High Frequency Scattering code in a
Distributed Processing Environment

THESIS

Scott Suhr
Captain, USAF

AFIT/GE/ENG/91J-04

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GE/ENG/91J-~~05~~ 04

High Frequency Scattering code in a Distributed Processing Environment

THESIS

Scott Suhr
Captain, USAF

AFIT/GE/ENG/91J-~~05~~ 04

Approved for public release; distribution unlimited

REPORT DOCUMENTATION PAGE

Form Approved
OMB No 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	June 1991	*Master's Thesis*

4. TITLE AND SUBTITLE
High Frequency Scattering code in a Distributed Processing Environment

5. FUNDING NUMBERS

6. AUTHOR(S)
Scott Suhr, Capt USAF

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Air Force Institute of Technology, WPAFB OH 45433-6583

8. PERFORMING ORGANIZATION REPORT NUMBER
AFIT/GE/ENG/91J-04

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Capt J. L. Fath
WL/AARA
WPAFB OH 45433

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for Public Release; Distribution Unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

Government agencies and academic institutions are very interested programming for concurrent processing to cut computer processing time. However, many of the world's problems have already been coded for conventional serial computers. This research demonstrates the feasability of modifying existing serial codes for execution in a concurrent processing environment. A electromagentic scattering prediction code known as NECBSC is incrementally modified to incorporated various levels of concurrent computing. The data processed by the code are completely independent, providing an avenue for data decomposition of the process. Portions of the data set are processed on each node and the results combined for final output. The final version of the code demonstrates a speedup of 3.59 on an eight node iPSC/2, verses the serial benchmark on that machine. Speedup for the iPSC/860 is 2.51, lower (vs its baseline) because of the faster processor, but it's elapsed time is shorter by 23%. Significantly better efficiencies are achievable when a more complex situation is simulated due to the relatively constant volume of output/communications. The success of this effort demonstrates that, at least for problems easily data-decomposed, the decomposition and implementation of existing serial codes for execution in a concurrent environment is both possible and profitable.

14. SUBJECT TERMS
Parallel Processing, Computer Programming, Electromagnetic Scattering. *OtherTerms*: Hypercube, Concurrent Programming

15. NUMBER OF PAGES
107

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

GENERAL INSTRUCTIONS FOR COMPLETING SF 298

The Report Documentation Page (RDP) is used in announcing and cataloging reports. It is important that this information be consistent with the rest of the report, particularly the cover and title page. Instructions for filling in each block of the form follow. It is important to ***stay within the lines*** to meet ***optical scanning requirements.***

Block 1. Agency Use Only *(Leave blank)*.

Block 2. Report Date. Full publication date including day, month, and year, if available (e.g. 1 Jan 88). Must cite at least the year.

Block 3. Type of Report and Dates Covered. State whether report is interim, final, etc. If applicable, enter inclusive report dates (e.g. 10 Jun 87 - 30 Jun 88).

Block 4. Title and Subtitle. A title is taken from the part of the report that provides the most meaningful and complete information. When a report is prepared in more than one volume, repeat the primary title, add volume number, and include subtitle for the specific volume. On classified documents enter the title classification in parentheses.

Block 5. Funding Numbers. To include contract and grant numbers; may include program element number(s), project number(s), task number(s), and work unit number(s). Use the following labels:

C	Contract	PR	Project
G	Grant	TA	Task
PE	Program Element	WU	Work Unit Accession No.

Block 6. Author(s). Name(s) of person(s) responsible for writing the report, performing the research, or credited with the content of the report. If editor or compiler, this should follow the name(s).

Block 7. Performing Organization Name(s) and Address(es). Self-explanatory.

Block 8. Performing Organization Report Number. Enter the unique alphanumeric report number(s) assigned by the organization performing the report.

Block 9. Sponsoring/Monitoring Agency Name(s) and Address(es). Self-explanatory.

Block 10. Sponsoring/Monitoring Agency Report Number. *(If known)*

Block 11. Supplementary Notes. Enter information not included elsewhere such as: Prepared in cooperation with...; Trans. of...; To be published in.... When a report is revised, include a statement whether the new report supersedes or supplements the older report.

Block 12a. Distribution/Availability Statement. Denotes public availability or limitations. Cite any availability to the public. Enter additional limitations or special markings in all capitals (e.g. NOFORN, REL, ITAR).

- **DOD** - See DoDD 5230.24, "Distribution Statements on Technical Documents."
- **DOE** - See authorities.
- **NASA** - See Handbook NHB 2200.2.
- **NTIS** - Leave blank.

Block 12b. Distribution Code.

- **DOD** - Leave blank.
- **DOE** - Enter DOE distribution categories from the Standard Distribution for Unclassified Scientific and Technical Reports.
- **NASA** - Leave blank.
- **NTIS** - Leave blank.

Block 13. Abstract. Include a brief ***(Maximum 200 words)*** factual summary of the most significant information contained in the report.

Block 14. Subject Terms. Keywords or phrases identifying major subjects in the report.

Block 15. Number of Pages. Enter the total number of pages.

Block 16. Price Code. Enter appropriate price code ***(NTIS only)***.

Blocks 17. - 19. Security Classifications. Self-explanatory. Enter U.S. Security Classification in accordance with U.S. Security Regulations (i.e., UNCLASSIFIED). If form contains classified information, stamp classification on the top and bottom of the page.

Block 20. Limitation of Abstract. This block must be completed to assign a limitation to the abstract. Enter either UL (unlimited) or SAR (same as report). An entry in this block is necessary if the abstract is to be limited. If blank, the abstract is assumed to be unlimited.

*U S GPO 1990 0 273-271

AFIT/GE/ENG/91J-~~05~~ 04

High Frequency Scattering Code in a Distributed Processing Environment

THESIS

Presented to the Faculty of the School of Engineering

of the Air Force Institute of Technology

Air University

In Partial Fulfillment of the

Requirements for the Degree of

Master of Science in Electrical Engineering

Scott Suhr, B.S.

Captain, USAF

June, 1991

Approved for public release; distribution unlimited

Acknowledgments

I would like to thank the many people who assisted me in surmounting the obstacles which littered my path in the course of this research. First of all, Rick Norris, the man who helped me find several of my most devious bugs and who rebooted the cube after many a meltdown. Also, my sponsor, Jeff Fath, who provided unfailing support to the extent that he was available. Then there is the author of NECBSC, Dr. Ron J. Marhefka, who imparted to me much of my understanding of the inner workings of the code, and provided me data on a two day turnaround. I would also like to thank my thesis committee members: Dr. Gary B. Lamont, whose guidance and direction of the research and this thesis was invaluable, Dr. Andrew J. Terzuoli, Jr. whose patience with me was outstanding, and Capt Phil Joseph. Last, but not least, I would like to acknowledge "the kids" who assisted me in waking up late many a night by taking me for a walk, and my wife Susan for her patience and support.

Scott Suhr

Table of Contents

Page

Page

Page

List of Figures

List of Tables

AFIT/GE/ENG/91J-05

Abstract

Government agencies and academic institutions are very interested programming for concurrent processing to cut computer processing time. However, many of the world's problems have already been coded for conventional serial computers. This research demonstrates the feasability of modifying existing serial codes for execution in a concurrent processing environment. A electromagentic scattering prediction code known as NECBSC is incrementally modified to incorporated various levels of concurrent computing. The data processed by the code are completely independent, providing an avenue for data decomposition of the process. Portions of the data set are processed on each node and the results combined for final output. The final version of the code demonstrates a speedup of 3.59 on an eight node iPSC/2, verses the serial benchmark on that machine. Speedup for the iPSC/860 is 2.51, lower (vs its baseline) because of the faster processor, but it's elapsed time is shorter by 23%. Significantly better efficiencies are achievable when a more complex situation is simulated due to the relatively constant volume of output/communications. The success of this effort demonstrates that, at least for problems easily data-decomposed, the decomposition and implementation of existing serial codes for execution in a concurrent environment is both possible and profitable.

High Frequency Scattering Code in a Distributed Processing Environment

I. Introduction

1.1 General

The reduction of the observables of current and future weapon systems constitutes a major thrust in Air Force research and development efforts; the pursuit of lower Radar Cross Section (RCS) receives considerable, if not the most, attention. Unfortunately, full-scale RCS range testing is generally very expensive, impossible in some cases . Alternately, many organizations use computer predictions of scattered electromagnetic fields to estimate RCS. All but the simplest of problems require considerable time, at a relatively high cost, even on potent mainframes

Locally, the Target Recognition Branch, Mission Avionics Division, Wright Laboratory (WL/AARA) uses the Numerical Electromagnetic Code - Basic Scattering Code (NECBSC, version 3), running it on mainframes and supercomputers. However, WL/AARA has an Intel iPSC/860 hypercube with eight i860 processors and the Air Force Institute of Technology (AFIT) has an Intel iPSC/2 with eight 80386 processors. Such concurrent architectures have the potential power to deliver supercomputer-like performance and accuracy at a much more affordable cost, if the software can be adapted to take advantage of the iPSC's concurrent processing capabilities. This research demonstrates the feasibility of modifying the existing FORTRAN prediction codes for solution in concurrent environments by porting NECBSC to the Wright-Patterson iPSC hypercubes.

Like all concurrent processing computers, a hypercube completes a task more quickly by dividing the problem into pieces for each node to process concurrently. The challenge is to divide the problem, compute partial answers, and combine the results more quickly than on a conventional sequential computer where all data is generally available "instantaneously" to any routine. Although a given combination of technique and architecture is usually optimum for only a particular class of problem, concurrent computers are designed to be flexible to allow hardware/software/data combinations which yield near maximum performance.

1.2 The Environment

1.2.1 The Computer: iPSC Hypercube The hypercube architecture consists of many individual "nodes", each consisting of a separate processor and memory. In the case of the AFIT iPSC/2 computer, each node also has a numeric coprocessor and a separate communications processor. As such, each node in this configuration can operate as an independent, self-contained computer, autonomously performing as many tasks as its memory will allow. In addition, a node can communicate with its neighbors through the communications processor and interconnection network, allowing it to coordinate its activities, share data, and, if necessary, load/execute new programming in order to continue work on the given problem.

1.2.2 The Software: NECBSC The code to be modified is NECBSC, a high-frequency scattering code initially written to evaluate antenna placement on a space station. As such, it allows multiple objects, antennas, and radiation sources to interact. It makes far- or near-field calculations of the energy reaching a specified observation point from specified angles, or can compute antenna coupling data. The code traces the path from each far-zone receiving direction backwards through each possible scattering path to each of the sources. It also has the capability to measure the total near-zone fields for a series of points along a specified path through space. Antenna coupling is accomplished similarly with the receiver antenna defined in terms of its free-field antenna pattern, the intensity of the fields arriving in a given direction modified by the appropriate gain. Shadowing is taken into account, and diffraction terms are calculated to smooth discontinuities in the reflected fields.

1.3 The Problem

Adapt the NECBSC computer code to run on the AFIT Intel iPSC/2 and validate the accuracy and efficiency of the combination. Evaluate the generic feasibility of porting existing FORTRAN code to run efficiently on concurrent computers.

1.4 Assumptions

NECBSC will run serially on a single node of an iPSC hypercube with no major modifications

1.5 Scope

This effort is limited to modifying NECBSC version 3/refnecbscman to run on the AFIT iPSC/2 and WL/AARA iPSC/860 hypercubes, and evaluating the conversion process. The modified code will retain as much of the basic structure of the sequential code as is practical, allowing direct comparison of accuracy and performance. The input section takes little of the total processing time for complex runs and its code is long so their is no need to modify this segment. However, the input code may be executed in parallel with the loading of the node programs. Once the modified program runs satisfactorily, test cases are executed to assess the speedup, efficiency, and accuracy of the new version. Test cases are executed with the original code on a VAX and on a single processor of the iPSC computers for comparative purposes.

1.6 Approach/Methodology

The generic approach applied is one of incremental conservativeness. NECBSC is first run on a VAX, then on other mainframes before executing on the iPSC. Once correct execution is confirmed, the program is run on a single node of the iPSC. The code is analyzed for structural and data segmentation which may allow simple decomposition into concurrent segments. A segmentation plan is formulated and implemented incrementally, to ensure accuracy at each stage. At each stage, the code is executed and times compared with benchmarks and previous versions to determine the worth of a given increment of modification.

1.7 Materials and Equipment

This research requires the use of the AFIT iPSC/2 and WL/AARA iPSC/860 computers to test the adapted routines, a SUN workstation on which to do the programming, and a VAX to run test cases. The use of FORGE^{TM} is desired, but not required.

1.8 Summary

This document records the relevant activities and processes used to complete the research introduced in this chapter. The next chapter relates some information on the general subject of concurrent computers, and coding for them. This is followed by a description of the requirements for this research. Next, the initial analyses and subsequent code modifications are described. Finally, the performance of the final version is described and conclusions rendered.

II. Background

2.1 Introduction

This chapter should impart to the reader a basic understanding of the issues involved in the adaptation of a given problem for efficient solution in a concurrent processing environment. The chapter begins with a discussion of computer hardware, then some concurrent programming fundamentals, and specific techniques and tools for code conversion.

2.2 Concurrent Computers

2.2.1 Philosophy Characteristics of the target computer have a large impact on the approach taken for decomposition of the source program. Concurrent processing computers fall into two fundamental classes of computing philosophy, each with its own advantages.

Single instruction, multiple data (SIMD) computers have multiple processors each with identical code, working on different data elements in "lock-step". These machines are especially appropriate for problems involving matrix/array solutions such as finite-difference methods where each processor calculates the same parameters for a single location in a grid or array (or a sub-array). It is common for SIMD computers to have processors numbering in the thousands. They are necessary to handle the types of problems which are the forté of SIMD.

Multiple instruction, multiple data (MIMD) computers have processors which may operate independently on different parts of the problem and/or different parts of the data. MIMD is most appropriate for solving problems where many unused options in a given run of the program and/or the data flows through a computationally intensive sequential pipeline of processes.

2.2.2 Memory The structure of a computer's memory also has a profound impact on the capability and suitability of a given architecture to a certain problem type.

When the computer has a single block of memory addressable by all of its processors, it is referred to as a "shared-memory" computer. Each processor can freely exchange data with the others through the intervening memory. This type of architecture lends flexibility to the programmer, but adds the penalty of hardware complexity/cost due to the communications network required and additional overhead due to memory contention and address decoding time.

When each processor has a dedicated block of memory to which it has direct exclusive access, the memory architecture is referred to as "distributed memory". This architecture is simpler, but may impose overhead when required data passes between the processors via messages. The distributed-memory architecture is appropriate for tasks that involve routines with large amounts of independent processing relative to the amount of data shared between processes.

Given the more common distributed memory MIMD computer, the physical interconnection topology of the processors has a large impact on the suitability of the machine to the task (with shared memory, one can "communicate" through memory locations.) Each processor can be connected directly to all others, but the complexity and serviceability of the hardware for even tens of processors is imposing. Various topologies have been used (Figure 1). One is a ring format that is suited to iterative control/data flow. A mesh interconnection scheme, with each processor communicating with each of its nearest neighbors, is easily translated to hardware connections. This works well for physically two-dimensional problems, but the scheme does not allow streamline communications for physical problems of three or more dimensions. The connection architecture for communicating directly with more than the physically closest nodes becomes unwieldy. There are other concurrent configurations, but the demonstrations of this research will deal exclusively with an architecture known as the hypercube.

2.2.3 The Hypercube The hypercube is a compromise architecture with a high degree of flexibility, maintaining the ability to communicate efficiently between nodes that are not nearest neighbors. A hypercube of dimension d has 2^d nodes each directly connected to d neighbors

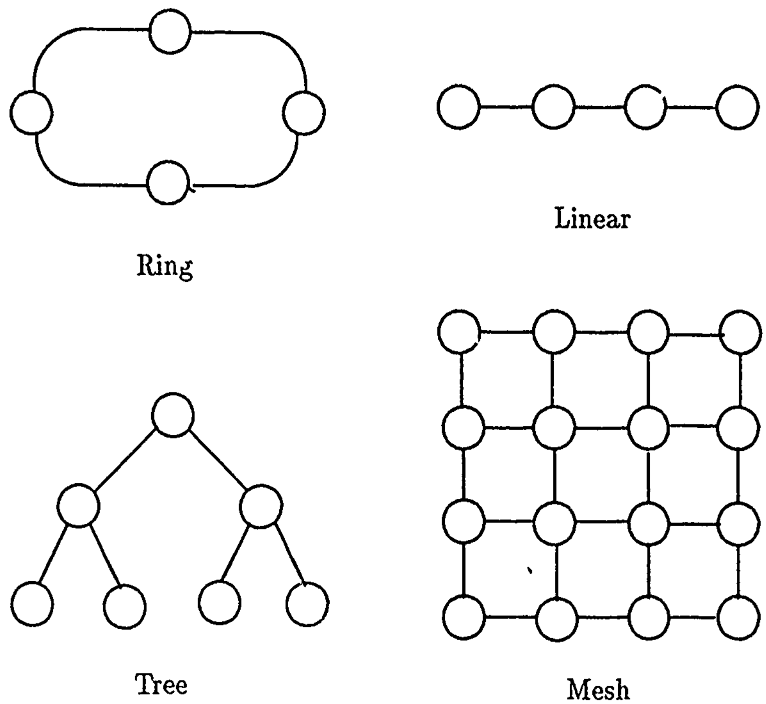


Figure 1. Selected concurrent computer topologies

(Figure 1c).

The name hypercube refers to the configuration of the interconnection network between the nodes of the computer. A d=3 (8 processor) configuration looks like a cube with a processor at each corner (Figure 2). At d=4 (16 processors), one has a cube within a cube. With higher dimensions it is more difficult to diagram, but "hyper-cube" is a good description.

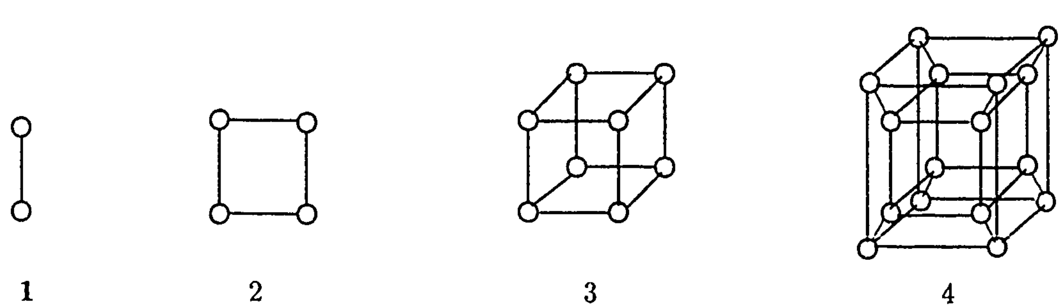


Figure 2. Hypercubes of Various Dimensions

This interconnection scheme becomes unwieldy when the dimension exceeds eight or so; the Intel iPSC/2 series is limited to 256 nodes. However, in cases with practical numbers of nodes, the hypercube exhibits a great degree of flexibility in adapting to problems of widely disparate "natural" topologies(8:49). For instance, the cube may be logically "unfolded" to adapt to problems with two or three dimensional meshes, or set up in a logical line or ring configuration for pipelined problems.

The iPSC/2 series hypercube nodes are also connected to a "host" ("front-end") computer, a 80386-based computer. The host has the responsibility of sending programming and data to each of the nodes, collecting any returned data, and serving as the conduit for all communications between the nodes and external entities such as the user, disk drives, and other computers. In addition compilation and linking of programs is usually accomplished on the host computer.

In the case of the iPSC/2 and /860 hypercubes, the node interconnect network is a 2.8 Mbytes/sec Direct-Connect^{tm} Network (3:10). Each node of the system is connected directly to each of its proximate neighbors through one of eight communications channels available in its Direct Connect Module (DCM). If the required communication is not to a direct neighbor, the message must be routed throug' preestablished link between the one or more intermediate nodes. The DCMs and their associated software also establish the availability of a receiving data buffer prior to creating a link. Only then do the DCMs actually forward any messages/data across the net (14:I-46). This linking process imposes an additional overhead burden on all communications, making 100 byte messages almost as "cheap" as single byte ones as shown in Table 1. In contrast, at the i860's 40 MHz clock speed, a single calculation takes considerably less than 1μsec, orders of magnitude less than even a short communication(16:17). This implies that the iPSC environment is inefficient when a problem is inherently "fine grained", that is, requires a large amount of data-passing between processes(4:1627). When the grain size is large, the communications overhead does not dominate, and reasonable efficiency can be obtained.

Since the iPSC hypercubes have only local memory, when a node needs data that is resident

Time per iPSC/860 Message	
Message Size (bytes)	Time (μsec)
1	81
10	85
100	122

Table 1. The Cost of Sending Messages

on a different node, it must request the data from the possessing node. The possessing node must be explicitly programmed to receive and interpret the request and to retrieve and send the data back to the requesting node. The requesting node must then explicitly receive the data (5:32). The greatest impact of this configuration is that there are no true global variables available to the programmer, as compared to the typical single-processor computer. This configuration requires more forethought on the part of the programmer, in terms of how the data is to be divided and shared among the processors. The situation is similar for the program code itself; the host must explicitly "load" a piece of code onto a specific node. Subsequently, if a node is to execute a different piece of code, the new code must be loaded through the standard interconnect network.

Though the physical environment is somewhat unconventional, Intel attempted to conform to existing standards in terms of the operating systems and compilers available for their machines. The iPSC/2 host processor uses the AT&T UNIX Version V operating system, and the processor runs NX/2 (a UNIX-compatible operating system). In addition, the iPSC/2 FORTRAN-386 compiler implements the full ANSI FORTRAN-77 standard (5:13). The iPSC/2's use of this standard software makes standard FORTRAN programs transportable to the target environment.

2.3 Fundamentals

One of the fundamental properties of concurrent processing is that you do not get something for nothing. Any sementing of work and/or data incurs a penalty in program loading and data transmission. There is always a tradeoff in the total design to achieve maximum efficiency.

2.3.1 Standards There are three major standards used to evaluate the desirability of concurrent processing implementations: speedup, efficiency, and accuracy.

Speedup is the ratio of the execution time in the sequential implementation to the fastest parallel implementation. It is a measure of merit of the success of the concurrent implementation vs the serial version and has a normal maximum of n (on n nodes). Speedup can be referenced to either the execution time on a reference computer or the parallel implementation running on a single node. The former is useful for showing the gains associated with moving to a parallel environment.

Efficiency is defined as speedup divided by the number of processors used. Qualitatively, efficiency is the proportion of time that the average node is spending to further the task at hand. Thus, efficiency has a theoretical maximum value of 1, corresponding to 100% processor utilization with no additional parallelization overhead (9:604). This peak (relative to a single node) is only theoretically achievable because there is always some setup and communications overhead associated with concurrent processing. Computationally idle time during overhead/message passing functions translates into degraded efficiency. Maximizing net performance while minimizing overhead should be a goal of every parallel program design process. Efficiency figures for different numbers of nodes can assist in determining the optimal number of processors to dedicate to a given process .

Accuracy is used in the normal sense. In general, does the implementation produce results that match the theoretical or actual solution? For the purposes of these investigations, accuracy refers to the number of digits of accuracy of the concurrent results, relative to measured results. Accuracy can vary due to the implmentation used on a given machine, as well as differences in round-off error due to the numeric precision of the hardware.

2.3.2 Concurrent Programming Using these measures of merit, the programmer attempts to maximize speedup and/or efficiency and maintain accuracy while designing and implementing a problem in software. Coding for a concurrent environment requires a totally different mind-set than sequential programming. Concurrent programming requires very high levels of abstraction in the early design to preclude the inadvertent application of serial constructs in a written program. One must identify only the prerequisite events for a given action in the initial design. Implementation requires knowledge of the specific target hardware: to determine if the prerequisites should be satisfied as sequential code on a given processor, or if the prerequisites are more efficiently completed on other processors with the resultant data or message passed to the dependent process. In many ways, programming in a concurrent environment is similar to object-oriented design/programming. The design/programming language called UNITY (6:8) is a good example of a language that allows clear distinction between processes and their interdependencies without overtly adding serial constructs to a design. All data and control dependencies must be identified and tracked throughout the software development effort if one is to work efficiently. Thus, to have the greatest flexibility in coding one has to design and write the code from scratch.

Unfortunately, designing and coding solutions to a given problem is very time-intensive and requires intimate knowledge of both the concurrent environment and the problem posed. Conversion of serial code is an alternate way of producing concurrent programs. Interest in this approach is great because "canned" sequential programs are available to solve many of the problems currently posed. Conversion of these codes is theoretically quick and cheap but there is always some added overhead, sometimes this penalty outweighs the gains made by multiple concurrent processes.

2.4 Program Conversion

2.4.1 Vectorizing Compilers From a user's point of view, a vectorizing compiler is the simplest way to adapt an existing program to run on a concurrent computer. The details of the conversion are hidden from the user by the compiler itself. The compiler attempts to identify sections of the code that are appropriate for concurrent computation. This is the method commonly used when porting a program for execution on the multi-processor supercomputers such as the

Cray XMP. Some experts believe that vectorizing compilers produce code that is more efficient than that of skilled programmers (13:237). However, a compiler can only identify and parallelize those constructs the compiler is specifically coded to find. In addition, a compiler does not execute a program with actual data; the specific options specified by the input stream can affect the optimal configuration and performance. The knowledge of an informed programmer is essential in designing the optimal parallel implementation of existing code.

2.4.2 Domain Decomposition One method of parallelization is to load identical programs on each node and divide up the data among the processors. This allows each processor to perform the task on part of the data. This method, called domain decomposition, is particularly effective when the problem has loops or operates on multiple sets data where each pass performs operations that are data-independent of all others. If the different passes of the loop are spread among different processors, there is no need for inter-processor communication during the calculations, the overhead is minimized, and efficiencies near one are possible. Of course, the code, input data, and output data have to be send to/from the nodes and a controlling program must coordinate their activities. All of these actions lower the demonstrated efficiency by some degree, but the technique generally shows good efficiencies when the number of iterations is high.

This approach also seems correct when the data-set is small, but the loading and data-transfer time may negate any parallelism gains, resulting in an inefficient resulting configuration. It is possible to formulate an expression to estimate the total execution time of a given segment of code on a given computer to aid in the decomposition process (10:719). Such a calculation requires a significant amount of information on the characteristics of a particular computer's architecture; the characteristics are often dependent on the software being evaluated. If so, the calculations would be based on many assumptions that may not apply to a general hardware/software combination.

It is also important to note that simply dividing the data at the outer-most loop or does not guarantee optimal efficiency. Imagine a problem with an outer loop with 15 iterations, encompassing an inner loop with 7 iterations(13:235). Mapping this to a computer with 27 processors, an obvious decomposition would be to divide the processors into three groups of seven, each processor calculating five components of the inner loop, with six inactive processors (which could be used for something else). However, there are other, more efficient solutions.

The number of iterations and the number of processors both play into the optimal solution. For example, one could "collapse" or "coalesce" the double loop to create a single loop of 105 iterations. If this loop were divided among all 27 processors, each would calculate only four loops and no processors would be idled. Collapsing is a process whereby multiply nested loops are converted into a single loop with a number of iterations equal to the product of the original nested loop sizes. This creates a one-dimensional vector where there might have been a doubly-subscripted variable in the original program. Coalescing is analogous, but maintains a one to one mapping between the variable subscripts in the nested- and single-loop cases . Thus the last iteration of an i=3, j=2, k=5 triply-nested loop would be element A(30) of a one-dimensional array in the case of collapsing or element A[f1(30),f2(30),f3(30)] of a three-dimensional array in the case of the coalesced loop.

2.4.3 Control Decomposition A second approach to parallelization is to divide the process into segments, putting different "subroutines" on each processor. This approach is called control decomposition. By passing multiple data in a stream through a sequence of nodes, concurrent processing occurs and a speedup is possible. However, because a given data set must pass between many or all of the node processes, there is a large communications penalty, unless the computations are time-consuming relative to the data-passing time required.

Once designed using control-decomposition, a problem can be mapped to the nodes in two fundamental ways. The first is a static configuration; the entire program structure is divided among the nodes and remains fixed throughout the lifetime of the program. The second uses "slave" nodes, loaded with programming and data as needed by one or more "master" nodes. This type of configuration is most appropriate when the entire program is too large to load and run on a single (every) node and the order of execution of the sub-processes (or whether they will be run at all) cannot be determined before run time.

2.5 Tools

There are many tools available to aid in the decomposition, analysis, and evaluation of parallel programming efforts. One tool appropriate for FORTRAN code conversion is FORGE^{TM}(7:5), a decomposition/analysis aid which creates a database of program and data entities and their usage, displaying them in a coherent format to the programmer/analyst . Other tools range from simple automated flowchart creators, to sophisticated debugging systems. Most of these tools are fairly intuitive and easy to use, with the notable exception of the debuggers.

2.6 Debugging

Debugging concurrent programs is more difficult to accomplish than debugging sequential programs. Since different processes may occur simultaneously and independently, it is possible to have a properly executing program give different answers on consecutive runs using the same input data. The cause may vary: the processors could be finishing in different orders each run or they could simply be finishing at the same time and competing for communications bus permission, transmitting in different orders each run. Little can be concluded directly about the specific causes of a case of erratic behavior.

In addition, symptoms of a problem can easily point in a spurious direction. Only with a concurrent debugger running in the background can enough relevant data be recorded to isolate any but the most obvious bugs. Unfortunately, the iPSC/860 does not allow multiple concurrent processes on a single node (2:1-1), limiting the capability of debuggers. The problem is further complicated by a manifestation of the Heisenburg uncertainty principle. Trying to accurately measure the behavior of the system influences the results obtained (12:594). Concurrent program bugs are often timing-dependent; the introduction of a concurrent debugging process can easily "cure" such a bug or activate others. Less intrusive passive event-recording systems are available, but they only store limited amounts of data and are therefore less useful.

2.7 Load Balancing

Load balancing should be an integral part of the design process. There are various ways to implement load balancing, from passive division of labor to complex token-passing master-slave systems.

2.7.1 Passive Passive load balancing can be achieved by decomposing the problem such that each processor gets an even share of the work. This is, of course, problem-dependent. Optimally the division must be such that when a node must stop to receive calculated data from another node the sending node has already completed the required calculations. Such a division is easier to accomplish and more likely to achieve high efficiency when the processes' calculations are independent of eachother's results, requiring no internode communication/synchronization. In addition, the total calculation time for each processor should be constant so no processors are idled at the end of the calculations.

2.7.2 Active Active load balancing refers to detecting activity on a each node and supplying more work, as needed. The detection and loading can be accomplished by the node itself, or by a supervisory node.

In one approach, each node reads in more data when it finishes with a given set of calculations. This scheme requires that the problem's input data stream be accessible in a central location, such as in a disk file, available to all of the affected nodes. Also, some pointer must be provided which indicates where the next unprocessed data is located. This type of load balancing has little overhead because no node or process is dedicated to supervisory duty.

In the master-slave approach, such a process exists and ensures that all other nodes are kept busy. When a processor is inactive, the master loads a new program or more data to keep it busy. As long as the master has work to perform, all of the slave processors will remain busy. The drawback is that a host or node is dedicated to supervisory duty. Assuming the supervisory overhead is low, a master-slave system can keep efficiency relatively high.

In one of the more complex schemes, a message "token", is constantly passed among the nodes and supervisory process. When a node is idle it adds its signature to the token. The token continues to be passed until a process has work to give to the idled node. The sender removes the recipient's signature from the token, assigns/sends the work and passes the token on. The disadvantage is that the overhead for passing the token(s) is always present, regardless of whether any work needs to be given away.

2.8 Summary

The field of concurrent program design is widely researched in the literature, but the issue of serial to concurrent conversion is not discussed in great depth. Concurrent computers have many ideosyncracies that must be taken into account when writing/converting code for them. However, there are tools available to help in the decomposition process. One can code from scratch, take the domain- or command-decomposition approach to sequential program conversion, or allow a commercial vectorizing compiler to control the parallelization task (if a compiler is available for the target computer). In the end, it is up to the individual programmer to analyze the problem, its data, and make the controlling decisions about how to effect the required parallelization. The efficiency of the resultant code is directly dependent on the qualifications of the programmer and the amount of time he is willing to invest in the development.

III. Requirements

3.1 The Sponsor's Needs

The Target Recognition Branch of Wright Laboratory has a need for a high frequency scattering code such as NECBSC which can be run on an on-going basis. They need relatively quick turn-around times on a flexible schedule. They have access to DOD supercomputers, but though the processing time is good, the turn-around time is slow, and funds need to be allocated to pay for the CPU time. They have, in house, an iPSC/860 hypercube capable of 60 peak MFLOPS per node (~6-10 avg.) (16:17) vs. 250 MFLOPS (~100 avg.) for an entire two processor Cray XMP supercomputer. At eight nodes, properly programmed, the iPSC/860 can exceed the capability of the Cray by a considerable margin. The sponsors need a version of NECBSC that will run efficiently on the iPSC/860 so they can control costs and turnaround times.

3.2 Academic Needs

From an academic point of view, the sponsor and AFIT would like to evaluate the feasibility of converting existing FORTRAN codes for use in distributed processing environments. The art of programming for concurrent processing is not close to perfection, and until such time, the utility of converting existing serial codes for execution on a multi-processor computer needs to be explored.

3.3 Specific Requirements and Approach

- Modify NECBSC to execute on the Intel iPSC/2 and iPSC/860 with concurrent processing.
- Accomplish the conversion in stages so as to always have a running version of the code if the most recent modification technique is not successful.
- At each stage of conversion, the converted code must take maximum advantage of the technique employed in that stage.
- Retain maximum compatibility with the original code to allow direct comparison of the timing

analysis and to ease use of the resulting code.

- Obtain timing results at each stage of the conversion process and compare in terms of the payoff verses the effort involved.
- Document any incompatibilities with NECBSC v3.0 and provide operating instructions for the modified code.

3.4 Goals

Produce an end product which has the following characteristics:

- Operable on iPSC/2 or iPSC/860 computers with 1 to 128 nodes without code modification.
- 100% compatible with the unmodified NECBSC code input streams, producing identical output formats.
- Output data with numerical precision at least as good as NECBSC run on a VAX.
- Speedup relative to sequential code on one node of the same machine $\geq$ 6.

IV. Initial Analysis

4.1 Introduction

Prior to modifying any source code, there are several issues to be resolved. Is the unmodified code compatible with the compilers available on the iPSCs? Are the details of the compilers and accuracies of the iPSCs equivalent to eachother and the VAXs so that the results can be compared? What is the speed of computation of the serial code on a VAX and the other platforms to be used in comparison? How should the structure of the code or data be analyzed and what basic approach should be used to decompose the code onto the multiple processors? How much memory is required for the code and its data? Will any restrictions be necessary due to limited memory on the hypercube nodes? These questions were answered in the Initial Analysis phase of this research, and described in the remainder of this chapter.

4.2 Compatibility

4.2.1 Portability The unmodified NECBSC code compiled and ran successfully on a VAX 11/780, the iPSC hosts, and a single node of each of the iPSC/2 and the /860 computers. Attempts to run the code on a Sun/3 workstation and Alexi computer were less successful. The sun version ran fine, but misinterpreted a boolean input in the input stream causing a phi-sweep to be performed when a theta-sweep was requested. The results were consistent with the sweep performed, but not with that requested. Due to the fundamental nature of the simple boolean comparison, use of the Sun platform is not recommended. The code failed to compile on the Alexi.

4.2.2 Accuracy Aside from the large differences in run-time, the iPSC/860 64 bit precision gave different results than the VAX 32 bit precision example output given in the documentation. Figure 3 of the i860 vs. all others shows the differences for Example 1c; see Appendix B for tabulated results. The traces show slight differences, but the answers are within 1 dB of eachother. The iPSC/2 (80386: 32 bit precision) results are identical to the example results. The differences occur

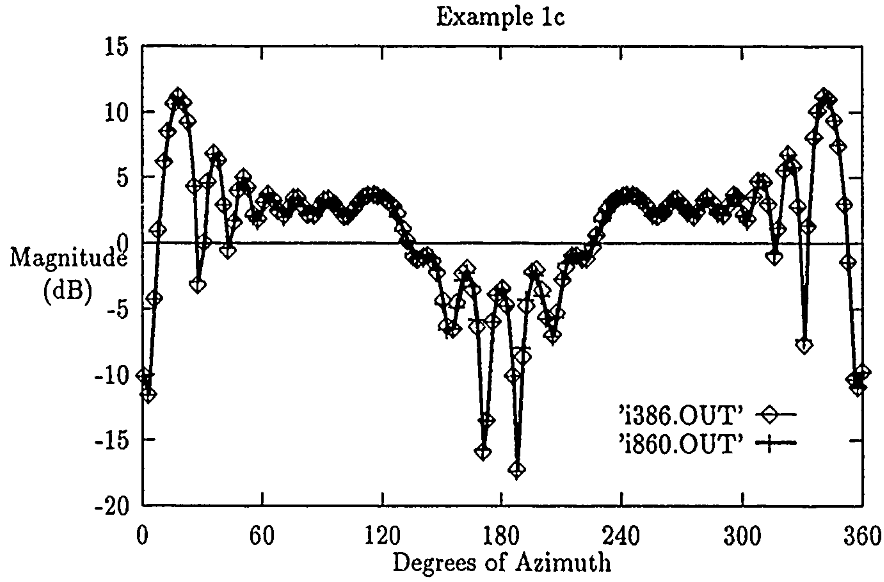


Figure 3. Numerical Precision, 32 vs. 64 bits

because the iPSC/860 hardware allows 64-bit precision internally, resulting in a larger dynamic range than the other platforms (all 32-bit) and therefore less roundoff/truncation error. Without a valid, numerically exact solution or very high precision test data, accuracy cannot be determined Since the VAX-generated reference data in the manual was originally verified against test data, one can only assume that the iPSC/860 results are more accurate.

4.3 Timing Baselines

To ease operation and timing of the serial code on the hypercubes, a short host program was written to load, execute, and time the serial code. Unfortunately, the Green Hills F77 compiler for the iPSC has no capability for recording elapsed time on the host (it does support node elapsed time). This is inadequate because measuring elapsed time from a totally independent node would not include the time required to load the programs from the host front-end to the nodes. This component can be significant for some of the possible implementations. Getting a true elapsed time would involve an embedded assembly-language or C routine, which may not be portable

Run Time (sec): Example 1c (no plot file)				
	VAX 11/780	microVAX	iPSC/2	iPSC/860
Time:	~143	~38	33.4	23.3
Speedup:	1.0	3.8	4.3	6.1

Table 2. Serial Run Times

between machines. Instead, timing data was collected to include the actual run time on the node, by sending a start and stop reference time from the node, but only the process time (CPU time) for the host to load the node routines was added. This method is consistent with a multi-user host. See Appendix D for an example of the format of this timing data. The host/node timing data is measured in the same manner in the later stages of the modification process.

Table 2 contains the timing baselines for this basic version of the code, from example 1c from the NECBSC manual(11:190). This example, described in Appendix A.1 was used as a benchmark throughout the modification of the code. It is the most complicated example included in the manual that includes both tabular and graphic data output formats for comparison. A tabular reference is necessary because modification of the plot-file generator routines was a secondary objective of this research and direct comparison of results is necessary in these analyses.

4.4 Memory Requirements

4.4.1 Memory Available The iPSC hosts have 8.5 Mb of memory, the iPSC/2 nodes have 12 Mb, and the iPSC/860 nodes have 16 Mb. This memory must hold not only the executable code, but the program's temporary and permanent data variables as well as any machine/operating system overhead memory burdens. Any additional data structures added in the code modification process must also be allowed for.

4.4.2 Code Size The source code for NECBSC is divided into six files which total 365 kilobytes (1 Kb = 1024 bytes) and compiles into a single executable file 590 Kb long on an iPSC host, 779 Kb bytes on the iPSC/2 node, and 889 Kb on the iPSC/860 node. Version 3 of NECBSC has the command processor in separate subroutines in a separate file to simplify creating overlays when memory is not adequate to keep the entire program in memory at all times(11:42). This is an advantage over Version 2, allowing the code to be ported to more limited hardware and making the code easier to read.

4.4.3 Data Size All data array sizing is determined by a few parameters in a single section of the NECBSC code. FORTRAN does not allow for dynamically allocated data structures, so the memory requirements are fixed for a given compilation of the code. Examination of the variable declarations for the arrays passed from the initialization routines reveals only 25 data structures of any consequence, amounting to 200 Kb of data space. These default array sizes are more than adequate, allowing 360° azimuth sweeps with a 0.2° step size (1801 steps). The other data structures are also adequate for typical problems.

4.4.4 Memory Needs Based on the above analysis, the memory available to each of the nodes on either of the iPSC computers is more than adequate to hold the programs and data necessary to execute NECBSC for reasonable problems. Therefore, memory conservation will not be a driving issue in the development of the concurrent version of the code and control decomposition is not required.

4.5 Code/Data Analysis

4.5.1 General With the portability was established and the precision and performance were determined, an approach to the analysis of NECBSC is formulated, analyses performed, and a code modification methodology created.

4.5.2 FORGE^{TM} Due to the sheer size of the NECBSC code (19,656 lines, 188 subroutines) maximum use of automated analysis tools is indicated. One tool developed by Pacific-Sierra Research Corporation and marketed by Intel Corporation for such analysis is FORGE^{TM}(7). By itself, this tool should have provided adequate information for guiding a partial decomposition process.

FORGE^{TM} parses standard FORTRAN-77 source code and develops a database of information on each individual subroutines, calling order, symbol usage, data interdependencies, etc. A timing profiler capability is also included. The package runs on a UNIX workstation under X-windows, under Sunview/Sunwindows, and also has a batch-mode capability for otherwise unsupported terminals when run on a UNIX host. The interactive development environment is quite extensive, allowing interactive source code formatting and modification as well as a host of informational output formats. FORGE^{TM} helps a programmer analyze existing code to determine the data and procedural relationships. This information, in turn, can assist in the decomposition process.

An attempt to analyze NECBSC with FORGE^{TM} was unsuccessful due to incompatibility with the NECBSC source code. The FORGE^{TM} manual claims it is compatible with FORTRAN-77, yet it will not allow the inclusion of data statements with complex arguments- which are used in NECBSC and allowed by all of the compilers used to date. The offending routines must be excluded or modified to conform to FORGE^{TM} F77 before a database can be created. Even so, FORGE^{TM} has another bug that causes an abnormal termination with core dump when processing some of the NECBSC files (6 files, largest. 218KB), presumably somehow related to the code length or memory requirements.

4.5.3 Manual Analysis Rather than modify the code to accommodate FORGE^{TM}, NECBSC was analyzed manually for control and data structures. A block diagram of NECBSC is provided in the documentation (1:24) and is reproduced here as Figure 4 in a slightly different format. This was used, along with the source code itself, to analyze the control structures. The NECBSC manual provides the input data format and syntax, as well as specific examples of input and the corresponding output for the code. Combined with the program structure, this information was deemed sufficient for the manual analysis.

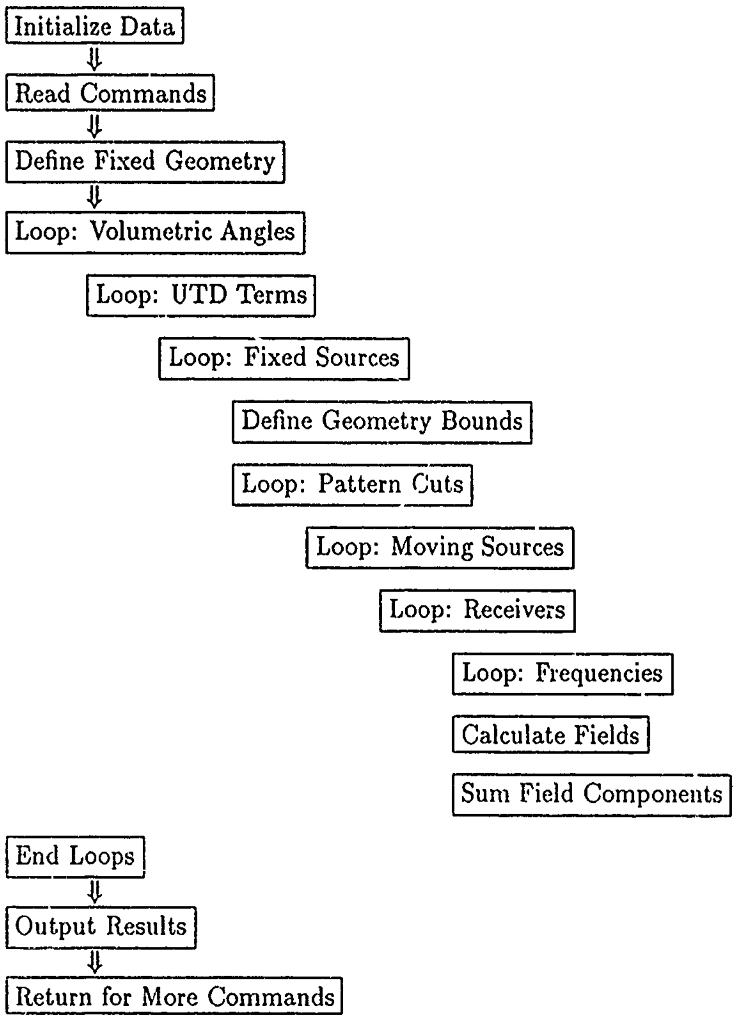


Figure 4. Block Diagram of NECBSC Version 3

4.5.4 Control Structure As shown in the figure, the NECBSC code is organized with a lengthy input parsing routine followed by a computation section composed of a series of nested do-loops, with the outer loops varying the pattern points. Inner loops calculate all the relevant scattering terms for all of the objects and sources.

Input is read from a disk file containing two-letter command codes, followed by the required data for that object, pattern, choice, etc. Multiple "runs" can be stacked in the input deck to allow unattended continuous computing (11:154). Output is normally tabular in format proceeded by header information that echoes the input deck in a more presentable format. This standard output can be sent to the screen, printer or disk. A file of output data in a standard binary form can also be requested for later plotting.

4.5.5 Data Dependencies Initial analysis showed that the code takes a brute force approach to the problem, determining each possible route from each source to the destination point (or direction if far-field). This can include up to three reflections and one diffraction (or one double diffraction) in the path(15). Data within each pass of at a given level of the looping hierarchy is independent of the data in other passes. Since the main program is constant to all calculation options, it seems suitable to accomplish the decomposition at this point in the code.

4.6 General Methodology

Since the code easily fits and runs within the limitations of a nodes' memory, and an independence exists in the data stream, data decomposition was chosen as the preferred method for breaking up the problem for concurrent processing. The general approach is to modify the code in increments. Any simple, yet potentially effective (low risk) modifications would be attempted first. Further modifications would subsequently be attempted, bolstered by the knowledge acquired during the earlier modifications. At each stage, timing would determine the effectiveness of the then current implementation, and guide the course of subsequent work. Load balancing issues will be considered if results indicate the need, though basic decomposition/communications modifications will be exploited to the point of diminishing return before attempting an active load balancing system.

4.6.1 Decomposition Options A direct approach is to modify the main FORTRAN routine to calculate segments of the required data, the specific segment for a given node determined by the total number of nodes and the number of the node in question. By indexing the loop on each node to a different start, step, and end point, each node will calculate an equal share of the data. If points in the sweep are interleaved, computationally intensive objects or directions would naturally be split among multiple nodes, forming a natural passive load-balancing system. The outer volumetric pattern angle loop is an obvious point to attempt this modification. By modifying the main routine, all underlying routines should have the correct data.

Another option initially considered was to modify each of the input routines as they read the input file and initialize the looping process index variables. By initializing the variables to a different start, step, and end, each node would calculate an equal share of the data. Because there were many such routines, this method was initially discarded in favor of the Main program modification described earlier.

4.6.2 Output Options After properly decomposing the problem, implementing a method for the retrieval of the output remained. Several options were considered:

- F77 writes from nodes w/indexed filenames
- Form ASCII array of output, send to host for output
- Individual messages to host for output
- Form array of data, send to host for output

The first option is simple, but involves post-processing the individual data files to form a composite output file. This would add an element of code which is entirely serial, adding directly to the run time. The second option would involves writing a subroutine and replacing each write statement with a subroutine call which would format the data properly before sending it to the host for final output. The third and last options are similar to the first two, but the data would be in a more compact format, four bytes per number, rather than one byte per character. Since the first option is simpler, it was chosen to be part of the initial implementation. Depending on the performance penalties produced by the initial modifications, other options would be exercised.

4.7 Summary

The initial analyses performed prior to code modification demonstrate that automation is not always necessary, or the best approach. Attempts to use the powerful analysis tool FORGE failed; manual methods found and formulated a viable approach to decomposition of NECBSC. This approach starts with the data decomposition of the problem, dividing the data streams in the outer loop of the main routine of NECBSC.

V. NECBSC Modifications

5.1 General

This chapter relates the actual modification of the NECBSC code. One of the first tasks was to code a robust host program to obtain the necessary inputs from the user and control/time the computation process. After that, the levels of code modification are described in terms of the structural changes and the performance results.

To ease in identifying the different versions of the code, a numbering scheme was established. Each version of the code has a modification number ranging from Mod0 to Mod4.0. Table 3 lists the different versions produced, and a brief description of their relevant features. For ease in handling, the source filenames are shortened to, for example: "32v.f" from the distribution name "NECBSC32V.F".

As discussed in the background chapter, background processes influence the elapsed and, in some cases, CPU times required to complete a process. The times given in this chapter are typically the result of a single run of the code and therefore may be off by a few percent. The numbers used for the baseline (Mod0.5) are averages from 5 runs. The elapsed times from the host, and VAX machines are measured manually and the minimum time obtainable over several runs is reported. Every attempt was made to take such measurements when the multi-user workload was minimum, lower times tended to indicate less background workload.

5.2 Host Program

The host program is greatly expanded for the concurrent modified versions of NECBSC. It now includes routines to get, parse, and transmit to the nodes the input filename , load executable programs on the nodes, open a file for host output, and get start/end times from the nodes. In addition, it writes a short header with version information, progress messages, and timing information to the screen. A complete listing of the Mod4.0 host routine is included as Appendix C.1.

Version	Files	Salient Features	Results
NECBSC	32.f, 32m.f, 32v.f, 31c.f, 31d.f, 31p.f	Unmodified code, no host	Runs on hosts, can't get input filename on nodes
Mod0	host.f, 32.f, 32m.f, 32v.f, 31c.f, 31d.f 31p.f	Original Code, less VAX-VMS-specific commands, ex1c.inp hard-wired	Gives only Host CPU time required
Mod0.5	host.f, 32.f, 32m.f, 32v.f, 31c.f,31d.f 31p.f	Host: get/load single node, get/send filename to one node, receive start/end times; Node: NECBSC + receive (vs. read from console) filename + send times	Long execution times, minimal timing data for baseline purposes.
Mod1.0	- same -	Host: same; Node: index outer loop in main program (32.f), replace VAX-VMS timing routine with iPSC calls (profile data)	All data calculated by 0th node, others only write banner
Mod1.1	- same -	Host: same; Node: index input data stream in command processor (32m.f)	Single iteration, no division of labor
Mod1.5	host.f, 32.f, 32m.f, 32v.f, 31c.f, 31d.f 31p.f	Host: merge individual output files, parse input filename on host – send corename, open & close node tempfiles and host outfile; Node: write to separate files whose names are indexed by node #	Works fine, but slower than unmodified code, quickest times on 2 nodes
Mod1.6	- same -	Host: adjust for header info; Node: write header info only from node 0	Times better (almost 1 speedup) but same "inverted" trend
Mod2.x	Host: host.f, 32h.f(32out.f), 32mh.f(32m.f); Node: 32.f, 32out.f(output routine), 32m.f, 32v.f, 31c.f, 31d.f 31p.f	Host: Parse input stream for banner, receive n 2D arrays, load into 3D array, write all output from host in duplicate output routine; Node: load 2D arrays with numerical data, output routine moved to separate file	numerical errors introduced, times good
Mod3.x	- same -	Same as 2.x with only the near-zone output routines modified	numerically OK, run times slightly faster than serial times, 2 and 4 nodes comparable – 2.4 speedup
Mod4.0	- same -	Similar to 3.x, but modify all output options, change to 1D array of data	times slightly better, near-zone & coupling inoperative

Table 3. Modified Versions of NECBSC

5.3 Alter Main Program (Mod1.0)

5.3.1 Volumetric Angle Loop The initial decomposition of the code consists of modifying the main segment of the NECBSC to index the outer DO Loop by the number of a given node. This modified code, labeled Mod1.0, runs flawlessly, but each node calculates the entire problem. The file from the first node contains the entire output, the other nodes produce only the "banner" information which precedes the actual tabulated data. Obviously the "volumetric angle" outer loop is not executed multiple times, and thus does not result in evenly divided output files.

5.3.2 Pattern Cut Loop Further investigation disclosed that the azimuth sweep index was incremented in a loop three levels deeper in the nest ("pattern cuts" in Fig 1). Dividing the work at this point in the code is successful in dividing the workload, but it seems the output files are indexed by other variables, not common-blocked from the main routine. The segmentation of the calculations is not passed to the output routine, and the nodes wrote NPN output points not $NPN/NUMNODES$ points as desired. Excerpts from the main node routine are included as Appendix C.2. Because there are many options and segments yet to adjust in the output routine, a simple, generic modification was not evident. Widespread modification also raises the risk of introducing a numerical error into the code. Therfore, the main program loop indexing approach was abandoned in favor of an alternate approach.

5.4 Index the Input Data Stream (Mod1.1)

The alternate approach involves modifying the several routines which initialize the indexed variables such that each node processes only the data points desired. With this approach, the input deck as "seen" by each node is modified to request only a portion of the problem. By modifying the pattern cut variables immediately after being read by the routine, the proper loop and the output routine are effectively modified and the structure of the code itself need not be changed, for the calculations or the output. The indexing is such that each successive angular increment is on a different node. This static load balancing attempts to divide up any computationally intensive regions between nodes.

The modifications also include modification of the VAX-VMS specific timing routines which report the amount of time spent in the different computational routines. These times are intended to supply guiding information to the user of NECBSC. In Mod1.1 the VAX calls are changed to call iPSC system routines. As a result of the concurrent processing, each output banner includes the information indicating the times elapsed on each node separately.

This version returns acceptable output, though the data remains spread over as many files as there are nodes, and each has a duplicate banner section with individual timing and data blocks. Elapsed time for eight nodes is roughly twice the serial time, without an attempt to collate the output. This modification does not meet basic requirements, it simply demonstrates multiple simultaneous file writing.

5.5 Merge on Host (mod1.5 & Mod1.6)

5.5.1 Mod1.5 This modification includes an expanded host program which merges the output files created by the nodes. The host now parses the input filename to obtain the root name for the output files and passes it along to the nodes when they are ready. This occurs while the nodes are initializing. The host then collects timing data and opens the node output files for input to the file merge routine. The collation process is simple because the format of the output is known. Duplicate header information has to be read, but is discarded before writing to the output file.

The performance of Mod1.5 is poor, partially due to the relatively inefficient F77 write emulation and partially due to an apparent overload of the node interconnect network. The F77 write requires a hefty setup period and is less efficient than an iPSC write . Therefore, the time for the multitude of individual writes accumulates rapidly. In addition, the structure of NECBSC is to perform all calculations for a given case, then call the output routine (see Figure 4 in Chapter

i860: Mod1.6, ex1c			
# of Nodes	Elapsed Time (ms)	Speedup	Efficiency
1	37303	1.24	124%
2	38478	1.20	60%
4	48496	0.95	24%
8	71964	0.64	8%

Table 4. Performance: Multiple-Banners, Merge on Host(Mod1.5)

4). Presuming the nodes start at essentially the same time and the passive load balancing causes them to finish all at once, thousands of individual F77 write "messages" are dumped on the system network together. The resulting saturation causes nodes to be idled, and elapsed times to increase as shown in Table 4. This saturation effect causes the concurrent times to greatly exceed the serial benchmarks. In addition, the banner information is replicated in all of the individual files, creating overhead.

An immediate improvement is to write the header information from only one node. Since the header is approximately half the total output, a reduction in communications traffic of nearly 50% is possible There is an additional synergistic effect if the communications traffic is reduced to below the saturation level.

5.5.2 Mod1.6 This incremental modification eliminates the passing of duplicate header information. Solely node 0 now writes the header block. The times indicated in Table 5 include elapsed time on one node and the host CPU time required to load the code on the nodes, merge the data, and output the results. The performance is poor, and has a disappointing trend. Ideally the times should drop as the number of nodes working on the problem increases. With the demonstrated performance it would be reasonable to use only 2 nodes (near maximum speedup with better processor utilization). This is obviously not a very worthwhile approach since the same problem can be solved on the host or one processor in less than 25 seconds.

Another limitation of this approach is it works with no more than 64 nodes, the F77 file

i860: Mod1.6, ex1c			
# of Nodes	Elapsed Time (ms)	Speedup	Efficiency
1	59685	.39	39%
2	40808	.57	29%
4	39463	.59	15%
8	46982	.50	6%

Table 5. Performance: Single Banner, Merge on Host (Mod1.6)

system allows only 99 units (files) open at a time. Work-arounds are possible (multi-stage merges) but would add further performance penalties. Performance could be improved by using the iPSC concurrent file system (CFS), but calls to the CFS do not correlate with formatted writes available in standard F77. The burden of conversion would overly complicate the code modification process and the penalty in execution time would likely cancel any gains made by using CFS. Other options for output are available and the programming to performance ratio is more favorable.

5.6 Data via 2D Array - Write from Host (Mod2.0 & Mod3.5)

5.6.1 Mod2.0 This modification requires duplication of the output routine on the host and nodes. At the point when the data would be written, the nodes load the numerical output data into an array of real numbers, in the order in which the data is normally printed out. After completing one segment of the output array (in near- and far-zone cases there are two segments), it is sent in a single iPSC message to the host (see Appendix C.4 for code excerpts). Because the setup overhead is limited to a few messages rather than thousands, the transfer efficiency are much higher than in the F77 writes approach.

The host parses the input deck and prints the banner information while the nodes are computing and waits for the nodes to complete. It receives the 2D arrays and loads them into a 3D array for ease of processing. Rather than calculate the output data, the array data is extracted and loaded back into the output variables and a version of the standard output routine prints the column headers (Appendix C.3).

i860: Mod3.5, ex1c			
# of Nodes	Elapsed Time (ms)	Speedup	Efficiency
1	21611	1.08	108%
2	17116	1.36	68%
4	15329	1.52	38%
8	15541	1.50	19%

Table 6. Performance: 2D Array messages, All Output from Host

While attempting Mod2.0, a numerical error was introduced. At the time, the error could not be located, so that version was set aside and ostensibly the same modifications were re-implemented in Mod3.0. Corrections and improvements advanced the version to Mod3.5 before fixing the configuration.

5.6.2 Mod3.5 This time the modifications implemented successfully despite the numerous problems that surfaced. In Mod3.5 the array scheme is applied to the near zone case only, to demonstrate the technique. Speedups significantly above one are obtained, as shown in Table 6. However, the trend is still skewed between 4 and 8 nodes: it is quicker to compute using 4 nodes than with 8.

Each segment of that output has 7 columns, necessitating a 7 by 361 (typically) real array for output data. Since F77 does not allow dynamic memory allocation, the array to be sent must be a fixed size. Because the array is declared larger than needed, only the length (number of output lines) necessary is sent. However, the near zone segments, not parallelized in Mod3.5, have between 5 and 12 columns. If the data array were sized for the maximum number of columns, sending a 5-column segment would require sending 140% more data than necessary.

5.7 Data via Linear Array (Mod4.0)

This version of the code implements a linear array representation of the output data. The "vectorization" is applied to all sections of the output routine because the technique is equally

i860: Mod4.0, ex1c			
# of Nodes	Elapsed Time (ms)	Speedup	Efficiency
1	20475	1.14	114%
2	16416	1.42	71%
4	14645	1.59	40%
8	14723	1.59	20%

Table 7. Performance: 1D Vector Messages, All Output from Host

efficient with any number of output columns since the vector only has one "column". A section of the declared data vector is filled and sent to the host where it is loaded into the same 3D array as used in Mod3.5.

Mod4.0 is the most efficient version created to date, for far zone output. Modifications to the near-zone and antenna coupling options contain bugs which prevent the use of these output options. The results for the benchmark example 1c, which uses the far-zone output option, are shown in Table 7. For this example, the 4 node case is still the fastest option.

5.8 Timing Summary

Each subsequent version of the code showed improvement over previous modifications as shown in Table 8. The most drastic improvement in speedup is when the output data is passed via message (actually starting with version 2.0). Beyond that change, subsequent improvements are slight.

5.9 General Observations

5.9.1 Choice of Loop Decomposition Division in the looping structure at the pattern cuts loop directly (through the input routines) implies that any calculations performed outside the pattern cut loop are repeated by all of the nodes. Unfortunately, if the structure of the problem is such that the volumetric angle variable has many iterations and the iterations of the pattern cut loop variable are few (and not a multiple of the number of nodes), a significant penalty is incurred.

Example 19 on iPSC/860 Speedup (SU) & Efficiency (EFF)				
			SU	Eff
Mod#	Nodes	Time (ms)	(vs. Mod0.5)	
4.0	1	43016	1.07	1.07
4.0	2	28064	1.64	.82
4.0	4	21036	2.19	.55
4.0	8	18359	2.51	.31
3.5	1	44348	1.04	1.04
3.5	2	28711	1.60	.80
3.5	4	22221	2.08	.52
3.5	8	18947	2.43	.30
1.6	1	39076	1.18	1.18
1.6	2	33000	1.40	.70
1.6	4	36217	1.27	.32
1.6	8	45104	1.02	.13
0.5	1	46132	1.00	1.00

Table 8. Speedup Summary (by Code Version)

In general, this is not true of NECBSC problems; users usually specify pattern cuts of some sort, resulting in a reasonable division of effort despite the use of differing reference locations.

There is a significant amount of code positioned prior to the pattern cut looping structure: initialization, input parsing, and geometry precalculations. The information is calculated on all the nodes and is needed by all nodes. Based on the penalty for using communications, it seems appropriate to leave the duplication, rather than compute the data once and send it to all the nodes. Normally (especially in the far zone) a complete azimuth sweep (360 degrees) is specified. Even at a step size of 5 degrees, 64 calculations are necessary, coincidentally a multiple of the number of nodes availab! If the user is judicious and chooses a multiple of 8, the code will always retain its efficiency. If the number of iterations drops to a small, non-integer multiple of the number of nodes, efficiency will drop due to the idled nodes.

5.9.2 GetCols Program Rather than implement a data transfer routine for the creation of a binary plot-data file, a simple data extractor was coded. This program (getcols.f, listed in Appendix C 5) processes an output file created by NECBSC (any version) and selectively extracts desired columns from each data section of the output. Getcols creates a standard space-delimited two-column output file that is compatible with most plotting programs. It also limits the number of points in the resulting file to 150, to fit the limitations of the gnuplot graphing program used in these analyses.

5.9.3 Understanding the Problem Almost without exception, any problem or bug encountered was a result of a misunderstanding of the true nature of the code or programming environments. It is essential to fully understand the nature and structure of the problem (and/or existing code) as well as the compiler, operating system, and hardware. Anything less than full understanding will be amplified by the concurrent environment and typically result in problems during coding.

5.9.4 Concurrent Debugging The complications introduced by the concurrent environment are well demonstrated by examining the debugging process. Simple programming errors can manifest themselves quite differently in a concurrent processing environment. For instance, an actual error (diagramed in Figure 5): a bug in the code of a node process blocks a message expected by the host process. The host process waits to receive a message that never comes and "hangs", causing the user to abort the run. The difficulty in diagnosing the problem is that the external manifestation of these events point in a different direction. A series of write statements that create an output file earlier in the process one code appears to have failed because the output file is incomplete.

In fact, the writes completed successfully prior to the buggy send-receive statements, writing the complete data to several system buffers; the last buffer is not completely filled by the writes, so the operating system waits until it does fill or is flushed by the normal termination of the program, the last buffer's contents are destroyed when the program is manually killed. Thus the indications are that the problem exists in the writes in the output routine on the host, when the problem is really in a second process on the node in a section that is executed after the symptomatic output

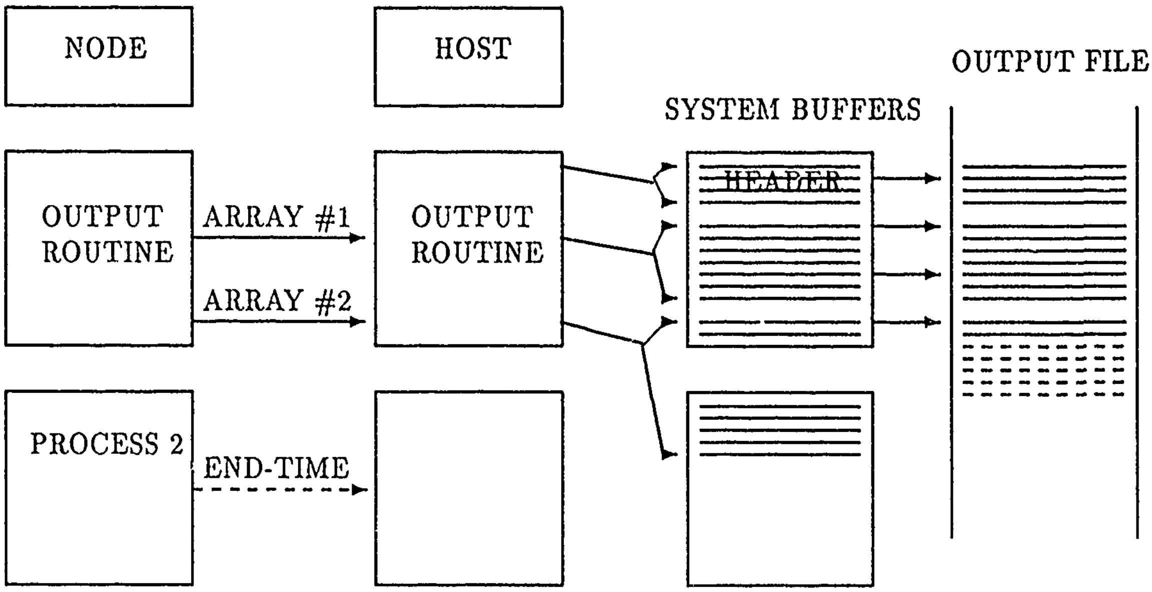


Figure 5. Example of Debugging in a Concurrent Environment

section has completed. Adding a debugging breakpoint to the program could confuse a programmer because the writes would terminate normally and one might search indefinitely for a timing problem that doesn't exist. In the case of this real problem, only a deep understanding (by others) of the hardware · d operating system unraveled the problem.

5.10 Summary

Modifications to the NECBSC source code were generally successful, though some were eminently more worthwhile than others. In a generic data decomposition problem, the input and output of the problem should be examined for data structures that are conducive to consolidation. The use of a post-process for the actual output of results is viable, though use of the first idle node for this purpose should be investigated, if the data structure permits output before all nodes complete. F77 writes should be avoided on node processes, and passed data should be consolidated to minimize the communications setup overhead. In any case, the modifications performed on NECBSC demonstrate the viability of modifying existing data-decomposable code.

VI. Mod4.0 Performance

6.1 General

6.1.1 Documentation No additional documentation for the iPSC versions of NECBSC was written in the course of this research. The Mod4.0 version is a direct replacement for the original code, with a few caveats: some commands are unavailable (see Section 6.1.2); the subroutine timing blocks provided for in the original code are disabled (they are included in Mod1.6); UNIX System V is case sensitive so the input filenames must be given in proper case; the input file *must* have a ".inp" extension (NECBSC requires ".INP"), ".inp" is added, if not entered with the filename.

6.1.2 NECBSC Command Limitations Mod4.0 has several limitations imposed by the modifications, over and above original limitations. The following options are inoperable at this time (see (11:45-156) for detailed explanation of commands):

- Near-Zone ("PN", "BN",)
- Antenna Coupling (uses"PN")
- Plottable Output ("PP")

6.2 Concurrent Performance

6.2.1 Speedup vs Mod0.5 The performance of Mod4.0 is outstanding. A speedup of 4.68 is achievable on the iPSC/2 (2.51 on the /860) as shown in Table 9. These data are from example 19, a far-zone, eight-sided cylinder section (11:298).

Since the AFIT iPSC/2 has both the Weitek and 80387 math coprocessors available, the code was compiled under each environment and timing data taken. These data show that the Wietex coprocessor holds a significant computational advantage over the 80387, though the communications time need be subtracted if one is to get a real figure of merit. The trend across the three platforms

Example 19 Speedup (SU) & Efficiency (Eff) by Machine										
		iPSC/860			iPSC/2 (Weitex)			iPSC/2 (80387)		
Mod#	Nodes	Time (ms)	SU	Eff	Time (ms)	SU	Eff	Time (ms)	SU	Eff
			(vs. Mod0.5)			(vs. Mod0.5)			(vs. 1 node)	
4.0	1	43016	1.07	107%	80562	1.07	107%	132032	1.00	100%
4.0	2	28064	1.64	82%	47308	1.82	91%	72761	1.81	91%
4.0	4	21036	2.19	55%	31496	2.72	68%	43897	3.01	75%
4.0	8	18359	2.51	31%	23923	3.59	45%	30231	4.37	55%
0.5	1	46132	1.00	100%	85931	1.00	100%	-	-	-

Table 9. Performance vs Computer

is that, although raw times are smaller on the faster machine, the speedup and efficiencies are better on the slowest machine (relative to a serial baseline on the same hardware.) The communications overhead for the three machines is fairly constant, causing the overhead-to-computation ratio to increase when computation speed is slower.

6.2.2 Speedup vs Mainframes When one references Mod4.0 speedups and efficiencies to those of the VAX and microVAX, the speedups and efficiencies given herein are simply multiplied by a constant factor equivalent to the corresponding speedup figures given in Table 2. For example, ex19 on the iPSC/860 produces a speedup of $2.51 \times 6.1 = 15.9$ referenced to the VAX 11/780 or 4.03 vs. the μVAX.

6.2.3 Speedup vs Problem Size The size of NECBSC output is relatively constant, independent of the complexity of the problem. Since the communication time is related to only the output requirements, larger problems have less overhead per calculation. As such, larger problems show better speedup and efficiency, as shown in Table 10.

6.3 Accuracy

All the modifications of NECBSC return results to the same degree of numeric precision as Mod0.5 run on the same hardware. Changing the math coprocessor used on the iPSC/2 did not

iPSC/860 Speedup (SU) & Efficiency (Eff) (Mod4.0 vs Mod0.5)									
	Problem 1c			Problem 6			Problem 19		
# of Nodes	Time (ms)	SU	Eff	Time (ms)	SU	Eff	Time (ms)	SU	Eff
1	20475	1.14	.14	40344	.83	.10	43016	1.07	1.07
2	16416	1.42	.18	26858	1.24	.16	28064	1.64	.82
4	14645	1.59	.20	20544	1.63	.20	21036	2.19	.55
8	14723	1.59	.20	18440	1.81	.23	18359	2.51	.31
Mod0.5:	23347			33438			46132		

Table 10. Performance vs Problem Size

effect precision. Without actual data or a numerically exact solution, accuracy as defined herein cannot be determined.

6.3.1 General Given a constant communications requirements, the combination of the machine with the slowest processor/numeric coprocessor and the most complex problem will have less overhead per calculation producing greater speedup and efficiency. The maximum speedup actually demonstrated by Mod4.0 is 4.37 (efficiency: 55%), relative to a baseline on the same hardware. This confirms that reasonable performance is achievable through existing program modification without resorting to the long and expensive method of re-coding from scratch.

VII. Conclusions

7.1 Existing Code Modification

7.1.1 NECBSC As demonstrated with Mod4.0, modifying existing serial code for execution in a distributed processing environment is both possible and profitable, though the results are machine-, code-, and problem-dependent. Results show that, given common communications hardware, the faster the processing capability of the machine, the less profitable it is to move from a single processor to concurrent processing with like processors. Likewise, the larger the ratio of calculations to output data size in a problem, the more efficient the code becomes. Since the overhead is I/O limited, and the output data size is normally fixed, the efficiency of an infinitely complex problem would approach 1. The data show actual speedups in the 4.6 range (28 referenced the VAX 11/780) for problems much simpler than those one would anticipate in real life. Higher speedups can be expected from more realistic problems.

7.1.2 Feasibility Based on the work accomplished, the generic task of porting existing serial codes which are data-decomposable is both achievable and profitable. The techniques used with the decomposition of NECBSC should be applicable to other problems of this type, and to aless er degree, to other types.

7.1.3 General It is essential the designer/programmer has a clear understanding of his operating environment (hardware/operating system) as well as the particular problem at hand. Almost without exception, any problem or bug encountered in this research was a result of a misunderstanding of the true nature of the code or environment.

7.2 Recommendations

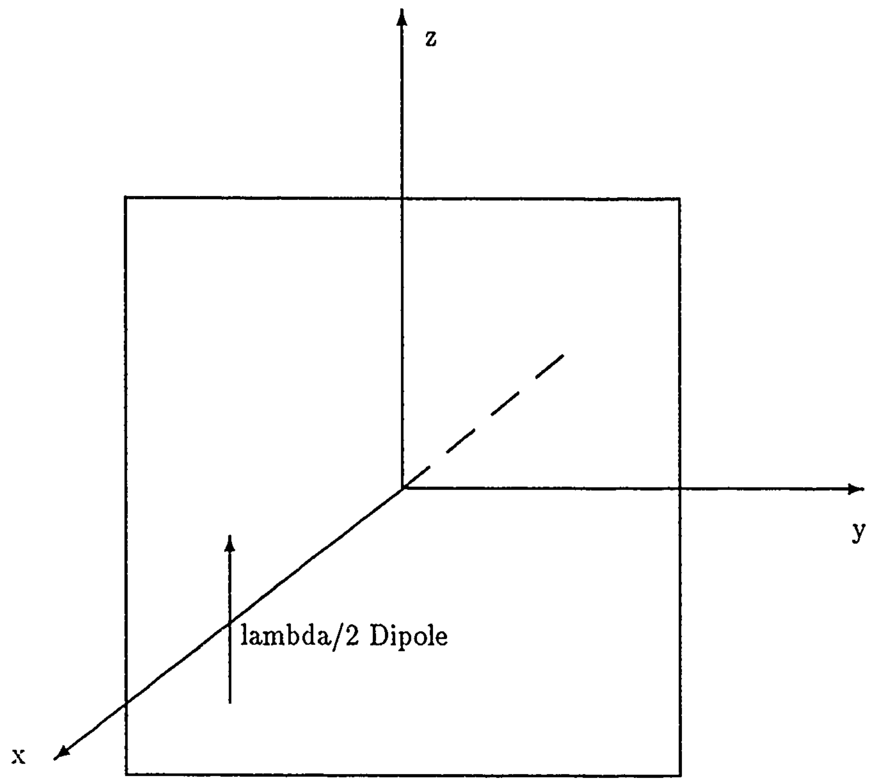
7.2.1 Complete Mod4.0 Though Mod4.0 is the most efficient version of NECBSC produced in this research, the implementation is not complete, and there are some significant restrictions on which options can be used. Most of these limitations can be removed without a deep technical knowledge of concurrent programming. Re-enabling the plot-file option may take a little extra effort, but the technique used for the tabular output can be applied directly, or the getcols program can be used or incorporated into the NECBSC code. The NECBSC subroutine timing blocks could be re-enabled in a similar fashion.

7.2.2 Change Control Structure Great gains could be made by streamlining the code through control decomposition of the problem, without using the structure of the existing program. The numeric subroutines could be retained to avoid the need to reprogram the electromagnetic details, but an intimate knowledge of the functioning of NECBSC would be required.

Appendix A. *Example 1c*

The following is a description of Example 1c (11:190), as modified for use as a benchmark in this research.

A.1 Example 1c Physical Problem



A.2 Example 1c Input Data

```
CE:   FAR ZONE PLATE TEST, EXAMPLE 1C.
UN:   UNITS IN INCHES
3
US:   SOURCE UNITS IN WAVELENGTHS
0
FR:   FREQUENCY IN GHZ
8.0
PF:   PATTERN CUT
45.,90.,90.,0.
T,90.
0.,1.,361
PG:   PLATE GEOMETRY
4,0
0.,3.5,3.5
0.,-3.5,3.5
0.,-3.5,-3.5
0.,3.5,-3.5
SG:   SOURCE GEOMETRY
5.12,0.,0.
0.,0.,90.,0.
-2,0.5,0.
1.,0.
LP:   LINE PRINTER OUTPUT
T
XQ:   EXECUTE CODE
EN:   END CODE
```

This input deck differs from that in the manual because the "PP" option is disabled in the modified code.

A.3 Example 1c Screen Output

```
mbvsrm>runbsc
   * NEC-BSC for iPSC/2 & iPSC/860 * 3.2i4.0,  6 May 91  *
   * Scott Suhr & Gary B. Lamont
     AFIT School of Engineering
     - Banner written to disk from host
     - Numeric data sent to host in real vector &
       written to disk from host
 Input desired Cube-Type (8rx, 4sx, etc):
8rx
 Number of nodes attached:          8
 Enter a filename for input (70 characters max)
ex1c.inp
 Input Filename = "ex1c.inp"
 Host:  CREAD Complete
 Receive Far Zone E-Field arrays from nodes
 Arrays received
 Receive Far Zone Total Field arrays from nodes
 Elapsed:  Node     Total Time (msec)
           0          6452
           1          6525
           2          6512
           3          6535
           4          6510
           5          6524
           6          6561
           7          6549
Output elapsed time (node 0, msec):   3793

 Host CPU time required for startup:   1930
 Host CPU time required for output:    6590
       Total Host CPU time required:   8530
Approx. total elapsed time required:  14972
      (node 0 + Startup + Output)
```

A.4 Example 1c Output Data

```
*******************************************************************************
*                                                                             *
*  NEC-BSC      3.2i4.0,  6 May 91                                            *
*                                                                             *
*     THE OHIO STATE UNIVERSITY                                               *
*     ELECTROSCIENCE LABORATORY                                               *
*     1320 KINNEAR RD.                                                        *
*     COLUMBUS, OHIO 43212                                                    *
*                                                                             *
*     WRITTEN BY RONALD J. MARHEFKA                                           *
*                                                                             *
*     MODIFED FOR iPSC/2 & iPSC/860 BY                                        *
*     SCOTT SUHR AND                                                          *
*     DR GARY B. LAMONT                                                       *
*     A.F. INSTITUTE OF TECHNOLOGY                                            *
*     AFIT/ENG                                                                *
*     WRIGHT PATTERSON AFB OH 45433-6583                                      *
*                                                                             *
*******************************************************************************


*******************************************************************************
*                                                                             *
*  CE:   FAR ZONE PLATE TEST, EXAMPLE 1C.                                     *
*                                                                             *
*                                                                             *
*******************************************************************************


*******************************************************************************
*                                                                             *
*  UN:   UNITS IN INCHES                                                      *
*                                                                             *
*                                                                             *
*     ALL THE LINEAR DIMENSIONS BELOW ARE ASSUMED TO BE IN INCHES             *
*                                                                             *
*******************************************************************************


*******************************************************************************
*                                                                             *
*  US:   SOURCE UNITS IN WAVELENGTHS                                          *
*                                                                             *
*                                                                             *
*     SOURCE LENGTH HS AND WIDTH HAWS ARE ASSUMED TO BE IN WAVELENGTHS        *
*                                                                             *
*******************************************************************************


*******************************************************************************
*                                                                             *
*  FR:   FREQUENCY IN GHZ                                                     *
*                                                                             *
*                                                                             *
*     FREQUENCY=   8.000 GIGAHERTZ                                            *
*                                                                             *
*     WAVELENGTH=   0.037474 METERS                                           *
*                                                                             *
*******************************************************************************


*******************************************************************************
*                                                                             *
*  PF:   PATTERN CUT                                                          *
*                                                                             *
```

```
*                                                                             *
*    PATTERN AXES ARE AS FOLLOWS:                                             *
*                                                                             *
*  VPC(1,1)=  1.00000  VPC(1,2)=  0.00000  VPC(1,3)=  0.00000                 *
*                                                                             *
*  VPC(2,1)=  0.00000  VPC(2,2)=  0.70711  VPC(2,3)= -0.70711                 *
*                                                                             *
*  VPC(3,1)=  0.00000  VPC(3,2)=  0.70711  VPC(3,3)=  0.70711                 *
*                                                                             *
*    PHI IS BEING VARIED WITH THETA=   90.00000                               *
*                                                                             *
*    START=    0.00000 STEP=   10.00000 NUMBER=   36                          *
*                                                                             *
*******************************************************************************

*******************************************************************************
*                                                                             *
* PG.   PLATE GEOMETRY                                                        *
*                                                                             *
*                                                                             *
*    THIS IS PLATE NO.    1 IN THIS SIMULATION.                               *
*                                                                             *
*                                                                             *
*    METAL PLATE USED IN THIS SIMULATION                                      *
*                                                                             *
* PLATE#  CORNER#   INPUT LOCATION IN INCHES    ACTUAL LOCATION IN METERS     *
* ------  -------  -------------------------- --------------------------      *
*                                                                             *
*    1       1       0.000,   3.500,   3.500     0.000,   0.089,   0.089      *
*                                                                             *
*    1       2       0.000,  -3.500,   3.500     0.000,  -0.089,   0.089      *
*                                                                             *
*    1       3       0.000,  -3.500,  -3.500     0.000,  -0.089,  -0.089      *
*                                                                             *
*    1       4       0.000,   3.500,  -3.500     0.000,   0.089,  -0.089      *
*                                                                             *
*******************************************************************************

*******************************************************************************
*                                                                             *
* SG:   SOURCE GEOMETRY                                                       *
*                                                                             *
*                                                                             *
*    THIS IS SOURCE NO.   1 IN THIS COMPUTATION.                              *
*                                                                             *
*                                                                             *
*    THIS IS AN ELECTRIC SOURCE OF TYPE  -2                                   *
*                                                                             *
*    SOURCE LENGTH=   0.50000 AND WIDTH=    0.00000 WAVELENGTHS               *
*                                                                             *
*    SOURCE LENGTH=   0.01874 AND WIDTH=    0.00000 METERS                    *
*                                                                             *
*    THE SOURCE WEIGHT HAS MAGNITUDE=   1.00000 AND PHASE=   0.00000          *
*                                                                             *
*                                                                             *
*   SOURCE#    INPUT LOCATION IN INCHES     ACTUAL LOCATION IN METERS         *
*   -------   -------------------------   --------------------------          *
*                                                                             *
*      1        5.120,   0.000,   0.000      0.130,   0.000,   0.000          *
*                                                                             *
*                                                                             *
*    THE FOLLOWING SOURCE ALIGNMENT IS USED:                                  *
*                                                                             *
*   VXSS(1,1, 1)= 1.00000  VXSS(1,2, 1)= 0.00000  VXSS(1,3, 1)= 0.00000       *
*                                                                             *
*   VXSS(2,1, 1)= 0.00000  VXSS(2,2, 1)= 1.00000  VXSS(2,3, 1)= 0.00000       *
```

```
*                                                                          *
*  VXSS(3,1, 1)= 0.00000  VXSS(3,2, 1)= 0.00000  VXSS(3,3, 1)= 1.00000     *
*                                                                          *
****************************************************************************


****************************************************************************
*                                                                          *
*  LP:   LINE PRINTER OUTPUT                                               *
*                                                                          *
*                                                                          *
*      DATA WILL BE OUTPUT ON LINE PRINTER !!!                             *
*                                                                          *
****************************************************************************


****************************************************************************
*                                                                          *
*  XQ:   EXECUTE CODE                                                      *
*                                                                          *
*                                                                          *
*********************************************************************************************


*********************************************************************************************
THE FAR ZONE ELECTRIC FIELD

THE FIELDS ARE REFERENCED TO THE PATTERN COORDINATE SYSTEM
```

		E-THETA			E-PHI		
THETA	PHI	MAGNITUDE	PHASE	DB	MAGNITUDE	PHASE	DB
90.00	0.00	8.8940E+00	-84.45	-9.79	8.8940E+00	-84.45	-9.79
90.00	10.00	4.7805E+01	173.96	4.82	4.7189E+01	175.34	4.70
90.00	20.00	9.7150E+01	-171.72	10.98	9.1802E+01	-172.26	10.48
90.00	30.00	1.6891E+01	63.95	-4.22	1.4346E+01	71.77	-5.64
90.00	40.00	4.5892E+01	-10.23	4.46	3.5318E+01	-8.07	2.19
90.00	50.00	4.9061E+01	168.09	5.04	3.2444E+01	165.52	1.45
90.00	60.00	3.7278E+01	-11.82	2.66	1.8220E+01	-8.70	-3.56
90.00	70.00	3.4701E+01	158.19	2.03	1.1771E+01	152.71	-7.36
90.00	80.00	3.9181E+01	-49.22	3.09	6.5564E+00	-40.56	-12.44
90.00	90.00	3.8790E+01	86.62	3.00	9.4583E-01	-114.94	-29.26
90.00	100.00	3.5702E+01	-123.80	2.28	5.1622E+00	55.67	-14.52
0							
90.00	110.00	4.0396E+01	18.61	3.35	1.4679E+01	-158.43	-5.44
90.00	120.00	4.1535E+01	-169.40	3.59	2.0830E+01	7.60	-2.40
90.00	130.00	3.2226E+01	18.32	1.39	2.0759E+01	-156.99	-2.43
90.00	140.00	2.4078E+01	-160.21	-1.14	1.7693E+01	16.28	-3.82
90.00	150.00	1.8294E+01	56.05	-3.53	1.5279E+01	-116.90	-5.09
90.00	160.00	1.8854E+01	-132.10	-3.27	1.7585E+01	44.68	-3.87
90.00	170.00	7.6055E+00	138.59	-11.15	6.9786E+00	-33.29	-11.90
90.00	180.00	1.8794E+01	-28.10	-3.29	1.8794E+01	151.90	-3.29
90.00	190.00	7.6055E+00	138.59	-11.15	6.9786E+00	-33.29	-11.90
90.00	200.00	1.8854E+01	-132.10	-3.27	1.7585E+01	44.68	-3.87
0							
90.00	210.00	1.8294E+01	56.05	-3.53	1.5279E+01	-116.90	-5.09
90.00	220.00	2.4078E+01	-160.21	-1.14	1.7693E+01	16.28	-3.82
90.00	230.00	3.2226E+01	18.32	1.39	2.0759E+01	-156.99	-2.43
90.00	240.00	4.1535E+01	-169.40	3.59	2.0830E+01	7.60	-2.40
90.00	250.00	4.0396E+01	18.62	3.35	1.4679E+01	-158.43	-5.44
90.00	260 00	3.5702E+01	-123.80	2.28	5.1622E+00	55.67	-14.52
90.00	270.00	3.8790E+01	86.62	3.00	9.4582E-01	65.06	-29.26
90.00	280.00	3.9181E+01	-49.22	3.09	6.5564E+00	-40.56	-12.44
90.00	290.00	3.4701E+01	158.19	2.03	1.1771E+01	152.71	-7.36
90.00	300.00	3.7278E+01	-11.82	2.66	1.8220E+01	-8.70	-3.56
0							
90.00	310.00	4.9061E+01	168.09	5.04	3.2444E+01	165.52	1.45
90.00	320.00	4.5892E+01	-10.23	4.46	3.5318E+01	-8.07	2.19
90.00	330.00	1.6891E+01	63.95	-4.22	1.4346E+01	71.77	-5.64

90.00	340.00	9.7150E+01	-171.72	10.98	9.1802E+01	-172.26	10.48
90.00	350.00	4.7805E+01	173.96	4.82	4.7189E+01	175.34	4.70

```
*******************************************************************************************************
```

TOTAL RADIATION INTENSITY IN DB

THE FIELDS ARE REFERENCED TO THE PATTERN COORDINATE SYSTEM

THETA	PHI	MAJOR	MINOR	TOTAL	AXIAL RATIO	TILT ANG	SENSE
90.00	0.00	-6.78	-100.00	-6.78	0.00000	45.00	LINEAR
90.00	10.00	7.77	-30.59	7.77	0.01208	44.63	LEFT
90.00	20.00	13.75	-32.82	13.75	0.00469	43.38	RIGHT
90.00	30.00	-1.88	-25.30	-1.86	0.06744	40.30	LEFT
90.00	40.00	6.48	-28.31	6.48	0.01823	37.58	LEFT
90.00	50.00	6.61	-27.10	6.62	0.02063	33.47	RIGHT
90.00	60.00	3.58	-29.76	3.59	0.02152	26.03	LEFT
90.00	70.00	2.50	-28.23	2.51	0.02905	18.68	RIGHT
90.00	80.00	3.21	-29.00	3.21	0.02453	9.40	LEFT
90.00	90.00	3.00	-37.96	3.00	0.00895	-1.30	LEFT
90.00	100.00	2.37	-55.23	2.37	0.00132	-8.23	LEFT
0							
90.00	110.00	3.89	-31.73	3.89	0.01657	-19.95	RIGHT
90.00	120.00	4.57	-29.01	4.57	0.02095	-26.61	LEFT
90.00	130.00	2.89	-25.68	2.90	0.03727	-32.75	RIGHT
90.00	140.00	0.73	-29.94	0.73	0.02926	-36.29	LEFT
90.00	150.00	-1.25	-25.59	-1.23	0.06064	-39.83	RIGHT
90.00	160.00	-0.55	-31.60	-0.55	0.02804	-43.00	LEFT
90.00	170.00	-8.52	-31.54	-8.50	0.07066	-42.51	RIGHT
90.00	180.00	-0.28	-100.00	-0.28	0.00000	-45.00	LINEAR
90.00	190.00	-8.52	-31.54	-8.50	0.07066	-42.51	RIGHT
90.00	200.00	-0.55	-31.60	-0.55	0.02804	-43.00	LEFT
0							
90.00	210.00	-1.25	-25.59	-1.23	0.06064	-39.83	RIGHT
90.00	220.00	0.73	-29.94	0.73	0.02926	-36.29	LEFT
90.00	230.00	2.89	-25.68	2.90	0.03727	-32.75	RIGHT
90.00	240.00	4.57	-29.01	4.57	0.02095	-26.61	LEFT
90.00	250.00	3.89	-31.73	3.89	0.01656	-19.95	RIGHT
90.00	260.00	2.37	-55.23	2.37	0.00132	-8.23	LEFT
90.00	270.00	3.00	-37.96	3.00	0.00895	1.30	RIGHT
90.00	280.00	3.21	-29.00	3.21	0.02453	9.40	LEFT
90.00	290.00	2.50	-28.23	2.51	0.02905	18.68	RIGHT
90.00	300.00	3.58	-29.76	3.59	0.02152	26.03	LEFT
0							
90.00	310.00	6.61	-27.10	6.62	0.02063	33.47	RIGHT
90.00	320.00	6.48	-28.31	6.48	0.01823	37.58	LEFT
90.00	330.00	-1.88	-25.30	-1.86	0.06744	40.30	LEFT
90.00	340.00	13.75	-32.83	13.75	0.00469	43.38	RIGHT
90.00	350.00	7.77	-30.59	7.77	0.01208	44.63	LEFT

```
*******************************************************************************************************

*****************************************************************************
*                                                                           *
*  EN:   END CODE                                                           *
*                                                                           *
*                                                                           *
*****************************************************************************
```

Appendix B. *Precision Comparison*

B.0.1 VAX '0386

```
*******************************************************************************
*  NEC-BSC      3.2i4.0,  6 May 91                                            *
*                                                                             *
*  CE:   FAR ZONE PLATE TEST, EXAMPLE 1C.                                     *
*******************************************************************************
```

THE FAR ZONE ELECTRIC FIELD

THE FIELDS ARE REFERENCED TO THE PATTERN COORDINATE SYSTEM

		E-THETA			E-PHI		
THETA	PHI	MAGNITUDE	PHASE	DB	MAGNITUDE	PHASE	DB
90.00	0.00	8.8941E+00	-84.45	-9.79	8.8941E+00	-84.45	-9.79
90.00	10.00	4.7239E+01	174.14	4.71	4.6800E+01	175.80	4.63
90.00	20.00	9.7978E+01	-171.28	11.05	9.2693E+01	-171.70	10.57
90.00	30.00	1.6908E+01	61.19	-4.21	1.4053E+01	68.78	-5.82
90.00	40.00	4.5705E+01	-10.26	4.43	3.4954E+01	-8.62	2.10
90.00	50.00	4.8741E+01	167.18	4.98	3.2245E+01	164.18	1.40
90.00	60.00	3.7481E+01	-11.56	2.70	1.8403E+01	-8.88	-3.48
90.00	70.00	3.4858E+01	157.66	2.07	1.2208E+01	152.74	-7.04
90.00	80.00	3.8895E+01	-48.68	3.02	6.0098E+00	-38.92	-13.20
90.00	90.00	3.9183E+01	86.42	3.09	1.3009E+00	-97.21	-26.49
90.00	100.00	3.5275E+01	-124.09	2.18	4.9598E+00	49.57	-14.86

B.0.2 i860

```
*******************************************************************************
*  NEC-BSC      3.2i4.0,  6 May 91                                            *
*                                                                             *
*  CE:   FAR ZONE PLATE TEST, EXAMPLE 1C.                                     *
*******************************************************************************
```

THE FAR ZONE ELECTRIC FIELD

THE FIELDS ARE REFERENCED TO THE PATTERN COORDINATE SYSTEM

		E-THETA			E-PHI		
THETA	PHI	MAGNITUDE	PHASE	DB	MAGNITUDE	PHASE	DB
90.00	0.00	8.8940E+00	-84.45	-9.79	8.8940E+00	-84.45	-9.79
90.00	10.00	4.7805E+01	173.96	4.82	4.7189E+01	175.34	4.70
90.00	20.00	9.7150E+01	-171.72	10.98	9.1802E+01	-172.26	10.48
90.00	30.00	1.6891E+01	63.95	-4.22	1.4346E+01	71.77	-5.64
90.00	40.00	4.5892E+01	-10.23	4.46	3.5318E+01	-8.07	2.19
90.00	50.00	4.9061E+01	168.09	5.04	3.2444E+01	165.52	1.45
90.00	60.00	3.7278E+01	-11.82	2.66	1.8220E+01	-8.70	-3.56
90.00	70.00	3.4701E+01	158.19	2.03	1.1771E+01	152.71	-7.36
90.00	80.00	3.9181E+01	-49.22	3.09	6.5564E+00	-40.56	-12.44
90.00	90.00	3.8790E+01	86.62	3.00	9.4583E-01	-114.94	-29.26
90.00	100.00	3.5702E+01	-123.80	2.28	5.1622E+00	55.67	-14.52

Appendix C. *Code Samples*

C.1 Mod4.0 Host Main Routine

```
      Program run_bscnode
CCC
C!!!  Host Program for NECBSC-iPSC/2 & i860
C!!!
C!!!  Get Input_File_Name, output decision, & size of cube from user
C!!!  Get & Load Cube
C!!!  Get start & end time from each node
C!!!  Receive data from nodes, Print Header and Data to file
C!!!  Calculate Elapsed time for each node
C!!!  Calculate CPU Time for host
C!!!  ------------------------------------------------------------------
C!!!  Version and date information.
        CHARACTER VERDAT*18
        PARAMETER (VERDAT='3.2i4.0,  6 May 91')
C!!!             (       '                  ')
        CHARACTER VERDTE*18
        COMMON/VRSDAT/VERDTE
integer MSG_LENGTH, START, STARTUP, MTIME
integer STARTIME(129), TOTIME(129), TIME, MIN_TIME, HOST_START
character NAMINP*70, LPRSAV*70, NAMOPN*70, NAMCOR*70, MSG*70
character STTOPN*7, FRMOPN*15
CCC!  Pattern information.
        INTEGER NPN NPV
        LOGICAL LPATR,LPATS,LVOLP
        COMMON/OUTPNV/NPN,NPV,LVOLP,LPATS,LPATR
C!!!  Frequency information.
        INTEGER NFQG
        LOGICAL LFQG
        REAL FQGI,FQGS,FRQG
        COMMON/OUTPFQ/FRQG,FQGS,FQGI,NFQG,LFQG
C!!!  Test information.
        LOGICAL LDEBUG,LOUT,LTEST,LWARN
        COMMON/TEST/LDEBUG,LTEST,LWARN,LOUT
c-----------------------------------------------------------------------
CCC   Node Information.
      character CUBE_TYPE*3
      integer INT_SIZE, REAL_SIZE
      integer MY_HOST, NUM_NODES, PER _NODE, NODE, INDEX8, INDEX2
      common/iPSC/MY_HOST,NUM_NODES, PER_NODE, INT_SIZE, REAL_SIZE
C!!!
      integer IUI,IUO,IUP,IUG,IUT,IUW,IURI,IUSI
      COMMON/INOUT/IUG,IUI,IUO,IUP,IURI,IUSI,IUT,IUW
      real OUTREAL
      integer OUT_START, HOST_START
      INT_SIZE=4
      REAL_SIZE=4
      HOST_START=mclock()
      VERDTE=VERDAT
C!!!
C!!!   Print Header Info to Screen
C!!!
      write(*,*)'  * NEC-BSC for iPSC/2 & iPSC/860 * ',VERDTE,'  *'
```

```
      write(*,*)'  * Scott Suhr & Gary B. Lamont '
      write(*,*)'    AFIT School of Engineering'
      write(*,*)'    - Banner written to disk from host'
      write(*,*)'    - Numeric data sent to host in real vector &'
      write(*,*)'      written to disk from host '
C!!!-------------------------------------------------------------------
C!!!  Get # of nodes from user & get_cube
C!!!
      write(*,*)'Input desired Cube-Type (8rx, 4sx, etc):'
      READ(*,*) CUBE_TYPE
      call getcube('necbsc',CUBE_TYPE,' ',0)
      call setpid(81)
      NUM_NODES=numnodes()
      write(*,*)'Number of nodes attached:', NUM_NODES
      if (NUM_NODES.LT.1) goto 999
C!!!-------------------------------------------------------------------
C!!!  Get filename from user & parse for "core" name
C!!!
 30   call Parse_Filenames(IC,NAMCOR)
C!!!-----------------------------------------------------------
C!!!  Open input file & call Parse_Filenames again if necessary
C!!!
NAMOPN(1:IC)=NAMCOR
NAMOPN(IC+1:IC+4)='.inp'
STTOPN(1:7)='OLD    '
FRMOPN(1:15)='UNFORMATTED'
        OPEN(UNIT=IUI,FILE=NAMOPN,FORM=FRMOPN,
     2                  STATUS=STTOPN,IOSTAT=IERR)
        IF (IERR.GT.0) THEN
          WRITE(*,*) ' CAN NOT OPEN FILE:  ',NAMOPN
          NAMOPN(1:ICX)=' '
          WRITE(*,FMT='(A)') ' TRY AGAIN'
          GO TO 30
        END IF
write(*,*)'Input Filename = "',NAMOPN(1:IC+4),'"'
C!!!-----------------------------------------------------------
C!!!  Send sizeof(NAMCOR)=IC and NAMCOR to nodes
C!!!
      call isend(100,IC,INT_SIZE,-1,0)
      call isend(101,NAMCOR,IC,-1,0)
C!!!-----------------------------------------------------------
      LPRSAV='Y'
      call load('bscnode', -1, 0)
C!!!-----------------------------------------------------------
C!!!  Open Single Output File
C!!!
if ((LPRSAV.NE.'N').AND.(LPRSAV.NE.'n')) then
  NAMOPN(1:IC)=NAMCOR(1:IC)
  NAMOPN(IC+1:IC+4)='.OUT'
  STTOPN(1:7)='UNKNOWN'
          FRMOPN(1:11)='FORMATTED  '
        OPEN(UNIT=IUO,FILE=NAMOPN,FORM=FRMOPN,
     2                STATUS=STTOPN,IOSTAT=IERR)
          if (IERR.NE.0) then
```

```
            WRITE(*,FMT='(A)') ' CAN NOT OPEN FILE "',NAMOPN,'"'
    STOP
  end if
c else
c   IUO=6
c   IUW=6
  end if
C!!!-------------------------------------------------------
C!!!  Initialize variables used by CREAD & OUTWRT
C!!!
        VERDTE=VERDAT
call CMDNX
goto 1001
C!!!-------------------------------------------------------
C!!!  Loop back to here if multiple runs in input deck
C!!!
 1000 continue
        HOST_START=mclock()
 1001   continue
C!!!-------------------------------------------------------
C!!!  Read input deck & produce header info
C!!!
c write(*,*)'Host:  call CREAD'
  call CREAD
write(*,*)'Host:  CREAD Complete'
STARTUP=mclock()-HOST_START
C!!!-------------------------------------------------------
C!!!  Get start time from each node
C!!!
      do 100 j=1,NUM_NODES
         call crecv(102,TIME,INT_SIZE)
         i=infonode()
         STARTIME(i+1)=TIME
 100    continue
C!!!-------------------------------------------------------
C!!!  Receive output data & write/print output files
C!!!
OUT_START=mclock()
C!!!  Pick number of idices for actual output
          IF (LFQG) THEN
            NFP1=NFQG
          ELSE
C!!!  Initialize pattern point number.
         IF (LDEBUG.OR.LTEST.OR.LFQG) THEN
           NFP1=1
         ELSE
           NFP1=NPN
         END IF
          END IF
C!!!
c write(*,*)'Call OUTWRT'
call OUTWRT(NFP1)
c write(*,*)'Host:  OUTWRT complete'
C!!!-------------------------------------------------------
```

```
C!!!  Get end time from each node,
C!!!  calculate total, & print to screen
C!!!
c goto 1000
      do 900 j=1,NUM_NODES
call crecv(200,TIME,INT_SIZE)
        i=infonode()
TOTIME(i+1)=TIME-STARTIME(i+1)
c  write(*,*) i, STARTIME(i+1), TIME, TOTIME(i+1)
 900  continue

      write(*,*)'Elapsed:  Node     Total Time (msec)'
      do 901 i=1,NUM_NODES
         write(*,*) i-1,'   ', TOTIME(i)
 901  continue

call crecv(201,TIME,INT_SIZE)
      write(*,'(A,i7/)'),'Output elapsed time (node 0, msec):', TIME
C!!!--------------------------------------------------------------
C!!!  Get Host elapsed times & print out
C!!!
      write(*,'(A,i7)'),' Host CPU time required for startup:', STARTUP
MTIME=mclock()-OUT_START
      write(*,'(A,i7)'),' Host CPU time required for output: ', MTIME
TIME=mclock()-HOST_START
      write(*,'(A,i7)'),'      Total Host CPU time required:', TIME
TIME=STARTUP+MTIME+TOTIME(1)
      write(*,'(A,i7)'),'Approx. total elapsed time required:', TIME
      write(*,*)       '     (node 0 + Startup + Output)'
C!!!
C!!!  Go back to next problem or exit normally from CREAD.CMD.EN
C!!!
goto 1000
 999    continue
C!!!--------------------------------------------------------------
C!!!  Close input & output files after fatal error
C!!!
call FLCLS(IUI)
call FLCLS(IUO)

end
C!!!==================================================================
```

C.2 Mod4.0 Node Main Routine Excerpts

```
.
.
.
C!!!
C!!!  0. iPSC STARTUP/OVERHEAD SECTION
C!!!
      INT_SIZE=4
      REAL_SIZE=4
      MY_HOST=myhost()
      MY_NODE=mynode()
      TIME=mclock()
C!!!
C!!!  Send Start-Time to host
C!!!
      call isend(102,TIME,INT_SIZE,MY_HOST,81)
C!!!
C!!!  1. INPUT SECTION
C!!!
C!!!  Initialize the input commands.
C!!!
        VERDTE=VERDAT
        CALL CMDNX
C!!!
C!!!  Open read and write files.
C!!!
        CALL FLOPN(IUI)
CALL FLOPN(IUO)
        IF(IUW.NE.IUO) CALL FLOPN(IUW)
C!!!-----------------------------------------------
C!!!  Read input commands.
C!!!
2999    CALL CREAD

C!!!
C!!!  2. INITIALIZATION SECTION
C!!!
.
.
.
C!!!
C!!!  3.  MAIN COMPUTATION SECTION
C!!!
C!!!  Loop thru volumetric pattern points.
        DO 1190 IIV=1,NPV
C!!!  Set E fields to zero.
          DO 2 I=1,NOX
            CT(I)=(0.,0.)
            DO 1 N=1,3
              ET(N,I)=(0.,0.)
              HT(N,I)=(0.,0.)
1           CONTINUE
2         CONTINUE
```

```
C!!!  Initialize arrays used to define double diffraction sectors.
          DO 42 J=1,NPX
            DO 41 I=1,NEX
              ID(I,J)=-1
41          CONTINUE
42        CONTINUE
          DO 43 I=1,NOX
            IDD(I)=0
43        CONTINUE
          IF (LTERM) THEN
            CALL FLOPN(IUG)
            CALL FLOPN(IUT)
          END IF
C!!!  Loop thru individual GTD fields.
          ITERM=0
1150      ITERM=ITERM+1
          CALL CTERMS(ITERM)
          IF (IIV.EQ.1) THEN
            if(W) WRITE(UNIT=IUO,FMT='(A,5X,A,T79,A)')
     2        ' *',CTERM,'*'
            CALL GETCP(IDTIM)
          END IF
          IF (CTERM.EQ.'END') GO TO 1160
C!!!  Loop thru sources when they do not move.
          DO 1250 MSF=1,MSXF
            IF (.NOT.LPATS) THEN
              MS=MSF
C!!!  Specify source geometry.
              CALL GEOMS
C!!!  Define various geometry properties of structure relative to the
C!!!  source.
              IF (LPLA) CALL GEOMPS
              IF (LCYL) CALL GEOMCS
              IF (LPLA.AND.LCYL) CALL GEOPCS
            END IF
C!!!  Loop thru pattern points.
            DO 1100 IIC=1,NPNP
              II=IIC
C!!!  If source moves, determine source point in reference coordinate
C!!!  system.
              IF (LPATS) THEN
                CALL PATPT(DS,RS,VPS,XPTS,VPTS,PATS,IPTS,LRCTS,LNEAS
     2          ,IIC,IIV)
              END IF
C!!!  If observer moves, determine observation point in reference
C!!!  coordinate system.
              IF (LPATR) THEN
                CALL PATPT(DR,RR,VPR,XPTR,VPTR,PATR,IPTR,LRCTR,LNEAR
     2          ,IIC,IIV)
              END IF
C!!!  Loop thru various sources if they move.
              DO 1200 MSM=1,MSXM
                IF (LPATS) THEN
                  MS=MSM
```

```
C!!! Specify source geometry.
            CALL GEOMS
C!!! Define various geometry properties of structure relative to the
C!!! source.
            IF (LPLA) CALL GEOMPS
            IF (LCYL) CALL GEOMCS
            IF (LPLA.AND.LCYL) CALL GEOPCS
          END IF
C!!! Loop thru receivers
          DO 1170 MR=1,MRXP
C!!! Specify receiver location.
            CALL GEOMR
C!!! Loop on frequencies, if specified.
            DO 1175 IIQ=1,NFQG
              IF (LFQG) THEN
                FQG=FQGS+FQGI*(IIQ-1)
                WL=.2997925/FQG
                WK=TPI/WL
              END IF
C!!! Initialize individual UTD field type storage variables.
              CTT=(0.,0.)
              DO 1116 N=1,3
                HTT(N)=(0.,0.)
                ETT(N)=(0.,0.)
1116          CONTINUE
C!!! Calculate fields for the different UTD terms.
              CALL CFIELD(CTERM,CTRM)
C!!! Pick storage idex.
              IF (LFQG) THEN
                IK=IIQ
              ELSE
                IK=IIC
              END IF
C!!! Write out subtotal fields
              IF (LOUT) THEN
                CALL PRIOUT('SUBTOTAL',ITERM,IK,0,0,ETT)
                IF (LRCVR) THEN
                  CALL PRIOUC('SUBTOTAL',ITERM,IK,0,0,CTT)
                END IF
              END IF
C!!! Superposition of the field components, and conversion of
C!!! the polarization to the pattern cut coordinate system.
              IF (LRCVR) THEN
                CT(IK)=CT(IK)+CTT
              ELSE IF (LSHDW) THEN
                DO 1205 NJ=1,3
                  DO 1204 NI=1,3
                    ET(NJ,IK)=ET(NJ,IK)+ETT(NI)
                    HT(NJ,IK)=HT(NJ,IK)+HTT(NI)
1204              CONTINUE
1205            CONTINUE
              ELSE
                DO 1203 NJ=1,3
                  DO 1202 NI=1,3
```

```
                    ET(NJ,IK)=ET(NJ,IK)+ETT(NI)*VPR(NJ,NI)
                    HT(NJ,IK)=HT(NJ,IK)+HTT(NI)*VPR(NJ,NI)
1202              CONTINUE
1203            CONTINUE
              END IF
C!!!  End of loop on frequencies.
1175          CONTINUE
C!!!  End of loop on receivers.
1170        CONTINUE
C!!!  End of loop on moving sources.
1200      CONTINUE
C!!!  End of loop on pattern cut points.
1100    CONTINUE
C!!!  End of loop on fixed sources.
1250  CONTINUE
      IF (IIV.EQ.1) THEN
        CALL GETCP(IETIM)
        IFTIM=IETIM-IDTIM
        CTIME=0.166667E-3*IFTIM
        IF (LTEST.OR.LDEBUG.OR.LOUT) THEN
          if(W) WRITE(UNIT=IUO,FMT='(A,5X,2A,F11.5,A,T79,A)')
     2     ' *',CTERM,' = ',CTIME,' CPU MINUTES','*'
        ELSE
          if(W) WRITE(UNIT=IUO,FMT='(1H+,A,5X,2A,F11.5,A,T79,A)')
     2     '*',CTERM,' = ',CTIME,' CPU MINUTES','*'
        END IF
      END IF
C!!!  Go get another UTD term.
      GO TO 1150
C!!!  End of loop on UTD terms.
1160  CONTINUE
      IF (LTERM) THEN
        CALL FLCLS(IUG)
        CALL FLCLS(IUT)
      END IF
C!!!  Pick number of idices
      IF (LFQG) THEN
        NFP=NFQG
      ELSE
        NFP=NPNP
      END IF
C!!!  Extended field output, if specified.
      IF (LOUT) THEN
        DO 1206 II=1,NFP
          IF (LRCVR) THEN
            CALL PRIOUC('TOTAL   ',II,II,II,II,CT(II))
          ELSE
            CALL PRIOUT('TOTAL   ',II,II,II,II,ET(1,II))
          END IF
1206    CONTINUE
      END IF
C!!!  Calculate cpu time for each subroutine.
CCC!  seconds instead of minutes
C!!!  -------------------------
```

```
          IF (IIV.EQ.NPV) THEN
            CALL GETCP(IBTIM)
            ICTIM=IBTIM-IATIM
c            CTIME=0.166667E-3*ICTIM
            CTIME=0.1*ICTIM
            if(W) WRITE(UNIT=IUO,FMT=FMBOX)
            if(W) WRITE(UNIT=6,FMT='(A,5X,A,F11.5,A,T79,A)')
     2       ' *','CPU TIME FOR FIELD EXECUTION = ',CTIME
     3       ,' SECONDS','*'
c     3       ,' MINUTES','*'
            if(W) WRITE(UNIT=IUO,FMT=FMBOX)
          END IF
C!!! Results are sent to unit IUO -- Output File
          IF (LWRITE) THEN
            CALL OUTWRT(ET,HT,CT,NFP,IIV)
          END IF
C!!! Write plot data to file, if desired.
C!!! Note that the plot routines are not included, since they can
C!!! not be used on all systems. The user's own plot algorithms
C!!! can be interfaced through the plot data files.
C!!!
CCC ''PP'' Disabled
c          IF (LVPLT.OR.LPLT) THEN
c            IF (IIV.EQ.1) CALL FLOPN(IUP)
c            CALL OUTPLT(ET,HT,CT,CTIME,NFP,IIV)
c            IF (IIV.EQ.NPV) CALL FLCLS(IUP)
c          END IF

C!!! End of volumetric pattern loop.
1190    CONTINUE

CCC--------------------------------------------
C!!! iPSC CLEAN-UP SECTION
C!!! (send end-time to host)
C!!!
        TIME=mclock()
        call csend(200,TIME,INT_SIZE,MY_HOST,81)
C!!! Return to read more commands (second input deck or END).
        GO TO 2999
C!!!
C!!! ----------------------- END MAIN PROGRAM -------------------------
C!!!
        END
```

C.3 Mod4.0 Host Output Routine Excerpts

```
CCC-----------------------------------------------------------------------
       SUBROUTINE OUTWRT(NFP1)
C!!!  ------------
C!!! ---- HOST VERSION ----  mod4.0 6 May 91
C!!!  ------------
C!!!  This subroutine is used to output coupling and field
C!!!  pattern data to a disk file.
C+++
C+++  Specification of maximum dimension sizes.
C+++
C+++  Maximum dimension for observation points.
       INTEGER NOX
       PARAMETER (NOX=1801)
.
.
.
CCC------------------------------------------
C!!!  Receive length of array(s) to be passed
C!!!
call crecv(900,NFP,INT_SIZE)
      NFPM1 = NFP-1
 NFP_REAL = NFP*REAL_SIZE
C!!!
CCC-----------------------------------------------------------------------

       IF (.NOT.LNEAR) THEN

CCC------------------------------------------------------------------
CCC--------------------------------------------
C!!!    Output E-theta and E-phi representations.
C!!!--------------------------------------------
CCC------------------------------------------
C!!!    Length of array(s) to be passed
C!!!
 if (LPATS) OUT_SIZE2=2*NFP_REAL
 OUT_SIZE7=7*NFP_REAL
 OUT_SIZE8=8*NFP_REAL
CCC------------------------------------------
C!!!    Write Header
C!!!
         WRITE(UNIT=IUO,FMT=FLINE)
         WRITE(UNIT=IUO,FMT='(/)')
         WRITE(UNIT=IUO,FMT=FLINE)
         WRITE(UNIT=IUO,FMT='(A,/)')
     2     ' THE FAR ZONE ELECTRIC FIELD '
         WRITE(UNIT=IUO,FMT='(2A,/)')
     2     ' THE FIELDS ARE REFERENCED TO THE PATTERN COORDINATE'
     3     ,' SYSTEM '
         WRITE(UNIT=IUO,FMT='(41X,A,31X,A)') 'E-THETA','E-PHI'
         IF (LFQG) THEN
           WRITE(UNIT=IUO
     2       ,FMT='(11X,A,14X,A,5X,A,6X,A,11X,A,5X,A,6X,A)')
```

```
     3       'FREQ.','MAGNITUDE','PHASE','DB'
     4       ,'MAGNITUDE','PHASE','DB'
          ELSE
            WRITE(UNIT=IUO
     2       ,FMT='(6X,A,6X,A,10X,A,5X,A,6X,A,11X,A,5X,A,6X,A)')
     3       'THETA','PHI','MAGNITUDE','PHASE','DB'
     4       ,'MAGNITUDE','PHASE','DB'
          END IF
          IMAX=11
C!!!
C!!!  End Write Header
CCC-----------------

CCC---------------------------------------
C!!!  Receive Far Zone array(s) #1 from node
C!!!
write(*,*)'Receive Far Zone E-Field arrays from nodes'
do 930 i2=1,NUM_NODES
             IF (LFQG) THEN
     call crecv(902,OUTREAL,OUT_SIZE7)
        INDEX8=infonode()+1
             ELSE
          IF (LPATS) call  crecv(901,OUTREAL2,OUT_SIZE2)
INDEX2=infonode()+1
   call crecv(902,OUTREAL,OUT_SIZE8)
        INDEX8=infonode()+1
             END IF
c      write(*,*)'Received  OUTREAL(2,2)=',OUTREAL(2,2)

do 931 i3=0,NFPM1
i37=i3*7
i38=i3*8
i3P1 = i3+1
             IF (LFQG) THEN
  do 929 i4=1,7
      OUTREAL8N(INDEX8,i4,i3P1) = OUTREAL(i37+i4)
 929   continue
            ELSE
  if(LPATS) then
    OUTREAL2N(INDEX2,1,i3P1)=OUTREAL2(i3*2+1)
    OUTREAL2N(INDEX2,2,i3P1)=OUTREAL2(i3*2+2)
  end if
  do 932 i5=1,8
      OUTREAL8N(INDEX8,i5,i3P1) = OUTREAL(i38+i5)
 932   continue
END IF
 931      continue
 930 continue
write(*,*)'Arrays received'
C!!!
C!!! End Receive Far Zone Data #1
CCC---------------------------------------------------

CCC--------------------------------
```

```
C!!   Write Out Far Zone Data #1 --
C!!
CCC  initialize maximums
          ETHMX=OUTREAL8N(1,3,1)*RANG
          EPHMX=CUTREAL8N(1,6,1)*RANG
          ETOTMX=ETHMX*ETHMX+EPHMX*EPHMX

          DO 2 I=1,NFP
   do 2 II=1,NUM_NODES
   ITEMP=((I-1)*NUM_NODES+II)
   if (ITEMP.GT.NFP1) goto 299
            IF (LFQG) THEN
CCC!  load node data back into appropriate variables
 FQG   = OUTREAL8N(II,1,I)
 ETHMR = OUTREAL8N(II,2,I)
 ETHP  = OUTREAL8N(II,3,I)
 ETHDB = OUTREAL8N(II,4,I)
 EPHMR = OUTREAL8N(II,5,I)
 EPHP  = OUTREAL8N(II,6,I)
   EPHDB = OUTREAL8N(II.7.I)
              WRITE(UNIT=IUO
     2          ,FMT='(1H ,6X,F9.3,5X,2(7X,1PE11.4,3X,0PF7.2,3X,F7.2))')
     3          FQG,ETHMR,ETHP,ETHDB,EPHMR,EPHP,EPHDB
            ELSE
              IF (LPATS) THEN
CCC!  load node data back into appropriate variables
PSA(2) = OUTREAL2N(II,1,I)
PSA(3) = OUTREAL2N(II,2,I)
                WRITE(UNIT=IUO,FMT='(1H ,2(3X,F7.2))') PSA(2),PSA(3)
              END IF
CCC!  load node data back into appropriate variables
PRA(2) = OUTREAL8N(II,1,I)
PRA(3) = OUTREAL8N(II,2,I)
ETHMR  = OUTREAL8N(II,3,I)
ETHP   = OUTREAL8N(II,4,I)
ETHDB  = OUTREAL8N(II,5,I)
EPHMR  = OUTREAL8N(II,6,I)
EPHP   = OUTREAL8N(II,7,I)
EPHDB  = OUTREAL8N(II,8,I)
C!!!
              WRITE(UNIT=IUO
     2          ,FMT='(1H ,2(3X,F7.2),2(7X,1PE11.4,3X,0PF7.2,3X,F7.2))')
     3          PRA(2),PRA(3),ETHMR,ETHP,ETHDB,EPHMR,EPHP,EPHDB
            END IF
            IF (ITEMP.GT.IMAX) IMAX=IMAX+10
            IF (ITEMP.EQ.IMAX)  WRITE(UNIT=IUO,FMT='(1H0)')
C!!!  Find maximums.
            IF (ETHM.GT.ETHMX) ETHMX=ETHM
            IF (EPHM.GT.EPHMX) EPHMX=EPHM
            ETOT2 = ETHM*ETHM + EPHM*EPHM
            IF (ETOT2.GT.ETOTMX) ETOTMX = ETOT2
 299   continue
 2        CONTINUE
```

```
CCC----------------------------------------------
C!!!  Output Far Zone total field representations.
CCC----------------------------------------------
C!!!  Write Header
C!!!
.
.
.
C!!!
C!!!  End write header
CCC------------------

CCC------------------------------------------
C!!!  Receive Far Zone total field array(s) from nodes
C!!!
.
.
.
CCC-------------------------------------

CCC-----------------------------------------------------------------
C!!!  Write out FZ data set #2
C!!!
.
.
.
C!!!
CCC------------------------------------------------------------------

        ELSE IF (.NOT.LRCVR) THEN

CCC------------------------------------------------------------------
CCC-----------------------------------
CCC  Near zone E-field representation.
CCC-----------------------------------
.
.
.
CCC-------------------------------------
C!!!  Near zone H-field representations.
CCC---------------
.
.
.
CCC----------------------------------------------------------------
CCC----------------------------------
C!!!  Near zone power representation.
CCC----------------------------------
.
.
.
CCC----------------------------------------------------------------

        ELSE
```

```
CCC---------------------------------------------------------------------
CCC-----------------------------------------------
C!!!  Antenna to antenna coupling representation.
CCC-----------------------------------------------
.
.
.
          WRITE(UNIT=IUO,FMT=FLINE)
        END IF
c write(*,*)'32h.f:  OUTWRT Complete'
        RETURN
        END
```

```
        SUBROUTINE OUTWRT (ET,HT,CT,NFP,IIV)
C!!!  ----------------
C!!!  -  NODE VERSION -  mod4.0 6 May 91
C!!!  ----------------
C!!!  This subroutine is used to output coupling and field
C!!!  pattern data to the host for output to disk.
C!!!
C+++
C+++  Specification of maximum dimension sizes.
C+++
C+++  Maximum dimension for observation points.
        INTEGER NOX
        PARAMETER (NOX=1801)
.
.
.
START_TIME = mclock()
INT_SIZE=4
REAL_SIZE=4
MY_NODE=mynode()
CCC-------------------------------------------
C!!!  Send length of data vectors to be transmitted
C!!!
call csend(900,NFP,INT_SIZE,MY_HOST,81)
NFP_REAL = NFP*REAL_SIZE
CCC-------------------------------------------
        IF (.NOT.LPATR) THEN
          DO 99 N=1,3
            PRA(N)=XR(N)
99        CONTINUE
        END IF
CCC-------------------------------------------------------------

        IF (.NOT.LNEAR) THEN

CCC-------------------------------------------------------------
C!!!  Set up constants for the far zone.
CCC-------------------------------------------------------------
          FRANG=CMPLX(1.,0.)
          IF (LRANG) THEN
            RANGL=RANG/WL-AINT(RANG/WL)
            FRANG=CEXP(CMPLX(0.,-TPI*RANGL))
          END IF
          IF (IPRAD.EQ.1) THEN
            FACP=1./(60.*PRADS)
          ELSE
            FACP=1./(240.*PI)
          END IF
          FACS=SQRT(FACP)
C!!!  Find maximums.
          ETHMX=BABS(ET(2,1))
          EPHMX=BABS(ET(3,1))
```

```
          ETOTMX=ETHMX*ETHMX+EPHMX*EPHMX
          DO 1 I=1,NFP
            ETHM=BABS(ET(2,I))
            IF (ETHM.GT.ETHMX)  ETHMX=ETHM
            EPHM=BABS(ET(3,I))
            IF (EPHM.GT.EPHMX) EPHMX=EPHM
            ETOT2=ETHM*ETHM+EPHM*EPHM
            IF (ETOT2.GT.ETOTMX) ETOTMX=ETOT2
 1        CONTINUE
          IMAX=11
CCC----------------------------------------------
C!!!  Output E-theta and E-phi representations.
CCC----------------------------------------------
C!!!   Length of vector(s) to be passed
C!!!
  if (LPATS) OUT_SIZE2 = 2*NFP_REAL
  OUT_SIZE7 = 7*NFP_REAL
  OUT_SIZE8 = 8*NFP_REAL
C!!!
          DO 2 I=1,NFP
IM1=I-1
I2=IM1*2
I7=IM1*7
I8=IM1*8
            ETHR=ET(2,I)*FRANG
            ETHM=BABS(ET(2,I))
            ETHMR=ETHM/RANG
            ETHP=DPR*BTAN2(AIMAG(ETHR),REAL(ETHR))
            ETHDB=20.*BLOG10(FACS*ETHM)
            EPHR=ET(3,I)*FRANG
            EPHM=BABS(ET(3,I))
            EPHMR=EPHM/RANG
            EPHP=DPR*BTAN2(AIMAG(EPHR),REAL(EPHR))
            EPHDB=20.*BLOG10(FACS*EPHM)
            IF (LFQG) THEN
              FQG=FQGS+FQGI*(I-1)
CCC
C!!!  Write output variables to real vector for transmission to host
C!!!
OUTREAL(I7+1) = FQG
OUTREAL(I7+2) = ETHMR
OUTREAL(I7+3) = ETHP
OUTREAL(I7+4) = ETHDB
OUTREAL(I7+5) = EPHMR
OUTREAL(I7+6) = EPHP
OUTREAL(I7+7) = EPHDB
            ELSE
              IF (LPATS) THEN
                CALL PATPAR(PSA,PATS,IPTS,LRCTS,I,IIV)
CCC
C!!!  Write output variables to real vector for transmission to host
C!!!
OUTREAL2(I2+1)=PSA(2)
OUTREAL2(I2+2)=PSA(3)
```

```
            END IF
            IF (LPATR) THEN
              CALL PATPAR(PRA,PATR,IPTR,LRCTR,I,IIV)
            END IF
CCC
C!!!  Write output variables to real vector for transmission to host
C!!!
OUTREAL(I8+1) = PRA(2)
OUTREAL(I8+2) = PRA(?)
OUTREAL(I8+3) = ETHMR
OUTREAL(I8+4) = ETHP
OUTREAL(I8+5) = ETHDB
OUTREAL(I8+6) = EPHMR
OUTREAL(I8+7) = EPHP
OUTREAL(I8+8) = EPHDB
          END IF
 2       CONTINUE

CCC-----------------------------------
C!!!  Send real vector(s) #1 to host
C!!!
            IF (LFQG) THEN
     call csend(902,OUTREAL,OUT_SIZE7,MY_HOST,81)
            ELSE
         IF (LPATS) call csend(901,OUTREAL2.OUT_SIZE2,MY_HOST,81)
   call csend(902,OUTREAL,OUT_SIZE8,MY_HOST,8i)
            END IF
CCC---------------------------------------
C!!!  Far Zone Total Field representations.
CCC---------------------------------------
.
.
.
CCC-----------------------------------
C!!!  Send FZ Total Field vector(s) to host
C!!!
.
.
.
C!!!
C!!! End Far Zone
CCC-------------
.
.
.
CCC-----------------------------------------------------------------

        ELSE IF (.NOT.LRCVR) THEN

CCC-----------------------------------------------------------------
CCC-----------------------------------
C!!!  Near zone E-field representation.
CCC-----------------------------------
.
```

```
.
.
CCC------------------------------------
C!!!  Near zone H-field representations.
CCC------------------------------------
.
.
.
C!!!--------------------------------
C!!!  Near zone power representation.
C!!!--------------------------------
.
.
.
CCC----------------------------------------------------------------------

        ELSE

CCC----------------------------------------------------------------------
CCC---------------------------------------------
C!!!  Antenna Coupling via the Reaction Principle
CCC---------------------------------------------
.
.
.
C!!!-------------------------------------------------------------------

        END IF

C!!!-------------------------------------------------------------------

   TIME=  mclock() - START_TIME
          call csend(201,TIME,INT_SIZE,MY_HOST,81)
c     write(*,*)'32out.f:  Returning from OUTWRT to main'
c     call forflush(6)
        RETURN
        END
```

C.5 GetCols

```
Program getcols
CCC  Scott Suhr - AFIT/EN
CCC  Program to extract a 2-column plotfile from
CCC    a NECBSC output file.
CCC
        character TEMP*120, TEMP1*11, TEMP2*11, ANSW, A*1
        character IN_FILE*70, OUT_FILE*70
integer IASC, COL_COUNT, COL1, COL2, SEG, LINE
real RA, RB
logical  DAT_SEG, IN_COL, SKIP
C------------------------------------------------------------------------
write(*,*)'Enter INPUT-FILE-NAME:'
read(*,FMT='(A)') IN_FILE
call Open_File(11,IN_FILE)
write(*,*)'>',IN_FILE,'<'

write(*,*)'Enter OUTPUT-FILE-NAME:'
read(*,FMT='(A)') OUT_FILE
call Open_File(13,OUT_FILE)
write(*,*)'>',OUT_FILE,'<'

call Open_File(10,'TEMP')

DAT_SEG = .false.
IN_COL = .false.

C------------------------------------------------------------------------
do 998 SEG = 1,3
SKIP = .true.
write(UNIT=6,FMT='(A,i1,A)')
     2  'Do you wish to retain from data section #', SEG, ' ? (Y/N)'
read(*,FMT='(A)') ANSW
if (((ANSW.EQ.'Y').OR.(ANSW.EQ.'y'))) SKIP=.FALSE.
if (.NOT.SKIP) then
   write(*,*)'Input Col you wish placed in output Col 1'
   read(*,FMT='(i)') COL1
   write(*,*)'Input Col you wish placed in output Col 2'
   read(*,FMT='(i)') COL2
end if
C------------------------------------------------------------------------
CCC    Infinite Loop until end of data section or EOF
C!!!
line = 0
 200 continue
  TEMP(1:121)=''
  TEMP1(1:15)=''
  TEMP2(1:15)=''
  Read(UNIT=11,FMT='(A)',IOSTAT=IERR,END=999) TEMP
          IF (IERR.GT.0) THEN
            WRITE(*,FMT='(A)') ' CAN NOT READ FILE "',NAMOPN,'"'
    goto 999
  end if
```

```
C!!!
C!!!  check for header info & skip
C!!!
if(TEMP(1:2).EQ.' *') goto 990
if(TEMP(1:2).EQ.'0 ') then
if (.NOT.DAT_SEG) then
  goto 990
else
  goto 200
end if
end if
if(TEMP(1:2).EQ.'1 ') goto 990
if(TEMP(1:2).EQ.'+ ') goto 990
if(TEMP(1:15).EQ.'               ') goto 990
CCC-----------------------
do 20 i=1,120
A = TEMP(i:i)
if(A.EQ.' ') then
  if((COL_COUNT.GE.COL1).AND.(COL_COUNT.GE.COL2)) goto 21
  if (IN_COL) then
   IN_COL = .false.
  end if
goto 20
end if
IASC = ichar(A)
C!!!  goto 990 indicates non-numeric data in current line:
if ((IASC.LT.48).OR.(IASC.GT.57)) then
  if((IASC.NE.43).AND.(IASC.NE.45).AND.
     2              (IASC.NE.46).AND.(IASC.NE.69)) goto 990
end if
C!!! DAT_SEG indicates currently in data Segment:
DAT_SEG = .true.

C!!! SKIP indicates don't want current section so Read new line:
if (SKIP) goto 200

C!!! IN_COL indicates traversing numeric data column
if (.NOT.IN_COL) then
  i2 = 1
  IN_COL = .true.
  COL_COUNT = COL_COUNT + 1
end if

if (COL_COUNT.EQ.COL1) then
  TEMP1(i2:i2) = A
else if (COL_COUNT.EQ.COL2) then
  TEMP2(i2:i2) = A
end if
i2 = i2 + 1

 20  continue
 21  continue
 write(UNIT=10,FMT='(3A)')TEMP1,'   ', TEMP2
c   write(*,*)TEMP1,'   ', TEMP2
```

```
 LINE = LINE + 1
 IN_COL = .false.
 COL_COUNT = 0
 goto 200
C!!!  Loop Back to read new line
CCC---------------------------

C!!!
C!!!  goto 990 indicates non-numeric data in current line:
 990 continue
IN_COL = .false.
C!!!  if were in data segment, now not
if (DAT_SEG) then
  write(*,*)line,' lines of data written from section #', SEG
  LINE = 0
  DAT_SEG = .false.
  goto 998
end if
C!!!  Go get New Line
    goto 200

C!!!   Goto 998 indicates loop back at start of new non-data segment
 998 continue
C!!!-------------------------------------------------

 999 continue
rewind(10)
write(*,*)'Data as output to file ',OUT_FILE(1:10),':'
step = 361/150+.5
rnext=1
do 500 i=1,10000
  read(UNIT=10,FMT='(2(PE11.4,3X))',END=1000)RA,RB
if(i.gt.rnext) then
  write(UNIT=6,FMT='(f15.7,A,f15.7)')RA,' ',RB
  write(UNIT=13,FMT='(f15.7,A,f15.7)')RA,' ',RB
  rnext=rnext+step
end if
 500 continue

1000 continue
write(UNIT=13,FMT='(f15.7,A,f15.7)')RA,'    ',RB
write(UNIT=6,FMT='(f15.7,A,f15.7)')RA,' ',RB
close(10)
close(11)
close(13)
write(*,*)'Exit GetCols',i,' lines output
end
C------------------------------------------------------------------
C------------------------------------------------------------------
Subroutine Open_file(IU,NAMOPN)
CHARACTER FRMOPN*11,NAMOPN*70,STTOPN*7

  FRMOPN(1:11)='FORMATTED'
  if (IU.EQ.11) then
```

```
        STTOPN(1:7)='OLD'
      else
        STTOPN(1:7)='UNKNOWN'
      end if
              OPEN(UNIT=IU,FILE=NAMOPN,FORM=FRMOPN,
         1         STATUS=STTOPN,IOSTAT=IERR)
      if(IERR.NE.0) write(*,*)'Error:',IOSTAT,
         1                          ' Opening TEMP files'
            RETURN
            END
C------------------------------------------------------------------
```

Appendix D. *Complete Timing Results*

Redundant output information is removed for conciseness (An example of the complete output from Mod4.0 is given in Table A.3. Results are from a single run , not averages(except Mod0.5 times); times may include variances due to UNIX overhead or other tasks running simultaneously on the host computer. Below is a sample of the results for different versions, with the calculated speedups and efficiencies.

Example 19 on iPSC/860 Speedup (SU) & Efficiency (EFF)				
			SU	Eff
Mod#	Nodes	Time (ms)	(vs. Mod0.5)	
4.0	1	43016	1.07	1.07
4.0	2	28064	1.64	.82
4.0	4	21036	2.19	.55
4.0	8	18359	2.51	.31
3.5	1	44348	1.04	1.04
3.5	2	28711	1.60	.80
3.5	4	22221	2.08	.52
3.5	8	18947	2.43	.30
1.6	1	39076	1.18	1.18
1.6	2	33000	1.40	.70
1.6	4	36217	1.27	.32
1.6	8	45104	1.02	.13
0.5	1	46132	1.00	1.00

D.1 *Mod0.5*

D.1.1 *i860: Mod0.5, Example 1c*

```
 NECBSC serial code on one node iPSC/2 or /860
  Mod 0.5 (13 Apr 91) - Serial code + timing
 Enter a filename for input (70 characters max)
ex1c.inp
 NODE: INPUT FILE NAME = ex1c.inp
 WRITE PRINTED OUTPUT TO DISK (T OR F)?
 T
     Host CPU time required for load:         440 ms
        Total Host CPU time required:         450
          Elapsed Node time required:       22870
 Approx. total elapsed time required:       23310
  (node elapsed + host cpu for load)
-- run #2:
     Host CPU time required for load:         430 ms
        Total Host CPU time required:         440
          Elapsed Node time required:       22933
 Approx. total elapsed time required:       23363
-- run #3:
     Host CPU time required for load:         490 ms
        Total Host CPU time required:         500
          Elapsed Node time required:       22899
 Approx. total elapsed time required:       23389
-- run #4:
    Host CPU time required for load:         420 ms
        Total Host CPU time required:         430
          Elapsed Node time required:       22905
 Approx. total elapsed time required:       23325
-------------------------------------------------------
mbvsrm>runbsc
 NECBSC serial code on one node iPSC/2 or /860
  Mod 0.5 (13 Apr 91) - Serial code + timing
 Enter a filename for input (70 characters max)
input.inp
 NODE: INPUT FILE NAME = input.inp

 WRITE PRINTED OUTPUT TO DISK (T OR F)?
 T
     Host CPU time required for load:         410 ms
        Total Host CPU time required:         420
          Elapsed Node time required:        5458
 Approx. total elapsed time required:        5868
  (node elapsed + host cpu startup)
```

D.1.2 i860: Mod0.5, Other Examples

```
NECBSC serial code on one node iPSC/860
 Mod 0.5 (13 Apr 91) - Serial code + timing
Enter a filename for input (70 characters max)
ex1a.inp
NODE: INPUT FILE NAME = ex1a.inp
    Host CPU time required for load:        470 ms
       Total Host CPU time required:        480
         Elapsed Node time required:       8548
Approx. total elapsed time required:       9018
-------------------------------------------------------
NODE: INPUT FILE NAME = ex1b.inp
    Host CPU time required for load:        420 ms
       Total Host CPU time required:        440
         Elapsed Node time required:      21202
Approx. total elapsed time required:      21622

-------------------------------------------------------
NODE: INPUT FILE NAME = ex6.inp
    Host CPU time required for load:        450 ms
       Total Host CPU time required:        450
         Elapsed Node time required:      43881
Approx. total elapsed time required:      44331

-------------------------------------------------------
NODE: INPUT FILE NAME = ex11a.inp
    Host CPU time required for load:        440 ms
       Total Host CPU time required:        450
         Elapsed Node time required:       9338
Approx. total elapsed time required:       9778
-------------------------------------------------------
NODE: INPUT FILE NAME = ex19.inp
    Host CPU time required for load:        430 ms
       Total Host CPU time required:        430
         Elapsed Node time required:      45702
Approx. total elapsed time required:      46132
```

D.1.3 386: Mod0.5, Example 1c

```
 NECBSC serial code on one node iPSC/2 or /860
  Mod 0.5 (13 Apr 91) - Serial code + timing
 Enter a filename for input (70 characters max)
ex1c.inp
 NODE: INPUT FILE NAME = ex1c.inp
 WRITE PRINTED OUTPUT TO DISK (T OR F)?
 T
     Host CPU time required for load:         500 ms
        Total Host CPU time required:         510
          Elapsed Node time required:       32590
 Approx. total elapsed time required:       33090
  (node elapsed + host cpu for load)
---- Run #2
     Host CPU time required for load:         520 ms
        Total Host CPU time required:         530
          Elapsed Node time required:       33000
 Approx. total elapsed time required:       33520
---- Run #3

     Host CPU time required for load:         520 ms
        Total Host CPU time required:         540
          Elapsed Node time required:       33101
 Approx. total elapsed time required:       33621
---- Run #4
     Host CPU time required for load:         540 ms
        Total Host CPU time required:         550
          Elapsed Node time required:       32983
 Approx. total elapsed time required:       33523
-------------------------------------------------------
-------------------------------------------------------
c386 9:runbsc
 NECBSC serial code on one node iPSC/2 or /860
  Mod 0.5 (13 Apr 91) - Serial code + timing
 Enter a filename for input (70 characters max)
input.inp
 NODE: INPUT FILE NAME = input.inp
 WRITE PRINTED OUTPUT TO DISK (T OR F)?
 T
     Host CPU time required for load:         520 ms
        Total Host CPU time required:         530
          Elapsed Node time required:        6575
 Approx. total elapsed time required:        7095
  (node elapsed + host cpu for load)
-------------------------------------------------------
```

D.1.4 386: Mod0.5, Other Examples

```
 NECBSC serial code on one node iPSC/2
  Mod 0.5 (13 Apr 91) - Serial code + timing
 Enter a filename for input (70 characters max)
ex1a.inp
 NODE: INPUT FILE NAME = ex1a.inp
     Host CPU time required for load:          540 ms
        Total Host CPU time required:          550
          Elapsed Node time required:        14230
 Approx. total elapsed time required:        14770
-------------------------------------------------------
 NODE: INPUT FILE NAME = ex1b.inp
     Host CPU time required for load:          480 ms
        Total Host CPU time required:          490
          Elapsed Node time required:        30582
 Approx. total elapsed time required:        31062
-------------------------------------------------------
 NODE: INPUT FILE NAME = ex6.inp
     Host CPU time required for load:          500 ms
        Total Host CPU time required:          520
          Elapsed Node time required:        90069
 Approx. total elapsed time required:        90569
-------------------------------------------------------
 NODE: INPUT FILE NAME = ex11a.inp
     Host CPU time required for load:          500 ms
        Total Host CPU time required:          510
          Elapsed Node time required:        11555
 Approx. total elapsed time required:        12055
-------------------------------------------------------
 NODE: INPUT FILE NAME = ex19.inp
     Host CPU time required for load:          520 ms
        Total Host CPU time required:          530
          Elapsed Node time required:        85411
 Approx. total elapsed time required:        85931
-------------------------------------------------------
```

D.2 Mod1.6

D.2.1 i860: Mod1.6, Example 1c

```
 NECBSC modified for iPSC/2 & /860 by Scott Suhr
     node: mod1.6              host: mod1.6
 (header write by node 0)     (w/merge sort
                            & multiple time blocks)
 Number of nodes attached:           8
 INPUT_FILE_NAME= "ex1c.inp"
 Elapsed:  Node     Total Time (msec)
             0         32854
             1         30827
             2         30685
             3         30856
             4         30776
             5         30890
             6         30866
             7         30867
 Host CPU time required for startup:    820
   Host CPU time required for merge:  11430
       Total Host CPU time required:  16080
Approx. total elapsed time required:  45104
      (node 0 + Startup + Merge)
--------------------------------------------------------
 Number of nodes attached:           4
 Elapsed:  Node     Total Time (msec)
             0         25207
             1         23943
             2         24047
             3         24070
 Host CPU time required for startup:    810
   Host CPU time required for merge:  10200
       Total Host CPU time required:  14480
Approx. total elapsed time required:  36217
--------------------------------------------------------
 Number of nodes attached:           2
 Elapsed:  Node     Total Time (msec)
             0         22560
             1         20738
 Host CPU time required for startup:    790
   Host CPU time required for merge:   9650
       Total Host CPU time required:  13610
Approx. total elapsed time required:  33000
--------------------------------------------------------
 Number of nodes attached:           1
 Elapsed:  Node     Total Time (msec)
             0         28846
 Host CPU time required for startup:    830
   Host CPU time required for merge:   9400
       Total Host CPU time required:  15720
Approx. total elapsed time required:  39076
--------------------------------------------------------
```

D.2.2 i860: Mod1.6, Example 6

```
 NECBSC modified for iPSC/2 & /860 by Scott Suhr
     node: mod1.6              host: mod1.6
 (header write by node 0)      (w/merge sort
                             & multiple time blocks)
 Number of nodes attached:           8
 INPUT_FILE_NAME= "ex6.inp"
 Elapsed:  Node     Total Time (msec)
             0         26873
             1         25133
             2         24928
             3         25133
             4         25143
             5         25294
             6         25240
             7         25311
 Host CPU time required for startup:    800
   Host CPU time required for merge:  12820
       Total Host CPU time required:  17460
Approx. total elapsed time required:  40493
---------------------------------------------------
 Number of nodes attached:           4
 Elapsed:  Node     Total Time (msec)
             0         24697
             1         23535
             2         23368
             3         23629
 Host CPU time required for startup:    880
   Host CPU time required for merge:  11570
       Total Host CPU time required:  16640
Approx. total elapsed time required:  37147
---------------------------------------------------
 Number of nodes attached:           2
 Elapsed:  Node     Total Time (msec)
             0         29257
             1         27118
 Host CPU time required for startup:    820
   Host CPU time required for merge:  11000
       Total Host CPU time required:  18990
Approx. total elapsed time required:  41077
---------------------------------------------------
 Number of nodes attached:           1
 Elapsed:  Node     Total Time (msec)
             0         46296
 Host CPU time required for startup:    800
   Host CPU time required for merge:  10780
       Total Host CPU time required:  25520
Approx. total elapsed time required:  57876
---------------------------------------------------
```

D.2.3 i860: Mod1.6, Example 19

```
 NECBSC modified for iPSC/2 & /860 by Scott Suhr
     node: mod1.6              host: mod1.6
 (header write by node 0)     (w/merge sort
                            & multiple time blocks)
 Number of nodes attached:          8
 INPUT_FILE_NAME= "ex19.inp"
 Elapsed:  Node     Total Time (msec)
             0         32572
             1         30620
             2         30093
             3         30711
             4         30250
             5         30842
             6         30813
             7         30874
 Host CPU time required for startup:    880
   Host CPU time required for merge:  13440
       Total Host CPU time required:  18320
Approx. total elapsed time required:  46892
------------------------------------------------------
 Number of nodes attached:          4
 Elapsed:  Node     Total Time (msec)
             0         27093
             1         25621
             2         25742
             3         25835
 Host CPU time required for startup:    840
   Host CPU time required for merge:  11530
       Total Host CPU time required:  16610
Approx. total elapsed time required:  39463
------------------------------------------------------
 Number of nodes attached:          2
 Elapsed:  Node     Total Time (msec)
             0         29338
             1         26227
 Host CPU time required for startup:    820
   Host CPU time required for merge:  10650
       Total Host CPU time required:  17740
Approx. total elapsed time required:  40808
------------------------------------------------------
 Number of nodes attached:          1
 Elapsed:  Node     Total Time (msec)
             0         48635
 Host CPU time required for startup:    850
   Host CPU time required for merge:  10200
       Total Host CPU time required:  26410
Approx. total elapsed time required:  59685
------------------------------------------------------
```

D.3 Mod3.5

D.3.1 i860: Mod3.5, Example 1c

```
  * NEC-BSC for iPSC/2 & iPSC/860 * 3.2i3.5,  9 Apr 91  *
 Input desired number of nodes:
 Number of nodes attached:           8
 Enter a filename for input (70 characters max)
 Input Filename = "ex1c.inp"
 Elapsed:  Node     Total Time (msec)
            0           7081
            1           7111
            2           7086
            3           7126
            4           7099
            5           7152
            6           7138
            7           7164
 Host CPU time required for startup:   1930
  Host CPU time required for output:   6530
       Total Host CPU time required:   8470
Approx. total elapsed time required:  15541
-----------------------------------------------------------
 Number of nodes attached:           4
 Elapsed:  Node     Total Time (msec)
            0           7149
            1           7140
            2           7172
            3           7189
 Host CPU time required for startup:   1660
  Host CPU time required for output:   6520
       Total Host CPU time required:   8190
Approx. total elapsed time required:  15329
-----------------------------------------------------------
 Number of nodes attached:           2
 Elapsed:  Node     Total Time (msec)
            0           8896
            1           8918
 Host CPU time required for startup:   1690
  Host CPU time required for output:   6530
       Total Host CPU time required:   8230
Approx. total elapsed time required:  17116
-----------------------------------------------------------
 Number of nodes attached:           1
 Elapsed:  Node     Total Time (msec)
            0          13231
 Host CPU time required for startup:   1630
  Host CPU time required for output:   6750
       Total Host CPU time required:   8400
Approx. total elapsed time required:  21611
-----------------------------------------------------------
```

D.3.2 i860: Mod3.5, Example 6

```
  * NEC-BSC for iPSC/2 & iPSC/860 * 3.2i3.5,  9 Apr 91  *
 Number of nodes attached:          8
 Input Filename = "ex6.inp"
 Elapsed:  Node     Total Time (msec)
             0         10204
             1         10238
             2         10226
             3         10250
             4         10262
             5         10301
             6         10287
             7         10278
 Host CPU time required for startup:   2860
  Host CPU time required for output:   6520
       Total Host CPU time required:   9400
Approx. total elapsed time required:  19584
      (node 0 + Startup + Merge)
------------------------------------------------------------
 Number of nodes attached:          4
 Elapsed:  Node     Total Time (msec)
             0         12116
             1         12127
             2         12178
             3         12127
 Host CPU time required for startup:   2660
  Host CPU time required for output:   6450
       Total Host CPU time required:   9110
Approx. total elapsed time required:  21226
------------------------------------------------------------
 Number of nodes attached:          2
 Elapsed:  Node     Total Time (msec)
             0         18466
             1         18412
 Host CPU time required for startup:   2470
  Host CPU time required for output:   6430
       Total Host CPU time required:   8900
Approx. total elapsed time required:  27366
------------------------------------------------------------
 Number of nodes attached:          1
 Elapsed:  Node     Total Time (msec)
             0         34035
 Host CPU time required for startup:   2600
  Host CPU time required for output:   6510
       Total Host CPU time required:   9120
Approx. total elapsed time required:  43145
```

D.3.3 i860: Mod3.5, Example 19

```
  * NEC-BSC for iPSC/2 & iPSC/860 * 3.2i3.5,  9 Apr 91  *
 Number of nodes attached:          8
 Input Filename = "ex19.inp"
 Elapsed:  Node     Total Time (msec)
             0          9967
             1          9892
             2          9902
             3          9961
             4          9967
             5          9956
             6         10001
             7          9995
 Host CPU time required for startup:   2280
  Host CPU time required for output:   6700
       Total Host CPU time required:   9000
Approx. total elapsed time required:  18947
      (node 0 + Startup + Merge)
----------------------------------------------------------
 Number of nodes attached:          4
 Elapsed:  Node     Total Time (msec)
             0         13411
             1         13371
             2         13445
             3         13395
 Host CPU time required for startup:   2170
  Host CPU time required for output:   6640
       Total Host CPU time required:   8820
Approx. total elapsed time required:  22221
----------------------------------------------------------
 Number of nodes attached:          2
 Elapsed:  Node     Total Time (msec)
             0         20031
             1         19977
 Host CPU time required for startup:   2040
  Host CPU time required for output:   6640
       Total Host CPU time required:   8680
Approx. total elapsed time required:  28711
----------------------------------------------------------
 Number of nodes attached:          1
 Elapsed:  Node     Total Time (msec)
             0         35538
 Host CPU time required for startup:   2150
  Host CPU time required for output:   6660
       Total Host CPU time required:   8810
Approx. total elapsed time required: .44348
```

D.4 Mod4.0

D.4.1 i860: Mod4.0, Example 1c

```
  * NEC-BSC for iPSC/2 & iPSC/860 * 3.2i4.0,  6 May 91  *
 Number of nodes attached:          8
 Input Filename = "ex1c.inp"
 Elapsed:  Node     Total Time (msec)
           0          6663
           1          6682
           2          6664
           3          6695
           4          6677
           5          6727
           6          6716
           7          6706
Output elapsed time (node 0, msec):   3592
 Host CPU time required for startup:   1570
 Host CPU time required for output:    6490
       Total Host CPU time required:   8090
Approx. total elapsed time required:  14723
----------------------------------------------------------
 Number of nodes attached:          4
 Elapsed:  Node     Total Time (msec)
           0          6605
           1          6618
           2          6606
           3          6632
Output elapsed time (node 0, msec):   3533
 Host CPU time required for startup:   1590
 Host CPU time required for output:    6450
       Total Host CPU time required.   8050
Approx. total elapsed time required:  14645
----------------------------------------------------------
 Number of nodes attached:          2
 Elapsed:  Node     Total Time (msec)
           0          8356
           1          8372
Output elapsed time (node 0, msec):   3552
 Host CPU time required for startup:   1620
 Host CPU time required for output:    6440
       Total Host CPU time required:   8070
Approx. total elapsed time required:  16416
----------------------------------------------------------
 Number of nodes attached:          1
 Elapsed:  Node     Total Time (msec)
           0         12365
Output elapsed time (node 0, msec):   3664
 Host CPU time required for startup:   1600
 Host CPU time required for output:    6510
       Total Host CPU time required:   8130
Approx. total elapsed time required:  20475
```

D.4.2 i860: Mod4.0, Example 6

```
  * NEC-BSC for iPSC/2 & iPSC/860 * 3.2i4.0,  6 May 91  *
 Number of nodes attached:          8
 Input Filename = "ex6.inp"
 Elapsed:  Node     Total Time (msec)
           0          9260
           1          9278
           2          9260
           3          9288
           4          9267
           5          9304
           6          9298
           7          9314
Output elapsed time (node 0, msec):   3666
 Host CPU time required for startup:   2640
 Host CPU time required for output:    6540
       Total Host CPU time required:   9200
Approx. total elapsed time required:  18440
--------------------------------------------------------
 Number of nodes attached:          4
 Elapsed:  Node     Total Time (msec)
           0         11514
           1         11518
           2         11560
           3         11514
Output elapsed time (node 0, msec):   3632
 Host CPU time required for startup:   2590
 Host CPU time required for output:    6440
       Total Host CPU time required:   9040
Approx. total elapsed time required:  20544
--------------------------------------------------------
 Number of nodes attached:          2
 Elapsed:  Node     Total Time (msec)
           0         17798
           1         17749
Output elapsed time (node 0, msec):   3636

 Host CPU time required for startup:   2630
 Host CPU time required for output:    6430
       Total Host CPU time required:   9060
Approx. total elapsed time required:  26858
--------------------------------------------------------
 Number of nodes attached:          1
 Elapsed:  Node     Total Time (msec)
           0         31274
Output elapsed time (node 0, msec):   3751
 Host CPU time required for startup:   2580
 Host CPU time required for output:    6490
       Total Host CPU time required:   9070
Approx. total elapsed time required:  40344
```

D.4.3 i860: Mod4.0, Example 19

```
  * NEC-BSC for iPSC/2 & iPSC/860 * 3.2i4.0,  6 May 91  *
 Number of nodes attached:          8
 Input Filename = "ex19.inp"
 Elapsed:  Node     Total Time (msec)
           0          9549
           1          9457
           2          9509
           3          9456
           4          9515
           5          9521
           6          9561
           7          9526
Output elapsed time (node 0, msec):   3698
 Host CPU time required for startup:   2140
 Host CPU time required for output:    6670
       Total Host CPU time required:   8820
Approx. total elapsed time required:  18359
--------------------------------------------------------
 Number of nodes attached:          4
 Elapsed:  Node     Total Time (msec)
           0         12256
           1         12195
           2         12242
           3         12200
Output elapsed time (node 0, msec):   3659
 Host CPU time required for startup:   2120
 Host CPU time required for output:    6660
       Total Host CPU time required:   8800
Approx. total elapsed time required:  21036
--------------------------------------------------------
 Number of nodes attached:          2
 Elapsed:  Node     Total Time (msec)
           0         19314
           1         19265
Output elapsed time (node 0, msec):   3668
 Host CPU time required for startup:   2140
 Host CPU time required for output:    6610
       Total Host CPU time required:   8760
Approx. total elapsed time required:  28064
--------------------------------------------------------
 Number of nodes attached:          1
 Elapsed:  Node     Total Time (msec)
           0         34266
Output elapsed time (node 0, msec):   3778

 Host CPU time required for startup:   2100
 Host CPU time required for output:    6650
       Total Host CPU time required:   8760
Approx. total elapsed time required:  43016
```

D.4.4 386(Weitex): Mod4.0, Example 1c

```
  * NEC-BSC for iPSC/2 & iPSC/860 * 3.2i4.0,  2 May 91  *
 Number of nodes attached:          8
 Input Filename = "ex1c.inp"
 Elapsed:  Node     Total Time (msec)
           0         11493
           1         11515
           2         11346
           3         11166
           4         11501
           5         11353
           6         11079
           7             0
Output elapsed time (node 0, msec):   5811
 Host CPU time required for startup:   1910
 Host CPU time required for output:    6670
       Total Host CPU time required:   8600
Approx. total elapsed time required:  16804
-------------------------------------------------------
 Number of nodes attached:          4
 Elapsed:  Node     Total Time (msec)
           0          9444
           1          9468
           2          9449
           3             0
Output elapsed time (node 0, msec):   3616
 Host CPU time required for startup:   1830
 Host CPU time required for output:    6480
       Total Host CPU time required:   8330
Approx. total elapsed time required:  16534
-------------------------------------------------------
 Number of nodes attached:          2
 Elapsed:  Node     Total Time (msec)
           0         12318
           1         12334
Output elapsed time (node 0, msec):   3611
 Host CPU time required for startup:   1790
 Host CPU time required for output:    6420
       Total Host CPU time required:   8210
Approx. total elapsed time required:  16434
-------------------------------------------------------
 Number of nodes attached:          1
 Elapsed:  Node     Total Time (msec)
           0         19928
Output elapsed time (node 0, msec):   3780
 Host CPU time required for startup:   1770
 Host CPU time required for output:    6480
       Total Host CPU time required:   8250
Approx. total elapsed time required:  16474
```

D.4.5 386(Weitex): Mod4.0, Example 6

```
  * NEC-BSC for iPSC/2 & iPSC/860 * 3.2i4.0,  2 May 91  *
 Number of nodes attached:          8
 Input Filename = "ex6.inp"
 Elapsed:  Node     Total Time (msec)
           0         15332
           1         15459
           2         15299
           3         15445
           4         15311
           5         15457
           6         15484
           7         15421
Output elapsed time (node 0, msec):   3696
 Host CPU time required for startup:   2680
 Host CPU time required for output:    6530
       Total Host CPU time required:   9230
Approx. total elapsed time required:  17434
--------------------------------------------------------
 Number of nodes attached:          4
 Elapsed:  Node     Total Time (msec)
           0         23035
           1         23033
           2         23074
           3         22853
Output elapsed time (node 0, msec):   3661
 Host CPU time required for startup:   2820
 Host CPU time required for output:    6450
       Total Host CPU time required:   9290
Approx. total elapsed time required:  17494
--------------------------------------------------------
 Number of nodes attached:          2
 Elapsed:  Node     Total Time (msec)
           0         40014
           1         40028
Output elapsed time (node 0, msec):   3700

 Host CPU time required for startup:   2770
 Host CPU time required for output:    6460
       Total Host CPU time required:   9230
Approx. total elapsed time required:  17454
--------------------------------------------------------
 Number of nodes attached:          1
 Elapsed:  Node     Total Time (msec)
           0         75064
Output elapsed time (node 0, msec):   3881
 Host CPU time required for startup:   2750
 Host CPU time required for output:    6530
       Total Host CPU time required:   9300
Approx. total elapsed time required:  17504
```

D.4.6 386(Weitex): Mod4.0, Example 19

```
  * NEC-BSC for iPSC/2 & iPSC/860 * 3.2i4.0,  2 May 91  *
 Number of nodes attached:           8
 Input Filename = "ex19.inp"
 Elapsed:  Node     Total Time (msec)
           0         14820
           1         14546
           2         14699
           3         14571
           4         14721
           5         14587
           6         14584
           7         14564
Output elapsed time (node 0, msec):   3722
 Host CPU time required for startup:   2360
 Host CPU time required for output:    6680
       Total Host CPU time required:   9050
Approx. total elapsed time required:  17264
--------------------------------------------------------
 Number of nodes attached:           4
 Elapsed:  Node     Total Time (msec)
           0         23486
           1         23478
           2         23441
           3         22863
Output elapsed time (node 0, msec):   4251
 Host CPU time required for startup:   2320
 Host CPU time required for output:    6710
       Total Host CPU time required:   9030
Approx. total elapsed time required:  17254
--------------------------------------------------------
 Number of nodes attached:           2
 Elapsed:  Node     Total Time (msec)
           0         38233
           1         38187
Output elapsed time (node 0, msec):   3781

 Host CPU time required for startup:   2330
 Host CPU time required for output:    6650
       Total Host CPU time required:   8990
Approx. total elapsed time required:  17204
 Input desired Cube-Type (8rx, 4sx, etc):
--------------------------------------------------------
 Number of nodes attached:           1
 Elapsed:  Node     Total Time (msec)
           0         71643
Output elapsed time (node 0, msec):   3929
 Host CPU time required for startup:   2220
 Host CPU time required for output:    6700
       Total Host CPU time required:   8930
Approx. total elapsed time required:  17144
```

D.4.7 386(80387): Mod4.0, Example 1c

```
  * NEC-BSC for iPSC/2 & iPSC/860 * 3.2i4.0,  6 May 91  *
 Number of nodes attached:          8
 Input Filename = "ex1c.inp"
 Elapsed:  Node     Total Time (msec)
           0         10622
           1         10501
           2         10645
           3         10500
           4         10666
           5         10509
           6         10511
           7             0
Output elapsed time (node 0, msec):   3707
 Host CPU time required for startup:   1760
 Host CPU time required for output:    6540
       Total Host CPU time required:   8320
Approx. total elapsed time required:  18922
--------------------------------------------------------
 Number of nodes attached:          4
 Elapsed:  Node     Total Time (msec)
           0         13743
           1         13748
           2         13737
           3         13668
Output elapsed time (node 0, msec):   3605
 Host CPU time required for startup:   1770
 Host CPU time required for output:    6410
       Total Host CPU time required:   8190
Approx. total elapsed time required:  21923
--------------------------------------------------------
 Number of nodes attached:          2
 Elapsed:  Node     Total Time (msec)
           0         22185
           1         22204
Output elapsed time (node 0, msec):   3733
 Host CPU time required for startup:   1710
 Host CPU time required for output:    6440
       Total Host CPU time required:   8150
Approx. total elapsed time required:  30335
--------------------------------------------------------
 Number of nodes attached:          1
 Elapsed:  Node     Total Time (msec)
           0         40343
Output elapsed time (node 0, msec):   4034
 Host CPU time required for startup:   1710
 Host CPU time required for output:    6500
       Total Host CPU time required:   8220
Approx. total elapsed time required:  48553
```

D.4.8 386(80387): Mod4.0, Example 6

```
  * NEC-BSC for iPSC/2 & iPSC/860 * 3.2i4.0,  6 May 91  *
 Number of nodes attached:          8
 Input Filename = "ex6.inp"
 Elapsed:  Node     Total Time (msec)
           0          22124
           1          21964
           2          22108
           3          21966
           4          22097
           5          22029
           6          22031
           7          21961
Output elapsed time (node 0, msec):   3766
 Host CPU time required for startup:   2830
 Host CPU time required for output:    6560
       Total Host CPU time required:   9400
Approx. total elapsed time required:  31514
----------------------------------------------------------
 Number of nodes attached:          4
 Elapsed:  Node     Total Time (msec)
           0          36018
           1          35994
           2          36071
           3          35903
Output elapsed time (node 0, msec):   3749
 Host CPU time required for startup:   2660
 Host CPU time required for output:    6520
       Total Host CPU time required:   9210
Approx. total elapsed time required:  45198
----------------------------------------------------------
 Number of nodes attached:          2
 Elapsed:  Node     Total Time (msec)
           0          66138
           1          66093
Output elapsed time (node 0, msec):   4505
 Host CPU time required for startup:   2650
 Host CPU time required for output:    6540
       Total Host CPU time required:   9210
Approx. total elapsed time required:  75328
----------------------------------------------------------
 Number of nodes attached:          1
 Elapsed:  Node     Total Time (msec)
           0         125757
Output elapsed time (node 0, msec):   4216
 Host CPU time required for startup:   2640
 Host CPU time required for output:    6500
       Total Host CPU time required:   9150
Approx. total elapsed time required: 134837
```

D.4.9 386(80387): Mod4.0, Example 19

```
 * NEC-BSC for iPSC/2 & iPSC/860 * 3.2i4.0,  6 May 91  *
 Number of nodes attached:          8
 Input Filename = "ex19.inp"
 Elapsed:  Node     Total Time (msec)
           0         21301
           1         21296
           2         21205
           3         21236
           4         21243
           5         21195
           6         21215
           7         21210
Output elapsed time (node 0, msec):   3782
 Host CPU time required for startup:   2260
 Host CPU time required for output:    6670
      Total Host CPU time required:   8940
Approx. total elapsed time required:  30231
-----------------------------------------------------------
 Number of nodes attached:          4
 Elapsed:  Node     Total Time (msec)
           0         35037
           1         35013
           2         35059
           3         34932
Output elapsed time (node 0, msec):   3800
 Host CPU time required for startup:   2250
 Host CPU time required for output:    6610
      Total Host CPU time required:   8870
Approx. total elapsed time required:  43397
-----------------------------------------------------------
  Number of nodes attached:          2
 Elapsed:  Node     Total Time (msec)
           0         63951
           1         63971
Output elapsed time (node 0, msec):   3914
 Host CPU time required for startup:   2220
 Host CPU time required for output:    6590
      Total Host CPU time required:   8810
Approx. total elapsed time required:  72761
-----------------------------------------------------------
 Number of nodes attached:          1
 Elapsed:  Node     Total Time (msec)
           0        123162
Output elapsed time (node 0, msec):   4160
 Host CPU time required for startup:   2200
 Host CPU time required for output:    6670
      Total Host CPU time required:   8880
Approx. total elapsed time required: 132032
```

Bibliography

1. *iPSC/2 and iPSC/860 Programmers Reference Manual*. Beaverton OR: Intel Scientific Computers, June 1990.

2. *iPSC/2 and iPSC/860 Release 3.2 Software Product Release Notes*. Beaverton OR: Intel Scientific Computers, July 1990.

3. *iPSC/2 and iPSC/860 Users Guide*. Beaverton OR: Intel Scientific Computers, June 1990.

4. Agrawal, Rakesh and H. V. Jagadish. "Partitioning Techniques for Large-Grained Parallelism," *IEEE Transactions on Computers*, *37*:1627–1633 (December 1988).

5. Beard, Andrew and Gary B. Lamont, "AFIT/ENG Intel Hypercube Quick Reference Manual Version 1.1." Class handout - EENG 689, School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, 6 February 1990.

6. Chandy, K. Mani and Jayadev Misra. *Parallel Program Design, a Foundation*. New York NY: Addison-Wesley Publishing Company, 1988.

7. Corporation, Pacific-Sierra Research. *iPSC/2 and iPSC/860 FORGE^{TM} Users Guide (The FORGE^{TM} User's Guide)*. Beaverton OR: Intel Scientific Computers, April 1990.

8. Geoffrey C. Fox, et al. *Solving Problems on Concurrent Processors, Vol I*. Englewood Cliffs NJ: Prentice Hall, 1988.

9. Kumar, Vipin and V. Nageshwara Rao. "Load Balancing on the Hypercube Architecture." *The Proceedings of the Fourth Conference on Hypercubes, Concurrent Computers, and Applications*. 603–608. Los Altos CA: Golden Gate Enterprises, 1989.

10. Lionel M. Ni, Chung-Ta King and Phillip Prins. "Parallel Algorithm Design Considerations for Hypercube Multiprocessors." *The Proceedings of the 1987 International Conference on Parallel Processing*. 717–720. University Park PA: Penn State University Press, 1987.

11. Marhefka, R. H. and J. W. Silvestro. "Near Zone - Basic Scattering Code User's Manual with Space Station Applications." Preliminary Version, March 1989.

12. McDowell, Charles E and David P. Helmbold. "Debugging Concurrent Programs," *ACM Computing Surveys*, *4*:593–622 (December 1989).

13. Polychronopoulous, Constantine D. "Loop Coalescing: A Compiler Transformation for Parallel Machines." *The Proceedings of the 1987 International Conference on Parallel Processing*. 235–241. University Park PA: Penn State University Press, 1987.

14. R. Beard, et al. *Compendium of Parallel Programs for the Intel iPSC Computers*. Air Force Institute of Technology, Wright-Patterson AFB OH: Department of Electrical and Computer Engineering, 1990.

15. R. Marhefka, Ohio State University. Interview, ACES Symposium, Monterrey CA, 20 March 1991.

16. Work, P. and R. Moserl, "CSCE 656 Project 2: Testing the Mesh and Numeric Programs." Paper, Department of Electrical and Computer Engineering, Air Force Institute of Technology, Wright-Patterson AFB OH, May 1991.

Imagen Laser Printer (iml32)

Owner	ssuhr
Host	wb17
printer	iml32
Date	Sat Jun 1 02:02:43 1991
User	ssuhr
formlength	66

inputbin letter
copies 1
pageduplex off
language ultrascript
jobheader on

System Version TURBO UltraScript 5.0T IP/II, Serial #89:10:51
Page images processed: 107
Pages printed: 107

Paper size (width, height):
  2560, 3328
Document length:
  388872 bytes