JAERI-M
86-151

VECTORIZATION OF KENO IV CODE AND AN ESTIMATE OF VECTOR-PARALLEL PROCESSING

October 1986

Kiyoshi ASAI, Kenji HIGUCHI,
Jun-ichi KATAKURA and Yutaka KURITA*

日本原子力研究所
Japan Atomic Energy Research Institute

JAERI-Mレポートは，日本原子力研究所が不定期に公刊している研究報告書です。
入手の問合わせは，日本原子力研究所技術情報部情報資料課（〒319－11茨城県那珂郡東海村）あて，お申しこしください。なお，このほかに財団法人原子力弘済会資料センター（〒319-11茨城県那珂郡東海村日本原子力研究所内）で複写による実費頒布をおこなっております。

JAERI-M reports are issued irregularly.
Inquiries about availability of the reports should be addressed to Information Division, Department of Technical Information, Japan Atomic Energy Research Institute, Tokai-mura, Naka-gun, Ibaraki-ken 319-11, Japan.

編集兼発行　日本原子力研究所
印　　刷　日立高速印刷株式会社

JAERI-M 86-151

VECTORIZATION OF KENO IV CODE AND AN ESTIMATE OF VECTOR-PARALLEL PROCESSING

Kiyoshi ASAI, Kenji HIGUCHI, Jun-ichi KATAKURA[+] and Yutaka KURITA[*]

Computing Center
Tokai Research Establishment
Japan Atomic Energy Research Institute
Tokai-mura, Naka-gun, Ibaraki-ken

(Received September 30, 1986)

The multi-group criticality safety code KENO IV has been vectorized and tested on FACOM VP-100 vector processor. At first the vectorized KENO IV on a scalar processor became slower than the original one by a factor of 1.4 because of the overhead introduced by the vectorization. Making modifications of algorithms and techniques for vectorization, the vectorized version has become faster than the original one by a factor of 1.4 and 3.0 on the vector processor for sample problems of complex and simple geometries, respectively. For further speedup of the code, some improvements on compiler and hardware, especially on addition of Monte Carlo pipelines to the vector processor, are discussed. Finally a pipelined parallel processor system is proposed and its performance is estimated.

Keywords: Monte Carlo, KENO IV, Nuclear Code, Vector, Parallel, Vectorization, Monte Carlo Pipeline

+ Department of Fuel Safety Research
*Nuclear Energy Data Center

KENO Ⅳコードのベクトル化とベクトル・パラレル処理の推定

日本原子力研究所東海研究所計算センター
浅井　清・樋口　健二・片倉　純一[+]・栗田　豊*

（1986年9月30日受理）

多群臨界安全性コードKENO Ⅳをベクトル化し，FACOM VP－100ベクトル計算機で実行した結果について報告する。ベクトル化したKENO Ⅳは，それによってもたらされたオーバーヘッドのせいで，スカラ計算機上では当初1.4倍遅くなっている。しかし，ベクトル計算機上では，複雑な幾何形状に対しては1.4倍，単純な幾何形状については3倍速くなっている。これ以上の速度向上倍率を得るためには，ソフトウェアではコンパイラの改良，ハードウェアでは，間接番地参照能力の向上，モンテカルロ・パイプラインと呼ぶベクトル要素分類命令が必要なことを述べた。最後にパイプライン化された並列計算機を想定した場合に得られる速度向上倍率を推定した。

東海研究所：〒319-11 茨城県那珂郡東海村白方字白根2－4
+　燃料安全工学部
*　原子力データセンター

CONTENTS

1. INTRODUCTION 1
2. TEST PROBLEMS 2
A. Problem 1 - ORNL Critical Experiment 2
B. Problem 2 - JAERI Critical Experiment 2
3. VECTORIZATION OF PHASE 1 4
A. Deletion of Backward Jump 4
B. Introduction of Neutron Banks 4
C. Vectorization of Random Number Generation 7
D. Logarithmic Computation of Flight Distance 8
E. Computation of Scattering Angles 8
F. Selection of an Energy Group after Collision 17
G. Isolation of Rare Events 21
H. Simultaneous Processing of Generations 23
4. VECTORIZATION OF PHASE 2 26
A. Further Isolations of Rare Events 26
B. Computation of Two Dimensional Subscripts 26
C. Time Reduction by Neutron Cutoff 26
5. VECTORIZATION OF PHASE 3 29
A. Deletion of Geometry Selection 29
B. Deletion of Event Selection 29
C. Deletion of Flux and Fission Density Calculations 29
D. Deletion of Unnecessary Conditional Branches 29
6. FINDINGS 34
A. Expansion of Computation Time 34
B. Low Degree of Vectorization 34
C. Large Computational Amount for Geometry and Event Recognition 34
7. DISCUSSIONS 35
A. Mixed Compilation of Scalar and Vector in a DO-loop 35
B. Addition of a Pipeline for Geometric Branch 35
C. Addition of a Pipeline for Event Branch 37
D. Speedup of Indirect Addressing 37
8. AN ESTIMATE OF VECTOR-PARALLEL EXECUTION OF KENO IV CODE 40
A. A Concept of Pipelined Parallel Processor System 40
B. Speedup Ratio 41
C. Data Transfer between Processors 43
D. No Multi-task Operation 45

9. CONCLUDING REMARKS 46
ACKNOWLEDGEMENTS 46
REFERENCES 46
APPENDIX --- The Input Data and Geometrical Scheme of Problem 2 48

目　　次

1. はじめに …… 1
2. テスト問題 …… 2
A. 問題１－ORNL臨界実験 …… 2
B. 問題２－JAERI臨界実験 …… 2
3. フェーズ１のベクトル化 …… 4
A. 逆戻り分岐の除去 …… 4
B. 中性子バンクの導入 …… 4
C. 乱数生成のベクトル化 …… 7
D. 飛程の対数計算 …… 8
E. 散乱角の計算 …… 8
F. 衝突後のエネルギー群の選択 …… 17
G. 稀な事象の分離処理 …… 21
H. 多数世代の同時処理 …… 23
4. フェーズ２のベクトル化 …… 26
A. ２次の稀現象分離処理 …… 26
B. ２次元添字の計算 …… 26
C. 中性子数カットオフによる時間削減 …… 26
5. フェーズ３のベクトル化 …… 29
A. 幾何形状選択の除去 …… 29
B. 事象選択の除去 …… 29
C. 中性子束及び核分裂密度計算の除去 …… 29
D. 不要な条件分岐の除去 …… 29
6. 知　　見 …… 34
A. 計算時間の拡張 …… 34
B. 低いベクトル化率 …… 34
C. 幾何形状及び事象認識のための大きな計算量 …… 34
7. 議　　論 …… 35
A. DOループ内のスカラ，ベクトル混在翻訳 …… 35
B. 幾何形状分岐のためのパイプラインの付加 …… 35
C. 事象分岐のためのパイプラインの付加 …… 37
D. 間接番地参照の速度向上 …… 37

8. KENO Ⅳコードのベクトル・パラレル処理の推定 …… 40
A. パイプライン化並列プロセッサの概念 …… 40
B. 速度向上比 …… 41
C. プロセッサ間のデータ転送 …… 43
D. 不要なマルチ・タスク操作 …… 45
9. 結　　語 …… 46
謝　　辞 …… 46
参考文献 …… 46
付　　録　テスト問題2の幾何体系と入力データ …… 48

1. INTRODUCTION

The multi-group criticality safety code KENO IV [1] has been vectorized and tested on FACOM VP-100 vector processor.[2] In the following we will describe two sample problems used for the tests, vectorization procedures, performance gains, findings in the vectorization, discussions on further compiler and processor improvements for speedup of Monte Carlo codes, and an estimation of speedup ratio by vector-parallel processing of the KENO IV code.

Our vectorization procedures consist of following three phases:

(1) Phase 1 - Modification of computational algorithms to suit for vectorization.

(2) Phase 2 - Isolation of judgement procedures on occurrences of physically rare events.

(3) Phase 3 - Modification of code specific to a simple geometry.

In addition to the above vectorization, we present an estimate of vector-parallel processing of KENO IV code.

2. TEST PROBLEMS

We have chosen following two experiments for the test problems of vectorization.

A. Problem 1 - ORNL Critical Experiment

This is one of a series of experiments performed at Oak Ridge National Laboratory with homogeneous low-enrichment UF_4 - Paraffin mixture.[3] In the experiments critical masses and critical dimensions were measured at various enrichments of ^{235}U and various H/U atomic ratios.

As our test problem, we have chosen an unreflected critical system of spherical geometry obtained from buckling conversions(Fig. 1).

Although one of the characteristics of the Monte Carlo method is that it can treat a complex geometry rather easily, we have chosen the problem with simple geometry as a first step to vectorization. In this simple system behaviors of neutrons depend on only the neutron cross sections in the region and there are no events of region boundary crossing. Therefore a speedup by vectorization is expected to be realized effectively.

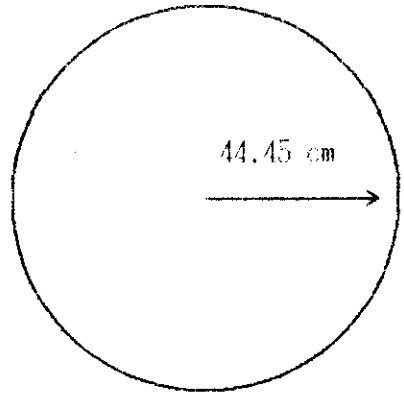

Fig. 1 Critical system of a spherical geometry(Problem 1)

B. Problem 2 - JAERI Critical Experiment

This is one of a series of TCA (Tank type Critical Assembly) critical experiments performed at JAERI[4]. The fuel rods of low-enrichment UO_2 (2.3% ^{235}U) were arranged in a 19 x 19 array in the core tank and the critical water level was measured(Fig.2). The fuel systems like this arrangement are often used for safety evaluations not only of reactors but also of storages or of transport casks.

Monte Carlo calculations required for safety evaluation deal with more complex systems. This system, however, contains fundamental problems on neutron behaviors of region boundary crossing. Therefore, we have chosen this experiment as the test problem for the second step to vectorization.

In the test calculations, we used the multi-group constant library MGCL with 137 energy groups which has 92 groups for fast and epithermal region and 45 groups with upscattering for thermal region[5]. The library has been developed at JAERI for evaluation of nuclear criticality safety.

For both problems we have used 300 neutrons in a generation and 203 generations in total, skipping the beginning 3 generations from the tally counts.

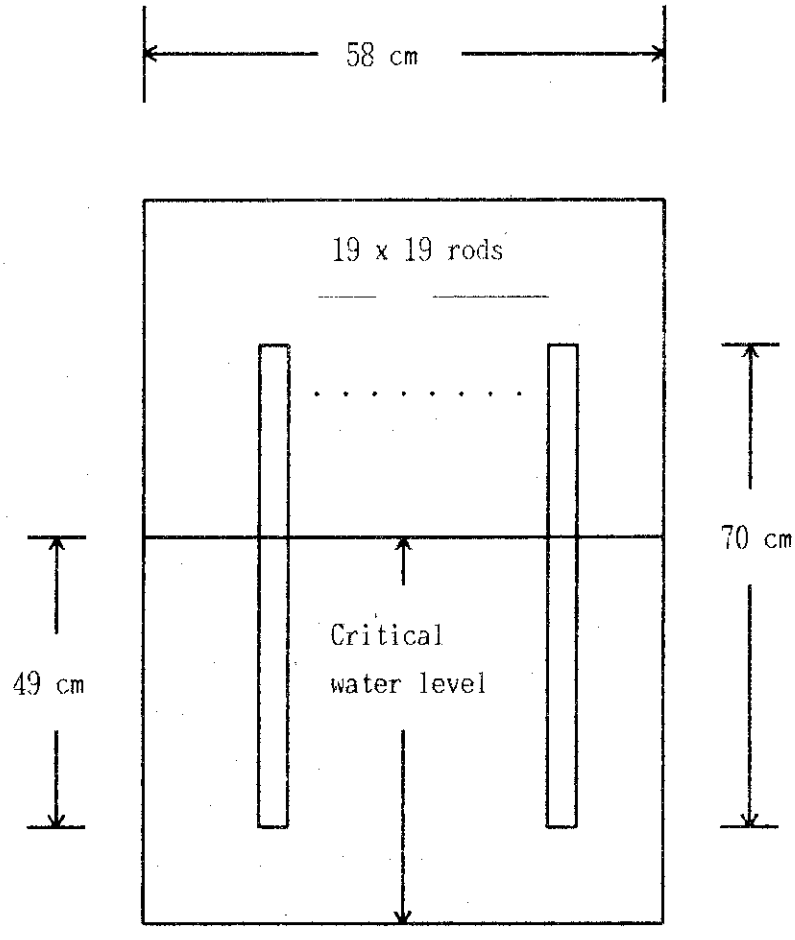

Fig. 2 Critical system of a tank type critical assembly(Problem 2)

3. VECTORIZATION OF PHASE 1

Before the the vectorization, we must know the parts of the code where most computation time is consumed. Figure 3 and 4 show the percentage of computation time consumed by subroutines in the original scalar version of KENO IV in case of the Problem 1, the Problem 2, respectively. Because more than 98% of computation time is consumed in this flow, i.e., subroutines BEGIN and CROS which is called in BEGIN, we have vectorized only these two subroutines. The major vectorization procedures of Phase 1 consist of following eight alternations of the algorithms.

A. Deletion of Backward Jump

Backward conditional or unconditional jumps in DO-loops of the the original scalar KENO IV were deleted by creating statements which are to be executed after the jumps. This is because the vector compiler inhibits backward jumps in a DO-loop.

B. Introduction of Neutron Banks

The computational flow of neutron tracking loop in KENO IV is shown in Fig. 5.[1] In the flow KENO IV has three implicit major computational loops. The first is a loop for generations. The second is a loop for neutrons in one generation. The third is a loop for migrations of a neutron in a given geometric system(Fig. 6). A nest of these three loops enforces KENO IV to consume a large amount of computation time. In the original KENO IV these loops are implicit in the sense that all variables in the loops are scalar and have no indices of the loops. Through the migration a neutron encounters many events such as inward/outward crossings of region boundaries, collisions, etc. We have provided a neutron stack called a bank with each event and subbanks with some subevents. Events correspond to the computational blocks in Fig. 5 and subevents are mainly branching operations for geometrical shape classifications. By issuing all seed neutrons in the beginning of a generation, they are collected in each bank at an arbitrary time. The neutrons in a bank can be processed in batch, which means that they may be handled by explicit DO loops. The introduction of banks changes the nested three implicit loops to explicit ones(Fig. 7). During the migration in a geometric system a neutron encounters various events randomly and is also stored randomly in a bank. This means that the variables such as coordinates, energies, scattering angles associated with the neutrons in the banks must also be accessed

NO.	ROUTINE	UNITS	LINES	ERR.	EXECUTIONS	COST	%	0....1....2....3....4....5....6....7....8....9.....
0001	BEGIN	1	773	0	1	528278272	99.2	**********
0002	FINALE	1	308	0	1	936478	0.2	
0003	CLEAR	1	10	0	3	859365	0.2	
0004	CROS	2	503	0	5451	528747	0.1	
0005	NSTART	1	47	0	103	474908	0.1	
0006	INPUT	1	177	0	1	439091	0.1	
0007	XSTAPE	1	242	0	1	391228	0.1	
0008	KEDIT	1	125	0	1	235508	0.0	
0009	FREAK	1	91	0	1	221347	0.0	
0010	START	1	430	0	1	57394	0.0	
0011	KENOG	1	404	0	1	49320	0.0	
0012	KENO	1	567	0	1	42347	0.0	
0013	AREAD	1	245	0	3	39851	0.0	
	IREAD				30			
	FREAD				140			
0014	VOLUME	1	152	0	1	25217	0.0	
0015	POSIT	1	69	0	300	22200	0.0	
0016	MESAGE	1	17	0	1	20039	0.0	
0017	FHLPR	1	31	0	3	20037	0.0	
0018	MAIN	1	71	0	1	13272	0.0	
0019	SAVE	1	19	0	5	3150	0.0	
0020	PULL	1	5	0	104	1144	0.0	
0021	DATIM	1	43	0	1	1078	0.0	
0022	CORSIZ	1	141	0	1	148	0.0	
0023	JOMCHK	1	483	0	1	68	0.0	
0024	ALOCAT	1	7	0	1	15	0.0	
0025	TIMFAC	1	6	0	1	14	0.0	
0026	NST2	1	133	0	0			
0027	REA	1	31	0	0			
0057	ALBIN	1	195	0	0			
	ALBEDO				0			
0058	AJOINT	1	79	0	0			
0059	SFLRAN	1	9	0	0			
0060	GEOM	1	80	0	0			
			11520	0		532660238		

Fig. 3 Execution cost(number of instructions) of Problem 1

NO.	ROUTINE	UNITS	LINES	ERR.	EXECUTIONS	COST	%	0.....1.....2.....3.....4.....5.....6.....7.....8..
0001	BEGIN	1	781	0	1	3346848936	83.1	**********
0002	CROS	1	503	0	7271488	675102106	16.8	**********
0003	INPUT	1	178	0	1	1154747	0.0	
0004	XSTAPE	1	243	0	1	1108734	0.0	
0005	FINALE	1	309	0	1	886179	0.0	
0006	CORSIZ	1	142	0	1	839472	0.0	
0007	START	1	431	0	1	838425	0.0	
0008	POSIT	1	70	0	13546	593920	0.0	
0009	NSTART	1	48	0	203	490327	0.0	
0010	KEDIT	1	126	0	1	422589	0.0	
0011	CLEAR	1	11	0	3	346946	0.0	
0012	BOX	1	51	0	1	267815	0.0	
0013	AREAD	1	253	0	36	134081	0.0	
	IREAD				89			
	FREAD				1142			
0014	VOLUME	1	153	0	1	130063	0.0	
0015	FREAK	1	89	0	1	74750	0.0	
0016	KENOG	1	405	0	1	72270	0.0	
0017	FILBOX	1	99	0	1	13298	0.0	
0018	KENO	1	568	0	1	12760	0.0	
0019	FHLPR	1	32	0	3	5451	0.0	
0020	MAIN	1	72	0	1	5160	0.0	
0021	MESAGE	1	18	0	1	4005	0.0	
0022	SAVE	1	20	0	5	3024	0.0	
0023	RDREF	1	113	0	1	1106	0.0	
0024	DATIM	1	44	0	1	1054	0.0	
0025	PULL	1	6	0	204	204	0.0	
0026	JOMCHK	1	485	0	1	152	0.0	
0027	TIMFAC	1	7	0	1	2	0.0	
0028	ALOCAT	1	8	0	1	2	0.0	
0029	STORE	1	32	0	0			
	STORE1				0			
0030	JOM7	1	70	0	0			
		⟩			⟩			⟩
0055	ALBIN	1	196	0	0			
	ALBEDO				0			
0056	AJOINT	1	80	0	0			
			7847	0		4029357578		

Fig. 4 Execution cost(number of instructions) of Problem 2

randomly. In the programming language Fortran, it is most suitable to use the indirect addressing technique for randomly accessed variables. Figure 8 shows the original scalar and the vectorized algorithms concerning the flight distances of neutrons. Figure 9 shows a procedure for separation of neutrons into two banks after the neutrons passed through the event.

Corresponding to the introduction of banks, the control flow of Fig. 5 is slightly modified. After the process of each event the control is transferred to a newly added section, where lengths of all banks are compared and an event in the longest bank length is activated(Fig. 10).

In Figs. 11 - 17, numbers of neutrons vs. Monte Carlo steps in major seven banks are shown for the Problem 2. At first the vectorized version was tested by issuing a neutron one by one as the original scalar version did and it was confirmed that the results of every computational step of the two versions were completely same.

C. Vectorization of Random Number Generation

The probability of a first collision between l and $l+dl$ along its line of flight is given by

$$p(l) = exp(-\Sigma_t l)\Sigma_t dl,$$

where Σ_t is the macroscopic total cross section of the medium and is interpreted as the probability of a collision per unit length[7].

Setting

$$\xi=\int_0^l e^{-\alpha}\Sigma_t ds=1-e^{-\beta},$$

$$\alpha=\Sigma_t s \ ,$$

$$\beta=\Sigma_t l \ ,$$

it follows that

$$l=-ln(1-\xi)/\Sigma_t \ .$$

Since $1-\xi$ is distributed in the same manner as ξ and hence $l=-ln\xi/\Sigma_t$. Thus l is obtained by giving a random number ξ between zero and unity. The computation of l, along with the computation of scattering angles, is the most easily vectorizable part of Monte Carlo code. The random number generator of the original KENO IV written in assembly language has been replaced by a vectorized library routine. It is specified to generate, e.g., 500 random numbers at a time to attain a long vector length in the generation. The generated numbers are stored in an array and are used for processing of neutrons in banks. Before a bank is processed, its bank length is checked and if the length is longer than that of the remaining random numbers, new random

numbers are generated and added to the array.

D. Logarithmic Computation of Flight Distance

An assembly language routine is used for the logarithmic computation of flight distance in the original code. This is replaced by a vectorized library routine. The reason of the replacement is twofold. One is because both computational speeds of the assembly language routine and our Fortran library routine are the same in scalar mode. This means that a vectorized library routine is much faster than the assembly routine. The other is because the algorithm used in the assembly routine contains a recurrence computation and is hard to vectorize. Although we have found no distinguished deviations by these replacements of B. and C. at present, thorough statistical tests should be done by making comparisons of the results of vectorized KENO IV with the results of many experiments before using the vectorized version.

E. Computation of Scattering Angles

Assembly language routines are used for computations of scattering angles. In our tests of the vectorized version it has been found that the computation time of these routines is the same as that of those written in Fortran statements. The situation is the same with regard to the logarithmic routine and the random number generator. This may be due to the recent improvements of algorithms of Fortran library routines. Thus we have replaced original assembly routines by Fortran statements, which have been easily vectorized.

As is described in the reference(1, p.11), direction cosines u', v', w' after a collision are calculated according to the relationship as shown in Fig. 18, where u, v and w are the initial direction cosines. If the anisotropic scattering data, MUBAR(μ), i.e., P_1 data, is zero, the new direction cosines are chosen from an isotropic distribution as follows: The assembly routine GTISO returns a set of random direction cosines u', v' and w' which are generated according to

$u' = \cos(\eta)\sin(\psi)$,

$v' = \sin(\eta)\sin(\psi)$,

$w' = \cos(\psi)$,

where $\cos(\eta)$ and $\sin(\eta)$ are obtained from AZIRN, The assembly routine AZIRN returns the sine and cosine of a random azimuthal angle, $\sin(\eta)$, $\cos(\eta)$, $\eta = 2\pi R$, and R is a random number between 0 and 1.

$\cos(\psi) = 2R-1$ and $\sin(\psi)$ = SQRT(1-cos(ψ)**2), where R is an another random

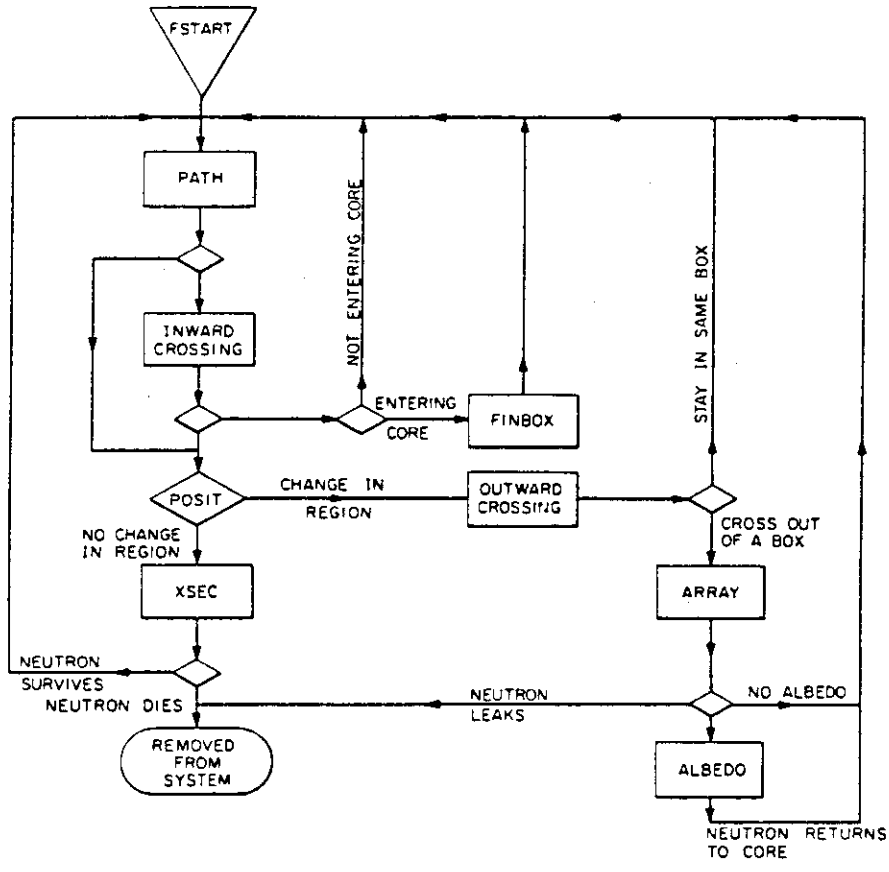

Fig. 5 Neutron tracking loop internal to the subroutine BEGIN

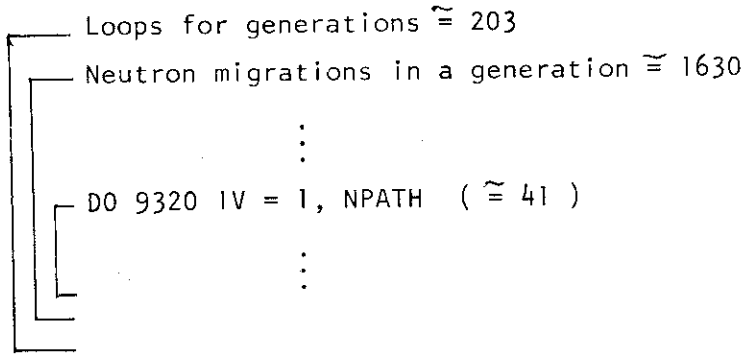

Fig. 6 Implicit major three loops in KENO IV(Problem 2)

```
      X1 = X + PATH * U
                 ↓
      DO 10 I = 1, NPATH
      J = PATHBANK(I)
10    X1(J) = X(J) + PATH(J) * U(J)
```

Fig. 7 An example of introduction of a bank concept

```
C*****************************START PATH*****
  320 IF ( RPTH.LE.0.0 ) RPTH = EXPRN(0)
      KR=MAT(K)
      IF(KR)340,330,340
  330 PTH=BIG
      GO TO 350
  340 PTH = RPTH*RSIGT(KR,IG)
  350 X1=X+PTH*U
      Y1=Y+PTH*V
      Z1=Z+PTH*W
      IF(X.EQ.X1) X1=X+SIGN(X,U)*(1.0E-6)
      IF(Y.EQ.Y1) Y1=Y+SIGN(Y,V)*(1.0E-6)
      IF(Z.EQ.Z1) Z1=Z+SIGN(Z,W)*(1.0E-6)
C*****************************END PATH*******
```

Fig. 8-a Computation of flight distance(original)

```
*VOCL LOOP,SCALAR
      DO 9321 IV=1,NPATH
      JV=PATHBA(IV)
 9321 IF (RPTH(JV).LE.0.0 ) RPTH(JV) = EXPRN(0)
*VOCL LOOP,NOVREC
      DO 9320 IV=1,NPATH
      JV=PATHBA(IV)
      KR(JV)=MAT(K(JV))
*VOCL STMT,IF(99)
      IF(KR(JV).NE.0) THEN
      PTH(JV)=RPTH(JV)*RSIGT(KR(JV),IG(JV))
      ELSE
      PTH(JV)=BIG
      ENDIF
      X1(JV)=X(JV)+PTH(JV)*U(JV)
      Y1(JV)=Y(JV)+PTH(JV)*V(JV)
      Z1(JV)=Z(JV)+PTH(JV)*W(JV)
*VOCL STMT,IF( 0)
      IF(X(JV).EQ.X1(JV)) X1(JV)=X(JV)+SIGN(X(JV),U(JV))*(1.0E-6)
*VOCL STMT,IF( 0)
      IF(Y(JV).EQ.Y1(JV)) Y1(JV)=Y(JV)+SIGN(Y(JV),V(JV))*(1.0E-6)
*VOCL STMT,IF( 0)
      IF(Z(JV).EQ.Z1(JV)) Z1(JV)=Z(JV)+SIGN(Z(JV),W(JV))*(1.0E-6)
 9320 CONTINUE
```

Fig. 8-b Computation of flight distance(vectorized)

```
      DO 9379 IV=1,NCROS1
      JV=CROSB1(IV)
*VOCL STMT,IF(36)
      IF(MCROS(JV).NE.0) THEN
        IF(LFLUX) FLUX(IGK(IV))=FLUX(IGK(IV))+WTP(IV)
        ICROS1=ICROS1+1
        CROSB1(ICROS1)=JV
      ELSE
        NPOS=NPOS+1
        POSBAN(NPOS)=JV
      ENDIF
 9379 CONTINUE
```

Fig. 9 Separation of particles in a bank

```
C
      MAXINX=MAXO(NPATH,NPOS,NALBED,NARRAY,NXSEC,NFIN,NCROS1,NCROS2,
     +                                                    NOUTAR)
      IF(MAXINX.LE.0) GO TO 962
  961 CONTINUE
      IF( NPATH.EQ.MAXINX) GO TO  180
      IF(  NPOS.EQ.MAXINX) GO TO 9430
      IF( NXSEC.EQ.MAXINX) GO TO  740
C..   IF(NCROS1.EQ.MAXINX) GO TO  360
      IF(NCROS2.EQ.MAXINX) GO TO  480
      IF(NARRAY.EQ.MAXINX) GO TO  491
      IF(NOUTAR.EQ.MAXINX) GO TO  560
      IF(  NFIN.EQ.MAXINX) GO TO 9400
      IF(NALBED.EQ.MAXINX) GO TO  731
```

Fig. 10 Selection of the longest bank

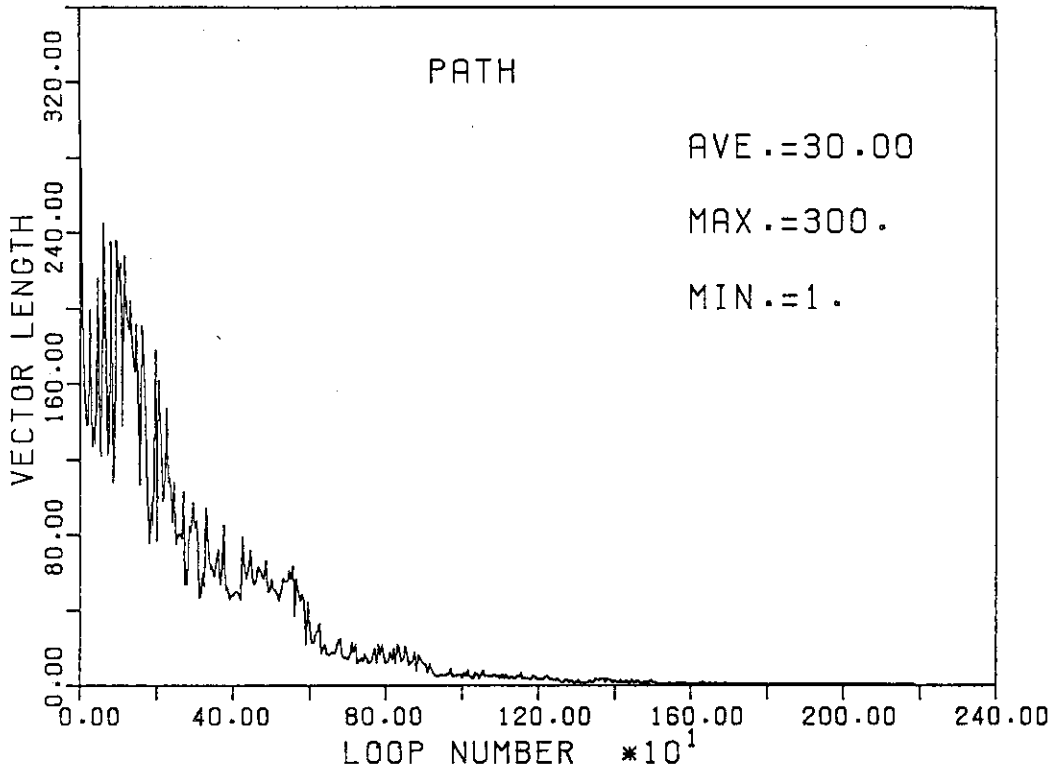

Fig. 11 Number of neutrons vs. Monte Carlo Steps(PATH bank)

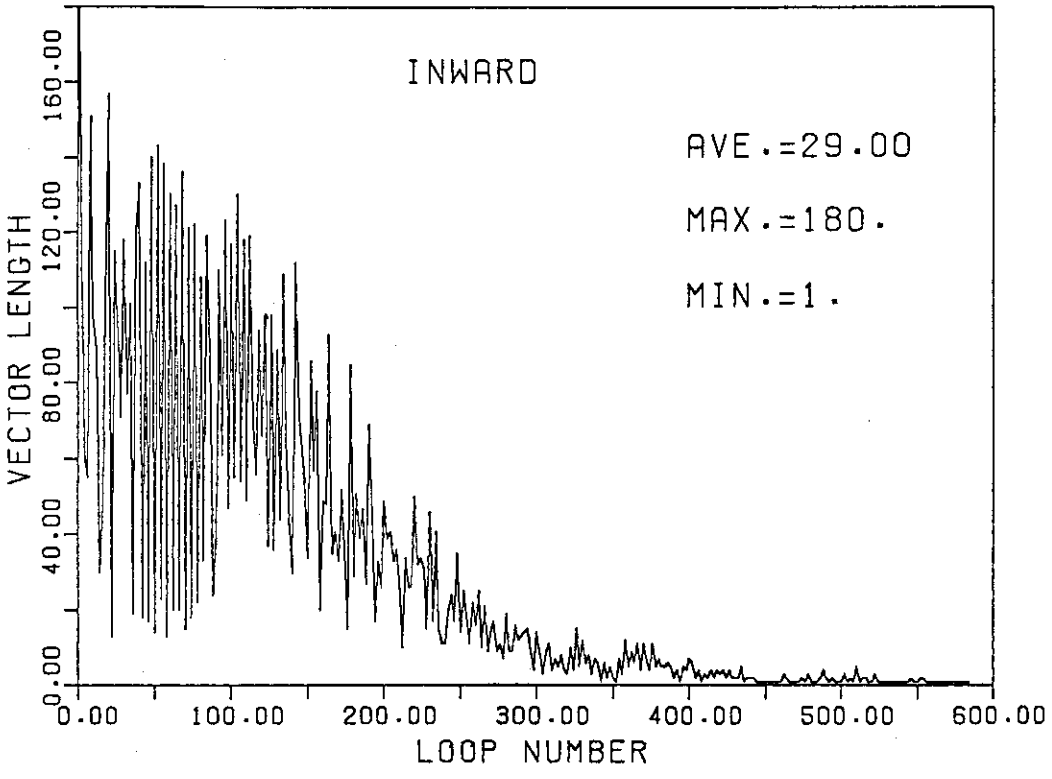

Fig. 12 Number of neutrons vs. MCS(INWARD bank)

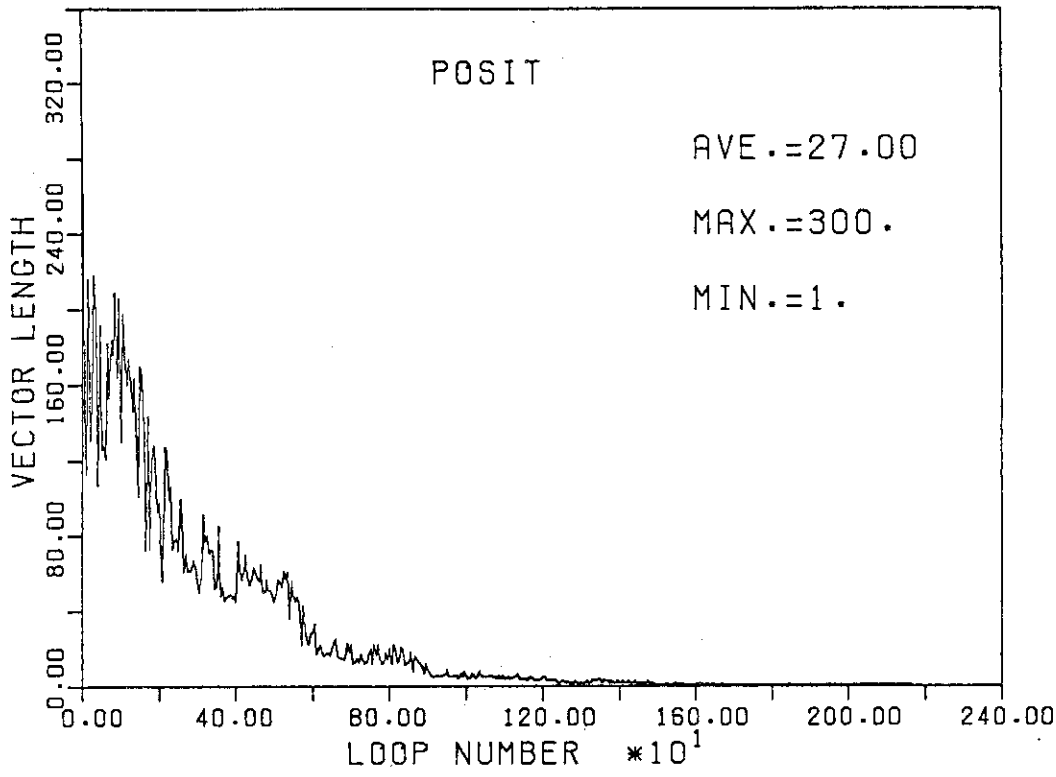

Fig. 13 Number of neutrons vs. MCS(POSIT bank)

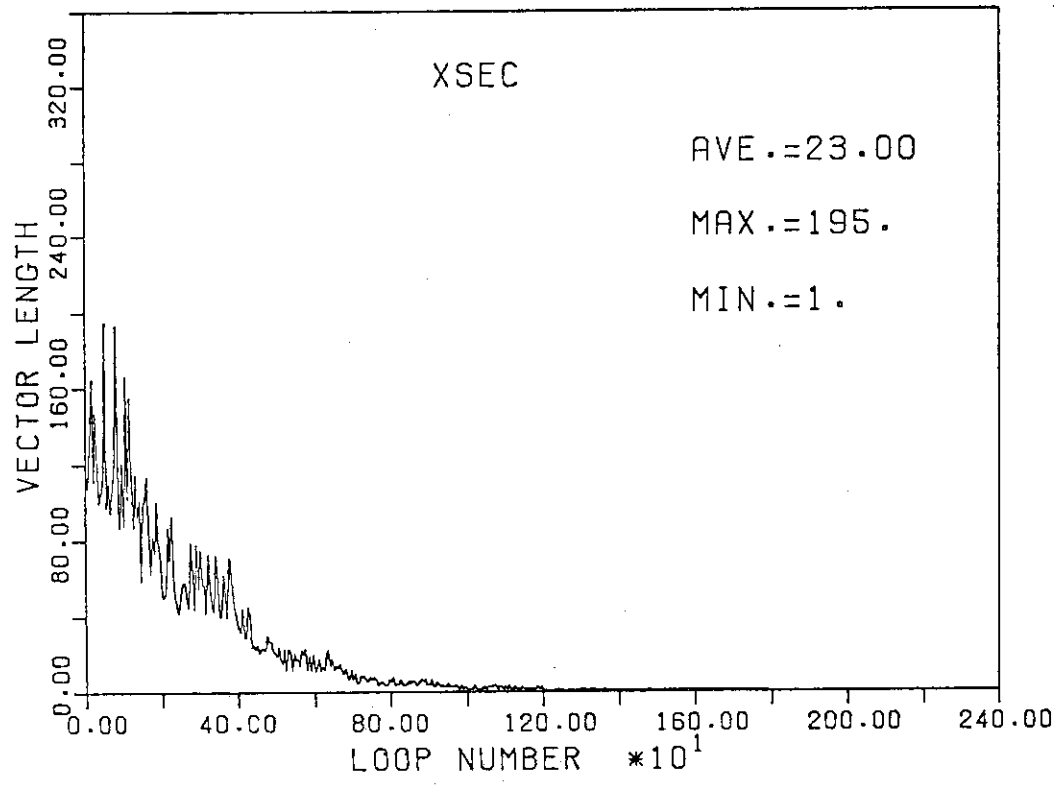

Fig. 14 Number of neutrons vs. MCS(XSEC bank)

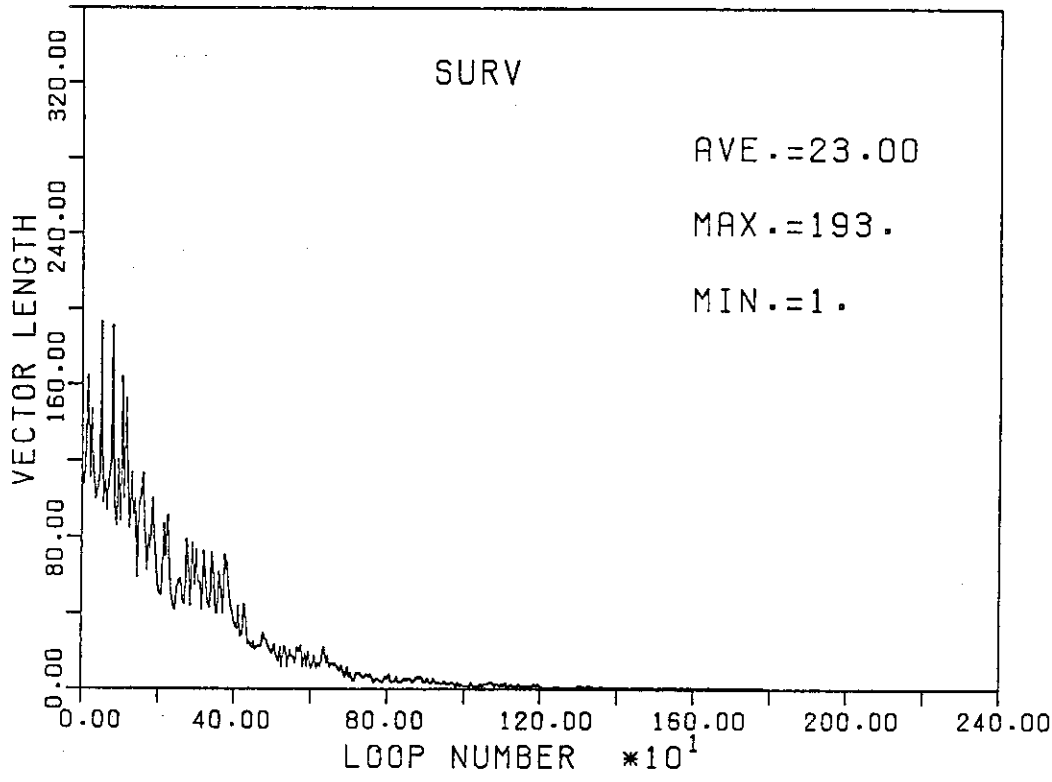

Fig. 15 Number of neutrons vs. MCS(SURV bank)

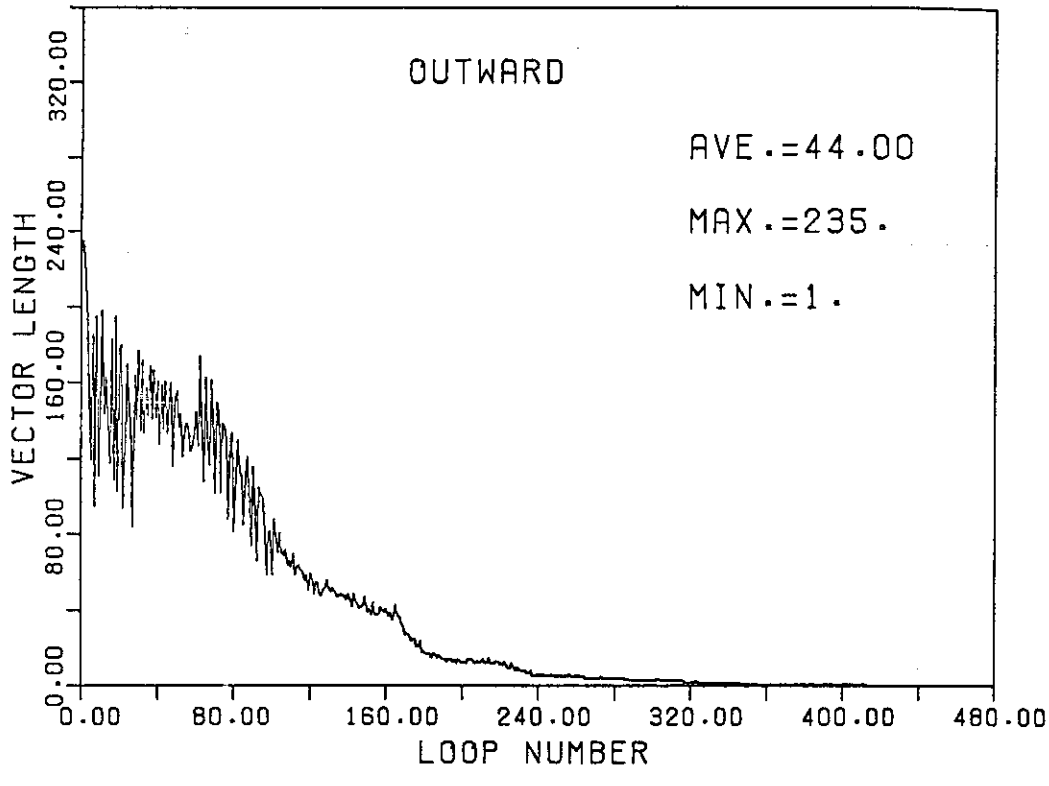

Fig. 16 Number of neutrons vs. MCS(OUTWARD bank)

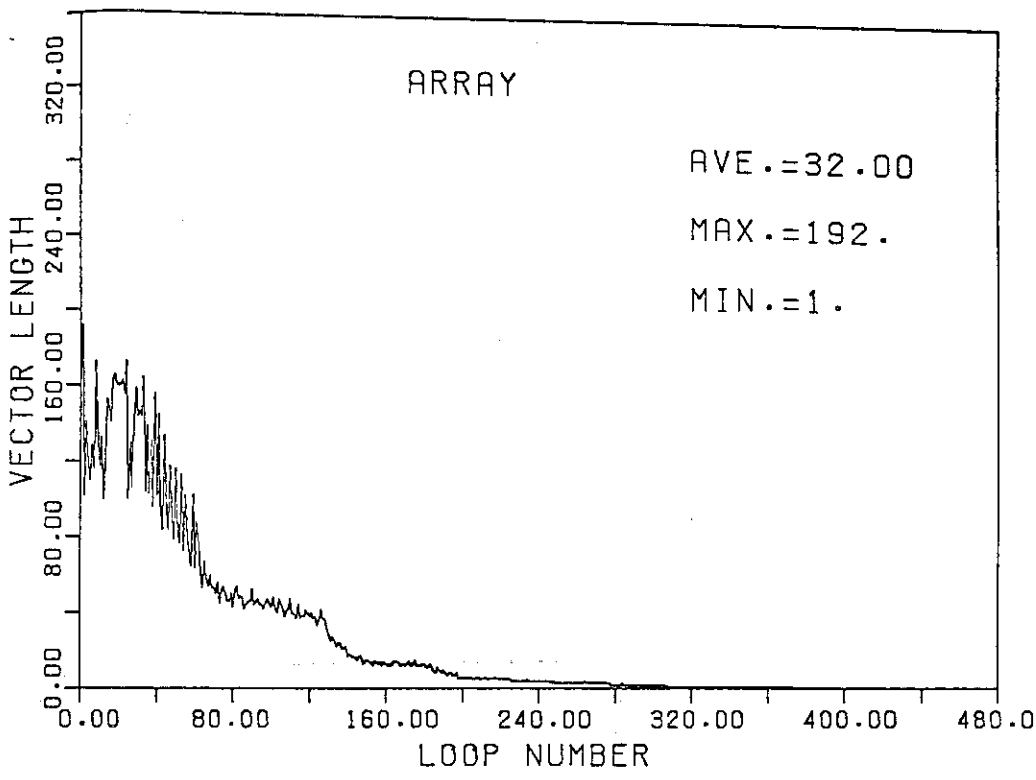

Fig. 17 Number of neutrons vs. MCS(ARRAY bank)

number between 0 and 1. Figure 19-a shows a sequence of source statements for calculation of the scattering angles used in the original scalar KENO IV. In our vectorized version, the sequence of statements have been changed as shown in Fig. 19-b. It may seem rather complicated since it is separated in three parts to isolate the execution of statements corresponding to physically rare events. For example, the value of 1-cos(ψ)**2 rarely becomes less than or equal to zero, so that it is better to execute a sequence of statements by excluding conditional jumps beforehand as in the DO-loop 9918 in the Fig. 19-b.

F. Selection of an Energy Group after Collision

The selection of a new energy group, i.e., the energy group j corresponding to the scattering cross section $\Sigma(i \rightarrow j)$ from the group i to j, is a rather time-consuming task and the original algorithm is hard to vetorize(Fig. 21-a). Since this type of table search is often used in the Monte Carlo computation for neutron transport, it will be better to make mention of a high speed technique. Let p_i be the probability of the occurrence of $X = x_i$. The problem to be solved is to get an integer k such that

$$x_1,\ 0<R<p_1,$$
$$x_2,\ p_1<R<p_2,$$
$$\cdot \quad \cdot$$
$$x_k,\ p_1+\ \ldots\ +p_k<R<1,$$

where $p_1+\ \ldots\ +p_k = 1$ and R is a uniform random number between 0 and 1. The binary search method of A.L. Walker[8], which is also found independently and first introduced by F.B. Brown[9] in vectorization, is also used in our version. The method of A. J. Walker and F. B. Brown is as follows:

Let K and V be an integer part and a fraction part of k*R, respectively. Then making two tables (P_0, P_1, ..., P_{k-1}) and (Y_0, Y_1, ..., Y_{k-1}), we can get

$$X = \begin{cases} x_{K+1}, & V<P_K, \\ Y_K, & \text{otherwise.} \end{cases}$$

The procedure to establish P_i and Y_i is as follows:

i) If the occurrences of (X_i) are all the same, then they have equi-probability 1/k. However in most cases, for some $p_i < 1/k$ and $p_j > 1/k$.

ii) Make a list (p_1, 1) (p_2, 2) ... (p_k, k) and sort the list by p_i in ascending order. Let the sorted list be (q_1, a_1) (q_2, a_2) ... (q_k, a_k). If q_1 is not equal to q_k, then $q_1 < 1/k$ and $q_k > 1/k$. Imagine k bins with capacity 1/k. The bin corresponding to (q_1, a_1) contains q_1 and it can be added 1/k -

$$u' = u\cos\psi - \sqrt{1-u^2}\sin\psi\cos\eta$$

$$v' = v\cos\psi + \frac{uv}{\sqrt{1-u^2}}\cos\eta\sin\psi - \frac{w}{\sqrt{1-u^2}}\sin\psi\sin\eta$$

$$w' = w\cos\psi + \frac{uw}{\sqrt{1-u^2}}\cos\eta\sin\psi + \frac{v}{\sqrt{1-u^2}}\sin\psi\sin\eta$$

where:

$\sin\psi = \sqrt{1-\bar{\mu}^2}$

$\cos\psi = \bar{\mu}$ = cosine of the scattering angle

η = a random azimuthal angle between 0 and 2π

Fig. 18 Direction cosines after collision

```
      IF(MUBAR(IZ+IGKR).NE.0.0) GO TO 840
      CALL GTISO(U,V,W)
      GO TO 890
  830 LAB = .FALSE.
      GO TO 890
  840 FMU=MUBAR(IZ+IGKR)
      SINPSI = SQRT(1.0-FMU*FMU)
      CALL AZIRN(SINETA,COSETA)
      STHETA = SQRT(1.0-U*U)
      IF(STHETA)860,860,870
  860 COSPHI = 1.0
      SINPHI = 0.0
      GO TO 880
  870 COSPHI = V/STHETA
      SINPHI = W/STHETA
  880 V = V*FMU+U*COSPHI*COSETA*SINPSI-SINPHI*SINPSI*SINETA
      W = W*FMU+ U*COSETA*SINPHI*SINPSI+COSPHI*SINPSI*SINETA
      U = U*FMU-COSETA*SINPSI*STHETA
C*****************************END XSEC*****************************
```

Fig. 19-a Scattering angles(original)

```
      IMU=0
      ISTHE=0
*VOCL LOOP,NOVREC
      DO 9916 IV=1,NSURV
      JV=SURVBA(IV)
      IF(MUBARX(IV).EQ.0.0) IMU=IMU+1
      IF(1.0-U  (JV)*U  (JV).LE.0.0) ISTHE=ISTHE+1
 9916 CONTINUE
      IF(IMU.GT.0.OR.ISTHE.GT.0) GO TO 9917
*VOCL LOOP,NOVREC
      DO 9918 IV=1,NSURV
      JV=SURVBA(IV)
        FMU=MUBARX(IV)
        SINPSI=SQRT(1.0-FMU*FMU)
C..     CALL AZIRN(SINETA(JV),COSETA(JV))
        SINETA=SIN(PI2*WK1(IV))
        COSETA=COS(PI2*WK1(IV))
        STHETA=SQRT(1.0-U  (JV)*U  (JV))
          COSPHI=V(JV)/STHETA
          SINPHI=W(JV)/STHETA
        V(JV)=V(JV)*FMU+U(JV)*COSPHI*COSETA*SINPSI
     +         -SINPHI*SINPSI*SINETA
        W(JV)=W(JV)*FMU+U(JV)*COSETA*SINPHI*SINPSI
     +         +COSPHI*SINPSI*SINETA
        U(JV)=U(JV)*FMU-COSETA*SINPSI*STHETA
 9918 CONTINUE
      GO TO 9919
 9917 CONTINUE
*VOCL LOOP,NOVREC
      DO 9912 IV=1,NSURV
      JV=SURVBA(IV)
*VOCL STMT,IF(100)
      IF(MUBARX(IV).NE.0.0) THEN
        FMU=MUBARX(IV)
        SINPSI=SQRT(1.0-FMU*FMU)
C..     CALL AZIRN(SINETA(JV),COSETA(JV))
        SINETA=SIN(PI2*WK1(IV))
        COSETA=COS(PI2*WK1(IV))
        STHETA=SQRT(1.0-U  (JV)*U  (JV))
*VOCL STMT,IF(100)
        IF(STHETA.GT.0.0) THEN
          COSPHI=V(JV)/STHETA
          SINPHI=W(JV)/STHETA
        ELSE
          COSPHI=1.0
          SINPHI=0.0
        ENDIF
        V(JV)=V(JV)*FMU+U(JV)*COSPHI*COSETA*SINPSI
     +         -SINPHI*SINPSI*SINETA
        W(JV)=W(JV)*FMU+U(JV)*COSETA*SINPHI*SINPSI
     +         +COSPHI*SINPSI*SINETA
        U(JV)=U(JV)*FMU-COSETA*SINPSI*STHETA
      ENDIF
 9912 CONTINUE
*VOCL LOOP,SCALAR
      DO 9920 IV=1,NSURV
      JV=SURVBA(IV)
      IF(MUBARX(IV).EQ.0.0) THEN
C..     CALL GTISO(U(JV),V(JV),W(JV))
        LOCR=LOCR+1
        WK2(IV)=RR(LOCR)
        WKS1=SIN(PI2*WK1(IV))
        WKC1=COS(PI2*WK1(IV))
        WKS2=2.0*WK2(IV)-1
        WKC2=SQRT(1-WKS2*WKS2)
        U(JV)=WKC2*WKC1
        V(JV)=WKC2*WKS1
        W(JV)=WKS2
      ENDIF
 9920 CONTINUE
```

Fig. 19-b Scattering angles(vectorized)

q_l amount of value of q_k. So we set n = k, P_t = k*q_l and Y_t = x_m, where t = (a_l -1) and m = a_n, respectively. Remove (q_l, a_l) and (q_n, a_n) from the list and insert (q_n -(1/k - q_l), n) into the appropriate position of the list.

iii) Set n = k-1 and repeat the procedure until n becomes zero.

Note that in the above procedure the value K and V play roles of key indices for the search. If we use only V = k*R = R, i.e., k=1, we must use the usual time-consuming sequential search.

A computer program for making the tables is shown in the reference(8).The algorithm is shown in the references(8,9,10). In the rewritten algorithm of Fig. 21-b, JV is the neutron number, IG is the energy group, NDS is the number of downscatters and also is k of the above procedure. R is a random number. IZZ is K, R2 is V, EFSP is P_k, and IFSP is Y_k. In the original scalar code, in the case of Problem 2, the searches are executed 3411*10^4 times, in which average 6 searches are made for one neutron JV and about 10% of computation time is consumed in the searches. The search time has been drastically reduced by the introduction of the technique.

```
  790 R=FLTRN(0)
      IGKR=(KR-1)*NDSNGP+(IG-1)*NDS
      DO 800 IZ=1,NDS
  800 IF(R.LE.FSP(IZ+IGKR)) GO TO 810


      WRITE(OUTPT,11800) KR,IG,IGKR,NDS,FSP(NDS+IGKR),R
11800 FORMAT('0***** ERROR IN DOWNSCATTERS,KR= ',I5,' IG= ',I5,
     1' IGKR= ',I5,' NDS= ',I5,' FSP= ',1PE15.8,' R= ',E15.8)
      STOP
  810 IG=IZ+IG-1
      IF(IG.GT.NGP)IG=IGOLD-(IG-NGP)
```

Fig. 21-a Selection of the energy group(original)

```
*VOCL LOOP,NOVREC
      DO 9906 IV=1,NSURV
      JV=SURVBA(IV)
 9906 IGKR(JV)=(KR(JV)-1)*NDSNGP+(IG(JV)-1)*NDS
      DO 9910 IV=1,NSURV
      JV=SURVBA(IV)
C9910 R(JV)=FLTRN(0)
 9910 R(JV)=RR(LOCR+IV)
      LOCR=LOCR+NSURV
*VOCL LOOP,NOVREC
      DO 9907 IV=1,NSURV
      JV=SURVBA(IV)
      IZZ(JV)=NDS*R(JV)
      R2=NDS*R(JV)-IZZ(JV)
      IF( R2.GE.EFSP(IZZ(JV)+IGKR(JV))) IZZ(JV)=IFSP(IZZ(JV)+IGKR(JV))
      IG(JV)=IZZ(JV)+IG(JV)-1
 9907 CONTINUE
```

Fig. 21-b Selection of the energy group(vectorized)

G. Isolation of Rare Events

Although most neutrons in a bank follow the same computational algorithms in an event, one or a few neutrons occasionally require special treatments. We can see it, for example, in the case of the computations of scattering angles(Fig. 19) and the flight distance(Fig. 8). In the scattering angle computation in the case of Problem 2, the case, when STETA is equal to zero only occurs one time, whereas the nonzero case occurs about $567*10^4$ times. In the flight distance computation, the new position of a neutron is slightly modified when the computed coordinates are the same as the old ones. For example, in the case of Problem 2, the statements in the DO-loop in Fig.8-b are executed about 1357 x 10^4 times, whereas the statements

```
X1(JV)=X(JV)+SIGN(X(JV), U(JV))*(1.0E-6)
Y1(JV)=· · · ·
Z1(JV)=· · · ·
```

are executed only 1850 times in total. Therefore the information

```
*VOCL STMT IF(0)· · · ·
```

is fed in to let the vector compiler know a percentage of true-branches of the IF-statement just after this information. If the percentage is of high value, the compiler generates instructions with masking operations. Otherwise it generates instructions with gather/scatter or list vector operations. It generates masking sets and instructions on register chaining for relating statements before execution. Furthermore a DO-loop requires prerequisite address calculations before the initialization of execution even for variables in statements which are rarely used. The vector compiler is usually designed to optimize the DO-loop entirely. Thus it is very important for a vectorized Monte Carlo code to separate rarely used statements from a DO-loop and isolate them so as not to be used in most computations. An example of separation and isolation is shown in Fig. 20. The same technique is used in computations of trials for the Russian roulette in Phase 1 and for most computations in Phase 2 vectorization.

```
*VOCL LOOP,NOVREC
      DO 9321 IV=1,NPATH
      JV=PATHBA(IV)
C     IF (RPTH(JV).LE.0.0 ) RPTH(JV) = EXPRN(0)
                            RPTH(JV) = -LOG(  RR(LOCR+IV)  )
 9321 CONTINUE
           LOCR=LOCR+NPATH
*VOCL LOOP,NOVREC
      DO 9320 IV=1,NPATH
      JV=PATHBA(IV)
      KR(JV)=MAT(K(JV))
*VOCL STMT,IF(99)
      IF(KR(JV).NE.0) THEN
      PTH(JV)=RPTH(JV)*RSIGT(KR(JV),IG(JV))
      ELSE
      PTH(JV)=BIG
      ENDIF
      X1(JV)=X(JV)+PTH(JV)*U(JV)
      Y1(JV)=Y(JV)+PTH(JV)*V(JV)
      Z1(JV)=Z(JV)+PTH(JV)*W(JV)
      XYZ(IV)=(X(JV)-X1(JV))*(Y(JV)-Y1(JV))*(Z(JV)-Z1(JV))
 9320 CONTINUE
      IXYZ=0
*VOCL LOOP,NOVREC
      DO 9323 IV=1,NPATH
      IF(XYZ(IV).EQ.0) IXYZ=IXYZ+1
 9323 CONTINUE
      IF(IXYZ.EQ.0) GO TO 9324
*VOCL LOOP,SCALAR
      DO 9322 IV=1,NPATH
      JV=PATHBA(IV)
*VOCL STMT,IF( 0)
      IF(X(JV).EQ.X1(JV)) X1(JV)=X(JV)+SIGN(X(JV),U(JV))*(1.0E-6)
*VOCL STMT,IF( 0)
      IF(Y(JV).EQ.Y1(JV)) Y1(JV)=Y(JV)+SIGN(Y(JV),V(JV))*(1.0E-6)
*VOCL STMT,IF( 0)
      IF(Z(JV).EQ.Z1(JV)) Z1(JV)=Z(JV)+SIGN(Z(JV),W(JV))*(1.0E-6)
 9322 CONTINUE
 9324 CONTINUE
```

Fig. 20 Isolation in flight distance

H. Simultaneous Processing of Generations

In KENO IV usually 300 neutrons are used in one generation. The number of neutrons, however, decreases rapidly through migrations. We have shown the relation of the vector length, i.e., number of neutrons and the number of iterations concerning PATH bank for the flight computation in Fig. 11. There are two ways to keep the vector length as long as possible. One is to use many neutrons in a generation and less generations in a run. The other is to do simultaneous processing of generations. At first we have tried to use the former technique and used 600 neutrons or 900 neutrons in a generation. In our vectorized version, however, the percentage of scalar operations is high, and as shown in Fig. 11, the number of neutrons decreases very rapidly, so that mere use of many neutrons, i.e., extending the vector length results in no speedup of the code. The current vectorized version has been modified for the latter technique. Because of restriction of memory size our vectorized version processes ten generations simultaneously.

In this case the relation of the number of neutrons vs. Monte Carlo steps changes as shown in Fig. 22. The simultaneous processing of generations has induced unwelcomed recurrence relations such as

```
total.fission(generation(J))=total.fission(generation(J)) +· · · ·
```

With the procedures in Phase 1 the vectorized version has attained the same performance as the original scalar one (see Table 1 for Problem 1 and Table 2 for Problem 2). The form of source program in this phase was the same as the original one except that variables in the vector version have subscripts. As seen in Table 1 and 2, there is no difference in the performances for simple and complex geometries. This is because that the most percentage of the computation time has been consumed in checking and branching operations for the geometry and events. Although the geometry and events are very simple in the case of Problem 1, the algorithms in the KENO IV code retain a generality for complex regions unless it is modified for a specific input data. If we wish to obtain a good performace on a Monte Carlo transport code, we must tailor the code to some specific geometry. This is a dilemma we must face with at present. In Phase 3 we will show a performance obtained by pruning off the unnecessary statements for the computation of Problem 1.

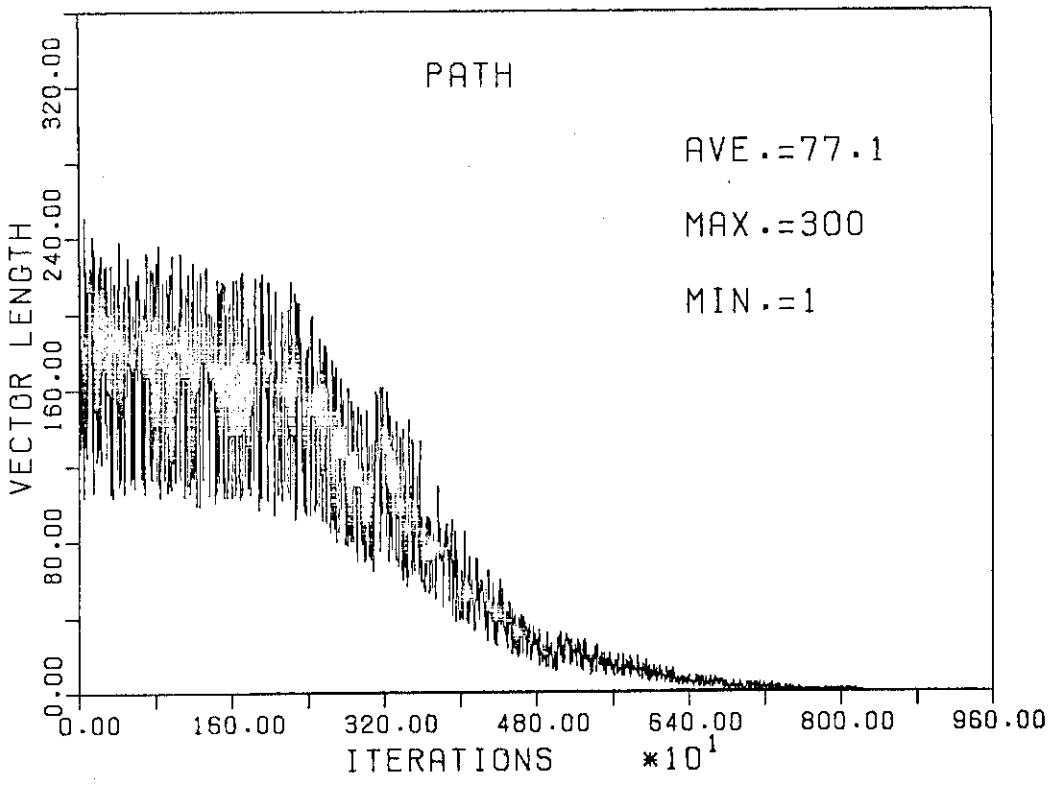

Fig. 22-a Number of neutrons vs. MCS in ten generations (PATH bank)

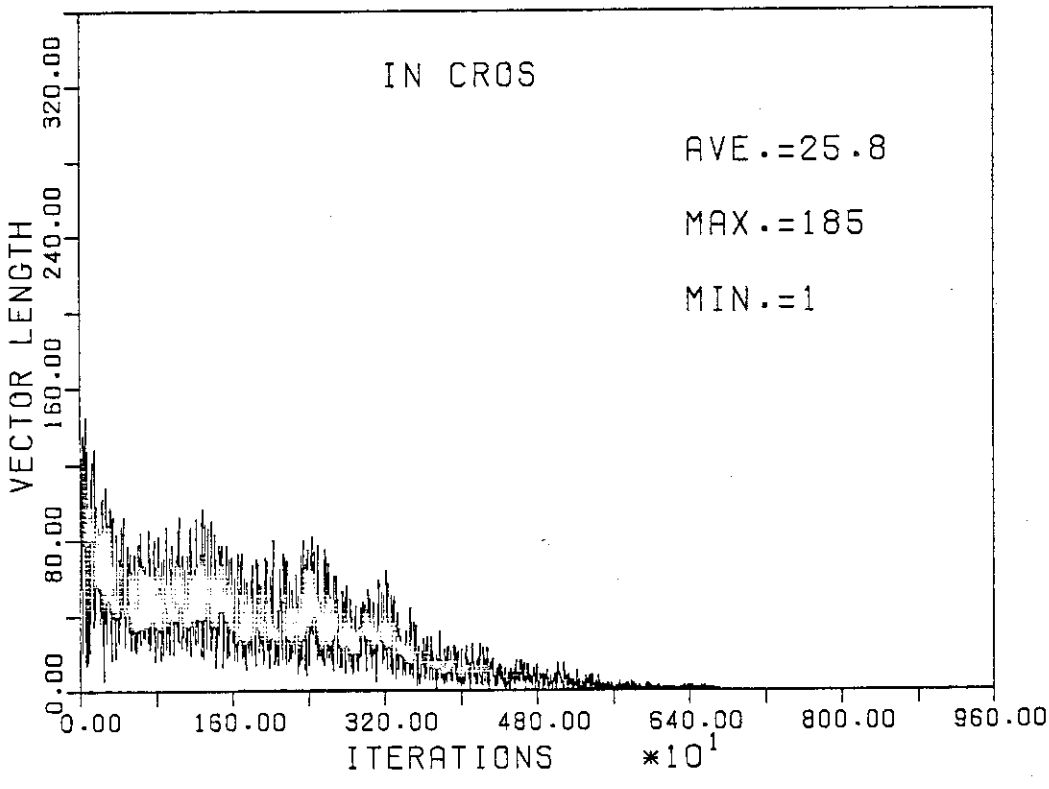

Fig. 22-b Number of neutrons vs. MCS in ten generations (INWARD bank)

Table 1 Result of Phase 1 vectorization for Problem 1

Time (sec.) / Event	Vectorized Version		
	Vector Processor (VP-100)		Scalar Processor (M-380)
	Vector Time	Scalar Time	Scalar Time
PATH	1. 4	4. 8	10. 8
Inward Cross	———	———	———
POSIT	———	5. 4	5. 0
Outward Cross	0. 0	0. 0	0. 0
ARRAY	———	———	———
XSEC	7. 1	14. 0	39. 9
Random Number	0. 3	———	3. 3
Total Time	33. 0 (sec.)		59. 0 (sec.)
Speedup Ratio	1. 4		0. 78

Note. 1) M-380 has performance of 25 MIPS or 9 MFLOPS.
2) Scalar performance of VP-100 is same as M-380.
3) Original scalar version for problem 1 required 46 seconds on M-380.
4) The zero value means a negligibly small computation time.

Table 2 Result of Phase 2 vectorization for Problem 2

Time (sec.) / Event	Vectorized Version		
	Vector Processor (VP-100)		Scalar Processor (M-380)
	Vector Time	Scalar Time	Scalar Time
PATH	14. 2	51. 4	105. 2
Inward Cross	46. 2	13. 8	70. 6
POSIT	———	48. 8	31. 7
Outward Cross	29. 2	20. 8	62. 8
ARRAY	5. 2	31. 7	41. 0
XSEC	38. 0	87. 6	249. 0
Random Number	2. 0	———	23. 3
Total Time	388. 9 (sec.)		583. 6 (sec.)
Speedup Raito	0. 98		0. 65

Note. 1) Original scalar version for the problem 2 required 380 seconds on M-380.

4. VECTORIZATION OF PHASE 2

A. Further Isolations of Rare Events

In the second phase we have adopted extensively the isolation techniques described in Chap. 3, G. Many tests were made whether statements concerning with every bank or subbank should be executed in the scalar or vector mode. Nested IF-statements for bank selection after the event are separated to reduce the amount of masking operations in the vector mode. For example, the boundary cross checking operation for top side of a region in the subroutine CROSS was isolated because in a criticality safety computation usually most neutrons did not cross the geometric top side region.

B. Computation of Two Dimensional Subscripts

The subscript address computations are done in the scalar mode. For the variables of two dimensional subscripts such as group and region-wise fluxes, absorptions, etc., it is faster to reduce the variables to one dimensio and to compute the subscripts in the vector mode(Now the compiler has been revised to vectorize multi-dimensional subscript address computation and the speed is almost same as that of the hand coded one).

By the above two techniques we have gained 20% improvement of performance. This means that the code has become 1.2 times faster than the original scalar version.

C. Time Reduction by Neutron Cutoff

At this stage we have introduced a time reduction technique which may cause much controversy. As seen in Fig. 22, during the migrations of neutrons, most neutrons diminish rapidly, whereas some neutrons survive for a long period. If the total number of surviving neutrons in a geometric system is small, say 30 for example, the number of neutrons in every bank would be < 10 and we cannot expect any merit of vector processing. As one of solutions to this problem we have tried to discard a small number of neutrons, 60 in our cases, in every ten generations. The number of remaining neutrons in each generation are counted and k_{eff} fluxes, fissions, absorptions, etc. of the generation are adjusted using the discarded number in each generation.

This technique may physically cause an under- or overestimate of k_{eff}, etc. Because of discarding neutrons we are losing possible fissions(underestimate) and making adjustment of the subtraction we are counting fictitious fissions

of unimportant neutrons(overestimate). In the case of Problem 2, it seems that we are slightly overestimating the k_{eff} (see Table 3). This technique might be used under the assumption that the under- and overestimate cancel each other. The reductions of computation time using this Phase 2 technique are shown in Table 4. The version in this phase has become faster than the original scalar code by a factor of 1.4. The form of source program has been changed and has become somewhat dirty. If we can use unlimited amount of memory we have no need to use this technique.

As for the vectorization up to Phase 2, the authors have described elsewhere comparing the performance of the vectorized version with the the results of original scalar version on FACOM M-380 scalar computer which has almost same performance as the VP-100 scalar processing unit.[3]

A brief summary up to Phase 2 vectorization is already described elsewhere.[11]

Table 3 Neutron cut off vs. non-cut off K-effective computations

	Neutron non Cut-off	Neutron Cut-off
Average Keff	9.96670E-01	1.00210E-00
Deviation	2.90932E-03	3.49249E-03

Table 4 Time reduction in Phase 2 for Problem 2

Time (sec.) / Event	Vectorized Version	
	Vector Processor (VP-100)	
	Cut-off	Non Cut-off
PATH	32.4	34.8
Inward Cross	33.5	36.1
POSIT	43.2	45.6
Outward Cross	34.8	38.7
ARRAY	30.6	33.6
XSEC	86.0	107.9
Random Number	1.9	2.0
Total Time	262.4 (sec.)	285.7 (sec.)
Speedup Ratio	1.45	1.33

5. VECTORIZATION OF PHASE 3

In the third phase we have simplified the vectorized version to a specific geometry. As seen in Tables 1 and 2, unvectorizable parts are those which select geometric treatments of neutrons and events next to come. We can expect a rather good performace if we use a version specific to a geometry. To confirm this for the Problem 1, a simplified vectorized version has been made by deleting and modifying Phase 2 version.

A. Deletion of Geometry Selection

A neutron is checked after the transport calculation if it remains in the same region or not. This is done by checking if their new coordinates are in or out of the old region as in Fig. 23-a. The "computed go to" statement generates the most efficient computer instructions in a case of scalar operation, but it is an obstacle to vectorization. In Problem 1, however, we need only sphere geometry and the vectorized version in Fig. 23-b may be simplified as in Fig. 23-c. The same technique is used in the subroutine CROSS where neutrons are checked if they have crossed region boundaries.

B. Deletion of Event Selection

The event selection, after neutrons have been checked for boundary crossing, as seen in Fig. 24-a, may be simplified as in Fig. 24-b for Problem 1.

C. Deletion of Flux and Fission Density Calculations

Calculations of flux, fission density, neutron life time, etc. are deleted, assuming they are less important in Problem 1.

D. Deletion of Unnecessary Conditional Branches

Some conditional branches for material, cell recognition, etc. have been deleted because they were needless in the Problem 1.

By these simplifications we have obtained a rather good performace, as shown in Table 5. This shows what are the obstacles against vector processing of the Monte Carlo neutron transport computation.

```
C*****************************START POSIT************************
  430 IGEO= IGEOM(K)
      GO TO(460,450,470,460,462,464,466,467),IGEO
  450 RSQ=X1*X1+Y1*Y1
      IF(Z1.LE. XX(K,2)    .AND.Z1.GE. XX(K,3)    .AND.RSQ.
     1 LE. XX(K,4))     GO TO 740
      GO TO 480
  460 IF(X1.LE. XX(K,1)    .AND.X1.GE. XX(K,2)    .AND.Y1.LE.
     1  XX(K,3)    .AND.Y1.GE. XX(K,4)    .AND.Z1.LE. XX(K,5)    .
     2 AND.Z1.GE. XX(K,6))     GO TO 740
      GO TO 480
  470 RSQ=X1*X1+Y1*Y1+Z1*Z1
      IF(RSQ.LE. XX(K,2))     GO TO 740
      GO TO 480
  462 RSQX=Y1*Y1+Z1*Z1
      IF(X1.LE.XX(K,2).AND.X1.GE.XX(K,3).AND.RSQX.LE.XX(K,4)) GO TO 740
      GO TO 480
  464 RSQY=X1*X1+Z1*Z1
      IF(Y1.LE.XX(K,2).AND.Y1.GE.XX(K,3).AND.RSQY.LE.XX(K,4)) GO TO 740
      GO TO 480
  466 TZ1=Y1*XX(K,3)
      IF(ABS(XX(K,3)).GE.5.0)TZ1=Z1*XX(K,3)
      IF(ABS(XX(K,3)).LE.2.0)TZ1=X1*XX(K,3)
      IF(( X1*X1+Y1*Y1+Z1*Z1).LE.XX(K,2).AND.TZ1.GE.0.0) GO TO 740
      GO TO 480
  467 NHCYL=ABS(XX(K,5))
      SGN=SIGN(1.0,XX(K,5))
      GO TO(471,472,473,474,475,476),NHCYL
  471 IF(SGN*X1.LT.0.0)GO TO 480
      GO TO 450
  472 IF(SGN*Y1.LT.0.0)GO TO 480
      GO TO 450
  473 IF(SGN*Z1.LT.0.0)GO TO 480
      GO TO 462
  474 IF(SGN*Y1.LT.0.0)GO TO 480
      GO TO 462
  475 IF(SGN*Z1.LT.0.0)GO TO 480
      GO TO 464
  476 IF(SGN*X1.LT.0.0)GO TO 480
      GO TO 464
C*****************************END POSIT**************************
```

Fig. 23-a POSIT selection(original)

```
      DO 9431 IV=1,NPOS
      JV=POSBAN(IV)
  430 IGEO(JV)= IGEOM(K(JV))
      GO TO(460,450,470,460,462,464,466,467),IGEO(JV)
  450 RSQ=X1(JV)*X1(JV)+Y1(JV)*Y1(JV)
      IF(Z1(JV).LE. XX(K(JV),2)    .AND.Z1(JV).GE. XX(K(JV),3)
     1    .AND.RSQ.LE. XX(K(JV),4))     GO TO 739
      GO TO 479
  460 IF(X1(JV).LE. XX(K(JV),1)
     +    .AND.X1(JV).GE. XX(K(JV),2)    .AND.Y1(JV).LE.
     1  XX(K(JV),3) .AND.Y1(JV).GE. XX(K(JV),4)
     +                                    .AND.Z1(JV).LE.XX(K(JV),5)
     2 .AND.Z1(JV).GE. XX(K(JV),6))     GO TO 739
      GO TO 479
  470 RSQ=X1(JV)*X1(JV)+Y1(JV)*Y1(JV)+Z1(JV)*Z1(JV)
      IF(RSQ.LE. XX(K(JV),2))     GO TO 739
      GO TO 479
  462 RSQX=Y1(JV)*Y1(JV)+Z1(JV)*Z1(JV)
      IF(X1(JV).LE.XX(K(JV),2).AND.X1(JV).GE.XX(K(JV),3).AND.
     +    RSQX.LE.XX(K(JV),4)) GO TO 739
     GO TO 479
  464 RSQY=X1(JV)*X1(JV)+Z1(JV)*Z1(JV)
      IF(Y1(JV).LE.XX(K(JV),2).AND.Y1(JV).GE.XX(K(JV),3).AND.
     +    RSQY.LE.XX(K(JV),4)) GO TO 739
      GO TO 479
  466 TZ1=Y1(JV)*XX(K(JV),3)
      IF(ABS(XX(K(JV),3)).GE.5.0)TZ1=Z1(JV)*XX(K(JV),3)
      IF(ABS(XX(K(JV),3)).LE.2.0)TZ1=X1(JV)*XX(K(JV),3)
      IF(( X1(JV)*X1(JV)+Y1(JV)*Y1(JV)+Z1(JV)*Z1(JV)).LE.XX(K(JV),2)
     +     .AND.TZ1.GE.0.0) GO TO 739
      GO TO 479
  467 NHCYL=ABS(XX(K(JV),5))
      SGN=SIGN(1.0,XX(K(JV),5))
      GO TO(471,472,473,474,475,476),NHCYL
  471 IF(SGN*X1(JV)..T.0.0)GO TO 479
      GO TO 450
  472 IF(SGN*Y1(JV).LT.0.0)GO TO 479
      GO TO 450
  473 IF(SGN*Z1(JV).LT.0.0)GO TO 479
      GO TO 462
  474 IF(SGN*Y1(JV).LT.0.0)GO TO 479
      GO TO 462
  475 IF(SGN*Z1(JV).LT.0.0)GO TO 479
      GO TO 464
  476 IF(SGN*X1(JV).LT.0.0)GO TO 479
      GO TO 464
  739 NXSEC=NXSEC+1
      XSECBA(NXSEC)=JV
      GO TO 9431
  479 NCROS2=NCROS2+1
      CROSB2(NCROS2)=JV
 9431 CONTINUE
```

Fig. 23-b POSIT selection(vectorized)

```
*VOCL LOOP,NOVREC
V       DO 9431 IV    = 1,NPOS
V          JV         = POSBAN(IV)
V          IGEO(JV)   = IGEOM(K(JV))
V          RSQ        = X1(JV)*X1(JV) + Y1(JV)*Y1(JV) + Z1(JV)*Z1(JV)
V          IF( RSQ.LE.XX(K(JV),2) ) THEN
V             NXSEC   = NXSEC + 1
V             XSECBA(NXSEC) = JV
V          ELSE
V             NCROS2 = NCROS2 + 1
V             CROSB2(NCROS2) = JV
V          END IF
V  9431 CONTINUE
```

Fig. 23-c POSIT selection(vectorized and simplified)

```
*VOCL LOOP,SCALAR
      DO 9482 IV=1,NCROS2
      JV=CROSB2(IV)
*VOCL STMT,IF( 0)
      IF ( LCOR(JV) ) THEN
*VOCL STMT,IF( 0)
        IF(K(JV).GT.KREFM) THEN
          NALBED=NALBED+1
          ALBEDB(NALBED)=JV
        ELSE
          IC(JV)=.FALSE.
          NPATH=NPATH+1
          PATHBA(NPATH)=JV
        ENDIF
      ELSE
*VOCL STMT,IF(35)
        IF(K(JV).LE.K2(JV)) THEN
          IC(JV)=.FALSE.
          NPATH=NPATH+1
          PATHBA(NPATH)=JV
        ELSE
*VOCL STMT,IF( 0)
          IF ( LSGUN )THEN
            NALBED=NALBED+1
            ALBEDB(NALBED)=JV
          ELSE
            K(JV)=K(JV)-1
            NARRAY=NARRAY+1
            ARRAYB(NARRAY)=JV
          ENDIF
        ENDIF
      ENDIF
 9482 CONTINUE
```

Fig. 24-a Event selection(vectorized)

```
      IF( LSGUN ) THEN
  *VOCL LOOP,NOVREC
V           DO 99482 IV = 1,NCROS2
V              JV       = CROSB2(IV)
V              NALBED   = NALBED + 1
V              ALBEDB(NALBED) = JV
V 99482     CONTINUE
      ELSE
  *VOCL LOOP,NOVREC
V           DO 9482 IV = 1,NCROS2
V              JV       = CROSB2(IV)
V              K(JV)    = K(JV) - 1
V              NARRAY   = NARRAY + 1
V              ARRAYB(NARRAY) = JV
V  9482     CONTINUE
      END IF
```

Fig. 24-b Event selection(simplified for Phase 3)

Table 5 Time reduction and speedup ratio of the simplified version

Time (sec.) / Event	Vectorized Version		
	Vector Processor (VP-100)		Scalar Processor (M-380)
	Vector Time	Scalar Time	Scalar Time
PATH	1. 2	4. 8	7. 5
Inward Cross	———	———	———
POSIT	0. 6	———	2. 3
Outward Cross	0. 0	0. 0	0. 0
ARRAY	———	———	———
XSEC	3. 8	9. 2	35. 1
Random Number	0. 3	——	3. 3
Total Time	15. 1 (sec.)		48. 2 (sec.)
Speedup Ratio	3. 04		0. 95

Note. 1) Original scalar version for problem 1 required 46 seconds on M-380.
2) The zero value means a negligibly small computation time.

6. FINDINGS

Let us summarize major findings in the process of our vectorization of KENO IV.

A. Expansion of Computation Time

Changes of variables from scalar to vector introduce a considerable amount of overhead time for DO-loop initializations and address calculations. In our case the overhead amounts to nearly 40%. If we cannot reduce the overhead, the vectorized version could only attain two times of performance at best. The isolation technique mentioned above eliminates a part of the overhead.

B. Low Degree of Vectorization

The vectorized version is suffering from a relatively low degree of vectorization. At first the ratio of vectorizable scalar computation time to the total one was estimated to be about 90%. However, measuring the time of each DO-loop, a considerable number of vectorized DO-loops have been changed back to scalar loops. This is due to i) the low computational densities of events which consist of statements intermingled with many IF-branches, ii) the diminishing vector length because of kill and leakage of neutrons, and iii) existences of many subevents which handle small numbers of neutrons.

C. Large Computational Amount for Geometry and Event Recognition

The speedup ratios in the case of Problem 1 in Phase 2 and Phase 3 indicate the large computational amount for geometric and event recognition. This is due to the fact that most event recognitions are unvectorizable. For efficient execution of the Monte Carlo codes on the vector processor, it is necessary to add extra features on the vector processor. This will be discussed in the following Chapter 7.

7. DISCUSSIONS

Following improvements on the compiler and the vector processor will upgrade the performance of the vectorized Monte Carlo code.

A. Mixed Compilation of Scalar and Vector in a DO-loop

As we have seen in the example of Fig. 8-b, the separation of rarely executed statements introduces another DO-loop, and as the result, redundant initialization of the loop and address calculations for variables are needed in the new DO-loop. Nearly 10% of the time of the final vectorized version is consumed for the overhead. Both the auxiliary information

```
*VOCL IF(0)····
```

referred to in Chapter 3, and a new pipeline for event branch proposed below will make the compiler capable of mixed compilation of scalar and vector operations efficiently.

B. Addition of a Pipeline for Geometric Branch

As is shown in Table 6, in our vectorized version about 60% of computation time remains as scalar, i.e., the categories (1) - (4). About 20% of this is due to the following calculation of group and region-wise fissions, absorptions and fluxes.

```
fission(i,k)=fission(i,k)+fission.weight( )*····
absorption(i,k)=absorption(i,k)+···
flux(i,k)=flux(i,k)+····
```

The simultaneous summations over group i and region k contain possible recurrences in vector operations. The remaining forty percentage scalar computation consists mainly of sorting of neutrons by geometries and events. The section POSIT in Table 4 is the most typical example of this case. It will be best illustrated in the form of programming language as follows:

```
      DO 100 I=1,NPOS
      J=POSITBANK(I)
      GO TO (1,2,···· ,8), GEOMETRY(region(K(J)))
    1 Check the position of the neutron J whether it is
      in the sphere region K, and
```

```
    GO TO 20 if it is in, else GO TO 30
  2 Check the position of the neutron J whether it is
    in the cuboid region K, and
    GO TO 20 if it is in, else GO TO 30
      .
      .
      .
 20 NXSEC=NXSEC+1 ; set collision bank
    XSECBANK(NXSEC)=J
    GO TO 100
 30 NCROS2=NCROS2+1 ; set outward crossing bank
    CROS2BANK(NCROS2)=J
100 CONTINUE
```

In the above example the computation of checking may be vectorized with creation of subbanks in scalar operation. However the performance gain by it cannot overcome the overhead introduced by the creation. This difficulty may be resolved if the vector processor has a pipeline for geometric branches. Figure 25 shows the mechanism. The pipeline is divided into twelve segments. Each segment i has a comparator to check whether k=i. If it is true, the segment stores the corresponding value J in its stack and increment the counter by one. The twelfth segment is for special purpose. It is for a value of k which is not equal to 1, 2, · · · · , or 11.

Usually practical geometries are categorized within eleven patterns. Even in a case of more than eleven patterns, contents of the counter and stack of the twelfth segment will be used for further refinement of patterns. Since the number of patterns is often limited to less than eleven, the comparator of every segment should have a switch which let the segment behave like the twelfth segment and stop to send k to the next (i+1)th segment. The switch can be set before the execution of a "computed go to" statement using an instruction generated by the compiler. Non-null operations of contiguous segments are essential for efficiency but the reordering time of k in the beginning of code execution to satisfy such condition is negligibly small. This geometric pipeline can also be used to sort very quickly, numbers k of recurrence relations such as

```
total.fission(generation(J))=total.fission(generation(J)) +· · · ·
```

where k=generation(J). As the result, the vectorization of such recurrence expression becomes very easy and the computation time will be halved. Similar computation can be applied on the recurrence expression such as

flux(group i(J), region k(J))=flux(group i(J), region k(J)) + ····

If a bank index sorted by the group i after sorting of the region k is less than two, the expression has no recurrence for the current collision bank XSECBANK and can be executed in the vector mode.

C. Addition of a Pipeline for Event Branch

A neutron sorted by a geometric pattern is checked whether it is in the region. If it is in the region, it is put into the collision bank and if not, it is put into the region crossing bank. The checking operations themselves are logical or arithmetic and can be vectorized and the results are logical variables of true or false. The creation of banks is executed using a combination of vector expand/compress instructions and a mask bit population count instruction generated by the compiler. However more efficient creation can be done by providing a following simple event pipeline(Fig. 26). It is assumed that a logical variable contains unity if it is true and zero otherwise.

For both pipeline operations the so-called vector registers may be used as the increment counters and stacks. After the operations the contents of registers can be stored into bank indices and banks.

All of arithmetic, logical, geometric and event pipelines can work in parallel.

The control of execution path, execution of the Russian roulette, storing and issuing a seed neutron will remain in scalar, which may consume 20% of the computation time. After the implementation of the two Monte Carlo pipelines we can vectorize the remaining 80% in which half is consumed by recurrence expressions. We may expect twice speedup for the recurrence computations and three to five times speedup for other 40%.Thus the performance in Table 6 -the improved computational time- is expected.

D. Speedup of Indirect Addressing

In our vector operation a statement of following type (i) requires twice computation time compared to that of (ii).

(i) A(I)=B(L(I)), I=1,・・・・, N

(ii) A(I)=B(I), I=1,・・・・, N

Therefore speedup of indirect addressing(type(i)) will increase the performance of the code very much because most statements of our vectorized KENO IV are of this type (i). However mere improvement of indirect addressing without Monte Carlo pipelines is of small importance because it does not increase the ratio of vectorization in KENO IV. Although we had tested the type (ii) KENO IV version, the gain by it could not overcome the overhead introduced by procedures for conversions from indirect to direct addressings and vice versa.

Table 6 Speedup ratio by Monte Carlo pipelines

Time (sec.) / Category	Current Computation Time	Improved Computation Time	
		Case 1	Case 2
(1) Classification of Geometries.	57.0	19.0 (3 times)	11.4 (5 times)
(2) Classification of Events.	63.0	21.0 (3 times)	12.6 (5 times)
(3) Recurrence Expressions.	48.0	24.0 (2 times)	16.0 (3 times)
(4) Others(scalar).	20.0	20.0	20.0
(5) Vectorized Expressions.	74.0	74.0	74.0
Total Time	262.0(sec.)	158.0(sec.)	134.0(sec.)
Speedup Ratio	1.44	2.41	2.84

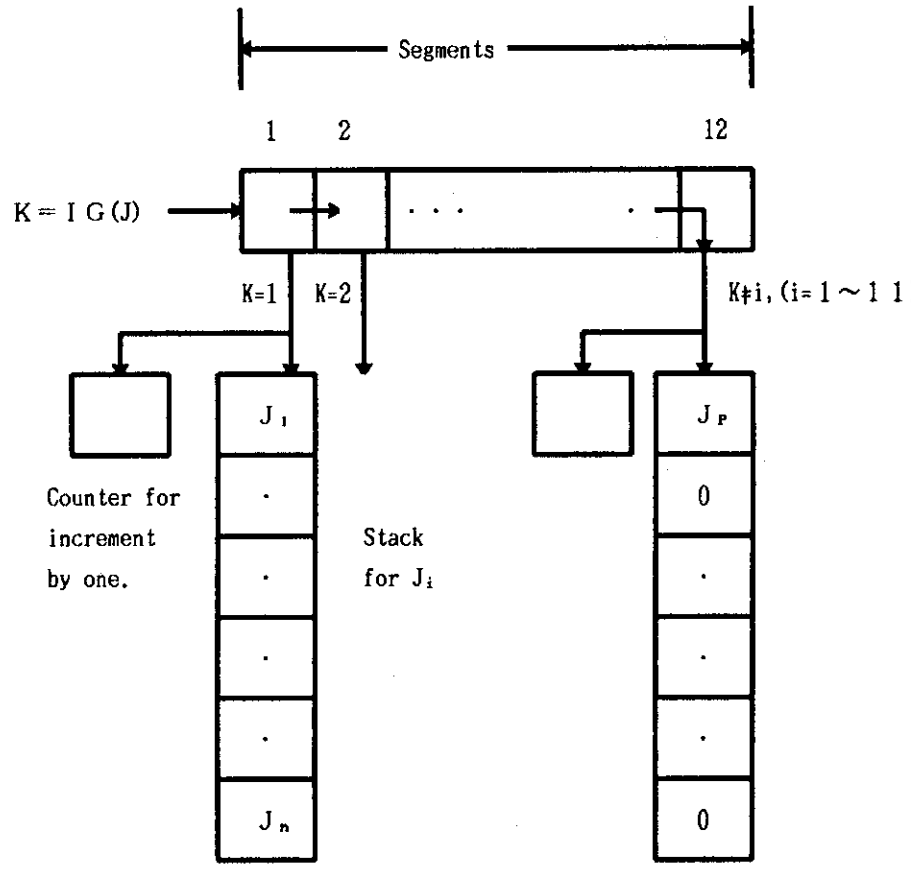

Fig. 25 Pipeline for geometric branches

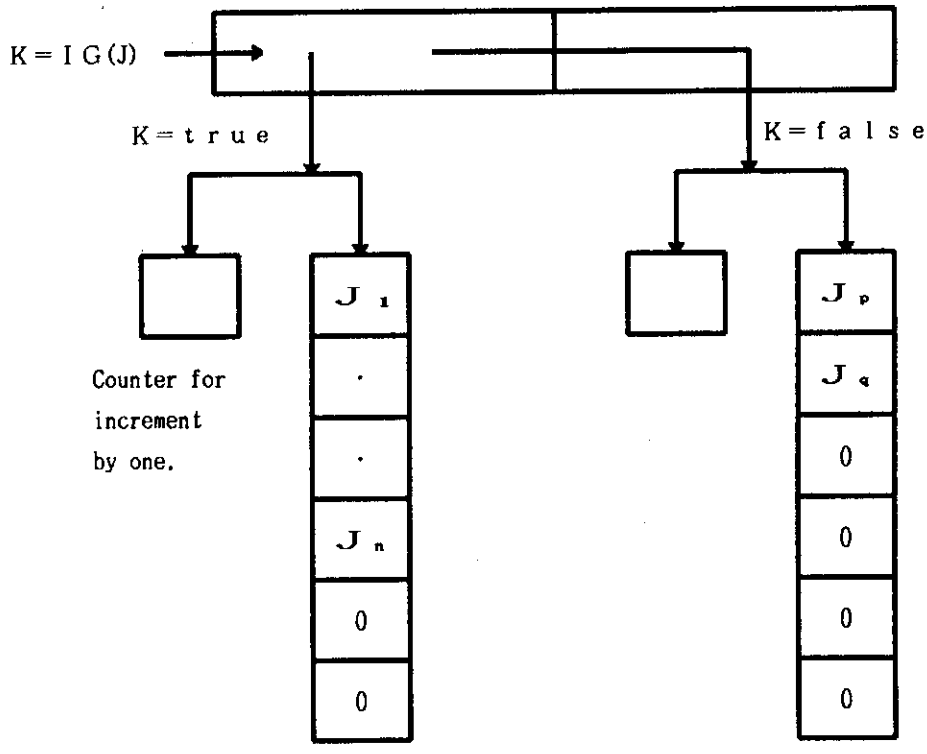

Fig. 26 Pipeline for event branches

8. AN ESTIMATE OF VECTOR-PARALLEL EXECUTION OF KENO IV CODE

Since it is difficult to obtain a good performance of Monte Carlo codes for neutron transport on vector processors, many proposals have been made to do it on multiprocessor systems. The NYU ultracomputer is one of the examples.[12] In this chapter we will propose a pipelined parallel processor system P^3S and give estimates of its performance in processing KENO IV for the Problem 2[13] .

A. A Concept of Pipelined Parallel Processor System

The concept of P^3S is shown in Fig. 27. Every processor has a scalar and vector processing unit with a local memory of 16MB size. In addition to these processors, there is a control processor, but we will not refer to the control processor because it is irrelevant to the following discussions. The copies of program and data of KENO IV code are loaded before the execution in this local memory set in the manner that one copy for one local memory. Data in a local memory are able to transmit onto the global memory and vice versa. It is assumed that these operations are executed by Fortran-like statements such as

```
LOCAL A(137,300)
GLOBAL B(137,300)
.
DO 10 J=1,300
DO 10 I=1,137
10 B(I,J)=A(I,J)
```

In our simulation of Monte Carlo computation, we need only the function of data transmission between the local and global memory.

The computation of k_{eff} by the Monte Carlo method on a vector-parallel computer system is one of the most crucial task. Since the computation begins with a prefixed initial state and the fissions in a sampling, i.e., in a generation, become the source of next generation, the computation proceeds sequentially. This precludes the parallel computation of generations. In the following let us consider the vector-parallel execution of the KENO IV on P^3S computer.

Let us assume the number of processors be 16, and the computation proceeds from left to right cyclically as Fig. 27. We begin with the computation at the processor 1, by giving a prefixed distribution to initial 300 neutrons. KENO

IV compares weights of neutrons and stores a source neutron when the neutron collides with a fissionable material keeping the maximum 300 neutrons in the storage. Hence we can assume the time interval between the two compare and store operations as an arrival time $1/\lambda$ in the traditional queueing theory.[14] On the other hand, if we denote $1/\mu$ as a life time, i.e., remaining time of a neutron in a processor, $1/\mu$ is larger than $1/\lambda$ (i.e., $\lambda \gg \mu$) in a usual criticality calculation because a neutron collides with fissionable materials many times in its life time. Let us feed a source neutron produced in n-th processor to (n+1)-th processor, ..., a source neutron in N-th processor to 1-th processor cyclically. When N = 16, in 2-th processor, for example, neutrons of 2, 18, 34, ... generations will be processed simultaneously. As seen in Figs. 11 - 17 in the case of processing of neutrons in only one generation, the number of neutrons decreases rapidly and we cannot keep the vector length for a long time. The simultaneos processing of neutrons of many generations keeps a long vector length and we can expect to utilize the vector unit of a processor for a long time. As stated in our vectorized KENO IV code for processing ten generations in a vector processor, it is easy to change the code for simultaneous processing.

B. Speedup Ratio

Let us assume the number of neutrons L in the 2-th processor by

$$L = (\lambda - \mu) * t ,$$

where t is the elapsed time of the system. This expression is closely similar to a processor utilization $\rho = \lambda/\mu$ (>1) in a transient state in the M/M/1 model of queueing theory[14] . In Problem 2 for simultaneous processing of ten generations, the average vector length is 77. Hence we can assume the average vector length of a state as 77 in which the number of neutrons increases from zero to 300 in each processor. Changing the number of neutrons, we have tested the performance of VP-100 vector processor using the neutron transport process of KENO IV. The result is shown in Fig. 28. In the case of vector length 77, 0.40 neutrons are processed in a microsecond, and 0.33 neutrons for length 38. From this we can assume the performance ratio f as 0.8. If we set the performance of the processor for 300 neutrons as unity, we can set the performance of the processor for starting up or ending time as 0.8. The time t_1 required for 2-th processor to reserve 300 neutrons is

$$t_1 = t + 1/\mu = 300/(\lambda - \mu) + 1/\mu .$$

Generally the time t_n for the (n+1)-th processor should be $t + n/\mu$, but we can approximate t_n by t_1 because n/μ is negligibly small. Let T_s be the scalar

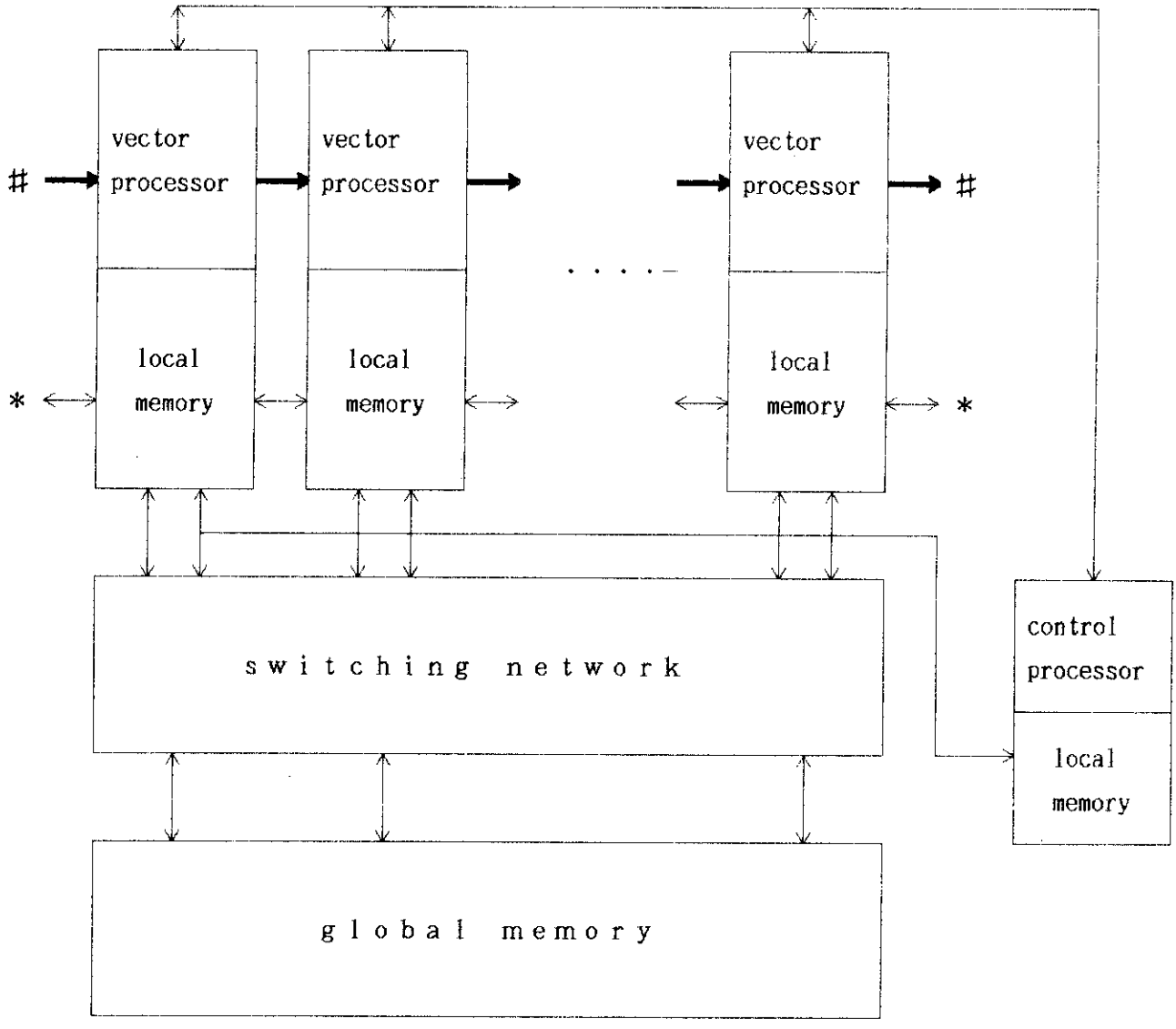

Fig. 27 Concept of pipelined parallel processor system

computation time per generation, V be the average vector/scalar performance ratio, T_s/V be the processing time of a vector processor for one generation, and $t_1 = \alpha * (T_s/V)$. Also let t_2 be $\beta*(T_s/V)$ where t_2 is the time which each processor consumes by processing neutrons from 300 to a truncatable number at the end of the total computation. The time t_1 or t_2 depends on λ, μ and the input problem. Usually the values of α and β are in a range between 0 and 1. Let M be the total number of generations and M/N be the number of generations per processor, then the processing time of each processor T_v is

$$T_v = (M/N - \alpha - \beta)* (T_s/V) + (\alpha + \beta)*(T_s/(f*V)).$$

As for the case of the Problem 2 on VP-100 vector processor, M = 203, α, β = 1, T_s = 1800 ms, λ = one neutron/ms, μ = 0.52 neutrons/ms, then we get

$$T_v = (M/N - 2)*(1.8/V) + 2*(1.8/(0.8*V)) .$$

Using this T_v we can estimate the performance P_1 of a vector processor in the pipelined scheme shown in Fig. 27 as

$$P_1 = (M*T_s/N)/T_v .$$

This P_1 shows a performance ratio of a vector processor in the pipelined scheme vs. a scalar processor as a single computer. In the case of the Problem 2 on the VP-100, V = 1.4. If we assume N = 16, i.e., sixteen units of pipelined processors, we can get T_v = 17.0 second. Since T_s = 1.8 second, the performance P_1 becomes as P_1 = 1.3. Also if we assume vector processors with the Monte Carlo pipelines described in Chapter 7, V = 2.8, and as the results T_v = 8.5 second and P_1 = 2.7. This means that we can enjoy 2.7*N performance compared to a scalar processor.

C. Data Transfer between Processors

The remaining problem which is important in use of multiprocessor system is the data transfer and memory conflicts between processors. A set of variables (X_i) for next generation which is generated by the processor i is pushed down(stacked) in the global memory of the processor system, which means the set is shared among processors. Just before the write-operation, i-th processor locks a global variable G_i (a variable in the global memory) and unlocks it just after the operation. The lock operation is done using the so called "Test and Set" instruction of the processor. In Problem 2, we need 203*300/N operations. The transfer of the information (X_i) from the i-th processor to (i+1)-th's is done just before the initial computation of every neutron in the (i+1)-th processor by locking the variable G_i. This sequence of operations is the same as the "fetch and add" method proposed by the NYU ultracomputer group for a Monte Carlo computer, though the method of the

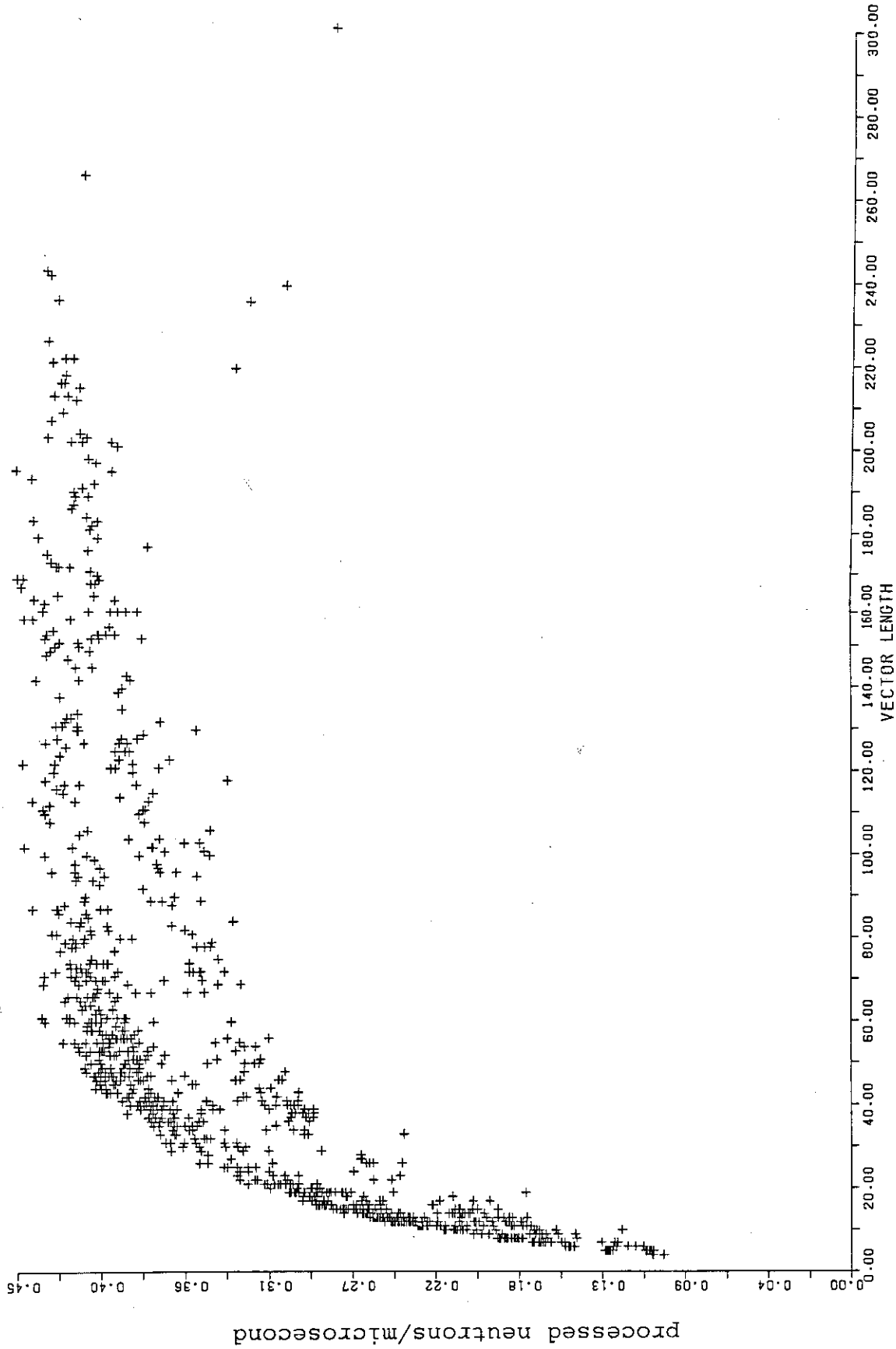

Fig. 28 Vector processor performance vs. vector length

latter is more elaborated. Since any computational flow of Monte Carlo computation returns to the particle transport computation, we can get necessary particles by this technique. In Problem 2, we need 170000/N operations and amount of data is eight words per operation. If we can complete the write or read operation within 5 - 10 microsecond(this can be easily attained in the current computer technology), the total time for the operations is negligibly small and the probability of memory conflicts between processors is very small. Thus it gives no effect on the performance P_1 of the processor system. The termination of the processor system can be initiated by generating no source neutrons for 204-th generation and also checking total histories stored in the global memory. The modification of the source program(i.e., KENO IV code) for these write and read operations will be an addition of 20 statements. Thus we can expect the total performance P_2 of the pipelined vector processor system as

$$P_2 = \begin{cases} 1.4*N \text{ (without Monte Carlo pipelines)} \\ 2.7*N \text{ (with Monte Carlo pipelines)}, \end{cases}$$

where N is the number of vector processors.

For example, in a case of a system with sixteen processors, i.e., N = 16,

$$P_2 = \begin{cases} 22 \text{ (without Monte Carlo pipelines)} \\ 43 \text{ (with Monte Carlo pipelines)} \end{cases}$$

compared to a scalar processor.

D. No Multi-task Operation

It should be noted that in the multiprocessor operations described in this Chapter 8, we do not use the multi-task operations at all. Almost all operations are done in the local processors with local memory and no task synchronization technique is used. Hence there is no loss time for the event waiting. This is the most important factor in the applications. Unless this characteristic, users will be forced to optimize the divisions of the code for multi-tasking according to each input data.

9. CONCLUDING REMARKS

We have seen many pros and cons on vector processing of Monte Carlo codes. However, at least to the authors' knowledge, no vectorized production code by the Monte Carlo method has appeared. Based on our experience with the KENO IV vectorization, we can say that the performance gain of a Monte Carlo code such as KENO IV will range in the factor of 1 to 2. If we wish to get much better performance, a very elaborated compiler and Monte Carlo pipelines and improvement of indirect addressing described in Chapter 7 will be required. When they are realized, we can enjoy a speedup of more than three times easily.

In our vectorization, we did not test a well-known method which keeps a long vector length by splitting a neutron and its weight into two when the number of neutron becomes small. In some cases we can expect a more good performance using this technique.

With our experiences in current vectorizations of MORSE-CG, MCNP and VIM which are now underway, the effectiveness of the Monte Carlo pipelines are decreased in a case of continuous energy Monte Carlo code VIM. We expect that we can make clear the reason elsewhere in the future. As for the computation of a Monte Carlo code on a multiprocessor system, we have shown a performance estimate by a tightly coupled, pipelined multiprocessor system. Many multiprocessor systems presented for Monte Carlo computations so far are based on processing of particles by multi-scalar processor systems. The system proposed in this paper, however, will be the best because it makes use of extensive vector and parallel processing.

ACKNOWLEDGEMENTS

We wish to express our thanks to Drs. Yasuaki Nakahara and Misako Ishiguro at JAERI whose comments have improved the readability of this report. We also express our thanks to Institute of Plasma Physics of Nagoya University, and Fujitsu, Ltd. They cordially had set up and helped the authors to use VP-100 and VP-200 vector processors, respectively in the early stage of this work.

REFERENCES

1. L. M. Petrie and N. F. Cross, "KENO IV-An Improved Monte Carlo Criticality Program," ORNL-4938, (1975).

9. CONCLUDING REMARKS

We have seen many pros and cons on vector processing of Monte Carlo codes. However, at least to the authors' knowledge, no vectorized production code by the Monte Carlo method has appeared. Based on our experience with the KENO IV vectorization, we can say that the performance gain of a Monte Carlo code such as KENO IV will range in the factor of 1 to 2. If we wish to get much better performance, a very elaborated compiler and Monte Carlo pipelines and improvement of indirect addressing described in Chapter 7 will be required. When they are realized, we can enjoy a speedup of more than three times easily.

In our vectorization, we did not test a well-known method which keeps a long vector length by splitting a neutron and its weight into two when the number of neutron becomes small. In some cases we can expect a more good performance using this technique.

With our experiences in current vectorizations of MORSE-CG, MCNP and VIM which are now underway, the effectiveness of the Monte Carlo pipelines are decreased in a case of continuous energy Monte Carlo code VIM. We expect that we can make clear the reason elsewhere in the future. As for the computation of a Monte Carlo code on a multiprocessor system, we have shown a performance estimate by a tightly coupled, pipelined multiprocessor system. Many multiprocessor systems presented for Monte Carlo computations so far are based on processing of particles by multi-scalar processor systems. The system proposed in this paper, however, will be the best because it makes use of extensive vector and parallel processing.

ACKNOWLEDGEMENTS

We wish to express our thanks to Drs. Yasuaki Nakahara and Misako Ishiguro at JAERI whose comments have improved the readability of this report. We also express our thanks to Institute of Plasma Physics of Nagoya University, and Fujitsu, Ltd. They cordially had set up and helped the authors to use VP-100 and VP-200 vector processors, respectively in the early stage of this work.

REFERENCES

1. L. M. Petrie and N. F. Cross, "KENO IV-An Improved Monte Carlo Criticality Program," ORNL-4938, (1975).

9. CONCLUDING REMARKS

We have seen many pros and cons on vector processing of Monte Carlo codes. However, at least to the authors' knowledge, no vectorized production code by the Monte Carlo method has appeared. Based on our experience with the KENO IV vectorization, we can say that the performance gain of a Monte Carlo code such as KENO IV will range in the factor of 1 to 2. If we wish to get much better performance, a very elaborated compiler and Monte Carlo pipelines and improvement of indirect addressing described in Chapter 7 will be required. When they are realized, we can enjoy a speedup of more than three times easily.

In our vectorization, we did not test a well-known method which keeps a long vector length by splitting a neutron and its weight into two when the number of neutron becomes small. In some cases we can expect a more good performance using this technique.

With our experiences in current vectorizations of MORSE-CG, MCNP and VIM which are now underway, the effectiveness of the Monte Carlo pipelines are decreased in a case of continuous energy Monte Carlo code VIM. We expect that we can make clear the reason elsewhere in the future. As for the computation of a Monte Carlo code on a multiprocessor system, we have shown a performance estimate by a tightly coupled, pipelined multiprocessor system. Many multiprocessor systems presented for Monte Carlo computations so far are based on processing of particles by multi-scalar processor systems. The system proposed in this paper, however, will be the best because it makes use of extensive vector and parallel processing.

ACKNOWLEDGEMENTS

We wish to express our thanks to Drs. Yasuaki Nakahara and Misako Ishiguro at JAERI whose comments have improved the readability of this report. We also express our thanks to Institute of Plasma Physics of Nagoya University, and Fujitsu, Ltd. They cordially had set up and helped the authors to use VP-100 and VP-200 vector processors, respectively in the early stage of this work.

REFERENCES

1. L. M. Petrie and N. F. Cross, "KENO IV-An Improved Monte Carlo Criticality Program," ORNL-4938, (1975).

2. K. Uchida and K. Miura, "FACOM Vector Processor VP-100/200," NATO ASI Series, Vol. F7, High-speed Computation, J. S. Kowalik(Ed.), P. 127, (1984).
3. S. J. Raffety and Mihalczo, "Homogeneous Critical Assemblies of 2 and 3% Enriched Uranium in Paraffin," Y-DR-14 (1969).
4. H. Tsuruta et al., "Critical Sizes of Light-Water Moderated UO_2 and $Pu O_2$-UO_2 Lattices," JAERI 1254 (1978).
5. Y. Naito et al., "MGCL-PROCESSOR:A Computer Code System for Processing Multigroup Constants Library MGCL," JAERI-M 9396 (1981).
6. L. L. Carter and E. D. Cashwell, "Particle-Transport Simulation with the Monte Carlo Method," TIS-ERDA (1975).
7. J. A. Ahrens and U. Dieter, "Computer Methods for Sampling from the Exponential and Normal Distributions," Comm. ACM, Vol. 15, No.10 (1972).
8. A. J. Walker, "An Efficient Method for Generating Discrete Random Variables With General Distributions," ACM Trans. Math. Soft., Vol.3, No. 3, P.253 (1977).
9. F. B. Brown, " Vectorized Monte Carlo," Ph.D Dissertation, Michigan Univ. (1981).
10. D. E. Knuth, "The Art of Computer Programming," Vol. II, Addison-Wesley, Reading, Mass. (1981).
11. K. Asai et al., "Vectorization of the KENO IV Code," Nuc. Sci. & Eng., 92, pp.298-307(1986).
12. A. Gotlieb et al., "The NYU Ultracomputer - Designing an MIMD shared Memory Parallel Computer," IEEE Trans. Comp., Vol. C.32. No. 2 (1983).
13. K. Asai et al., "Vector-Parallel Computation of the Monte Carlo Method," 1D-2, The 31th Nat.'l Conf. of Infor. Proc. Soc. of Japan, Tokyo (1985) (in Japanese).
14. T. Suzuki, "Queueing Theory," Shoukabou Publ., Tokyo (1969) (in Japanese).

APPENDIX --- The Input Data and Geometrical Scheme of Problem 2

Following Fig. A, B, and C are the input data for KENO IV code, top view, side view of the geometrical scheme of the experiment for Problem 2.

TCA CRITICAL EXPERIMENT,LATTICE NAME 1.50U,PATTERN 18(19*19) 26GR

NUMBER OF GENERATIONS	203	START TYPE	0
NUMBER PER GENERATION	300	GENERATIONS BETWEEN CHECKPOINTS	0
NUMBER OF GENERATIONS TO BE SKIPPED	3	LIST INPUT X-SECTIONS READ FROM TAPE	NO
NUMBER OF ENERGY GROUPS	137	LIST 1-D MIXTURE X SECTIONS	NO
MAX. NUMBER OF ENERGY TRANSFERS	137	LIST 2-D MIXTURE X-SECTIONS	NO
NUMBER OF INPUT NUCLIDES	3	LIST FISS. AND ABS. BY REGION	NO
NUMBER OF MIXTURES	3	USE X-SECTIONS FROM PREVIOUS CASE	NO
NUMBER OF MIXING TABLE ENTRIES	3	USE GEOMETRY FROM PREVIOUS CASE	NO
NUMBER OF GEOMETRY CARDS	8	USE VELOCITIES FROM PREVIOUS CASE	NO
NUMBER OF BOX TYPES	4	COMPUTE MATRIX K-EFFECTIVE BY UNIT	NO
NUMBER OF UNITS IN X DIRECTION	63	COMPUTE MATRIX K-EFFECTIVE BY BOX TYPE	NO
NUMBER OF UNITS IN Y DIRECTION	63	LIST FISS PROB MATRIX BY UNIT	NO
NUMBER OF UNITS IN Z DIRECTION	3	ADJOINT CALCULATION	NO
NUMBER OF NUCLIDES READ FROM TAPE	3	USE EXPONENTIAL TRANSFORM	NO
ALBEDO TYPE	1	CALCULATE FLUX	YES
SEARCH TYPE	0	CALCULATE FISSION DENSITIES	YES

THIS PROBLEM WILL BE RUN WITH SPECULARLY REFLECTING BOUNDARY CONDITION

THE ALBEDOS ARE +X = 0.0 -X = 0.0 +Y = 0.0 -Y = 0.0 +Z = 0.0 -Z = 0.0

MAXIMUM TIME = 100.0000 MINUTES

STORAGE LOCATIONS REQUIRED FOR THIS JOB = 153196
REMAINING AVAILABLE LOCATIONS= 126804

Fig. A Input data for KENO IV(Problem 2)

TCA CRITICAL EXPERIMENT,LATTICE NAME 1.50U,PATTERN 18(19*19) 26GR

MIXTURE	NUCLIDE	DENSITY
1	-1	1.00000E+00
2	2	1.00000E+00
3	3	1.00000E+00

CROSS SECTIONS READ FROM TAPE

NUCLIDE = 1 * U(2.6)O2 PELLET LATTICE NAME 1.50U
NUCLIDE = 2 * AL CLAD WITH AIR GAP
NUCLIDE = 3 * WATER

TCA CRITICAL EXPERIMENT,LATTICE NAME 1.50U,PATTERN 18(19*19) 26GR

GEOMETRY DESCRIPTION

BOX TYPE 1

REGION

1	CUBOID	3	+X = 9.2450E-01	-X =-9.2450E-01	+Y = 9.2450E-01	-Y =-9.2450E-01	+Z = 4.9725E+01	-Z =-4.9725E+01

BOX TYPE 2

REGION

1	CUBOID	0	+X = 9.2450E-01	-X =-9.2450E-01	+Y = 9.2450E-01	-Y =-9.2450E-01	+Z = 2.2350E+01	-Z =-2.2350E+01

BOX TYPE 3

REGION

1	CYLINDER	1	RADIUS = 6.2510E-01	+Z = 4.9725E+01	-Z =-4.9725E+01							
2	CYLINDER	2	RADIUS = 7.0850E-01	+Z = 4.9725E+01	-Z =-4.9725E+01							
3	CUBOID	3	+X = 9.2450E-01	-X =-9.2450E-01	+Y = 9.2450E-01	-Y =-9.2450E-01	+Z = 4.9725E+01	-Z =-4.9725E+01

BOX TYPE 4

REGION

1	CYLINDER	1	RADIUS = 6.2510E-01	+Z = 2.2350E+01	-Z =-2.2350E+01							
2	CYLINDER	2	RADIUS = 7.0850E-01	+Z = 2.2350E+01	-Z =-2.2350E+01							
3	CUBOID	0	+X = 9.2450E-01	-X =-9.2450E-01	+Y = 9.2450E-01	-Y =-9.2450E-01	+Z = 2.2350E+01	-Z =-2.2350E+01

```
.A CRITICAL EXPERIMENT,LATTICE NAME 1.5OU,PATTERN 18(19*19)  26GR

:IGHTING FUNCTION

)X TYPE   1                                        GROUP   WTLOW    WT AVG    WT HI

      REGION   1 DEFINED BY GEOMETRY CARD   1
                                                    1     0.167     0.500     1.500
                                                  GROUPS 2 TO  137 SAME AS ABOVE

)X TYPE   2                                        GROUP   WTLOW    WT AVG    WT HI

      REGION   1 DEFINED BY GEOMETRY CARD   2
                                                    1     0.167     0.500     1.500
                                                  GROUPS 2 TO  137 SAME AS ABOVE

X TYPE   3                                         GROUP   WTLOW    WT AVG    WT HI

      REGION   1 DEFINED BY GEOMETRY CARD   3
                                                    1     0.167     0.500     1.500
                                                  GROUPS 2 TO  137 SAME AS ABOVE

      REGION   2 DEFINED BY GEOMETRY CARD   4
                                                    1     0.167     0.500     1.500
                                                  GROUPS 2 TO  137 SAME AS ABOVE

      REGION   3 DEFINED BY GEOMETRY CARD   5
                                                    1     0.167     0.500     1.500
                                                  GROUPS 2 TO  137 SAME AS ABOVE

X TYPE   4                                         GROUP   WTLOW    WT AVG    WT HI

      REGION   1 DEFINED BY GEOMETRY CARD   6
                                                    1     0.167     0.500     1.500
                                                  GROUPS 2 TO  137 SAME AS ABOVE

      REGION   2 DEFINED BY GEOMETRY CARD   7
                                                    1     0.167     0.500     1.500
                                                  GROUPS 2 TO  137 SAME AS ABOVE

      REGION   3 DEFINED BY GEOMETRY CARD   8
                                                    1     0.167     0.500     1.500
                                                  GROUPS 2 TO  137 SAME AS ABOVE
```

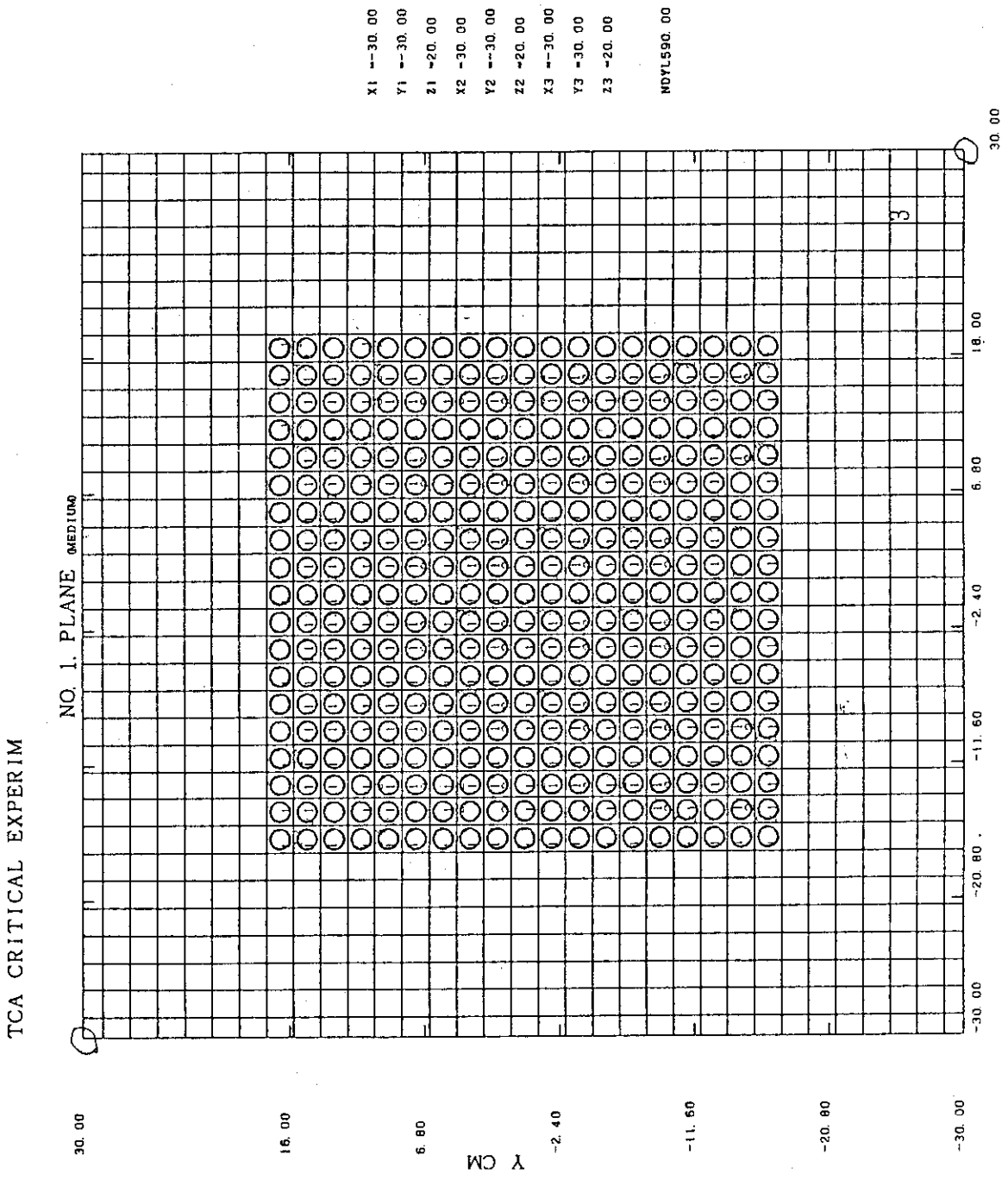

Fig. B Geometrical scheme of the experiment (top view)

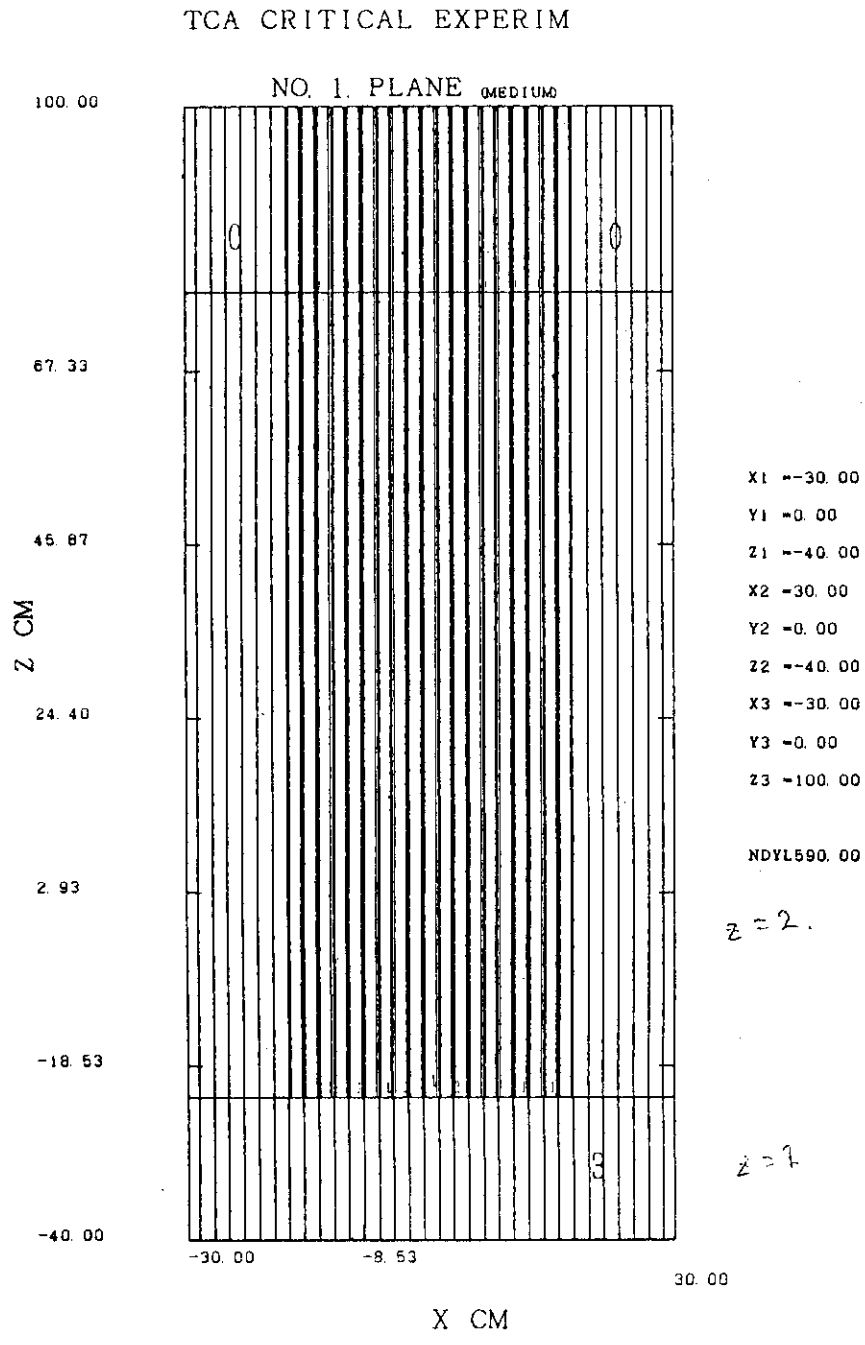

Fig. C Geometrical scheme of the experimment(side view)