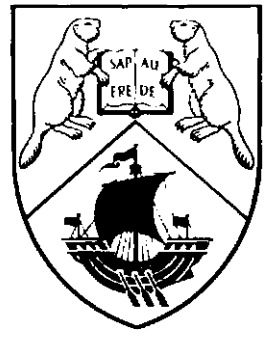
ADAPTATION OF A MONTE CARLO RADIATION TRANSPORT CODE TO SUPERCOMPUTERS

BY

TASSOS ANDREA TASSOU

TR86-032, JANUARY 1986

SCHOOL OF COMPUTER SCIENCE



UNIVERSITY OF NEW BRUNSWICK

FREDERICTON, N. B. CANADA E3B 5A3

ADAPTATION OF A MONTE CARLO RADIATION TRANSPORT CODE TO SUPERCOMPUTERS

by

Tassos Andrea Tassou

A THESIS SUBMITTED IN PARTIAL FULFILMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

Master of Science in Computer Science

in the School

of

Computer Science

This Thesis is accepted.

.....

Dean of Graduate Studies and Research

THE UNIVERSITY OF NEW BRUNSWICK

January, 1986

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my supervisors, Professors V.C Bhavsar, E. Hussein and U.G Gujar, for their valuable help and guidance throughout this project work.

I am grateful to the School of Computer Science for awarding me the Graduate Teaching Assistantship, and to Dr. V.C Bhavsar for partially supporting me during the months of September to December 1985, through his National Sciences and Engineering Research Council of Canada grant (No. A0089).

Free CPU-time was provided by the Supercomputing Services of the University of Calgary on the Cyber-205 computer. I wish to acknowledge the Supercomputing Services personnel, in particular Mr. Rod J. Wittig, Mr. Doug J. Baker, Ms. Pat L. Comer, and Ms. Kristen E. Mollin, for their help and support.

Finally, I wish to thank Professor D.M Luke for allowing me access to his office and his IBM personal computer, which made possible the communication with the Cyber-205 in Calgary.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION

1.1 INTRODUCTION

The Monte Carlo method is an experimental mathematical technique dealing with random numbers. In this method the solution is represented by a parameter randomly sampled from a hypothetical population designed to represent the physical nature of the problem. The required solution is then determined by statistical estimates of the parameter.

The Monte Carlo method has found extensive use in the fields of nuclear physics, quantum chemistry, numerical analysis, information theory, biology, statistical mechanics and others [10,9,21]. It provides a powerful tool for radiation transport analysis in nuclear engineering [8].

Although Monte Carlo computations are time consuming and expensive, they sometimes provide the only mean for solving problems with complicated three-dimensional geometries or for validating approximate computational methods. Solution of these problems with ordinary numerical methods become very complex in 3-Dimensional space. Additionally, the Monte Carlo method allows one to easily visualize the

physical aspects of the problem which otherwise are buried in complex equations.

In the Monte Carlo method, the accuracy of the results is limited by the computing time required to reduce statistical uncertainties. The solution of practical problems is constrained because their simulation requires large execution time on sequential computers.

With computer execution time as the major disadvantage of the Monte Carlo method, it is natural to consider its implementation on parallel computers. Parallel computers, otherwise called Supercomputers, like the Cray-1 and Cyber-205 vector computers and the HEP multiprocessor machine, offer high execution speeds compared to their scalar counterparts. This improvement in speed is due to the vector and parallel architecture of the supercomputers.

Supercomputers reach their peak performance if the computations are suitably converted to fit their architectural characteristics. Conventional Monte Carlo codes in radiation transport follow one particle at a time through a random walk. This involves tracking the particle through the problem geometry and randomly sampling from probability distributions representing the physics of the problem. However, the Monte Carlo method has some degree of intrinsic parallelism since the random walks are independent of each other [1]. This parallelism can be exploited for

the implementation of Monte Carlo computations on multiprocessor machines. On the other hand, this parallelism is not directly useful for implementation on vector computers. Vector processing demands that operations should be performed on contiguous vector elements. Monte Carlo codes, however, include plenty of conditional statements and very few DO-loops. This makes direct vectorization of these codes through existing vectorizing compilers very inefficient, and the improvement in speed from local recoding very poor. Effective vectorization can be achieved if new algorithms capable of exploiting the vector machines capabilities are developed [3,2,18,26].

Recent efforts have demonstrated that vectorization of Monte Carlo computations requires major changes in the global algorithm [3,2,18,26]. Although some work has been done on implementing Monte Carlo methods on vector supercomputers, little has been reported on implementing them on multiprocessor machines [16].

1.2 SCOPE OF THE PROJECT

The main motivation for this work is the desire to adapt a general purpose Monte Carlo Radiation Transport code such as the MORSE code [31], on supercomputers. As a first step towards this goal, it was decided to consider the implementation of a relatively less complex Monte Carlo code on the supercomputers with the following objectives:

1. Study the architectural and programming characteristics of existing supercomputers.
2. Develop skills and become familiar with implementation techniques of Monte Carlo codes on supercomputers.
3. Develop vector and parallel algorithms for the Monte Carlo code.
4. Study the behaviour of these algorithms

In the context of these objectives the contents of this report are summarized below:

In Chapter 2, commercially available supercomputers such as the Cyber-205 vector computer and the HEP multiprocesor are described. The general architectural characteristics and programming aspects of these computers are also reviewed and some programming examples are provided.

In Chapter 3, the Monte Carlo method as applied to radiation transport analysis is described. The different aspects involved in solving the Boltzmann transport equation are explained and a sequential algorithm is given. Some previous work on implementing Monte carlo methods on supercomputers is reviewed and some principles for converting scalar codes to parallel codes are stated.

In Chapter 4, the Center-of-Mass Monte Carlo program is described in detail. Some computational results of COM on

the IBM 3081 computer and the Cyber-205 scalar processor are presented. The procedure followed in designing VCOM (the vectorized version of COM) is discussed together with the computational results of VCOM on the Cyber-205 vector machine.

In Chapter 5, the performance of COM on multiprocessors is predicted for different computational schemes, based on available theoretical formulae. The performance is also estimated through simulation on the IBM 3081. A parallel algorithm is proposed for the implementation of COM on multiprocessors.

Finally, the results of this project are summarized in Chapter 6. Some concluding remarks and recommendations for future work are given.

The Job Control Language for the NOS Software system [30] and sign on procedures for the Cyber-205 computer are given in the Appendix, along with listings and sample results for the COM and VCOM FORTRAN codes.

Chapter 2

SUPERCOMPUTERS

2.1 INTRODUCTION

Based on current technology, a computer is considered a supercomputer if it can perform over 100 MFLOPS (million floating point operations per second) with word length of 64 bits and a main memory capacity of at least one million words [7].

Supercomputers can be classified into three architectural configurations. Vector computers, array processors, and multiprocessor systems. All three utilize parallel processing but in different forms. Vector computers perform overlapped computations on contiguous data elements (vectors). Array processors use multiple arithmetic logic units that can operate in parallel and are synchronized by a control unit. Multiprocessor systems consist of multiple processors which share software/hardware resources and operate in parallel in an asynchronous fashion. Both vector and array supercomputers are designed mainly for processing of large arrays of data while multiprocessors have the most flexible computer architecture in exploiting arbitrarily structured parallelism.

At present, most commercially available supercomputers are vector machines like Cray-1 and Cyber-205 with few multiprocessors like HEP. In the ensuing sections, the architecture and programming aspects of the Cyber-205 and HEP supercomputers are reviewed.

2.2 THE CYBER-205 VECTOR COMPUTER

The Cyber-205 represents more than 20 years of evolution in scientific computing by the Control Data Corproration (CDC). As a commercial vector computer it became available in 1981. With four vector-pipes, it can perform operations at 200 and 400 MFLOPS rates when operating on 64 and 32-bit data respectively [14].

2.2.1 Architecture

The Cyber-205 computer is a large scale, high speed, computing system. The basic components of the system are [27]:

1. Cental Processing Unit - CPU.
2. Central memory - 1 million, 64-bit words.
3. Eight I/O ports.
4. Maintenance Control unit - MCU.

A basic block diagram for the Cyber-205 is shown in Figure 2.1.

The Central Processing Unit consists of a vector processor, a scalar processor and the Input/Output ports. Both the scalar and vector processors are heavily pipelined for instruction fetching and instruction execution. Instruction execution control resides in the Scalar Unit. It receives and decodes all instructions from memory, directly executes scalar instructions and dispatches vector and string instructions to the four vector pipes and to the String Unit for execution.

With independent vector and scalar instruction controls, the scalar unit can execute scalar instructions in parallel with the execution of most vector instructions.

The Scalar Arithmetic Unit contains five functional pipes for add/subtract, multiply, logical, shift, and divide/square root operations over 32 or 64-bit scalars. The peak speed of the scalar processor is 50 MFLOPS. A basic block diagram of the vector-floating point pipeline is shown in Figure 2.2.

The vector processor can have one, two, or four floating point arithmetic pipes. Each vector pipeline is directly connected to the main memory without using vector registers.

The Stream Unit, manages the data streams between central memory and the vector pipelines. A vector arithmetic pipeline can perform add/subtract, multiply, divide, square

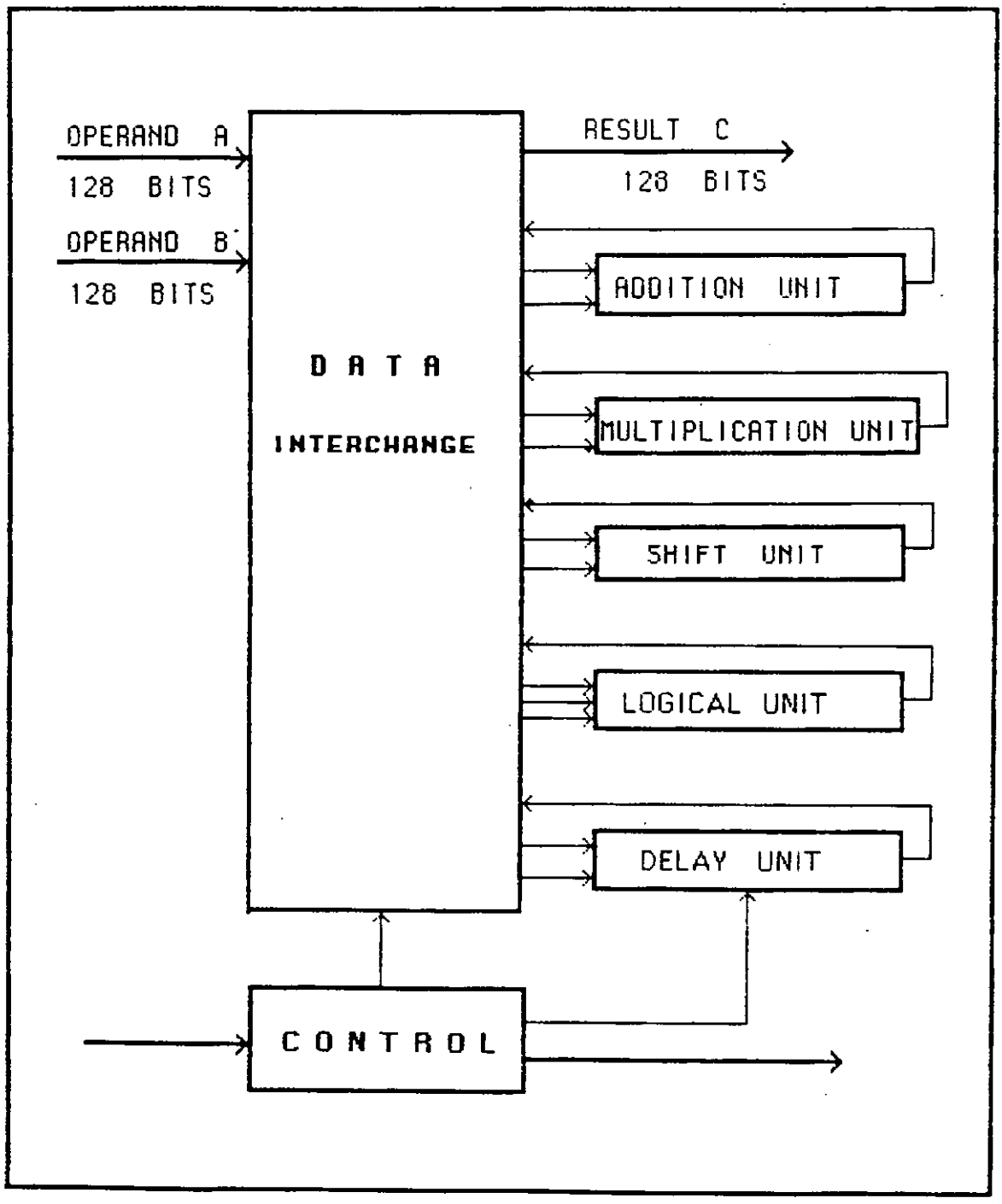


Fig. 2.2: Basic block diagram of Vector-Floating Point pipeline

root, logical and shift over 32 or 64-bit vector operands. With the exception of divide and square root operations, these pipelines each accept a new pair of operands every minor cycle (20 ns) for 64-bit floating point operations; for 32-bit operations the rate is doubled.

The String Unit performs logical operations on strings of data to allow bit operations on bit boundaries.

For the Cyber-205, a vector may consist of 65,635 (64K) consecutive memory words. The vector start-up times are large because of the memory to memory operations. Pipelining produces one result per clock period of 20 ns. The result from any of the units can be routed directly to the input of other units without the use of intermediate registers.

With four vector pipes, the Cyber-205 can produce 200 MFLOPS for 64-bit results and 400 MFLOPS for 32-bit results in vector add/subtract or in vector multiply operations. It can be also used to perform Linked triad operations of the type

vector*(scalar+vector) or vector+(scalar*vector)

with effective rates of 800 and 400 MFLOPS for 32-bit and 64-bit results, respectively.

The Cyber-205 can have 8 to 16 I/O channels, each 16 bits wide. The I/O channels provide the control and data paths

for communication between the central memory and the external devices. Each I/O channel has a maximum transfer rate of 200 megabits.

The Central memory is a random access memory using 16K MOS circuits (series 600). The memory words are 78 bits, providing a 64-bit data word with 7 bits single error correction and double error detection (SECDED) for each 32-bit half-word. The basic memory size is 1 million words with expansions to 2,4,8,12 or 16 million words. It is a virtual memory system with advanced memory management features such as Key and Lock for memory protection and separation, hardware mapping from virtual to physical address and user program data sharing capability. The main memory bandwidth is 400 MWords/sec and this is needed to support the memory-to-memory pipeline operations.

A powerful feature of the Cyber-205 is the bit vector cabability. For decision making operations in vector coding, hardware and addressing are provided for bit vectors, with single bits representing "true" or "false" conditions. Microcoded vector macroinstructions that dynamically reconfigure the vector pipes provide direct vector implementation of dot products, summation of vector elements and many other functions. Additionally, a hardware instruction is provided for vector square root operation.

Where vectorizing compilers fail, it becomes the programmer's task to convert the scalar into vector code using vector instructions or calls to vector subprograms. The remainder of this section discusses some vector operations available on the Cyber-205 for vector data handling and decision making. These are the compress/expand, merge, gather/scatter, and selective store operations. A complete set of instructions and how they are used can be found in the Fortran-200 reference manual [28].

● The compress operation removes unwanted elements from a vector by reducing its length. The elements to be removed correspond to the 0's in a bit control vector. For example, compressing a vector A to a vector R according to the control vector CV results in:

```
CV: 1 1 0 0 1      - length 5
A : 5 4 3 2 1      - length 5
R : 5 4 1          - length 3
```

● The merge operation merges two vectors according to a bit control vector. For example, merging a vector A with a vector B to produce a vector R according to bit control vector CV results in:

```
A: 5 4 3 2 1
B: 6 7 8 9 10
CV: 1 1 0 0 1 1 0 0      - length 8
```

R: 5 4 6 7 3 2 8 9 - length 8

- The expand operation creates a longer vector having zeros placed in elements corresponding to zeros in a bit control vector. For example, expanding a vector C to a vector A according to CV produces:

C: 2 5
CV: 0 1 0 0 1
A: 0 2 0 0 5

- The gather operation is used to form a contiguous vector from random data. The elements are collected according to an index list. For example:

index list I: 1 5 3 7
vector A : 10 11 12 13 14 15 16 17 18
result vector R: 10 14 12 16

- The scatter operation stores the elements of a vector into random locations in another vector according to an index list. For example:

vector A: 10 11 12 13 14
index list I: 1 5 3 7 9
old vector R: 0 0 0 0 0 0 0 0 0
new vector R: 10 0 12 0 11 0 13 0 14

● The selective store operation stores elements in a vector according to a bit control vector. For example:

```
     A: 1 2 3 4 5
    CV: 0 1 1 0 0
old R: 6 7 8 9 10
new R: 6 2 3 9 10
```

Bit control vectors are usually generated by logical expressions involving two vectors or a vector and a scalar operand, or by using the vector function subprograms available on the Cyber-205.

Vectorization of decision making calculations can be done by combining the operations described above in different ways, or by using the WHERE Fortran-200 instruction described next.

● The WHERE instruction allows the execution of an operation on selected elements of one or two vectors according to a bit control vector. For example, the operation "WHERE (CV) R = A + B" is carried out as follows:

```
     A: 1  2  3  4  5
     B: 6  7  8  9 10
    CV: 0  0  1  1  0
     R: 0  0 11 13  0
```

In handling decision making with data handling operations, additional calculations are forced in the vector code which are unnecessary in a scalar code. Usually this extra work involved is offset by the much higher computation rates obtained from the vector calculations.

2.3 HEP MULTIPROCESSOR

The Heterogenous Element processor (HEP) became available in 1982 as the first truly commercial multiprocessor. It can execute a number of sequential (SISD) or parallel (MIMD) programs simultaneously. It can have up to 16 process execution modules (PEMs) with peak performance of 160 million instructions per second (MIPS) [16,14,7,22,13]. The architecture and programming aspects of the HEP are discussed in the following subsections.

2.3.1 Architecture

The HEP multiprocessor can have up to 16 process execution modules (PEMs) and up to 128 data memory modules (DMMs). These are connected with the INPUT/OUTPUT and control subsystem through a high-speed message switched network. A basic configuration of the HEP with four PEMs, four DMMs, a mass storage subsystem and an I/O control processor is shown in Figure 2.3

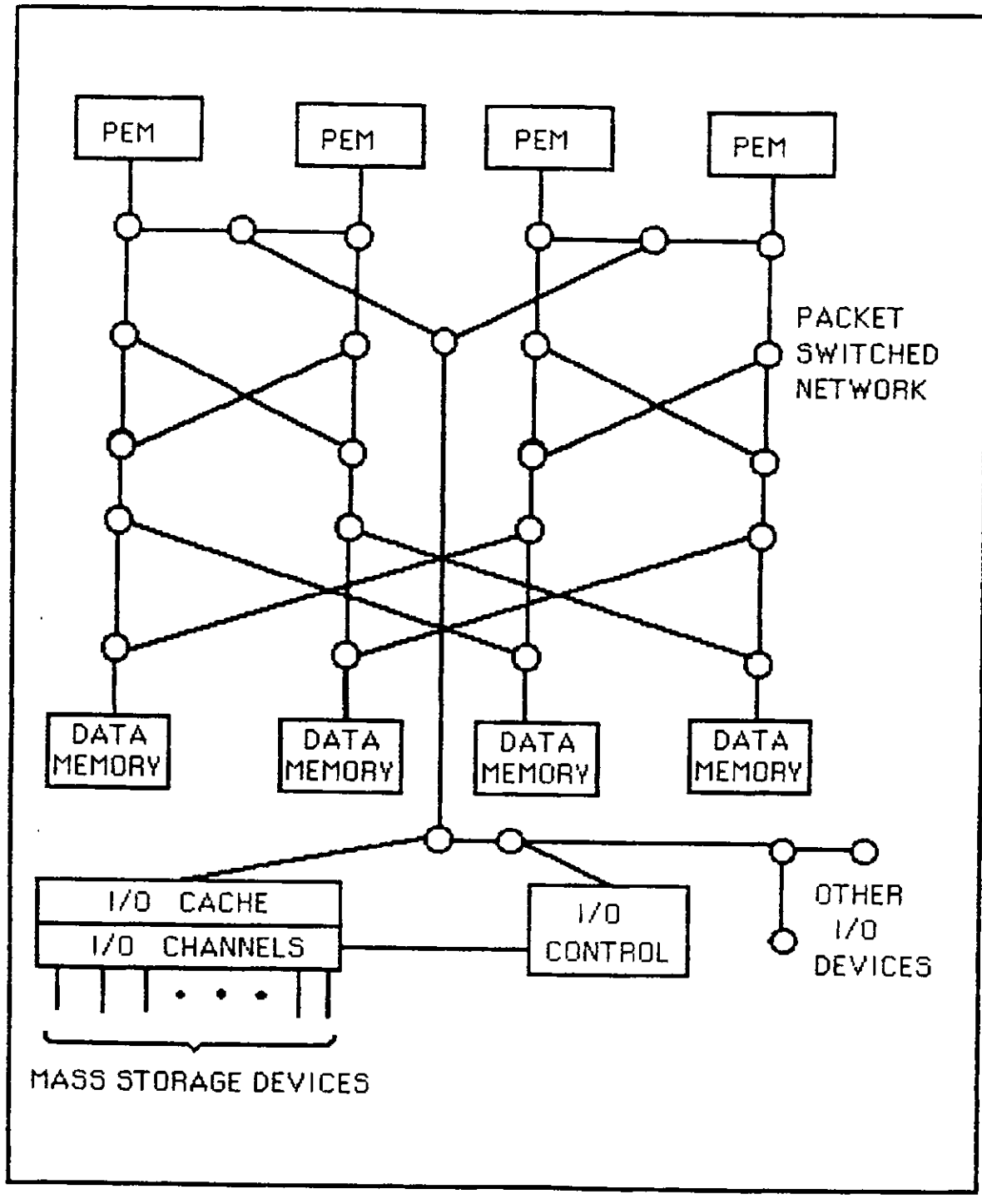


Fig. 2.3: Basic configuration of the HEP multiprocesor with four processors

The mass-storage subsystem consists of three main components. A large MOS buffer memory provides an I/O cache function to match the seek and rotational delays of disks. Disk storage modules provide storage increments of 600 megabytes. I/O channels couple the disk storage modules to the I/O cache and are controled by the I/O control processor. The cache memory can handle 32 I/O channels simultaneously, yielding a potential transfer rate of 80 megabytes/s.

The I/O control processor controls the communication between the disk controllers and their interfaces to the cache memory.

The process execution modules (PEMs)can execute multiple instruction streams on multiple data streams simultaneously. This is done by pipelining each PEM with multiple functional units. Because the multiple instructions executed concurrently by an MIMD machine are independent of each other (thus they don't influence one another), full parallelism in processing can be achieved.

Each PEM has its own program memory (ranging from 1 to 8 megabytes), and own instruction processing unit (IPU). The program memory buffers the instructions of active processes allocated to the PEM. Up to 50 instructions may be in various stages of execution operating on one or more data streams simultaneously, but the speed of execution is limited by the 100 ns instruction fetch rate.

There are four kinds of memories in a HEP system: program memory, register memory, constant memory and data memory. Program memory is read only and is local to a PEM. Register and constant memories are also local to a PEM and are similar in use except that constant memory cannot be altered by a user process. There are 2048 registers and 4096 constant locations in each PEM so that many processes running in parallel can have access to private working storage. Data memory is shared between the PEMs.

A single PEM can support up to 128 cooperating processes of which up to 64 may be user and up to 64 may be supervisor processes. Under the assumption that multiple processes will cooperate on a given job or task and thus share memory, memory is allocated and protected on the basis of a structure called the Task. Each of up to 16 Tasks is described by a Task Status Word (TSW) containing base and limit values for each memory type. A task may have up to 64 processes.

Potentially, an instruction may be completed in every 100 ns. Thus 10 MIPS is an execution rate representing a maximum utilization of the hardware of one PEM. Functional units are of two types: Synchronous and asynchronous. The synchronous function units are pipelined with 8 linear segments; thus, instructions are completed every 800 ns. Asynchronous function units like the divider do not

necessarily complete their operations within 800 ns. Usually one process cannot execute faster than 1.25 MIPS but if 8 to 12 processes are executing in a PEM they are sufficient for the Instruction Processing Unit to use all instruction cycles and thus achieve 10 MIPS execution rate.

Protection mechanisms, like hardware address check and bound check of TSWs, are provided for protection of one user process from another.

Scheduling of user processes to PEMs can be either static or dynamic. The HEP multiprocessor uses a distributed self-scheduling technique to balance the execution time of processes in a task. The number of processes in a task is defined by the programmer.

Cooperating processes synchronize by means of accesses to shared data. In data memory, the access states are "full" and "empty". A third access state "reserved" is implemented in the registers. More on software support is discussed in the next Section.

2.3.2 Programming Aspects

Language support for parallel processing in HEP is provided by extended Fortran-77. A new data type has been introduced to enable synchronization between cooperating and competing processes. This is called the asynchronous

variable which uses the access-state capability of the HEP hardware to support the correct interaction between processes. An asynchronous variable is identified by a "$" symbol and can have any data type. It may be written into only when its location is empty and be read if its location is full.

Other types of reads or writes that are also allowed are: VALUE($NAME), which accesses the variable $NAME regardless of its state; PURGE which clears the state of the variable; SETE($NAME) which reads the value of $NAME regardless of its state and sets it to empty. In total, there are two statements added and five intrinsic functions for testing and manipulating access states of asynchronous variables. For more details see references [16,14,22,13].

Parallel process creation can be handled in HEP by the CREATE/RESUME instructions which are also extensions to Fortran-77. CREATE is similar to a Fortran CALL but it causes the created subroutine to run in parallel with its creator. RESUME is syntactically like a RETURN from a subroutine. It causes the caller of a subroutine to resume execution in parallel with the subroutine. If the subroutine was created by a CREATE statement then RESUME has no effect.

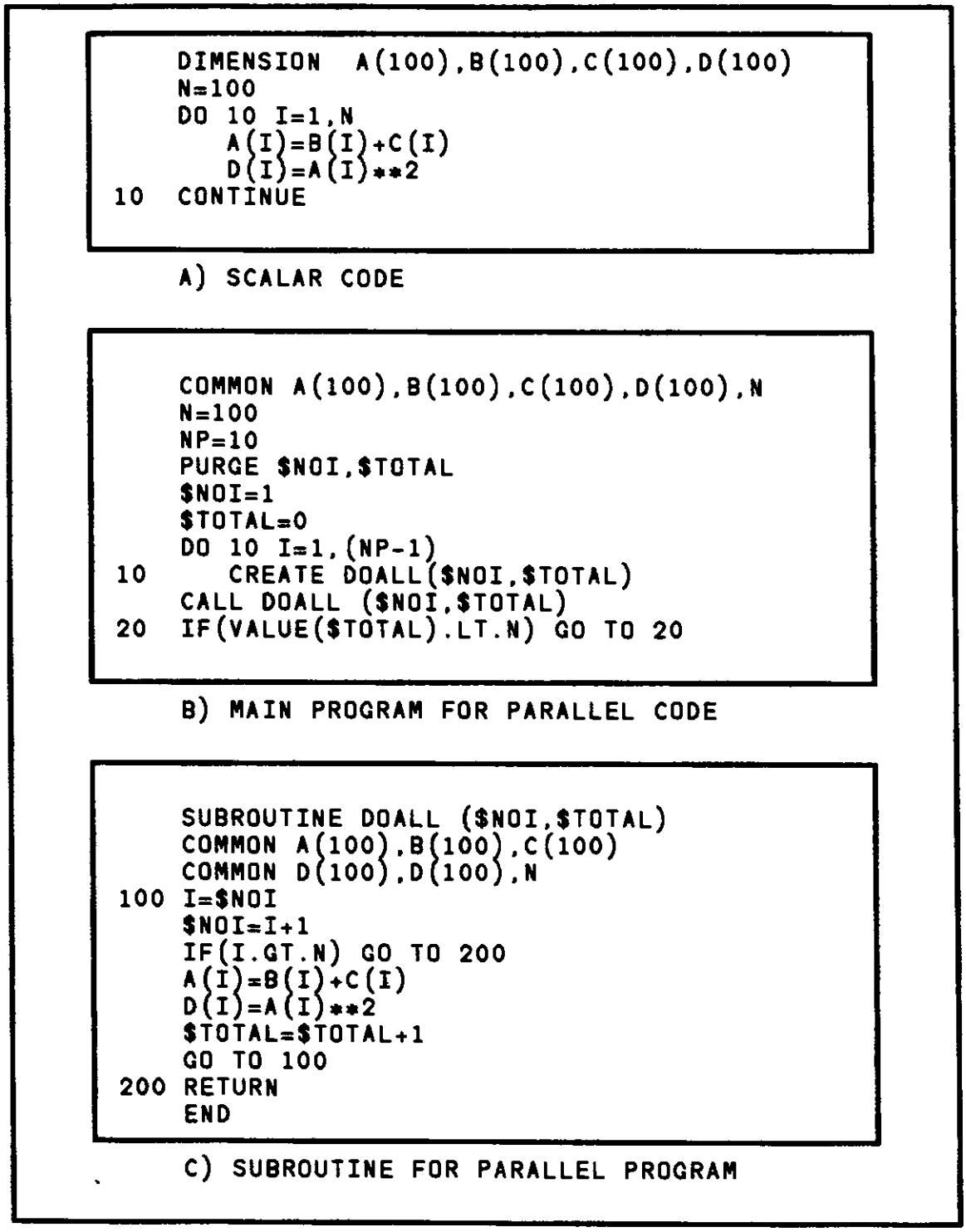
HEP Fortran generates fully reentrant code and dynamically allocates registers and local variables in data

memory as required by the program. Thus, it is easy to create several processes which execute identical programs on different data simultaneously.

Process scheduling can either be static or dynamic as mentioned in the Subsection before. The example given in Figure 2.4 demonstrates the conversion of a serial code to a parallel code with self scheduling. The DOALL task with 100 iterations is partitioned into 10 processes and each process can acquire the next iteration dynamically when it finishes the previous one [13].

In the above example two synchronization variables are used: $NOI keeps track of the iteration count while $TOTAL makes sure that all 100 iterations have been finished before the main program continues. First, the program creates NP-1 processes all executing subroutine DOALL, and then itself executes the subroutine DOALL by calling it, with the result that NP processes are executing DOALL at the end.

Self scheduling is more efficient than static scheduling when the number of iterations is much greater than the number of processes. In cases where the execution time between iterations varies widely, static scheduling may take more time because one process may get all the long iterations.



```
      DIMENSION  A(100),B(100),C(100),D(100)
      N=100
      DO 10 I=1,N
         A(I)=B(I)+C(I)
         D(I)=A(I)**2
10    CONTINUE
```

A) SCALAR CODE

```
      COMMON A(100),B(100),C(100),D(100),N
      N=100
      NP=10
      PURGE $NOI,$TOTAL
      $NOI=1
      $TOTAL=0
      DO 10 I=1,(NP-1)
10       CREATE DOALL($NOI,$TOTAL)
      CALL DOALL ($NOI,$TOTAL)
20    IF(VALUE($TOTAL).LT.N) GO TO 20
```

B) MAIN PROGRAM FOR PARALLEL CODE

```
      SUBROUTINE DOALL ($NOI,$TOTAL)
      COMMON A(100),B(100),C(100)
      COMMON D(100),D(100),N
100   I=$NOI
      $NOI=I+1
      IF(I.GT.N) GO TO 200
      A(I)=B(I)+C(I)
      D(I)=A(I)**2
      $TOTAL=$TOTAL+1
      GO TO 100
200   RETURN
      END
```

C) SUBROUTINE FOR PARALLEL PROGRAM

Fig. 2.4: Conversion of scalar to parallel code for HEP

2.4 SUMMARY AND CONCLUSIONS

The architecture and programming aspects of the Cyber-205 vector computer and the HEP multiprocessor are discussed in this Chapter. It is indicated that todays supercomputers can achieve high execution rates, provided the computations are making full utilization of their architectural characteristics.

The execution rates of 400 MFLOPS for Cyber-205 and 160 MIPS for the HEP multiprocessor are much larger compared to that of conventional computers. This simulates interest in developing new algorithms suitable for supercomputers and in converting existing scalar algorithms to parallel ones.

In the remaining Chapters specific aspects of converting an existing Monte Carlo scalar code to a parallel one are discussed. First, the fundamentals of the Monte Carlo method are discribed in the next Chapter, in order to illustrate the inherent parallelism of the method.

Chapter 3

THE RADIATION TRANSPORT MONTE CARLO METHOD AND SUPERCOMPUTERS

3.1 INTRODUCTION

The interactions of particles (radiation) travelling through a medium are stochastic in nature. Therefore, radiation transport problems may be solved by simulating the random processes involved and then infering the required solution by observing the behavior of tracked particles [7]. The simulation of the random processes is known as the Monte Carlo method.

Monte Carlo simulations are generally time consuming and therefore efforts directed towards making use of supercomputers to reduce the execution time are encouraged.

In this Chapter, the Monte Carlo method as applied to radiation transport problems is described in Section 3.2. In Section 3.3, previous work done on implementing Monte Carlo methods on supercomputers is reviewed and in Sections 3.4 and 3.5 the aspects of implementing Monte Carlo methods on supercomputers are discussed. Concluding remarks are given in Section 3.5.

3.2 MONTE CARLO METHODS IN RADIATION TRANSPORT

The Monte Carlo method is used to solve the Boltzmann transport equation. This is achieved by randomly tracking sufficient number of particles through the problem geometry. The sampling is governed by probability distributions representing the physics of the problem. An estimate of the quantity of interest is then obtained by accumulating the contribution of individual particles.

In order to determine how the Monte Carlo method is used to solve the Boltzmann transport equation let us consider the integral form of this equation [8] :

$$f(\underline{P}) = \int G(\underline{P}' \rightarrow \underline{P}) * f(\underline{P}')\, d\underline{P}' + S(\underline{P}) \qquad (1)$$

Where : $\underline{P}$ is a point in phase space with coordinates $(\underline{r}, E, \underline{\Omega}, t)$.

$\underline{P}'$ is another point in space.

$\underline{r}$ = position vector

E = energy variable

$\underline{\Omega}$ = direction cosine vector

t = time variable

$G(\underline{P}' \rightarrow \underline{P})\, d\underline{P}$ = the expected number of particles emerging from a collision at $\underline{P}$ is $d\underline{P}$, given that a particle comes out of collision at $\underline{P}$.

$S(\underline{P})$ = density of source particles generated at point $\underline{P}$.

$f(\underline{P}')d\underline{P}'$ = density of particles leaving a source or emerging from a collision is $d\underline{P}'$ at $\underline{P}'$.

In physical terms the Boltzmann equation is simply a bookkeeping mechanism for particle transport in space. The kernel $G(\underline{P}'\rightarrow\underline{P})$ provides the probability density function upon which a Monte Carlo simulation can be based. This kernel also accounts for both particle transport and collision in space.

The solution of the above equation provides the neutron (gamma-ray) density distribution in the $(\underline{r},E,\underline{\Omega},t)$ space. This quantity is essential in determining the behaviour of a nuclear system. Two of the major applications in this area are radiation shielding design and reactor criticality calculations[10].

To solve the Boltzmann equation using the Monte Carlo method one must provide the following items for each particle history (random walk).

1. Source particle
2. Transport distance
3. Geometry encountered
4. Collision physics
5. Scoring means

The above aspects are discussed in the following paragraphs.

1. Source Particle: A radiation source is defined by its position, energy and directional cosines $(\underline{r}, E, \underline{\Omega})$. These source variables are produced by randomly sampling from given probability distributions. Source particles are assigned a starting weight, and a starting age in the case of time dependent problems. This provide the $S(\underline{P})$ term in the transport equation (1).

2. Transport Distance: A particle defined by $(\underline{r}, E, \underline{\Omega})$ travelling through a medium eventually collides with a nucleus of the medium and consequently changes its energy and direction. The probability distribution function of the distance between collisions, travelled by a particle is given by [8],

$$T(s) = \sum_T(s) \text{ EXP } \left[-\int_0^S \sum_T(s')\, ds' \right] \qquad (2)$$

where

$\sum_T$ = total macroscopic cross-section and depends on the energy of the particle and the nature of the surrounding medium.

Equation (2) provides essentially the transport part $G(\underline{P}' \rightarrow \underline{P})$ of equation (1).

The distance travelled is determined in terms of the number of mean-free-paths travelled (λ), which can be converted to centimeters by dividing λ by the total cross-section of the medium,

$$D = \lambda / \Sigma \tag{3}$$

The value of s is randomly sampled from T(s).

3. Geometry Encountered: Generally the medium involved is not homogeneous but may consist of regions of different materials. If the lengths of the flight paths for each medium are defined as $s_1, s_2, s_3, \ldots, s_\eta$ then if,

$$\sum_{\zeta=1}^{\eta-1} \Sigma_\zeta s_\zeta \leq \lambda < \sum_{\zeta=1}^{\eta} \Sigma_\zeta s_\zeta \tag{4}$$

the flight terminates in the n-th region, encountered at a distance of

$$s'_\eta = \frac{1}{\Sigma_\eta} \left(\lambda - \sum_{\zeta=1}^{\eta} \Sigma_\zeta s_\zeta \right) \tag{5}$$

beyond the n-th region. is the cross-section of the medium in region ζ and λ is the number of mean-free paths travelled.

Three-dimensional geometries can be described easily using a combinatorial concept [31]. Some

basic geometrical shapes like sphere, cylinder, box, are defined and then the geometrical shape is defined by combining the basic shapes using the Append (+), Delete(-), and (OR) logical operators. Other methods for describing geometry, like the analytical or finite difference numerical methods become very complex in the case of 3-D geometries.

4. Collision Physics: If the particle does not escape the system, it is tested for collision consequences at the end of its flight path. When a collision takes place, mainly one of three events may happen. First, the particle may be absorbed into the medium. Second, the particle may be scattered, that is it leaves the point of collision in a new direction with a new energy. Third, fission of the struck nucleus may occur; in this case (if the original particle is a neutron) several other neutrons leave the point of collision with different energies and directions. Each of these possibilities has a certain probability specified by an appropriate cross-section of the medium that defines the interaction probability. This is the collision part of the $G(\underline{P}' \rightarrow \underline{P})$ kernel of equation (1). For the Monte Carlo purposes, random sampling is performed based on these probability distributions.

5. Scoring Means: The repetition of the procedure described above for a large number of particles can yield estimates of some physical parameters. For example, the number of particles escaping out of the medium gives an estimate for the escape probability. The number of particles reaching a lower thermal energy gives an estimate of the thermalization probability. In the same manner, the probability of occurence of other measurable quantities such as absorption rates may be estimated.

The above discussion considers an analog method where the particle is allowed to be absorbed by the medium. In some cases, for example in the simulation of deep penetration problems, an analog method yields very poor statistics since most of the particles are absorbed before they reach the region of interest. For this reason a non-analog method can be used where particle absorption is not permitted. Rather, the particle weight is multiplied by the non-absorption probability after the particle encounters a collision. This method yields better statistics because more particles are allowed to contribute to the quantity of interest, though with a reduced weight.

A typical flow diagram of a Monte Carlo program for radiation transport problems is shown in Figure 3.1. The Monte Carlo random walk follows the source particle

transport and collisions until the particle, and particles generated by fission if any, are killed by absorption, escape from the system or reach a low energy cut-off. Another source particle is then introduced and followed throughout its history and so on.

Typical problems can involve the processing of thousands of particle histories in order to achieve sufficient accuracy in the scores. Since the Monte Carlo analysis is based on random sampling from existing probability distributions, large memory capacity is required for the storage of these distributions. Therefore Monte Carlo calculations require: large computing time and large memory capacity.

For these reasons, it is desirable to implement Monte Carlo methods on supercomputers to speed up the computations.

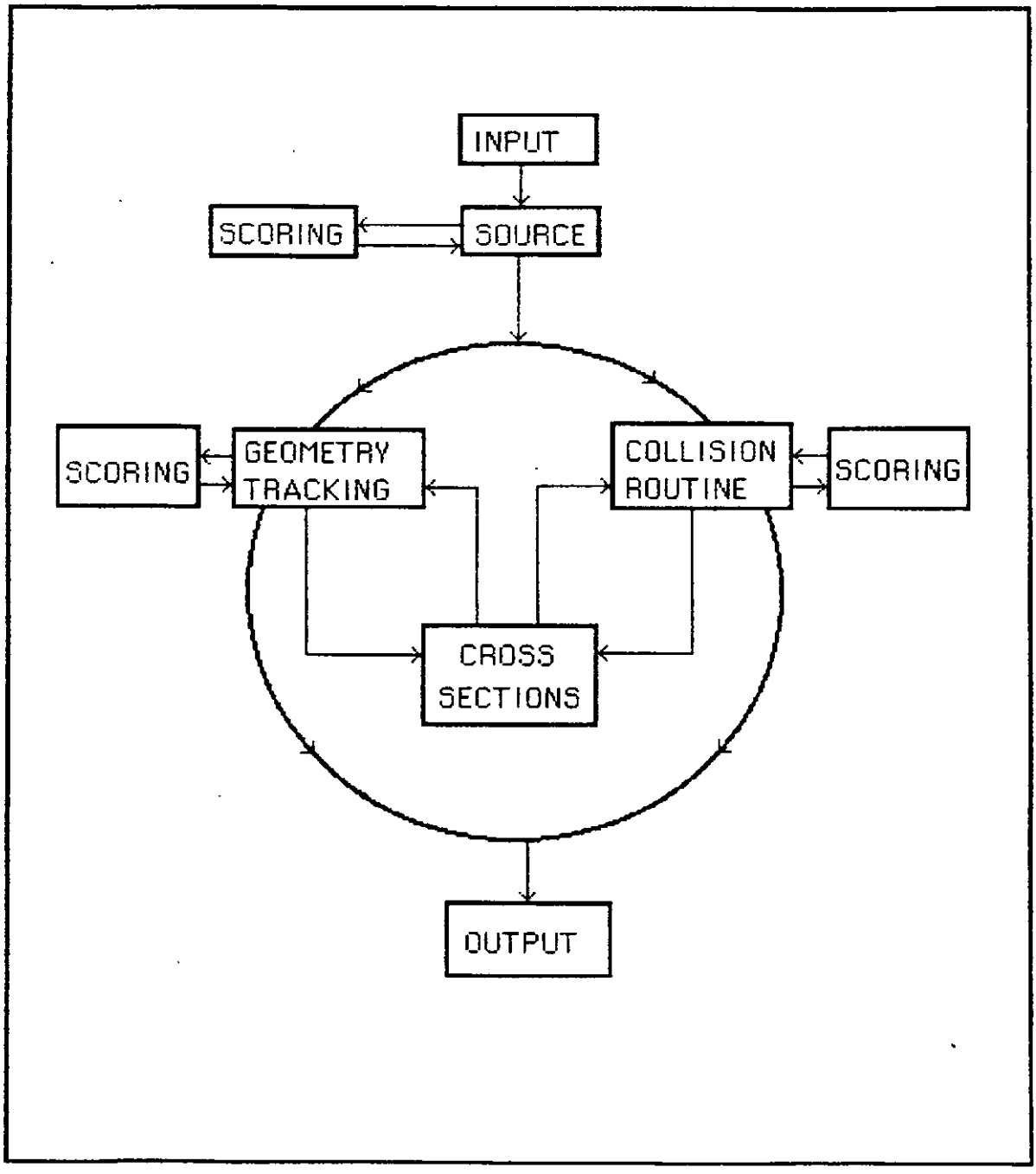


Fig. 3.1: Flow Diagram for the Monte Carlo method in radiation transport

3.3 REVIEW OF MONTE CARLO METHODS ON SUPERCOMPUTERS

Early work on implementing Monte Carlo methods on supercomputers was reported by Troubetzkoy et al.(1973) [25], who adapted a version of the continuous-energy code SAM-CE on the ILLIAC-IV computer. The ILLIAC-IV is a SIMD machine with 64 processing elements. The basic approach was to follow a number of histories in each of the 64 processing elements, and perform a given computation if "enough" processing elements had at least one particle waiting for that particular computation. The processing elements without waiting particles were disabled for that computation. This study introduced the basic technique of grouping the particles in stacks according to the type of computation they have to undergo. This technique has formed the basis for the development of almost all the recent vectorized Monte Carlo codes. Since the ILLIAC-IV was under development at that time, the code was simulated on a scalar computer with estimated speedup of 20 over a scalar code.

In the following subsections the previous work done on vector and multiprocessor computers is reviewed.

3.3.1 Monte Carlo Methods On Vector Computers

In 1979, D.A Calahan and his associates [4], started preliminary studies on vectorizing Monte Carlo codes. A simple 300 line Monte Carlo code for the continuous energy transport of gamma rays was implemented on the Cray-1 vector computer. The vectorization of this code was intended for basic algorithmic studies and development of vectorization techniques. The speedups obtained were of the order of 5-10 over scalar codes on the CDC 7600.

F. B Brown [2,3], in 1981, developed a demonstration code to investigate the potential of Monte Carlo vectorization. This was a multigroup Monte Carlo code for reactor shielding applications called MCVMG. It included a subset of the basic capabilities of the MORSE [31] and ANDY [20] codes. The speedups reported for small test problems were of the order of 20-40 over the codes on the CDC 7600. Many of the vectorization techniques developed for this demonstration code were used for the later development of a continuous-energy Monte Carlo code (MCV) for the Cyber-205 computer.

The MCV code was developed by Brown for the Knolls Atomic Power Laboratory, in 1983 [26]. In 1984, Brown and Mendelson [3] developed a general purpose neutron transport code for nuclear reactor analysis. This code uses a detailed pointwise cross-section representation, explicit collision physics models, and a continuous treatment of

neutron energy. Speedups of the order of 10-85 compered to the corresponding scelar codes on the CDC 7600 were obtained on running the codes on the Cyber-205.

Bobrowicz et al (1983) [26], have vectorized a photon transport Monte Carlo code for the CRAY-1 computer. In this code, each distinct set of computations required to solve the problem is assigned to a different stack. The stack is processed only when it is full for as long as possible. This approach is suitable for Cray-1 because it minimizes particle redistributions and keeps the vector length optimal (Cray-1 has no vector data shuffling capabilities and the vector length is limited to 64). The speedups obtained were of the order of 7-10 over an optimized CRAY-1 scalar code.

Martin et al (1985) [18], implemented an "all zone" 2-dimensional photon transport code on the Cray-1 and Cray-XMP (single processor) supercomputer. Speedups obtained from these implementations are of the order of 7-10.

3.3.2 Monte Carlo Methods on Multiprocessors

Neil et al. (1982) [11], implemented the Monte Carlo Metropolis algorithm on the Cm* multiprocessor. This is an experimental MIMD machine developed at Carnegie Mellon University. It is a cluster of LSI-11 processors. Reported speedups over the scalar algorithm are of the order of 20-30.

A report from Los Alamos Laboratory (1985), demonstrates the implementation of GAMTEB [16], a gamma-ray transport simulation code using the Monte Carlo method, on the HEP multiprocessor. Since the tracking of each particle is independent from all other histories, simulation is almost completely parallel. Only the accumulation of statistics couples one history to another. This creates the need for a critical region in the code which is updated by only one process at a time. As implemented on the HEP, each photon history is started by available processes using a self scheduling technique. Simulations were done on a single processing element (PEM), with speedups of the order of 7 over a scalar code.

3.4 PRINCIPLES OF VECTORIZING MONTE CARLO

Vector supercomputers attain their high execution rates when they execute identical operations on contiguous data elements (vectors). Conventional Monte Carlo codes, solving particle transport problems, treat one particle at a time independently and randomly according to probabilistic laws. This may imply that conventional Monte Carlo codes are not successful candidates for vector computers. Direct translation of scalar to vector codes may result in very small improvement in speedup, while direct vectorization through existing vectorizing compilers could be very inefficient. Vectorizing compilers [28,23,19] can generate

vectorized codes only for DO-loops not including IF statements, GO TO statements, recursive operations, subroutine calls etc. This is not the case in a conventional Monte Carlo code which, because of its probabilistic nature, includes a large number of conditional statements and conditional branches. Explicitly stated DO-loops are rarely used, rather implicit loops are used terminated by setting a condition within the loop.

From the above discussion it is obvious that conventional Monte Carlo codes are not directly vectorizable. Thus, some new techniques must be developed for vectorizing them efficiently, so that the vector hardware of vector computers can be utilized effectively.

Vectorization of Monte Carlo codes can take place in two stages. First, the scalar algorithm is restructured so that a global structure suited for vectorization can be obtained. Second, vectorization of local coding is done as required by the global algorithm.

In a conventional algorithm, a particle is followed from collision to collision through its random walk until its history is terminated. This approach is forced by the architecture of sequential computers as they can execute only one operation at a time. Vector computers, on the other hand, exploit the idea of executing identical operations on contiguous data elements (vectors). Thus a

vectorized algorithm should be designed to follow many particles through their random walks simultaneously. This is done by forming particle vectors and then applying the same operations to all the particles in the vector using vector instructions to speed up the computation rates.

To do this, a random walk is broken down into blocks with each block representing an event like introduction of particles from a source, sampling the distance to collision, etc.

As was mentioned before, the conditional branches are the main obstacle in vectorizing Monte Carlo codes. Conditional statements are involved mainly in implicit loops to decide the exit condition, in conditional coding to select between operations, and in optional coding where they decide which part of the code to execute depending on some input value (decision of which geometry to use). Following are some techniques useful for vectorizing conditional statements.

1. Implicit loops: In a scalar algorithm a particle iterates in an implicit loop until some exit condition is satisfied. The number of iterations are not known in advance and they differ from particle to particle. In a vectorized algorithm, where many particles are followed at the same time, some particles may satisfy the exit condition from the first pass through the loop while others may require

many passes. A general technique for solving this difficulty is termed "SHUFFLING" [2,3].

At the end of each pass through an implicit loop, particles that satisfied the exit condition are transfered to a storage area and held there until all particles have exited from the loop. The main vector is then compressed to provide contiguous vectors for the next pass through the loop. The implicit loop terminates when the main vector is empty.

2. Conditional coding: In a scalar algorithm the operations to be carried out on a particle are selected or skipped depending on the value of some condition. In a vectorized algorithm some of the particles in the main vector may qualify for a certain operation while others may not.

Cyber-205 has bit-controlled operation capability via the WHERE or block WHERE statements. A vector operation is performed on those elements of the vector corresponding to the '1' bits in a bit control vector, with the other elements remain unchanged.

The data handling operations of the Cyber-205 can also be used to vectorize conditional coding. The GATHER/SCATTER or COMPRESS/EXPAND operations can be used so that data can be operated on selectively.

Particles to be operated on are transfered into contiguous vectors. The operation is then performed on them and the result is transferred back to the main vector using the above operations. This approach involves a great deal of overhead cost because of the shuffling of particles back and forth.

3. Optional coding: This part of Monte carlo code is skipped or operated upon by all the particles according to some input option (selection of geometry shape). In the scalar code, a branch over the optional code is done by all the particles while in a vectorized code, this can be done by a single branch for all the particles in the main vector.

3.5 PRINCIPLES OF IMPLEMENTING MONTE CARLO ON MULTIPROCESSORS

Monte carlo methods for particle transport problems have an intrinsic parallelism in them since the histories of individual particles are independent from one another. Regarding the scalar Monte Carlo algorithm, the only thing that couples the random walks together is the collection of statistical parameters.

Thus, scalar Monte Carlo algorithms can be directly implemented on multiprocessor supercomputers, with only few changes. These changes depend on the requirements of the target machine and on the number of processors available.

The additional coding required is mainly for process communication and synchronization. Processes must be able to acquire access to a data structure in shared memory so that they accumulate their statistics. The required number of random walks is divided into processes depending on the available processors on a machine. Scheduling of the execution of random walks can either be static or dynamic. Dynamic scheduling creates more overheads but is preferable over static scheduling when the number of random walks is much greater than the number of processors [13].

3.6 SUMMARY AND CONCLUSIONS

In this Chapter, the Monte Carlo method as applied to radiation transport has been described in general. It is indicated that the Monte Carlo method although a very powerful tool for handling complicated problems with 3-dimensional geometries, has the drawback of requiring large execution times. One can overcome this drawback by converting Monte Carlo algorithms to suit the architectural characteristics of the new supercomputers.

A review of the previous work done on implementing Monte Carlo methods on vector computers shows that they can be effectively vectorized with considerable speedups over scalar codes. Little has been reported on implementing Monte Carlo codes on multiprocessors. Some principles of

implementation of Monte Carlo codes on vector and multiprocessor machines are discussed. Vectorizing Monte Carlo codes requires major changes in the global algorithm. Direct vectorization is not possible because of the many conditional statements present in the codes and the absence of explicitly stated loops. The intrinsic parallelism in the Monte Carlo algorithms can be exploited for implementing them on multiprocessor machines with little additional effort.

Chapter 4

THE CENTER-OF-MASS PROGRAM ON CYBER-205

4.1 INTRODUCTION

While the previous Chapter discussed principles of vectorizing Monte Carlo codes in general, this Chapter describes the specific implementation of the Center-of-Mass program on the Cyber-205 vector computer. Previous efforts in vectorizing Monte Carlo [26,2,3,18,25,4] showed that major restructuring in the vectorized algorithm is needed to achieve high speedups, as opposed to local recoding or direct translation from scalar to vector codes.

In this Chapter, the Center-of-Mass program is described in detail in Section 4.2. Some computational results of COM on the IBM 3081 computer and the CYBER-205 scalar processor are presented in Section 4.3, in order to provide a basis for comparison of the vectorized implementation of COM. In Section 4.4, the steps followed in designing VCOM (the vectorized COM program) are described along with the computational results of VCOM on the Cyber-205 vector computer. The results obtained in this Chapter are summarized in Section 4.5.

4.2 THE CENTER-OF-MASS MONTE CARLO PROGRAM (COM)

COM is a Center-Of-Mass Monte Carlo program that simulates the scattering of fast neutrons incident on a water-vapor two-phase flow in a pipe. The program is used to aid in designing neutron densitometers for use in two-phase flow systems [12].

The COM program assumes isotropic scattering in the center-of-mass system and utilizes functional fits for neutron cross-sections. Seven different flow regime geometries are treated using simple analytical geometry methods.

The neutron source used in COM is a uniform line source, i.e. a beam. A schematic represantation of the problem solved by the COM program is shown in Figure 4.1. Neutrons incident on the surface of the pipe are followed through the pipe until they reach a lower energy cut-off or they escape. The escape probability and the thermalization probability are calculated.

The particle random walk is defined in terms of the following steps:

1. Source Particle Position

   The X-coordinate of the source particle position is determined by choosing a random position on the line source. Since no collision is assumed in the

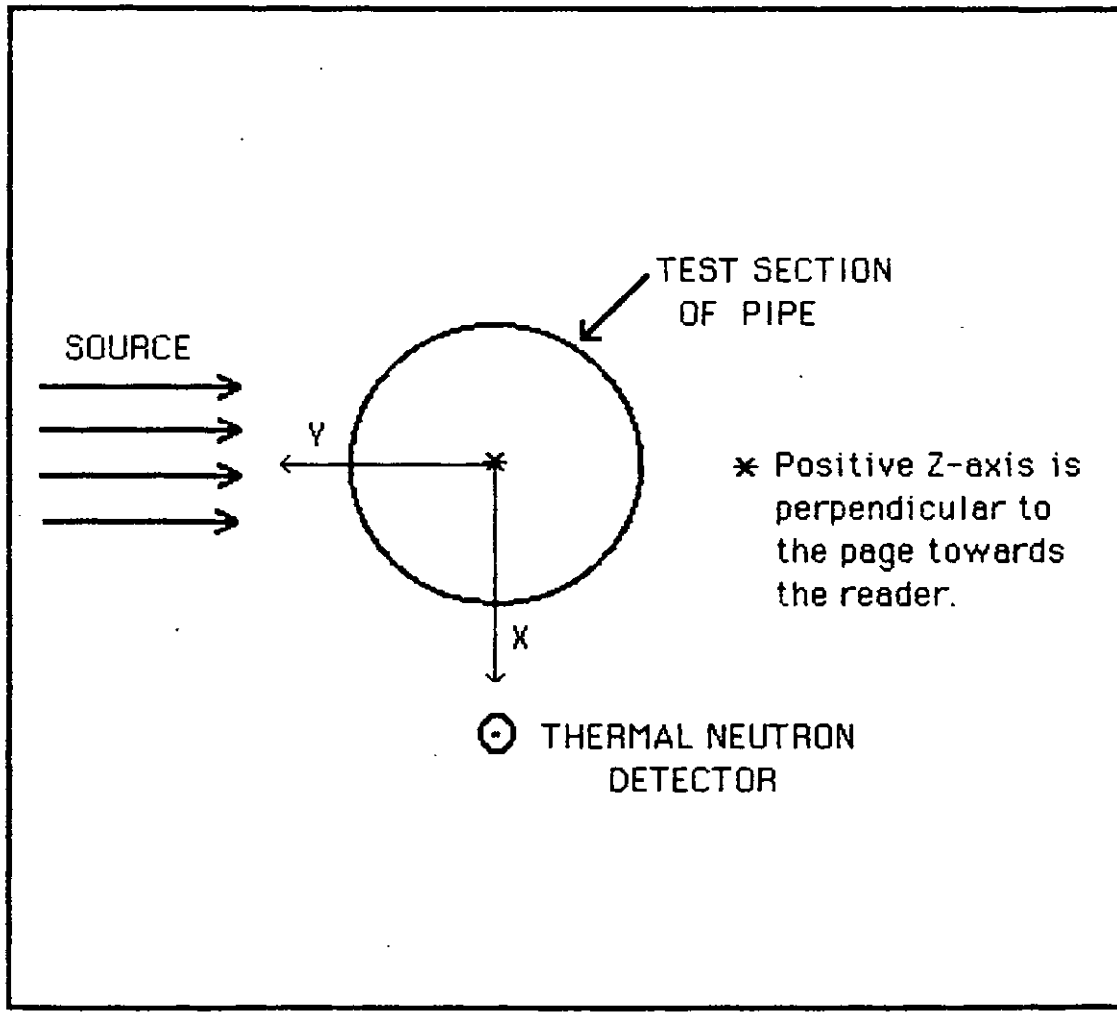


Fig. 4.1: A schematic representation of the COM problem

air between the source and the wall of the pipe, the Y-coordinate of the particle position lies on the pipe surface. The Z-coordinate is equal to that of the source.

2. Position of Next Collision

The distance (D), traveled by a particle before suffering a collision, is randomly picked from an exponential distribution such that :

$$D = - \ln \zeta / \Sigma \qquad (6)$$

where

Σ=the macroscopic cross-section measured in cm^{-1}

ζ=uniformly distributed random number in the interval [0,1].

The new position of the particle is then calculated given the incoming particle's directional cosines.

3. Particle Escape

If the newly calculated position of the particle lies outside the pipe or it is inside a void zone, then the particle escapes the scattering medium and the random walk is terminated.

4. New Particle Energy

a) Determine if neutron collides with the Oxygen or Hydrogen of water.

if $P_H > \zeta$, then collision is with Hydrogen, otherwise it is with oxygen.

Where

$$P_H = \frac{\text{Hydrogen cross-section in water}}{\text{total water cross-section}}$$

ζ=random number in the interval [0,1].

b) The new energy after a collision is calculated based on simple particle kinetics:

$$E_{new} = \alpha E_{old} + \zeta (E_{old} - \alpha E_{old}) \qquad (7)$$

Where

E_{old} = energy of incoming particle before scattering.

$$\alpha = \frac{(A-1)^2}{(A+1)^2}$$

A = Mass number (1 for Hydrogen, 16 for Oxygen)

ζ = random number between [0,1].

5. Energy Cut-off

If the new neutron energy lies below a preassigned value for the thermal energy, the neutron is considered to be thermalized, the thermal neutron counter is updated and the random walk is terminated.

6. New Directional Cosines.

Isotropic scattering is assumed in the center-of-mass system. The polar angle is randomly chosen in the center-of-mass system and then is transformed to the laboratory system (fixed frame of reference). For the azimuthal direction, isotropic scattering in the laboratory system is assumed, enabling direct sampling of the azimuthal angle.

Scoring Techniques

The COM program calculates two quantities: the thermalization probability and the escape probability of thermal neutrons to the detector. The product of the two provides an estimate of the detector response. Two different estimators are used for the thermalization probability. The first estimator evaluates at every collision the probability of reaching the thermal energy. The second estimator is the ratio of the number of thermalized particles over the total number of particles.

The two estimators are usually in agreement with each other, if a sufficient number of random walks is used. For each thermalized particle, the escape probability is estimated as the probability of the particle reaching the detector after escaping from the pipe. The escape probability is exponential in nature, therefore poor statistics are obtained for large pipes.

At the beginning of the COM program the pipe geometry is selected and a source biasing is introduced so that neutrons are prohibited from hitting the void in the pipe. At the end of the run the calculated probabilities are normalized.

The algorithm and the flow diagram for the random walks in COM are shown in Figures 4.2 and 4.3, respectively.

```
input parameters
select pipe geometry and introduce biasing
DO UNTIL ( particles exausted)
   compute particle position
   WHILE ( not escaped or thermalized ) DO
     IF( thermalized ) THEN DO
        increment thermal counter
        calculate escape probability
        END DO
     ELSE DO
        determine path length
        propagate particle
        IF ( escaped ) THEN
           increment escape counter
        ELSE DO
           determine new energy
           estimate thermalization probability
           calculate new directional cosines
        END DO
     END DO
   END DO
END DO
calculate statistical estimates
output results
STOP
```

Fig. 4.2: Algorithm for COM program.

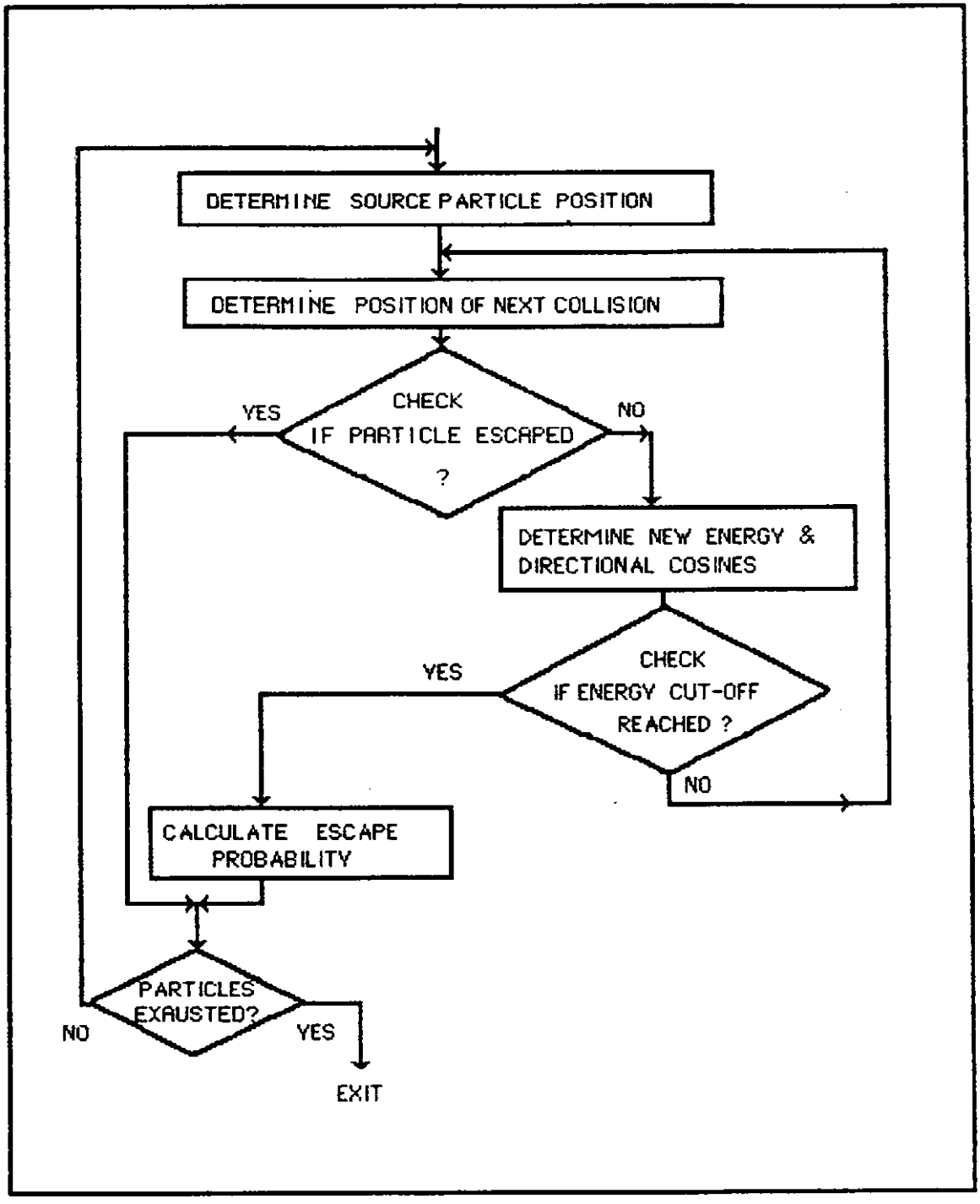


Fig. 4.3: Flow diagram for COM random walks

4.3 COMPUTATIONAL RESULTS ON SEQUENTIAL COMPUTERS

The COM program was run on the UNB IBM 3081 computer for different values of source energy. Two runs with different random number generator seeds were performed for each value of source energy. Convergence of the Monte Carlo solutions is assured for a large number of random walks as shown in Figure 4.4.

The duration of a random walk in COM can be increased by increasing the source energy, the pipe radius, and the liquid fraction accordingly. In a practical application, the source energy, pipe radius and liquid fraction values are low. Thus the generated random walks are short and the variance of their lengths is small. In the following runs of the COM program, artificially high source energy, large pipe radius and high liquid density are used in an effort to increase the variance of the random walk lengths. This is desirable, for the study of the behaviour of COM on multiprocessors, to create variance in the execution times of the different processors.

The thermalization probability obtained by following 5100 particle histories, is shown in Table 4.1 along with the estimated error and the CPU time consumed by each run.

TABLE 4.1

Thermalization probability by COM on the IBM 3081

ENERGY (MEV)	SEED1=58756421		SEED2=130948070	
	THERMAL. PROB.	TIME (SEC)	THERMAL. PROB.	TIME (SEC)
20	0.7610	18.13	0.7557	18.59
30	0.7192	17.47	0.7024	18.20
50	0.6286	16.29	0.6249	16.68

Note: Pipe radius = 20 cm.
Liquid fraction = 5.
5100 particle histories.

The thermalization probability is plotted in Figure 4.4 for batches of 255 particle histories. As it is seen from this Figure, the Monte Carlo solutions converge rapidly regardless of the initial random generator seed. This is due to the relatively small variance of 7.064, of the random walk lengths. The time recorded in Table 4.1 is the CPU time consumed by the random walks. Time used for INPUT/OUTPUT operations is not included.

In addition to the runs performed on the IBM 3081 computer, COM was also run on the University of Calgary CYBER-205 scalar processor. The Monte Carlo solutions along with the CPU time consumed by each run are shown in Table 4.2.

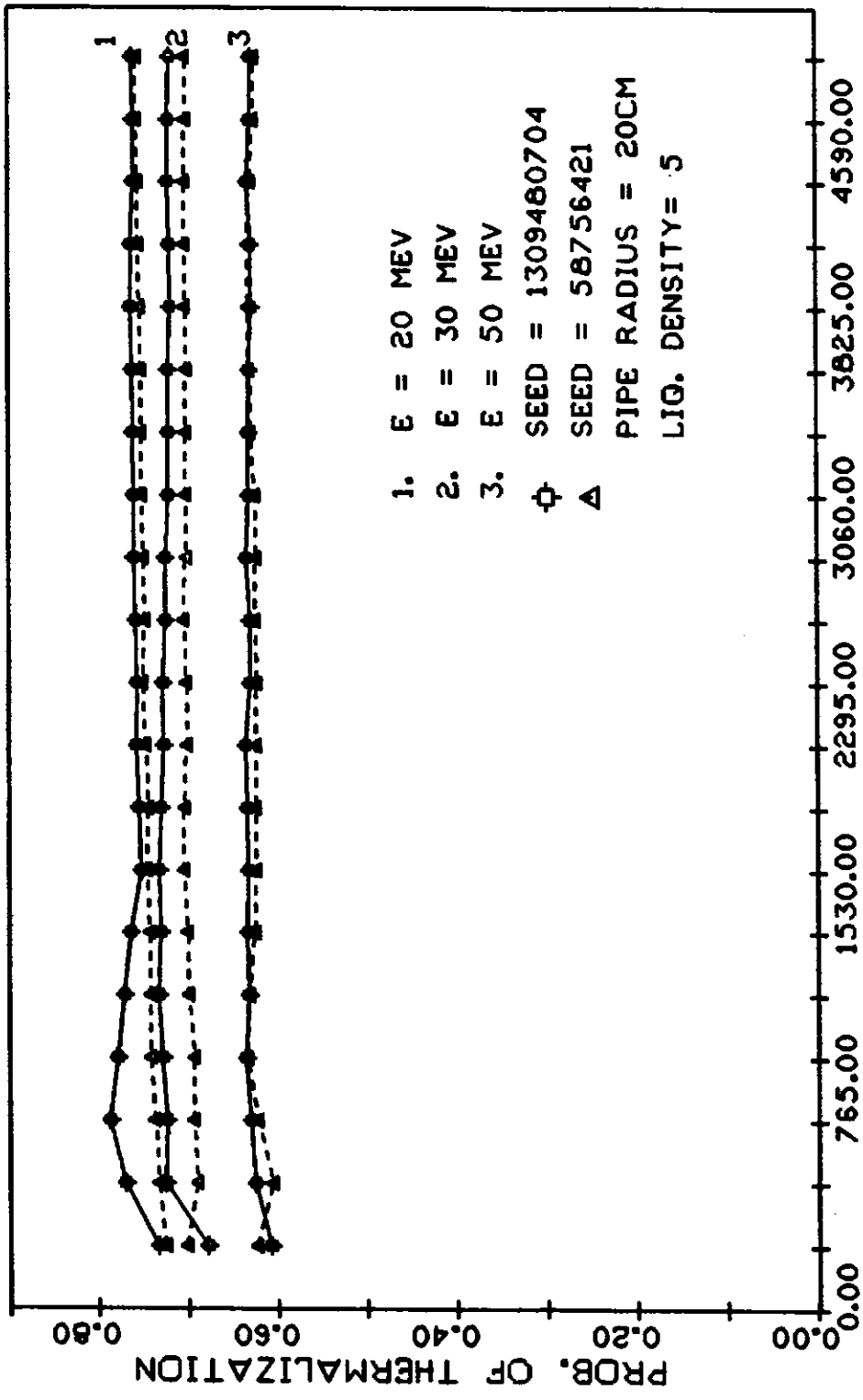


FIG. 4.4 : Thermalization Probability for batches of 255 Particles.

Comparing Table 4.1 with Table 4.2 it can be seen that the Monte Carlo solutions obtained on the Cyber-205 are very close to those obtained on the IBM 3081. Since the Cyber-205 uses 64-bit arithmetic, while the IBM 3081 uses 32-bit arithmetic, COM was run on double precision on the IBM 3081 so that the results are compatible. Comparing the timing requirements of COM on the two machines it is obvious that Cyber-205 is faster than the IBM 3081 by a factor of about 2.5. This was expected because the Cyber-205 scalar processor is heavily pipelined and has shorter clock cycle (20 ns) than the IBM 3081 (26 ns).

The above computetions were performed in order to provide a basis for comparison for the vectorized implementation of the COM program.

TABLE 4.2

Thermalization Probability by COM on the Cyber-205

ENERGY (MEV)	MONTE CARLO SOLUTIONS		TIME (SEC)
	THERMAL. PROBAB.	ERROR	
20	0.7498	0.01919	7.411555
30	0.7002	0.02142	7.197368
50	0.6276	0.02448	6.763295

Note : Pipe radius = 20 cm.
Liquid fraction = 5.
5100 particle histories.

4.4 VCOM = THE VECTORIZED COM

Architectural and programming aspects of the Cyber-205 vector computer for which VCOM was designed and some principles for vectorizing Monte Carlo codes in general, are discussed in Sections 2.2 and 2.4, respectively.

The basic idea for vectorization of COM is as follows. First the random walk is broken down into events. Then many particles are grouped into vectors and they are followed from event to event through the random walk. The operations in each event block are applied to all the particles in the vector using vector instructions. The steps followed in designing the VCOM program are as follows:

STEP 1. Form the particle vector:

In COM, a particle is defined by its position (3 variables), energy, weight and directional cosines (3 variables). Thus a stack[1] of eight vectors is required to hold all the necessary information about the particles. "VSTAK", a group of eight vectors of length N, is created to form the main working stack where all the operations in different events of the random walk are carried out. At the end of each pass through the random walk, the particles that have escaped or have

[1] Not the same as a FIFO STACK but a collection of vectors instead.

been thermalized are deleted from VSTAK; the stack has then to be compressed so that contiguous vectors are formed for the next pass. Deleting particles reduces the VSTAK length therefore a decision has then to be made on when to refill it with new particles so that maximum utilization of the vectors is achieved.

STEP 2. Decide when to refill VSTAK :

Because of the high vector start-up times of the Cyber-205 computer, it is more economical to work with long vectors for as much time as possible. Therefore, it is felt that it would be more efficient to refill VSTAK at the end of each pass through the random walk, regardless of the number of particles needed (the number of particles deleted from the stack each time varies randomly). However, the overheads incurred from the frequent refilling of VSTAK may cancel the benefits of working with full vectors at all times. Consequently, it was decided to add an input variable to the program to control the minimum VSTAK length (MINSTK) below which the stack is refilled. By varying MINSTK, the most efficient solution can be chosen at run time.

STEP 3. Vectorization of random walk events 1 to 3 :

The first three events of the random walk namely: the selection of source particle, calculation of distance traveled by the particle before collision, and a check to determine if the particle escaped, are vectorized by using bit control operations via the block WHERE statement for the decision points and other vector instructions and calls to vector functions. In the scalar algorithm, when a particle escapes, its random walk is terminated immediately. In our vector algorithm, escaped particles are kept in the stack and thus undergo further random walk events unecessarily. This forces the execution of unecessary operations, but on the other hand saves the compression of the stack from being carried out more than once in each pass through the random walk. The compression of the stack is rather a time consuming part of the job.

STEP 4. Calculation of the escape probability :

One has two alternatives for the calculation of the escape probability of thermalized neutrons. The first is to calculate it every time some of the particles are thermalized, and the second is to store the particles in a secondary stack and then calculate the escape probability when the stack is full. The latter

approach seems to be more efficient because it cuts down on the frequent execution of the part of code that computes the escape probability. Thus, a second vector stack (VDST) is set up to hold the necessaary information about thermalized particles. VDST consists of four vectors of length NM and holds the position and weight of particles. To control the processing of VDST another input variable (MAX) was introduced in the program defining the point above which VDST is to be executed.

Overflow of VDST is detected and taken care of by the program. In the case where VDST cannot hold the new thermalized particles, it is executed immediately and then the new particles are stored in it.

STEP 5. Compression of stack

After the escape probability is calculated, particles that have escaped or have been thermalized are deleted from VSTAK and the stack is compressed so that the remaining particles occupy contiguous elements. The new directional cosines are then calculated for the particles remaining in the VSTAK.

A vectorized algorithm and the flow diagram of the random walk in VCOM are shown in Figures 4.5 and 4.6 respectively.

In Figure 4.6 the operations performed in each of the two stacks are enclosed in their corresponding squares.

```
DO UNTIL ( particles exausted )
  IF ( VDST not full ) THEN DO
     propagate particles
     IF ( escaped ) THEN increment escape counter
     calculate new energies
     IF ( thermalized ) THEN increment thermal counter
     delete escaped particles
     transfer thermalized particles in VDST
     compress VSTAK
     calculate new directional cosines
     END DO
   ELSE DO
     calculate escape probability
   END DO
     increment particle counter
END DO
```

Fig. 4.5: Vectorized algorithm for COM random walks

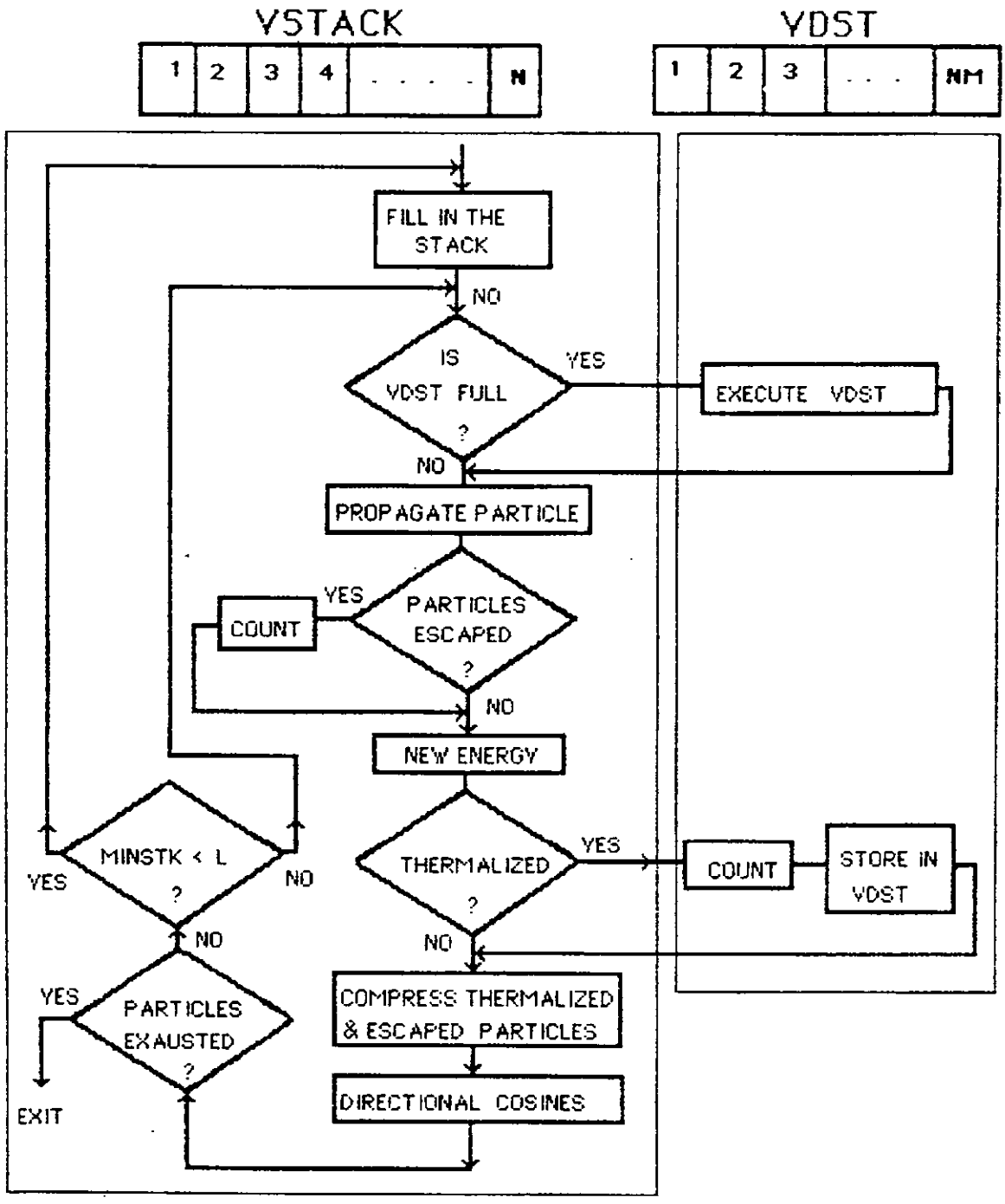


Fig. 4.6: Flow diagram for vectorized random walks

4.5 IMPLEMENTATION AND TESTING OF VCOM

VCOM has been coded according to the vectorized algorithm discussed above using FORTRAN-200. Almost 100% of the code for the random walks has been vectorized. The only non-vectorized parts are decision points for stack handling and optional coding. A listing of VCOM is given in Appendix 4.

Bit control operations via the block WHERE statement of Fortran-200 are used to vectorize decision points and logical branches. The COMPRESS, EXPAND, MERGE operators are utilized for handling the stacks. Vector library functions and hardware coded functions are also used in the vectorized code.

In order to debug the vector program, the random number generators in both COM and VCOM were set to return a constant number of 0.5. At every event of the random walk, the results were dumped out and compared. This provides an efficient way of debugging VCOM which otherwise would have been very difficult.

To make sure that VCOM was working correctly, Monte Carlo solutions were obtained for different numbers of particle histories both for COM and VCOM on the Cyber-205. The solutions obtained are listed in Table 4.3 and plotted in Figure 4.7.

TABLE 4.3

Comparison of the Monte Carlo solutions of COM and VCOM

NUMBER OF PARTICLE HISTORIES	MONTE CARLO SOLUTIONS (THERMALIZATION PROBABILITY)	
	COM	VCOM
100	0.7300	0.7700
500	0.7460	0.7370
1000	0.7410	0.7305
2000	0.7555	0.7360
3000	0.7547	0.7312
4000	0.7497	0.7373
5000	0.7498	0.7352

Note: Parameters for - VSTAK (N=1000, MINSTK=500)
- VDST (NM=1000, MAX=700)

As it is seen from Figure 4.7, the two programs result in almost the same solution and thus VCOM is correctly coded. Further, these solutions converge to a constant value as the number of particles increases; this is expected for Monte Carlo solutions. The small difference in the results obtained for COM and VCOM is due to statistical variations (e.g. sequence of random numbers used). For details on random number generation and testing see [17].

Table 4.4 shows the timing requirements for COM and VCOM for different number of particle histories. For VCOM, the vector length used for these runs was 1000. VSTACK was refilled after it was emptied below 500, while VDST was executed after it was filled above 700. These parameters

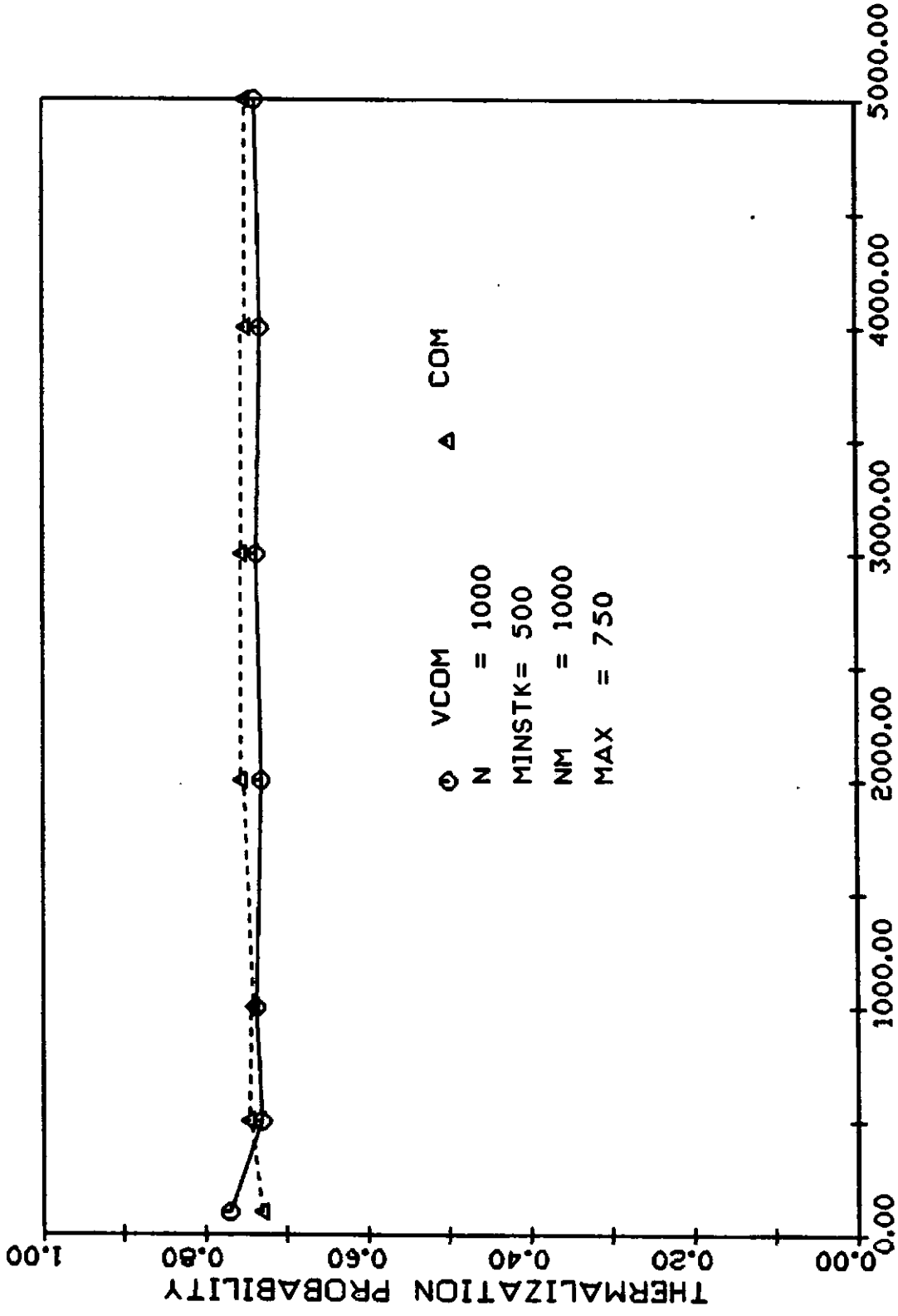


FIG. 4.7 : Comparison of Monte Carlo solutions of COM and VCOM.

are also listed in Figura 4.8. The speedup of VCOM over COM was calculated and it is also listed in Table 4.4. This speedup represents the ratio of the execution time for COM over the execution time for VCOM on the Cyber-205 excluding the input/output time.

To obtain the speedup of VCOM over the time taken for COM on the IBM 3081 the numbers given in Table 4.4 should be multiplied by a factor of 2.5.

As it can be seen from Figure 4.8, up to about 1000 particles, the speedup rises steeply but begins to flatten down as the number of particles increases beyond that point. This is so because, having a large number of particles the number of iterations (i.e. passes through the random walk) increases. As a result, the vector start-up overhead increases which has negative effect on speedup. The maximum speedup obtained at 5000 particles is of the order of 16 over the Cyber-205 and of the order of 40 over the IBM 3081.

For the previous runs, the selection of the stack parameters was done intuitively. In order to determine how the speedup is affected by varying the various stack parameters the following experiments were conducted.

EXPERIMENT 1:

The speedup as a factor of VSTACK length N was determined for 5100 particle histories. The speedup

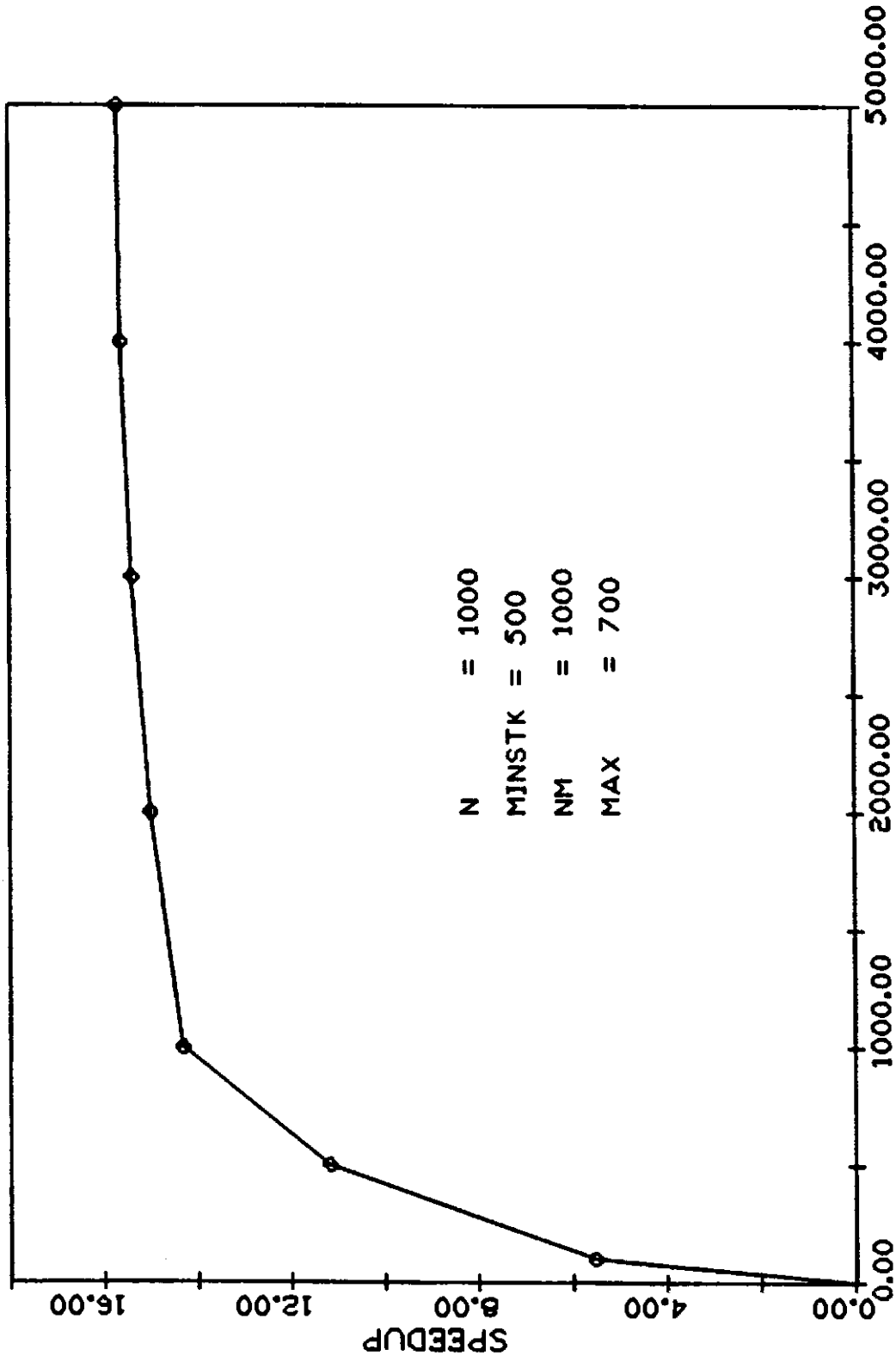


FIG. 4.8 - Speedup of VCOM over COM versus Number of Particles.

TABLE 4.4

Timing Requirements of COM and VCOM

NUMBER OF PARTICLE HISTORIES	TIME FOR COM (SECONDS) CYBER-205	TIME FOR VCOM (SECONDS) CYBER-205	SPEEDUP
100	0.161817	0.029208	5.540
500	0.722538	0.064625	11.180
1000	1.432753	0.100098	14.313
2000	2.934277	0.195696	14.994
3000	4.375813	0.284513	15.380
4000	5.800019	0.371672	15.605
5000	7.411555	0.472985	15.670

Note: Parameters for - VSTAK (N=1000, MINSTK=500)
- VDST (NM=1000, MAX=700)

obtained is plotted in Figure 4.9. The stack parameters used for these runs along with the timing requirements of VCOM and the speedup over COM are listed in Table 4.5. The relation between N and the other stack parameters are shown in Figure 4.9; as it can be seen the speedup increases rapidly for vector lengths up to a 1000. As the vector length increases beyond this value the speedup tends to become constant as was expected. The maximum speedup at vector length of 5000 is of the order of 16.

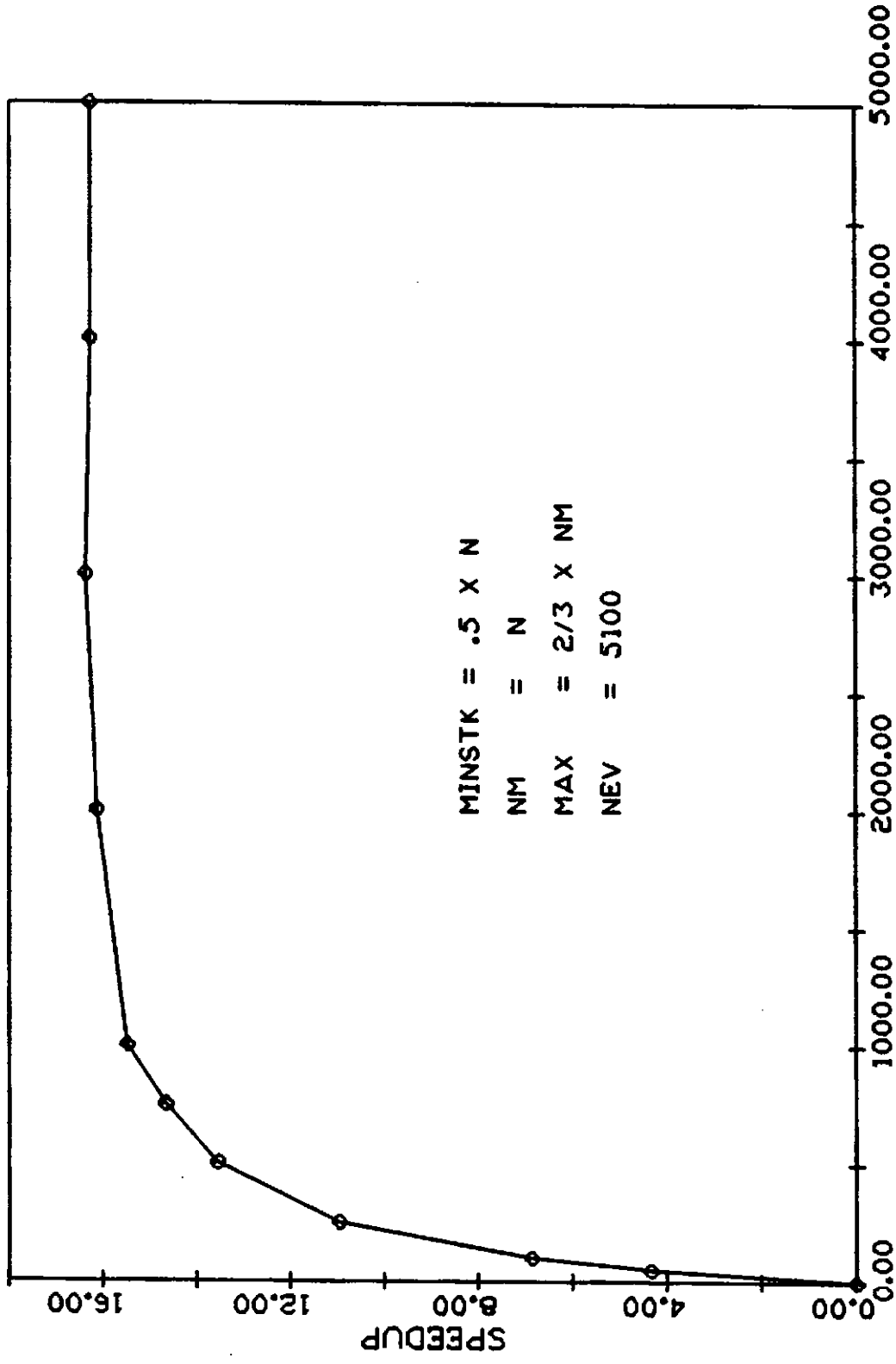


FIG. 4.9 : Speedup of VCOM over COM as a factor of VSTACK length.

TABLE 4.5

Timing Requirements of VCOM and Speedup over COM.

VSTACK		VDST		TIME (SECONDS)	SPEEDUP
LENGTH (N)	MINIMUM LENGTH (MINSTK)	MINIMUM LENGTH (MAX)	LENGTH (NM)		
50	25	30	50	1.706552	4.34
100	50	65	100	1.079389	6.87
250	125	180	250	0.675997	10.96
500	250	350	500	0.546942	13.55
750	375	500	750	0.506006	14.65
1000	500	750	1000	0.479060	15.47
2000	1000	1600	2000	0.459742	16.12
3000	1500	2000	3000	0.452758	16.36
4000	2000	3000	4000	0.455409	16.27
5000	2500	3600	5000	0.455837	16.26

Note: Time for COM = 7.411555 seconds.

EXPERIMENT 2:

This experiment was conducted in order to determine the effect of the minimum VSTAK length on the speedup. As discussed earlier, it was felt that maximum speedup can be achieved by working with full vectors as long as possible. Thus, one can expect the speedup to vary considerably by varying MINSTK between a length of 1 and 1000. Table 4.6, shows the timings for VCOM and the speedup over COM for different values of MINSTK and for 5100 particle histories. The speedup versus MINSTK is also plotted in figure 4.10. Surprisingly the

speedup was found to be almost constant and thus not dependent on the minimum vector length (MINSTK). In order to further investigate this trend the following experiment was conducted.

TABLE 4.6

Speedup of VCOM over COM as a factor of MINSTK

MINIMUM VSTACK LENGTH	TIME (SECONDS)	SPEEDUP
10	0.517648	14.317
50	0.504781	14.683
100	0.500726	14.801
200	0.493406	15.021
400	0.490043	15.124
500	0.480237	15.433
600	0.473296	15.659
700	0.477753	15.513
800	0.473736	15.644
900	0.473553	15.650
1000	0.487537	15.202

Note: Time for COM = 7.411555 seconds
Parameters of - VSTAK (N=1000)
- VDST (NM=1000, MAX=700)

EXPERIMENT 3 :

The speedup was obtained by varying all the stack parameters equally from length 1 to 1000. The resulting timings for VCOM and the speedup are listed in Table 4.7. The plot of the speedup versus the stack parameters is shown in Figure 4.11.

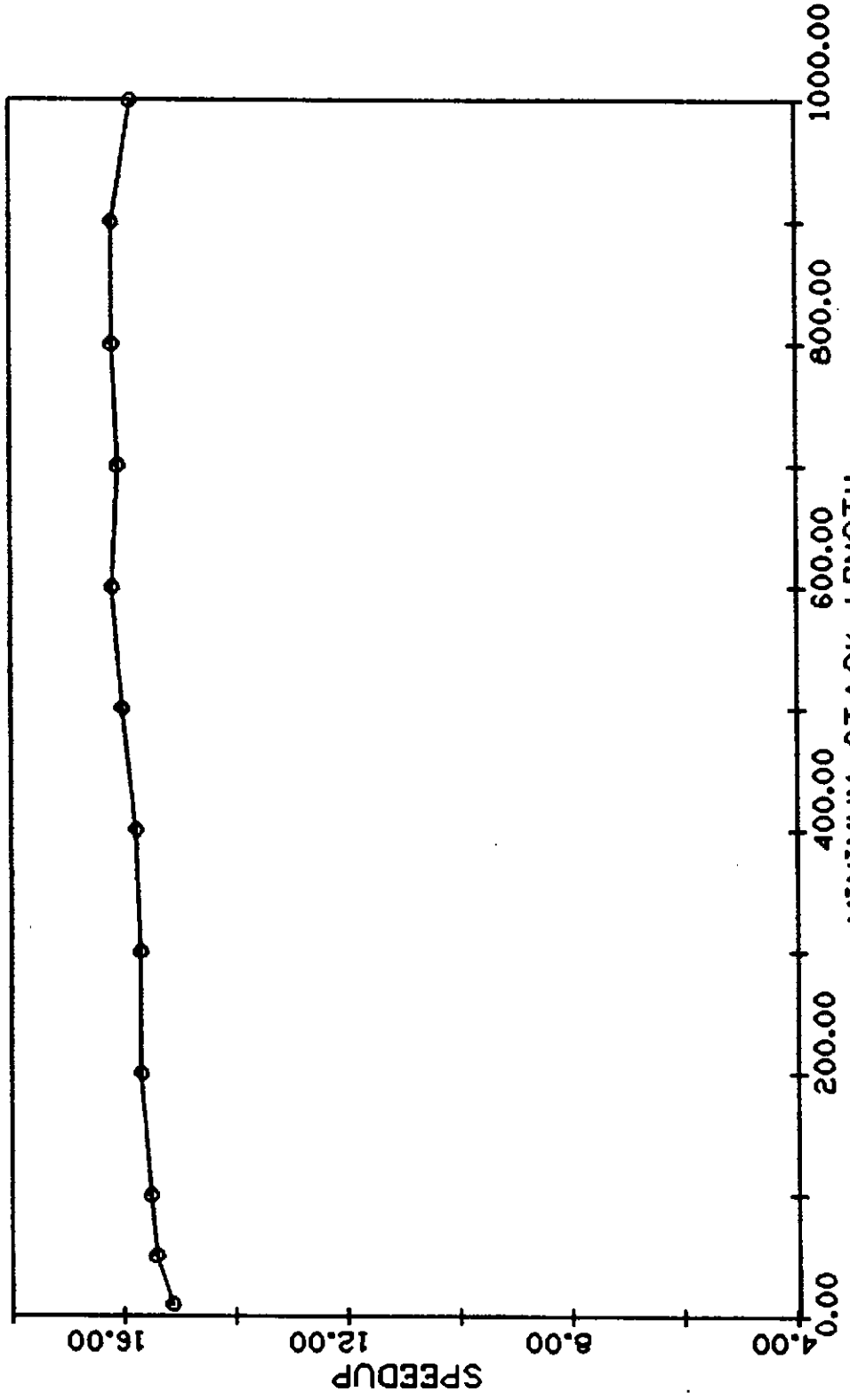


FIG. 4.10 : Speedup of VCOM over COM as a factor of MINSTK.

TABLE 4.7

Speedup OF VCOM as a factor of the length of Stack parameters.

STACK PARAMETERS LENGTH	TIME (SECONDS)	SPEEDUP
1	42.169812	0.175
2	21.901034	0.338
5	9.348774	0.792
10	5.022366	1.475
20	2.922976	2.535
50	1.480099	5.007
75	1.137861	6.513
100	0.988661	7.496
125	0.858883	8.629
150	0.777622	9.531
200	0.693978	10.679
250	0.643482	11.517
300	0.600691	12.338
350	0.585217	12.664
400	0.554737	13.360
450	0.547741	13.531
500	0.527785	14.042
550	0.523185	14.166
600	0.516761	14.342
650	0.507983	14.590
700	0.504616	14.687
750	0.500459	14.809
1000	0.487037	15.217

Note: 5100 particles were used for these runs.
N=NM=MINSTK=MAX, Time for COM=7.411555 sec.

As it is seen from Figure 4.11 the slope of the speedup rises steeply until about 500. From 500 to 1000 length the speedup is almost constant. Based on this, one can explain the rather constant speedup obtained in experiment 2 as follows: By having N=1000 and varying MINSTK from 1 to 1000 we were getting an

average vector length between 500 and 1000. But as it was seen in experiment 3 the speedup was almost constant for vector lengths between 500 and 1000.

EXPERIMENT 4 :

The speedup was also calculated as a function of MAX (maximum VDST length), for 5100 particle histories. The values of the rest of stack parameters are shown in Table 4.8 along with the timings for VCOM and the speedup. As it was expected, there is not much variation in the speedup as MAX varies from a length of 5 to 1000. This is because MAX controls only the frequency at which VDST is executed. The number of particles thermalized and thus stored in VDST at each iteration is not constant. For example, MAX may be equal to 5 but 100 particles may be thermalized in one pass, which is equivalent to MAX being equal to 100.

EXPERIMENT 5 :

So far, the speedups obtained, show how much faster VCOM is with respect to COM. This experiment examines the variation in the timing requirements of VCOM as the length of stack parameters increases as shown in Table 4.7. The ratio of the time taken for different lengths

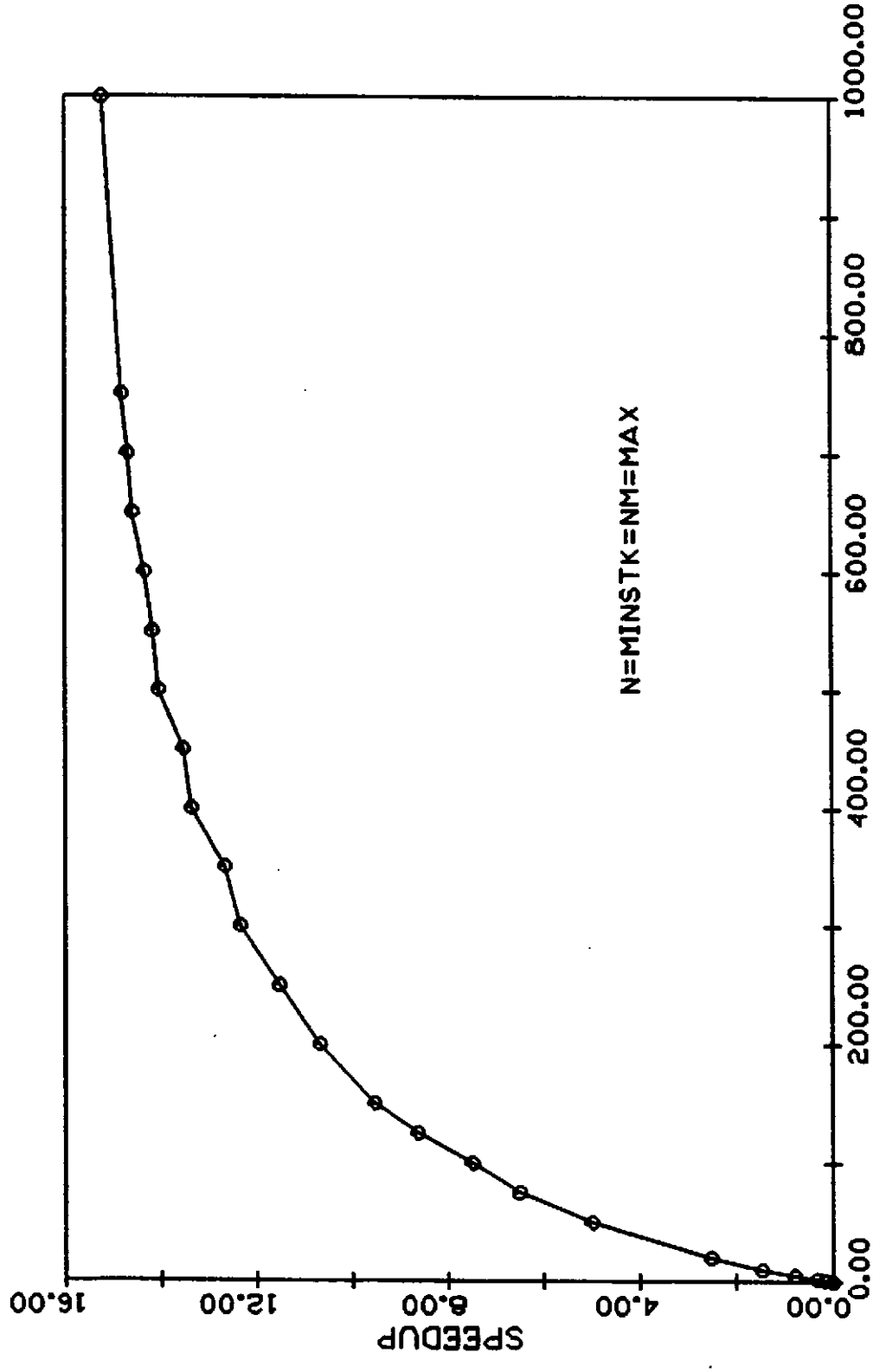


FIG. 4.11 : Speedup of VCOM over COM for same parameters for VSTACK and VDST.

TABLE 4.8

Speedup of VCOM over COM as a factor of MAX for 5100 particles.

MINIMUM VDST LENGTH (MAX)	TIME (SECONDS)	SPEEDUP
5	0.511518	14.489
50	0.495710	14.951
100	0.487737	15.195
400	0.479000	15.473
600	0.472985	15.670
800	0.473209	15.662
1000	0.486870	15.223

Note: Parameters for - VSTAK (N=1000, MINSTK =500)
- VDST (NM = 1000)

of stack parameters over the time taken by stack parameters of length one is plotted in Figure 4.12. As it can be seen the slope of the curve rises rapidly up to a vector length of 500 and then tends to stay constant. The maximum speedup obtained at a vector length of 1000 is of the order of 90. This result shows very well the effect of the vector length on the speedup. An interesting observation is that half of the maximum speedup is achieved at a very small vector length of 125.

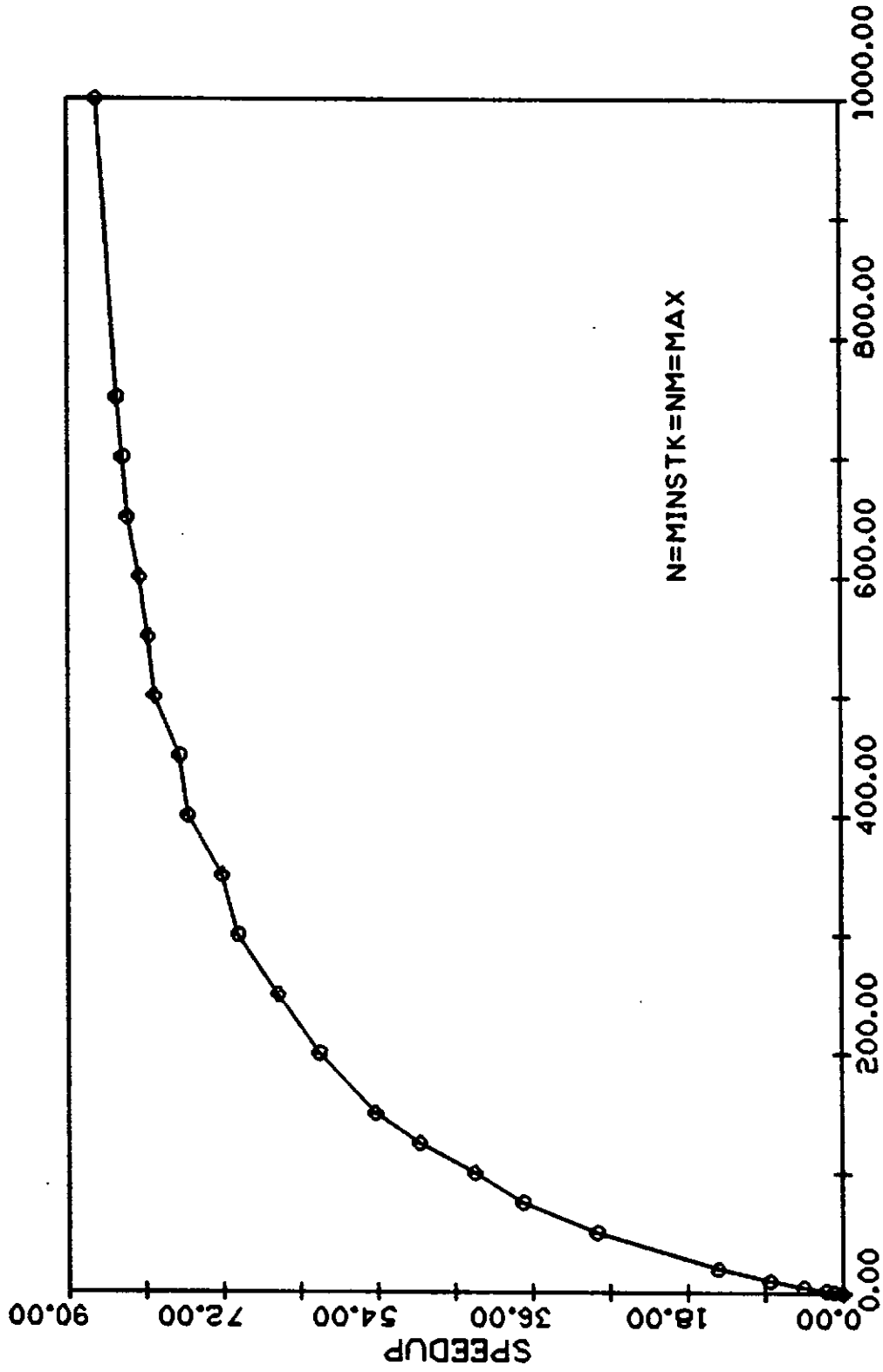


FIG. 4.12: Variation in speedup in VCOM versus stack parameters length.

4.6 A PRACTICAL APPLICATION OF VCOM

As it was mentioned in Section 4.3, artificially high incident neutron energy, high liquid fraction and large pipe radius were used in COM so that the variance of the random walk lengths may increase. In order to find out the speedup of VCOM over COM for an actual engineering application, some runs with realistic parameters have been performed as follows. For 3 Mev source energy, 5 cm pipe radius and liquid fraction of 1, the time taken for COM on the IBM 3081 is 5.2241 sec, while the time taken on the CYBER-205 scalar processor is 2.2968 sec. The time taken for VCOM on the CYBER-205 vector processor is 0.167039 sec which results in a speedup of 31.27 over the IBM 3081 and of 13.75 over the CYBER-205 scalar processor. These speedups are a little lower than the ones found in Section 4.5; but relatively of the same order. This shows that the random walk length does not affect the speedup very much.

4.7 SUMMARY AND CONCLUSIONS

In this Chapter, the Center-of-Mass Monte Carlo program has been described. Computational results of COM are obtained on the IBM 3081 and the CYBER-205 computers. These results show that a speedup of about 2.5 over the IBM 3081 is obtained by running COM on the CYBER-205.

The steps followed in designing the vectorized algorithm for COM are explained, along with the implementation of the vectorized code (VCOM) on the CYBER-205. Computational results obtained on the CYBER-205 show that vectorization of COM is succesful. The speedup obtained for 5100 particle histories with vector lengths of 1000 is of the order of 14 to 16 compared to that of the scalar mode run on the same machine and of the order of 32 to 40 compared to that of the IBM 3081 computer. Some experiments are carried out to determine the effect of varying the different stack parameters on the speedup.

It is shown that the speedup increases considerably up to a stack length of 1000. After that the speedup remains almost constant. It was believed that MINSTK should affect speedup a great deal, but this is not correct. On the contrary, it is proved that the speedup remains almost unaffected by varying MINSTK because of the averaging effect on the vector length. The variation of the secondary stack parameter MAX does not affect the speedup very much. This was expected since MAX does not have control over the number of thermalized particles at each iteration. The effect of the variation of the vector length on the speedup is demonstrated with a speedup factor of 90 for vector length 1000 over a vector length of 1. Finally, it is shown that the speedup does not depend on the length of the random walks in VCOM.

Comparing the speedups obtained from this implementation to those reported in the literature, it can be said that our vector code is effectively vectorized since the obtained speedup is among the highest reported[2-4,17,25].

Chapter 5

THE CENTER-OF-MASS PROGRAM ON MULTIPROCESSORS

5.1 INTRODUCTION

General aspects for implementing Monte Carlo Algorithms on multiprocessors are discussed in Chapter 3. This Chapter considers the specific implementation of the Center-of-Mass program on multiprocesssors. In Section 5.2, the performance of COM on such machines is predicted for different computational schemes, based on existing theoretical formulae [1]. The performance of COM on multiprocessors is also estimated through simulation on the IBM 3081 computer and the results are given in Section 5.3. A parallel algorithm for the implementation of COM on multiprocessors is proposed in Section 5.4. The concluding remarks for this Chapter are given in Section 5.5.

5.2 PREDICTED PERFORMANCE ON MULTIPROCESSORS

The computation of 4096 (this number is chosen so that is a power of 2) particle histories (random walks) is considered for the prediction of the performance of COM on multiprocessor machines. Two basic schemes for the assignment of the random walks to the processors in a

parallel computer are used: first is the static computation assignment scheme, and second is the dynamic computation assignment scheme. A static computation assignment (SCA) scheme is defined as the assignment scheme in which a fixed number of random walks are assigned to each of the processors, before any random walks are initiated. A dynamic computation assignment (DCA) scheme is defined as the assignment scheme in which the number of random walks carried out by each process is determined dynamically (during the computations). The above schemes were designed by Bhavsar [1] for predicting the Monte Carlo speedups on multiprocessor machines.

5.2.1 Performance of SCA schemes

Three SCA schemes are considered based on the number of processors used (P) and the number of random walks to be carried out (K). The computation of 4096 random walks is considered with the number of processors varying logarithmically from 2 to 4096. The three SCA schemes used are as follows:

1. Scheme 1 ($P \ll K$): The number of processors used is much smaller than the number of random walks to be carried out. The computations have been performed for P=2,4,8,.....,256,512.

2. Scheme 2 ($K/P \sim 10$): The number of processors used is about one tenth of the number of random walks to be carried out. Computations are performed for P=256,512,1024,2048.
3. Scheme 3 ($P = K$): The number of processors used are equal to the number of random walks to be carried out (P=4096).

The formulae used for the prediction of the upper and lower bounds of the speedup for each of the three SCA schemes, given in [1], are listed in Table 5.1. The sample mean and the standard deviation of the duration of the particle histories obtained from the runs of COM on the IBM 3081 were substituted in these formulae.

Figure 5.1.a, shows the upper and lower bounds on the speedup expected from executing the COM program on a multiprocessor machine, on a log-log scale. These values are also listed in Table 5.2. Until about P=128 the two bounds are almost the same. The difference in the upper and lower bounds increases rapidly as the number of processors increase above 128. This is because in calculating the Upper bound of speedup, overheads due to process communication and synchronization are not taken into consideration.

TABLE 5.1

Theoretical Speedups for Different Computation Schemes [1].

SCHEME	EXPECTED SPEEDUP
SCA Scheme 1 $P \ll K$	$\dfrac{P}{1+\left[(\sigma/\mu)/2\sqrt{k}\right]} \leq S(P,K) \leq P$
SCA Scheme 2 $K/P \sim 10$	$\dfrac{P}{1+\dfrac{\sqrt{P}\cdot(P-1)}{\sqrt{(2P-1)K}}} \leq S(P,K) \leq P$
SCA Scheme 3 $P = K$	$\dfrac{K}{1+\dfrac{(K-1)(\sigma/\mu)}{\sqrt{2K-1}}} \leq S(K,K) \leq K$
DCA Scheme 1	$S(P,K) \cong \dfrac{P}{1+\dfrac{(P-1)(\mu^2-\sigma^2)}{2K\mu^2}}$.
DCA Scheme 2	$\dfrac{P}{a+\dfrac{(P-1)P(\sigma/\mu)}{\sqrt{2P-1}}} \leq S(P,K) \leq a$ where $a = \dfrac{P}{1+\dfrac{(P-1)(\mu^2-\sigma^2)}{2K\mu^2}}$

K=desired number of random walks (primary estimates)
μ=mean of independent identically distributed (i.i.d.) random walk times
σ=standard deviation of i.i.d. random walk times
P=number of processors
S(P,K)= Expected speedup

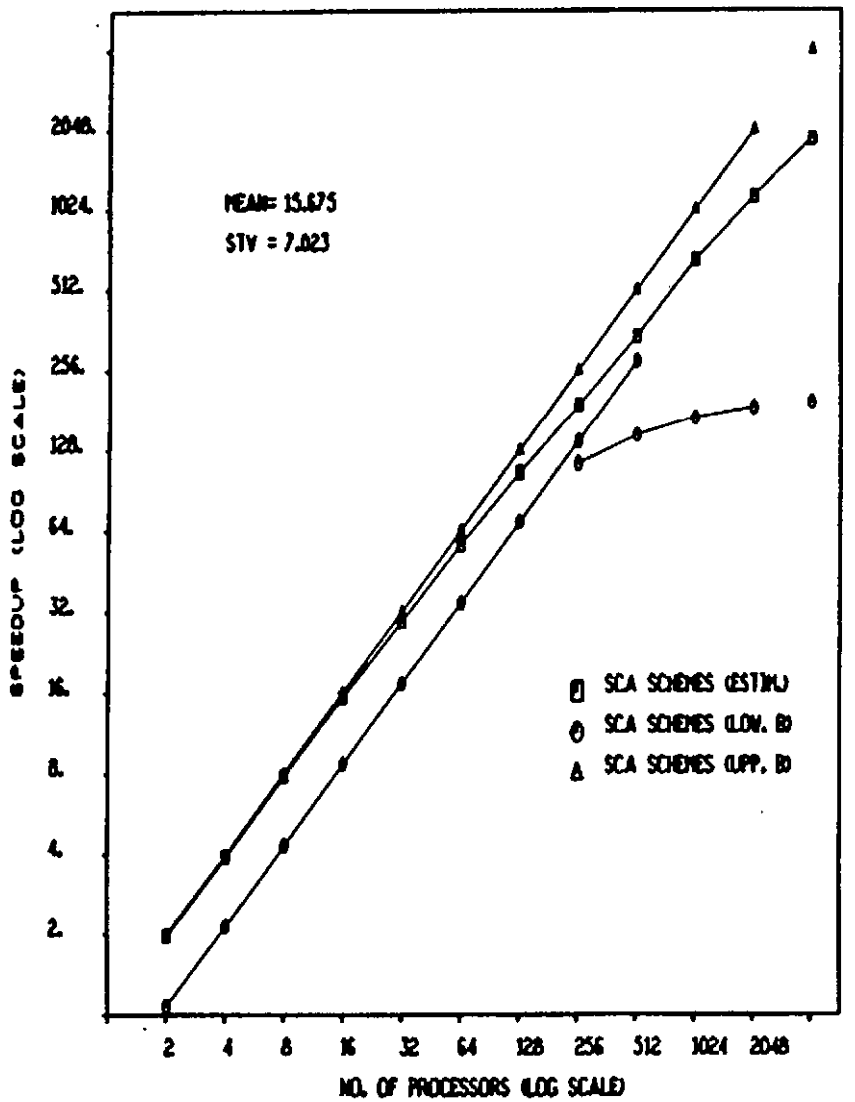


FIG. 5.3.A: Predicted and Estimated speedup of COM on Multiprocessors for SCA Schemes.

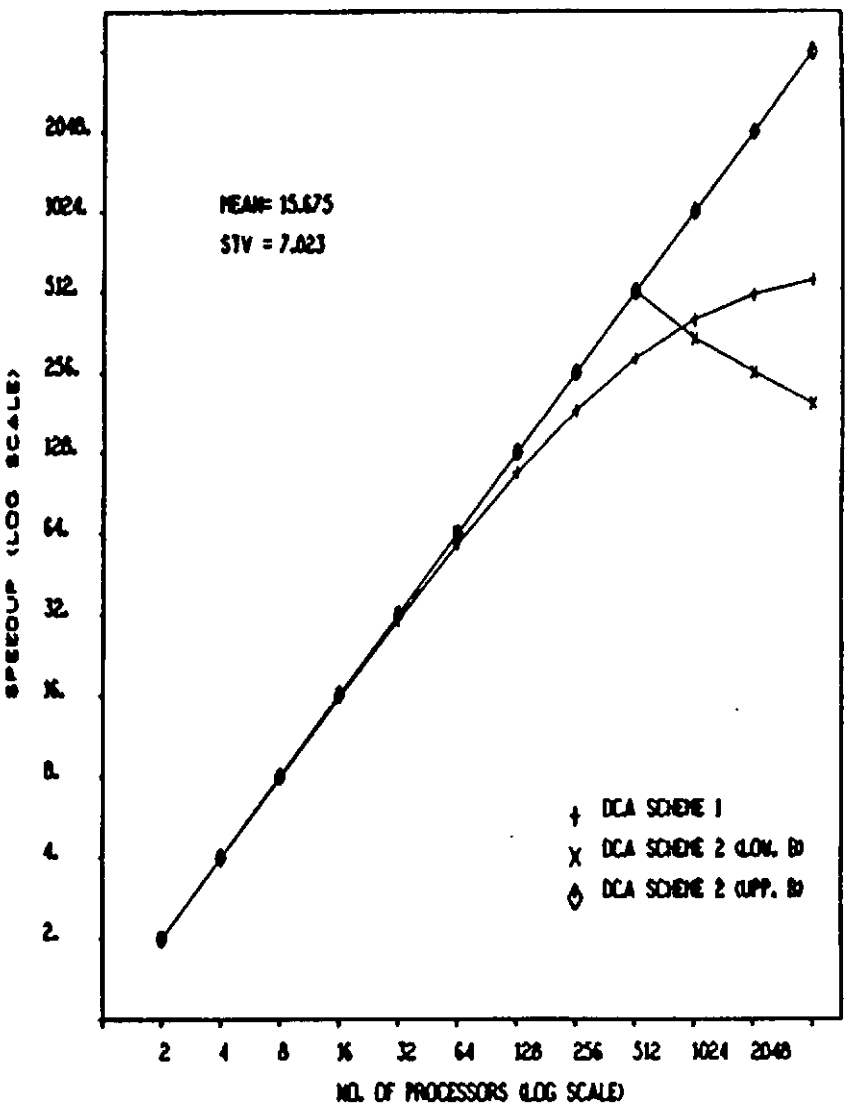


FIG. 5.3.B: Predicted speedup of COM on Multiprocessors for DCA Schemes.

TABLE 5.2

Speedup for COM on multiprocessors for SCA schemes

NUMBER OF PROCESSORS (P)	SPEEDUP		
	ESTIMATED	PREDICTED	
		LOWER BOUND	UPPER BOUND
S C A SCHEME # 1 (K << P)			
2	1.975	1.075	2.0
4	3.912	2.151	4.0
8	7.789	4.301	8.0
16	15.236	8.602	16.0
32	29.451	17.204	32.0
64	56.666	34.409	64.0
128	104.565	68.817	128.0
256	188.277	137.635	256.0
512	341.505	275.269	512.0
S C A SCHEME # 2 (K/P 10)			
256	188.279	113.098	256.0
512	341.505	145.011	512.0
1024	661.886	168.830	1024.0
2048	1146.482	183.937	2048.0
S C A SCHEME # 3 (K = P)			
4096	1888.323	192.552	4096.0

Note: K = number of particle histories.

5.2.2 Performance of DCA schemes

Two basic dynamic computation assignment schemes are considered:

1. Scheme 1: Can be used in the development of parallel algorithms employing any number of processors. The

computational results are obtained for P=2,4,8...,4096.

2. Scheme 2: Designed for parallel algorithms using less than K processors. The computational results are obtained for P=2,4,8...,4096.

The expected speedup from using the two DCA schemes is computed using the formulae shown in Table 5.1, by substituting the sample mean and standard deviation of the duration of particle histories. The expected speedup versus the number of processors used are plotted in Figure 5.1.b; the corresponding values are listed in Table 5.3.

TABLE 5.3

Speedup of COM on multiprocessors for DCA schemes

NUMBER OF PROCESSORS (P)	SPEEDUP		
	DCA SCHEME # 1	DCA SCHEME # 2	
		LOWER BOUND	UPPER BOUND
2	1.997	2.000	2.000
4	3.982	4.000	4.000
8	7.915	8.000	8.000
16	15.641	16.000	16.000
32	30.552	32.000	32.000
64	58.376	64.000	64.000
128	107.185	128.000	128.000
256	184.176	256.000	256.000
512	287.402	512.000	512.000
1024	399.302	341.333	1024.000
2048	445.825	256.000	2048.000
4096	583.993	195.048	4096.000

The following can be observed from this figure:

1. For DCA scheme 1, until P=128 the speedup is linear (S(P,K)=P). For higher values of P the slope of the speedup decreases.
2. For DCA scheme 2, the upper bound is the same as in DCA scheme 1. Until about P=512, the lower bound follows exactly a linear speedup of S(P,K)=P. After P=512, the slope of the curve falls rapidly.

5.3 ESTIMATION OF PERFORMANCE THROUGH SIMULATION

During the runs of the COM on the IBM 3081, the duration of the random walks were recorded for estimating the performance of the parallel algorithm. This is possible because the characteristics of random processes operating in parallel can be derived by successive observations on a single process, due to the ergodicity principle [1].

In SCA schemes, the K particle histories are split between the P processors. Thus M=(K/P) histories are assigned to each processor; note M is an integer. Then the following steps can be used to estimate the performance of SCA parallel algorithms:

1. STEP 1: Divide the set of samples (length of random walks) into P disjoint sets with only M successive samples in each set.

2. STEP 2: Estimate the execution time for the parallel algorithm by finding the maximum of the sums of the durations of random walks for each of the P sets.
3. STEP 3: Estimate the speedup to be equal to the sequential computer execution time (which is given by the sum of all K random walk lengths) divided by the maximum obtained in STEP 2.

The speedup estimated using the procedure described above is shown in Figure 5.1.a. and in Table 5.2. From this Figure the following can be observed:

1. Until about P=128, the estimated speedup lies between the upper and lower bounds as expected, and is very close to the upper bound. This result is expected because the fewer processors we have the less overhead there is and thus speedup is almost proportional to the number of processors used.
2. As the number of processors increases above 128, the estimated speedup droops down from the upper bound, still remaining above the lower bound. As expected, the slope of the speedup decreases as the number of processors increases.

In general, it can be said that the estimated speedup is almost linear. This result can be attributed to the small variance of the lengths of random walks.

The performance of DCA schemes cannot be estimated through simulation. The requirements for estimating the execution time of a sequential algorithm and the corresponding DCA algorithm are incompatible, because in a DCA algorithm some of the histories can get aborted and the set of histories executed on each processor may not be successive [1].

5.4 A PARALLEL ALGORITHM FOR COM

For the development of the parallel algorithm, the execution of K particle histories (random walks) is considered on a multiprocessor system with P processors. In this algorithm, the random walks are assigned dynamically to available processors. The sequence of operations is as follows. P processes are initiated in parallel each running on a different processor. Each of the processes executes a random walk. Whenever a random walk is completed, another is initiated on that processor provided that the K random walks have not already been initiated. When any one of the processes detects that the K random walks have been completed, it immediately terminates all the running processes including itself.

A parallel algorithm for COM using DCA scheme 2 is given in Figure 5.2 Synchronization between processes is done using Critical Regions [5]. When a process is in the Critical Region no other process can access it. In this algorithm a Critical Region is indicated as follows:

REGION varname: operation on varname

This means that "varname" is a shared variable between all the processes running in parallel. In order to prevent more than one process from accessing the variable at the same time, the Critical Region is introduced. Five shared variables giving rise to five Critical Regions are used in the algorithm namely, "escpr" which represents the escape probability, "therm" which is a counter for thermalized particles, "esc" which is a counter for escaped particles, "thpr" which is the thermalization probability and "n" which keeps track of the number of completed random walks.

Figure 5.2 comprises a main process that initiates P processes in parallel; the COBEGIN/COEND construct is used [5]. After the K random walks are completed, the required statistical estimates (solutions) are obtained from the statistics accumulated by different processes.

```
read input parameters
select geometry and introduce biasing
COBEGIN process1;process2; . . .processp COEND
calculate statistical estimates
output results
END
```

A) MAIN PROCESS

```
PROCESS i
initializa (k,escpr,therm,esc,thpr)
REPEAT
   get a new source particle
   WHILE (particle not escaped or thermalized) THEN DO
      IF (thermalized) THEN DO
         calculate escape probability as ep
         REGION escpr: escpr=escpr+ep
         REGION therm: therm=therm+1
         END DO
      ELSE DO
         determine distance traveled by particle
         propagate the particle
         IF (particle escaped) THEN
           REGION esc: esc=esc+1
         ELSE DO
           determine new energy
           calculate thermalization probability as tp
           REGION thpr: thpr=thpr+tp
           compute new directional cosines
         END DO
      END DO
      REGION n: DO
         n=n+1
         IF (n=k) THEN terminate all processes
         END DO
   END DO
FOREVER
```

B) PROCESS i

Fig. 5.2: A Parallel algorithm for COM using DCA scheme 2

5.5 SUMMARY AND CONCLUSIONS

The upper and lower bounds of the speedup expected from the implementation of the COM program on multiprocessors are predicted using exising theoretical formulae. The performance of COM on multiprocessors is estimated through simulation on the IBM 3081 computer. The estimated speedup is almost linear and lies between the upper and lower bounds of the predicted speedup. The linear speedup is due to the relatively small variance of the random walk lengths. Prediction of the performance of COM using DCA scheme 1, shows that the speedup is almost equal to the number of processors used for upto about 128 processors; subsequently the speedup becomes progressively smaller compared to the number of processors as the later increases.

A parallel algorithm for the implementation of COM on multiprocessors is proposed. Since random walks are independent, synchronization between processes is needed only for the accumulation of overall statistics. This algorithm has not been implemented on a commercial multiprocessor because of lack of access to such machine.

Chapter 6

CONCLUSION

6.1 OVERVIEW OF PRESENT WORK

The Monte Carlo method, although time consuming and expensive, provides a powerful tool for radiation transport analysis in Nuclear Engineering problems. The development of supercomputers with vector and multiprocessor architectures offers large computational power. With computer execution time as the main drawback of the Monte Carlo method, it is natural to consider its implementation on supercomputers. The adaptation of a Monte Carlo radiation transport code on such computers has been considered in this project.

The architectural and programming aspects of two supercomputers, namely the Cyber-205 vector computer and the HEP multiprocessor, have been studied.

Previous work done on parallel implementation of Monte Carlo codes has been reviewed; the implementation aspects on vector and multiprocessor supercomputers have been examined. It is noted that global restructuring of the Monte Carlo algorithm is needed for efficient vectorization. For the

development of an algorithm for multiprocessor machines, the additional coding is required for process syncronization and communication.

A vectorized algorithm for a special purpose Monte Carlo code, the Center-of-Mass code, has been developed and succesfully implemented on the Cyber-205 2-pipe vector computer. The speedup obtained over the Cyber-205 scalar processor is of the order of 14 to 16, while the speedup over the IBM 3081 is of the order of 32 to 40. A number of experiments have been carried out in order to study the performance of the vectorized algorithm. It has been found that the speedup is largely dependent on the average vector length. The length of the random walks in the code does not affect the speedup very much. The speedups obtained in this work prove that efforts in vectorizing Monte Carlo codes are worth while and that the vectorization of more general Monte Carlo codes is feasible.

The performance of COM on multiprocessors is predicted using the theoretical results given in [1]. The predicted speedup, using DCA scheme 1, is almost equal to the number of processors used for upto about 128 processors; subsequently the speedup becomes progressively smaller compared to the number of processors as the later increases. The performance of COM on multiprocessors is also estimated by simulation on the IBM 3081. The estimated speedup is

almost linear; it lies between the predicted upper and lower bounds of speedup. A parallel algorithm has been proposed for the implementation of COM on multiprocessor machines. Implementation of this algorithm on a commercial multiprocessor has not been possible due to the inaccessibility of such a machine.

6.2 FUTURE WORK

Future work may include:

- The implementation of the parallel algorithm for the Center-of-Mass code proposed in this project on multiprocessor supercomputers.

- The adaptation of the COM code on the CRAY-1 vector computer and comparison of its performance with that obtained on the CYBER-205.

- The development of other vectorization strategies to improve the efficiency of the vector algorithm.

- The development of a Mathematical model for the prediction of the performance of vector Monte Carlo algorithms, based on statistics collected from the scalar codes.

- The implementation of a general purpose Monte Carlo code, like the MORSE [31] code, on multiprocessors using the

guidelines developed in this work. Appendix VI discusses some additional problems that one may encountered in dealing with the vectorization of the MORSE code.

APPENDICES

Appendix I

HOW TO RUN PROGRAMS ON THE CYBER-205 COMPUTER

I.1 SIGN ON PROCEDURE

The Cyber-205 is a back-end computer. In order to access it, one has to first sign on to the front end machine, to which the Cyber-205 is connected. To sign on to the system from a remote computer, access to a communication network and to the proper communication software is required. At present, the only way of communication between UNB and the University of Calgary (where the Cyber-205 is located and connected via the CDC Cyber-175 front-end computer), is through DATAPAC, using the PC-TALK [6] software package on an IBM-PC.

An IBM personal computer and a MODEM (modulator-demodulator) should be available. The sign on procedure through DATAPAC is as follows:

1. Run PC-TALK and set proper communication parameters according to the specifications of the MODEM, and the PC-TALK software (see PC-TALK user's Guide [6]). For the present MODEM the baud rate has to be set to 300; all other parameters are preset.

2. Dial the local DATAPAC number through the dialing directory of PC-TALK, or for manual dialing enter:

   ATDT4549462

3. After the response from DATAPAC enter a dot (.).

4. Enter the phone number of the target computer.

5. Sign on to the computer using the account number and the password provided.

I.2 NOS CONTROL LANGUAGE

After sign on, the program has to be submitted using the following control cards:

```
/JOB
/NOSEQ
ADEY,locid.
USER,U=userno,AC=acctn.
RESOURCE,TL=100,JCAT=LPRIME.
FTN200,SC.
LOAD,LIB=F200LIB,CDF=450.
GO.
/EOR
      PROGRAM name(INPUT,OUTPUT)
      *
      * The program goes here.
      *
/EOR
  *
  *  Input data
  *
/EOF
```

where,

locid = local identifier for the Cyber-205
userno = user number
acctn = account number
EOR = end of read mark
EOF = end of file mark

The program is then queued for execution on the Cyber-205. After execution is completed, the output can be retrieved using the following command:

QGET,jobname,PR

For more information on the NOS-software system see references [27,28].

Appendix II

THE COM PROGRAM

```
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                        C   O    M                                  C
C                        ----------                                  C
C  COM IS A CENTER-OF-MASS MONTE CARLO PROGRAM THAT SIMULATES        C
C  THE SCATTERING OF FAST NEUTRONS INCIDENT ON A WATER-VAPOR         C
C  TWO-PHASE FLOW IN A PIPE.  THE PROGRAM IS USED TO AID IN          C
C  DESIGNING NEUTRON DENSITOMETERS FOR USE IN TWO PHASE FLOW         C
C  SYSTEMS.  IT ASSUMES ISOTROPIC SCATTERING IN THE CENTER-          C
C  OF-MASS SYSTEM AND UTILIZES FUNCTIONAL FITS FOR NEUTRON           C
C  CROSS-SECTIONS.  SEVEN DIFFERENT FLOW REGIME GEOMETRIES           C
C  ARE TREATED USING SIMPLE ANALYTICAL GEOMETRY METHODS.             C
C                                                                    C
C  WRITTEN BY    : DR. E. HUSSEIN                                    C
C                  CHEMICAL ENG. DEPT., UNB                          C
C  DATE WRITTEN  : MAY, 1985.                                        C
C  MODIFIED BY   : T. TASSOU                                         C
C                  SCHOOL OF COMPUTER SC., UNB                       C
C  DATE MODIFIED: AUGUST, 1985.                                      C
C                                                                    C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C
      CHARACTER*3 JEOM(7)
      DIMENSION TITLE(20)
      DOUBLE PRECISION X(3),C(3),XI(3),CI(3)
      DOUBLE PRECISION E1,PX,PX2,PESCP2,PESCP,PRPX,PRPX2
      DOUBLE PRECISION RNR,RNORM,TH,TH1,TH2,CTH2,XRX,ESCP,ESCP2
      DOUBLE PRECISION RHO,R,ZB,THICK,EMIN,E01,XD(3),RR,RHOW,RW
      COMMON/INFORM/ RHO,R,ZB,THICK,EMIN,E01,XD,RR,RHOW,RW
      COMMON/DFORM/NSTEP,ICCT,IDENS,IGEOM
      COMMON/RANGEN/ KSEED
      DATA CI/0.0,-1.0,0.0/
      DATA JEOM/'HOM','IAN','ANN','STD','STU','STL','STR'/
C
C HOMOGENIZED FLOW IS ASSUMED IN THIS VERSION
C TITLE IS THE JOB TITLE
C KSEED=SEED FOR RANDOM NUMBER GENERATOR
C RHO = LIQUID DENSITY
C R   = PIPE RADIUS
C ZB  = PIPE HEIGHT
C THICK= PIPE THICKNESS
C E1  = PARTICLE ENERGY
C EMIN= CUT OFF ENERGY
C NEV = PARTICLE HISTORIES
C IT  = MAXIMUM TIME IN CPUS
C IGEOM= KIND OF GEOMETRY (SEE FIGURE II.1 AT END OF LISTING)
C XD = DETECTOR POSITION
C (0,0,0) IS CENTER OF CYLINDER
C X DIRECTION IS TO BOTTOM OF PAGE
C Y DIRECTION IS TO L.H.S OF PAGE
C
      READ(5,56) TITLE
      READ(5,*) KSEED
      READ(5,4) RHO,R,ZB,THICK
      READ(5,5) E1,EMIN
      READ(5,6) NEV,IT,IGEOM
```

```
      READ(5,4) (XD(I),I=1,3)
      READ(5,*)IDENS
C
      WRITE(6,5007) KSEED
      IF(IGEOM.LE.0) IGEOM=1
      E01=E1
C
      NSS=0
      ESCP=0.0
      ESCP2=0.0
      NESCP=0
      NTH=0
      IN=0
      PX=0.0
      PX2=0.0
C
C NSS=NO. OF EPITHERMAL COLLISIONS
C NESCP=NO. OF PARTICLES ESCAPED
C NTH=NO. OF PARTICLES REACHED THERMAL ENERGY
C IN=PARTICLE NUMBER
C
C START A RANDOM WALK
C A BEAM INCIDENT ON THE MIDDLE OF THE CYLINDER IS ASSUMED
C
C ********************** MODIFIED PART *************
      GO TO (11,12,13,14,15,16,16),IGEOM
C HOMOGENEOUS
11    RHOW=RHO
      RW=R
      RR=0.0
      RNR=R
      RNORM=1.
      GO TO 88
C ***************INVERTED ANNULAR *******
12    RHOW=1.0
      RW=DSQRT(RHO)*R
      RNORM=R/RW
      RR=R
      RNR=RW
      GO TO 88
C ************** ANNULAR ***************************
13    RHOW=1.0
      RW=R
      RR=DSQRT(1.-RHO)*R
      RNR=RW
      RNORM=1.0
      GO TO 88
C ************** STRATIFIED ***********************
14    RHOW=1.0
      RW=R
      RNR=R
      TH=4.*RHO*ASIN(1.)
      TH1=TH
19    TH2=TH+DSIN(TH1)
      CTH2=DCOS(TH2)
      TH1=(TH2-TH1*CTH2)/(1.-CTH2)
      IF(DABS(TH1-TH2).GT.1.E-06) GO TO 19
      RR=R*DCOS(TH1/2.)
      XRX=R-RR
      RMR=RR
      RNORM=2*R/XRX
      GO TO 88
15    RHOW=1.0
      RW=R
      TH=4.*RHO*ASIN(1.)
      TH1=TH
```

```
29      TH2=TH+DSIN(TH1)
        CTH2=DCOS(TH2)
        TH1=(TH2-TH1*CTH2)/(1.-CTH2)
        IF(DABS(TH1-TH2).GT.1.E-06) GO TO 29
        RR=R*DCOS(TH1/2.)
        XRX=-(R+RR)
        RMR=RR
        RNR=R
        RNORM=DABS(2*R/XRX)
        GO TO 88
16      RHOW=1.0
        RW=R
        TH=4.*RHO*ASIN(1.)
        TH1=TH
30      TH2=TH+DSIN(TH1)
        CTH2=DCOS(TH2)
        TH1=(TH2-TH1*CTH2)/(1.-CTH2)
        IF(DABS(TH1-TH2).GT.1.E-06) GO TO 30
        RR=R*DCOS(TH1/2.)
        RNR=R
        RNORM=1.0
        IF(IGEOM.EQ.7)RR=-RR
88      CONTINUE
C
C************ START OF RANDOM WALKS ******************
C
         CALL CPUTIM(ISS)
         DO 103 IN=1,NEV
              RNM1=UNIRAN(Y)
              IF(IGEOM.EQ.4.OR.IGEOM.EQ.5)THEN
                 X(1)=XRX*(RNM1)+RMR
              ELSE
                 X(1)=RNR*(-1.+2.*RNM1)
              ENDIF
              X(2)=DSQRT(RNR*RNR-X(1)*X(1))
              X(3)=0.0
              E1=EO1
              C(1)=CI(1)
              C(2)=CI(2)
              C(3)=CI(3)
C +++++++++ FOLLOW NEUTRON
              NS=0
              NSTEP=0
              CALL FOLNUT(X,C,E1,NS,NESCP,NTH,PTH,ESCP,ESCP2)
              NSS=NSS+NS
C +++++++++ STATITICAL PARAMETERS
              PX=PX+PTH
              PX2=PX2+PTH*PTH
 103    CONTINUE
        CALL CPUTIM(ISS)
        RJ=FLOAT(ISS)/10000.
              IF(NTH.EQ.0) GO TO 400
              XNEV=DFLOAT(NEV)*RNORM
              PRPX=PX/XNEV
              PRPX2=(PX2/XNEV-PRPX**2)/(XNEV-1.)
              IF(PRPX.NE.0.0) PRPX2=DSQRT(PRPX2)/PRPX
              XNET=DFLOAT(NTH)
              PESCP=ESCP/XNET
              PESCP2=(ESCP2/XNET-PESCP**2)/(XNET-1.)
              IF(PESCP.NE.0.0) PESCP2=DSQRT(PESCP2)/PESCP
              IN=RNORM*NTOTAL
C
  400         WRITE(5,58) TITLE
              WRITE(6,*)'DENSITY OF LIQUID=',IDENS
              WRITE(6,3005) R,THICK,ZB
              WRITE(6,350)XD
```

```
            WRITE(6,57) RHO,JEOM(IQEOM)
            WRITE(6,300) NEV,E01,CI
            WRITE(6,3781)
            WRITE(6,3782) NEV,NESCP,NTH,EMIN
            WRITE(6,3780) NSS
            RESP=PRPX*PESCP
            RESP2=SQRT(PRPX2**2+PESCP2**2)
            WRITE(6,5780) PRPX,PRPX2,PESCP,PESCP2,RESP,RESP2
C DIRECT ESTIMATOR
            IF(XNEV.GT.0)THEN
               PRPX=XNET/XNEV
               PRESP=PRPX*PESCP
               WRITE(6,7780) PRPX,PESCP,PRESP
               WRITE(6,5008)
               WRITE(6,*) KSEED
            ENDIF
            PRINT 3778,RJ
C
C************** FORMAT STATETMENTS ****************************
  300   FORMAT(1HO/I7,'  NEUTRONS OF ENERGY OF  ',E10.4,
     1 /,'  INCIDENT WITH    ',3(1X,F7.4),
     2   '   DIRCTIONAL COSINES ',/)
 3780   FORMAT(' NUMBER OF EPITHERMAL COLLISIONS = ',I10)
 3781   FORMAT(//,' SUMMARY OF RANDOM WALK')
 3782   FORMAT(2X,I5,' PARTICLES IN TOTAL ',/,
     1         2X,I5,' ESCAPED ',/,
     2 2X,I5,' REACHED ',E10.4,' MEV ENERGY CUTOFF',/)
 5780        FORMAT(' THERM. PROB. = ',E10.4,'(',E10.4,')'
     1,/,            ' ESCP.  PROB. = ',E10.4,'(',E10.4,')'
     2,/,            ' THERM. RESP. = ',E10.4,'(',E10.4,')')
 7780       FORMAT('  -DIRECT ESTIMATOR - ',
     1 '  THERM. PROB. = ',E10.4,/,
     2 '  ESCP.  PROB. = ',E10.4,/,
     3 '  THERM.  RESP. = ',E10.4,/)
 5008    FORMAT(//,' LAST SEED USED =')
   57    FORMAT(' LIQUID FRACTION = ',F7.3,'  (',A3,')')
   58          FORMAT(1X,20A4)
 3005   FORMAT(' CYLINDRICAL PIPE OF RADIUS = ',F8.3,/
     1         ,'  THICKNESS = ',F8.3, '  (CM)'
     2         ,' ,AND HEIGHT= ',F8.3, '  (CM)')
 2910 FORMAT(' CPU TIME OF ',I5,' EXCEEDS MAX. ALLOWED OF ',I5,//)
    6     FORMAT(3I5)
    5     FORMAT(2E10.4)
    4     FORMAT(10E11.4)
 5007    FORMAT(' SEED FOR RANDOM NUMBER GENERATOR = ',I12,/)
   56   FORMAT(20A4)
 3778    FORMAT(' RANDOM WALK TIME = ',R8.5,'SECONDS')
  350    FORMAT(' DISTANCE TO DETECTOR =',3(2X,E11.4))
C*************************************************************
      STOP
      END
```

```
C*****************************************************
C*                    F O L N U T                    **
C*****************************************************
C
      SUBROUTINE FOLNUT(X,C,E1,NS,NESCP,NTH,PTHT,ESCP,ESCP2)
C
      DOUBLE PRECISION X(3),XI(3),C(3),CN(3),W,PTHT,E1,ATOM,E
      DOUBLE PRECISION SSOX,SAOX,PATH,D,X1,Y1,X2,Y2,D1,D2
      DOUBLE PRECISION DD,CSCM,RMX,RM2,RM1,CSL,E2,PTH,PHI,ESCP2
      DOUBLE PRECISION DIST2,ARG,ARGW,SIGTH,SIGW,ESCP
      DOUBLE PRECISION RHO,R,ZB,THICK,EMIN,EO1,XD(3),RR,RHOW,RW
      COMMON/INFORM/RHO,R,ZB,THICK,EMIN,EO1,XD,RR,RHOW,RW
      COMMON/DFORM/NSTEP,ICCT,IDENS,IGEOM
      DATA RM1/939.55/,RM2/14897.61/,ATOM/0.03346/
      SSH(E)=(17.6023/(1.+7.417*E+0.1105*E**2)
     * +2.71811/(1.+0.2427*E+0.0028*E**2))+2.*ATOM*RHOW
       DATA ICALL/0/
       DATA   SIGTH,SIGW /1.512,1.155/
C      DATA   SIGTH,SIGW /3.452,1.155/
C
      W=1.0
      PTHT=0.0
2000  CONTINUE
      IF(E1.LE.EMIN) GO TO 200
      NSTEP=NSTEP+1
C  FOLNUT IS CALLED ONCE PER INCIDENT NEUTRON TO FOLLOW THE
C  NEUTRON THROUGH THE PIPE
C
      DO 100 I=1,3
100      XI(I)=X(I)
      NS=NS+1
      CALL OXY(RHOW,E1,SSOX,SAOX)
      PATH=1./((SSH(E1)+SSOX+SAOX)*IDENS)
C PROPAGATE NEUTRON A DISTANCE D
      D=-PATH*ALOG(UNIRAN(Y))
      DO 61 I=1,3
61       X(I)=X(I)+D*C(I)
C NEUTRON ESCAPED ?
      IF((X(1)**2+X(2)**2).GE.RW*RW) GO TO 300
      IF(ABS(X(3)).GE.ZB/2.) GO TO 300
      GO TO (50,50,23,24,25,26,27),IGEOM
C ANNULAR
23    CALL INTERS(XI,X,RR,X1,Y1,X2,Y2 ,INTER)
      IF(INTER.LE.0) GO TO 50
      IF((X(1)**2+X(2)**2).LE.RR**2) GO TO 101
      D1=(XI(1)-X1)**2+(XI(2)-Y1)**2
      D2=(XI(1)-X2)**2+(XI(2)-Y2)**2
      IF(D1.LE.D2) GO TO 50
  101 DD=1.-C(3)**2
      DD=2.*SQRT(RR**2*DD-(C(1)*X(2)-C(2)*X(1))**2)/DD
      DO 102 I=1,3
  102 X(I)=X(I)+DD*C(I)
      IF((X(1)**2+X(2)**2).GE.RW*RW) GO TO 300
      IF(ABS(X(3)).GE.ZB/2.) GO TO 300
      GO TO 50
C STRATIFIED DOWN
   24 IF(X(1).LE.RR) GO TO 300
      GO TO 50
C STRATIFIED UP
   25 IF(X(1).GT.RR) GO TO 300
      GO TO 50
C STRATIFIED LEFT
   26 IF(X(2).LE.RR) GO TO 300
      GO TO 50
C STRATIFIED RIGHT
   27 IF(X(2).GE.RR) GO TO 300
```

```
   50 CONTINUE
      CSCM=-1.+2.*UNIRAN(Y)
      IF((SSOX+SAOX)*PATH.GT.UNIRAN(Y)) GO TO 5
C COLLISION WITH HYDROGEN
      RMX=RM1
      GO TO 6
C COLLISION WITH O
    5 RMX=RM2
      W=W*SSOX/(SSOX+SAOX)
    6 CALL KINNER(RM1,RMX,RM1,RMX,E1,CSCM,CSL,E2)
      IF(E2.LE.EMIN) GO TO 58
      IF(RMX.NE.RM1) GO TO 54
C THERMALIZATION PROBABILTY
      PTH=1.-(E1-EMIN)/E1
      GO TO 60
   54 IF(49./81.*E1.GT.EMIN) GO TO 59
      PTH=1.-(E1-EMIN)/(E1-49./81.*E1)
      GO TO 60
   58 PTH=1.0
      GO TO 60
   59 PTH=0.0
   60 PTHT=PTH*W+PTHT
C KINNER IS KINEMATICS ROUTINE IN C.O.M. SYSTEM
C ISOTROPIC SCATT. IN LAB IS ASSUMED FOR THE AZIMUTHAL DIRECTION
      PHI=6.2832*UNIRAN(Y)
C GET NEW DIRECTIONAL COSINES
      CALL SCATT(C,CSL,PHI,CN)
      E1=E2
      DO 99 I=1,3
   99 C(I)=CN(I)
      GO TO 2000
C THRMALIZED
  200 NTH=NTH+1
C CALCUALTE ESCAPE PROB. (TO DETECTOR)
      DIST2=(X(1)-XD(1))**2+(X(2)-XD(2))**2+(X(3)-XD(3))**2
      CALL OUT(XD,X,DIST2,IGEOM,RW,RR,ARG,R+THICK,ARGW)
      ARG=SIGTH*RHOW*ARG+SIGW*ARGW
      IF(ARG.LE.10.) ICCT=ICCT+1
      ARG=EXP(-ARG)/(12.56637*DIST2)*W
      ESCP=ESCP+ARG
      ESCP2=ESCP2+ARG**2
      GO TO 1092
C ESCAPED
  300 NS=NS-1
      NESCP=NESCP+1
1092  CONTINUE
      RETURN
      END
```

```
C*************************************************
C**                K I N N E R                  **
C*************************************************
C** RELATIVISTIC KINEMATICS ROUTINE IN C-O-M ***
C*************************************************
C
      SUBROUTINE KINNER(EM1,EM2,EM3,EM4,T1,CSCM,CSL,T3)
C
      DOUBLE PRECISION EM1,EM2,EM3,EM4,T1,CSCM,CSL,T3,P1,W1,WCM
      DOUBLE PRECISION PCM,BO,GO
C
      P1=SQRT(T1*(T1+2.*EM1))
      W1=T1+EM1
      WCM=.5*(EM1*EM1+EM2*EM2+EM3*EM3-EM4*EM4+2.*EM2*W1)
      IF(WCM.LE.0.0) GO TO 10
      WCM=WCM/SQRT(EM1*EM1+EM2*EM2+2.*EM2*W1)
      IF(WCM.LE.EM3) GO TO 10
      PCM=SQRT(WCM*WCM-EM3*EM3)
      BO=P1/(W1+EM2)
      GO=1./SQRT(1.-BO*BO)
      T3=GO*(WCM+BO*PCM*CSCM)-EM3
      IF(T3.LT.0.025E-06)  T3=0.025E-06
      CSL=GO*(PCM*CSCM+BO*WCM)/SQRT(T3*(T3+2.*EM3))
      IF(CSL.GT.1.) CSL=1.0
      RETURN
 10   T3=0.0
      CSL=0.0
      RETURN
      END
C
C*************************************************
C**                S C A T T                    **
C*************************************************
C**    CALCULATES NEW DIRECTIONAL COSINES       **
C*************************************************
C
      SUBROUTINE SCATT(C,COSTH,PHI,CNEW)
C
       DOUBLE PRECISION C(3),COSTH,PHI,CNEW(3),X,CT,ST,SP,CP,S,CXP
       DOUBLE PRECISION CYP,CZP
       SI(X)=SQRT(1.-X**2)
C
       CT=C(3)
       IF(CT-1.) 6,4,2
    2    CT=1.
    4    ST=0.
         SP=0.
         CP=1.
         GO TO 30
    6    IF(-CT-1.) 20,4,8
    8    CT=-1.
         GO TO 4
   20    ST=SI(CT)
         CP=C(1)/ST
         IF(CP.GT.1.) CP=1.
         IF(CP.LT.-1.) CP=-1.
         SP=C(2)/ST
         IF(SP.GT.1.) SP=1.
         IF(SP.LT.-1.) SP=-1.
   30    IF(COSTH.GT.1.) COSTH=1.
         IF(COSTH.LT.-1.) COSTH=-1.
         S=SI(COSTH)
         CXP=S*COS(PHI)
         CYP=S*SIN(PHI)
         CZP=COSTH
         CNEW(1)=CXP*CT*CP-CYP*SP+CZP*ST*CP
```

```
          CNEW(2)=CXP*CT*SP+CYP*CP+CZP*SP*ST
          CNEW(3)=-CXP*ST+CZP*CT
        RETURN
        END
C
              FUNCTION UNIRAN(X)
C
C*************************************************************
C*  A MACHINE DEPENDENT UNIFORM RANDOM NUMBER GENERATOR      *
C*************************************************************
C
        COMMON/RANGEN/ KSEED
        DOUBLE PRECISION SEED1,SEED2,DBLE,DMOD
C
         SEED1=DBLE(FLOAT(KSEED))*6.5539D+04
         SEED2=DMOD(SEED1,2.147483648D+09)
         KSEED=IDINT(SEED2)
         IF(KSEED) 1,2,2
    1    KSEED=KSEED+2147483647+1
    2    TEMP=FLOAT(KSEED)
         UNIRAN=TEMP*4.656613E-10
       RETURN
       END
C
C
            SUBROUTINE OXY(RHO,E,SS,SA)
C
C*************************************************************
C**  TO CALCULATE OXYGEN AND HYDROGEN CROSS-SECTIONS.     **
C*************************************************************
        DOUBLE PRECISION RHO,E,SS,SA,SS03,SS0,SA0,ATOM
        DATA SS03/4./,SS0/1.4/,SA0/0.04/,ATOM/0.03346/
C
C  16-O CROSS SECTIONS
            IF(E-3.) 1,2,2
    2       SS=SS0*ATOM*RHO
            SA=SA0*ATOM*RHO
            RETURN
    1       SS=SS03*ATOM*RHO
            SA=0.0
        RETURN
        END
```

```
      SUBROUTINE OUT(XD,X,DIST,IGEOM,R,RR,D,RWL,DWL)
C
C*************************************************************
C* DETERMINES IF THE PARTICLE LIES IN THE PIPE GEOMETRY **
C*************************************************************
C
      DOUBLE PRECISION X(3),XD(3),DIST,R,RR,D,RWL,DWL,F,X1,Y1,X2,Y2
      DOUBLE PRECISION XW1,YW1,XE2,YW2,XO1,YO1,XO2,YO2,X3,Y3,X4,Y4
      DOUBLE PRECISION D31,YL,XL
C
      F=DIST/((X(1)-XD(1))**2+(X(2)-XD(2))**2)
      CALL INTERS(X,XD,R,X1,Y1,X2,Y2,INTER)
      CALL INTERS(X,XD,RWL,XW1,YW1,XW2,YW2,INTER)
      DWL=(X1-XW1)**2+(Y1-YW1)**2
      DWL=SQRT(F*DWL)
      GO TO (1,2,3,4,5,6,6),IGEOM
C HOMOGENEOUS
    1 D=SQRT(F*((X(1)-X1)**2+(X(2)-Y1)**2))
      RETURN
C INVERTED ANNULAR
    2 CALL INTERS(X,XD,RR,XO1,YO1,XO2,YO2,INTER)
      DWL=(XO1-XW1)**2+(YO1-YW1)**2
      DWL=SQRT(DWL*F)
      GO TO 1
C ANNULAR
    3 CALL INTERS(X,XD,RR,X3,Y3,X4,Y4,INTER)
      IF(INTER.LE.0) GO TO 1
      D31=(X3-X1)**2+(Y3-Y1)**2
      D=(X(1)-X1)**2+(X(2)-Y1)**2
      IF(D.LT.D31) GO TO 33
      D=SQRT(D)-SQRT((X3-X4)**2+(Y3-Y4)**2)
      D=D*D
   33 D=SQRT(F*D)
      RETURN
C STRATIFIEED DOWN
    4 GO TO 1
C STRATIFIED UP
    5 YL=(RR-X(1))*(Y1-X(2))/(X1-X(1))+X(2)
      D=(X(1)-RR)**2+(X(2)-YL)**2
      D=SQRT(F*D)
      RETURN
C STRATIFIED (SIDE)
    6 D=(X(1)-X1)**2+(X(2)-Y1)**2
      XL=(RR-X(2))*(X1-X(1))/(Y1-X(2))+X(1)
      IF((XL**2+RR**2).GT.R**2) GO TO 66
      D=(X(1)-XL)**2+(X(2)-RR)**2
   66 D=SQRT(F*D)
      RETURN
      END
```

```
      SUBROUTINE INTERS(X,XD,R,X1,Y1,X2,Y2,INTER)
C
C*************************************************************
C*         DETERMINES POINTS OF INTERSECTIONS              **
C*************************************************************
C
       DOUBLE PRECISION X(3),XD(3),R,X1,Y1,X2,Y2,XM,B,XM2,ROOT,XX,YY
C
      INTER=1
      IF(XD(1).EQ.X(1)) GO TO 10
      XM=(XD(2)-X(2))/(XD(1)-X(1))
      B=X(2)-XM*X(1)
      XM2=XM**2+1.0
      ROOT=XM2*R**2-B**2
      IF(ROOT.LT.0) GO TO 20
      ROOT=SQRT(ROOT)
      X1=(-XM*B+ROOT)/XM2
      X2=(-XM*B-ROOT)/XM2
      Y1=XM*X1+B
      Y2=XM*X2+B
      GO TO 5
   10 ROOT=  R**2-X(2)**2
      IF(ROOT.LT.0) GO TO 20
      X1=SQRT(ROOT)
      X2=-X1
      Y1=X(2)
      Y2=Y1
    5 D1=(XD(1)-X1)**2+(XD(2)-Y1)**2
      D2=(XD(1)-X2)**2+(XD(2)-Y2)**2
      IF(D1.LE.D2) RETURN
      XX=X1
      YY=Y1
      X1=X2
      Y1=Y2
      X2=XX
      Y2=YY
      RETURN
   20 INTER=-1
      RETURN
      END
```

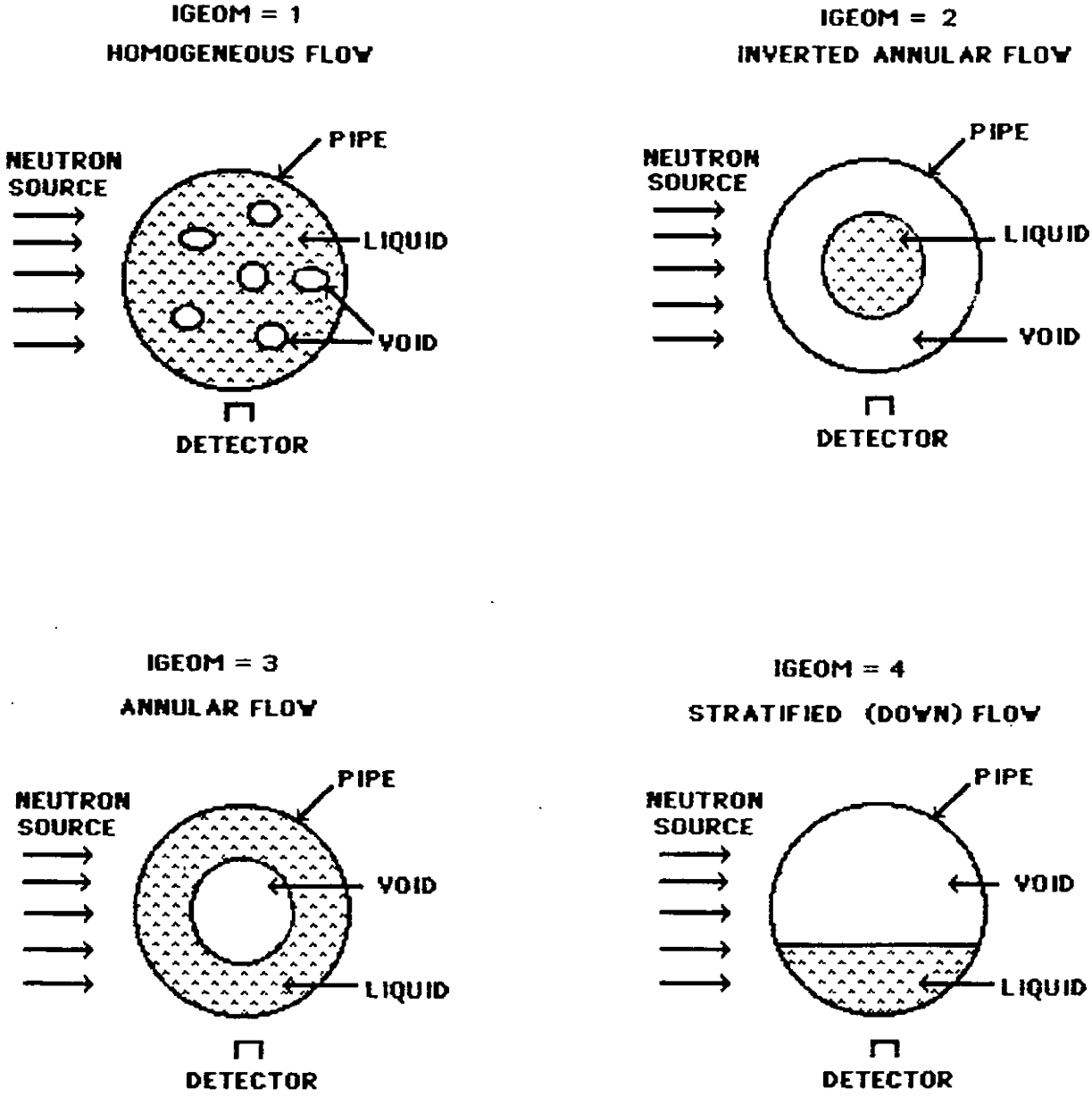
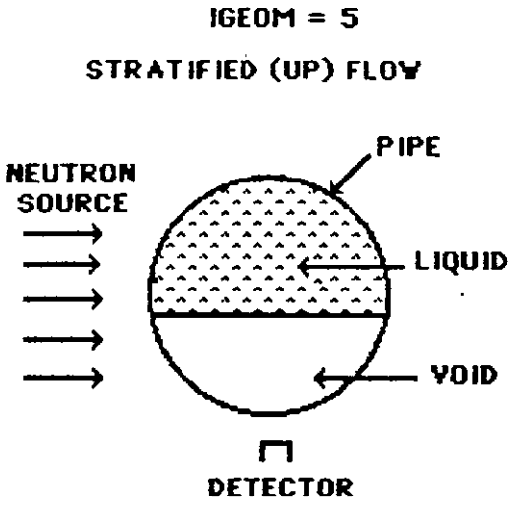
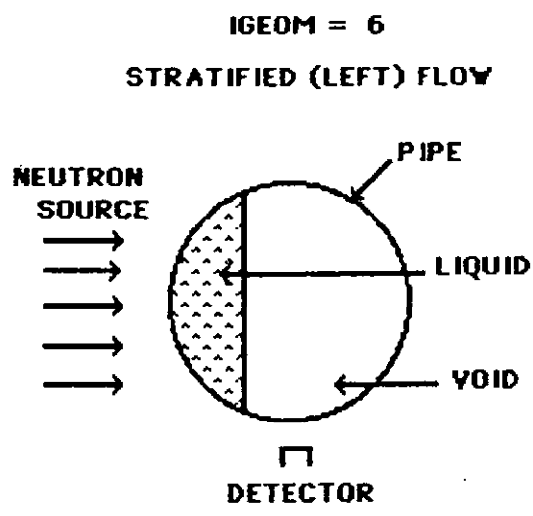


Fig. II.1: Geometries used in COM





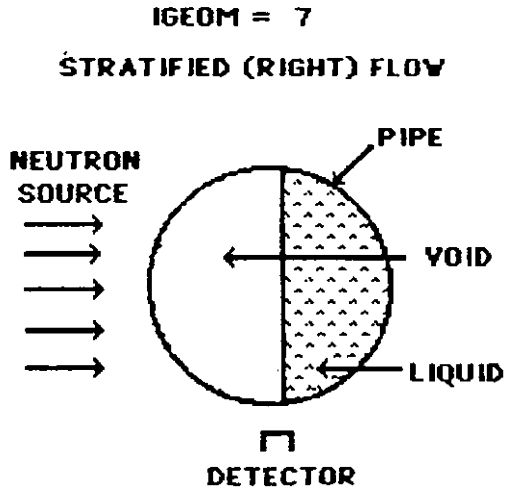


Fig. II.1: Geometries used in COM (continued)

Appendix III

SAMPLE INPUT AND OUTPUT FOR COM

SAMPLE INPUT FOR COM

```
100 MM PIPE - 3 MEV - 0.25
58756421
01.0000E+0005.0000E+0001.0000E+0100.0000E+00
3.0000E+000.5000E-06
051001000000001
03.0000E+0100.0000E+0000.0000E+00
1
```

SAMPLE OUTPUT FOR COM

```
SEED FOR RANDOM NUMBER GENERATOR =      58756421

100 MM PIPE - 3 MEV - 0.25
DENSITY OF LIQUID= 1
CYLINDRICAL PIPE OF RADIUS =     5.000
THICKNESS =     0.000   (CM) ,AND HEIGHT=    10.000   (CM)
DISTANCE TO DETECTOR =    0.3000E+02    0.0000E+00    0.0000E+00
LIQUID FRACTION =    1.000   (HOM)

5100   NEUTRONS OF ENERGY OF  0.3000E+01
INCIDENT WITH      0.0000 -1.0000  0.0000   DIRECTIONAL COSINES


SUMMARY OF RANDOM WALK
  5100 PARTICLES IN TOTAL
  4615 ESCAPED
   485 REACHED 0.5000E-06 MEV ENERGY CUTOFF

NUMBER OF EPITHERMAL COLLISIONS =       21875
THERM. PROB. = 0.1463E+00(0.4336E-01)
ESCP.  PROB. = 0.4748E-05(0.1389E+00)
THERM. RESP. = 0.6900E-06(0.1455E+00)
 -DIRECT ESTIMATOR -   THERM. PROB. = 0.9510E-01
 ESCP.  PROB. = 0.4748E-05
 THERM.  RESP. = 0.4515E-06


LAST SEED USED =
  167501568
RANDOM WALK TIME =    5.22410 SECONDS
```

Appendix IV

THE VECTOR PROGRAM VCOM

```
      PROGRAM MAIN(INPUT,OUTPUT,TAPE5=INPUT,TAPE6=OUTPUT)
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                       V   C   O   M                                 C
C                       -------------                                 C
C VCOM IS A CENTER-OF-MASS MONTE CARLO PROGRAM THAT SIMULATES         C
C  THE SCATTERING OF FAST NEUTRONS INCIDENT ON A WATER-VAPOR          C
C  TWO-PHASE FLOW IN A PIPE.  THE PROGRAM IS USED TO AID IN           C
C  DESIGNING NEUTRON DENSITOMETERS FOR USE IN TWO PHASE FLOW          C
C  SYSTEMS.  IT ASSUMES ISOTROPIC SCATTERING IN THE CENTER-           C
C  OF-MASS SYSTEM AND UTILIZES FUNCTIONAL FITS FOR NEUTRON            C
C  CROSS-SECTIONS.  SEVEN DIFFERENT FLOW REGIME GEOMETRIES            C
C  ARE TREATED USING SIMPLE ANALYTICAL GEOMETRY METHODS.              C
C  VCOM IS A VECTORIZED VERSION OF COM FOR THE CYBER-205              C
C  VECTOR COMPUTER USING FORTRAN-200.                                 C
C                                                                     C
C  WRITTEN BY  : T.TASSOU                                             C
C                SCHOOL OF COMPUTER SC., UNB                          C
C  DATE WRITTEN: OCTOBER, 1985.                                       C
C                                                                     C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C
      CHARACTER*3 JEOM(7)
      ROWWISE VSTAK(8,1000),VX(2,1000),VY(2,1000)
      ROWWISE VXI(3,1000),VCI(3,1000),VXR(2,1000),VYR(2,1000)
      ROWWISE VD(10,1000),VDST(4,1000),VXO(2,1000),VYO(2,1000)
      ROWWISE VX1(2,1000),VY1(2,1000),VX3(2,1000),VY3(2,1000)
      DIMENSION TITLE(20),CI(3),VINTER(1000),VPHI(1000),V1(1000)
      DIMENSION VCSCM(1000),VPATH(1000),VE(1000),VCSL(1000),V2(1000)
      DIMENSION VXM(1000),VB(1000),VXM2(1000),VROOT(1000),VRAND(1000)
      DIMENSION VD1(1000),VD2(1000),VXX(1000),VYY(1000),XD(3)
      DIMENSION VARG(1000),FF(1000),FD(1000)
      BIT VBIT1(1000),VBIT2(1000),VBIT3(1000),VSTAT(1000),VBB(1000)
      DATA CI/0.0,-1.0,0.0/
      DATA SIGTH,SIGW /1.512,1.155/
      DATA RM1/939.55/,RM2/14897.61/,ATOM/0.03346/
      DATA SS03/4./,SS0/1.4/,SA0/.04/
      DATA JEOM/'HOM','IAN','ANN','STD','STU','STL','STR'/
C
C
C HOMOGENIZED FLOW IS ASSUMED IN THIS VERSION
C TITLE IS THE JOB TITLE
C KSEED=SEED FOR RANDOM NUMBER GENERATOR
C RHO = LIQUID DENSITY
C R   = PIPE RADIUS
C ZB  = PIPE HEIGHT
C THICK= PIPE THICKNESS
C E1  = PARTICLE ENERGY
C EMIN= CUT OFF ENERGY
C NEV = PARTICLE HISTORIES
C IT  = MAXIMUM TIME IN CPUS
C IGEOM= KIND OF GEOMETRY
C XD = DETECTOR POSITION
C (0,0,0) IS CENTER OF CYLINDER
C X DIRECTION IS TO BOTTOM OF PAGE
```

```
C Y DIRECTION IS TO L.H.S OF PAGE
C
C++ READ INPUT PARAMETERS
      READ(5,56) TITLE
      READ(5,*) KSEED
      READ(5,4) RHO,R,ZB,THICK
      READ(5,5) E1,EMIN
      READ(5,6) NEV,IT,IGEOM
      READ(5,4) (XD(I),I=1,3)
      READ(5,*)RDENS
      READ(5,*)N,MINSTK,MAX,NM
      WRITE(6,5007) KSEED
      IF(IGEOM.LE.0) IGEOM=1
C
      NSS=0
      ESCP=0.0
      ESCP2=0.0
      NESCP=0
      LESCP=0
      NTH=0
      NTOT=0
      IN=NEV
      PX=0.0
      PX2=0.0
      NN=N
C
C++ SELECT GEOMETRY TO PROCESS
      GO TO (11,12,13,14,15,16,16),IGEOM
11    RHOW=RHO
      RW=R
      RR=0.0
      RNR=R
      RNORM=1.
      GO TO 88
12    RHOW=1.0
      RW=SQRT(RHO)*R
      RNORM=R/RW
      RR=R
      RNR=RW
      GO TO 88
13    RHOW=1.0
      RW=R
      RR=SQRT(1.-RHO)*R
      RNR=RW
      RNORM=1.0
      GO TO 88
14    RHOW=1.0
      RW=R
      RNR=R
      TH=4.*RHO*ASIN(1.)
      TH1=TH
19    TH2=TH+SIN(TH1)
      CTH2=COS(TH2)
      TH1=(TH2-TH1*CTH2)/(1.-CTH2)
      IF(ABS(TH1-TH2).GT.1.E-06) GO TO 19
      RR=R*COS(TH1/2.)
      XRX=R-RR
      RMR=RR
      RNORM=2*R/XRX
      GO TO 88
15    RHOW=1.0
      RW=R
      TH=4.*RHO*ASIN(1.)
      TH1=TH
29    TH2=TH+SIN(TH1)
      CTH2=COS(TH2)
```

```
      TH1=(TH2-TH1*CTH2)/(1.-CTH2)
      IF(ABS(TH1-TH2).GT.1.E-06) GO TO 29
      RR=R*COS(TH1/2.)
      XRX=-(R+RR)
      RMR=RR
      RNR=R
      RNORM=ABS(2*R/XRX)
      GO TO 88
16    RHOW=1.0
      RW=R
      TH=4.*RHO*ASIN(1.)
      TH1=TH
30    TH2=TH+SIN(TH1)
      CTH2=COS(TH2)
      TH1=(TH2-TH1*CTH2)/(1.-CTH2)
      IF(ABS(TH1-TH2).GT.1.E-06) GO TO 30
      RR=R*COS(TH1/2.)
      RNR=R
      RNORM=1.0
      IF(IGEOM.EQ.7)RR=-RR
88    CONTINUE
C
C=========================== RANDOM WALK ============================
      L=0
      LDS=0
      NTERM=1
C++ SET THE COUNTER TO TRACK RANDOM WALK TIME
      NBEG=2**32-1
      CALL Q8WJTIME(NBEG)
C
C++ IN      = PARTICLES REMAINING TO BE PROCESSED
C++ IFIL    = NUMBER OF PARTICLES TO PUT IN STACK
C++ L       = NUMBER OF RETURNING PARTICLES IN STACK
C++ N,NN    = LENGTH OF STACK. IN CASE REMAINING PARTICLES
C             ARE LESS THAN N THEN N=REMAINING PARTICLES.
C             IF THE STACK IS NOT REFILLED THEN N TAKES
C             THE VALUE OF L.
C++ MINSTK  = IF PARTICLES IN STACK FALL BELOW MINSTK
C             THE STACK IS REFILLED.
C++ IFLAG   = IF ZERO THEN ALL PARTICLES ARE GONE.
C++ IF ALL THE PARTICLES ARE PROCESSED EXIT
C
999   IF(NTERM.EQ.0.AND.IN.LE.0)GO TO 9999
      IF(L.LE.MINSTK.AND.IFLAG.NE.0) THEN
          N=NN
          IFIL=N-L
          IF(IFIL.GE.IN) THEN
            IFIL=IN
            N=L+IFIL
            IFLAG=0
          ENDIF
C
C++ IF PARTICLES REMAINING, OR STACK  EMPTY BELOW
C++ A MINIMUM THEN GENERATE "IFIL" PARTICLES AND STORE
C++ THEM IN THE STACK.SINCE THE STACK MAY ALREADY CONTAINS
C++ PARTICLES Q8VMKZ GENERATES A CONTROL VECTOR WITH '1' IN
C++ THE POSITIONS TO BE FILLED AND '0'S ELSEWHERE. THEN, BY
C++ USING Q8VXPND THE RANDOM NUMBERS ARE PUT IN THE PROPER
C++ POSITION.
C
          CALL VRANF(VD,IFIL)
          VBIT1(1:N)=Q8VMKZ(L,N:N)
          VRAND(1:N)=Q8VXPND(VD(1,1:IFIL),VBIT1(1:N):N)
          IF(IGEOM.EQ.4.OR.IGEOM.EQ.5)THEN
            WHERE(VBIT1(1:N))VSTAK(2,1:N)=XRX*VRAND(1:N)+RMR
          ELSE
```

```
            WHERE(VBIT1(1;N))VSTAK(2,1;N)=RNR*(-1.+2.*VRAND(1;N))
          ENDIF
          WHERE(VBIT1(1;N))
            VD(1,1;N)=RNR*RNR-VSTAK(2,1;N)*VSTAK(2,1;N)
            VSTAK(3,1;N)=VSQRT(VD(1,1;N);N)
            VSTAK(4,1;N)=0.0
            VSTAK(1,1;N)=E1
            VSTAK(5,1;N)=CI(1)
            VSTAK(6,1;N)=CI(2)
            VSTAK(7,1;N)=CI(3)
            VSTAK(8,1;N)=1.0
          END WHERE
        ELSE
          N=L
          IFIL=0
        ENDIF
        IN=IN-IFIL
C
C++ IF VDST IS EMPTY OR FILLED BELOW 'MAX'
C++ THEN PROCEED WITH THE RANDOM WALK ELSE PROCESS
C++ THE VDST (CALCULATE ESCAPE PROBABILITY)
C++ LDS= NUMBER OF KILLED PARTICLES
        IF(LDS.LT.MAX)THEN
          VXI(1,1;N)=VSTAK(2,1;N)
          VXI(2,1;N)=VSTAK(3,1;N)
          VXI(3,1;N)=VSTAK(4,1;N)
C
C++ DETERMINE DISTANCE TRAVELLED BY PARTICLE USING THE
C++ MACROSCOPIC CROSS SECTION OF WATER.
          SCVAL=ATOM*RHOW
          WHERE(VSTAK(1,1;N).LT.3.)
            VD(5,1;N)=SSO3*SCVAL
            VD(6,1;N)=0.0
          OTHERWISE
            VD(5,1;N)=SSO*SCVAL
            VD(6,1;N)=SAO*SCVAL
          END WHERE
          SCVAL=2.*SCVAL
          FF(1;N)=VD(5,1;N)+VD(6,1;N)
          FD(1;N)=VD(5,1;N)/FF(1;N)
         VD(4,1;N)=(17.6023/(1.+VSTAK(1,1;N)*(7.417+.1105*VSTAK(1,1;N)))
     1    +2.71811/(1.+VSTAK(1,1;N)*(.2427+.0028*VSTAK(1,1;N))))*SCVAL
         VD(7,1;N)=1./((VD(4,1;N)+FF(1;N))*RDENS)
         CALL VRANF(VRAND,N)
         VD(8,1;N)=VLOG(VRAND(1;N);N)
         VPATH(1;N)=-VD(7,1;N)*VD(8,1;N)
         VSTAK(2,1;N)=VSTAK(2,1;N)+VPATH(1;N)*VSTAK(5,1;N)
         VSTAK(3,1;N)=VSTAK(3,1;N)+VPATH(1;N)*VSTAK(6,1;N)
         VSTAK(4,1;N)=VSTAK(4,1;N)+VPATH(1;N)*VSTAK(7,1;N)
C
C==================== NEUTRON ESCAPED? ========================
         VSTAT(1;N)=(VSTAK(2,1;N)**2+VSTAK(3,1;N)**2).GE.RW**2
         VD(4,1;N)=VABS(VSTAK(4,1;N);N)
         VSTAT(1;N)=(VD(4,1;N).GE.ZB*.5).OR.VSTAT(1;N)
         IF (IGEOM.EQ.3) THEN
           VINTER(1;N)=1.0
           VBIT3(1;N)=VXI(1,1;N).NE.VSTAK(2,1;N)
           WHERE (VBIT3(1;N))
            VXM(1;N)=(VSTAK(3,1;N)-VXI(2,1;N))/(VSTAK(2,1;N)-VXI(1,1;N))
            VB(1;N)=VXI(2,1;N)-VXM(1;N)*VXI(1,1;N)
            VXM2(1;N)=VXM(1;N)*VXM(1;N)+1.
            VROOT(1;N)=VXM2(1;N)*RR*RR-VB(1;N)*VB(1;N)
           OTHERWISE
            VROOT(1;N)=RR*RR-VXI(2,1;N)*VXI(2,1;N)
           END WHERE
           VBIT2(1;N)=(VROOT(1;N).GE.0.).AND.(.NOT.VBIT3(1;N))
```

```
      VBIT1(1;N)=(VROOT(1;N).GE.0.).AND.VBIT3(1;N)
      WHERE (VBIT1(1;N))
        VROOT(1;N)=VSQRT(VROOT(1;N);N)
        VD(1,1;N)=-VXM(1;N)+VB(1;N)
        VXR(1,1;N)=(VD(1,1;N)+VROOT(1;N))/VXM2(1;N)
        VXR(2,1;N)=(VD(1,1;N)-VROOT(1;N))/VXM2(1;N)
        VYR(1,1;N)=VXM(1;N)*VXR(1,1;N)+VB(1;N)
        VYR(2,1;N)=VXM(1;N)*VXR(2,1;N)+VB(1;N)
      END WHERE
      WHERE (VBIT2(1;N))
        VXR(1,1;N)=VSQRT(VROOT(1;N);N)
        VXR(2,1;N)=-VXR(1,1;N)
        VYR(1,1;N)=VXI(2,1;N)
        VYR(2,1;N)=VYR(1,1;N)
      END WHERE
      VD1(1;N)=(VSTAK(2,1;N)-VXR(1,1;N))**2
     $       (VSTAK(3,1;N)-VYR(1,1;N))**2
      VD2(1;N)=(VSTAK(2,1;N)-VXR(2,1;N))**2
     $       (VSTAK(3,1;N)-VYR(2,1;N))**2
      WHERE(VD1(1;N).GT.VD2(1;N))
        VXX(1;N)=VXR(1,1;N)
        VYY(1;N)=VYR(1,1;N)
        VXR(1,1;N)=VXR(2,1;N)
        VYR(1,1;N)=VYR(2,1;N)
        VXR(2,1;N)=VXX(1;N)
        VYR(2,1;N)=VYY(1;N)
      END WHERE
      WHERE (VROOT(1;N).LT.0.) VINTER(1;N)=-1
      VBIT3(1;N)=VINTER(1;N).GT.0.
      VBIT1(1;N)=(VSTAK(2,1;N)**2+VSTAK(3,1;N)**2).GT.RR*RR
      WHERE(VBIT1(1;N).AND.VBIT3(1;N))
          V1(1;N)=VXI(1,1;N)-VX(1,1;N)
          V2(1;N)=VXI(2,1;N)-VY(1,1;N)
          VD(1,1;N)=(V1(1;N)*V1(1;N))+(V2(1;N)*V2(1;N))
          V1(1;N)=VXI(1,1;N)-VX(1,1;N)
          V2(1;N)=VXI(2,1;N)-VY(2,1;N)
          VD(2,1;N)=(V1(1;N)*V1(1;N))+(V2(1;N)*V2(1;N))
      END WHERE
      WHERE((VD(4,1;N).GT.VD(5,1;N)).AND.VBIT3(1;N))
          VD(6,1;N)=1.-(VSTAK(7,1;N)*VSTAK(7,1;N))
          VD(1,1;N)=VSTAK(5,1;N)*VSTAK(3,1;N)
          V1(1;N)=VD(1,1;N)-(VSTAK(6,1;N)*VSTAK(2,1;N))
          VD(7,1;N)=V1(1;N)*V1(1;N)
          VD(8,1;N)=RR*RR+VD(6,1;N)-VD(7,1;N)
          VD(7,1;N)=2.*VSQRT(VD(8,1;N);N)/VD(6,1;N)
          VSTAK(2,1;N)=VSTAK(2,1;N)+VD(7,1;N)*VSTAK(5,1;N)
          VSTAK(3,1;N)=VSTAK(3,1;N)+VD(7,1;N)*VSTAK(6,1;N)
          VSTAK(4,1;N)=VSTAK(4,1;N)+VD(7,1;N)*VSTAK(7,1;N)
       VD(2,1;N)=VSTAK(2,1;N)*VSTAK(2,1;N)+VSTAK(3,1;N)*VSTAK(3,1;N)
          VD(3,1;N)=VABS(VSTAK(4,1;N);N)
       OTHERWISE
          VD(2,1;N)=0.
          VD(3,1;N)=0.
       END WHERE
       VSTAT(1;N)=(VD(2,1;N).GE.RW**2).OR.(VD(3,1;N).GE.ZB*.5)
       VSTAT(1;N)=VSTAT(1;N).AND..NOT.VBIT3(1;N)
     ELSE IF(IGEOM.EQ.4) THEN
       VSTAT(1;N)=VSTAK(2,1;N).LE.RR
     ELSE IF(IGEOM.EQ.5) THEN
       VSTAT(1;N)=VSTAK(2,1;N).GT.RR
     ELSE IF (IGEOM.EQ.6) THEN
       VSTAT(1;N)=VSTAK(3,1;N).LE.RR
     ELSE IF (IGEOM.EQ.7) THEN
       VSTAT(1;N)=VSTAK(3,1;N).GE.RR
     ENDIF
C
```

```
C++++++++++++++++++++++++++ THERMAL ENERGY +++++++++++++++++
      CALL VRANF(VRAND,N)
      VCSCM(1;N)=-1.+2.*VRAND(1;N)
      CALL VRANF(VRAND,N)
      WHERE(FF(1;N)*VD(7,1;N).LE.VRAND(1;N))
        VD(4,1;N)=RM1
      OTHERWISE
        VD(4,1;N)=RM2
        VSTAK(8,1;N)=VSTAK(8,1;N)*FD(1;N)
      END WHERE
      VD(5,1;N)=VSTAK(1,1;N)*(VSTAK(1,1;N)+2.*RM1)
      VD(6,1;N)=VSQRT(VD(5,1;N);N)
      VD(5,1;N)=VSTAK(1,1;N)+RM1
      VD(9,1;N)=(2.*RM1**2+2.*VD(4,1;N)*VD(5,1;N))*.5
      WHERE(VD(9,1;N).GT.0.0)
        V1(1;N)=(RM1*RM1)+(VD(4,1;N)*VD(4,1;N))
        VD(8,1;N)=V1(1;N)+2.*VD(4,1;N)*VD(5,1;N)
        VD(10,1;N)=VD(9,1;N)/VSQRT(VD(8,1;N);N)
      OTHERWISE
        VE(1;N)=0.0
        VCSL(1;N)=0.0
      END WHERE
      WHERE(VD(10,1;N).GT.RM1)
        VD(8,1;N)=(VD(10,1;N)*VD(10,1;N))-RM1*RM1
        VD(9,1;N)=VSQRT(VD(8,1;N);N)
        VD(8,1;N)=VD(6,1;N)/(VD(5,1;N)+VD(4,1;N))
        VD(6,1;N)=1.-VD(8,1;N)*VD(8,1;N)
        VD(5,1;N)=1./VSQRT(VD(6,1;N);N)
        VD(6,1;N)=VD(9,1;N)*VCSCM(1;N)*VD(8,1;N)+VD(10,1;N)
        VE(1;N)=VD(5,1;N)*VD(6,1;N)-RM1
      END WHERE
      WHERE(VE(1;N).LT.0.025E-06)VE(1;N)=0.025E-06
      WHERE(VD(10,1;N).GT.RM1)
      VD(1,1;N)=VD(5,1;N)*(VD(9,1;N)*VCSCM(1;N)+VD(8,1;N)*VD(10,1;N))
        VD(8,1;N)=VE(1;N)*(VE(1;N)+2.*RM1)
        VCSL(1;N)=VD(1,1;N)/VSQRT(VD(8,1;N);N)
      OTHERWISE
        VE(1;N)=0.0
        VCSL(1;N)=0.0
      END WHERE
      WHERE(VCSL(1;N).GT.1.)VCSL(1;N)=1.0
C
C======================== THERMALIZATION PROBABILITY================
      VBIT1(1;N)=VE(1;N).LE.EMIN
      VBIT2(1;N)=(VD(4,1;N).NE.RM1).AND.(.NOT.VBIT1(1;N))
       VD(5,1;N)=0.
      VD(6,1;N)=1.-(VSTAK(1,1;N)-EMIN)/(VSTAK(1,1;N)-49./81.*
     2                VSTAK(1,1;N))
      WHERE(VD(6,1;N).LT.0.)VD(6,1;N)=0.
      WHERE(VBIT2(1;N))
         VD(5,1;N)=VD(6,1;N)
      OTHERWISE
         VD(5,1;N)=1.-(VSTAK(1,1;N)-EMIN)/VSTAK(1,1;N)
      END WHERE
      WHERE(VBIT1(1;N)) VD(5,1;N)=1.
      PTH= Q8SDOT(VD(5,1;N),VSTAK(8,1;N))
      PX=PX+PTH
      PX2=PX2+PTH**2
C++ LESCP = NUMBER OF ESCAPED NEUTRONS
C++ NESCP = TOTAL NUMBER OF ESCAPED NEUTRONS
C++ NTH   = NUMBER OF THERMALIZED NEUTRONS
C++ NTOT  = TOTAL NUMBER OF THERMALIZED NEUTRONS
      LESCP=Q8SCNT(VSTAT(1;N))
      NESCP=NESCP+LESCP
      NTH=Q8SCNT(VBIT1(1;N))
      NTOT=NTOT+NTH
```

```
        L=N-(LESCP+NTH)
        NTERM=L
       ENDIF
299    CONTINUE
C
C++ CHECK IF VDST OVERFLOW IS GOING TO OCCUR. IF YES EXECUTE VDST.
C+
       IF(LDS.LT.MAX)THEN
         IF(NTH.GT.0)THEN
         VBB(1;NM)=Q8VMKZ(LDS,NM;NM)
         IF(NTH.GT.(NM-LDS))THEN
            IFL=1
            GO TO 199
         ENDIF
C++ TRANSFER THERMALIZED PARTICLES IN VDST
        LDS=LDS+NTH
        VD(1,1;NTH)=Q8VCMPRS(VSTAK(2,1;N),VBIT1(1;N);NTH)
        VD(2,1;NTH)=Q8VCMPRS(VSTAK(3,1;N),VBIT1(1;N);NTH)
        VD(3,1;NTH)=Q8VCMPRS(VSTAK(4,1;N),VBIT1(1;N);NTH)
        VD(4,1;NTH)=Q8VCMPRS(VSTAK(8,1;N),VBIT1(1;N);NTH)
        VDST(1,1;NM)=Q8VMERG(VD(1,1;NM),VDST(1,1;NM),VBB(1;NM);NM)
        VDST(2,1;NM)=Q8VMERG(VD(2,1;NM),VDST(2,1;NM),VBB(1;NM);NM)
        VDST(3,1;NM)=Q8VMERG(VD(3,1;NM),VDST(3,1;NM),VBB(1;NM);NM)
        VDST(4,1;NM)=Q8VMERG(VD(4,1;NM),VDST(4,1;NM),VBB(1;NM);NM)
       ENDIF
C
C++ COMPRESS STAK BY DELETING THERMALIZED & ESCAPED PARTICLES.
        IF(L.GT.0) THEN
          VSTAK(1,1;N)=VE(1;N)
          IF((NTH+LESCP).GT.0) THEN
          VBIT2(1;N)=.NOT.(VSTAT(1;N).OR.VBIT1(1;N))
          VSTAK(1,1;L)=Q8VCMPRS(VSTAK(1,1;N),VBIT2(1;N);L)
          VSTAK(2,1;L)=Q8VCMPRS(VSTAK(2,1;N),VBIT2(1;N);L)
          VSTAK(3,1;L)=Q8VCMPRS(VSTAK(3,1;N),VBIT2(1;N);L)
          VSTAK(4,1;L)=Q8VCMPRS(VSTAK(4,1;N),VBIT2(1;N);L)
          VSTAK(5,1;L)=Q8VCMPRS(VSTAK(5,1;N),VBIT2(1;N);L)
          VSTAK(6,1;L)=Q8VCMPRS(VSTAK(6,1;N),VBIT2(1;N);L)
          VSTAK(7,1;L)=Q8VCMPRS(VSTAK(7,1;N),VBIT2(1;N);L)
          VSTAK(8,1;L)=Q8VCMPRS(VSTAK(8,1;N),VBIT2(1;N);L)
        ENDIF
C
C++ CALCULATE NEW DIRECTIONAL COSINES FOR PARTICLES IN STAK
          NSS=NSS+L
          CALL VRANF(VRAND,L)
          VPHI(1;L)=6.2832*VRAND(1;L)
          VD(4,1;L)=VSTAK(7,1;L)
          WHERE(VD(4,1;L).GT.1.)VD(4,1;L)=1.
          WHERE(VD(4,1;L).LT.-1.)VD(4,1;L)=-1.
          WHERE(VABS(VD(4,1;L);L).EQ.1.)
            VD(5,1;L)=0.
            VD(6,1;L)=0.
            VD(7,1;L)=1.
          OTHERWISE
            VD(8,1;L)=1.-(VD(4,1;L)*VD(4,1;L))
            VD(5,1;L)=VSQRT(VD(8,1;L);L)
            VD(7,1;L)=VSTAK(5,1;L)/VD(5,1;L)
            VD(6,1;L)=VSTAK(6,1;L)/VD(5,1;L)
          END WHERE
          WHERE(VD(6,1;L).GT.1)VD(6,1;L)=1.
          WHERE(VD(6,1;L).LT.-1)VD(6,1;L)=-1.
          WHERE(VD(7,1;L).GT.1)VD(7,1;L)=1.
          WHERE(VD(7,1;L).LT.-1)VD(7,1;L)=-1.
          WHERE(VCSL(1;L).GT.1)VCSL(1;L)=1.
          WHERE(VCSL(1;L).LT.-1)VCSL(1;L)=-1.
          VD(9,1;L)=1.-VCSL(1;L)**2
          VD(8,1;L)=VSQRT(VD(9,1;L);L)
```

```
          VD(9,1;L)=VD(8,1;L)*VCOS(VPHI(1;L);L)
          VD(10,1;L)=VD(8,1;L)*VSIN(VPHI(1;L);L)
          VD(3,1;L)=VCSL(1;L)
          VSTAK(5,1;L)=VD(9,1;L)*VD(4,1;L)*VD(7,1;L)-VD(10,1;L)
     3                 *VD(6,1;L)+VD(3,1;L)*VD(5,1;L)*VD(7,1;L)
          VSTAK(6,1;L)=VD(9,1;L)*VD(4,1;L)*VD(6,1;L)+VD(10,1;L)
     4                 *VD(7,1;L)+VD(3,1;L)*VD(6,1;L)*VD(5,1;L)
          VSTAK(7,1;L)=-VD(9,1;L)*VD(5,1;L)+VD(3,1;L)*VD(4,1;L)
        ENDIF
       ENDIF
C==============ESCAPE PROBABILITY============================
199       CONTINUE
       IF(LDS.GE.MAX.OR.IFL.EQ.1.OR.L.EQ.0)THEN
         M=LDS
         VD(4,1;M)=(VDST(1,1;M)-XD(1))**2+(VDST(2,1;M)-XD(2))**2
     5             +(VDST(3,1;M)-XD(3))**2
         VD(5,1;M)=VD(4,1;M)/((VDST(1,1;M)-XD(1))**2
     6             (VDST(2,1;M)-XD(2))**2)
         CALL VINTERS(VDST,XD,RW,VX,VY,VINTER,M)
         CALL VINTERS(VDST,XD,R+THICK,VX1,VY1,VINTER,M)
         VD(6,1;M)=(VX(1,1;M)-VX1(1,1;M))**2+(VY(1,1;M)-VY1(1,1;M))**2
         VD(7,1;M)=VD(5,1;M)*VD(6,1;M)
         VD(6,1;M)=VSQRT(VD(7,1;M);M)
         IF(IGEOM.EQ.1.OR.IGEOM.EQ.4)THEN
           VD(2,1;M)=((VDST(1,1;M)-VX(1,1;M))**2+(VDST(2,1;M)-
     7               VY(1,1;M))**2)*VD(5,1;M)
           VD(1,1;M)=VSQRT(VD(2,1;M);M)
         ELSE IF(IGEOM.EQ.2)THEN
           CALL VINTERS(VDST,XD,RR,VXO,VYO,VINTER,M)
           VD(6,1;M)=(VXO(1,1;M)-VX1(1,1;M))**2+(VYO(1,1;M)-
     8               VY1(1,1;M))**2
           VD(7,1;M)=VD(5,1;M)*VD(6,1;M)
           VD(6,1;M)=VSQRT(VD(7,1;M);M)
           VD(2,1;M)=((VDST(1,1;M)-VX(1,1;M))**2+(VDST(2,1;M)-
     9               VY(1,1;M))**2)*VD(5,1;M)
           VD(1,1;M)=VSQRT(VD(2,1;M);M)
         ELSE IF(IGEOM.EQ.3) THEN
         CALL VINTERS(VDST,XD,RR,VX3,VY3,VINTER,M)
         WHERE(VINTER(1;N).LE.0.)
           V1(1;M)=VDST(1,1;M)-VX(1,1;M)
           V2(1;M)=VDST(2,1;M)-VY(1,1;M)
           VD(2,1;M)=(V1(1;M)*V1(1;M)+V2(1;M)*V2(1;M))*VD(5,1;M)
           VD(1,1;M)=VSQRT(VD(2,1;M);M)
         OTHERWISE
           V1(1;M)=VX3(1,1;M)-VX(1,1;M)
           V2(1;M)=VY3(1,1;M)-VY(1,1;M)
           VD(7,1;M)=V1(1;M)*V1(1;M)+V2(1;M)*V2(1;M)
           V1(1;M)=VDST(1,1;M)-VX(1,1;M)
           V2(1;M)=VDST(2,1;M)-VY(1,1;M)
           VD(1,1;M)=V1(1;M)*V1(1;M)+V2(1;M)*V2(1;M)
         END WHERE
         WHERE(VD(1,1;M).GE.VD(7,1;M).AND.VINTER(1;M).GT.0)
          V1(1;M)=VX3(1,1;M)-VX3(2,1;M)
          V2(1;M)=VY3(1,1;M)-VY3(2,1;M)
          VD(2,1;M)=V1(1;M)*V1(1;M)+V2(1;M)*V2(1;M)
          VD(3,1;M)=VSQRT(VD(1,1;M);M)-VSQRT(VD(2,1;M);M)
          VD(1,1;M)=VD(3,1;M)*VD(3,1;M)
         END WHERE
         WHERE(VINTER(1;M).GT.0)
          VD(2,1;M)=VD(5,1;M)*VD(1,1;M)
          VD(1,1;M)=VSQRT(VD(2,1;M);M)
         END WHERE
        ELSE IF(IGEOM.EQ.5) THEN
         VD(2,1;M)=(RR-VDST(1,1;M))*(VY(1,1;M)-VDST(2,1;M))/
     $             (VX(1,1;M)-VDST(1,1;M))+VDST(2,1;M)
         VD(3,1;M)=(VDST(1,1;M)-RR)**2+(VDST(2,1;M)-VD(2,1;M))**2
```

```
        VD(2,1;M)=VD(3,1;M)*VD(5,1;M)
        VD(1,1;M)=VSQRT(VD(2,1;M);M)
       ELSE
        VD(1,1;M)=(VDST(1,1;M)-VX(1,1;M))**2+(VDST(2,1;M)-VY(1,1;M))**2
        VD(2,1;M)=(RR-VDST(2,1;M))*(VX(1,1;M)-VDST(1,1;M))/
     $             (VY(1,1;M)-VDST(2,1;M))+VDST(1,1;M)
        WHERE((VD(2,1;M)*VD(2,1;M)+RR*RR).LE.R*R)
         V1(1;M)=VDST(1,1;M)-VD(2,1;M)
         V2(1;M)=VDST(2,1;M)-RR
         VD(1,1;M)=V1(1;M)*V1(1;M)+V2(1;M)*V2(1;M)
        END WHERE
        VD(2,1;M)=VD(1,1;M)*VD(5,1;M)
        VD(1,1;M)=VSQRT(VD(2,1;M);M)
       ENDIF
        VARG(1;M)=SIGTH*RHOW*VD(1,1;M)+SIGW*VD(6,1;M)
        VD(2,1;M)=VEXP(-VARG(1;M);M)
        VARG(1;M)=VD(2,1;M)/(12.56637*VD(4,1;M)*VDST(4,1;M))
        ESCP=ESCP+Q8SSUM(VARG(1;M))
        V1(1;M)=VARG(1;M)**2
        ESCP2=ESCP2+Q8SSUM(V1(1;M))
        LDS=0
         IF(IFL.EQ.1) THEN
          IFL=0
          GO TO 299
         ENDIF
       ENDIF
       GO TO 999
9999   CONTINUE
C
       CALL Q8RJTIME(,,NEND)
C +++++++++ STATITICAL PARAMETERS
            IF(NTOT.EQ.0)GO TO 400
             XNEV=FLOAT(NEV)*RNORM
             PX=PX/XNEV
             PX2=(PX2/XNEV-PX**2)/(XNEV-1.)
             IF(PX.NE.0.0.AND.PX2.GE.0.) PX2=SQRT(PX2)/PX
             XNET=FLOAT(NTOT)
             ESCP=ESCP/XNET
             ESCP2=(ESCP2/XNET-ESCP**2)/(XNET-1.)
             IF(ESCP.NE.0.0.AND.ESCP2.GE.0.) ESCP2=SQRT(mScP2)/ESCP
C
  400        WRITE(6,58) TITLE
             WRITE(6,*)'DENSITY OF LIQUID=',RDENS
             WRITE(6,3005) R,THICK,ZB
             WRITE(6,350)XD
             WRITE(6,57) RHO,JEOM(IGEOM)
             WRITE(6,300) NEV,E1,CI
             WRITE(6,*)'STACK PARAMETERS,N,MISTK,MAX'
             WRITE(6,*)N,MINSTK,MAX
             WRITE(6,3781)
             WRITE(6,3782) NEV,NESCP,NTOT,EMIN
             WRITE(6,3780) NSS
             RESP=PX*ESCP
             RESP2=SQRT(PX2**2+ESCP2**2)
             WRITE(6,5780) PX,PX2,ESCP,ESCP2,RESP,RESP2
C DIRECT ESTIMATOR
             RPX=XNET/XNEV
             RESP=PX*ESCP
             WRITE(6,7780) RPX,ESCP,RESP
             WRITE(6,5008)
             CALL RANGET(KSEED)
             WRITE(6,*) KSEED
             WRITE(6,*) 'TIME IN MICROSECONDS = ',NBEG-NEND
C
C************** FORMAT STATETMENTS ***************************
  300   FORMAT(1HO/I7,'  NEUTRONS OF ENERGY OF  ',E10.4,
```

```
     1 /,'  INCIDENT WITH    ',3(1X,F7.4),
     2 '    DIRCTIONAL COSINES ',/)
 3780  FORMAT(' NUMBER OF EPITHERMAL COLLISIONS = ',I10)
 3781  FORMAT(//,' SUMMARY OF RANDOM WALK')
 3782  FORMAT(2X,I5,' PARTICLES IN TOTAL ',/,
     1        2X,I5,' ESCAPED ',/,
     2 2X,I5,' REACHED ',E10.4,' MEV ENERGY CUTOFF',/)
 5780       FORMAT(' THERM. PROB. = ',E10.4,'(',E10.4,')'
     1,/,           ' ESCP.  PROB. = ',E10.4,'(',E10.4,')'
     2,/,           ' THERM. RESP. = ',E10.4,'(',E10.4,')')
 7780      FORMAT('  -DIRECT ESTIMATOR - ',
     1 '  THERM. PROB. = ',E10.4,/,
     2 '  ESCP.  PROB. = ',E10.4,/,
     3 '  THERM.  RESP. = ',E10.4,/)
 5008   FORMAT(//,' LAST SEED USED =')
   57   FORMAT(' LIQUID FRACTION = ',F7.3,'  (',A3,')')
   58         FORMAT(1X,20A4)
 3005  FORMAT(' CYLINDRICAL PIPE OF RADIUS = ',F8.3,/
     1       ,'  THICKNESS = ',F8.3, '  (CM)'
     2       ,' ,AND HEIGHT= ',F8.3, '  (CM)')
    6    FORMAT(3I5)
    5    FORMAT(2E10.4)
    4    FORMAT(10E11.4)
 5007   FORMAT(' SEED FOR RANDOM NUMBER GENERATOR = ',I12,/)
   56  FORMAT(20A4)
 350    FORMAT(' DISTANCE TO DETECTOR =',3(2X,E11.4))
C*************************************************************
           STOP
           END
```

```
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C  THIS SUBROUTINE DETERMINES POINTS OF INTERSECTION                  C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
      SUBROUTINE VINTERS(VDST,XD,R,VXR,VYR,VINTER,N)
C
      DIMENSION VINTER(1000),XD(3),VXX(1000),VYY(1000)
      DIMENSION VD2(1000),VD(1000)
      DIMENSION VXM(1000),VB(1000),VXM2(1000),VROOT(1000),VD1(1000)
      ROWWISE VDST(4,1000),VXR(2,1000),VYR(2,1000)
      BIT VBIT(1000),VBIT1(1000),VBIT2(1000)
      VINTER(1;N)=1.0
      VBIT(1;N)=XD(1).NE.VDST(1,1;N)
      WHERE (VBIT(1;N))
        VXM(1;N)=(XD(2)-VDST(2,1;N))/(XD(1)-VDST(1,1;N))
        VB(1;N)=VDST(2,1;N)-VXM(1;N)*VDST(1,1;N)
        VXM2(1;N)=VXM(1;N)*VXM(1;N)+1.
        VROOT(1;N)=VXM2(1;N)*R*R-VB(1;N)*VB(1;N)
      OTHERWISE
        VROOT(1;N)=R*R-VDST(2,1;N)*VDST(2,1;N)
      END WHERE
        VBIT1(1;N)=(VROOT(1;N).GE.0).AND.VBIT(1;N)
     WHERE(VROOT(1;N).GE.0) VROOT(1;N)=VSQRT(VROOT(1;N);N)
     WHERE (VBIT1(1;N))
       VD(1;N)=-VXM(1;N)*VB(1;N)
       VXR(1,1;N)=(VD(1;N)+VROOT(1;N))/VXM2(1;N)
       VXR(2,1;N)=(VD(1;N)-VROOT(1;N))/VXM2(1;N)
       VYR(1,1;N)=VXM(1;N)*VXR(1,1;N)+VB(1;N)
       VYR(2,1;N)=VXM(1;N)*VXR(2,1;N)+VB(1;N)
       VXR(1,1;N)=VROOT(1;N)
       VXR(2,1;N)=-VXR(1,1;N)
       VYR(1,1;N)=VDST(2,1;N)
       VYR(2,1;N)=VYR(1,1;N)
     END WHERE
     VD1(1;N)=(XD(1)-VXR(1,1;N))**2
    $         (XD(2)-VYR(1,1;N))**2
     VD2(1;N)=(XD(1)-VXR(2,1;N))**2
    $         (XD(2)-VYR(2,1;N))**2
     WHERE(VD1(1;N).GT.VD2(1;N))
       VXX(1;N)=VXR(1,1;N)
       VYY(1;N)=VYR(1,1;N)
       VXR(1,1;N)=VXR(2,1;N)
       VYR(1,1;N)=VYR(2WYvN)
       VXR(2,1;N)=VXX(1;N)
       VYR(2,1;N)=VYY(1;N)
     END WHERE
     WHERE (VROOT(1;N).LT.0.) VINTER(1;N)=-1
     RETURN
     END
```

Appendix V

SAMPLE INPUT AND OUTPUT FOR VCOM

SAMPLE INPUT FOR VCOM

```
100 MM PIPE - 3 MEV - 1.0
58756421
01.0000E+0005.0000E+0001.0000E+0100.0000E+00
3.0000E+000.5000E-06
061001000000001
03.0000E+0100.0000E+0000.0000E+00
1
1000 500 700 1000
```

SAMPLE OUTPUT OF VCOM

```
 SEED FOR RANDOM NUMBER GENERATOR =      58756421

 100 MM PIPE - 20 MEV - 0.100
 DENSITY OF LIQUID=1.0000000000000
 CYLINDRICAL PIPE OF RADIUS =     5.000
 THICKNESS =     0.000   (CM) ,AND HEIGHT=    10.000   (CM)
 DISTANCE TO DETECTOR =    0.3000E+02    0.0000E+00    0.0000E+00
 LIQUID FRACTION =    1.000   (HOM)
   5100  NEUTRONS OF ENERGY OF   0.3000E+01
   INCIDENT WITH      0.0000 -1.0000  0.0000    DIRCTIONAL COSINES

 STACK PARAMETERS,N,MISTK,MAX
 1000 500 900


 SUMMARY OF RANDOM WALK
   5100 PARTICLES IN TOTAL
   4548 ESCAPED
    552 REACHED 0.5000E-06 MEV ENERGY CUTOFF

 NUMBER OF EPITHERMAL COLLISIONS =       21319
 THERM. PROB. = 0.1623E+00(0.1642E+00)
 ESCP.  PROB. = 0.6419E-05(0.1283E+00)
 THERM. RESP. = 0.1042E-06(0.2083E+00)
 -DIRECT ESTIMATOR -   THERM. PROB. = 0.1082E+00
 ESCP.  PROB. = 0.6419E-05
 THERM.  RESP. = 0.1042E-05


 LAST SEED USED =
 28714000345165
 TIME IN MICROSECONDS = 167039
```

Appendix VI

ON VECTORIZING MORSE-CG CODE

The COM program has the essential features of a Monte Carlo code in terms of the random walk process. However, there are some basic differences between a general purpose code like MORSE and the COM code.

In MORSE and other codes, extensive table look-up process is involved for the cross-section processing. In the COM code, this is simplified by using functional fits.

In COM a simplified analytical method is used for geometry tracking while in MORSE a combinatorial geometry method is used. The combinatorial geometry method involves the identification of the neighborhoud bodies in order to determine the next collision site. A learning process is introduced here to reduce the time consumed in geometry tracking.

Both the table look-up and the geometry learning processes were not included in the COM program. These processes may have negative effect on the speedup and may give rise to other complications due to the simultaneous request of information from different particles.

In order to examine the implications on the speedup of the above mentioned differences between COM and MORSE, a table look-up and a geometry learning process should be included in the COM code before the adaptation of the MORSE on supercomputers is attempted.

LIST OF REFERENCES

1. Bhavsar, V. C., "Parallel Algorithms for Monte Carlo solutions of some Linear Operator problems," Ph.D. thesis, I.I.T , Bombay, 1981.

2. Brown, B. F., and Martin, R. W., "Monte Carlo Methods for radiation transport analysis on vector Computers", Progress in Nuclear Energy, vol. 14, no. 3, pp. 269-399, 1984.

3. Brown, B. F., "Vectorized Monte Carlo", Ph.D. dissertation, Michingan Univerity, Michingan, 1981.

4. Calahan, D. A., Martin, W. R., et al., "Final report for preliminary studies on vectorized Monte Carlo for Los Alamos National Laboratory, University of Minchingan Report, FY80, August 15 1980.

5. Deitel, M. H., "An Introduction to Operating Systems", Addison-Wesley Publishing Company, Reading, Massachusetts, 1984.

6. Fluegelman, A., "PC-IALK III, User's Guide", The Headlands Press, Inc., Tiburon, 1983.

7. Gajski, D. D., and Peir, J. K., "Essential issues in Multiprocessor Systems", Computer,vol.18, No.6, pp. 9-27, June 1985.

8. Greenspan, H., Kelber, C. N., and Or Kent, D., "Computing methods in Reactor Physics", Gordon and Breach science Publishers, New York, 1968.

9. Halton, J.H., "A retrospective and prospective survey of the Monte Carlo method", SIAM Review, vol.12, No.1, pp. 1-63, 1970.

10. Hammersley, J.M., and Handscomb, D.C, "Monte Carlo methods", Methuen & Co. LTD, London 1964.

11. Harris, D. R., "ANDYMG3", LA-4339, Los Alamos National Laboratory, Los Alamos, CA., 1970.

12. Hussein, E. M. A., "Modeling of a Fast Neutron Scatterometer for Two-Phase Void-Fraction Measurement", in preparation, Nuclear Engineering Group, Department of Chemical Engineering, University of New Brunswick, Fredericton, NB., 1985.

13. Hwang, K., "Multiprocessor Supercomputers for Scientific/Enginering Applications", Computer,vol.18, No.6, pp. 57-73, June 1985.

14. Hwang, K., and Briggs, F.A, "Computer architecture and parallel processing", McGraw-hill Computer Science Series, New York, 1984.

15. Johnson, M. Paul, "An Introduction to Vector Processing", CRAY Research, Inc., Minneapolis, Minnesosta, 1984.

16. Kowalick, J. S., "HEP supercomputer and its applications", Scientific Computation Series, MIT press, Cambridge, Massachusetts, 1985.

17. Lambros, A. L., "Pseudorandom Number Generation in Parallel Computers", M.sc. Thesis, University of New Brunswick, Fredericton N.B, 1986.

18. Martin, W. R., Rathkupf, J. A., and Nowak, P. F., "Monte Carlo Photon Transport on the CRAY-XMP", Trans. American Nuclear Society, Nov. 1985.

19. Patankar, S. V., "Programming Techniques for the CRAY-I supercomputer", Course notes, University of Minnesota, Minneapolis, 1984.

20. Rodrigue, G., "Parallel Computations", Academic Press Inc., New York, 1982.

21. Shreider, A. Yu, "Methods of Statistical Testing", Elsevier P.C, 1964.

22. Smith, B. J., "Architecture and Applications of the HEP Multiprocessor Computer System", Real-time Signal Processing IV, vol.298, Society of Photo-Optical Instrumentation Engineers, pp. 241-248, Aug. 1981.

23. Sydow, J. Peter, "Optimization Guide SN-0220", CRAY Research, Inc., Minnesota, May 1982.

24. Tassou, T., Bhavsar, V. C., Hussein, E. and Gujar, U. G, "Adapting a Monte Carlo radiation transport code to supercomputers", Proc. of the APICS Computer Science conference, Department of Mathematics, Statistics & Computer Science, Dalhaousie University, Halifax NS, pp.132-144, November 8-9, 1985.

25. Troubetzkoy, E., Steinberg, H. and Kalos, M., "Monte Carlo radiation penetration calculations on a parallel computer", Trans. Am. Nucl. Soc., vol.17, p.260, 1973.

26. , A conference on "Monte Carlo methods and future computer architectures", Brookhaven National Laboratory, Upton, N.Y, May 2-5, 1983.

27. , Control Data Corporation, CDC Cyber 200 Model/205 computer system Hardware Reference Manual, No.60256020, 1984.

28. , Control Data Corporation, Fortran 200 Version 1 Reference Manual, No. 60480200, 1984.

29. , Control Data Corporation, CDC VSOS Version 2 Reference Manual, No. 60459410, 1984.

30. , Control Data Corporation, NOS Version 2 Reference Set Guide to System Usage, No. 60459670, 1984.

31. , "MORSE-CG", RSIC Computer Code Collection Oak Ridge National Laboratory, Report No. CCC-203C & D, 1984.

VITA

Candidate's full name:	Tassos Andrea Tassou
Place and date of birth:	Pano Amiantos, Limassol, Cyprus, January 21, 1960.
Permanent address:	Kalopanayiotis Nicosia, Cyprus.
Schools attended:	Solea Gymnasium (A), Evrichou, Solea, Cyprus, 1972-1978.
	Higher Technical Institute, Nicosia, Cyprus, Diploma of Technician Engineer, 1978-1980.
Universities attended:	University of New Brunswick, Fredericton, Canada, Bsc(EE), 1982-1984.

Publications:

1. Tassou, T., Bhavsar, V. C., Hussein, E. and Gujar, U. G., "Adapting a Monte Carlo radiation transport code to supercomputers," Proc. of the APICS Computer Science conference, Department of Mathematics, Statistics & Computer Science, Dalhousie University, Halifax NS, pp. 132-144, November 8-9, 1985.

TECHNICAL REPORTS

SCHOOL OF COMPUTER SCIENCE

Number	Date	Author	Title	No.of Pages	Price
TR74-001	Feb 1974	L.F. Johnson	A Search Algorithm for the Simple Cycles of a Directed Graph	33	$1.65
TR74-002	Oct 1974	W.D. Wasson R. McIssaac	A New Spanning Tree Algorithm	15	$0.75
TR74-003	Oct 1974	U.G. Gujar	Remote Job Entry and Output Through APL	13	$0.65
TR75-004	Apr 1975	U.G. Gujar	Subroutines with Variable Numbers of Arguments	17	$0.85
TR75-005	Jul 1975	L.E. Garey	Block Methods for Nonlinear Volterra Integral Equations	16	$0.80
TR75-006	Aug 1975	D.M. Fellows	Comments on "A General Fortran Emulator for IBM 360/370 Random Number Generator 'RANDU'"	13	$0.65
TR75-007	Aug 1975	L.E. Garey M. LeBlanc	Quadrature Formulae for Functions of Two Variables and Applications	22	$1.10
TR75-008	Sep 1975	L.F. Johnson	Determining Cliques of a Graph	14	$0.70
TR75-009	Oct 1975	D.M. Miller	An Algorithm for Determining the Chromatic Number of a Graph	21	$1.05
TR76-010	Jan 1976	L.E. Garey	Step by Step Methods for the Numerical Solution of Volterra Integro-Differential Equations	27	$1.35
TR76-011	Jan 1976	U.G. Gujar	A device Independent Computer Plotting System	21	$1.05
TR76-012	Mar 1976	P.P. Emin	A Partition Monitor for Fast-Batch-Processing with Limited Execution (Fable)	54	$2.70
TR77-013	Dec 1976	U.G. Gujar J.A. Fitzgerald	A Driver for Raster-Like Plotting Devices	24	$1.20
TR77-014	Jan 1977	U.G. Gujar D.M. Fellows	Automatic Job Scheduling in HASP	30	$1.50
TR79-015	May 1979	U.G. Gujar J.M. DeDourek M.E. McIntyre	A Method for Designing a Lexical Analyzer	30	$1.50

Number	Date	Author	Title	No.of Pages	Price
TR79-016	Jun 1979	U.G. Gujar A.R. Nagesh	ϕSCUBA A Buffered Core Graphics System	33	$1.65
TR79-017	Jun 1979	T.A. Middleton	A Transformation Approach to Implementing Aggregate Operations	30	$1.50
TR79-018	Jul 1979	T.A. Middleton	On Assignment Between Data Paths	40	$2.00
TR79-019	Aug 1979	T.A. Middleton	On the Use of "Fixed" Data Paths for Assignment	25	$1.25
TR79-020	Aug 1979	T.A. Middleton	Representing Data Paths by Program Structures	35	$1.75
TR79-021	Sep 1982	J.D. Horton	Sets With No Empty Convex 7-Gons	10	$0.50
TR83-022	Jun 1983	J.D. Horton	Resolvable Path Designs	29	$1.10
TR83-023	Aug 1983	U.G. Gujar F.W.L. So	Computer Display of Characters	62	$3.10
TR84-024	Jun 1984	J.D. Horton	A Lower Bound on the Number of One-Factors in Bicubic Graphs	22	$1.10
TR84-025	Jul 1984	U.G. Gujar	3-D Graphics in APL: User Perspective	27	$1.35
TR84-026	Aug 1984	J.D. Horton	A Polynomial-Time Algorithm to Find the Shortest Cycle Basis of a Graph	16	$.80
TR84-027	Sep 1984	L. Goldfarb	The Design of Efficient Pattern Recognition Systems	19	$0.95
TR84-028	Nov 1984	W.R. Knight	A Fortran to PL1 Dictionary	15	$0.75
TR85-029	Sep 1985	U.G. Gujar C.A. Crawford	Computer Processing of Musical Scores	40	$2.00
TR85-030	Dec 1985	V.C. Bhavsar J.R. Isaac	Design and Analysis of Parallel Monte Carlo Algorithms	49	$2.45
TR86-031	Jan 1986	N.N. Datar U.G. Gujar V.C. Bhavsar	Analysis and Synthesis of Interpolating Functions for 3-D Objects	22	$1.10
TR86-032	Jan 1986	T.A. Tassou	Adaptation of a Monte Carlo Radiation Transport Code to Supercomputers	143	$7.15