0 / N = 18

	Q5	Q6	Q7	Q8	Q9	Q10
NUN2	-0.40514 0 0953	-0 39575 0.1040	-0.41797 0.0843	-0.47636 0.0457	0 53462 0.0223	-0.40647 0.0942
NCAPN1	-0 49112 0 0385	-0 52993 0.0237	-0 54972 0 0181	-0 61738 0 0063	0 67083 0 0023	-0 53535 0 0220
NCAPN2	-0 52881 0.0240	-0 54244 0 0200	0 56335 0 0149	-0 64292 0 0040	0 69807 0 0013	-0 53535 0.0220
NEFRT	-0 54765 0 0186	-0 55950 0.0158	-0 58152 0.0114	-0 66401 0 0027	0 71509 0 0009	-0.55518 0.0168
TUN1	-0.18373 0 4655	-0.32751 0.1846	-0.33619 0.1726	-0 32534 0.1877	0 36436 0.1371	-0 35690 0.1460
TUN2	-0 30268 0 2221	-0 38892 0.1107	-0 39979 0 1002	-0 42750 0.0768	0 45970 0 0549	-0.39903 0.1009
TUN1N2	-0 26527 0.2874	-0.36200 0.1399	-0 37065 0 1300	-0.39182 0.1078	0 42268 0.0805	-0 36843 0.1325
TCAPN1	-0.41810 0.0842	-0.46739 0.0505	-0.49066 0.0387	-0 56075 0.0155	0.62996 0.0051	-0.49569 0.0364
TCAPN2	-0 46403 0.0524	-0.50719 0.0317	-0.52700 0.0246	-0 59628 0 0090	0.65039 0.0035	-0.51552 0.0285
TCAPN1N2	-0.44519 0 0641	-0.49013 0.0389	-0 51337 0.0293	-0.58184 0.0113	0.65039 0 0035	-0 51552 0 0285
TEFRT	-0 42634 0.0777	-0.47307 0.0474	-0 49520 0.0367	-0.56075 0 0155	0 63337 0.0048	-0 49569 0 0364

Correlation Analysis

3 'WITH' Variables: HOSTVG NODEVG TOTVG
6 'VAR' Variables: Q5 Q6 Q7 Q8 Q9 Q10

Simple Statistics

Variable	N	Mean	Std Dev	Median	Minimum	Maximum
HOSTVG	18	14.444444	14.455702	7.000000	2.000000	47.000000
NODEVG	18	57.666667	49.413025	35.000000	6.000000	152.000000
TOTVG	18	72.111111	53.198727	52.500000	10.000000	158.000000
Q5	18	13.000000	1.455214	14.000000	10.000000	14.000000
Q6	18	13.055556	1.696787	14.000000	10.000000	15.000000
Q7	18	14.444444	1.199128	15.000000	12.000000	16.000000
Q8	18	14.166667	1.424574	15.000000	12.000000	16.000000
Q9	18	10.888889	3.027111	10.000000	5.000000	16.000000
Q10	18	4.222222	1.437136	4.000000	3.000000	8.000000

Spearman Correlation Coefficients / Prob > |R| under Ho: Rho=0 / N = 18

	Q5	Q6	Q7	Q8	Q9	Q10
HOSTVG	0.16194 0.5209	-0.09164 0.7176	-0.06864 0.7867	-0.02461 0.9228	-0.01029 0.9677	-0.08364 0.7414
NODEVG	-0.53053 0.0235	-0.55837 0.0160	-0.58439 0.0109	-0.65081 0.0034	0.73288 0.0005	-0.59545 0.0091
TOTVG	-0.39259 0.1071	-0.46047 0.0545	-0.47979 0.0439	-0.53409 0.0224	0.61016 0.0072	-0.49621 0.0362

Correlation Analysis

7 'WITH' Variables. HMSGSND HMSGREC NMSGSND NMSGREC TMSGSND TMSGREC TCOMMSG
6 'VAR' Variables. Q5 Q6 Q7 Q8 Q9 Q10

Simple Statistics

Variable	N	Mean	Std Dev	Median	Minimum	Maximum
HMSGSND	18	4.277778	2.803476	4.500000	0	9.000000
HMSGREC	18	2.611111	1.144752	2.000000	1.000000	5.000000
NMSGSND	18	8.888889	6.641128	9.000000	2.000000	32.000000
NMSGREC	18	10.555556	6.921596	10.000000	2.000000	33.000000
TMSGSND	18	13.166667	7.270003	13.000000	3.000000	35.000000
TMSGREC	18	13.166667	7.196813	13.000000	3.000000	35.000000
TCOMMSG	18	26.333333	14.418126	27.000000	6.000000	70.000000
Q5	18	13.000000	1.455214	14.000000	10.000000	14.000000
Q6	18	13.055556	1.696787	14.000000	10.000000	15.000000
Q7	18	14.444444	1.199128	15.000000	12.000000	16.000000
Q8	18	14.166667	1.424574	15.000000	12.000000	16.000000
Q9	18	10.888889	3.027111	10.000000	5.000000	16.000000
Q10	18	4.222222	1.437136	4.000000	3.000000	8.000000

Spearman Correlation Coefficients / Prob > |R| under Ho: Rho=0 / N = 18

	Q5	Q6	Q7	Q8	Q9	Q10
HMSGSND	0.39115 0.1085	0.02127 0.9332	0.05512 0.8280	0.11395 0.6526	-0.20484 0.4149	0.00877 0.9724
HMSGREC	-0.20030 0.4255	-0.16569 0.5111	-0.19015 0.4498	-0.29062 0.2420	0.33377 0.1759	-0.18121 0.4718
NMSGSND	-0.33473 0.1746	-0.34211 0.1646	-0.38461 0.1150	-0.48289 0.0424	0.57827 0.0119	-0.43963 0.0679
NMSGREC	-0.00120 0.9962	-0.21963 0.3812	-0.21355 0.3949	-0.23658 0.3446	0.22096 0.3783	-0.24693 0.3232
TMSGSND	-0.02843 0.9108	-0.22931 0.3600	-0.24673 0.3236	-0.28477 0.2521	0.31165 0.2081	-0.33277 0.1772
TMSGREC	-0.14713 0.5602	-0.26630 0.2855	-0.27803 0.2639	-0.34595 0.1597	0.38407 0.1156	-0.30336 0.2211
TCOMMSG	-0.09535 0.7067	-0.25905 0.2993	-0.27643 0.2668	-0.32665 0.1858	0.35959 0.1427	-0.34274 0.1638

Correlation Analysis

6 'WITH' Variables: R1U R1T R2U R2T R3U R3T
6 'VAR' Variables: Q5 Q6 Q7 Q8 Q9 Q10

Simple Statistics

Variable	N	Mean	Std Dev	Median	Minimum	Maximum
R1U	18	1342.900000	732.644191	1326.050000	296.700000	3444.500000
R1T	18	21311	15218	18777	3234.100000	51397
R2U	18	1439.194444	696.135651	1424.650000	610.400000	3409.300000
R2T	18	21741	15753	19021	3207.000000	53797
R3U	18	1238.461111	624.101364	1223.250000	505.800000	3022.500000
R3T	18	19770	14487	17229	2821.800000	49388
Q5	18	13.000000	1.455214	14.000000	10.000000	14.000000
Q6	18	13.055556	1.696787	14.000000	10.000000	15.000000
Q7	18	14.444444	1.199128	15.000000	12.000000	16.000000
Q8	18	14.166667	1.424574	15.000000	12.000000	16.000000
Q9	18	10.888889	3.027111	10.000000	5.000000	16.000000
Q10	18	4.222222	1.437136	4.000000	3.000000	8.000000

Spearman Correlation Coefficients / Prob > |R| under Ho: Rho=0 / N = 18

	Q5	Q6	Q7	Q8	Q9	Q10
R1U	0.02002 0.9371	-0.07278 0.7741	-0.08178 0.7470	-0.14768 0.5587	0.17367 0.4907	-0.08675 0.7322
R1T	-0.44519 0.0641	-0.49013 0.0389	-0.51337 0.0293	-0.58184 0.0113	0.65039 0.0035	-0.51552 0.0285
R2U	-0.26499 0.2879	-0.36845 0.1325	-0.37708 0.1229	-0.39752 0.1023	0.43246 0.0731	-0.37673 0.1233
R2T	-0.44519 0.0641	-0.49013 0.0389	-0.51337 0.0293	-0.58184 0.0113	0.65039 0.0035	-0.51552 0.0285
R3U	-0.26513 0.2877	-0.36182 0.1401	-0.37045 0.1302	-0.39161 0.1080	0.42587 0.0780	-0.36824 0.1327
R3T	-0.44519 0.0641	-0.49013 0.0389	-0.51337 0.0293	-0.58184 0.0113	0.65039 0.0035	-0.51552 0.0285

Correlation Analysis

10 'WITH' Variables HEXELNS HBLKLNS HCMTLNS NEXELNS NBLKLNS NCMTLNS TEXELNS TBLKLNS TCMTLNS TOTLNS
10 'VAR' Variables. HEXELNS HBLKLNS HCMTLNS NEXELNS NBLKLNS NCMTLNS TEXELNS TBLKLNS TCMTLNS TOTLNS

Simple Statistics

Variable	N	Mean	Std Dev	Median	Minimum	Maximum
HEXELNS	18	118 444444	73 807558	97 500000	29 000000	253 000000
HBLKLNS	18	34 777778	19 028015	36 000000	8 000000	84 000000
HCMTLNS	18	94 833333	69 959862	62 500000	20 000000	233 000000
NEXELNS	18	277 944444	204 744453	206 000000	58 000000	795 000000
NBLKLNS	18	75 500000	59 602852	49 000000	12 000000	228 000000
NCMTLNS	18	246 722222	170.113044	283.500000	64 000000	600 000000
TEXELNS	18	393 000000	239 235301	361.500000	99 000000	884 000000
TBLKLNS	18	108 555556	73.533684	90 000000	25.000000	312 000000
TCMTLNS	18	337 555556	212 931975	329 500000	93 000000	721 000000
TOTLNS	18	839 111111	446 659087	1003 000000	231 000000	1427 000000

Spearman Correlation Coefficients / Prob > |R| under Ho Rho=0 / N = 18

	HEXELNS	HBLKLNS	HCMTLNS	NEXELNS	NBLKLNS	NCMTLNS	TEXELNS	TBLKLNS	TCMTLNS	TOTLNS
HEXELNS	1.00000 0.0	0.84599 0.0001	0 67907 0.0019	0 63120 0 0050	0 50981 0.0307	0 67562 0.0021	0.74342 0.0004	0 53485 0 0222	0 70418 0 0011	0 79029 0.0001
HBLKLNS	0.84599 0.0001	1 00000 0.0	0 44623 0 0634	0 75917 0.0003	0 70491 0.0011	0 58191 0.0113	0 85124 0 0001	0 74070 0 0004	0 52996 0 0237	0.89664 0.0001
HCMTLNS	0.67907 0.0019	0 44623 0.0634	1.00000 0.0	0 08372 0.7412	-0.00413 0.9870	0 53333 0.0227	0 22521 0.3689	0 03616 0.8867	0 80475 0 0001	0.40052 0.0995
NEXELNS	0.63120 0.0050	0.75917 0.0003	0.08372 0.7412	1 00000 0 0	0.91632 0.0001	0.59556 0.0091	0 96231 0 0001	0.91275 0 0001	0 37997 0 1199	0.87862 0.0001
NBLKLNS	0.50981 0 0307	0 70491 0.0011	-0 00413 0 9870	0 91632 0 0001	1 00000 0 0	0 60744 0.0075	0.85493 0.0001	0 99226 0 0001	0 35829 0 1443	0.80062 0.0001
NCMTLNS	0.67562 0 0021	0.58191 0.0113	0.53333 0 0227	0 59556 0 0091	0 60744 0.0075	1.00000 0 0	0.57305 0 0129	0 61538 0 0066	0 88797 0 0001	0.71798 0.0008
TEXELNS	0.74342 0 0004	0.85124 0.0001	0.22521 0.3689	0 96231 0.0001	0 85493 0.0001	0 57305 0 0129	1 00000 0 0	0 86171 0 0001	0 44066 0 0672	0 93030 0.0001
TBLKLNS	0 53485 0.0222	0 74070 0 0004	0 03616 0.8867	0 91275 0 0001	0 99226 0 0001	0 61538 0 0066	0 86171 0.0001	1 00000 0 0	0 37874 0 1212	0 80330 0.0001
TCMTLNS	0 70418 0.0011	0 52996 0.0237	0 80475 0.0001	0 37997 0 1199	0 35829 0 1443	0 88797 0 0001	0 44066 0.0672	0 37874 0 1212	1 00000 0 0	0.63191 0 0049
TOTLNS	0 79029 0 0001	0 89664 0 0001	0 40052 0 0995	0 87862 0 0001	0 80062 0 0001	0 71798 0 0008	0 93030 0 0001	0 80330 0 0001	0 63191 0 0049	1 00000 0.0

Correlation Analysis

17 'WITH' Variables: HUN1 HUN2 HCAPN1 HCAPN2 HEFRT NUN1 NUN2 NCAPN1 NCAPN2 NEFRT TUN1 TUN2 TUN1N2 TCAPN1 TCAPN2 TCAPN1N2 TEFRT

10 'VAR' Variables: HEXELNS HBLKLNS HCMTLNS NEXELNS NBLKLNS NCMTLNS TEXELNS TBLKLNS TCMTLNS TOTLNS

Simple Statistics

Variable	N	Mean	Std Dev	Median	Minimum	Maximum
HUN1	18	37.555556	11.642833	35.000000	23.000000	61.000000
HUN2	18	55.888889	25.840891	54.500000	19.000000	113.000000
HCAPN1	18	338.944444	225.123880	249.000000	69.000000	827.000000
HCAPN2	18	188.777778	122.758757	143.500000	36.000000	446.000000
HEFRT	18	302424	362591	129673	12337	1140590
NUN1	18	44.500000	12.701320	46.000000	27.000000	73.000000
NUN2	18	72.388889	40.744742	69.500000	26.000000	190.000000
NCAPN1	18	942.222222	710.268844	687.000000	151.000000	2565.000000
NCAPN2	18	591.833333	465.025521	412.000000	80.000000	1608.000000
NEFRT	18	.2813805	3679114	1054636	56230	13686523
TUN1	18	82.055556	22.367182	83.000000	50.000000	134.000000
TUN2	18	128.277778	61.622003	127.000000	56.000000	303.000000
TUN1N2	18	210.333333	83.223300	211.500000	106.000000	437.000000
TCAPN1	18	1281.166667	808.642017	1193.500000	274.000000	2824.000000
TCAPN2	18	780.611111	511.298478	720.500000	142.000000	1764.000000
TCAPN1N2	18	2061.777778	1317.423937	1889.000000	416.000000	4588.000000
TEFRT	18	5209993	5339407	3614345	180012	18474019
HEXELNS	18	118.444444	73.807558	97.500000	29.000000	253.000000
HBLKLNS	18	34.777778	19.028015	36.000000	8.000000	84.000000
HCMTLNS	18	94.833333	69.959862	62.500000	20.000000	233.000000
NEXELNS	18	277.944444	204.744453	206.000000	58.000000	795.000000
NBLKLNS	18	75.500000	59.602852	49.000000	12.000000	228.000000
NCMTLNS	18	246.722222	170.113044	283.500000	64.000000	600.000000
TEXELNS	18	393.000000	239.235301	361.500000	99.000000	884.000000
TBLKLNS	18	108.555556	73.533684	90.000000	25.000000	312.000000
TCMTLNS	18	337.555556	212.931975	329.500000	93.000000	721.000000
TOTLNS	18	839.111111	446.659087	1003.000000	231.000000	1427.000000

Correlation Analysis

Spearman Correlation Coefficients / Prob > |R| under Ho. Rho=0 / N = 18

	HEXELNS	HBLKLNS	HCMTLNS	NEXELNS	NBLKLNS	NCMTLNS	TEXELNS	TBLKLNS	TCMTLNS	TOTLNS
HUN1	0 94775 0 0001	0 80538 0 0001	0 67029 0.0023	0 66477 0 0026	0 47750 0 0451	0 69374 0 0014	0 75802 0.0003	0 50776 0 0315	0 73320 0 0005	0.79979 0 0001
HUN2	0 81156 0 0001	0.70558 0 0011	0 46074 0 0543	0 55550 0 0167	0 41507 0 0867	0 45225 0 0595	0 63055 0 0050	0 45098 0 0603	0 44272 0 0658	0 55343 0.0172
HCAPN1	0 93340 0 0001	0.82128 0 0001	0 59711 0 0089	0 63913 0 0043	0 47909 0 0443	0 58028 0 0116	0 72343 0 0007	0 51496 0 0287	0 57276 0 0130	0 70005 0.0012
HCAPN2	0 93960 0.0001	0 85021 0.0001	0.60227 0.0082	0 63500 0.0046	0 47703 0 0453	0 57202 0 0131	0 72755 0.0006	0 51084 0 0303	0 56863 0 0138	0 72277 0.0007
HEFRT	0 96644 0.0001	0.83471 0.0001	0.66012 0.0029	0 61642 0.0064	0 45638 0.0569	0 63191 0 0049	0.71311 0.0009	0 48813 0 0399	0 64499 0 0038	0.73929 0.0005
NUN1	0 66339 0 0027	0.73047 0.0006	0.35954 0.1428	0 85471 0.0001	0 81489 0 0001	0 77766 0 0001	0 84651 0 0001	0 82377 0 0001	0 65530 0 0032	0.86712 0 0001
NUN2	0 62674 0.0054	0 68905 0.0016	0 26446 0.2889	0 88487 0.0001	0 77233 0.0002	0 72070 0 0007	0 85759 0 0001	0 78535 0.0001	0 54386 0 0196	0 82912 0.0001
NCAPN1	0 63810 0 0044	0.79236 0.0001	0.10537 0.6773	0 96954 0.0001	0 89623 0.0001	0 60919 0.0073	0 95666 0 0001	0 90093 0 0001	0 40144 0.0987	0 89623 0.0001
NCAPN2	0 60712 0.0075	0 76343 0.0002	0.06302 0.8038	0.98503 0.0001	0 92308 0 0001	0.60919 0.0073	0 95046 0.0001	0 92570 0.0001	0 38493 0 1147	0.87558 0.0001
NEFRT	0.59060 0 0099	0 75517 0.0003	0 05269 0.8355	0 97264 0 0001	0 93753 0 0001	0 63810 0 0044	0 92982 0.0001	0 93808 0 0001	0 39938 0.1006	0.87145 0.0001
TUN1	0 89004 0.0001	0.81715 0.0001	0 62397 0.0056	0 75065 0 0003	0 60093 0 0084	0 77852 0.0001	0 82250 0.0001	0 63055 0.0050	0 77915 0 0001	0 85596 0.0001
TUN2	0 81053 0.0001	0.82541 0.0001	0 46694 0.0507	0 83015 0.0001	0 68456 0.0017	0 67424 0 0022	0 86791 0 0001	0 71930 0 0008	0 60784 0 0075	0 83738 0.0001
TUN1N2	0 84134 0.0001	0 80558 0 0001	0 51706 0.0280	0 80052 0.0001	0.64393 0.0039	0.69302 0.0014	0 84917 0.0001	0 67872 0 0020	0 65186 0.0034	0 81757 0.0001
TCAPN1	0 66908 0.0024	0 81095 0.0001	0 18905 0.4525	0 95509 0.0001	0 84770 0 0001	0 60919 0.0073	0.95253 0.0001	0 86584 0 0001	0 43240 0 0731	0.87661 0.0001
TCAPN2	0 67321 0.0022	0 82025 0.0001	0 14979 0.5530	0 96438 0.0001	0 87558 0 0001	0 60093 0 0084	0 95872 0 0001	0 89061 0 0001	0 40764 0 0931	0.87971 0.0001
TCAPN1N2	0 67734 0 0020	0.82231 0 0001	0 17459 0.4884	0 96231 0 0001	0 87248 0.0001	0 61125 0 0070	0 96285 0 0001	0 88854 0 0001	0 43240 0 0731	0 88384 0 0001
TEFRT	0 68353 0 0018	0 82438 0 0001	0.18079 0 4728	0 96644 0 0001	0 87042 0 0001	0 61538 0 0066	0 97110 0.0001	0 88442 0 0001	0 44272 0 0658	0 90036 0.0001

Correlation Analysis

3 'WITH' Variables: HOSTVG NODEVG TOTVG
10 'VAR' Variables: HEXELNS HBLKLNS HCMTLNS NEXELNS NBLKLNS NCMTLNS TEXELNS TBLKLNS TCMTLNS TOTLNS

Simple Statistics

Variable	N	Mean	Std Dev	Median	Minimum	Maximum
HOSTVG	18	14.444444	14.455702	7.000000	2.000000	47.000000
NODEVG	18	57.666667	49.413025	35.000000	6.000000	152.000000
TOTVG	18	72.111111	53.198727	52.500000	10.000000	158.000000
HEXELNS	18	118.444444	73.807558	97.500000	29.000000	253.000000
HBLKLNS	18	34.777778	19.028015	36.000000	8.000000	84.000000
HCMTLNS	18	94.833333	69.959862	62.500000	20.000000	233.000000
NEXELNS	18	277.944444	204.744453	206.000000	58.000000	795.000000
NBLKLNS	18	75.500000	59.602852	49.000000	12.000000	228.000000
NCMTLNS	18	246.722222	170.113044	283.500000	64.000000	600.000000
TEXELNS	18	393.000000	239.235301	361.500000	99.000000	884.000000
TBLKLNS	18	108.555556	73.533684	90.000000	25.000000	312.000000
TCMTLNS	18	337.555556	212.931975	329.500000	93.000000	721.000000
TOTLNS	18	839.111111	446.659087	1003.000000	231.000000	1427.000000

Correlation Analysis

Spearman Correlation Coefficients / Prob > |R| under Ho: Rho=0 / N = 18

	HEXELNS	HBLKLNS	HCMTLNS	NEXELNS	NBLKLNS	NCMTLNS	TEXELNS	TBLKLNS	TCMTLNS	TOTLNS
HOSTVG	0.83517 0 0001	0.65974 0.0029	0.65610 0.0031	0 46803 0 0501	0.19969 0 4269	0.46959 0 0493	0 57382 0.0128	0 22350 0 3727	0.58110 0 0114	0 64328 0 0040
NODEVG	0 60879 0 0073	0 78645 0 0001	0 13289 0 5991	0 95401 0 0001	0 91163 0 0001	0 66925 0 0024	0 93079 0 0001	0 92046 0 0001	0 47211 0 0479	0.89457 0 0001
TOTVG	0 68320 0 0018	0 82265 0.0001	0 22182 0 3764	0 93540 0 0001	0 81654 0 0001	0 61447 0 0067	0 94938 0 0001	0 83368 0 0001	0 46591 0 0513	0 90698 0 0001

Correlation Analysis

7 'WITH' Variables: HMSGSND HMSGREC NMSGSND NMSGREC TMSGSND TMSGREC TCOMMSG
10 'VAR' Variables: HEXELNS HBLKLNS HCMTLNS NEXELNS NBLKLNS NCMTLNS TEXELNS TBLKLNS TCMTLNS TOTLNS

Simple Statistics

Variable	N	Mean	Std Dev	Median	Minimum	Maximum
HMSGSND	18	4.277778	2.803476	4.500000	0	9.000000
HMSGREC	18	2.611111	1.144752	2.000000	1.000000	5.000000
NMSGSND	18	8.888889	6.641128	9.000000	2.000000	32.000000
NMSGREC	18	10.555556	6.921596	10.000000	2.000000	33.000000
TMSGSND	18	13.166667	7.270003	13.000000	3.000000	35.000000
TMSGREC	18	13.166667	7.196813	13.000000	3.000000	35.000000
TCOMMSG	18	26.333333	14.418126	27.000000	6.000000	70.000000
HEXELNS	18	118.444444	73.807558	97.500000	29.000000	253.000000
HBLKLNS	18	34.777778	19.028015	36.000000	8.000000	84.000000
HCMTLNS	18	94.833333	69.959862	62.500000	20.000000	233.000000
NEXELNS	18	277.944444	204.744453	206.000000	58.000000	795.000000
NBLKLNS	18	75.500000	59.602852	49.000000	12.000000	228.000000
NCMTLNS	18	246.722222	170.113044	283.500000	64.000000	600.000000
TEXELNS	18	393.000000	239.235301	361.500000	99.000000	884.000000
TBLKLNS	18	108.555556	73.533684	90.000000	25.000000	312.000000
TCMTLNS	18	337.555556	212.931975	329.500000	93.000000	721.000000
TOTLNS	18	839.111111	446.659087	1003.000000	231.000000	1427.000000

Correlation Analysis

Spearman Correlation Coefficients / Prob > |R| under Ho: Rho=0 / N = 18

	HEXELNS	HBLKLNS	HCMTLNS	NEXELNS	NBLKLNS	NCMTLNS	TEXELNS	TBLKLNS	TCMTLNS	TOTLNS
HMSGSND	0.64148 0.0041	0.40524 0.0952	0.49089 0.0386	0.12005 0.6352	-0.12579 0.6189	0.06159 0.8082	0.25980 0.2978	-0.10851 0.6682	0.19302 0.4429	0.25158 0.3139
HMSGREC	0.60723 0.0075	0.49260 0.0378	0.47618 0.0458	0.63020 0.0051	0.57386 0.0128	0.62145 0.0059	0.60254 0.0081	0.62003 0.0061	0.58504 0.0108	0.50766 0.0315
NMSGSND	0.28201 0.2569	0.45498 0.0578	0.04633 0.8551	0.67119 0.0023	0.58170 0.0113	0.28565 0.2505	0.61156 0.0070	0.61156 0.0070	0.13625 0.5898	0.44330 0.0654
NMSGREC	0.61037 0.0071	0.63655 0.0045	0.22116 0.3778	0.53493 0.0222	0.32708 0.1852	0.05961 0.8142	0.62007 0.0060	0.34800 0.1570	0.04429 0.8615	0.46213 0.0535
TMSGSND	0.58516 0.0107	0.65092 0.0034	0.29974 0.2269	0.62825 0.0052	0.43355 0.0723	0.24611 0.3249	0.66218 0.0028	0.45045 0.0607	0.20447 0.4157	0.55452 0.0169
TMSGREC	0.64234 0.0040	0.69649 0.0013	0.22363 0.3724	0.72903 0.0006	0.55042 0.0179	0.23239 0.3534	0.78084 0.0001	0.57937 0.0117	0.17851 0.4785	0.59534 0.0091
TCOMMSG	0.62773 0.0053	0.69403 0.0014	0.28676 0.2486	0.66564 0.0026	0.49378 0.0373	0.24922 0.3186	0.71407 0.0009	0.52103 0.0266	0.21277 0.3966	0.56854 0.0138

Correlation Analysis

6 'WITH' Variables: R1U R1T R2U R2T R3U R3T
10 'VAR' Variables: HEXELNS HBLKLNS HCMTLNS NEXELNS NBLKLNS NCMTLNS TEXELNS TBLKLNS TCMTLNS TOTLNS

Simple Statistics

Variable	N	Mean	Std Dev	Median	Minimum	Maximum
R1U	18	1342.900000	732.644191	1326.050000	296 700000	3444.500000
R1T	18	21311	15218	18777	3234 100000	51397
R2U	18	1439.194444	696.135651	1424.650000	610.400000	3409.300000
R2T	18	21741	15753	19021	3207 000000	53797
R3U	18	1238.461111	624.101364	1223.250000	505.800000	3022.500000
R3T	18	19770	14487	17229	2821.800000	49388
HEXELNS	18	118.444444	73.807558	97.500000	29.000000	253.000000
HBLKLNS	18	34.777778	19.028015	36.000000	8.000000	84.000000
HCMTLNS	18	94.833333	69.959862	62.500000	20 000000	233.000000
NEXELNS	18	277.944444	204.744453	206.000000	58.000000	795.000000
NBLKLNS	18	75.500000	59.602852	49 000000	12 000000	228.000000
NCMTLNS	18	246.722222	170.113044	283.500000	64.000000	600.000000
TEXELNS	18	393.000000	239.235301	361.500000	99.000000	884.000000
TBLKLNS	18	108.555556	73.533684	90.000000	25.000000	312.000000
TCMTLNS	18	337.555556	212.931975	329.500000	93.000000	721.000000
TOTLNS	18	839.111111	446.659087	1003.000000	231.000000	1427.000000

Correlation Analysis

Spearman Correlation Coefficients / Prob > |R| under Ho: Rho=0 / N = 18

	HEXELNS	HBLKLNS	HCMTLNS	NEXELNS	NBLKLNS	NCMTLNS	TEXELNS	TBLKLNS	TCMTLNS	TOTLNS
R1U	0.71864 0.0008	0.61984 0 0061	0.37190 0.1286	0.70005 0.0012	0.52142 0.0265	0.42230 0 0808	0.74200 0 0004	0.52322 0 0259	0.41176 0 0895	0.64533 0.0038
R1T	0.67734 0 0020	0 82231 0.0001	0.17459 0.4884	0.96231 0.0001	0.87248 0 0001	0.61125 0 0070	0.96285 0.0001	0 88854 0 0001	0.43240 0 0731	0.88384 0.0001
R2U	0 84151 0 0001	0.81302 0 0001	0.50000 0 0346	0.81053 0.0001	0.64739 0.0037	0.68147 0.0018	0 86171 0.0001	0 68215 0 0018	0 63674 0 0045	0.82499 0.0001
R2T	0 67734 0.0020	0 82231 0.0001	0.17459 0.4884	0 96231 0 0001	0.87248 0 0001	0.61125 0.0070	0.96285 0.0001	0.88854 0.0001	0 43240 0.0731	0.88384 0.0001
R3U	0.83781 0.0001	0 81137 0.0001	0.50646 0.0320	0 80733 0.0001	0.64669 0.0037	0.68027 0 0019	0 85803 0.0001	0.68147 0 0018	0 63913 0 0043	0.82335 0.0001
R3T	0.67734 0.0020	0 82231 0.0001	0.17459 0 4884	0.96231 0.0001	0 87248 0.0001	0 61125 0.0070	0 96285 0 0001	0.88854 0 0001	0 43240 0 0731	0.88384 0.0001

Correlation Analysis

17 'WITH' Variables: HUN1 HUN2 HCAPN1 HCAPN2 HEFRT NUN1 NUN2 NCAPN1 NCAPN2 NEFRT TUN1 TUN2 TUN1N2 TCAPN1 TCAPN2 TCAPN1N2 TEFRT

17 'VAR' Variables: HUN1 HUN2 HCAPN1 HCAPN2 HEFRT NUN1 NUN2 NCAPN1 NCAPN2 NEFRT TUN1 TUN2 TUN1N2 TCAPN1 TCAPN2 TCAPN1N2 TEFRT

Simple Statistics

Variable	N	Mean	Std Dev	Median	Minimum	Maximum
HUN1	18	37.555556	11 642833	35.000000	23 000000	61 000000
HUN2	18	55.888889	25.840891	54.500000	19.000000	113.000000
HCAPN1	18	338.944444	225.123880	249.000000	69.000000	827.000000
HCAPN2	18	188.777778	122.758757	143.500000	36.000000	446.000000
HEFRT	18	302424	362591	129673	12337	1140590
NUN1	18	44.500000	12.701320	46.000000	27.000000	73.000000
NUN2	18	72.388889	40.744742	69.500000	26.000000	190.000000
NCAPN1	18	942.222222	710.268844	687.000000	151.000000	2565.000000
NCAPN2	18	591.833333	465.025521	412.000000	80.000000	1608.000000
NEFRT	18	2813805	3679114	1054636	56230	13686523
TUN1	18	82.055556	22.367182	83.000000	50.000000	134.000000
TUN2	18	128.277778	61.622003	127.000000	56.000000	303.000000
TUN1N2	18	210.333333	83.223300	211.500000	106.000000	437.000000
TCAPN1	18	1281.166667	808.642017	1193.500000	274.000000	2824.000000
TCAPN2	18	780.611111	511.298478	720.500000	142.000000	1764.000000
TCAPN1N2	18	2061.777778	1317.423937	1889.000000	416.000000	4588.000000
TEFRT	18	5209993	5339407	3614345	180012	18474019

Correlation Analysis

Spearman Correlation Coefficients / Prob > |R| under Ho: Rho=0 / N = 18

	HUN1	HUN2	HCAPN1	HCAPN2	HEFRT	NUN1	NUN2	NCAPN1	NCAPN2
HUN1	1 00000 0.0	0 81903 0 0001	0.92451 0 0001	0.92451 0 0001	0 95450 0 0001	0 73433 0.0005	0.74561 0 0004	0 69183 0 0015	0 65357 0.0033
HUN2	0 81903 0 0001	1 00000 0.0	0 92776 0 0001	0 90712 0 0001	0 87616 0 0001	0 55091 0 0178	0 61404 0 0067	0 59959 0 0085	0 56656 0 0142
HCAPN1	0 92451 0 0001	0 92776 0 0001	1 00000 0 0	0 99587 0 0001	0 98555 0 0001	0 65840 0 0030	0 66770 0 0025	0 65738 0 0030	0 62642 0 0054
HCAPN2	0 92451 0 0001	0.90712 0.0001	0 99587 0.0001	1 00000 0 0	0 98968 0 0001	0 65013 0 0035	0 65325 0.0033	0 65531 0 0032	0 62023 0.0060
HEFRT	0.95450 0 0001	0.87616 0.0001	0.98555 0.0001	0 98968 0 0001	1 00000 0 0	0 66563 0 0026	0 65531 0 0032	0 63674 0 0045	0 60165 0 0083
NUN1	0 73433 0 0005	0.55091 0 0178	0 65840 0 0030	0 65013 0.0035	0 66563 0 0026	1 00000 0 0	0 92300 0 0001	0 90026 0.0001	0.89199 0 0001
NUN2	0 74561 0 0004	0 61404 0.0067	0 66770 0.0025	0 65325 0.0033	0 65531 0 0032	0 92300 0 0001	1 00000 0.0	0 92363 0 0001	0 90712 0 0001
NCAPN1	0 69183 0 0015	0 59959 0 0085	0 65738 0.0030	0 65531 0.0032	0.63674 0.0045	0.90026 0 0001	0.92363 0.0001	1 00000 0 0	0 99174 0.0001
NCAPN2	0 65357 0 0033	0.56656 0.0142	0.62642 0.0054	0.62023 0.0060	0 60165 0 0083	0 89199 0.0001	0.90712 0 0001	0 99174 0.0001	1 00000 0 0
NEFRT	0.62978 0.0051	0.52735 0.0245	0.60372 0.0080	0 59959 0.0085	0 58308 0.0111	0.90749 0.0001	0.89680 0.0001	0 98555 0.0001	0.99381 0.0001
TUN1	0.95864 0.0001	0.79154 0.0001	0 87822 0 0001	0 87203 0.0001	0 89886 0.0001	0 85685 0.0001	0 85552 0.0001	0.78535 0.0001	0.75851 0.0003
TUN2	0 88728 0 0001	0.84520 0.0001	0 86791 0 0001	0 85346 0.0001	0 84727 0.0001	0 85788 0.0001	0 90918 0 0001	0 86997 0 0001	0.84727 0.0001
TUN1N2	0 91667 0.0001	0 86364 0.0001	0.88946 0.0001	0.87190 0 0001	0 87293 0 0001	0 84635 0.0001	0.88946 0 0001	0 83471 0 0001	0 81302 0.0001
TCAPN1	0.73733 0 0005	0.66357 0.0027	0 72755 0 0006	0 72136 0 0007	0 69453 0.0014	0 90749 0.0001	0 93189 0 0001	0 98142 0 0001	0.97317 0 0001
TCAPN2	0 71872 0 0008	0.66770 0.0025	0 72136 0 0007	0 71517 0 0008	0 68834 0 0016	0 89923 0 0001	0 91744 0 0001	0 98762 0 0001	0.98349 0.0001
TCAPN1N2	0 72596 0 0006	0 66563 0 0026	0 72343 0 0007	0 71723 0.0008	0 69040 0.0015	0 90646 0 0001	0 91950 0 0001	0.98349 0 0001	0 97936 0.0001
TEFRT	0 73320 0 0005	0 64293 0 0040	0 71311 0 0009	0 70898 0.0010	0 68834 0 0016	0 91163 0.0001	0 91744 0 0001	0 98555 0 0001	0 98142 0.0001

Correlation Analysis

Spearman Correlation Coefficients / Prob > |R| under Ho: Rho=0 / N = 18

	NEFRT	TUN1	TUN2	TUN1N2	TCAPN1	TCAPN2	TCAPN1N2	TEFRT
HUN1	0.62978 0.0051	0 95864 0.0001	0,88728 0 0001	0 91667 0.0001	0 73733 0.0005	0 71872 0 0008	0 72596 0.0006	0 73320 0 0005
HUN2	0 52735 0 0245	0 79154 0.0001	0.84520 0 0001	0 86364 0 0001	0 66357 0.0027	0 66770 0 0025	0 66563 0.0026	0 64293 0 0040
HCAPN1	0 60372 0.0080	0.87822 0.0001	0 86791 0 0001	0 88946 0 0001	0 72755 0 0006	0 72136 0 0007	0 72343 0 0007	0 71311 0 0009
HCAPN2	0.59959 0.0085	0 87203 0.0001	0 85346 0.0001	0 87190 0.0001	0 72136 0 0007	0 71517 0.0008	0 71723 0 0008	0 70898 0 0010
HEFRT	0.58308 0.0111	0.89886 0.0001	0.84727 0.0001	0.87293 0.0001	0 69453 0 0014	0.68834 0.0016	0 69040 0 0015	0 68834 0 0016
NUN1	0 90749 0.0001	0.85685 0.0001	0.85788 0.0001	0 84635 0.0001	0 90749 0.0001	0 89923 0.0001	0 90646 0.0001	0 91163 0 0001
NUN2	0.89680 0.0001	0.85552 0.0001	0.90918 0.0001	0.88946 0.0001	0 93189 0.0001	0 91744 0.0001	0.91950 0 0001	0.91744 0.0001
NCAPN1	0.98555 0.0001	0.78535 0.0001	0.86997 0.0001	0.83471 0.0001	0.98142 0.0001	0.98762 0.0001	0.98349 0.0001	0.98555 0.0001
NCAPN2	0.99381 0.0001	0.75851 0.0003	0.84727 0.0001	0.81302 0.0001	0.97317 0.0001	0.98349 0.0001	0 97936 0.0001	0.98142 0.0001
NEFRT	1.00000 0.0	0.73787 0.0005	0.81631 0.0001	0.78099 0.0001	0.96285 0.0001	0.97317 0.0001	0 96904 0.0001	0 97110 0.0001
TUN1	0.73787 0.0005	1.00000 0.0	0.94634 0.0001	0 96798 0.0001	0.82869 0.0001	0 81218 0.0001	0 82456 0.0001	0 82869 0 0001
TUN2	0.81631 0.0001	0.94634 0.0001	1.00000 0 0	0.99277 0.0001	0.91125 0 0001	0.90299 0 0001	0.90506 0 0001	0.89886 0.0001
TUN1N2	0 78099 0.0001	0.96798 0.0001	0 99277 0.0001	1 00000 0.0	0 88430 0.0001	0.87293 0.0001	0.87913 0 0001	0 87397 0.0001
TCAPN1	0 96285 0.0001	0.82869 0.0001	0.91125 0.0001	0.88430 0.0001	1 00000 0.0	0.99381 0.0001	0 99587 0.0001	0.99381 0.0001
TCAPN2	0.97317 0.0001	0.81218 0.0001	0.90299 0.0001	0 87293 0.0001	0 99381 0.0001	1.00000 0.0	0 99794 0 0001	0 99587 0.0001
TCAPN1N2	0 96904 0 0001	0 82456 0.0001	0 90506 0 0001	0.87913 0.0001	0 99587 0.0001	0.99794 0.0001	1 00000 0 0	0 99794 0 0001
TEFRT	0 97110 0.0001	0.82869 0.0001	0 89886 0 0001	0 87397 0.0001	0.99381 0 0001	0 99587 0.0001	0 99794 0.0001	1 00000 0 0

Correlation Analysis

3 'WITH' Variables. HOSTVG NODEVG TOTVG
17 'VAR' Variables. HUN1 HUN2 HCAPN1 HCAPN2 HEFRT NUN1 NUN2 NCAPN1 NCAPN2 NEFRT TUN1 TUN2
TUN1N2 TCAPN1 TCAPN2 TCAPN1N2 TEFRT

Simple Statistics

Variable	N	Mean	Std Dev	Median	Minimum	Maximum
HOSTVG	18	14.444444	14 455702	7 000000	2 000000	47 000000
NODEVG	18	57 666667	49 413025	35 000000	6 000000	152 000000
TOTVG	18	72 111111	53 198727	52 500000	10 000000	158 000000
HUN1	18	37.555556	11.642833	35 000000	23 000000	61 000000
HUN2	18	55.888889	25.840891	54 500000	19 000000	113.000000
HCAPN1	18	338.944444	225.123880	249.000000	69 000000	827.000000
HCAPN2	18	188 777778	122.758757	143.500000	36 000000	446.000000
HEFRT	18	302424	362591	129673	12337	1140590
NUN1	18	44.500000	12.701320	46 000000	27 000000	73 000000
NUN2	18	72.388889	40.744742	69.500000	26.000000	190 000000
NCAPN1	18	942 222222	710.268844	687.000000	151 000000	2565.000000
NCAPN2	18	591.833333	465.025521	412.000000	80.000000	1608.000000
NEFRT	18	2813805	3679114	1054636	56230	13686523
TUN1	18	82.055556	22.367182	83.000000	50.000000	134.000000
TUN2	18	128.277778	61.622003	127.000000	56.000000	303 000000
TUN1N2	18	210.333333	83.223300	211.500000	106.000000	437.000000
TCAPN1	18	1281.166667	808.642017	1193.500000	274.000000	2824.000000
TCAPN2	18	780.611111	511.298478	720.500000	142.000000	1764.000000
TCAPN1N2	18	2061.777778	1317.423937	1889.000000	416.000000	4588 000000
TEFRT	18	5209993	5339407	3614345	180012	18474019

Correlation Analysis

Spearman Correlation Coefficients / Prob > |R| under Ho: Rho=0 / N = 18

	HUN1	HUN2	HCAPN1	HCAPN2	HEFRT	NUN1	NUN2	NCAPN1	NCAPN2
HOSTVG	0.92085 0.0001	0.71831 0.0008	0 82123 0.0001	0 83162 0.0001	0 86489 0 0001	0 51901 0.0273	0 55823 0 0161	0 49689 0 0359	0.44596 0.0636
NODEVG	0.66304 0 0027	0 52169 0 0264	0 60641 0 0076	0 60641 0 0076	0 59814 0.0087	0 91930 0.0001	0 90703 0 0001	0 96694 0 0001	0.97624 0.0001
TOTVG	0.76553 0.0002	0.61260 0 0069	0 70145 0.0012	0 70351 0 0011	0 69318 0 0014	0 91412 0 0001	0 92975 0 0001	0 96901 0 0001	0 95558 0 0001

	NEFRT	TUN1	TUN2	TUN1N2	TCAPN1	TCAPN2	TCAPN1N2	TEFRT
HOSTVG	0.41061 0.0905	0.80979 0.0001	0.73079 0.0006	0.76276 0.0002	0.55095 0.0178	0.51976 0.0270	0 52184 0.0263	0.53640 0.0217
NODEVG	0.97624 0.0001	0.79029 0.0001	0.83574 0.0001	0.81024 0.0001	0.95661 0.0001	0.95971 0 0001	0 96384 0 0001	0.96798 0.0001
TOTVG	0.94525 0.0001	0.85021 0.0001	0.89360 0.0001	0.87332 0.0001	0 97934 0 0001	0 97004 0.0001	0 97417 0 0001	0 98037 0 0001

Correlation Analysis

7 'WITH' Variables: HMSGSND HMSGREC NMSGSND NMSGREC TMSGSND TMSGREC TCOMMSG
17 'VAR' Variables: HUN1 HUN2 HCAPN1 HCAPN2 HEFRT NUN1 NUN2 NCAPN1 NCAPN2 NEFRT TUN1 TUN2 TUN1N2 TCAPN1 TCAPN2 TCAPN1N2 TEFRT

Simple Statistics

Variable	N	Mean	Std Dev	Median	Minimum	Maximum
HMSGSND	18	4.277778	2.803476	4.500000	0	9.000000
HMSGREC	18	2.611111	1.144752	2.000000	1.000000	5.000000
NMSGSND	18	8.888889	6.641128	9.000000	2.000000	32.000000
NMSGREC	18	10.555556	6.921596	10.000000	2.000000	33.000000
TMSGSND	18	13.166667	7.270003	13.000000	3.000000	35.000000
TMSGREC	18	13.166667	7.196813	13.000000	3.000000	35.000000
TCOMMSG	18	26.333333	14.418126	27.000000	6.000000	70.000000
HUN1	18	37.555556	11.642833	35.000000	23.000000	61.000000
HUN2	18	55.888889	25.840891	54.500000	19.000000	113.000000
HCAPN1	18	338.944444	225.123880	249.000000	69.000000	827.000000
HCAPN2	18	188.777778	122.758757	143.500000	36.000000	446.000000
HEFRT	18	302424	362591	129673	12337	1140590
NUN1	18	44.500000	12.701320	46.000000	27.000000	73.000000
NUN2	18	72.388889	40.744742	69.500000	26.000000	190.000000
NCAPN1	18	942.222222	710.268844	687.000000	151.000000	2565.000000
NCAPN2	18	591.833333	465.025521	412.000000	80.000000	1608.000000
NEFRT	18	2813805	3679114	1054636	56230	13686523
TUN1	18	82.055556	22.367182	83.000000	50.000000	134.000000
TUN2	18	128.277778	61.622003	127.000000	56.000000	303.000000
TUN1N2	18	210.333333	83.223300	211.500000	106.000000	437.000000
TCAPN1	18	1281.166667	808.642017	1193.500000	274.000000	2824.000000
TCAPN2	18	780.611111	511.298478	720.500000	142.000000	1764.000000
TCAPN1N2	18	2061.777778	1317.423937	1889.000000	416.000000	4588.000000
TEFRT	18	5209993	5339407	3614345	180012	18474019

Correlation Analysis

Spearman Correlation Coefficients / Prob > |R| under Ho Rho=0 / N = 18

	HUN1	HUN2	HCAPN1	HCAPN2	HEFRT	NUN1	NUN2	NCAPN1	NCAPN2
HMSGSND	0.59908 0.0086	0 63958 0 0043	0.69696 0.0013	0 70740 0 0010	0 69905 0 0012	0.06792 0.7889	0 10016 0.6925	0 13668 0 5886	0 08660 0 7326
HMSGREC	0.65912 0.0029	0 64956 -0.0035	0 69439 0.0014	0 64628 0 0038	0 63972 0 0042	0 71300 0 0009	0 70533 0 0011	0 61019 0 0072	0 62988 0 0051
NMSGSND	0 37416 0 1261	0.51796 0.0277	0 50860 0.0311	0 48259 0.0425	0 40355 0 0968	0 61043 0.0071	0 67709 0 0020	0 67085 0 0023	0 68749 0 0016
NMSGREC	0 58384 0.0110	0.78458 0.0001	0.77720 0.0001	0 77825 0.0001	0 70127 0 0012	0 36333 0.1383	0.41233 0.0891	0.52833 0.0242	0.50829 0.0313
TMSGSND	0 62091 0.0060	0.74106 0.0004	0.77842 0.0001	0.78050 0.0001	0 70370 0 0011	0.52287 0.0260	0 58745 0.0104	0.62274 0.0058	0 60094 0.0083
TMSGREC	0.65484 0 0032	0.80590 0.0001	0.79755 0.0001	0.78293 0.0001	0 70986 0.0010	0 57504 0.0125	0.62739 0 0053	0 71717 0.0008	0 70986 0.0010
TCOMMSG	0 64535 0.0038	0 80541 0.0001	0.80956 0.0001	0.80333 0.0001	0.72238 0.0007	0.54210 0.0201	0 59887 0.0086	0 65907 0.0029	0 64246 0.0040

	NEFRT	TUN1	TUN2	TUN1N2	TCAPN1	TCAPN2	TCAPN1N2	TEFRT
HMSGSND	0 05008 0.8436	0.42152 0 0815	0.37665 0.1234	0.41412 0.0875	0 20450 0.4157	0 19093 0.4479	0 18572 0 4606	0 18259 0 4683
HMSGREC	0.62222 0.0058	0.72392 0.0007	0.76001 0.0003	0.77885 0.0001	0 68893 0.0016	0.66378 0.0027	0 67799 0.0020	0.65721 0.0030
NMSGSND	0.67189 0.0023	0.50132 0.0341	0.66045 0.0029	0.63042 0.0050	0.75197 0.0003	0.72389 0.0007	0.73429 0.0005	0.70205 0.0012
NMSGREC	0 46189 0.0536	0.53465 0.0223	0.63589 0.0046	0 63549 0.0046	0 61374 0.0067	0 61163 0.0070	0 61163 0.0070	0.59476 0.0092
TMSGSND	0.57084 0.0134	0.61651 0.0064	0.71719 0.0008	0 70390 0.0011	0 71511 0.0009	0 68917 0.0016	0.69643 0 0013	0 67256 0.0022
TMSGREC	0 66706 0.0025	0.66393 0.0027	0.78502 0.0001	0.77956 0.0001	0.79859 0.0001	0 79024 0.0001	0 79650 0.0001	0.77876 0.0001
TCOMMSG	0 60717 0 0075	0.64765 0.0037	0 75870 0 0003	0 75066 0.0003	0 75248 0.0003	0 73483 0.0005	0 74417 0.0004	0.71926 0 0008

Correlation Analysis

6 'WITH' Variables R1U R1T R2U R2T R3U R3T
17 'VAR' Variables. HUN1 HUN2 HCAPN1 HCAPN2 HEFRT NUN1 NUN2 NCAPN1 NCAPN2 NEFRT TUN1 TUN2
TUN1N2 TCAPN1 TCAPN2 TCAPN1N2 TEFRT

Simple Statistics

Variable	N	Mean	Std Dev	Median	Minimum	Maximum
R1U	18	1342.900000	732.644191	1326.050000	296.700000	3444.500000
R1T	18	21311	15218	18777	3234.100000	51397
R2U	18	1439.194444	696.135651	1424.650000	610.400000	3409.300000
R2T	18	21741	15753	19021	3207.000000	53797
R3U	18	1238.461111	624.101364	1223.250000	505.800000	3022.500000
R3T	18	19770	14487	17229	2821.800000	49388
HUN1	18	37.555556	11.642833	35.000000	23.000000	61.000000
HUN2	18	55.888889	25.840891	54.500000	19.000000	113.000000
HCAPN1	18	338.944444	225.123880	249.000000	69.000000	827.000000
HCAPN2	18	188.777778	122.758757	143.500000	36.000000	446.000000
HEFRT	18	302424	362591	129673	12337	1140590
NUN1	18	44.500000	12.701320	46.000000	27.000000	73.000000
NUN2	18	72.388889	40.744742	69.500000	26.000000	190.000000
NCAPN1	18	942.222222	710.268844	687.000000	151.000000	2565.000000
NCAPN2	18	591.833333	465.025521	412.000000	80.000000	1608.000000
NEFRT	18	2813805	3679114	1054636	56230	13686523
TUN1	18	82.055556	22.367182	83.000000	50.000000	134.000000
TUN2	18	128.277778	61.622003	127.000000	56.000000	303.000000
TUN1N2	18	210.333333	83.223300	211.500000	106.000000	437.000000
TCAPN1	18	1281.166667	808.642017	1193.500000	274.000000	2824.000000
TCAPN2	18	780.611111	511.298478	720.500000	142.000000	1764.000000
TCAPN1N2	18	2061.777778	1317.423937	1889.000000	416.000000	4588.000000
TEFRT	18	5209993	5339407	3614345	180012	18474019

Correlation Analysis

Spearman Correlation Coefficients / Prob > |R| under Ho. Rho=0 / N = 18

	HUN1	HUN2	HCAPN1	HCAPN2	HEFRT	NUN1	NUN2	NCAPN1	NCAPN2
R1U	0.73113 0 0006	0 74200 0.0004	0 76883 0.0002	0 75232 0.0003	0 71930 0 0008	0 58398 0.0109	0 61610 0 0065	0 66770 0 0025	0.64912 0.0036
R1T	0 72596 0.0006	0 66563 0.0026	0.72343 0 0007	0 71723 0 0008	0 69040 0 0015	0 90646 0 0001	0 91950 0 0001	0 98349 0 0001	0 97936 0.0001
R2U	0 91520 0.0001	0 85965 0 0001	0 88854 0 0001	0 87203 0 0001	0 87203 0 0001	0 84548 0 0001	0 88854 0 0001	0 84314 0 0001	0 82250 0 0001
R2T	0.72596 0 0006	0.66563 0.0026	0 72343 0.0007	0 71723 0.0008	0 69040 0.0015	0 90646 0 0001	0 91950 0 0001	0 98349 0 0001	0.97936 0.0001
R3U	0.91464 0.0001	0.85906 0.0001	0 88591 0.0001	0 86939 0.0001	0.86939 0.0001	0 84747 0.0001	0 89004 0 0001	0.84151 0 0001	0.81982 0.0001
R3T	0.72596 0.0006	0.66563 0.0026	0.72343 0.0007	0 71723 0.0008	0 69040 0.0015	0 90646 0 0001	0 91950 0 0001	0 98349 0 0001	0.97936 0.0001

	NEFRT	TUN1	TUN2	TUN1N2	TCAPN1	TCAPN2	TCAPN1N2	TEFRT
R1U	0 62023 0.0060	0.68627 0.0017	0.74613 0.0004	0.75103 0.0003	0 71723 0.0008	0.70485 0.0011	0 71104 0.0009	0 70691 0.0010
R1T	0.96904 0.0001	0 82456 0.0001	0.90506 0.0001	0.87913 0.0001	0.99587 0.0001	0 99794 0.0001	1 00000 0 0001	0 99794 0.0001
R2U	0.78947 0.0001	0.96698 0.0001	0.99174 0.0001	0.99897 0.0001	0.89267 0.0001	0 88235 0.0001	0 88854 0 0001	0.88442 0 0001
R2T	0.96904 0.0001	0 82456 0.0001	0.90506 0.0001	0 87913 0.0001	0 99587 0.0001	0 99794 0.0001	1 00000 0 0001	0 99794 0 0001
R3U	0 78678 0 0001	0 96644 0.0001	0.99329 0.0001	0 99948 0 0001	0 89107 0 0001	0 87971 0 0001	0 88591 0 0001	0 88178 0.0001
R3T	0.96904 0.0001	0 82456 0.0001	0.90506 0.0001	0 87913 0.0001	0.99587 0.0001	0 99794 0.0001	1 00000 0 0001	0 99794 0 0001

Correlation Analysis

3 'WITH' Variables: HOSTVG NODEVG TOTVG
3 'VAR' Variables: HOSTVG NODEVG TOTVG

Simple Statistics

Variable	N	Mean	Std Dev	Median	Minimum	Maximum
HOSTVG	18	14.444444	14.455702	7.000000	2.000000	47.000000
NODEVG	18	57.666667	49.413025	35.000000	6.000000	152.000000
TOTVG	18	72.111111	53.198727	52.500000	10.000000	158.000000

Spearman Correlation Coefficients / Prob > |R| under Ho: Rho=0 / N = 18

	HOSTVG	NODEVG	TOTVG
HOSTVG	1.00000 0.0	0.44642 0.0633	0.60875 0.0073
NODEVG	0.44642 0.0633	1.00000 0.0	0.96691 0.0001
TOTVG	0.60875 0.0073	0.96691 0.0001	1.00000 0.0

Correlation Analysis

7 'WITH' Variables. HMSGSND HMSGREC NMSGSND NMSGREC TMSGSND TMSGREC TCOMMSG
3 'VAR' Variables: HOSTVG NODEVG TOTVG

Simple Statistics

Variable	N	Mean	Std Dev	Median	Minimum	Maximum
HMSGSND	18	4.277778	2 803476	4.500000	0	9 000000
HMSGREC	18	2 611111	1 144752	2 000000	1 000000	5.000000
NMSGSND	18	8 888889	6 641128	9.000000	2 000000	32 000000
NMSGREC	18	10 555556	6 921596	10.000000	2 000000	33 000000
TMSGSND	18	13.166667	7.270003	13.000000	3.000000	35.000000
TMSGREC	18	13.166667	7.196813	13.000000	3 000000	35.000000
TCOMMSG	18	26.333333	14.418126	27.000000	6 000000	70.000000
HOSTVG	18	14.444444	14.455702	7.000000	2.000000	47.000000
NODEVG	18	57.666667	49 413025	35.000000	6.000000	152 000000
TOTVG	18	72.111111	53.198727	52.500000	10.000000	158.000000

Spearman Correlation Coefficients / Prob > |R| under Ho: Rho=0 / N = 18

	HOSTVG	NODEVG	TOTVG
HMSGSND	0.73569 0.0005	0.03133 0.9018	0.19844 0.4299
HMSGREC	0.45052 0.0606	0.62505 0.0055	0 62834 0.0052
NMSGSND	0.22106 0.3780	0.66998 0.0023	0.66998 0.0023
NMSGREC	0 58317 0.0111	0.44020 0 0675	0 55632 0.0165
TMSGSND	0 58285 0.0111	0.55429 0.0170	0 64884 0.0036
TMSGREC	0 56362 0.0149	0.64894 0.0036	0.73358 0.0005
TCOMMSG	0.56665 0.0142	0.59221 0.0096	0.67949 0.0019

Correlation Analysis

6 'WITH' Variables. R1U R1T R2U R2T R3U R3T
3 'VAR' Variables: HOSTVG NODEVG TOTVG

Simple Statistics

Variable	N	Mean	Std Dev	Median	Minimum	Maximum
R1U	18	1342 900000	732 644191	1326.050000	296 700000	3444 500000
R1T	18	21311	15218	18777	3234 100000	51397
R2U	18	1439 194444	696 135651	1424.650000	610 400000	3409 300000
R2T	18	21741	15753	19021	3207 000000	53797
R3U	18	1238.461111	624.101364	1223 250000	505 800000	3022 500000
R3T	18	19770	14487	17229	2821.800000	49388
HOSTVG	18	14 444444	14.455702	7 000000	2 000000	47 000000
NODEVG	18	57 666667	49.413025	35.000000	6.000000	152.000000
TOTVG	18	72 111111	53.198727	52 500000	10.000000	158.000000

Spearman Correlation Coefficients / Prob > |R| under Ho Rho=0 / N = 18

	HOSTVG	NODEVG	TOTVG
R1U	0.64763 0.0037	0 55062 0.0179	0 64153 0.0041
R1T	0.52184 0.0263	0.96384 0.0001	0.97417 0.0001
R2U	0.76405 0 0002	0.82025 0.0001	0.88430 0.0001
R2T	0.52184 0.0263	0.96384 0.0001	0.97417 0.0001
R3U	0 76445 0.0002	0 81705 0 0001	0.88217 0.0001
R3T	0.52184 0 0263	0 96384 0.0001	0.97417 0.0001

Correlation Analysis

7 'WITH' Variables: HMSGSND HMSGREC NMSGSND NMSGREC TMSGSND TMSGREC TCOMMSG
7 'VAR' Variables: HMSGSND HMSGREC NMSGSND NMSGREC TMSGSND TMSGREC TCOMMSG

Simple Statistics

Variable	N	Mean	Std Dev	Median	Minimum	Maximum
HMSGSND	18	4.277778	2.803476	4 500000	0	9 000000
HMSGREC	18	2.611111	1 144752	2.000000	1 000000	5 000000
NMSGSND	18	8.888889	6.641128	9 000000	2 000000	32 000000
NMSGREC	18	10.555556	6.921596	10.000000	2 000000	33 000000
TMSGSND	18	13.166667	7.270003	13.000000	3.000000	35 000000
TMSGREC	18	13.166667	7.196813	13.000000	3 000000	35.000000
TCOMMSG	18	26.333333	14.418126	27 000000	6.000000	70.000000

Spearman Correlation Coefficients / Prob > |R| under Ho: Rho=0 / N = 18

	HMSGSND	HMSGREC	NMSGSND	NMSGREC	TMSGSND	TMSGREC	TCOMMSG
HMSGSND	1 00000 0.0	0 23659 0.3445	0 14459 0.5670	0 72819 0.0006	0 61176 0.0070	0 54512 0 0193	0.57976 0.0117
HMSGREC	0.23659 0.3445	1.00000 0.0	0.73290 0.0005	0.43412 0.0718	0.59169 0.0097	0 65817 0 0030	0.64338 0 0040
NMSGSND	0.14459 0.5670	0.73290 0.0005	1.00000 0.0	0.62068 0.0060	0.80544 0.0001	0 78328 0.0001	0 80544 0.0001
NMSGREC	0.72819 0.0006	0.43412 0.0718	0.62068 0.0060	1.00000 0.0	0.90573 0.0001	0.93551 0.0001	0.93278 0.0001
TMSGSND	0.61176 0.0070	0.59169 0.0097	0.80544 0.0001	0 90573 0.0001	1.00000 0.0	0.91445 0 0001	0.97599 0.0001
TMSGREC	0.54512 0 0193	0 65817 0.0030	0.78328 0.0001	0.93551 0.0001	0.91445 0.0001	1 00000 0 0	0.97114 0.0001
TCOMMSG	0.57976 0.0117	0 64338 0.0040	0.80544 0.0001	0.93278 0.0001	0.97599 0.0001	0.97114 0.0001	1.00000 0.0

Correlation Analysis

6 'WITH' Variables R1U R1T R2U R2T R3U R3T
7 'VAR' Variables HMSGSND HMSGREC NMSGSND NMSGREC TMSGSND TMSGREC TCOMMSG

Simple Statistics

Variable	N	Mean	Std Dev	Median	Minimum	Maximum
R1U	18	1342.900000	732.644191	1326.050000	296.700000	3444.500000
R1T	18	21311	15218	18777	3234 100000	51397
R2U	18	1439.194444	696.135651	1424 650000	610 400000	3409 300000
R2T	18	21741	15753	19021	3207 000000	53797
R3U	18	1238 461111	624 101364	1223 250000	505.800000	3022 500000
R3T	18	19770	14487	17229	2821 800000	49388
HMSGSND	18	4.277778	2.803476	4.500000	0	9.000000
HMSGREC	18	2.611111	1.144752	2.000000	1.000000	5 000000
NMSGSND	18	8.888889	6.641128	9.000000	2.000000	32.000000
NMSGREC	18	10.555556	6.921596	10.000000	2 000000	33.000000
TMSGSND	18	13.166667	7.270003	13.000000	3 000000	35.000000
TMSGREC	18	13.166667	7.196813	13.000000	3 000000	35.000000
TCOMMSG	18	26.333333	14.418126	27.000000	6 000000	70.000000

Spearman Correlation Coefficients / Prob > |R| under Ho. Rho=0 / N = 18

	HMSGSND	HMSGREC	NMSGSND	NMSGREC	TMSGSND	TMSGREC	TCOMMSG
R1U	0.52168 0.0264	0 62988 0.0051	0 51588 0.0284	0.72553 0.0007	0.75663 0.0003	0.81947 0 0001	0 80437 0.0001
R1T	0 18572 0 4606	0.67799 0.0020	0.73429 0.0005	0.61163 0.0070	0.69643 0 0013	0 79650 0 0001	0 74417 0.0004
R2U	0.42152 0 0815	0 76766 0.0002	0.63444 0 0047	0 65065 0.0035	0.70992 0.0010	0 79128 0 0001	0.75767 0.0003
R2T	0 18572 0 4606	0 67799 0.0020	0.73429 0.0005	0.61163 0.0070	0.69643 0.0013	0 79650 0 0001	0.74417 0.0004
R3U	0 41234 0.0890	0.76806 0.0002	0.63165 0.0049	0.64413 0.0039	0 70666 0.0010	0 78699 0 0001	0.75442 0.0003
R3T	0.18572 0 4606	0.67799 0.0020	0 73429 0.0005	0.61163 0.0070	0.69643 0.0013	0.79650 0 0001	0 74417 0.0004

Correlation Analysis

6 'WITH' Variables: R1U R1T R2U R2T R3U R3T
6 'VAR' Variables: R1U R1T R2U R2T R3U R3T

Simple Statistics

Variable	N	Mean	Std Dev	Median	Minimum	Maximum
R1U	18	1342.900000	732.644191	1326 050000	296 700000	3444 500000
R1T	18	21311	15218	18777	3234 100000	51397
R2U	18	1439 194444	696.135651	1424 650000	610 400000	3409 300000
R2T	18	21741	15753	19021	3207 000000	53797
R3U	18	1238 461111	624.101364	1223 250000	505.800000	3022 500000
R3T	18	19770	14487	17229	2821 800000	49388

Spearman Correlation Coefficients / Prob > |R| under Ho: Rho=0 / N = 18

	R1U	R1T	R2U	R2T	R3U	R3T
R1U	1.00000 0.0	0.71104 0.0009	0 75026 0.0003	0.71104 0.0009	0.75168 0.0003	0 71104 0 0009
R1T	0.71104 0.0009	1.00000 0.0	0.88854 0.0001	1.00000 0.0001	0.88591 0.0001	1.00000 0.0001
R2U	0.75026 0.0003	0.88854 0.0001	1.00000 0.0	0.88854 0.0001	0.99948 0.0001	0.88854 0.0001
R2T	0.71104 0.0009	1.00000 0.0001	0.88854 0.0001	1.00000 0.0	0.88591 0.0001	1.00000 0.0001
R3U	0.75168 0.0003	0.88591 0.0001	0.99948 0.0001	0.88591 0.0001	1.00000 0.0	0.88591 0.0001
R3T	0 71104 0.0009	1.00000 0.0001	0.88854 0.0001	1.00000 0.0001	0.88591 0.0001	1.00000 0.0

Correlation Analysis

6 'WITH' Variables: R1U R1T R2U R2T R3U R3T
6 'VAR' Variables: Q5 Q6 Q7 Q8 Q9 Q10

Simple Statistics

Variable	N	Mean	Std Dev	Median	Minimum	Maximum
R1U	18	1342.900000	732.644191	1326.050000	296.700000	3444.500000
R1T	18	21311	15218	18777	3234.100000	51397
R2U	18	1439.194444	696.135651	1424.650000	610.400000	3409.300000
R2T	18	21741	15753	19021	3207.000000	53797
R3U	18	1238.461111	624.101364	1223.250000	505.800000	3022.500000
R3T	18	19770	14487	17229	2821.800000	49388
Q5	18	13.000000	1.455214	14.000000	10.000000	14.000000
Q6	18	13.055556	1.696787	14.000000	10.000000	15.000000
Q7	18	14.444444	1.199128	15.000000	12.000000	16.000000
Q8	18	14.166667	1.424574	15.000000	12.000000	16.000000
Q9	18	10.888889	3.027111	10.000000	5.000000	16.000000
Q10	18	4.222222	1.437136	4.000000	3.000000	8.000000

Spearman Correlation Coefficients / Prob > |R| under Ho: Rho=0 / N = 18

	Q5	Q6	Q7	Q8	Q9	Q10
R1U	0.02002 0.9371	-0.07278 0.7741	-0.08178 0.7470	-0.14768 0.5587	0.17367 0.4907	-0.08675 0.7322
R1T	-0.44519 0.0641	-0.49013 0.0389	-0.51337 0.0293	-0.58184 0.0113	0.65039 0.0035	-0.51552 0.0285
R2U	-0.26499 0.2879	-0.36845 0.1325	-0.37708 0.1229	-0.39752 0.1023	0.43246 0.0731	-0.37673 0.1233
R2T	-0.44519 0.0641	-0.49013 0.0389	-0.51337 0.0293	-0.58184 0.0113	0.65039 0.0035	-0.51552 0.0285
R3U	-0.26513 0.2877	-0.36182 0.1401	-0.37045 0.1302	-0.39161 0.1080	0.42587 0.0780	-0.36824 0.1327
R3T	-0.44519 0.0641	-0.49013 0.0389	-0.51337 0.0293	-0.58184 0.0113	0.65039 0.0035	-0.51552 0.0285

APPENDIX H

VARIABLES USED IN THE REGRESSION ANALYSES

VARIABLE NAMES WITH DESCRIPTIONS USED IN THE REGRESSION ANALYSIS

VARIABLE NAME	DESCRIPTION	METRIC
APLNO	Application Number	None
APLNAME	Application Name	None
Q5	Question 5 in the Questionnaire	Subjective
Q6	Question 6 in the Questionnaire	"
Q7	Question 7 in the Questionnaire	"
Q8	Question 8 in the Questionnaire	"
Q9	Question 9 in the Questionnaire	"
Q10	Question 10 in the Questionnaire	"
HEXELNS	Host Executable Lines	Size
HBLKLNS	Host Blank Lines	"
HCMTLNS	Host Commented Lines	"
NEXELNS	Node Executable Lines	"
NBLKLNS	Node Blank Lines	"
NCMTLNS	Node Commented Lines	"
TEXELNS	Total Executable Lines	"
TBLKLNS	Total Blank Lines	"
TCMTLNS	Total Commented Lines	"
TOTLNS	Total Lines in an Application	"
HUN1	Host Unique Operators	Software Science
HUN2	Host Unique Operands	"
HCAPN1	Host Total Operators	"
HCAPN2	Host Total Operands	"

HEFRT	Host Effort (E)	"
NUN1	Node Unique Operators	"
NUN2	Node Unique Operands	"
NCAPN1	Node Total Operators	"
NCAPN2	Node Total Operands	"
NEFRT	Node Effort (E)	"
TUN1	Total Unique Operators	"
TUN2	Total Unique Operands	"
TUN1N2	Total Unique Operators & Operands	"
TCAPN1	Total Operators	"
TCAPN2	Total Operands	"
TCAPN1N2	Total Operators & Operands	"
TEFRT	Total Effort (E)	"
HOSTVG	Host Cyclomatic Complexity	Cyclomatic Complexity
NODEVG	Node Cyclomatic Complexity	"
TOTVG	Total Cyclomatic Complexity	"
HMSGSND	Host Message Send statements	Communication Complexity
HMSGREC	Host Message Receive statements	"
NMSGSND	Node Message Send statements	"
NMSGREC	Node Message Receive statements	"
TMSGSND	Total Message Send statements	"
TMSGREC	Total Message Receive statements	"
TCOMMSG	Total Message statements (Send & Receive)	"
R1U	Unique Operators + Unique Operands	Residual Complexity
R1T	Total Operators + Total Operands	"

R2U	Host Unique Operators & Operands + Node Unique Operators & Operands	"
R2T	Host Total Operators & Operands + Node Total Operators & Operands	
R3U	Host Unique Operators + Host Unique Operands + Node Unique Operators + Node Unique Operands	"
R3T	Host Total Operators + Host Total Operands + Node Total Operators + Node Total Operands	"

APPENDIX I

REGRESSION ANALYSES

Stepwise Procedure for Dependent Variable Q9

Step 1 Variable NEXELNS Entered R-square = 0.33814370 C(p) = 7 55116459

	DF	Sum of Squares	Mean Square	F	Prob>F
Regression	1	52 67527364	52 67527364	8 17	0 0114
Error	16	103 10250414	6 44390651		
Total	17	155.77777778			

Variable	Parameter Estimate	Standard Error	Type II Sum of Squares	F	Prob>F
INTERCEP	8 49929118	1 02787937	440 58565215	68 37	0 0001
NEXELNS	0 00859739	0 00300703	52 67527364	8 17	0 0114

Bounds on condition number. 1, 1

Step 2 Variable HCMTLNS Entered R-square = 0 37205845 C(p) = 8 44684259

	DF	Sum of Squares	Mean Square	F	Prob>F
Regression	2	57 95843885	28 97921943	4 44	0 0305
Error	15	97 81933892	6 52128926		
Total	17	155 77777778			

Variable	Parameter Estimate	Standard Error	Type II Sum of Squares	F	Prob>F
INTERCEP	9 17381289	1 27703908	336 53082011	51 60	0 0001
HCMTLNS	-0.00801976	0.00891006	5 28316521	0 81	0 3823
NEXELNS	0.00890687	0.00304451	55 81473936	8 56	0 0104

Bounds on condition number 1 01292, 4 051678

Step 3 Variable HCMTLNS Removed R-square = 0 33814370 C(p) = 7 55116459

	DF	Sum of Squares	Mean Square	F	Prob>F
Regression	1	52 67527364	52 67527364	8 17	0 0114
Error	16	103 10250414	6 44390651		
Total	17	155 77777778			

Variable	Parameter Estimate	Standard Error	Type II Sum of Squares	F	Prob>F
INTERCEP	8 49929118	1 02787937	440 58565215	68 37	0 0001
NEXELNS	0 00859739	0 00300703	52 67527364	8 17	0 0114

Bounds on condition number 1, 1

All variables left in the model are significant at the 0 1500 level
No other variable met the 0 5000 significance level for entry into the model

Summary of Stepwise Procedure for Dependent Variable Q9

Step	Variable Entered	Removed	Number In	Partial R**2	Model R**2	C(p)	F	Prob>F
1	NEXELNS		1	0 3381	0 3381	7.5512	8 1744	0.0114
2	HCMTLNS		2	0 0339	0 3721	8 4468	0 8101	0 3823
3		HCMTLNS	1	0.0339	0 3381	7 5512	0 8101	0 3823

Obs	Dep Var Q9	Predict Value	Std Err Predict	Lower95% Mean	Upper95% Mean	Lower95% Predict	Upper95% Predict	Residual	Std Err Residual	Student Residual	-2-1-0 1 2	Cook's D
1	5 0000	8 9979	0.892	7.1073	10 8886	3 2942	14.7017	-3 9979	2 377	-1 682	***| |	0 199
2	5 0000	9 1785	0 846	7 3849	10 9721	3 5061	14 8509	-4 1785	2 393	-1 746	***| |	0 190
3	10 0000	12 2735	0 770	10 6417	13 9054	6 6502	17.8968	-2 2735	2 419	-0 940	*| |	0 045
4	10 0000	12 4025	0.799	10 7089	14 0961	6 7610	18 0440	-2 4025	2 409	-0 997	*| |	0 055
5	10 0000	12 3509	0 787	10 6824	14 0194	6 7169	17 9850	-2 3509	2 413	-0 974	*| |	0 050
6	10 0000	10 0038	0.674	8 5757	11 4319	4 4362	15 5714	-0 00384	2 447	-0 002	| |	0 000
7	10 0000	10 5197	0 612	9 2221	11 8173	4 9841	16 0552	-0 5197	2 464	-0 211	| |	0 001
8	10 0000	9 3762	0 799	7 6831	11 0694	3 7348	15 0176	0 6238	2 410	0 259	| |	0 004
9	10 0000	9 3762	0 799	7 6831	11 0694	3 7348	15 0176	0 6238	2 410	0 259	| |	0 004
10	10 0000	9 4966	0 771	7 8612	11 1320	3 8723	15 1209	0 5034	2 418	0 208	| |	0 002
11	10 0000	9 1527	0 852	7 3455	10 9599	3 4760	14 8294	0 8473	2 391	0 354	| |	0 008
12	15 0000	11 4998	0.635	10 1529	12 8466	5 9525	17 0471	3 5002	2 458	1 424	| |** |	0 068
13	15 0000	9 7889	0 711	8 2809	11 2969	4 2003	15 3775	5 2111	2 437	2 139	| |**** |	0 195
14	13 0000	12.8066	0.899	10 9012	14 7120	7 0979	18 5153	0 1934	2 374	0.081	| |	0 000
15	13 0000	12 7722	0.890	10 8857	14 6587	7 0698	18 4746	0 2278	2.377	0 096	| |	0 001
16	14 0000	10 6486	0 604	9.3678	11.9295	5 1170	16 1803	3 3514	2 466	1 359	| |** |	0 055
17	16.0000	15.3342	1.666	11 8026	18.8659	8 8975	21 7709	0 6658	1 915	0 348	| |	0 046
18	10.0000	10 0210	0.671	8.5988	11 4433	4 4549	15.5871	-0 0210	2 448	-0 009	| |	0 000

Sum of Residuals	0
Sum of Squared Residuals	103 1025
Predicted Resid SS (Press)	125 5567

Stepwise Procedure for Dependent Variable Q9

Step 1 Variable NCAPN2 Entered R-square = 0.43424720 C(p) = -1 26424364

	DF	Sum of Squares	Mean Square	F	Prob>F
Regression	1	67 64606356	67.64606356	12 28	0 0029
Error	16	88 13171422	5.50823214		
Total	17	155 77777778			

Variable	Parameter Estimate	Standard Error	Type II Sum of Squares	F	Prob>F
INTERCEP	8 35014180	0 91149916	462 26114097	83 92	0 0001
NCAPN2	0.00428963	0.00122407	67.64606356	12 28	0 0029

Bounds on condition number. 1, 1

Step 2 Variable TEFRT Entered R-square = 0.53203399 C(p) = -1 46553862

	DF	Sum of Squares	Mean Square	F	Prob>F
Regression	2	82 87907323	41 43953662	8 53	0 0034
Error	15	72 89870454	4 85991364		
Total	17	155 77777778			

Variable	Parameter Estimate	Standard Error	Type II Sum of Squares	F	Prob>F
INTERCEP	7.24089834	1 06093990	226.37678485	46.58	0 0001
NCAPN2	0 01308687	0 00510028	31 99726518	6 58	0 0215
TEFRT	-0.00000079	0.00000044	15 23300967	3 13	0 0970

Bounds on condition number 19.67711, 78 70846

Step 3 Variable NCAPN1 Entered R-square = 0.55461989 C(p) = 0 02602637

	DF	Sum of Squares	Mean Square	F	Prob>F
Regression	3	86 39745469	28.79915156	5 81	0 0085
Error	14	69 38032309	4 95573736		
Total	17	155.77777778			

Variable	Parameter Estimate	Standard Error	Type II Sum of Squares	F	Prob>F
INTERCEP	6 01882945	1 80315164	55 21637037	11 14	0 0049
NCAPN1	0.01075275	0 01276152	3 51838145	0 71	0 4136
NCAPN2	0 00276554	0 01328819	0 21465335	0.04	0 8381
TEFRT	-0 00000132	-0 00000078	14.28201509	2 88	0 1117

Bounds on condition number 281 8324, 1416 923

Step 4 Variable NCAPN2 Removed R-square = 0.55324195 C(p) = -1 94295446

	DF	Sum of Squares	Mean Square	F	Prob>F
Regression	2	86.18280133	43 09140067	9 29	0 0024
Error	15	69 59497644	4 63966510		
Total	17	155 77777778			

Variable	Parameter Estimate	Standard Error	Type II Sum of Squares	F	Prob>F
INTERCEP	5 80278985	1 42659130	76 76463306	16 55	0 0010
NCAPN1	0 01320108	0 00478585	35 30099329	7 61	0 0146
TEFRT	-0 00000141	0 00000064	22 79685965	4 91	0 0425

Bounds on condition number. 42 33763, 169.3505

Step 5 Variable NUN2 Entered R-square = 0 60712685 C(p) = -1 15596661

	DF	Sum of Squares	Mean Square	F	Prob>F
Regression	3	94 57687215	31 52562405	7 21	0 0037
Error	14	61 20090563	4 37149326		
Total	17	155 77777778			

Variable	Parameter Estimate	Standard Error	Type II Sum of Squares	F	Prob>F
INTERCEP	6 52465750	1 47949560	85.01937209	19 45	0 0006
NUN2	-0 02857018	0 02061777	8 39407081	1 92	0 1875
NCAPN1	0 01499220	0 00482195	42.25852159	9 67	0 0077
TEFRT	-0 00000148	0 00000062	24.81746324	5 68	0 0319

Bounds on condition number 45 61534, 272 8355

Step 6 Variable NUN2 Removed R-square = 0 55324195 C(p) = -1 94295446

	DF	Sum of Squares	Mean Square	F	Prob>F
Regression	2	86 18280133	43 09140067	9 29	0 0024
Error	15	69 59497644	4 63966510		
Total	17	155 77777778			

Variable	Parameter Estimate	Standard Error	Type II Sum of Squares	F	Prob>F
INTERCEP	5 80278985	1 42659130	76 76463306	16 55	0 0010
NCAPN1	0 01320108	0 00478585	35 30099329	7 61	0 0146
TEFRT	-0 00000141	0 00000064	22 79685965	4 91	0 0425

Bounds on condition number 42 33763, 169 3505

All variables left in the model are significant at the 0 1500 level
No other variable met the 0 5000 significance level for entry into the model

Summary of Stepwise Procedure for Dependent Variable Q9

Step	Variable Entered	Removed	Number In	Partial R**2	Model R**2	C(p)	F	Prob>F
1	NCAPN2		1	0 4342	0 4342	-1 2642	12 2809	0 0029
2	TEFRT		2	0 0978	0 5320	-1 4655	3 1344	0 0970
3	NCAPN1		3	0 0226	0 5546	0 0260	0 7100	0 4136
4		NCAPN2	2	0 0014	0 5532	-1 9430	0 0433	0 8381
5	NUN2		3	0 0539	0 6071	-1 1560	1 9202	0 1875
6		NUN2	2	0 0539	0 5532	-1 9430	1 9202	0 1875

Obs	Dep Var Q9	Predict Value	Std Err Predict	Lower95% Mean	Upper95% Mean	Lower95% Predict	Upper95% Predict	Residual	Std Err Residual	Student Residual	-2-1-0 1 2	Cook's D
1	5.0000	7.5421	0 942	5 5342	9 5500	2 5312	12 5531	-2 5421	1.937	-1 312	*|	0 136
2	5.0000	8 4736	0 761	6 8519	10 0953	3 6045	13 3427	-3 4736	2 015	-1 724	**|	0 141
3	10.0000	12 5446	0 802	10 8350	14 2543	7 6455	17 4437	-2 5446	1 999	-1 273	*|	0 087
4	10.0000	11.2132	0.516	10 1131	12 3133	6 4921	15 9342	-1 2132	2 091	-0 580	*	0 007
5	10 0000	12 5081	1.080	10 2058	14 8104	7 3721	17.6441	-2 5081	1 864	-1 346	*|	0 203
6	10 0000	8 4294	0.847	6 6251	10 2337	3 4965	13 3623	1 5706	1 981	0 793	*	0 038
7	10 0000	8 7562	0 925	6 7840	10 7283	3 7594	13 7529	1 2438	1 945	0 639	*	0 031
8	10 0000	8 4889	0 754	6 8816	10 0961	3 6246	13 3531	1 5111	2 018	0 749	*	0 026
9	10 0000	9 2686	0 665	7 8515	10 6857	4 4637	14 0734	0 7314	2 049	0 357		0 004
10	10 0000	10 7476	0.788	9 0675	12 4276	5 8587	15 6364	-0 7476	2 005	-0 373		0 007
11	10 0000	8 7900	0 715	7 2659	10 3140	3 9525	13 6274	1 2100	2 032	0 596	*	0 015
12	15 0000	13.7175	0.864	11 8759	15 5591	8 7708	18 6642	1 2825	1 973	0 650	*	0 027
13	15 0000	11 0286	0.809	9.3039	12.7534	6 1242	15 9330	3 9714	1 996	1 989	***	0 217
14	13 0000	13 6519	0.848	11 8442	15 4597	8 7177	18 5861	-0 6519	1 980	-0 329		0 007
15	13 0000	14 5703	1.044	12 3448	16.7957	9 4682	19 6723	-1 5703	1 884	-0.833	*	0 071
16	14.0000	12 6316	0.921	10.6690	14 5942	7 6386	17 6246	1 3684	1 947	0.703	*	0 037
17	16 0000	13 5933	1 509	10.3780	16.8087	7 9883	19 1984	2 4067	1 538	1.565	***	0 786
18	10 0000	10.0447	0.571	8.8277	11 2616	5 2950	14 7943	-0 0447	2 077	-0 022		0 000

Sum of Residuals	0
Sum of Squared Residuals	69.5950
Predicted Resid SS (Press)	112 8749

Stepwise Procedure for Dependent Variable Q9

Step 1 Variable NODEVG Entered R-square = 0 42839393 C(p) = 2 17330924

	DF	Sum of Squares	Mean Square	F	Prob>F
Regression	1	66.73425471	66 73425471	11 99	0 0032
Error	16	89 04352307	5 56522019		
Total	17	155 77777778			

Variable	Parameter Estimate	Standard Error	Type II Sum of Squares	F	Prob>F
INTERCEP	8 57664654	0 86893077	542 18441332	97 42	0 0001
NODEVG	0.04009669	0 01157911	66 73425471	11 99	0 0032

Bounds on condition number 1, 1

Step 2 Variable HOSTVG Entered R-square = 0 46986168 C(p) = 3 00000000

	DF	Sum of Squares	Mean Square	F	Prob>F
Regression	2	73 19400802	36 59700401	6 65	0 0086
Error	15	82 58376975	5 50558465		
Total	17	155 77777778			

Variable	Parameter Estimate	Standard Error	Type II Sum of Squares	F	Prob>F
INTERCEP	9.10640891	0 99304760	462.97506998	84.09	0 0001
HOSTVG	-0 04298327	0 03968197	6.45975332	1.17	0 2958
NODEVG	0 04167658	0 01160890	70 95870491	12.89	0 0027

Bounds on condition number 1 016039, 4 064155

Step 3 Variable HOSTVG Removed R-square = 0 42839393 C(p) = 2.17330924

	DF	Sum of Squares	Mean Square	F	Prob>F
Regression	1	66 73425471	66 73425471	11 99	0 0032
Error	16	89 04352307	5 56522019	.	
Total	17	155 77777778			

Variable	Parameter Estimate	Standard Error	Type II Sum of Squares	F	Prob>F
INTERCEP	8 57664654	0 86893077	542 18441332	97 42	0 0001
NODEVG	0 04009669	0 01157911	66 73425471	11.99	0 0032

Bounds on condition number 1, 1

All variables left in the model are significant at the 0 1500 level
No other variable met the 0 5000 significance level for entry into the model

Summary of Stepwise Procedure for Dependent Variable Q9

Step	Variable Entered	Removed	Number In	Partial R**2	Model R**2	C(p)	F	Prob>F
1	NODEVG		1	0 4284	0 4284	2 1733	11 9913	0 0032
2	HOSTVG		2	0 0415	0 4699	3 0000	1 1733	0 2958
3		HOSTVG	1	0 0415	0.4284	2 1733	1 1733	0 2958

Obs	Dep Var Q9	Predict Value	Std Err Predict	Lower95% Mean	Upper95% Mean	Lower95% Predict	Upper95% Predict	Residual	Std Err Residual	Student Residual	-2-1-0 1 2	Cook s D
1	5 0000	8.8172	0.817	7 0858	10 5487	3 5250	14 1095	-3 8172	2 213	-1 725	***| |	0 203
2	5 0000	9 0578	0 767	7 4312	10 6845	3.7989	14 3167	-4 0578	2 231	-1 819	***| |	0 196
3	10 0000	11 1829	0 562	9 9905	12 3753	6 0418	16 3241	-1 1829	2 291	-0 516	*| |	0 008
4	10 0000	11.0626	0 558	9 8791	12 2462	5 9235	16 2018	-1 0626	2 292	-0 464	| |	0 006
5	10 0000	12.8670	0 797	11 1771	14 5569	7 5882	18 1458	-2 8670	2 220	-1 291	**| |	0 107
6	10 0000	9 7795	0 642	8 4192	11.1399	4 5968	14 9623	0 2205	2 270	0 097	| |	0 000
7	10 0000	10 1805	0 592	8 9245	11 4365	5 0242	15 3368	-0 1805	2 283	-0 079	| |	0 000
8	10 0000	9 2583	0.729	7 7137	10 8029	4 0242	14 4924	0 7417	2 244	0 331	| |	0 006
9	10 0000	9 2583	0 729	7 7137	10 8029	4 0242	14 4924	0 7417	2 244	0 331	| |	0 006
10	10 0000	9 5791	0 672	8 1534	11 0047	4 3788	14 7793	0.4209	2 261	0 186	| |	0 002
11	10 0000	9 1781	0 744	7 6013	10 7549	3 9344	14 4218	0 8219	2 239	0 367	| |	0 007
12	15 0000	14 1100	1.084	11 8126	16 4073	8 6066	19 6134	0.8900	2 095	0 425	| |	0 024
13	15 0000	9.7395	0.648	8.3666	11 1123	4.5535	14 9254	5 2605	2 268	2 319	| |****	0 219
14	13.0000	13.6689	0.977	11.5987	15 7392	8 2564	19.0815	-0 6689	2 147	-0 311	| |	0 010
15	13 0000	13 6689	0 977	11 5987	15 7392	8 2564	19 0815	-0.6689	2 147	-0 311	| |	0 010
16	14 0000	10 2607	0.585	9 0208	11.5006	5 1083	15 4131	3.7393	2 285	1 636	| |***	0.088
17	16.0000	14 6713	1 226	12 0730	17 2697	9 0356	20 3070	1 3287	2 016	0 659	| |*	0 080
18	10.0000	9.6593	0 660	8.2607	11 0579	4 4664	14.8521	0.3407	2 265	0 150	| |	0 001

Sum of Residuals	0
Sum of Squared Residuals	89.0435
Predicted Resid SS (Press)	109 4661

Stepwise Procedure for Dependent Variable Q9

Step 1 Variable NMSGSND Entered R-square = 0.33152328 C(p) = 0.98429269

	DF	Sum of Squares	Mean Square	F	Prob>F
Regression	1	51.64396035	51.64396035	7.94	0 0124
Error	16	104.13381743	6 50836359		
Total	17	155 77777778			

Variable	Parameter Estimate	Standard Error	Type II Sum of Squares	F	Prob>F
INTERCEP	8 55601660	1 02344210	454 87124703	69 89	0 0001
NMSGSND	0.26244813	0.09316861	51 64396035	7 94	0 0124

Bounds on condition number. 1, 1

Step 2 Variable HMSGREC Entered R-square = 0 37274166 C(p) = 2 06035888

	DF	Sum of Squares	Mean Square	F	Prob>F
Regression	2	58 06486767	29 03243384	4.46	0 0303
Error	15	97 71291011	6.51419401		
Total	17	155.77777778			

Variable	Parameter Estimate	Standard Error	Type II Sum of Squares	F	Prob>F
INTERCEP	7.33957755	1.59674492	137 63590146	21 13	0.0003
HMSGREC	0.56061540	0.56467319	6 42090732	0 99	0.3365
NMSGSND	0.23461675	0.09733450	37 84818715	5.81	0 0292

Bounds on condition number 1 090449, 4 361798

Step 3 Variable HMSGREC Removed R-square = 0.33152328 C(p) = 0.98429269

	DF	Sum of Squares	Mean Square	F	Prob>F
Regression	1	51 64396035	51 64396035	7 94	0 0124
Error	16	104 13381743	6.50836359		
Total	17	155 77777778			

Variable	Parameter Estimate	Standard Error	Type II Sum of Squares	F	Prob>F
INTERCEP	8 55601660	1 02344210	454.87124703	69 89	0 0001
NMSGSND	0 26244813	0 09316861	51 64396035	7.94	0 0124

Bounds on condition number 1, 1

All variables left in the model are significant at the 0 1500 level
No other variable met the 0 5000 significance level for entry into the model

Summary of Stepwise Procedure for Dependent Variable Q9

Step	Variable Entered	Removed	Number In	Partial R**2	Model R**2	C(p)	F	Prob>F
1	NMSGSND		1	0 3315	0 3315	0.9843	7 9350	0 0124
2	HMSGREC		2	0.0412	0.3727	2 0604	0 9857	0 3365
3		HMSGREC	1	0.0412	0 3315	0.9843	0 9857	0 3365

Obs	Dep Var Q9	Predict Value	Std Err Predict	Lower95% Mean	Upper95% Mean	Lower95% Predict	Upper95% Predict	Residual	Std Err Residual	Student Residual	-2-1-0 1 2	Cook's D
1	5 0000	9.6058	0 754	8.0067	11.2050	3 9662	15 2455	-4 6058	2 437	-1 890	***| |	0.171
2	5.0000	9 3434	0.814	7 6178	11 0690	3.6666	15 0202	-4 3434	2 418	-1 796	***| |	0.183
3	10.0000	9 0809	0 879	7 2165	10 9454	3.3604	14.8014	0 9191	2 395	0 384	| |	0 010
4	10 0000	11 1805	0.610	9.8870	12 4740	5.6198	16 7412	-1 1805	2 477	-0 477	| |	0 007
5	10 0000	11 1805	0 610	9 8870	12 4740	5.6198	16.7412	-1 1805	2 477	-0 477	| |	0 007
6	10 0000	10 9180	0.601	9 6431	12 1930	5 3616	16.4745	-0 9180	2 479	-0 370	| |	0.004
7	10 0000	10 9180	0 601	9 6431	12 1930	5 3616	16 4745	-0 9180	2 479	-0 370	| |	0 004
8	10 0000	10 3932	0.627	9 0650	11 7213	4 8243	15 9620	-0 3932	2 473	-0 159	| |	0 001
9	10.0000	10.3932	0 627	9.0650	11 7213	4.8243	15 9620	-0 3932	2 473	-0 159	| |	0 001
10	10 0000	11 1805	0 610	9 8870	12 4740	5 6198	16 7412	-1 1805	2 477	-0 477	| |	0 007
11	10 0000	9 8683	0 702	8 3800	11 3565	4 2591	15 4775	0 1317	2 453	0 054	| |	0 000
12	15 0000	11 1805	0 610	9 8870	12 4740	5 6198	16 7412	3 8195	2 477	1 542	| |***	0 072
13	15 0000	9 3434	0 814	7 6178	11 0690	3 6666	15 0202	5 6566	2 418	2 340	| |****	0.310
14	13 0000	11 7054	0 668	10.2903	13 1205	6 1152	17.2956	1 2946	2 462	0 526	| |*	0 010
15	13 0000	11 4429	0 633	10.1018	12 7841	5 8710	17.0149	1 5571	2 471	0 630	| |*	0.013
16	14.0000	11 7054	0 668	10 2903	13 1205	6 1152	17.2956	2 2946	2.462	0 932	| |*	0.032
17	16 0000	16 9544	2.236	12.2151	21 6936	9 7635	24 1453	-0 9544	1 229	-0 777	| *| |	0.998
18	10.0000	9 6058	0.754	8 0067	11.2050	3.9662	15.2455	0.3942	2 437	0 162	| | |	0 001

Sum of Residuals	0
Sum of Squared Residuals	104 1338
Predicted Resid SS (Press)	140 8333

Stepwise Procedure for Dependent Variable Q9

Step 1 Variable R2T Entered R-square = 0.34424671 C(p) = 5.56976302

	DF	Sum of Squares	Mean Square	F	Prob>F
Regression	1	53 62598723	53 62598723	8 40	0 0105
Error	16	102.15179055	6 38448691		
Total	17	155.77777778			

Variable	Parameter Estimate	Standard Error	Type II Sum of Squares	F	Prob>F
INTERCEP	8.43771930	1 03441233	424 80417165	66 54	0 0001
R2T	0 00011275	0 00003890	53.62598723	8 40	0 0105

Bounds on condition number: 1, 1

Step 2 Variable R1U Entered R-square = 0.48454134 C(p) = 3 38292532

	DF	Sum of Squares	Mean Square	F	Prob>F
Regression	2	75 48077246	37 74038623	7.05	0 0069
Error	15	80.29700532	5.35313369		
Total	17	155.77777778			

Variable	Parameter Estimate	Standard Error	Type II Sum of Squares	F	Prob>F
INTERCEP	9.82834273	1 17082632	377.21045288	70 47	0 0001
R1U	-0 00206541	0 00102220	21 85478523	4 08	0 0616
R2T	0.00017636	0.00004754	73.66705577	13 76	0 0021

Bounds on condition number. 1.781157, 7.124629

Step 3 Variable R3T Entered R-square = 0.59054829 C(p) = 2.21934082

	DF	Sum of Squares	Mean Square	F	Prob>F
Regression	3	91.99430002	30.66476667	6.73	0.0049
Error	14	63.78347776	4.55596270		
Total	17	155.77777778			

Variable	Parameter Estimate	Standard Error	Type II Sum of Squares	F	Prob>F
INTERCEP	6 50429403	2.05307476	45 72681605	10.04	0 0068
R1U	-0 00250758	0.00097120	30 37163707	6 67	0 0217
R2T	0.01513110	0.00785518	16 90478139	3 71	0.0746
R3T	-0 01624706	0.00853385	16.51352756	3 62	0 0777

Bounds on condition number 57135 9, 342506 9

Step 4 Variable R1T Entered R-square = 0.62861921 C(p) = 3 08318342

	DF	Sum of Squares	Mean Square	F	Prob>F
Regression	4	97 92490391	24 48122598	5 50	0 0081
Error	13	57 85287387	4 45022107		
Total	17	155.77777778			

Variable	Parameter Estimate	Standard Error	Type II Sum of Squares	F	Prob>F
INTERCEP	5 31019342	2 27755082	24 19168466	5 44	0 0365
R1U	-0 00126333	0 00144328	3 40967621	0 77	0 3973
R1T	-0 00210321	0 00182190	5 93060389	1 33	0 2691
R2T	0 02135799	0 00945342	22 71562703	5 10	0 0417
R3T	-0 02085151	0 00932980	22 22860275	4 99	0 0436

Bounds on condition number 84717 59, 629776 6

Step 5 Variable R1U Removed R-square = 0 60673113 C(p) = 1.73639327

	DF	Sum of Squares	Mean Square	F	Prob>F
Regression	3	94 51522771	31.50507590	7.20	0 0037
Error	14	61 26255007	4 37589643		
Total	17	155.77777778			

Variable	Parameter Estimate	Standard Error	Type II Sum of Squares	F	Prob>F
INTERCEP	4 51955424	2 07324682	20.79487058	4 75	0.0468
R1T	-0 00329415	0 00120151	32.89256475	7 52	0 0159
R2T	0 02378698	0.00896118	30.83299237	7 05	0 0189
R3T	-0 02228464	0 00910799	26 19587066	5 99	0 0282

Bounds on condition number. 77417 73, 439059 4

All variables left in the model are significant at the 0 1500 level
No other variable met the 0 5000 significance level for entry into the model

Summary of Stepwise Procedure for Dependent Variable Q9

Step	Variable Entered	Variable Removed	Number In	Partial R**2	Model R**2	C(p)	F	Prob>F
1	R2T		1	0 3442	0.3442	5 5698	8 3994	0.0105
2	R1U		2	0 1403	0.4845	3 3829	4 0826	0.0616
3	R3T		3	0 1060	0.5905	2 2193	3.6246	0 0777
4	R1T		4	0 0381	0.6286	3.0832	1 3327	0 2691
5		R1U	3	0 0219	0 6067	1 7364	0.7662	0 3973

Obs	Dep Var Q9	Predict Value	Std Err Predict	Lower95% Mean	Upper95% Mean	Lower95% Predict	Upper95% Predict	Residual	Std Err Residual	Student Residual	-2-1-0 1 2	Cook's D
1	5.0000	7.2680	1.110	4.8876	9.6485	2.1890	12.3470	-2.2680	1.773	-1.279	*|	0.160
2	5.0000	8.0226	0.995	5.8883	10.1569	3.0542	12.9909	-3.0226	1.840	-1.643	**|	0.197
3	10.0000	12.3061	0.700	10.8045	13.8076	7.5749	17.0372	-2.3061	1.971	-1.170	*|	0.043
4	10.0000	10.6364	0.958	8.5823	12.6905	5.7019	15.5709	-0.6364	1.860	-0.342	|	0.008
5	10.0000	9.7871	1.771	5.9877	13.5865	3.9079	15.6663	0.2129	1.113	0.191	|	0.023
6	10.0000	10.0509	0.674	8.6062	11.4955	5.3374	14.7643	-0.0509	1.980	-0.026	|	0.000
7	10.0000	10.5974	0.901	8.6640	12.5307	5.7119	15.4828	-0.5974	1.888	-0.316	|	0.006
8	10.0000	8.6940	0.746	7.0938	10.2943	3.9306	13.4575	1.3060	1.954	0.668	|*	0.016
9	10.0000	9.3175	0.663	7.8965	10.7386	4.6113	14.0238	0.6825	1.984	0.344	|	0.003
10	10.0000	9.0790	0.662	7.6584	10.4997	4.3729	13.7852	0.9210	1.984	0.464	|	0.006
11	10.0000	8.4511	0.784	6.7692	10.1329	3.6596	13.2425	1.5489	1.939	0.799	|*	0.026
12	15.0000	13.5770	0.784	11.8957	15.2583	8.7857	18.3683	1.4230	1.939	0.734	|*	0.022
13	15.0000	10.6852	0.705	9.1740	12.1963	5.9509	15.4194	4.3148	1.970	2.191	|****	0.154
14	13.0000	14.8496	1.010	12.6829	17.0162	9.8672	19.8319	-1.8496	1.832	-1.010	*|	0.078
15	13.0000	14.9086	1.049	12.6579	17.1593	9.8891	19.9281	-1.9086	1.810	-1.055	*|	0.094
16	14.0000	12.2032	0.810	10.4659	13.9405	7.3920	17.0144	1.7968	1.929	0.932	|*	0.038
17	16.0000	14.2987	1.668	10.7208	17.8766	8.5601	20.0372	1.7013	1.262	1.348	|**	0.793
18	10.0000	11.2677	0.865	9.4117	13.1237	6.4124	16.1230	-1.2677	1.904	-0.666	*|	0.023

Sum of Residuals	0
Sum of Squared Residuals	61.2626
Predicted Resid SS (Press)	108.2328

Stepwise Procedure for Dependent Variable Q9

Step 1 Variable TCAPN2 Entered R-square = 0.37964848 C(p) =

	DF	Sum of Squares	Mean Square	F	Prob>F
Regression	1	59.14079716	59 14079716	9 79	0 0065
Error	16	96 63698061	6 03981129		
Total	17	155 77777778			

Variable	Parameter Estimate	Standard Error	Type II Sum of Squares	F	Prob>F
INTERCEP	8 04128646	1 07873453	335 61802446	55 57	0 0001
TCAPN2	0 00364791	0 00116577	59.14079716	9 79	0 0065

Bounds on condition number. 1, 1

Step 2 Variable R1T Entered R-square = 0.54426289 C(p) =

	DF	Sum of Squares	Mean Square	F	Prob>F
Regression	2	84.78406343	42 39203171	8 96	0 0028
Error	15	70.99371435	4.73291429		
Total	17	155.77777778			

Variable	Parameter Estimate	Standard Error	Type II Sum of Squares	F	Prob>F
INTERCEP	6.41259601	1.18383402	138.87202453	29 34	0.0001
TCAPN2	0.03006499	0 01139595	32 94199871	6 96	0 0186
R1T	-0.00089120	0 00038287	25.64326626	5 42	0 0343

Bounds on condition number 121 9467, 487 787

Step 3 Variable TMSGSND Entered R-square = 0.62526937 C(p) =

	DF	Sum of Squares	Mean Square	F	Prob>F
Regression	3	97 40307327	32 46769109	7.79	0 0027
Error	14	58.37470450	4.16962175		
Total	17	155.77777778			

Variable	Parameter Estimate	Standard Error	Type II Sum of Squares	F	Prob>F
INTERCEP	4 89730048	1 41186382	50.16774049	12 03	0.0038
TCAPN2	0 03680662	0.01137668	43.64326223	10.47	0 0060
TMSGSND	0 18184108	0.10452681	12 61900984	3 03	0 1038
R1T	-0 00117938	0 00039571	37.03860181	8 88	0 0099

Bounds on condition number 147 8584, 864.4986

Step 4 Variable TEXELNS Entered R-square = 0.63938286 C(p) = .

	DF	Sum of Squares	Mean Square	F	Prob>F
Regression	4	99 60164170	24.90041042	5 76	0 0068
Error	13	56 17613608	4 32124124		
Total	17	155.77777778			

Variable	Parameter Estimate	Standard Error	Type II Sum of Squares	F	Prob>F
INTERCEP	5 16563932	1 48572190	52 23740821	12 09	0 0041
TEXELNS	-0 00466248	0.00653659	2.19856842	0 51	0 4883
TCAPN2	0 03742320	0 01161389	44 86779216	10 38	0 0067
TMSGSND	0 18952761	0 10695455	13 56922320	3 14	0 0998
R1T	-0.00113333	0.00040798	33.34573196	7 72	0 0157

Bounds on condition number: 151.6568, 1209.511

Step 5 Variable TEXELNS Removed R-square = 0.62526937 C(p) =

	DF	Sum of Squares	Mean Square	F	Prob>F
Regression	3	97 40307327	32.46769109	7.79	0.0027
Error	14	58.37470450	4.16962175		
Total	17	155.77777778			

Variable	Parameter Estimate	Standard Error	Type II Sum of Squares	F	Prob>F
INTERCEP	4.89730048	1 41186382	50 16774049	12 03	0 0038
TCAPN2	0 03680662	0 01137668	43.64326223	10.47	0 0060
TMSGSND	0 18184108	0 10452681	12 61900984	3.03	0 1038
R1T	-0.00117938	0.00039571	37.03860181	8 88	0 0099

Bounds on condition number 147 8584, 864 4986

All variables left in the model are significant at the 0 1500 level.
No other variable met the 0.5000 significance level for entry into the model.

Summary of Stepwise Procedure for Dependent Variable Q9

Step	Variable Entered	Variable Removed	Number In	Partial R**2	Model R**2	C(p)	F	Prob>F
1	TCAPN2		1	0.3796	0.3796	.	9 7918	0.0065
2	R1T		2	0 1646	0.5443		5 4181	0.0343
3	TMSGSND		3	0.0810	0.6253		3 0264	0 1038
4	TEXELNS		4	0.0141	0 6394		0 5088	0.4883
5		TEXELNS	3	0.0141	0.6253		0 5088	0 4883

Obs	Dep Var Q9	Predict Value	Std Err Predict	Lower95% Mean	Upper95% Mean	Lower95% Predict	Upper95% Predict	Residual	Std Err Residual	Student Residual	-2-1-0 1 2	Cook's D
1	5.0000	7 2188	0 905	5 2771	9.1605	2.4281	12.0095	-2 2188	1 830	-1.212	*|	0 090
2	5 0000	7.5775	0.907	5.6325	9 5225	2.7855	12.3696	-2.5775	1 830	-1 409	*|	0 122
3	10.0000	10.2305	0 916	8.2651	12 1958	5.4301	15.0308	-0 2305	1 825	-0.126	|	0 001
4	10 0000	11 0368	0.692	9.5518	12.5217	6 4123	15.6612	-1 0368	1 921	-0.540	* |	0 009
5	10 0000	9 3768	1 774	5 5713	13 1823	3 5748	15.1788	0 6232	1 011	0 617	| *	0 293
6	10 0000	10.1089	0.619	8.7803	11 4374	5 5322	14 6855	-0 1089	1 946	-0 056	|	0 000
7	10 0000	10.4952	0 556	9 3028	11 6876	5 9562	15.0342	-0 4952	1 965	-0 252	|	0 001
8	10 0000	9 4897	0.786	7.8028	11 1765	4 7965	14 1829	0 5103	1 884	0.271	|	0 003
9	10.0000	9 9294	0 732	8 3597	11 4992	5.2770	14 5818	0 0706	1 906	0 037	|	0 000
10	10.0000	9 4819	0 855	7 6484	11 3153	4 7340	14.2297	0 5181	1 854	0 279	|	0.004
11	10 0000	8.7063	0.736	7 1280	10 2846	4.0511	13.3616	1 2937	1 905	0 679	| *	0 017
12	15 0000	12.8355	0.839	11.0358	14.6351	8 1006	17.5704	2 1645	1.862	1 163	| **	0 069
13	15.0000	9 7669	0 814	8.0208	11.5130	5 0521	14.4817	5.2331	1 873	2 794	| *****	0 369
14	13 0000	14 6600	1 109	12.2812	17 0388	9 6761	19.6439	-1 6600	1 715	-0 968	* |	0.098
15	13 0000	14 5062	1.095	12 1587	16.8536	9 5372	19 4752	-1.5062	1 724	-0 874	* |	0 077
16	14.0000	12 9162	0.796	11.2095	14 6229	8 2158	17.6166	1 0838	1 881	0 576	| *	0.015
17	16.0000	15 5720	1.576	12.1926	18 9513	10 0402	21 1037	0 4280	1 299	0 330	|	0.040
18	10 0000	12 0916	0 734	10.5164	13 6667	7 4374	16 7458	-2 0916	1 905	-1 098	*|	0 045

Sum of Residuals	0
Sum of Squared Residuals	58 3747
Predicted Resid SS (Press)	93.1659

VITA

Imtiaz Ahmad

Candidate for the Degree of

Master of Science

Thesis: SOFTWARE METRICS FOR PARALLEL PROGRAMS

Major Field: Computer Science

Biographical:

Personal Data: Born in Karachi, Pakistan, December 29, 1962, the youngest son of Mr. & Mrs. K. M. Khan.

Education: Graduated from Federal Government Model School, Satellite Town, Rawalpindi, Pakistan, in June 1976; received Bachelor of Science degree in Mathematics from University of the Punjab, Lahore, Pakistan, in November 1981; completed requirements for the Master of Science degree at the Computer Science Department of Oklahoma State University in July 1991.

Professional Experience: Graduate Research Assistant, University Computer Center, Oklahoma State University, January 1989 to May 1991.