MONTE CARLO NEUTRON TRANSPORT ON THE ALLIANT FX/8

by

T. Tassou, V. Bhavsar, E. Hussein, K. Gallivan

TR86-035, June 1986

Faculty of Computer Science
University of New Brunswick
P.O. Box 4400
Fredericton, N.B. E3B 5A3

Phone: (506) 453-4566
Fax: (506) 453-3566

MONTE CARLO NEUTRON TRANSPORT

ON

THE ALLIANT FX/8

T. Tassou and V.C. Bhavsar
School of Computer Science

E. Hussein
(Nuclear Engineering Group)
Dept. of Chemical Engineering

UNIVERSITY OF NEW BRUNSWICK
Fredericton, N.B., Canada

and

K.A. Gallivan
Center for Supercomputing Research & Development
University of Illinois
Urbana-Champaign, IL. 61801
U.S.A.

JUNE 1986

Research partially supported by the National Science and Engineering Research Council of Canada, Grant No. A0089.

ABSTRACT

A Center-of-Mass Monte Carlo program (COM) used in designing neutron densitometers is considered and its implementations are carried out on the Alliant FX/8 multiprocessor. The schemes for static and dynamic computation assignment parallel Monte Carlo algorithms developed in earlier work are used for carrying out the implementations. It is shown that linear speedup, almost equal to the number of processors, could be achieved with dynamic computation assignment schemes. It is found that the Alliant machine with eight processors is about 8 times slower than the CYBER-205 with two pipes.

CONTENTS

1. Introduction ... 1

2. The Center-of-Mass Monte Carlo Program (COM) ... 2

   2.1 Computational Results on Sequential Computers ... 5

3. Review of Vectorized COM Implementation on Cyber-205 ... 5

4. Adaption of COM on Alliant ... 7

   4.1 Random Number Generation ... 8
   4.2 Schemes for Parallel Implementations of COM ... 9
   4.3 Dynamic Computation Assignment Scheme ... 10
   4.4 Static Computation Assignment Scheme ... 13
   4.5 Cyber-205 Vectorization Scheme ... 18

5. Conclusion ... 20

1. INTRODUCTION

The particle transport calculations required in the design of current fission reactors, fusion devices, and radiation shielding involve detailed 3-dimensional analysis. Presently, Monte Carlo method is the only method capable of treating complicated 3-dimensional geometries. In the Monte Carlo method, the data from a large number (10^3 - 10^5 is not uncommon) of particles have to be gathered to reduce statistical uncertainties. Consequently, the Monte Carlo programs sometimes require several hours (or even days) of CPU time on sequential computers. Thus, it is natural to consider the implementations of such Monte Carlo programs on supercomputers.

In the earlier work [1,3,8], a Center-of-Mass Monte Carlo program (COM), which simulates the scattering of fast neutrons incident on a water-vapor two phase flow in a pipe, was successfully vectorized on a CYBER-205. The aim of the present work is to reconsider and carry out its implementations on FX/8 series multiprocessor at the Center for Supercomputing Research and Development, University of Illinois, Urbana, to gain insight into the programming and performance of Monte Carlo codes on multiprocessors. The multiprocessor algorithms are developed using the schemes proposed in [2].

This report is organized as follows. An overview of the COM program and the computational results obtained with sequential (scalar) processing on IBM 3081D and scalar Cyber 205 in [3,8] are given in Section 2. Section 3 briefly reviews the vectorized COM (VCOM) program for Cyber-205 and the computational results on Cyber-205 are given. In Section 4, first we discuss the problems in random number generation on multiprocessor machines and the approach chosen for the present imple-

mentations on FX/8 is given. Subsequently, the schemes for developing multiprocessor Monte Carlo algorithms and the modifications required in the COM program to implement these schemes on FX/8 are discussed. The computational results obtained on FX/8 are compared with the earlier results. The vectorized COM (VCOM) is also adapted to FX/8 and its performance is studied. Finally, concluding remarks are presented in Section 5.

2. THE CENTER-OF-MASS MONTE CARLO PROGRAM (COM)

The COM program [1], simulates the scattering of fast neutrons incident on a water-vapor two-phase flow in a pipe. It is used to aid in designing neutron densitometers for use in two-phase flow systems. COM 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. A uniform line source (beam) is used as the neutron source.

A schematic representation of the problem solved by the COM program is shown in Figure 1. The 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 and thermalization probabilities are determined by the program. The product of the two provides an estimate of the detector response.

The flow chart of the COM algorithm is given in Figure 2. The position of a source particle on the line source, the distance travelled by the particle before undergoing a collision, the new particle energy and directional cosines are all computed based on samples from a random number generator and particle physics.

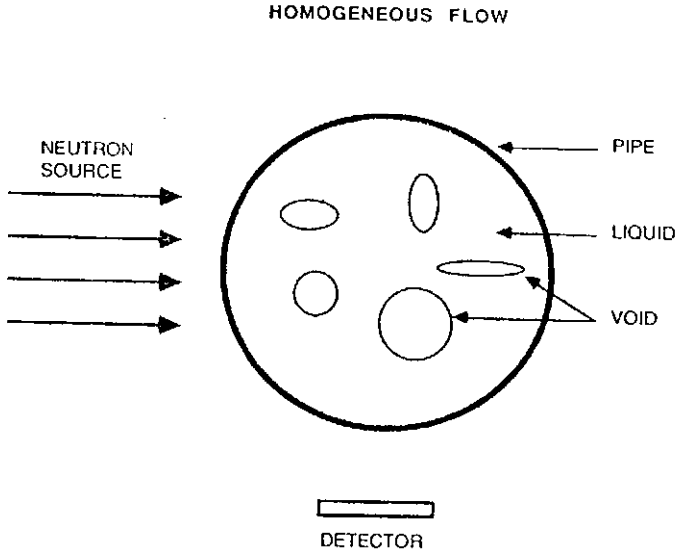


Figure 1: Schematic representation of the problem solved by COM

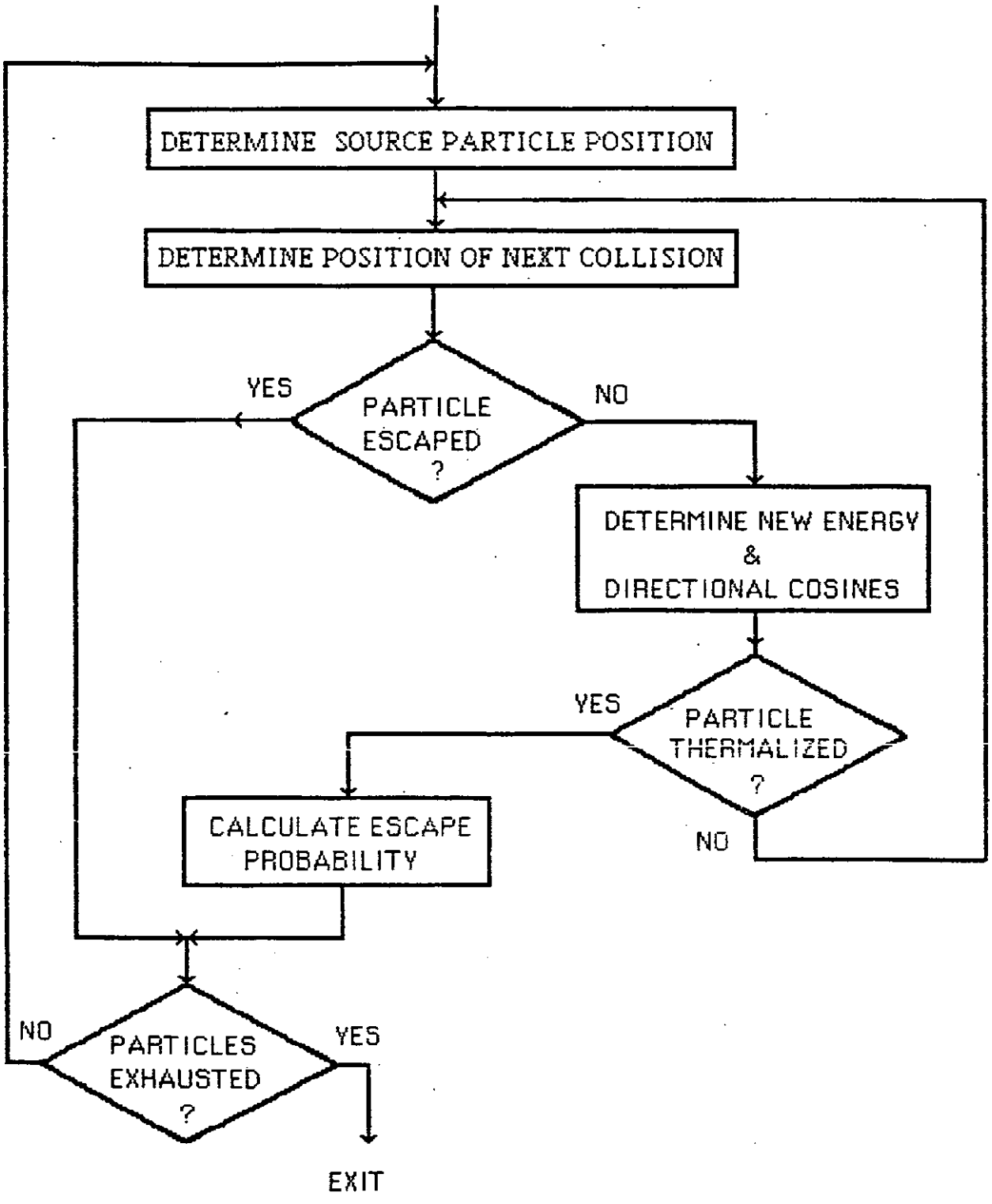


Figure 2: Flow diagram for scalar random walks

2.1 COMPUTATIONAL RESULTS ON SEQUENTIAL COMPUTERS [3]

COM was first implemented on the IBM 3081D at the University of New Brunswick and then on the Cyber-205 at the University of Calgary to run in the scalar mode. The input parameters used for these runs are as follows: 5100 particles, 20 Mev energy, 20 cm pipe radius and a liquid fraction of 5.0. The results obtained along with the timing requirements are shown in Table 1 below:

TABLE 1

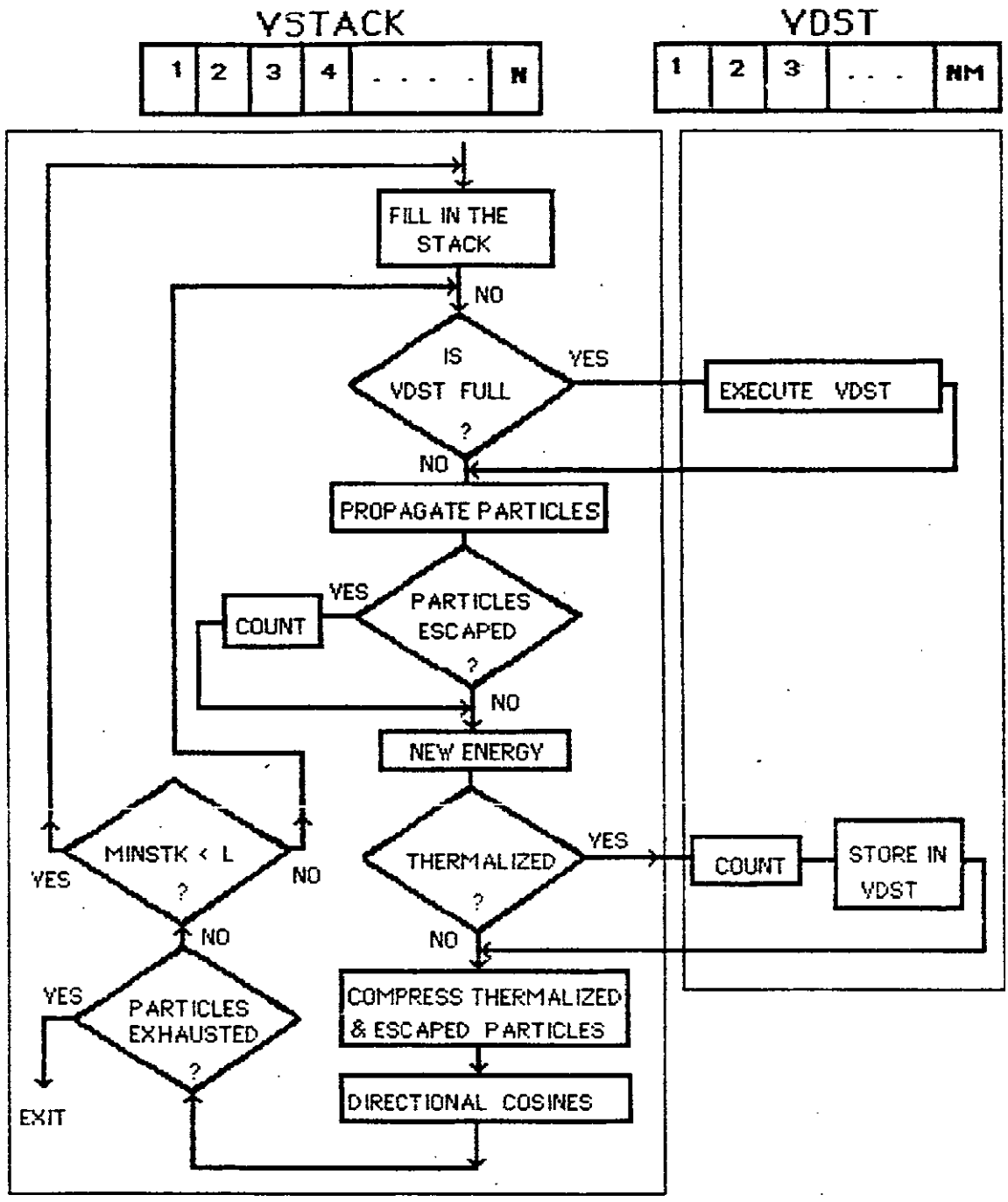
Thermalization Probability by COM on IBM 3081D and Cyber-205

COMPUTER SYSTEM	THERMAL. PROB.	CPU TIME (SEC)
IBM 3081D	0.761	18.59
CYBER-205/scalar mode	0.7498	7.4116

3.0 REVIEW OF VECTORIZED COM (VCOM) IMPLEMENTATION ON CYBER-205

COM was vectorized (VCOM) for the Cyber-205 (2-pipe) vector computer at the University of Calgary, Canada [3]. The basic idea for vectorization is as follows. First, the random walk for a particle is broken down to its basic events. Secondly, many particles are grouped into vectors (a vector is defined for each attribute of a particle, e.g. energy, position, direction). Then, the particles are followed from event to event through the random walk with the operations in each event block applied to all particles using vector instructions. A vector algorithm for VCOM is shown in Figure 3.

Two vector stacks (collection of vectors, not FIFOs) are used in this algorithm. VSTAK contains all the necessary information about particles. If its contents (no. of particles in it) fall below a



L : Current length (number of particles) of STACK
MINSTK : Minimum length of STACK

Figure 3: Flow diagram for vectorized random walks

specified value (MINSTK), VSTAK is refilled with new particles. VDST is the secondary stack and contains information about thermalized particles. If the number of thermalized particles exceeds a specified value (MAX), VDST is executed for the calculation of the escape probability.

The timing requirements of VCOM for the same parameters as given in section 2.1 are given in Table 2.

TABLE 2

Thermalization probability by VCOM on Cyber-205

COMPUTER SYSTEM	THERMAL. PROB.	CPU TIME (SEC)
CYBER-205/2-pipe	0.7352	0.4729

4. ALLIANT IMPLEMENTATION

The Alliant architecture offers two forms of parallelism based on two distinct but interconnected resource classes [5]:

a) The interactive processors (IPs) comprise an expendable pool of computers that execute interactive user jobs and the operating system in parallel with each other and with the computational complex.

b) The computational complex introduces a new form of parallelism, namely the "Alliant concurrency". It groups up to eight processors (CEs) in a complex. These processors can work in parallel on a single application transparently to the Fortran programmer. Each CE can deliver up to 11.8 MFLOPS performance for 32-bit arithmetic. The computational element (CE) is a 4450-kwhetstone (32-bit) general purpose microprogrammed computer with an integrated vector instruction set.

The FX/Fortran compiler implements the ANSI Fortran-77 programming language. Extension to the language include most of the array processing proposals for the next ANSI Fortran. The user has no control over

the Alliant parallel processing, except from the use of compiler directives that instruct the compiler to apply concurrency or vectorization whenever possible.

4.1 RANDOM NUMBER GENERATION

Pseudo-random number generation is a fundamental component in Monte Carlo simulations. The earlier work [4, 6] examined three methods for generating parallel pseudo-random number (PRN) sequences: parallel pseudo-random number sequences constructed from non-contiguous subsequences of a single Linear Congruential sequence, a number of Pseudo-Random Tree sequences, and multiple Chebyshev Mixing Transformations sequences. The results from the conducted tests showed that parallel PRN sequences constructed from non-contiguous subsequences are the most suitable for multiprocessor implementation.

Non-contiguous subsequences of a PRN sequence can be obtained basically in the following way on a multiprocessor [6]: A PRN generating algorithm is used with different "initial seeds" on each processor. The seeds must be chosen in such a way that the resulting subsequences do not overlap during run of a Monte Carlo program. The PRN generation program can be shared between all processors or the program code could be replicated for each processor (or process).

A PRN sequence (α) was generated on the Alliant machine using "rand" PRN generator supplied by the Alliant to form a basis for comparison with the results of the strategies for generating parallel PRN sequence on 8 processors. The following experiments were carried out:

1. First, the objective was to verify whether the "rand" code is reentrant so that it could be shared by processes on each processor. It was found that the code is not reentrant for simultaneous use by processors.
2. Secondly, the lockon and lockoff primitives developed by D. Sorenson [7] were used to assure mutual exclusion for using the "rand" code. The same default seed was used in eight processes, so that the resulting PRN sequences could be compared with the sequential case of generating only one PRN sequence (α). The PRN sequences produced were, rather strangely, not identical to α. Moreover, the eight PRN sequences were identical to each other except in every two other numbers. It was not possible to determine the cause of such a strange behavior.

In conclusion, it was decided to use the machine independent PRN generator, UNIRAN, used in our earlier work [3]. The UNIRAN subroutine was compiled recursively on the Alliant to allow for reentrancy.

4.2 SCHEMES FOR PARALLEL IMPLEMENTATIONS OF COM

The two schemes proposed in [2, 6] are used for assigning random walks (primary estimate computations (PECs)) to the processors. In a static computation assignment (SCA) scheme, a fixed number of PECs are assigned to each of the processors before any PECs are initiated. In contrast, in a dynamic computation assignment (DCA) scheme the number of PECs carried out by each processor is determined during the computations, based on some global criteria.

In SCA schemes, very small (if at all) time overheads are expected from the computation assignment decisions during the computations. However, the parallel algorithms based on these schemes cannot adapt to

the variations in the PEC times (which are random variables), in the sense that the computation time is minimized based on the run-time behavior. The DCA schemes, which try to minimize the run-time, may have a large time overhead due to the computation assignment decisions.

4.3 DCA SCHEME

Two DCA schemes have been proposed in [2, 6]. In DCA Scheme 2, given P processors, first we initiate P PECs in parallel. Whenever a PEC is complete, we initiate another PEC on that processor only if the required K PECs have not been already initiated. Fortunately, this scheme can be directly implemented by using the "CVD$1" compiler directive while compiling a FORTRAN DO loop, as illustrated below:

```
CVD$1 cncall
        DO 10 I = 1, K
          Primary Estimate Computation
      10 CONTINUE
```

In COM, the required statistics is accumulated during a random walk (i.e. primary estimate computation) using carry around scalars, as illustrated in Fig. 4(a). Since such scalars inhibit parallel execution, COM was modified as illustrated in Fig. 4(b). This figure also shows the modification in the main program. The carry around scalars are replaced by arrays so that each process can accumulate its statistics into a unique location (array element).

```
REAL ESCP
----
----
DO IN = 1, NEV
   ---
   CALL FOLNUT (--, --, ESCP, ---, ---)
   ---
   ---
END DO
---
STOP


SUBROUTINE FOLNUT (---, ---, ESCP, ---, ---)
---
---
ARG = EXP(-ARG)/(12.56637 x DIST2)*N
ESCP = ESCP + ARG
---
RETURN
```

(a) Scalar COM

```
    REAL ESCP(5100)
CVD$1 cncall
    DO IN = 1, NEV
       CALL FOLNUT (---, ---, ESCP(IN), ---, ---)
    END DO
    TOT = SUM (ESCP (1:NEV))
    ---
    STOP

    SUBROUTINE FOLNUT (---, ---, ESCP, ---, ---)
    ---
    ---
    ARG = EXP(-ARG)/(12.56637 * DIST2)*N
    ESCP = ARG
    ---
    RETURN
```

(b) Modified COM

Figure 4. Illustration of changes in the COM program for DCA Scheme 2.

The use of arrays, although inconvenient for a large number of random walks, was necessary because of the unavailability of synchronization primitives on the Alliant computer. Thus the random walk computations are completely independent of each other. At the completion of all the random walks, the required statistics is computed using the arrays.

Another change was made to the COM to move the initializations that were being performed in the main loop in calling the subroutine FOLNUT, to the subroutine itself. The performance of the modified COM for DCA Scheme 2 using 64-bit arithmetic for a varying number of processors is summarized in Table 3; the timing excludes input/output time. The running time required for modified COM with one processor is higher than the sequential COM due to the extra processing in the modified COM.

TABLE 3

Performance of the modified COM for DCA Scheme 2.

Processors	Time	Speedup 1	Speedup 2
1	30.23(t1)	1.0	0.99
2	15.05	2.009	1.99
3	10.0	3.023	3.00
4	7.51	4.025	4.0
5	6.1	4.956	4.92
6	5.08	5.95	5.91
7	4.26	7.096	6.88
8	3.86	7.83	7.78

Notes:
1. 64-Bit arithmetic used in COM and the modified COM.
2. The time required for sequential COM on Alliant (t2) = 30.04 sec.
3. Total number of particles (random walks) = 5100.
4. Speedup 1 = Time/t1.
5. Speedup 2 = Time/t2.

It is seen from Table 3 that the speedup 1 obtained in some cases is greater than the number of processors.

This behavior can be attributed to the following:

(a) if mean length of the random walks (μ) is smaller than the standard deviation (σ) of the length of the random walks, as explained in [2], and/or (b) the use of different sequence of random numbers in individual random walks in various cases of processors.

Figure 6 depicts the computing time required on the Alliant as the number of processors used vary from 1 to 8. Figure 7 shows the speed-up2 achieved as the number of processors vary from 1 to 8. It is seen that the speedup is almost linear, as expected from the results in [2].

4.4 AN SCA SCHEME

In this scheme, we assign the required K PECs among P processors in such a way that their sum is equal to K and each processor is assigned the same number of PECs, as far as possible; obviously, if P divides K exactly then we will assign (K/P) PECs to each of the processors. From the time complexity analysis in [2,6] for $P << K$, almost linear speedup is expected.

The COM program used for DCA Scheme2 was further modified to implement the SCA scheme as follows. The main program, instead of calling the random walk routine (FOLNUT) "NEV" times (i.e., the number of random walks required), calls a new subroutine (STLOOP) NPROC times, where NPROC represents the number of processors. The number NPROC is used to find out how many random walks will be carried out by each processor. For each processor, a starting location in the arrays (used for statistics collection) is calculated at the run time and is passed to the subroutine STLOOP. The subroutine STLOOP calls FOLNUT "NVM" times, where NVM represents the number of random walks to be executed by a processor. At the end, when all the random walks are completed, the required results are computed using the arrays used for statistics collection. Figure 5 illustrates the modifications made to the COM program.

```
      REAL ESCP(5100)
      ----
      ----
CVD$1 cncall
      DO IN = 1,NPROC

         CALL STLOOP(IN,---,ESCP(((IN-1)*int(NEV/NPROC))+1),---,---)
      END DO
      TOT = SUM(ESCP(1:NEV))
      ----
      STOP

      SUBROUTINE STLOOP(IOUT,---,ESCP,---,---)
      REAL ESCP(1)
      NUM = int(NEV/NPROC)
      IF(IOUT.EQ.NPROC)NVM = NEV - (NUM*(NPROC-1))
      DO IST = 1, NUM
         CALL FOLNUT (---,---,ESCP(IST),---,---)
      END DO
      RETURN


      SUBROUTINE FOLNUT(---,---,ESCP,---,---)
         ----
         ----
         ARG = EXP(-ARG)/(12.56637*DIST2)*W
         ESCP = ARW
         ---
         RETURN
```

Figure 5. Modifications for the SCA Scheme.

Table 4 gives the timing requirements and the speedup obtained for the COM using SCA scheme over the sequential program.

TABLE 4

Timing requirements and speedup obtained
using the modified COM for SCA Scheme

Processors	Time	Speedup 1	Speedup 2
1	31.02(t1)	1.0	0.97
2	15.49	2.002	1.94
3	10.26	3.023	2.93
4	7.78	3.987	3.86
5	6.34	4.893	4.74
6	5.27	5.886	5.7
7	5.07	6.118	5.92
8	3.99	7.774	7.53

Notes: 1. 64-Bit arithmetic used in the COM and the modified COM.
2. The time required for sequential COM(t2) = 30.04 sec.
3. Speedup 1 = (Time/t1).
4. Speedup 2 = (Time/t2).

Figure 6 shows the computing time required and the resulting speedup2 for the modified COM for SCA scheme as the number of processors vary from 1 to 8. It is seen that the speedup obtained is always smaller than that achieved with the DCA scheme. This implies that the computing time minimization obtained by using dynamic allocation is much more than the software overhead incurred and hence is the best for use on the Alliant System. Further, it is seen that when seven processors are used, there is a sharp decrease in the speedup due to the uneven distribution of random walks; in all the remaining cases, the number of processors exactly divides the total number of random walks (5100) required.

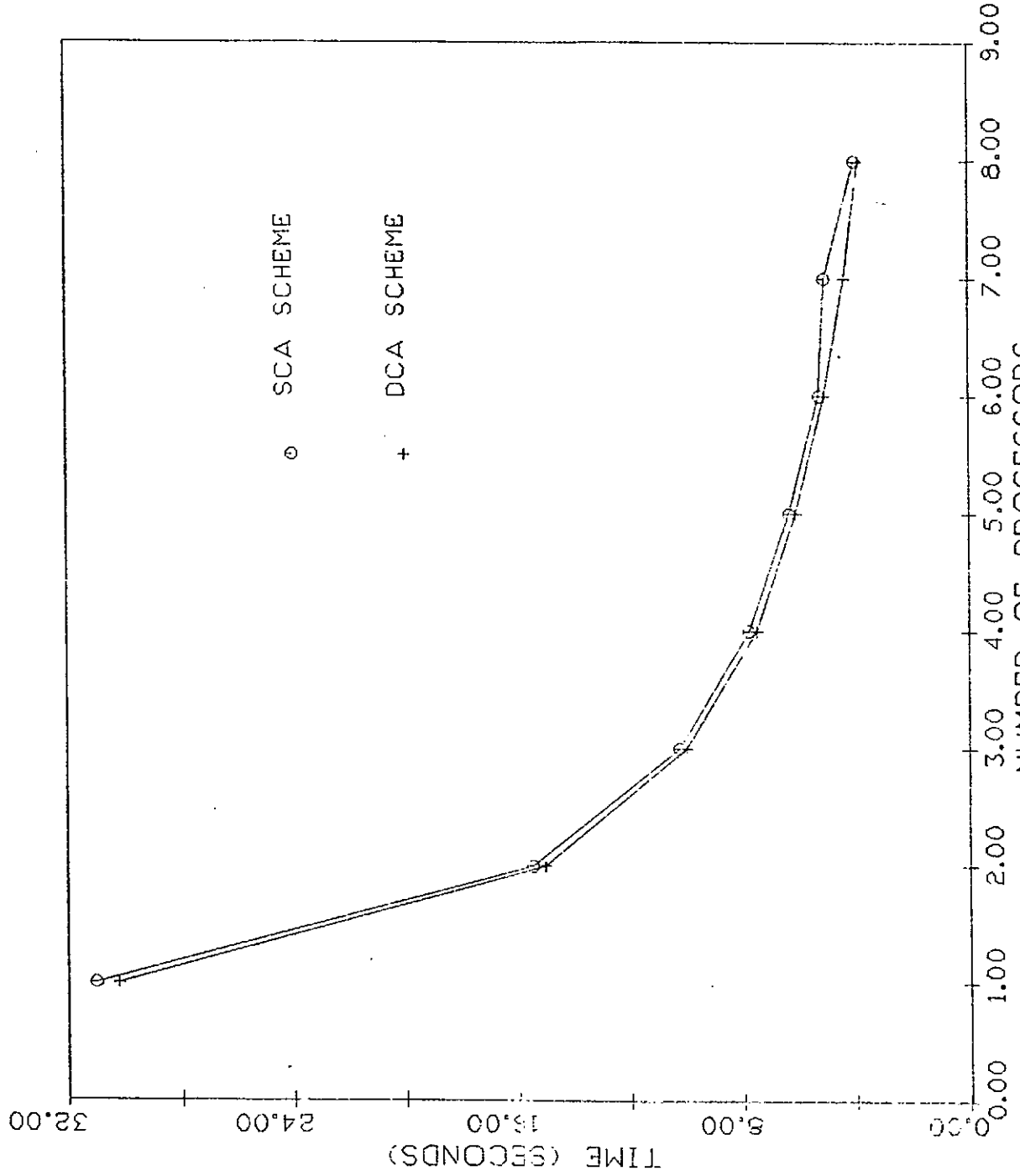


Figure 6. Computational time for parallel COM algorithms on the Alliant.

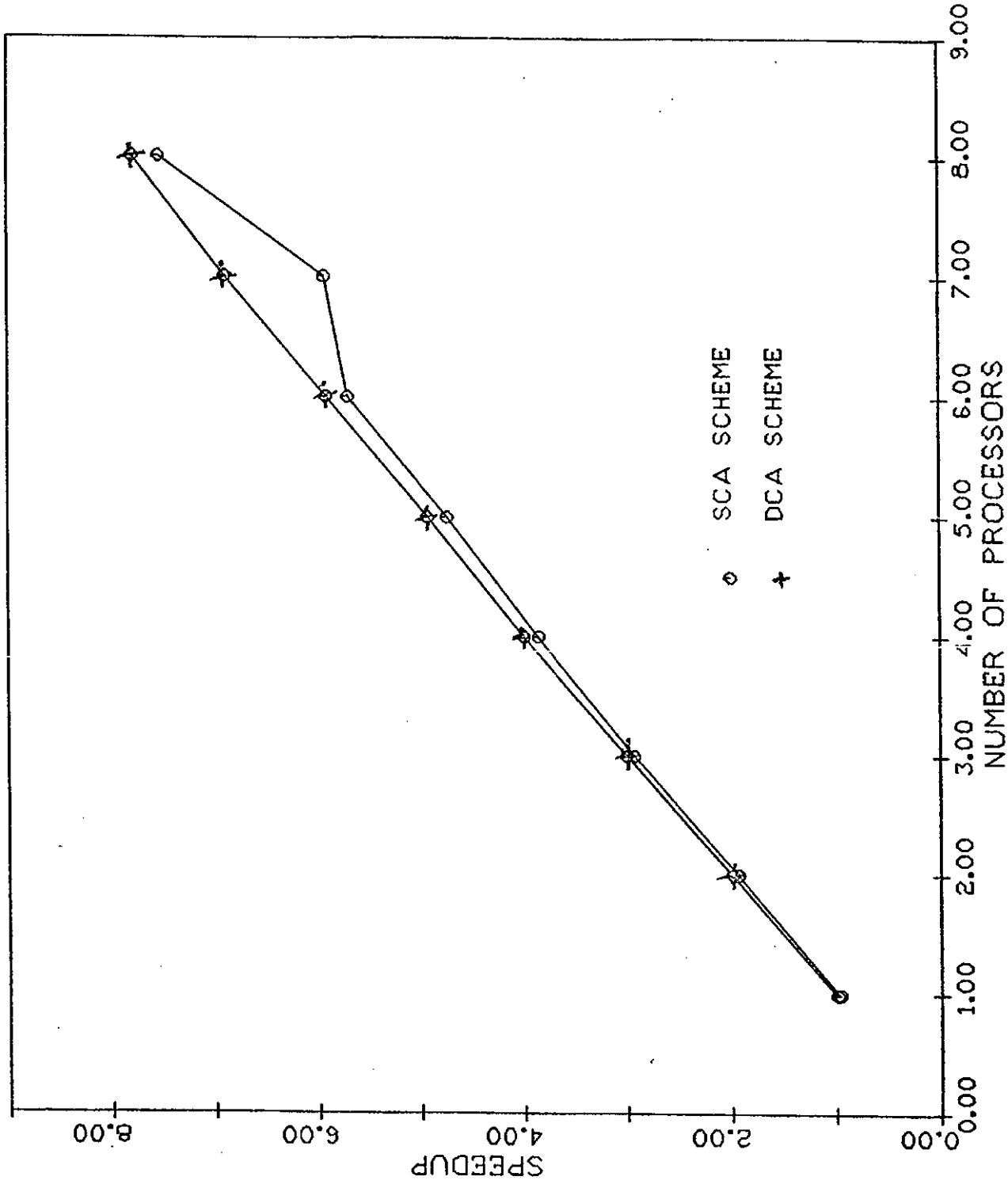


Figure 7. Speedup achieved with parallel COM algorithms on the Alliant.

4.5 CYBER-205 VECTORIZATION SCHEME

The vectorized version of COM (VCOM), designed to run on the Cyber-205 at the University of Calgary, Calgary, Canada, was syntactically modified to run on the Alliant. It was found that all the vector functions used in the VCOM are available on the Alliant, with different names and differing arguments. Further, a sequence of vector elements are referenced in different ways on the two machines. For example, to reference N elements of a vector X, starting from the 3rd element, we have to use:

(a) X(3;N) on the Cyber-205, and

(b) X(3:N+2) on the Alliant.

This necessitated major syntax changes to the VCOM. Finally, the bit arrays used in the VCOM were changed into the logical arrays.

Table 5 summarizes the computational time requirements for the modified VCOM on the Alliant. The various parameters chosen for these results are the following:

1. The dimensions of the various arrays were declared equal to 2000.
2. The main working stack (VSTAK) length (N) is varied from 1000 to 2000, in the steps of 100. The value of N decides the maximum number of particles that may be tracked at a time.
3. The minimum stack length (MINSTK), which is used to decide when to refill the VSTAK with particles, is chosen to be equal to 500.
4. The length (NM) of the secondary stack (VDST) is equal to 1000 and the maximum (MAX) stack length (i.e. the number of particles in VDST), used to determine when to execute the program to process information of thermalized particles, is chosen to be equal to 700.

5. Total number of particle histories used are equal to 5100.

6. All the problem parameters are the same as used in the sequential COM program.

From Table 5, it is seen that the execution time does not significantly change as the VSTAK length is varied.

TABLE 5

Computational times with the vectorized COM (VCOM) program modified for the Alliant.

STACK (N)	TIME
1000	8.3
1100	8.34
1200	8.37
1300	8.4
1400	8.5
1500	8.48
1600	8.53
1700	8.36
1800	8.42
1900	8.3
2000	8.54

Note: The parameters used the Alliant program are as follows:

1. Dimension of Arrays = 2000.
2. MINSTK = 500.
3. NM = 1000.
4. MAX = 700.
5. Total Number of particles = 5100.

To study the effect of change in the dimensions of the arrays, the various arrays sizes were decreased to 1000. The computing time on the Alliant with this modification, with N = 1000 and all the other parameters same as the above, was observed to be reduced to 6.78 seconds. It is felt that the reduction in the declared dimensions of the arrays changes the memory allocation and that reduces the computing time.

By comparing with the Cyber 205 running time, Alliant implementation of VCOM runs about 14 times slower.

5. CONCLUSION

In this report, we reconsidered the Center-of-Mass Monte Carlo program (COM), which simulates the scattering of fast neutrons on a water vapor two phase flow in a pipe. The COM was adapted to run on the Alliant FX/8 using the static and dynamic computation assignment schemes developed in the earlier work. A vectorized version of COM (VCOM) implemented on CYBER-205 was also modified to run on the Alliant machine.

Since the maximum number of processors available on the Alliant is only eight, which is much smaller than the number of random walks (5100) carried out, the speedup achieved is almost linear, as expected from the earlier time complexity studies. The Alliant machine, with 8 processors and using DCA Scheme2, is found to be about 8 times slower than CYBER-205 with two pipes at the University of Calgary, Canada.

REFERENCES

1. E.M.A. Hussein, "Modelling and Design of a Fast Neutron Scatterometer for Two-Phase Void-Fraction Measurement", In preparation.

2. V.C. Bhavsar and J.R. Isaac, 'Design and Analysis of Parallel Monte Carlo Algorithms', School of Computer Science, University of New Brunswick, Fredericton, N.B., TR85-030, Dec. 1985, to appear in SIAM J. of Scientific and Statistical Computing.

3. T.A. Tassou, 'Adaptation of a Monte Carlo Radiation Transport Code to Supercomputers', M.Sc. Thesis, School of Computer Science, University of New Brunswick, Fredericton, N.B., TR86-032, Jan. 1986.

4. L.A. Lambrou, 'Pseudo-Random Number Sequences for Parallel Computers', M.Sc. Project Report, School of Computer Science, University of New Brunswick, N.B., TR86-033, Feb. 1986.

5. ---, 'FX/Series Product Summary', Alliant Computer Systems Corporation, Acton, Massachusetts 01720, June 1985.

6. V.C. Bhavsar, 'Parallel Algorithms for Monte Carlo Solutions of Some Linear Operator Problem's, Dept. of Electrical Engg., Indian Institute of Technology, Bombay, 1981.

7. D. Sorenson, CSRD, University of Illinois, Urbana, personal communication, April 1986.

8. T. Tassou, V.C. Bhavsar, U.C. Gujar, E.M.A. Hussein, "Monte Carlo Neutron Transport on the Cyber 205", Super•C Newsletter, Supercomputing Services, The University of Calgary, Calgary, Canada, Vol. 2, No. 2, pp. 8-10 and p. 15, Spring 1986.