To the Graduate Council:

I am submitting herewith a dissertation written by Daniel Francis Hollenbach entitled "Vectorization Methods Development For A New Version of the KENO-V.a Criticality Safety Code." I have examined the final copy of this dissertation for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Doctor of Philosophy, with a major in Nuclear Engineering.

H. L. Dodds, Major Professor

We have read this dissertation
and recommend its acceptance:

Accepted for the Council:

Associate Vice Chancellor and
Dean of the Graduate School

VECTORIZATION METHODS DEVELOPMENT FOR A NEW VERSION OF THE KENO-V.a CRITICALITY SAFETY CODE

A Dissertation

Presented for the

Doctor of Philosophy

Degree

The University of Tennessee, Knoxville

Daniel Francis Hollenbach

August 1992

DEDICATION

This dissertation is dedicated to my wife

Patricia A. Moulton

who, when told I wanted to quit my job, sell the house,
and return to school, replied
"Where are we going and when will you graduate?"

ACKNOWLEDGEMENTS

I would like to thank my major professor, Dr. H. L. Dodds, for his guidance and support over the last three years and the other members of my committee, Dr. P. N. Stevens, Dr. L. M. Miller, and Dr. V. Alexiades for their comments and assistance on my dissertation. I would also like to thank Lester Petrie and Nancy Landers without whose assistance in understanding the intricacies of KENO-V.a this dissertation would not have been possible and Martin Marietta Energy Systems, Inc. for funding this project. Lastly, I would like to thank my family: Patricia Moulton and Natasha Hollenbach for never doubting that I would eventually graduate and never complaining about the sacrifices we had to make to permit me to return to school.

ABSTRACT

The objective of this research project is to develop a vectorized version of the KENO-V.a Criticality Safety Code, benchmark it against the original version of the code, and determine its speed-up factor for various classes of problems. The current generation of supercomputers are equipped with vector processors which allow the same operation to be simultaneously performed on a string of data. Unfortunately, the Monte Carlo Algorithm used in KENO-V.a, which tracks particles individually, cannot utilize these vector processors. A new Monte Carlo Algorithm needed to be developed which would efficiently utilize the vector processors currently used in computers.

The algorithm developed for the vectorized version of KENO-V.a is an event-based, stack-driven, all-zone, tagged particle Monte Carlo algorithm. This algorithm divides the particles into one of four stacks: free-flight, inward-crossing, outward-crossing, or collision. A fifth stack, Kill, contains all particles that have either leaked from the system or been terminated by Russian roulette. The stack containing the largest number of particles is the next stack processed. The generation is completed when the four main stacks are empty. All the particles in the longest stack are processed simultaneously. Only the particle number is transferred between stacks, the particle data remain in

permanent vector locations and are updated as the particles traverse through the system.

This approach minimizes data transfer between stacks and optimizes the vector length thus maximizing the speed-up. For the 25 benchmark problems, speed-up factors ranging from 1.8 to 5.7 relative to the optimized scalar version of KENO-V.a were obtained. In addition, problem geometry, material composition, and the number of histories per generation all have significant effects on the speed-up factor.

TABLE OF CONTENTS

List of Tables

List of Figures

CHAPTER 1

INTRODUCTION

1.1 Purpose

Monte Carlo particle transport computer codes have been a mainstay of nuclear engineering and physics for over 40 years.[1] In many cases Monte Carlo calculations provide the only feasible means of solving problems with complex geometry and/or neutron physics. The Monte Carlo method allows the geometry of a problem to be precisely modelled reducing the systematic errors to those associated with probability or cross-section uncertainty. The errors associated with the probabilities are a result of experimental measurement uncertainty and cross-section processing and are present to some degree in all computer codes that use them. The precision of the Monte Carlo results is associated with the number of histories processed and can be improved by increasing the number of histories. However, the standard deviation of the results is inversely proportional to the square root of the number of histories processed so a large increase in the number of histories is required for a modest decrease in the standard deviation.[1,2,3]

Given the inherent advantages and accuracy of Monte Carlo methods their use is very attractive to the engineering and science community. Unfortunately, Monte carlo methods have a serious drawback in that their use can require large

amounts of computer time. A large problem solved by tallying data from histories may require several hours or possibly days of CPU time. Conventional (scalar) Monte Carlo codes process histories sequentially, one at a time from birth to death. A history is tracked through the problem's geometry until it is either eliminated using Russian roulette or by leakage from the system. Since the CPU time needed for particle tracking generally accounts for the vast majority of the CPU time required by the problem, the number of histories is directly proportional to the problem's CPU time for conventional Monte Carlo.

The need for large amounts of CPU time has been partially ameliorated by the development of faster computers such as the CYBER-205, CRAY-1, CRAY-XMP, CRAY-YMP, CRAY-2 and the IBM-3090. Significant speed-ups of conventional Monte Carlo codes on these machines are primarily due to the reduced cycle time and improved processor architecture. In addition, these computers have one important difference from all previous computers. This new generation of computers utilizes vector processors which promise additional speed-up factors of 10 to 30 over their scalar speeds if fully utilized.

Vector processors can only be used for non-recursive operations on ordered sets of data. Unfortunately, conventional Monte Carlo codes are not well suited for direct vectorization due to the use of a history based algorithm.

Conventional Monte Carlo codes contain few loops and large numbers of conditional statements which means that a minimum of structure is present that can be operated on as vectors. The very nature of the history based Monte Carlo algorithm precludes the use of vectorization software or simple recoding to produce a Monte Carlo code that would efficiently utilize vector processors. What is needed is a new Monte Carlo algorithm which processes large numbers of particles simultaneously taking advantage of the similarities associated with given events such as collision or free-flight, minimizing the number of conditional statements, and moving the remaining conditional statements out of the loops.

Since the advent of vector processors, several attempts have been made both to vectorize existing Monte Carlo codes and to produce new vectorized Monte Carlo codes. A vectorized Monte Carlo particle transport code, produced at KAPL by Brown et. al, achieves speed-up factors as high as 10 on the CYBER-205 when compared to itself run in scalar mode.[4] A second set of vectorized Monte Carlo particle transport codes, produced at JAERI by Nakagawa et. al, achieve speed-up factors as high as 10 on the FACOM VP-100 when compared to MORSE or KENO-IV.[5] Another group at JAERI had limited success producing a vectorized version of KENO-IV which achieved a speed-up factor of 3 when optimized for a specific problem.[6] All attempts to measure speed-up factors, with the exception of KENO-IV, involved comparing the vectorized Monte

Carlo code to either the same code run in scalar mode or to what is termed a comparable code. Comparing a vectorized Monte Carlo code to itself in scalar mode or a comparable code does not provide a true indication of the actual speed-up associated with vectorizing a Monte Carlo code. The speed of a Monte Carlo code is strongly dependant on the specific algorithm, geometry, and options used in the code. Also, a vectorized Monte Carlo code runs significantly slower in scalar mode than the same code optimized for a scalar processor due to the additional steps needed to shuffle data to and from banks.

This dissertation considers the vectorization of the KENO-V.a criticality safety code. A vectorized version of KENO-V.a has been developed that contains the same geometry package and all the options currently in the scalar version of KENO-V.a with the exception of super-grouping. Since the current version of KENO-V.a is optimized for a scalar processor, comparing the CPU time of the vectorized version of KENO-V.a to the CPU time of the current scalar version of KENO-V.a for a specific problem on the CRAY-XMP at Oak Ridge National Laboratory will produce a true measure of the actual speed-up which can be achieved through vectorization. The vectorized version of KENO-V.a is first benchmarked against the current version of KENO-V.a for 25 problems which include all the geometry types and use most of the options in various combinations. Finally, speed-up factors for several

different classes of problems are determined and analyzed.

All changes made to KENO-V.a are transparent to the user. Although the cross-section storage and retrieval method, the particle storage method, and the particle tracking algorithm have been altered, the input parameters and their sequence are identical to the original (scalar) version of KENO-V.a. All options previously available in KENO-V.a are still available in the vectorized version of KENO-V.a with the exception of supergrouping. Because the input listing of a problem for the original version of KENO-V.a is identical to the input listing used in the vectorized version of KENO-V.a the current version of the KENO-V.a manual is applicable to the vectorized version of KENO-V.a.[17]

1.2 Previous Work

The first known work related to vectorizing Monte Carlo was performed in 1973 by E. Troubetzkoy, H. Steinberg, and M. Kalos using the SAM-CE computer code oriented to the ILLIAC-IV computer.[7] The ILLIAC-IV had 64 processing elements (PEs), each with its own memory, all controlled by a single control unit. The PEs all carry out the same operation at the same time. Their approach was to follow many particles in each PE, performing an operation only if enough PE's had at least one particle needing to undergo the same operation. Since the computer was not yet built, a simulation of the program showed this code should run about 20 times faster on

the ILLIAC-IV than on a conventional computer.

The first work related to vectorizing Monte Carlo codes on the current generation of supercomputers was performed in 1981 by Martin, Brown, and Calahan at the University of Michigan.[8] Their objective was to determine the speed-up associated with vectorizing a simple Monte Carlo code involving the transport of gamma-rays through a cylinder of carbon using a continuous-energy spectrum. Since a supercomputer was not available for their use, a timing model which simulated a CYBER-205 was used to project speed-up factors for the vectorized version of their Monte Carlo code.[9]

The initial approach used to vectorize Monte Carlo codes involved transferring particles with their characteristics into stacks set up for each event type in each region. The vectorized code worked as follows: first it looped over each event in a region looping over the number of surfaces for the region to determine boundary crossings, then it looped over the number of regions, and finally over the number of generations. In addition, a method was developed which permitted the cross-section data stored in tabular form to be retrieved as a vector thus permitting the vectorization of collision events.[8,10] Even for this crude vectorization technique, a speed-up factor of 40 on the CYBER-205 was predicted.

Vectorized Monte Carlo algorithms can be divided into

classes based on the method used to sort particles into stacks and process the stacks.[11] A vectorized Monte Carlo algorithm could be either event driven, i.e. each event type is processed in a specific order regardless of the number of particles in the event stack, or stack driven, i.e. the stack with the largest number of particles is processed next. For a given event the zones could either be processed individually, which is called a one-zone algorithm, or simultaneously, which is called an all-zone algorithm. All the particles and their data could be stored in one bank with a tag indicating the next task to be performed on each particle, or the particles could be divided into separate explicit stacks for each event. Finally, all the particle data could be stored in one large implicit array. Instead of transferring particle data between stacks, pointers used to reference the particle data could be transferred between stacks. Only the data required for a particular operation would be referenced, and updated data would be dispersed back into the implicit array.

Vectorized Monte Carlo has been investigated continuously since 1981. Brown developed simple multi-group codes and continuous-energy codes that have shown significant speed-ups when vectorized.[11] A two-dimensional, all-zone, stack-driven continuous energy neutron energy code was also developed.[11] This code had a speed-up factor ranging from 20 to 85 on the CYBER-205 depending on the problem. Finally, a

three-dimensional, general-geometry, continuous-energy Monte Carlo Code called RACER3D[12,13] was written for production use. RACER3D[13,14] is equivalent in capability to other production codes such as MCNP[15], MORSE[16] and KENO-IV.[17] RACER3D uses an all-zone, stack-driven algorithm with one large array for particle data. Pointers referring to the appropriate particle data are shuffled between stacks for each event. The stack containing the largest number of pointers is processed. Speed-up factors of 10 have been achieved for three-dimensional, full-core analyses using a CYBER-205.

In the mid-1980's a group at the Japan Atomic Energy Research Institute, JAERI, did extensive work on vectorized Monte Carlo. K. Asia, K. Higuchi, J. Katakura, and Y. Kurita[6,18] attempted to produce a vectorized version of KENO-IV that would run on a FACOM VP-100 vector processor. The maximum theoretically attainable speed-up factor on the FACOM VP-100 by fully vectorizing a scalar code is approximately 28. The structure of KENO-IV was not significantly altered. Instead, a number of banks were provided for each event and all the histories were run simultaneously. The length of all banks were compared and the event with the longest bank length was processed first. A vectorized version of the random number generator was also developed in assembly language. The algorithm used to select new particle energies and directions was replaced by the scheme used by Brown.[8] Finally, rare events were isolated so they were not used in

most computations. With all these modifications, the vectorized version of KENO-IV was actually slower than the original code.

The poor performance of the vectorized version of KENO-IV was attributed to the small vector lengths that occur toward the end of a generation. Two methods for extending the vector length were considered; using more neutrons per generation with fewer generations and simultaneously processing many generations. This vectorized version of KENO-IV contained a high percentage of scalar operations so increasing the number of particles per generation from 300 to 900 produced no speed-up. However, by processing 10 generations of 300 particles simultaneously, the vectorized version of KENO-IV attained the same speed as the original code. Unfortunately, this produced recursion problems for quantities such as total fissions and fission densities which depend on the previous generation. By altering the way two-dimensional variable subscripts were processed in KENO-IV, the vectorized version of the code obtained a speed-up factor of 1.2. Finally, to eliminate the relatively long time period needed to process the last few particles, the last 60 particles in every 10 generations were discarded and keff, fluxes, fissions, etc. were adjusted to account for the discarded particles. Eliminating the last few particles resulted in a final speed-up factor of 1.4.

The last phase of the JAERI project involved deleting

all unnecessary geometry from KENO-IV, simplifying the event section of the code, deleting flux, fission density, neutron life-time and other extraneous calculations, and deleting any conditional branches that are not necessary for the specific problem. When the vectorized version of KENO-IV had been stripped of everything not needed to run the specific problem, a speed-up factor of slightly greater than 3 was obtained.

A second group working at JAERI, consisting of M. Nakagawa, T. Mori, and M. Sasaki, developed a general-purpose, generalized geometry, multi-group vectorized Monte Carlo Code called GMVP.[5,19,20] Two versions of GMVP were written: the event-selection version which processes all zones undergoing a given event and the zone-selection version which processes only the zone for an event containing the largest number of particles. Lattice geometry was added to the GMVP code to increase the number of particles in a zone. Both versions of this code were optimized for a FACOM VP-100 vector supercomputer. The FACOM VP-100 has one processor, three pipelines, a 32 Kbyte vector register, and a 7.5 ns cycle time. Its maximum scalar performance is 9 MFLOPS and its maximum vector performance is 250 MFLOPS for a maximum theoretical vector speed-up factor of approximately 28.

The CPU time of both versions of GMVP were compared to the CPU time of different versions of MORSE, KENO-IV, and to each other for several different problems. For a problem

consisting of a 17 x 17 pin fuel assembly with 5000 histories per batch, GMVP produced a speed-up factor for the zone selection version of 9.9 and for the event selection version of 6.8 compared to MORSE-CGA. For a problem consisting of plutonium storage tanks in a water pool, GMVP produced speed-up factors for the zone selection version of 2.79 for 300 hpb (histories per batch) to 8.63 for 6000 hpb and for the event selection version speed-up factors ranging from 2.90 for 300 hpb to 8.12 for 6000 hpb compared to KENO-IV.

When compared to each other, the event selection version of GMVP produced slightly better speed-up factors than the zone selection version of GMVP for problems with either a small number of histories per batch or a large number of zones. However, for a large number of histories per batch or few zones the zone selection version of GMVP can produce a speed-up factor up to 30% higher than the speed-up factor produced by the event selection version of GMVP. This is primarily due to the short vector lengths for each zone not offsetting the added overhead involved in sorting by zone.

The same group at JAERI also wrote a continuous energy version of GMVP, called MVP,[21] that uses the zone selection algorithm developed in GMVP[25]. For a 17 x 17 pin fuel assembly, MVP produced a speed-up factor of 8 for 5000 histories per batch compared to VIM[22] which increased to a speed-up factor of 10 for 20000 histories per batch.

1.3 Vector Computers

Flynn devised a scheme to classify computer architectures by the type of data and instruction stream used.[23] Flynn proposed four separate categories: Single Instruction Single Data (SISD), Single Instruction Multiple Data (SIMD), Multiple Instruction Single Data (MISD), and Multiple Instruction Multiple Data (MIMD). Current scalar computers which execute one instruction on one piece of data are representative of the SISD machines. A SIMD machine allows multiple data to be operated on by the same instruction simultaneously. The CRAY-XMP falls into this category. Machines which fall under the category of MISD do not exist today and are not planned in the future. MIMD machines perform multiple operations on multiple sets of data simultaneously. Parallel processing machines which contain multiple CPUs under the direction of a master CPU fall into this category.

Before the increase in performance due to vectorization can be evaluated, a measure of performance must first be established. The oldest measure of performance for a computer is Millions of Instructions Per Second (MIPS). While this is an acceptable measure for scalar computers, vector and parallel computers can issue one instruction which produces many results. Another measure of performance is Millions of Floating Point Operations per Second (MFLOPS) which is a measure of the total number of operations

performed and therefore a better indication of the execution speed.

The CRAY-XMP performs operations both in scalar mode as an SISD machine and in vector mode as an SIMD machine depending on the structure of the data stream, the presence of possible recursive relationships, and compiler directives issued to the program. The CRAY-XMP has a scalar speed of 20 MFLOPS and a maximum single processor vector speed of 210 MFLOPS resulting an a maximum theoretical speed-up of 10.5 if all operations are optimally performed in vector mode. Amdahl's Law[24] provides a means of measuring potential code speed-up based on the fraction of operations performed in scalar mode and the fraction performed in vector mode. To calculate the potential speed-up of a partially vectorized program verses a totally scalar program let:

T_s = Scalar program CPU time

T_v = Vectorized program CPU time

F_s = Fraction of operations performed in scalar mode

F_v = Fraction of operations performed in vector mode.

The total CPU time of the partially vectorized program is

$$T = T_s*F_s + T_v*Fv \tag{1.1}$$

which, upon replacing F_s, becomes

$$T = T_s*(1 - F_v) + T_v*F_v. \tag{1.2}$$

By dividing through by T_s the speed-up of the partially vectorized program over the program run totally in scalar mode becomes:

$$SU = 1 / (1 - F_v + F_v*T_v/T_s). \quad (1.3)$$

Since the times required to run the program in the scalar and vector modes are inversely proportional to their speeds in MFLOPS, for the CRAY-XMP,

$$T_v/T_s = 20/210 = 0.095. \quad (1.4)$$

The theoretical speed-up for the CRAY-XMP is therefore

$$SU = 1 / (1 - F_v + 0.095*F_v) \quad (1.5)$$

which, upon rearranging, becomes:

$$SU = 1 / (1 - 0.905*F_v). \quad (1.6)$$

This procedure provides a means of estimating the speed-up of a code based on the fraction of operations vectorized and permits the determination of the amount of code that must be vectorized to achieve a set goal. Figure 1.1 is a plot of the theoretical speed-up based on the amount of code vectorized for the CRAY-XMP using equation 1.6. The graph clearly shows that the majority of code must be vectorized to achieve a significate speed-up factor. Although vectorizing 100% of the code will achieve a theoretical speed-up of 10.5, 55% of the code must be vectorized to achieve a theoretical speed-up of only 2.

The Cray-XMP achieves its speed through the use of a pipelining processor, overlapping, and chaining operations.[25] Pipelining, or vectorizing, is exemplified by an assembly line. Pipelining is broken down into two phases: the startup phase where the pipeline is initially filled and the streaming phase where results are produced every clock cycle.

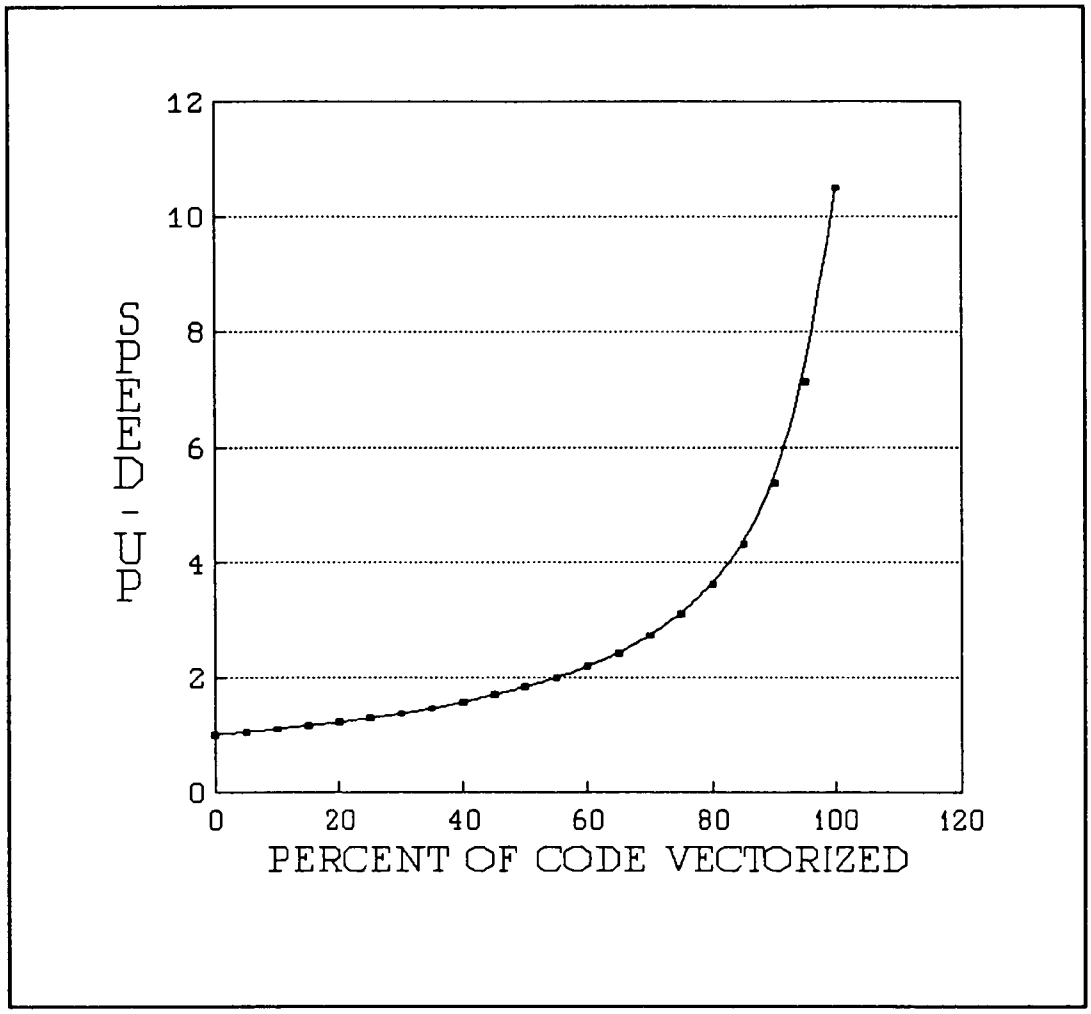

Figure 1.1 Speed-Up Factors for the CRAY-XMP verses Percentage of Code Vectorized

The pipeline works by unrolling or segmenting the operation into a set of subtasks. After a subtask is performed on a set of data, the result moves to the next subtask and the next data set moves to the subtask just completed. In this manner a string of data marches lockstep down an assembly line of subtasks with consecutive results being produced every clock cycle after the first result is obtained. Pipeline architectures are very fast if the vectors are long enough so that startup times are not significant.

Overlapping is the ability to execute an instruction which simultaneously performs different operations on different operands. Overlapping is possible through the use of multiple registers and the unusual architecture of the CRAY-XMP. Given the following line of code in a DO loop,

$$A(I) = (B(I) + C(I))/(D(I)*E(I)), \qquad (1.7)$$

overlapping would allow the two operations in parentheses to be simultaneously computed.

Chaining is the ability to link together vector instructions such that the output of one instruction is the input to another instruction. This means the results of an operation simultaneously feed into both the register containing the operation result and the functional unit of the second operation. The term CHIME (Chained Vector Time) has been coined to represent the number of clock cycles equal to the vector length plus a few more for the chaining of one instruction to the next, e.g. for a given vector length of 50, a chime for two chained instructions would be approximately 58, and a chime for three chained instructions would be approximately 67.

The CRAY-XMP at Oak Ridge National Laboratory contains one CPU with two READ functional units, one WRITE functional unit, and a separate memory path for I/O with an effective memory bandwidth of 50 gigabits per second. Gather/scatter and compressed index capabilities have been included in this machine which enable the efficient handling of indirect

accessing and sparse matrices. Chaining and overlapping are also used to enhance the speed of the machine. The CRAY is a model XMP/14 which means it contains one CPU and 4 megawords of static MOS memory with each word containing 64 bits. The CPU contains 8 registers which are utilized for both scalar and vector operations. A base cycle time of 9.5 nanoseconds (105M clock cycles/second) enables the CPU to operate at a scalar speed of 20 MFLOPS and a maximum vector speed of 210 MFLOPS. The vector registers are limited to a length of 64 words which would require breaking up larger vector lengths into several smaller vectors 64 words in length or less.

The CRAY-XMP runs under a UNICOS operating system using the CFT77 fortran compiler.[26,27] The Fortran code compiles using the CFT77 compiler which conforms to the American National Standard Institute (ANSI) standard X3.9-1978 usually referred to as Fortran 77. Extensions to this standard, which enable a code to take advantage of the features of the CRAY computer system, also included in the CFT77 compiler.

The compiler options include the ability to specify the level of optimization used when compiling a code. Three levels of optimization are possible: full, novector, and off. Full optimization will optimize and vectorize the entire code when possible. Novector will compile the code in the optimized scalar mode. Off will compile the code as written with no vectorization or optimization. The timing studies in

this research were performed with the original and vectorized versions of KENO-V.a fully optimized.

1.4 Overview and Originality of This Work

The main objective of this work is to develop a version of the KENO-V.a Criticality Safety Code optimized for the vector processor on the CRAY-XMP at Oak Ridge National Laboratory. Significant work has already been done by others on vectorized Monte Carlo with two production level vectorized Monte Carlo codes existing today. Both of these codes contain geometry packages different from that contained in KENO-V.a and are optimized for computers radically different than the CRAY-XMP. Although both codes use event-based algorithms, the particle vectors are ordered and processed differently than in KENO-V.a. The vectorized version of KENO-V.a is also the first production level Monte Carlo code compared against an optimized scalar version of itself. Previous speed-up factors were calculated by comparing the vectorized version of the code to either the vectorized version of the code run in scalar mode or what is termed an equivalent scalar code. Neither of these methods produce a true measure of the potential speed-up obtainable by producing a vectorized verses a scalar version of a code. Finally, this work will result in a new, faster version of an existing Monte Carlo Criticality Safety Code that can be readily used by the nuclear community.

To carry out this project, a working knowledge of the current scalar version of KENO-V.a was required as well as an understanding of Monte Carlo methods, neutron transport theory, variance reduction techniques, FORTRAN-77, vector operations, and the CRAY-XMP operating system. This dissertation as written assumes a basic knowledge of Monte Carlo and transport theory and therefore does not include the derivation of the Boltzmann Transport Equation into the form utilized by KENO-V.a. The starting point for this dissertation is the existing scalar version of KENO-V.a

Chapter 2 contains a brief overview of the Monte Carlo algorithm used in the scalar version of KENO-V.a The geometry package used by this code is described including the various ways the geometry can be combined to form a system to be analyzed. Also included is a description of the method used to store and retrieve cross-section data, a listing of several available options, and an overview of the scalar particle tracking algorithm.

Chapter 3 describes the methods used to optimize KENO-V.a for the CRAY-XMP vector processor. First, the modifications involving the storage and handling of cross-section data are described in some detail. The particle tracking algorithm is then described in detail. The particle tracking algorithm is primarily contained in the subroutines CROS and TRACK. Subroutine CROS, which determines if particles cross a boundary, is divided into sections

according to geometry and crossing type. Each crossing and geometry type are described. Subroutine TRACK, which traces particles through the system, is divided into 4 main sections; free-flight, inward-crossing, outward-crossing, and collision. The logic contained in each section is examined.

Chapter 4 contains comparisons of results obtained using the new vectorized version of KENO-V.a and the original scalar version of KENO-V.a. The results are divided into three sections. The first section shows the scalar speed-up associated with the new cross-section storage and handling methodology. The next section contains an initial benchmarking of the vectorized version of KENO-V.a against the scalar version of KENO-V.a. The benchmarking consists of the 25 problems contained in the KENO-V.a manual.[17] These 25 problems contain all the geometry types available in KENO-V.a, most of the options available in KENO-V.a, and a good variety of geometry and option combinations. Finally, speed-up factors are determined for various types of problems and particle batch sizes. For several problems, the number of particles per batch are varied and the changes in speed-up factors are calculated.

Finally, Chapter 5 presents the conclusions of this work as well as some suggestions on how the code could be further improved. Conditions needed to optimize the speed-up factor, comparison to speed-up factors of past codes, and further benchmarking requirements are also discussed.

CHAPTER 2

SCALAR KENO-V.a

2.1 Geometry Package

KENO-V.a is a 3-dimensional, multigroup Monte Carlo criticality safety program routinely used to calculate the k_{eff} of a fissile system. Systems are constructed from geometric shapes consisting of spheres, hemispheres, cylinders, hemicylinders, and rectangular parallelpipeds also known as cuboids. Table 2.1 contains a list of all geometric shapes available in KENO-V.a. All geometric shapes must be orientated along either the X-axis, Y-axis, or Z-axis. Geometric shapes are used to construct regions. Each region consists of a specific geometric shape of specified dimensions and contains a specified material composition.

Units are the basic building blocks of a system. Although, each unit has its own coordinate system, all coordinate systems must have the same orientation. This is done by correlating all coordinate systems to the absolute coordinate system of a global unit or array. Units are constructed by fully enclosing one region by another. A geometric region may be placed anywhere within a unit as long as it does not intersect another region and completely encloses all interior regions. Regions may however touch at a tangent and share faces. Figure 2.1 is an example of acceptable units. Regions may be stacked to any depth as

Table 2.1

Geometric Shapes Available in KENO-V.a

KENO-V.a Geometry Word	DESCRIPTION
CUBE	cube, +X = +Y = +Z, -X = -Y = -Z
CUBOID	Rectangular Parallelpiped
SPHERE	Origin Centered Sphere, may offset
CYLINDER ZCYLINDER	Cylinder oriented along Z-axis
XCYLINDER	Cylinder oriented along X-axis
YCYLINDER	Cylinder oriented along Y-axis
HEMISPHERE HEMISPHE+X	Hemisphere, +X spherical surface
HEMISPHE-X	Hemisphere, -X spherical surface
HEMISPHE+Y	Hemisphere, +Y spherical surface
HEMISPHE-Y	Hemisphere, -Y spherical surface
HEMISPHE+Z	Hemisphere, +Z spherical surface
HEMISPHE-Z	Hemisphere, -Z spherical surface
XHEMICYL+Y	X-Hemicylinder, +Y cylindrical surface
XHEMICYL-Y	X-Hemicylinder, -Y cylindrical surface
XHEMICYL+Z	X-Hemicylinder, +Z cylindrical surface
XHEMICYL-Z	X-Hemicylinder, -Z cylindrical surface
YHEMICYL+X	Y-Hemicylinder, +X cylindrical surface
YHEMICYL-X	Y-Hemicylinder, -X cylindrical surface
YHEMICYL+Z	Y-Hemicylinder, +Z cylindrical surface
YHEMICYL-Z	Y-Hemicylinder, -Z cylindrical surface
ZHEMICYL+X	Z-Hemicylinder, +X cylindrical surface
ZHEMICYL-X	Z-Hemicylinder, -X cylindrical surface
ZHEMICYL+Y	Z-Hemicylinder, +Y cylindrical surface
ZHEMICYL-Y	Z-Hemicylinder, -Y cylindrical surface

long as each region entirely encloses only one interior region. Special options are available which circumvent this restriction. These options, which include ARRAYS and HOLES, are described later.

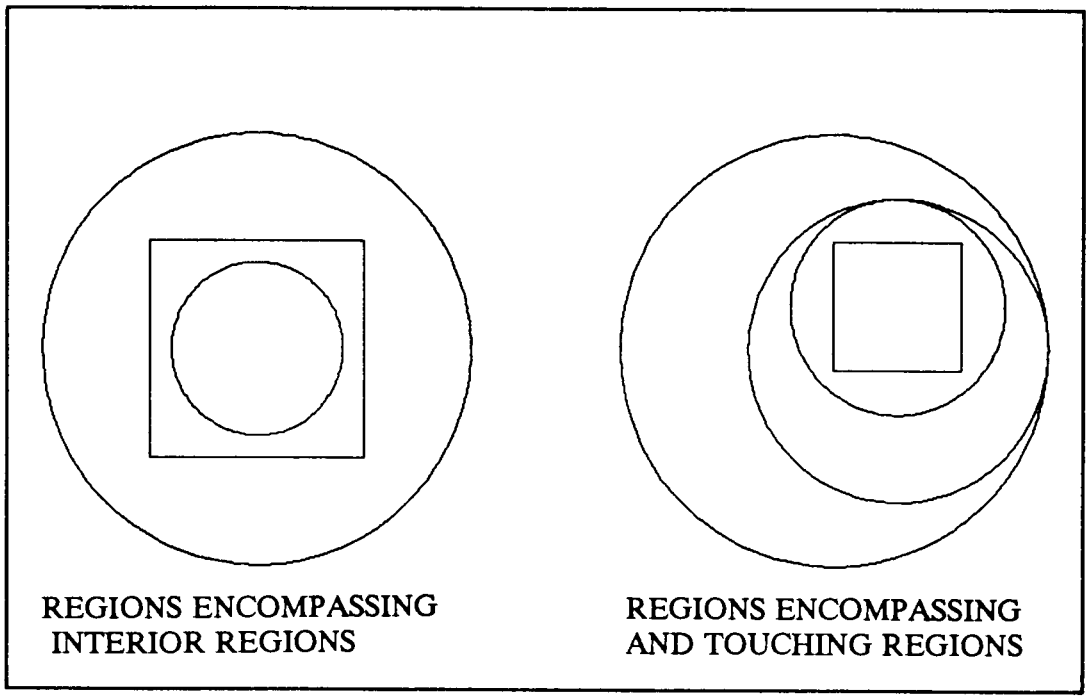

Figure 2.1 Examples of Correct KENO-V.a Units

The use of the hole option is the simplest method of avoiding the restriction of being able to enclose only one region within another region. Given a set of regions in unit 1 and a second set of regions in unit 2, unit 1 may be placed anywhere within a region in unit 2 as a hole as long as the hole does not intersect either the enclosed region or the boundary of the region containing the hole. Figure 2.2 is an example of how holes are used in KENO-V.a. Holes may contain not only units but arrays and other holes. This is done by

first making a unit consisting of an array or hole. This unit may then be inserted into a region as a hole. The practice of placing holes within holes is called nesting. Holes may be nested down to any depth limited only by computer space restrictions.

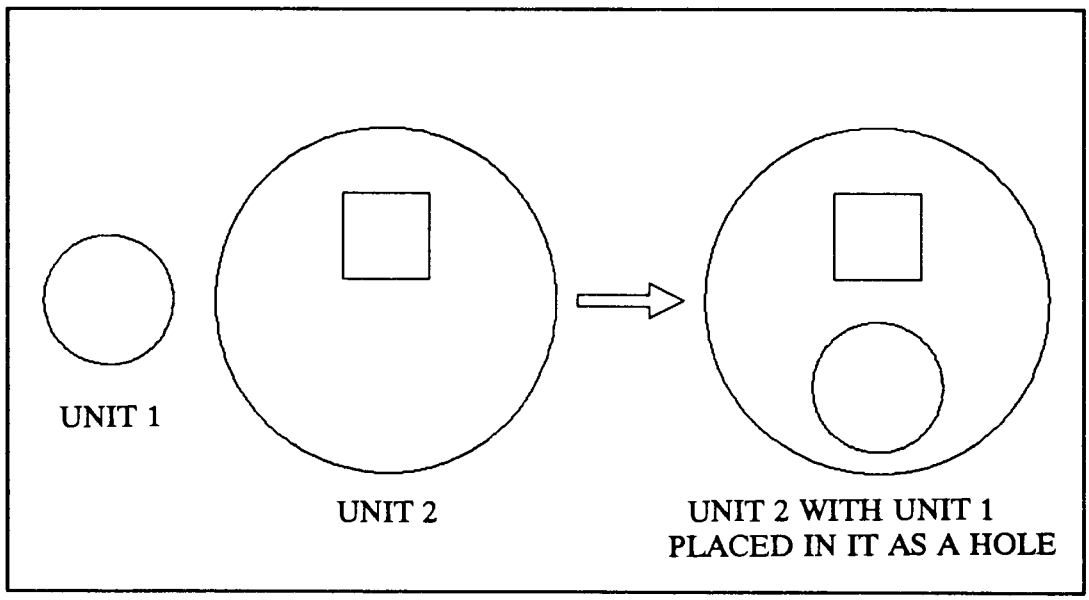

Figure 2.2 Example of Holes in KENO-V.a

Lattices or arrays are a method of stacking units in an ordered arrangement. The outer regions of the stacked units must be rectangular parallelpipeds. The dimensions of the adjacent faces of the adjacent units stacked in the array must exactly match. Only one array can be placed directly in a unit. A nesting option allows the creation of arrays composed of other arrays. This is done by enclosing an array in a unit and using that unit in an array. In this manner arrays may be nested down to any depth limited only by

computer memory restrictions. Figure 2.3 is an example of how arrays are constructed in KENO-V.a. It is also possible to place multiple arrays in a unit using holes. This is done by placing the unit containing the array in the hole rather than the array itself.

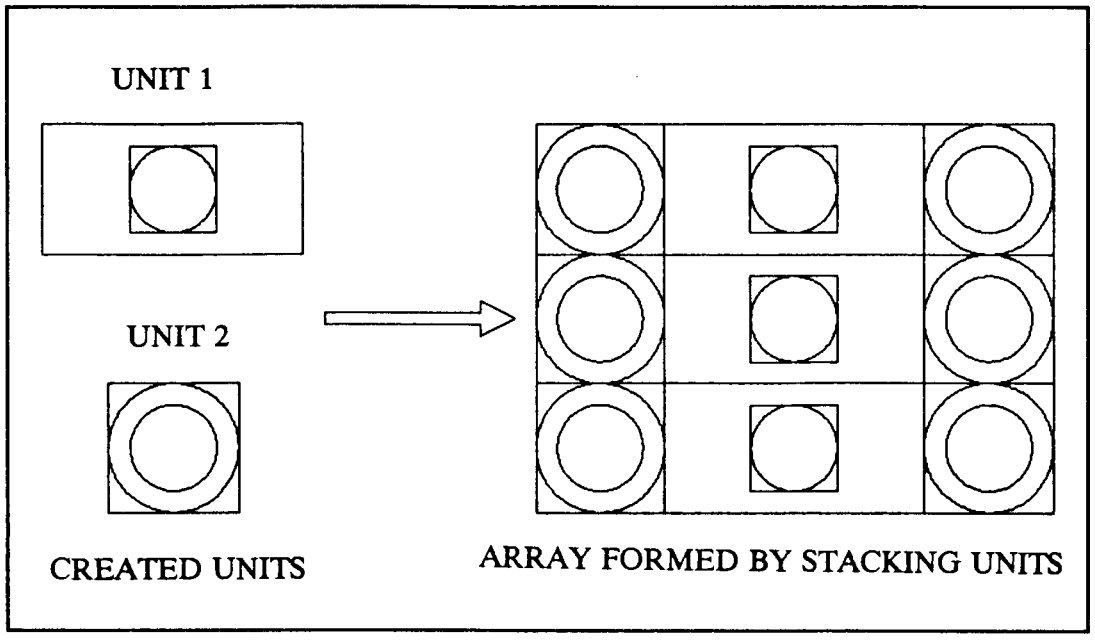

Figure 2.3 Example of Array Construction

KENO-V.a also allows the use of albedos on the outer surfaces of the system. In addition to the standard mirror and periodic albedos, differential albedos are available for different thickness of concrete, polyethylene, graphite, paraffin, water, and heavy water. The differential albedos were generated using the Hansen-Roach 16-group cross-sections. By matching lethargy boundaries, the differential albedos can be used with different cross-section data.

2.2 Cross-Section Data

In KENO-V.a the cross-section data consists of three components for each material: group-to-group scattering marginal probabilities, transfer-dependent scattering angle conditional probabilities, and corresponding transfer-dependent scattering angles. The probabilities are stored as sets of cumulative distribution functions (CDFs). The angles are stored as cosines of the outgoing particle azimuthal angle as measured from the angle of the incoming particle.

The angular distribution of scattered particles is represented using discrete scattering angles. The angles and probabilities are generated by preserving the moments of the angular scattering distributions for each group-to-group transfer. The moments are derived from the coefficients of a P_n Legendre polynomial expansion. For n discrete scattering angles, the first 2n - 1 moments are preserved. A one to one correspondence exists such that n Legendre coefficients yields n moments.

New energy groups and angles are calculated for each particle when it undergoes a collision. Since both energy and angle probabilities are stored as CDFs, a table search must be done for each CDF each time a new energy and angle is required. This involves generating two random numbers, one for the energy probability table and one for the angle probability table. First a new energy group is determined, then a new angle is determined based on the scattering angle

probabilities of the new energy group. To reduce the size of the table searches, angle probabilities are stored from the most probable to the least probable.

2.3 Options

KENO-V.a contains two distinct types of options that are specified in the parameter data. The first option type enables one to classify the problem being run. This set of options includes specifying such things as the seed random number, maximum problem execution time, default weights, number of generations to be run, number of histories per generation, neutron and fission bank size, number of words of storage allocated to the problem, and specifying the problem as an adjoint calculation. Although these options are important to the type and quality of results obtained, the final output consists of only the default calculations.

The second option type calculates and outputs additional information. Although KENO-V.a is a criticality safety code designed to calculate the k_{eff} of a system, several options are available which enable other characteristics of a problem to be calculated such as fission densities, flux densities, fissions and absorption by region, k_{eff} by unit or hole, fission production matrix by array or array position, fission production matrix by unit, matrix k_{eff} by unit type or location, and the average number of neutrons by fission to name just a few. Most of these options can be used with a

minimum of additional CPU time. However, the matrix co-factor option for large problems may as much as double the CPU time of the problem.

2.4 Particle Tracking

KENO-V.a contains 175 subroutines. It can be run as a module in the SCALE system or as a stand alone program. The heart of KENO-V.a consists of the particle tracking subroutines TRACK and CROS which can take up to 98% of the CPU time required to solve a problem. A history consists of following the entire path a particle through the system from birth to elimination. Subroutine TRACK follows a particle through the geometry of the system calling CROS to determine if a particle crosses a boundary and if so where on the boundary the crossing occurs. All of the particle history data for a generation are contained in one large two-dimensional array called NUBANK. Prior to processing a history, the data describing the history are transferred to a set of variables contained in the common block NUTRON. These variables are updated as the particle travels through the system. History data are transferred between subroutines through the common block NUTRON. When a particle is either eliminated using Russian roulette or leaks from the system, the next particle's data are transferred from the array NUBANK to the variables in the common block NUTRON.

Although at first glance there appears to be little

structure to the particle tracking algorithm used, TRACK is actually divided into 10 distinct sections: FSTART, PATH, INWARD, FINDBOX, POSIT, OUTWARD, COLLISION, FISSION, ARRAY, and ALBEDO. Unfortunately, the logic used to transfer particles between these sections and process the particles in each section does not vectorize. In order to give an understanding of how the scalar version of TRACK functions, each of these sections will be briefly discussed. Figure 2.4 is a logical flow chart of subroutine TRACK and should be kept in mind while reading about each section.

FSTART is the section of TRACK that transfers the data for the particle history to be processed from the array NUBANK to the common block NUTRON. All pertinent variables are initialized and logical flags set. The history could be transferred to different locations at this point depending on its status. A new history would proceed directly to the PATH section. A history that was previously in the system, such as one that was transferred to a new supergroup, would first undergo Russian roulette and if it survives be transferred to the PATH section. If the history is a previously stored split history it proceeds to the XSEC section of TRACK.

The PATH section of TRACK determines the distance a particle would travel assuming the material in the region containing the particle was infinite in extent. If the remaining path length of the particle is zero, such as for a new particle or one just having a collision, a new path

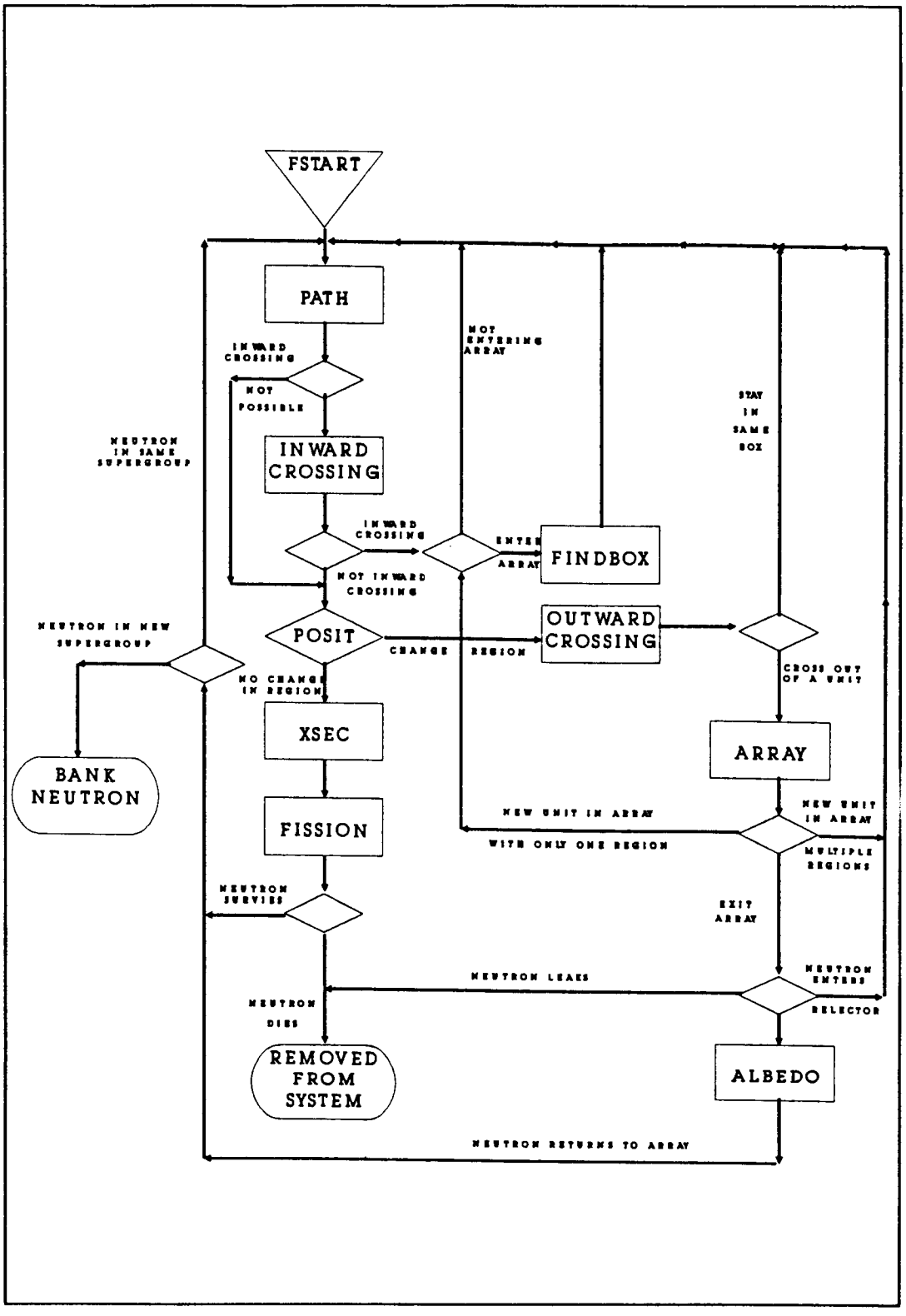

Figure 2.4 Logical Flow Chart for Subroutine TRACK

length consisting of the number of mean free paths the particle will travel to its next collision is calculated based on a random selection from an exponential distribution. The distance travelled is the number of mean free paths remaining divided by the macroscopic total cross section of the mixture contained in the region. If the region is a void, the path length is the maximum chord length of the system.

The INWARD section of TRACK determines if the particle could undergo an inward crossing. If an inward crossing is possible, subroutine CROS is called which determines if an inward crossing occurred and if so where on the boundary the crossing occurred. Prior to calling CROS, an indicator is set which informs CROS to check for an inward crossing, and the appropriate geometry information is transferred to the common block NUTRON. When CROS is completed, it sets an indicator which informs TRACK if a crossing occurred. If holes are present in the original region, each hole is checked in turn for a possible inward crossing. The crossing involving the shortest distance is selected as the actual crossing. If a crossing has occurred the remaining mean free paths are calculated, the position and region are updated, and the flux and neutron age summed. If the particle crossed into the innermost region in the unit and no holes are present in the region, a flag is set indicating inner crossing is now not possible. If a crossing has not

occurred, the particle proceeds to the POSIT section; otherwise, it proceeds to the FINDBX section.

The FINDBX section determines if a particle enters a new array. If a new array is entered, the array position is calculated, the new unit and region location are determined, and the particle's coordinates are transformed. The particle then proceeds back to the PATH section.

The POSIT section determines if a particle has a collision in the same region in which it is currently contained. If the collision occurs in the current region, the particle is transferred to the XSEC section; otherwise, it goes to the OUTWARD section.

The OUTWARD section of TRACK is called only after the POSIT section determines that a particle did not have a collision in the current region. The only option left is for the particle to cross out of the current region. The OUTWARD section determines where on the boundary the crossing occurs and the new region the particle enters. A flag is set indicating this is an outward crossing, the appropriate geometry data are transferred to the NUTRON common block, and subroutine CROS is called so that the boundary crossing coordinates can be determined. The particle history contributions to the flux and neutron age are then summed, the number of mean free paths remaining are updated, and the region number is incremented. If the new region is not the outermost region in the unit, the particle proceeds to the

PATH section. If the current region is the last region in the unit the particle crosses out of the unit and is transferred to the ARRAY section. If the problem is a single unit problem or the particle is in the external reflector, the history proceeds to the ALBEDO section. If the particle exits a unit that is specified as an array position, the history proceeds to the ARRAY section.

The XSEC section of TRACK is used to process a history after the particle suffers a collision. First the neutron age and flux are summed, and the remaining path length set to zero. The absorption weight, fission weight, self-multiplication, and the contribution to the average number of neutrons are calculated. A new particle weight is calculated using the nonabsorption probability and a new energy group and new direction cosines are chosen. The particle undergoes either Russian roulette or splitting if appropriate. If the particle is split, the new particle is stored in the neutron bank NUBANK. If the particle is terminated in Russian roulette, a new particle begins in FSTART. The surviving particle is transferred to the FISSION section if the collision occurs in a region containing fissile material; otherwise, it is transferred to the PATH section.

The FISSION section is entered only if a particle has a collision in a region containing fissile material. This section generates the fission neutrons used in the next generation. To ensure sufficient fission neutrons are

generated to fill the neutron bank, a minimum production factor (MPF), calculated at the beginning of each generation, is defined to be:

$$MFP = \bar{k}\left(1.0 - \frac{3.0}{\sqrt{HG}}\right) \qquad (2.1)$$

where: $\bar{k}$ *is the running average of the system* k_{eff} *calculated after each generation*

HG is the number of histories per generation.

Equation 2.1 represents the 99% confidence interval lower limit for the distribution of the generation k_{eff}.

Upon entering the FISSION section, a random number is generated and a pseudo fission weight, defined as the fission weight divided by the random number, is calculated. If this result is less than the minimum production factor, the history either proceeds to the PATH section or is stored in the neutron bank if it leaves the current supergroup. If the history remains and its fission weight is greater than the minimum production factor, the pseudo fission weight is redefined as the production factor divided by the random number and, if the fission bank is not full, the appropriate neutron data are stored in the fission bank. If the fission bank if full, the bank is searched for the smallest pseudo fission weight which is compared to the current pseudo fission weight. The fission point having the larger pseudo fission weight is stored in the fission bank. After the

fission point has been stored, the fission weight is decremented by the minimum production factor. If the remaining fission weight is greater than zero, the FISSION section is repeated; otherwise, the particle proceeds to the PATH section.

The ARRAY section processes particles that cross out of a unit or boxtype. The particle is first checked to determine if it crossed out of a unit in an array or a hole. If the particle crossed out of a hole, the region and unit are updated, the coordinates are transformed to those of the new unit, and the history proceeds to the PATH section. If the unit exited was an array position, each face of the unit is checked in sequence and the particle is forced to the appropriate face. If the particle remains in the array, the new array position, region, and unit are updated and the coordinates are transformed to those of the new unit. If the new unit contains only one region, the particle is transferred to FINDBX and checked to see if it is crossing into another array; otherwise, it goes to PATH.

A particle leaving an array will either cross into another region or leave the system if it is a global array. If the particle crosses into another region, the region and unit are updated and the coordinates are transformed. A particle crossing out of a global array either leaves the system and is terminated or, if albedos are present, proceeds to the ALBEDO section.

The ALBEDO section is used to place albedo boundary conditions on the outer faces of a system. It can only be used if the outermost region of the system is a rectangular parallelpiped. When a particle enters the ALBEDO section, each boundary face is checked in sequence to determine if the history leaks from the system in which case it is terminated, undergoes mirror reflection, undergoes periodic reflection, or undergoes a differential albedo reflection.

A particle undergoing periodic reflection is moved to the opposite face of the system leaving the energy and direction cosines unchanged. A particle undergoing mirror reflection has the sign of direction cosines changed. A history undergoing a differential albedo has its energy, weight, and angles updated based on the material represented by the differential albedo. After being updated by an albedo the history is transferred to the PATH section.

2.5 Boundary Crossing

As a particle traverses a system in subroutine TRACK, the particle will cross from one material region to another. Whenever a particle history has a potential for crossing a boundary, subroutine CROS is called. The current particle position, particle position at the end of the path (called it's flight position), crossing region geometry data, and crossing type are sent to subroutine CROS. CROS then determines if the particle does indeed cross a boundary and

if it does, at what location on the boundary the crossing occurs and the fraction of path length used to reach the crossing site.

Although each of the 24 geometry types listed in Table 2.1 could undergo both inward and outward crossings, there are only 10 distinct possible crossing types to consider, 5 inward and 5 outward. The 5 inward and outward types consist of spheres, cylinders, cuboids, hemispheres, and hemicylinders. The number of crossing types is reduced to this number by rotating the geometry of the cylinders, hemispheres, and hemicylinders in which a potential crossing occurs to a given coordinate system, processing the potential crossing, and rotating the geometry back to the original coordinate system. The logic used to process each crossing type is examined below.

The crossing into or out of a cuboid is the simplest of the crossing types to understand. The particle's initial and flight positions are examined relative to each face of the cuboid. For a face lying between the particle's initial and flight positions, the distance between the particle's initial position and the face is calculated and the position where the particle crosses the face is determined. A outward crossing occurs when a face is between the two particle locations and the crossing location is within the dimensions of the facial area. There is only one location where this is possible for an outward crossing. An inward crossing could

meet the above criteria on more than one face so an additional constraint is included. The possible crossing face nearest to the particle's initial position is the cuboid face which is crossed. If a crossing occurs, a flag is set, the particle's position is updated, and the fraction of the path length used is calculated. The fraction of the path length used is simply the path length to the face divided by the total path length.

Sphere crossings are not as straight forward. First the x, y, and z coordinate distances between the flight position and initial position of the particle are calculated and set as the variables r, s, and t. The particles initial position (x,y,z) is shifted so that the system origin is the center of the sphere. The particle crossing position on the sphere can then be represented as follows:

$$x_r = x + \eta * r \tag{2.2}$$

$$y_r = y + \eta * s \tag{2.3}$$

$$z_r = z + \eta * t \tag{2.4}$$

where: x_r is the x crossing position on the sphere

y_r is the y crossing position on the sphere

z_r is the z crossing position on the sphere

η is the fraction of path length to the sphere.

Recalling that the equation which describes a sphere is

$$R^2 = x_r^2 + y_r^2 + z_r^2 \tag{2.5}$$

where R is the radius of the sphere,

inserting equations 2.2, 2.3, and 2.4 into equation 2.5

yields the following equation

$$R^2 = (x + \eta*r)^2 + (y + \eta*s)^2 + (z + \eta*t)^2. \quad (2.6)$$

Rearranging equation 2.6 by combining all terms involving each power of η yields

$$\eta^2*(r^2+s^2+t^2) + 2*\eta*(r*x+s*y+t*z) + (x^2+y^2+z^2-R^2) = 0. \quad (2.7)$$

By setting the terms in brackets to the following values

$$p = r^2 + s^2 + t^2 \quad (2.8)$$

$$q = r*x + s*y + t*z \quad (2.9)$$

$$f = x^2 + y^2 + z^2 - R^2 \quad (2.10)$$

the following simple quadratic equation is formed:

$$p*\eta^2 + 2*q*\eta + f = 0. \quad (2.11)$$

Equation 2.11 can easily be solved for η as follows:

$$\eta = -q \pm \frac{\sqrt{q^2 - p*f}}{p} \quad (2.12)$$

Equation 2.12 is actually two solutions since there is both a + and - sign in front of the square root. Crossings only occur if η is positive and less than 1.0. For both inward and outward crossings, the value inside the square root must be positive for a crossing to occur.

For inward crossings the form of equation 2.12 with the minus sign in front of the square root is used. For a positive η to occur, the value of q must be positive. If $q \geq 0$, η is calculate using equation 2.12. If $0.0 \leq \eta \leq 1.0$, a crossing has occurred. A flag is then set, indicating a crossing, and the coordinates and the path length are updated

using the calculated value of η.

For an outward crossing, the form of equation 2.12 with the plus sign in front of the square root is used. If $0.0 \leq \eta \leq 1.0$, a crossing has occurred. A flag is then set, indicating a crossing, and the coordinates and the path length are updated using the calculated value of η.

Hemispheres use the same equations as spheres to calculate a crossing with some variation. First the hemisphere is rotated so the spherical section is in the +z direction. The spherical section of the hemisphere is checked for a crossing in the same manner as the sphere while ensuring the crossing occurrs above the hemispherical face. The flat hemispherical face is then checked for a crossing using the method described above for a cuboid. If a crossing does occur, a crossing flag is set, the position and path length are updated, and the position is rotated back to the original coordinate system.

Cylinder and hemicylinder crossings employ a combination of the methodologies used for sphere and cuboid crossings. First the cylinder is rotated so that the center lies along the z-axis. The flight distances r, s, and t are defined the same way they are for a sphere. The cylinder crossing location is x_r, y_r, and z_r are defined using equations 2.2 and 2.3 as they are for a sphere. Now using the equation representing a circle and replacing x_r and y_r by their equivalences shown in equation 2.2 and 2.3, one gets

$$R^2 = (x + \eta*r)^2 + (y + \eta*s)^2 \quad (2.13)$$

where R is the radius of a circle.

Rearranging equation 2.13 yields an equation very similar to equation 2.7,

$$\eta^2*(r^2 + s^2) + 2*\eta*(r*x + s*y) + (x^2 + y^2 - R^2) \quad (2.14)$$

By defining the terms in the brackets as follows:

$$p = r^2 + s^2 \quad (2.15)$$

$$q = r*x + s*y \quad (2.16)$$

$$f = x^2 + y^2 - R^2 \quad (2.17)$$

an equation identical to equation 2.11 can be derived. The value of η can then be calculated using equation 2.12 substituting in the values derived in equations 2.15, 2.16, and 2.17. Just as for spheres, crossings only occur if η is positive and less than 1.0. For both inward and outward crossings, the value inside the square root must be positive for a crossing to occur.

For inward crossings the form of equation 2.12 with the minus sign in front of the square root is used. For a positive η to occur, the value of q must be positive. If $q \geq 0$, η is calculated using equation 2.12. If $0.0 \leq \eta \leq 1.0$ and the crossing location is between the top and bottom of the cylinder, a crossing has occurred on the curved surface. If a crossing does not occur on the curved surface, both ends are checked for possible crossings as is done for an inward crossing check of a cuboid. If a crossing occurs, a flag is

set indicating a crossing, the position and path length are updated using the calculated value of η, and the position is rotated back to the original coordinate system.

For an outward crossing, the form of equation 2.12 with the plus sign in front of the square root is used. If $0.0 \leq \eta \leq 1.0$ and the crossing location is between the top and bottom of the cylinder, a crossing has occurred on the curved surface. If a crossing does not occur on the curved surface, both ends are checked for possible crossings as is done for an outward crossing check of a cuboid. If a crossing occurs, a flag is set indicating a crossing, the position and path length are updated using the calculated value of η, and the position is rotated back to the original coordinate system.

Hemicylinders are checked for crossings using the same set of equations as cylinders. First the hemicylinders are rotated so the hemicylinder runs along the z-axis with the curved face in the +x direction. The particles are checked for curved surface crossings ensuring any crossing of this type occurs on the correct side of the hemicylinder face. Crossings of the hemicylinder face and both ends are then checked. If a crossing occurs, a flag is set indicating a crossing, the position and path length are updated using the calculated value of η, and the position is rotated back to the original coordinate system.

CHAPTER 3

VECTORIZED KENO-V.A

Since up to 98% of the scalar KENO-V.a CPU time is used for history tracking, the approach used to create a vectorized version of KENO-V.a is to vectorize the subroutines TRACK and CROS and to only alter the other subroutines which require modifications as a result of the changes to TRACK and CROS. The end result is that 42 of the 175 subroutines have been changed due to the vectorization of TRACK and CROS.

3.1 Data Storage and Handling Modifications

Modifications in the method used to store and retrieve both history and cross-section data are the major reason a majority of the subroutines were altered. History and cross-section data have been arranged so that new energy groups and angles can be retrieved as vectors.

In the current scalar version of KENO-V.a, cross-section data consist of three components for each material in the problem: group-to-group scattering probabilities, transfer-dependent scattering angle probabilities, and transfer-dependent scattering angles. Each set of probabilities is stored as a set of discrete cumulative distribution functions (CDFs) and the angles are stored as cosines of the difference between the incoming particle angle and the outgoing particle

angle. A method was devised and implemented to store the cross-section data as a combination of a discrete probability density function (PDF) and a conditional probability density function.[8,10,28]

The cross-section data now consists of a set of three numbers for each energy/angle combination (q_k, x_k, y_k), where k is the energy/angle combination number, X and y represent possible energy/angle combinations, and q is the probability of selecting x over y. An energy/angle combination, x or y, is stored as one number where the integer part of the number is the new energy group and the fractional part of the number represents the new relative outgoing angle. This storage strategy allows new energy/angle combinations to be retrieved as vectors by generating two sets of random numbers and retrieving two sets of numbers. As well as being a useful method for storing and retrieving data as vectors, this method also produces up to a 9% decrease in CPU time when implemented into the scalar version of KENO-V.a. The only disadvantage of this strategy is a 1/3 increase in the storage requirements for the cross-section data.

Devising and implementing a system to store and retrieve history data was difficult because of the number of subroutines which utilize these data. Many of the subroutines were modified only because of the new method for storing data. As previously mentioned, in the scalar version of KENO-V.a, all history data are stored in one large array.

Prior to processing a history, the data for that history are transferred to a set of variables in a common block which is used to transfer data between subroutines.

This method of data handling will not work when data are processed and transferred as vectors due to use of the common block. Common blocks contain a fixed amount and type of data. Unfortunately, the length of the vectors being processed vary greatly with problem type, number of histories specified, particular event being processed, and length of time the problem has been processing the generation. Thus, the common block was eliminated and data are transferred directly between subroutines via argument lists. Transferring the entire set of history data between subroutines is inefficient since most subroutines need only a fraction of the history data.

In the vectorized version of KENO-V.a, history data are stored and processed as a set of one dimensional vectors with each vector representing one component of the history data.[29] For example, the x-position of all particles constitutes a single vector whose length is the number of particles in a generation. Each particle is represented by a pointer which represents the location of the data used to specify the particle in each vector. The data for a particle is retrieved and stored throughout the life of the particle using its pointer. Only the data needed to perform the specified operations are retrieved and updated data are

scattered back into the storage vectors using the particle pointers. Since each component is a separate vector, only the vectors containing the required data are transferred between subroutines in the argument lists.

3.2 Modifications to Subroutine TRACK

TRACK is the subroutine that controls the processing of history data as all the histories in a generation simultaneously traverse the system specified in a problem. TRACK employs an event-based, stack-driven, all-zone, tagged-particle Monte Carlo algorithm.[30] In addition to the vectors used to store the history data and problem description, TRACK contains 13 intermediate integer storage vectors, 24 intermediate real storage vectors, and 1 intermediate logical storage vector each of the same length as the history storage vectors. These intermediate storage vectors are used throughout TRACK as temporary storage for history data and intermediate results as histories are processed.

TRACK is divided into four main events or sections: free-flight, inward-crossing, outward-crossing, and collision/fission. Each section possesses a pointer stack containing the histories that are to be processed by that section. A fifth pointer stack, kill, is used to store history pointers that have either leaked from the system or have been killed by Russian roulette. Pointers from the kill stack are used when new histories are generated by splitting.

All histories that are not in the kill pointer stack are in one and only one of the four main event stacks. Each of the 4 main sections will be discussed separately and then how the 4 sections function together to track particle histories is examined.

3.2.1 Free-Flight Section

The free-flight section of the vectorized version of TRACK performs the same function as the PATH section in the original version of TRACK plus Russian roulette and splitting. Unlike the original version of KENO-V.a, the free-flight section is the only section which performs Russian roulette or splitting. A logical flow chart of the free-flight section of TRACK is shown in figure 3.1.

The free-flight Section contains two distinct subsections, Russian roulette and Crossing. The first subsection determines if any of the particle weights fall between the maximum and minimum weights of the regions containing the histories. Particles having weights that are within these limits proceed to the crossing subsection. Particles with weights below the minimum weight of their region are processed using Russian roulette and the surviving particles, with increased weights, proceed to the crossing subsection. Particles with weights above the maximum weight of their region are split using positions contained in the kill stack. The split weights are again checked to determine

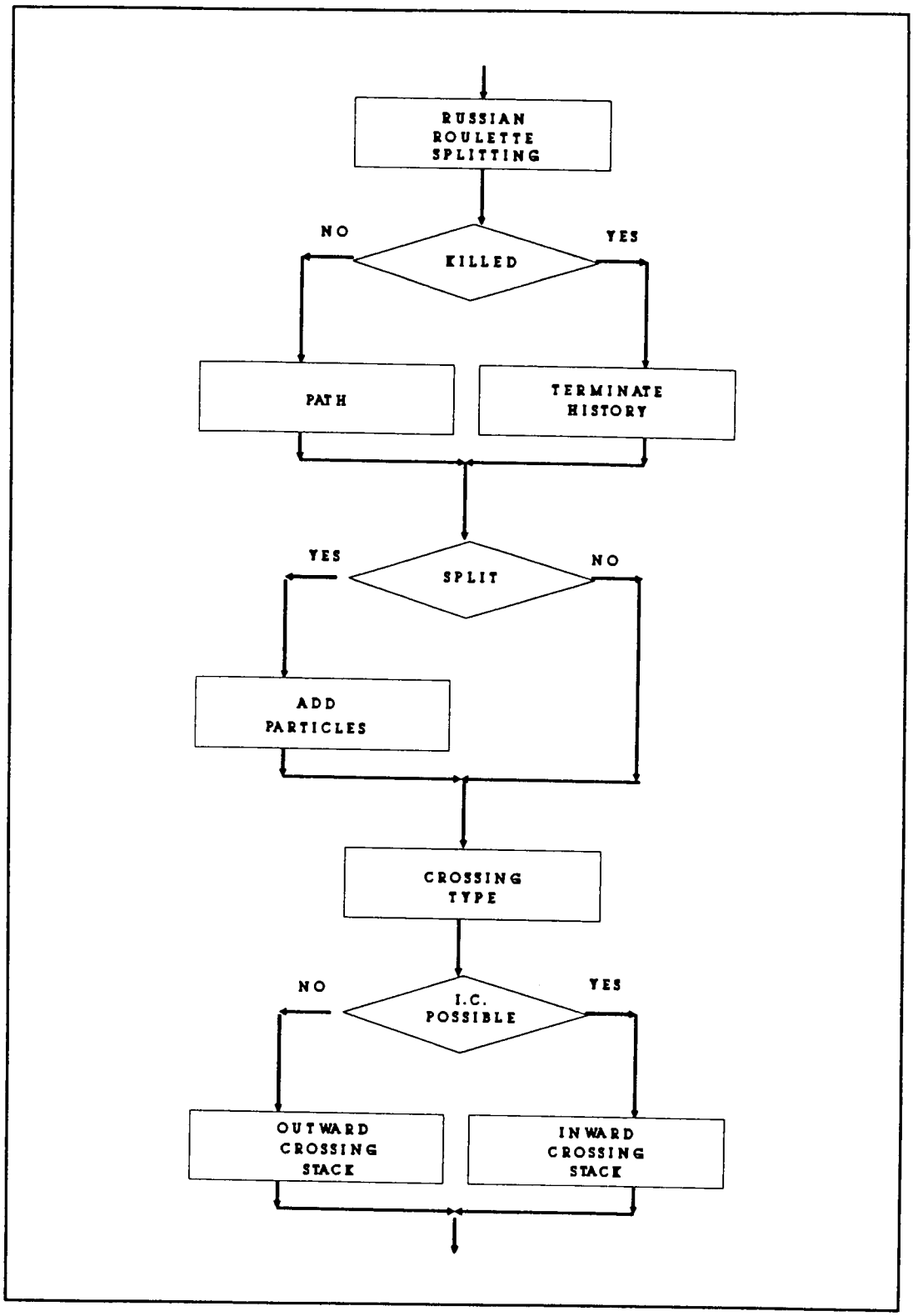

Figure 3.1 Logical Flow Chart for the Free-Flight Section of the Vectorized Subroutine TRACK

if they now fall between the maximum and minimum weight limits. Splitting will continue until the weights fall within the acceptable range at which time the original and newly generated particles will proceed to the crossing subsection.

The crossing subsection first determines the path length each particle would travel assuming the materials are infinite in extent. This is done the same way as it is done in the original PATH section except it is done for all the particle histories in the free-flight stack simultaneously. Theoretical collision sites for each history are then calculated based on each particle's path length. Finally, each history is checked to see if an inward-crossing is possible. Inward crossings are not possible if a particle is in the innermost region in a unit or if the previous event is an outward crossing and there are no holes in the region containing the particle. All history pointers that could possibly undergo an inward crossing are transferred to the inward-crossing pointer stack and all remaining history pointers are transferred to the outward-crossing pointer stack. The free-flight pointer stack is then zeroed and the section with the largest number of history pointers is processed next.

3.2.2 Inward-Crossing Section

The inward-crossing section of the vectorized version of

TRACK performs the same function as the INWARD and FINDBOX sections in the original version of TRACK. A logical flow chart of the free-flight section of the subroutine TRACK is shown in figure 3.2.

The inward-crossing Section contains the logic that determines if particles cross into a region entirely contained by the region containing the histories. Arrays or holes require additional checks to determine if the particles could possibly cross into them. If arrays are present in the problem, the first step is to transfer the pointers of all particles crossing into an array into a temporary pointer stack and into the free-flight pointer stack. Next, if holes are present in the problem, the particles which could cross into the next inner region are collected. A particle that has just crossed out of the inner region, into a region containing holes, can still undergo an inner crossing into a hole but may not cross back into the inner region just exited. The particles then being checked for a possible crossing into the next inner region are separated by the next inner region geometry type. There are 9 possible types of geometry crossing in the vectorized version of KENO-V.a: cuboids, spheres, hemispheres, x-cylinders, y-cylinders, z-cylinders, x-hemicylinders, y-hemicylinders, and z-hemicylinders. Subroutine CROS is called to determine which, if any, histories actually cross into the inner region.

Hole checking is part of the inner crossing check of

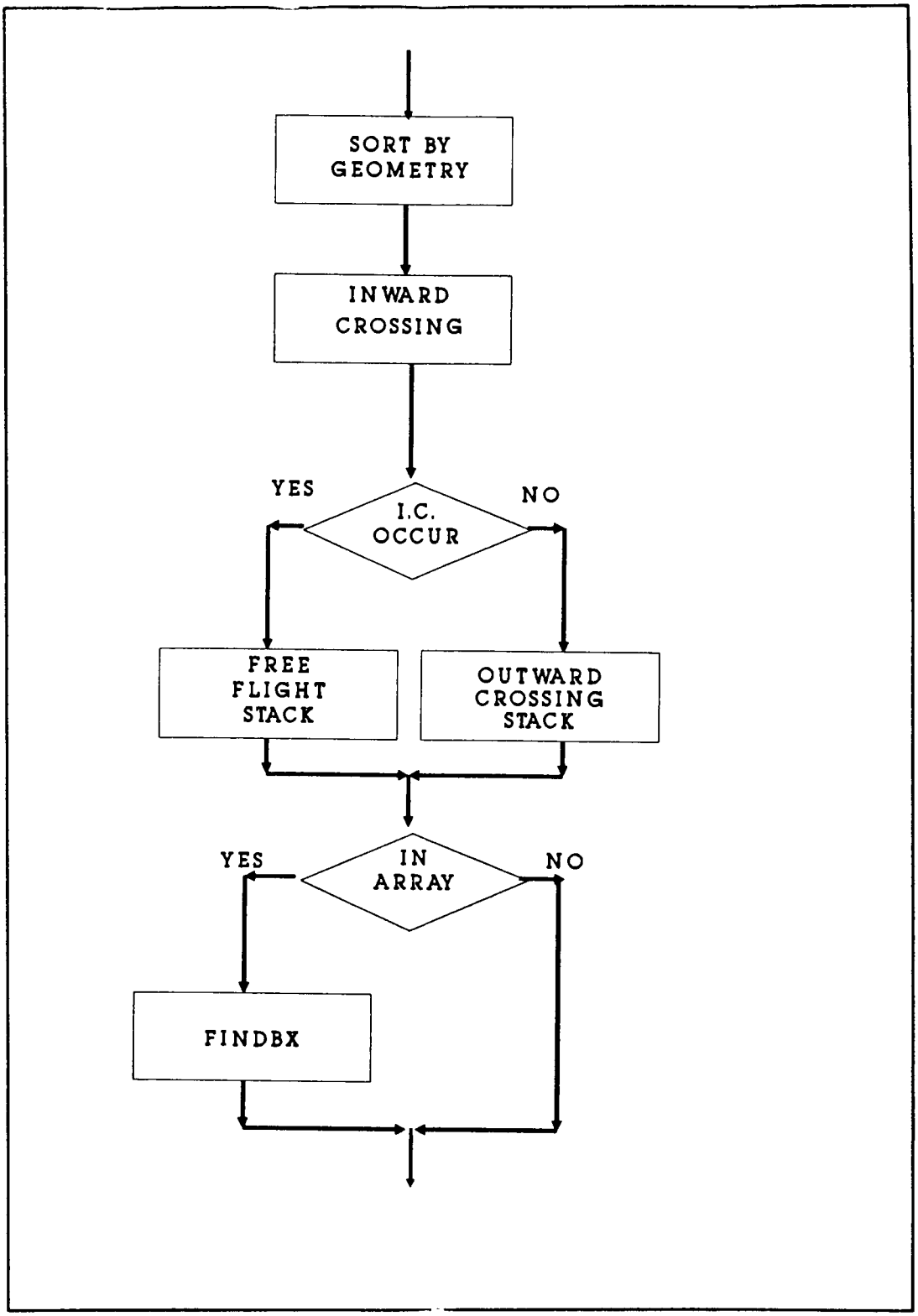

Figure 3.2 Logical Flow Chart for the Inward-Crossing Section of the Vectorized Subroutine Track

this section. If holes are present, the module then determines if the histories cross into one of the holes. This involves collecting all the histories in a region and checking each hole in that region for inward crossings by calling CROS for each hole. After each hole is checked, for each particle having a crossing into the hole, the distance between the particle's initial position and the crossing position is determined. If a history has not had a previous inner crossing or the new crossing distance is less than a previously calculated crossing distance, the history position is updated with the new inner crossing. After each hole in a region is completed, the histories for the next region are collected and the process is repeated. This continues until all holes in all regions have been checked for possible inner crossings.

After all regions and holes have been checked for inner crossings, the pointers for all histories that have had an inner crossing are transferred to the free-flight pointer stack and the remaining pointers are collected in the outward-crossing pointer stack. Also, if arrays are present, all particles that have crossed into a hole composed of an array are added to the temporary array pointer stack which is processed next.

All particles crossing into an array are processed in a subsection called FINDBX. This section determines the new array position for each particle. Once the array positions

have been determined, the new region, unit, and coordinates of each particle are determined.

Finally, the inward-crossing pointer stack is zeroed and the module with the largest number of history pointers is processed next.

3.2.3 Outward-Crossing Section

The outward-crossing section of the vectorized version of TRACK performs the same function as the POSIT, OUTWARD, ARRAY, and ALBEDO sections in the original version of TRACK. A logical flow chart of the free-flight section of TRACK is shown in figure 3.3.

The outward-crossing Section is divided into two distinct subsections: POSIT, which determines if the particles have collisions within their respective regions and if so where; and CROSSING, which determines the location where the particles cross out of their respective regions, and updates the appropriate history data such as region, unit, position, and array and hole if appropriate. This module also terminates a history by transferring its pointer to the kill pointer stack if the particle leaves the system.

POSIT first sorts the history pointers by the geometry type of their respective regions. The program then loops over the geometry types to determine if the histories have collisions in their respective regions. The pointers of the histories that undergo collisions in their respective regions

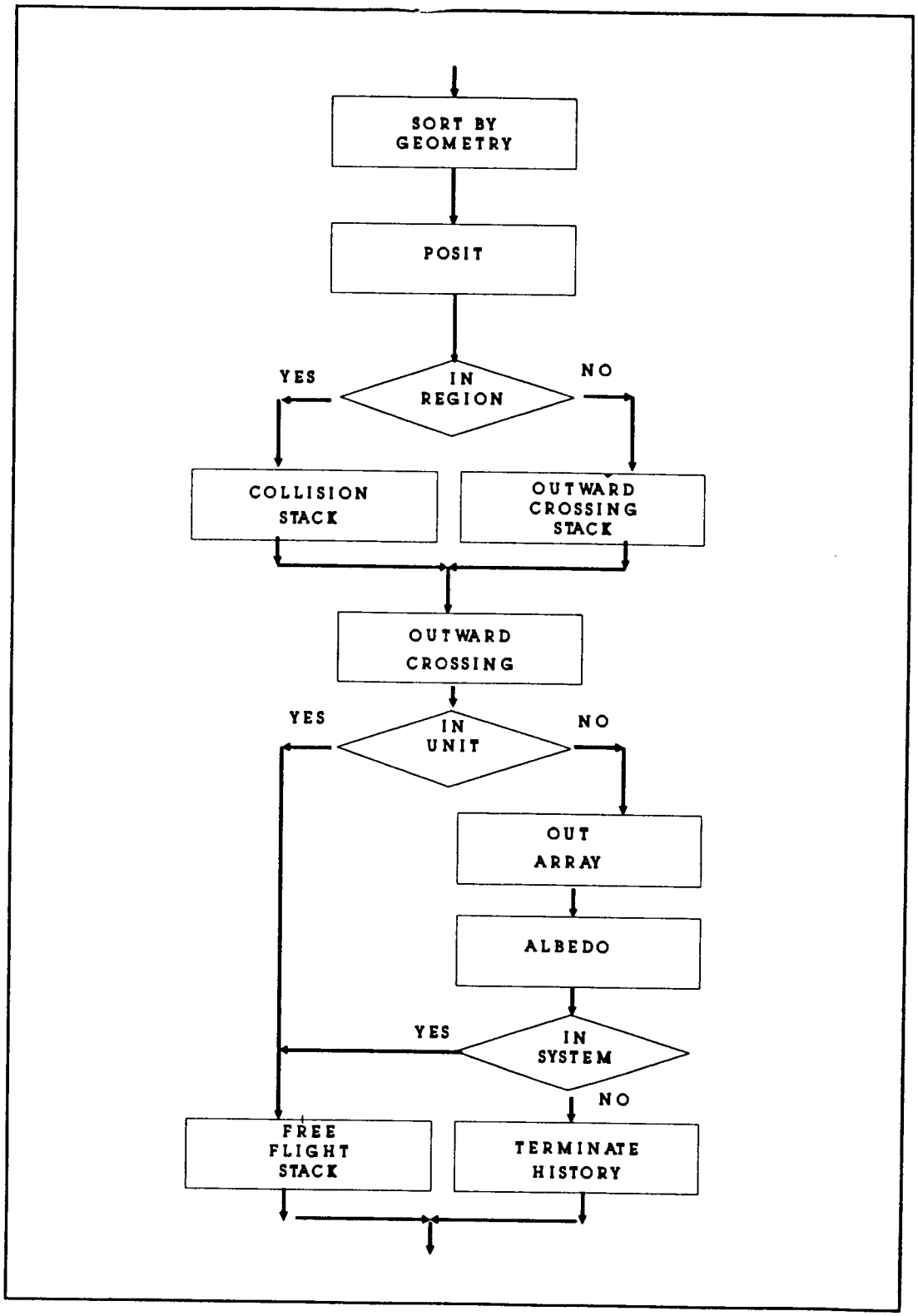

Figure 3.3 Logical Flow chart for the Outward-Crossing Section of the Vectorized Subroutine TRACK

are transferred to the collision stack. The crossing position of the remaining histories, still sorted by geometry type, are determined by calling the subroutine CROS.

Several possibilities exist for particles having undergone an outward crossing. Particles that were not initially in the outermost region of their respective units simply cross over to the next outer region. For these particles, their region number is incremented, position updated, and the pointers are transferred to the free-flight stack. The remaining particles have crossed out of their respective units and have either crossed out of the system, crossed out of a hole, or are changing their positions in an array. Particles crossing out of the system have their respective pointers transferred to a temporary pointer stack. For particles crossing out of a hole, the particles enter the regions surrounding the respective holes. These particles have their region, unit, and position updated, and their respective pointers transferred to the free-flight pointer stack.

If a particle crosses out of a unit that is in an array position, the position face each particle crossed is first determined. Given the crossing face and the original array position for each particle, new array positions are calculated. For the particles not exiting the array, new array positions and coordinates are determined, the appropriate history data are updated, and their respective

pointers are transferred to the free-flight pointer stack. The remaining particles exiting the arrays either leak from the system if they exit the global array or enter the surrounding region. The pointers for particles exiting a global array are transferred to a temporary pointer stack. For particles exiting the array but not the system, the appropriate history data are updated and their respective pointers are transferred to the free-flight stack. If there are no albedos in the problem, the pointers in the temporary pointer stack are transferred to the kill pointer stack.

There remains one additional type of interaction that could occur to outward-crossing particles. If any outer face of the outermost or global array or unit has an albedo, the particles in the temporary pointer stack are checked to see if they cross a boundary having an albedo. Albedos can only be used when the outermost region of the global array or unit is a cuboid. A different albedo may exist on each of the six faces of the outermost region. For particles crossing a surface represented by an albedo, the particles are processed by albedo type and crossing face. Three classes of albedos exist in KENO-V.a: periodic, mirror, and differential. If albedos exist in the problem and particles are crossing out of the system's outermost faces, the types of albedos present are first determined. If periodic albedos exist, the type of albedo at each face is checked. All particles crossing a face possessing a periodic albedo are processed

simultaneously by transferring the particles to the opposite side of the unit and, if appropriate, array. Next, if mirror albedos exist, the type of albedo at each face is again checked. All the particles crossing a face possessing a mirror albedo are processed simultaneously by reversing their direction cosines.

Finally, if differential albedos are present on any outer boundary in the system, each face is again checked for albedo type. When a particle is found that crosses a face possessing a differential albedo, the subroutine ALBEDO is called which updates the appropriate history data. Due to the different types of differential albedos, the rarity with which this option is used, and the fact that this section would not efficiently vectorize, differential albedos are processed in the scalar mode. After all albedo options have been checked, the pointers for the histories that crossed a face containing an albedo option are transferred to the free-flight pointer stack and the remaining pointers are transferred to the kill pointer stack.

3.2.4 Collision Section

The collision section of the vectorized version of TRACK performs the same function as the COLLISION and FISSION sections in the original version of TRACK. A logical flow chart of the free-flight section of the vectorized subroutine TRACK is shown in figure 3.4.

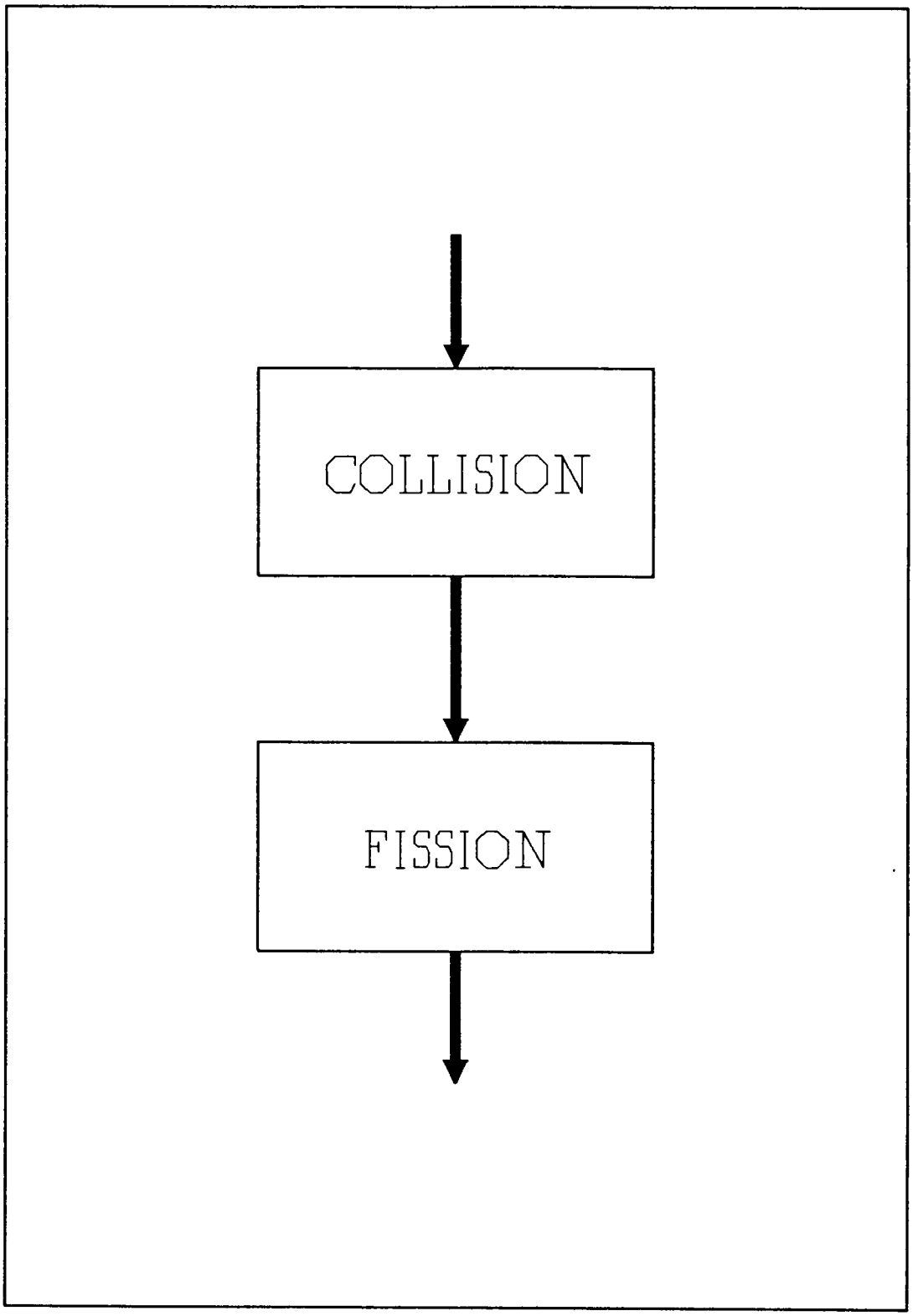

Figure 3.4 Logical Flow Chart for the Collision Section of the Vectorized Subroutine TRACK

The collision section is divided into two subsections: the collision subsection, which updates the data for all particles that have just undergone a collision, and the fission subsection, which generates and stores fission neutrons in the integer and real fission banks for use in the next generation. All history pointers are transferred to the free-flight pointer stack upon the completion of this module.

For each history which has just undergone a collision, the collision section first updates the coordinates, weights, and time. The total fission weight, total neutron lifetime, and neutron age are also summed for the current generation. Several different options such as flux densities, fission densities, absorptions, and matrix information are summed in this section. Lastly, new energies and angles are calculated for each particle using the algorithm described in section 3.1. The energy and angles of all the particles in the collision stack are done simultaneously as vectors. However, the summations such as total fission weight, neutron age, neutron lifetime, flux, and many more options cannot be done as vectors due to the recursive nature of the applicable vector loops.

The fission subsection collects all the particles that had collisions in regions containing fissile material and generates new fission particles based on their fission weights. The logic is identical to the that of the FISSION section in the original version of TRACK with one exception,

new fission particle energies are not updated until the end of the generation.

Fission particles are stored in real and integer fission banks. The real fission bank contains the x, y and z location of the particle which generated the new fission particle. The integer fission bank contains the region, energy, and array position of the particle which generated the new fission particle. In addition, the integer fission bank could contain the hole number of the particle if holes are present and the array or hole nesting level of the particle if nested arrays or holes are present.

3.2.5 Subroutine TRACK Logic

The vectorized version of subroutine TRACK performs the same functions as the original scalar version of subroutine TRACK but in a fraction of the time. This is done by tracking and processing all the particles in a generation simultaneously. The vectorized version of subroutine TRACK contains 4 main sections, each with it's own pointer stack, a fifth pointer stack called kill which contains the pointers of all histories that have either been terminated by Russian roulette or leaked from the system, the overall logic which determines which section to process next, and an initialization section which is performed at the beginning of each generation. A logical flow chart for the overall subroutine TRACK is shown in figure 3.5.

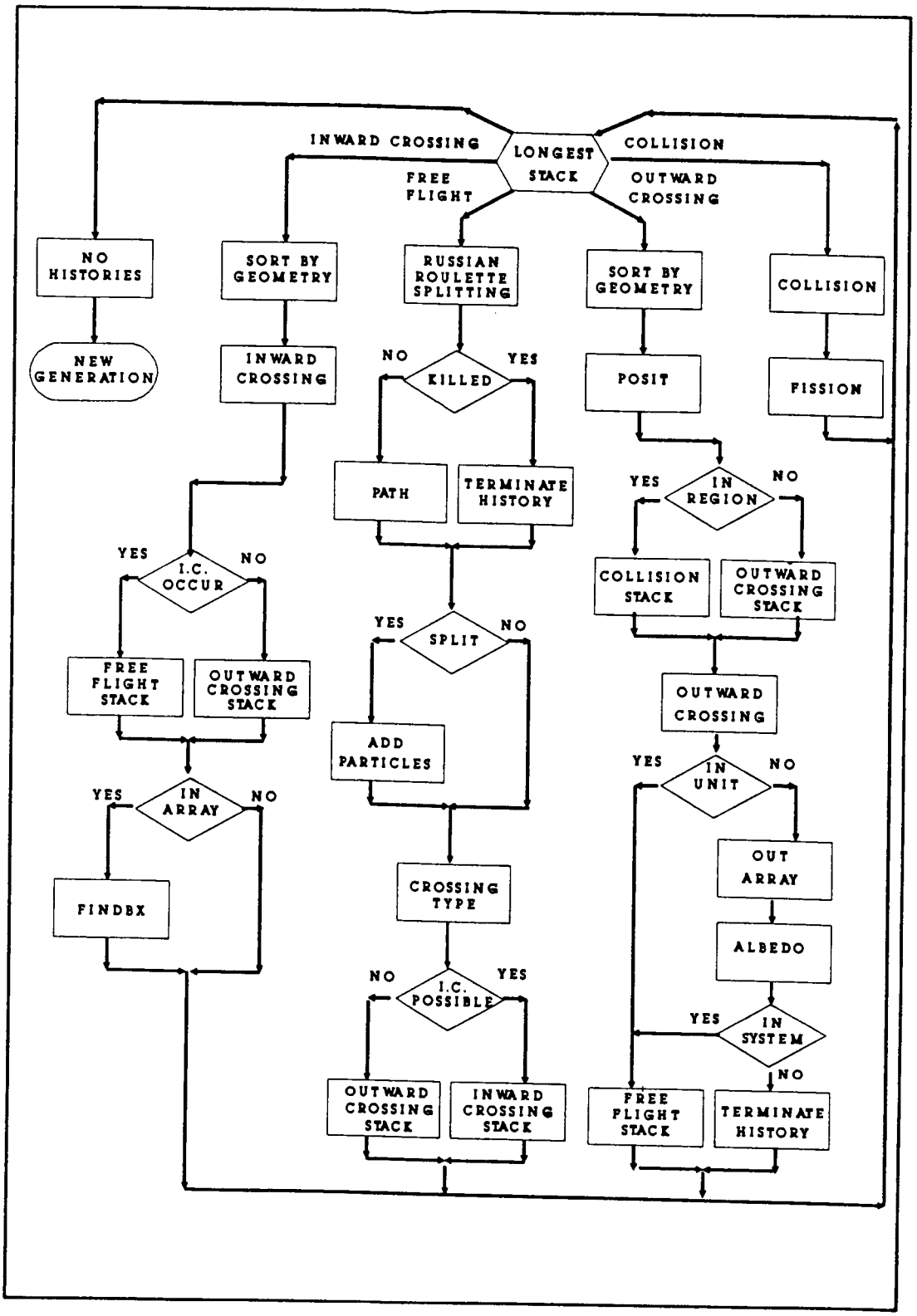

Figure 3.5 Logical Flow Chart for the Vectorized Subroutine TRACK

When subroutine TRACK is initially called, at the start of each new generation, the values of all the appropriate variables are determined and set and all the appropriate flags are set. This section is similar to the FSTART section in the original version of TRACK. An extra 25 history locations are provided by default to account for the possibility of splitting before any histories are terminated. These locations are transferred to the kill stack. After all flags and variables are set, the inward-crossing, outward-crossing, and collision stack lengths are set to zero, the free-flight stack length is set equal to the number of histories in the generation, all particle pointers are transferred to the free-flight stack, and particle tracking begins at the crossing subsection of the inward-crossing section.

All particle histories are contained in one and only one of the five pointer stacks at all times. After the particle histories begin traversing the system in the free-flight section, the control section maintains overall control regarding which of the 4 main sections is processed next. After one of the main sections is completed, its pointer stack length is set to zero and the program transfers to the control section. The control section searches the 4 main pointer stack lengths for the longest stack length. The section with the longest stack length is processed next. When all stack lengths are zero, control is passed to another

small section which calculates fission energies for all particles in the fission bank and then returns to its calling subroutine which processes the generation and prepares for the next generation. A program listing of Subroutine TRACK is contained in Appendix A.

3.3 Modifications to Subroutine CROS

CROS is the subroutine called by TRACK which determines if particles actually cross a boundary. CROS processes both inward crossings and outward crossings. The vector containing the list of history pointers as well as the ancillary information needed to process the history crossings are passed into CROS through its argument list. Particles are processed both by crossing type (i.e. inward or outward) and by the geometry of the possible crossing region. Since the crossing type is controlled by the module in TRACK being processed, a simple flag passed to CROS determines the crossing type of the current batch of histories.

The histories to be processed are contained in one vector which is sorted by geometry type. A second vector contains the list of geometry types to be processed and a third vector contains the beginning and ending positions of each geometry type in the sorted vector. This arrangement ensures that particles undergoing possible crossings of the same geometry type (i.e. cuboid, cylinder, sphere, etc.) are processed as vectors.

Particle crossing types are divided into nine separate geometry types: cuboids, spheres, hemispheres, x-cylinders, y-cylinders, z-cylinders, x-hemicylinders, y-hemicylinders, and z-hemicylinders. The cuboids, spheres, and cylinders are all processed using the coordinate system of the original histories. Since a hemisphere could be in any of six separate orientations, hemispheres are first rotated so their faces share a common orientation, then processed as a vector, and finally rotated to their original orientation. The same procedure is used for each of the three hemicylinder types. The equations used to develop the algorithms used in the original version of CROS are identical to those used in the vectorized version of cross. The primary difference is that the particle crossings are processed as vectors instead of individually.

After the histories are processed, CROS sends a flag vector back to TRACK indicating which particles successfully crossed a boundary. Along with the flag, updated x, y, and z coordinates are sent to TRACK for the particles that crossed a boundary. A program listing of Subroutine CROS is contained in Appendix B.

CHAPTER 4

RESULTS

The results presented in this dissertation are divided into three sections: speed-up due to change in cross-section data handling, vectorized KENO-V.a benchmarking, and timing studies. First the code speed-up due to the alternate method of storing and processing cross section data is examined. Three different cross-section libraries are examined. Then, for a given cross section library, the number of angles and the number of histories per generation are changed and the effect on the problem CPU time is examined. Benchmarking involves running a set of standard problems using both the original version of KENO-V.a and the vectorized version of KENO-V.a and comparing the results. Some of the test problems are mock-ups of actual critical experiments and some are artificial problems. The problem set should be diverse enough to include all geometry types and as many combinations of geometries and options as possible.

After benchmarking, several sets of problems were run to determine how speed-up factors change with the number of histories per generation. Again, different geometry configurations were examined. In these cases, the options not explicitly needed to describe the problem were removed so that the speed-up associated with history tracking is more apparent. Eliminating unnecessary options removes

nonvectorized summations which tend to reduce speed-up factors by reducing the percentage of the code that is vectorized.

4.1 Cross-Section Handling

Prior to vectorizing KENO-V.a, the method used to store and retrieve cross-section data was modified so that these data could be retrieved and processed as vectors. In addition to using less CPU time on a vector machine, the method used to store and retrieve cross-section data also requires less CPU time on a scalar computer due the replacement of the two table searches by two direct retrievals required to obtain a new energy and angle combination. The speed-up associated with changing the cross-section data storage and retrieval method is examined below. The only difference between the two versions of KENO-V.a examined is the method used to store and retrieve cross-section data. Neither method has as yet taken advantage of the available vector processors.

First, the effect different cross-section sets have on speed-up factors is examined. Four different cross-section sets are examined for a given system which consists of a highly enriched uranium solution sphere surrounded by an H_2O reflector. As shown in table 4.1, speed-up factors range from 1.0, which means no speed-up, to 1.09, which is approximately an 8% decrease in CPU time. The lack of any

speed-up using the Hansen-Roach cross-section set is a result of only one scattering angle being generated for each material thus eliminating the table search of cross-section angles. The other three cross-section libraries show significant speed-ups.

The variability in k_{eff} with respect to cross section set is due to the fact three different cross section libraries are used. The Hansen-Roach set uses the Hansen-Roach Library, which is the oldest cross section library. The 27GROUPNDF4 and 218GROUPNDF4 cross section sets are derived from the same base cross section library and therefore should

Table 4.1

Comparison of the Modified Form of Cross Section Handling to the Original Form for Different Cross Section Libraries

	Original		Modified		
Cross-Section Set	k_{eff} ($\pm\sigma$)	CPU Time (min)	k_{eff} ($\pm\sigma$)	CPU Time (min)	Speed-up Factor
Hansen-Roach	1.0475 (0.0049)	2.7613	1.0646 (0.0049)	2.7548	1.00
27GROUP NDF4	1.0719 (0.0048)	3.0003	1.0600 (0.0052)	2.7915	1.07
123GROUP MTH	1.0269 (0.0045)	2.4377	1.0211 (0.0051)	2.3997	1.02
218GROUP NDF4	1.0674 (0.0050)	3.0243	1.0619 (0.0043)	2.7788	1.09

35% Uranium (97% U-235), 65% H_2O SPHERE (20-cm diameter)
H_2O Reflector (40-cm Cube)
300 Neutrons per Generation
103 Generations

produce similar k_{eff}s within statistics (i.e. within $\pm 2\sigma$). The 123GROUPMTH cross section set comes from a third cross section library. This cross section set seems to produce a lower k_{eff} than the other three sets, possibly due to the lack of U-235 resonance correction which is present in the other sets.

For the sample problem examined above and utilizing the same cross-section set, 27GROUPNDF4, the number of scattering angles is then changed from zero, which represents isotropic scattering, to four. As shown in table 4.2 the speed-up factors vary from 1.0, representing no increase in speed, to 1.10, which corresponds to approximately a 9% decrease in CPU time. The speed-up factor jumped dramatically from 1.0 to 1.09 when more than one scattering angle is possible and then appears to increase very slowly. This appears demonstrates once multiple scattering angles are specified, increasing the number of scattering angles has only a weak effect on the speed-up factor. It should also be noted that except for the case of isotropic scattering, which is actually a different problem, the cases appear to be calculating the same k_{eff} within statistics.

Table 4.2

Comparison of the Modified Form of Cross Section Handling to the Original Form for Various Numbers of Angles

	Original		Modified		
Number of Angles	k_{eff} ($\pm\sigma$)	CPU Time (min)	k_{eff} ($\pm\sigma$)	CPU Time (min)	Speed-up Factor
0 Isotropic	1.2627 (0.0050)	2.1308	1.2730 (0.0055)	2.1467	1.00
1	1.0605 (0.0051)	3.0342	1.0543 (0.0049)	3.0203	1.00
2	1.0704 (0.0046)	3.0045	1.0691 (0.0050)	2.7655	1.09
3	1.0622 (0.0055)	2.9973	1.0665 (0.0050)	2.7487	1.09
4	1.0602 (0.0047)	3.0013	1.0641 (0.0053)	2.7277	1.10

35% Uranium (97% U-235), 65% H_2O SPHERE (20-cm diameter)
H_2O Reflector (40-cm Cube)
300 Neutrons per Generation, 103 Generations
27GROUPNDF4

Next, the effect of changing the number of particles per generation on the speed-up factor is examined for the problem described above. As shown in table 4.3, changing the number of histories per generation from 300 to 1200 changes the speed-up factor from 1.07 to 1.09. The small change in speed-up factor shows that the number of histories per generation has a minimal effect on the overall speed-up of the problem.

Table 4.3

Comparison of the Modified Form of Cross Section Handling to the Original Form for Various Numbers of Histories per Generation for a Simple Sphere

	Original		Modified		
Histories per Generation	k_{eff} ($\pm\sigma$)	CPU Time (min)	k_{eff} ($\pm\sigma$)	CPU Time (min)	Speed-up Factor
300	1.0719 (0.0048)	3.0003	1.0600 (0.0050)	2.7915	1.07
600	1.0599 (0.0035)	6.0088	1.0559 (0.0033)	5.5903	1.07
900	1.0643 (0.0031)	8.9855	1.0571 (0.0030)	8.3332	1.08
1200	1.0602 (0.0027)	12.0977	1.0636 (0.0027)	11.0978	1.09

35% Uranium (97% U-235), 65% H_2O SPHERE (20-cm diameter)
H_2O Reflector (40-cm Cube)
103 Generations
27GROUPNDF4

Finally, utilizing the 27GROUPNDF4 cross-section set, a problem consisting of an H_2O filled tank containing 216 fuel pins and an H_2O reflector is examined. Just like the previous problem, the system is well thermalized but now the fissile material is separated from the moderator. The number of histories per generation is varied from 300 to 1200 and the relative k_{eff}s, CPU times, and speed-up factors are shown in table 4.4.

Table 4.4

Comparison of the Modified Form of Cross Section Handling to the Original Form for Various Numbers of Histories per Generation for a Fuel Storage Tank

	Original		Modified		
Histories per Generation	k_{eff} ($\pm\sigma$)	CPU Time (min)	k_{eff} ($\pm\sigma$)	CPU Time (min)	Speed-up Factor
300	0.8615 (0.0047)	2.2867	0.8675 (0.0055)	2.2783	1.03
600	0.8569 (0.0030)	4.6308	0.8648 (0.0033)	4.4572	1.04
900	0.8690 (0.0027)	6.9308	0.8679 (0.0028)	6.6230	1.05
1200	0.8645 (0.0025)	9.2415	0.8698 (0.0024)	8.8233	1.05

TANK (18-cm ID, 60-cm Height, 0.5-cm thick)
216 Fuel Pins in a H_2O Moderator
PIN (0.75-cm OD, 30-cm Height, 20 w% U-235)
30-cm H_2O Reflector surrounding Tank
103 Generations
27GROUPNDF4 Latticecell

All cases appear to be calculating the same k_{eff} within statistics. The speed-up factors are lower than in the previous case, ranging from 1.03 to 1.05. This is probably due to a larger percentage of the CPU time being devoted to calculating crossings instead of collisions.

This method of storing and retrieving cross-section data has proven to be faster than the original method for two significantly different problems. The inclusion of this storage and retrieval method has shown a decrease in the CPU time from 5% to 9%. Although this method has only been examined for two problems, this method should provide similar

speed-ups for any problem.

4.2 Benchmarking

A set of 25 problems was used for initial benchmarking of the vectorized version of KENO-V.a against the original version of KENO-V.a. The 25 problems are those listed in the section F.11 of the SCALE Documentation.[17] The problem set include all geometry types and most options including holes, arrays, nested holes, and nested arrays in a variety of combinations. Several problems are actually the same problem described in a different manner such as cylinders described as two hemicylinders with a common face. A KENO-V.a input listing for this set of problems is contained in Appendix C. Table 4.5 contains a list of the computed k_{eff}s, standard deviations, CPU times, and speed-up factors for all 25 problems.

An examination of the k_{eff}s and standard deviations show that the two codes are calculating the same results within statistics (i.e. $\pm 2\sigma$). This indicates that the vectorized version of KENO-V.a is indeed working properly. Speed-up factors listed in Table 4.5 range from a low of 1.8 to a high of 4.8 with most falling in the range of 2.0 to 2.5. The low of 1.8 is due to the large number of options specified for this problem which increased the percentage of the problem run in the scalar mode. The high of 4.8 was obtained for a

TABLE 4.5

Comparison of the Vectorized Version of KENO-V.a with the Original Version for Several Different Problems

	Original		Vectorized		
Problem Number	k_{eff} ($\pm\sigma$)	CPU Time (min)	k_{eff} ($\pm\sigma$)	CPU Time (min)	Speed -up Factor
1	1.0116 (0.0042)	0.1746	1.0087 (0.0038)	0.0795	2.2
2	1.0116 (0.0042)	0.1896	1.0087 (0.0038)	0.0848	2.2
3	1.0064 (0.0054)	1.4928	1.0090 (0.0058)	0.6831	2.2
4	1.0078 (0.0047)	1.4895	1.0121 (0.0050)	0.6933	2.1
5	1.0174 (0.0039)	0.2666	1.0217 (0.0036)	0.1368	2.0
6	0.7508 (0.0039)	0.1171	0.7514 (0.0037)	0.0460	2.5
7	1.0035 (0.0044)	0.1813	1.0028 (0.0043)	0.0763	2.4
8	0.9476 (0.0036)	0.1526	0.9451 (0.0039)	0.0586	2.6
9	2.3073 (0.0062)	0.1826	2.3047 (0.0081)	0.0616	3.0
10	1.0116 (0.0042)	0.1860	1.0087 (0.0038)	0.0835	2.2
11	1.0186 (0.0043)	0.0975	1.0087 (0.0038)	0.0428	2.3
12	1.0095 (0.0050)	0.4280	1.0141 (0.0053)	0.1840	2.3
13	1.0020 (0.0048)	0.2115	0.9983 (0.0044)	0.0900	2.4
14	1.0061 (0.0048)	0.1601	1.0038 (0.0042)	0.0598	2.7
15	1.0159 (0.0043)	3.4605	1.0112 (0.0045)	1.7268	2.1

TABLE 4.5 cont.

Comparison of the Vectorized Version of KENO-V.a with the Original Version for Several Different Problems

	Original		Vectorized		
Problem Number	k_{eff} ($\pm\sigma$)	CPU Time (min)	k_{eff} ($\pm\sigma$)	CPU Time (min)	Speed -up Factor
16	0.9940 (0.0027)	0.6205	0.9913 (0.0026)	0.1703	3.6
17	1.0097 (0.0202)	0.8853	0.9908 (0.0191)	0.1843	4.8
18	1.0236 (0.0070)	1.3565	1.0189 (0.0046)	0.7751	1.8
19	1.0184 (0.0054)	0.4333	1.0058 (0.0048)	0.1978	2.2
20	1.0017 (0.0059)	0.4588	1.0001 (.0062)	0.1558	2.9
21	0.9985 (0.0033)	1.3208	0.9930 (0.0031)	0.4813	4.8
22	1.0116 (0.0042)	0.3056	1.0019 (0.0041)	0.1508	2.0
23	1.0116 (0.0042)	0.2205	1.0077 (0.0043)	0.1016	2.2
24	1.0020 (0.0042)	0.2165	1.0050 (0.0044)	0.0983	2.1
25	1.0050 (0.0040)	0.2175	1.0046 (0.0043)	0.1021	2.1

problem involving a bare, partially filled, solution sphere. This is a well thermalized problem with relatively few boundary crossings. Due to the well thermalized nature of the problem, a large percentage of particles remain in the system for an extended period of time thus resulting in long vector lengths for a larger percentage of the CPU time.

4.3 Timing Studies

To measure the effect of the number of histories per generation on the speed-up factor, three sample problems from table 4.5 where examined further. For each of these problems, the original and vectorized versions of KENO-V.a were run using 300, 600, 1200, and 2400 histories per generation. All unnecessary options were eliminated from the problems to maximize the speed-up associated with history tracking. The results of these runs are tabulated and plotted below.

The first problem is a 2x2x2 array of bare highly enriched uranium cylinders. This is a fast system from the standpoint of average neutron energy due to the absence of any moderating material. Most of the particles leak from the system after relatively few interactions in the cylinders resulting in short vector strings for a large portion of the total CPU time. As shown in Table 4.6, a maximum speed-up factor of 3.8 is achieved with 2400 histories per generation.

The second problem is a triangular pitched array of seven bare, highly enriched solution cylinders. This is a slower system from the standpoint of average neutron energy than the first case due to the presence of moderating material in the cylinders. The array is constructed by placing the cylinders in holes. Due to the presence of moderating material, histories have more interactions in the solution cylinders resulting in longer vector lengths for a larger portion of the total CPU time. As shown in Table 4.7

TABLE 4.6

Comparison of the Vectorized Version of KENO-V.a with the Original Version for a 2x2x2 array of cylinders. Same as Problem 2, Table 4.5 minus all options

	Original		Vectorized		
Histories per Generation	k_{eff} ($\pm\sigma$)	CPU Time (min)	k_{eff} ($\pm\sigma$)	CPU Time (min)	Speed-up Factor
300	1.0116 (0.0042)	0.1711	1.0087 (0.0038)	0.0715	2.4
600	1.0141 (0.0031)	0.3388	1.0020 (0.0030)	0.1150	2.9
1200	1.0019 (0.0022)	0.6891	1.0042 (0.0021)	0.2013	3.4
2400	1.0046 (0.0016)	1.4320	1.0053 (0.0016)	0.3786	3.8

TABLE 4.7

Comparison of the Vectorized Version of KENO-V.a with the Original Version for a triangular pitched array. Same as Problem 20, Table 4.5 minus all options

	Original		Vectorized		
Histories per Generation	k_{eff} ($\pm\sigma$)	CPU Time (min)	k_{eff} ($\pm\sigma$)	CPU Time (min)	Speed -up Factor
300	1.0017 (0.0059)	0.4443	1.0001 (0.0062)	0.1535	2.9
600	0.9992 (0.0039)	0.8861	1.0050 (0.0042)	0.2493	3.6
1200	1.0032 (0.0029)	1.7668	1.0075 (0.0027)	0.4546	3.9
2400	1.0047 (0.0019)	3.5578	1.0029 (0.0018)	0.8341	4.3

a maximum speed-up factor of 4.3 is achieved with 2400 histories per generation.

The third problem is a partially filled bare, low enriched solution sphere. This is a thermal problem from the standpoint of average neutron energy due to the presence of moderating material in a large vessel. Most of the particles remain in the system a long time with many interactions in the solution resulting in long vector strings for a large portion of the total CPU time. As shown in table 4.8, a maximum speed-up factor of 5.7 is achieved with 2400 histories per generation.

Since the problems considered above are limited in size and complexity, two additional problems, consisting of a

TABLE 4.8

Comparison of the Vectorized Version of KENO-V.a with the Original Version for a Bare Solution Hemisphere. Same as Problem 21, Table 4.5 minus all options

	Original		Vectorized		
Histories per Generation	k_{eff} ($\pm\sigma$)	CPU Time (min)	k_{eff} ($\pm\sigma$)	CPU Time (min)	Speed-up Factor
300	0.9985 (0.0033)	2.3256	0.9930 (0.0031)	0.4891	4.8
600	0.9917 (0.0024)	4.5690	0.9905 (0.0026)	0.8808	5.2
1200	0.9929 (0.0017)	9.1363	0.9974 (0.0015)	1.6516	5.5
2400	0.9943 (0.0013)	18.1458	0.9948 (0.0015)	3.1598	5.7

spent fuel storage cask and a spent fuel shipping cask, are examined below.

The spent fuel storage cask problem involves 21 spent fuel assemblies in a water filled stainless steel vessel. Intersticial support steel and boron are included in the problem description. The representation of this problem requires 6 material compositions, 12 distinct units, 77 holes, and one array. The array represents the fuel assembly. The fuel assemblies, boron, and steel are inserted into the cask as holes. The KENO-V.a input listing for this problem is contained in Appendix D.

This problem is thermal from the standpoint of average neutron energy due to the presence of moderating material in a large vessel. Most of the particles remain in the system a long time with many interactions in the solution resulting in long vector strings crossing inside the arrays but very short vector strings crossing in the cask between holes. A large percentage of the CPU time is spent calculating the crossing of a very few particles between holes. As shown in table 4.9, a maximum speed-up factor of 2.0 is achieved with 1200 histories per generation.

The relatively small speed-up in the spent fuel storage cask problem is primarily due to the large number of holes present which accounts for a very small percentage of total crossings. This problem could be posed in such a way to incorporate the majority of the arrays and intersticial

TABLE 4.9

Comparison of the Vectorized Version of KENO-V.a with the Original Version for a Spent Fuel Storage Cask Keeping the Number of Generations Constant

	Original		Vectorized		
Histories per Generation	k_{eff} ($\pm\sigma$)	CPU Time (min)	k_{eff} ($\pm\sigma$)	CPU Time (min)	Speed-up Factor
300	0.9191 (0.0043)	11.2388	0.9141 (0.0048)	9.0083	1.2
600	0.9227 (0.0028)	22.9933	0.9153 (0.0029)	14.2421	1.6
1200	0.9123 (0.0021)	46.2373	0.9209 (0.0022)	22.9488	2.0

material as another array thus reducing the number of holes. This would decrease the overall running time of both the scalar and vector versions of the code and increase the speed-up factor.

So far, speed-up factors have been increased by increasing the number of histories per generation thus increasing the number of histories processed by the problem. If the standard deviation of the original problem is acceptable, the total number of histories would not need to be increased and thus lower speed-up factors would be obtained. Instead of increasing the number of total histories in the problem, the total number of particles in the problem will now be held approximately constant by increasing the number of histories per generation and decreasing the number of generations. Table 4.10 shows the

results of changing the number of histories per generation from 300 to 2400 while keeping the total number of histories approximately constant. The k_{eff}s shown in table 4.10 appear to remain stable for 103 and 52 generations showing an increase in the speed-up factor from 1.2 to 1.7 with no increase in the total number of particles and no decrease in accuracy. Unfortunately, the cases involving 26 and 13 generations show significant variations in their k_{eff} values. This is due to the procedure used to calculate k_{eff} and its standard deviation in KENO-V.a. For each generation, a value of k_{eff} is calculated. After all generations have been processed, a system k_{eff} is calculated using all the previously calculated k_{eff}s and the standard deviation is also determined. Unfortunately, the particle history distribution

TABLE 4.10

Comparison of the Vectorized Version of KENO-V.a with the Original Version for a Spent Fuel Storage Cask Keeping the Total Number of Histories Constant

	Original		Vectorized		
Num. Gens/ His. per Gen	k_{eff} ($\pm\sigma$)	CPU Time (min)	k_{eff} ($\pm\sigma$)	CPU Time (min)	Speed-up Factor
103/ 300	0.9191 (0.0043)	11.2388	0.9141 (0.0048)	9.0083	1.2
52/ 600	0.9225 (0.0035)	11.9296	0.9157 (0.0045)	6.9833	1.7
26/ 1200	0.8999 (0.0044)	12.1373	0.9129 (0.0043)	5.7995	2.1
13/ 2400	0.9049 (0.0059)	12.4245	0.9190 (0.0012)	5.4590	2.3

is not known prior to the beginning of the problem so, unless otherwise specified, a flat fission distribution over all fissile material is assumed for the first generation. Several generations are usually required before a good representation of the true fission distribution is obtained. For the cases using 26 and 13 generations, the assumed initial fission distribution is having a disproportionate affect on the final result. It appears that reducing the number of generations to 52 is acceptable, but much below 52 the final results become unreliable.

Since the primary interest in this project is to reduce the CPU time without significantly changing the accuracy or precision (i.e. statistics) of the problem, the spent fuel shipping cask was examined by keeping the total number of histories approximately constant by altering the number of generations and histories per generation. The spent fuel shipping cask problem involves 31 spent fuel assemblies in a water filled stainless steel vessel. The spent fuel assemblies and intersticial material are all incorporated in arrays. There are seven distinct arrays composed of spent fuel assembly arrays. The seven arrays are then stacked in a global unit using 7 holes. This is a far more complex problem than the storage cask examined above. The KENO-V.a input listing of this problem is contained in Appendix E.

This problem is thermal from the standpoint of the average neutron energy due to the presence of moderating

material in a large vessel. Most of the particles remain in the system for a long time with many interactions in the solution resulting in long vector strings crossing inside the arrays but very short vector strings crossing in the cask between holes. A much smaller percentage of the CPU time is spent calculating the crossing of a very few particles between holes than is done for the spent fuel storage cask thus resulting in higher speed-up factors. As shown in table 4.11, the speed-up factor varies from 2.3 to 3.2 as the number of generations is reduced from 203 to 53.

The k_{eff}s shown in Table 4.11 of both the scalar and vector versions of KENO-V.a appear to differ more than would be expected from their standard deviations. This is primarily a result of more histories being needed to

TABLE 4.11

Comparison of the Vectorized Version of KENO-V.a with the Original Version for a Spent Fuel Shipping Cask Keeping the Total Number of Histories Constant

	Original		Vectorized		
Num. Gens/ His. per Gen	k_{eff} ($\pm\sigma$)	CPU Time (min)	k_{eff} ($\pm\sigma$)	CPU Time (min)	Speed-up Factor
203/ 1000	0.8116 (0.0011)	20.2526	0.8116 (0.0013)	8.9366	2.3
103/ 2000	0.8088 (0.0012)	20.3736	0.8091 (0.0014)	7.3373	2.8
53/ 4000	0.8063 (0.0014)	21.5891	0.8078 (0.0012)	6.7955	3.2

adequately represent this large and unusual system. The system contains a large number of regions and units with unusual mixtures of fissile materials and poisons. The fuel assemblies are modelled in such a way that the highest burn-up material is in the center of the fuel assemblies and the lowest burn-up material at the ends. This produces an unusual flux shape with the higher fluxes being at the ends of the fuel assemblies rather than in the middle as is the usual case.

As a possible aide to future research in the area of vectorized Monte Carlo, abbreviated output listings for the five problems discussed in this timing study are presented in Appendix F. Appendix E of the KENO-V.a manual[17] contains abbreviated output listings similar to those shown in Appendix F for different problems. Since all input parameters used in the vectorized version of KENO-V.a are identical to those used in the original version of KENO-V.a, the existing KENO-V.a manual is applicable to the vectorized version of KENO-V.a.

CHAPTER 5

CONCLUSIONS AND RECOMMENDATIONS

Significant reductions in the CPU time required by KENO-V.a are achievable by vectorizing the particle tracking section of the program. This required a complete rework of the entire particle tracking algorithm, the method used to store and process particle history data, and any subroutine that uses the particle history data. In total, 42 of the 175 subroutines which make up KENO-V.a were either modified or completely rewritten to implement the changes made in the two particle tracking subroutines TRACK and CROS.

The speed-up factors were obtained by comparing the vectorized version of KENO-V.a to the optimized scalar version of KENO-V.a. This was done to determine if the new method used to store and retrieve cross-section data was more efficient in the scalar mode. Speed-up factors were determined for various parameters, number of discrete angles specified, cross-section set, and number of histories per generation. Speed-up factors up 1.10 were obtained for a solution sphere and up to 1.05 for a water filled tank containing 216 fuel pins. Once multiple angles were specified, the number of histories per generation and the number of angles had relatively small effects on the speed-up factors. The only disadvantage in this technique is that it uses approximately 30% more storage for the cross-section

data than the original technique. However, since storage is plentiful on the current generation of mainframe computers, the additional required storage should not be a significant concern.

After the particle tracking algorithm in KENO-V.a had been vectorized and the program appeared to be functioning correctly, a diverse set of problems was run and a new set of speed-up factors were calculated. First, a set of 25 problems containing all geometries and most options in a diverse set of combinations was analyzed, the results are presented in table 4.5. Speed-up factors varying from 1.8 to 4.8 were shown for this group of problems. The wide range of speed-up factors is due to the differences in system geometry, the methods used to set up the problems, and the options used. The largest speed-up factor was obtained for a problem where a large percentage of the CPU time was spent in the free-flight and collision sections of TRACK, and a minimum of CPU time was spent calculating crossings. Conversely, the slowest problem, number 18, used a significant amount of time calculating crossings. This problem contains 4 arrays placed in a unit as holes. There is relatively little crossing between arrays with the majority of crossings occurring within each array. Unfortunately, this means a disproportionately large amount of CPU time was spent calculating crossings for the few particles moving from one array to another. Additionally,

when a particle undergoes a potential inward crossing in a region, all the holes in that region must be checked for a possible inward crossing. Since very few particles are crossing between arrays in this problem, very short vector lengths are used while calculating possible crossings between holes.

Holes in themselves do not significantly reduce the speed-up factor as long as large numbers of particles are simultaneously checked for possible crossings into the holes. Problem 20 is a triangular pitched array of 7 solution cylinders constructed using 7 holes. Since a large percentage of the particles leak from each rod, the vector lengths used when processing possible inward crossings into the holes are long enough to provide significant speed-ups. The speed-up factor of 2.9 lies between the speed-up factor of 4.8 for the solution sphere case and the speed-up factor of 1.8 for problem 20.

Next, three of the 25 problems were examined in greater detail to determine the effect of changing the number of particles per generation had on the speed-up factors. For the original scalar version of KENO-V.a, while keeping the number of generations constant, doubling the number of particles per generation simply doubled the CPU time. However, in the vectorized version of KENO-V.a, doubling the number of particles per generation increased the CPU time by a factor less than 2.0, thus increasing the speed-up factor.

This is due to the increase in the average length of the vectors being processed, which increases the efficiency of the vector processor.

Speed-up factors for a problem consisting of a 2x2x2 array, a problem consisting of multiple holes, and a problem consisting of a large solution sphere were calculated for various numbers of particles per generation. All three cases shown increase the speed-up factors as the number of particles per generation is increased. This increase in the speed-up factor should eventually level off when the time spent processing the last few particles remaining in the system becomes insignificant. Since the CRAY-XMP has a maximum vector processor length of 64, several thousand particles per generation should be sufficient to reach this plateau.

Finally, two real problems consisting of a spent fuel storage cask and a spent fuel shipping cask were examined to show that significant speed-up factors are attainable for the type of problems KENO-V.a solves routinely. For the spent fuel shipping cask, a speed-up factor of only 1.2 was obtained for the original problem with no modifications in the number of histories per generation. This low speed-up factor is attributed to the same factors that affected problem number 18. The storage cask consists of 21 fuel assemblies, the intersticial support structure, and poison assembled using 77 holes resulting a large percentage of the

CPU time being spent calculating the crossings of very few particles between holes. This problem was partially alleviated by increasing the number of histories per generation. A speed-up factor of 2.0 was obtained using 1200 histories per generation.

Keeping the number of histories constant and varying the number of generations and histories per generation is a way to increase the speed-up factor without altering the statistics. An increase in the speed-up factor for the storage cask problem from 1.2 to 1.7 was achieved by reducing the number of generations from 103 to 52 with no significant effects on the accuracy of the results or statistics. For fewer than 52 generation the results become unreliable due primarily to contamination by the initial assumed flux distribution.

The shipping cask problem, a far more complex problem, resulted in much better speed-up factors than the storage cask problem. The primary reason for the larger speed-up factors is the reduction in the total number of holes used to contain the spent fuel assembly arrays. This problem also shows that the speed-up factor can be increased without a significant change in statistics by reducing the number of generations and increasing the number of histories per generation while keeping the total number of histories approximately constant. A maximum speed-up factor for this problem of 3.2 was obtained when the number of generations

was reduced to 53. The minimum number of generations examined for this problem is 53 due to the unreliability of the final results when using fewer generations.

The speed-up factors presented here represent the true speed-up as obtained between a program optimized to utilize a vector processor and one optimized to utilize a scalar processor. A speed-up factor of 5.7 on the CRAY-XMP theoretically requires approximately 90% of the utilized code to be optimally vectorized. The variation in vector length as a generation is processed means the maximum theoretical speed-up factor will never be achieved.

All speed-up factors in previous studies were calculated by comparing a program run in the vector mode against either the vectorized code run in scalar mode or an equivalent scalar program. Vectorized Monte Carlo codes run in scalar mode use significantly more CPU time than an optimized scalar version of the code due to the need in the vectorized version of the code to continuously shuffle data between stacks. Also, since the CPU times of Monte Carlo codes vary dramatically with respect to the tracking algorithm, options, data handling and other factors, no two independently created Monte Carlo codes are truly equivalent. Although previous speed-up factors gave a fair indication of the potential gains associated with vectorizing a Monte Carlo code, this work, which compares optimized vectorized and scalar versions of the same code, shows that vectorization can produce

substantial gains.

More work is needed before the vectorized version of KENO-V.a is ready for public use. Realistic problems of all types need to be examined to ensure that the code functions properly for all combinations of geometries and options. Also, more timing studies are required for various classes of problems involving holes, arrays, albedos, reflectors, and various combinations. Finally, the code must be made available to others for testing. Only after being rigorously examined by experienced users will a vectorized version of KENO-V.a be made available to the public.

Although the vectorized version of KENO-V.a is optimized for the CRAY-XMP, it should be readily portable to any computer possessing a vector processor such as an IBM-3090. The IBM-3090 at Oak Ridge National Laboratory does not currently contain a vector processor, but there are plans to upgrade the machine which include adding one or more vector processors. Since the CRAY-XMP will no longer be available as of September 30, 1992, if the IBM-3090 is upgraded, the vectorized version of KENO-V.a would be an excellent addition to the package of computer codes currently on the IBM-3090.

REFERENCES

REFERENCES

1. J. Spanier and E. Gelbard, "Monte Carlo Principles and Neutron Transport Problems", Addison-Wesley Publishing Co., Reading MA (1969)

2. P. Stevens, "NE-582 Monte Carlo Methods Class Notes", University of Tennessee (1989)

3. L.L. Carter and E.D. Cashwell,"Particle Transport Simulation with the Monte Carlo Method", Los Alamos Scientific Laboratory, TID-26607 (1975)

4. F. Brown, "Present Status of Vectorized Monte Carlo", Tran. Am. Nucl. Soc. (55), 323 (1987)

5. M. Nakagawa, T. Mori and M. Sasaki, "Monte Carlo Calculations on Vector Supercomputers using GMVP", Prog. Nuc. Eng. 24,(1-3), 183 (1990)

6. K. Asai, K. Higuchi, J. Katakura and Y. Kurita, "Vectorization of the KENO-IV Code," Nucl. Sci. Eng.92, 298 (1986).

7. E. Troubetzkoy, H. Steinberg and M. Kalos, "Monte Carlo Radiation Penetration Calculations on a Parallel Computer", Tran. Am. Nucl. Soc. (17), 260 (1973)

8. F. Brown, W. Martin, and D. Calahan, "A Discrete Sampling Method for Vectorized Monte Carlo Calculations", Tran. Am. Nucl. Soc. (38), 354 (1981)

9. F. Brown, " Vectorized Monte Carlo," Dissertation, University of Michigan, Department of Nuclear Engineering (1981)

10. A. J. Walker, "An Efficient Method for Generating Discrete Random Variables with General Distributions", ACM Trans. on Mathematical Software, Vol. 3, NO. 3, 253 (September 1977)

11. W. Martin and F. Brown, "Status of Vectorized Monte Carlo for Particle Transport Analysis", Int. J. of Supercomputer Appl. Vol. 1, Num. 2, 11 (1987)

12. F. Brown, "Vectorization of 3-D General-Geometry Monte Carlo", Tran. Am. Nucl. Soc. (60), 336 (1989)

13. F. Brown and F. Bischoff, "Computational Geometry for Reactor Applications", Tran. Am. Nucl. Soc. (57), 122 (1988)

14. W. Martin and F. Brown, "Monte Carlo Methods for Particle Transport", Tran. Am. Nucl. Soc. (60), 336 (1989)

15. W. L. Thompson, et. al., "MCNP-A General Monte Carlo Code for Neutron and Photon Transport," LA-7396-M, Los Alamos National Laboratory (1979)

16. M. B. Emmett, "MORSE-CGA, A Monte Carlo Radiation Transport Code with Array Geometry Capability," ORNL-6174, Oak Ridge National Laboratory (April 1985)

17. L. M. Petrie and N. F. Landers, "KENO-V.a: An Improved Monte Carlo Criticality Program with Supergrouping," NUREG/CR-0200, ORNL/NUREG/CSD-2, Oak Ridge National Laboratory (1985)

18. K. Asai, K. Higuchi, J. Katakura and Y. Kurita, "Vectorization of KENO-IV Code and an Estimate of Vector-Parallel Processing," JAERI-M, 86-151, Japan Atomic Energy Research Institute (October 1986)

19. M. Nakagawa, T. Mori and M. Sasaki, "Comparison of Vectorization Methods Used in a Monte Carlo Code", Nucl. Sci. Eng. (107), 58 (1991)

20. M. Nakagawa, T. Mori and M. Sasaki, "Development of Monte Carlo Code for Particle Transport Calculations of Vector Processors", Proc. Supercom. Nuc. Appl., p.160 (1990)

21. M. Nakagawa, T. Mori and M. Sasaki, "MVP, A Continuous Energy Monte Carlo Code for Vector Supercomputers" Proc. Inter. Topical Meeting Advances in Math., Comp., Reac. Phy. vol.5, (30.4) 4-1 (1991)

22. M. Suganuma et. al., "Vectorization of the Continuous Energy Monte Carlo Code VIM", JAERI-M (86), 196 (1987)

23. M. Flynn, "Some Computer Organizations and their Effectiveness", IEEE Trans. on Computers, C-21, No. 9, 948 (1972)

24. G. Amdahl, "Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities", Proc. Spring Joint Conf. AFLPS, (30) 483 (1967)

25. "Fortran Programming on the Cray Computers, Student Workbook", Pacific-Sierra Research Corp., Los Angeles, CA (1986)

26. "UNICOS User Commands Reference Manual", SR-2011.50, Cray Research Inc. (1989)

27. "CFT77 Reference Manual", SR-0018C, Cray Research Inc. (1988)

28. D. F. Hollenbach, K. H. Reynolds, H. L. Dodds, N. F. Landers and L. M. Petrie, "An Improved Method for Storing and Retrieving Tabulated Data in a Scalar Monte Carlo Code", Trans. Am. Nucl. Soc. (61), 191 (1990)

29. D. F. Hollenbach, W. Newmeyer, B. Basoglu, H. L. Dodds, N. F. Landers, and L. M. Petrie "Vectorization of the KENO-V.a Monte Carlo Criticality Code", Trans. Am. Nuc. Soc.(64), 348 (1991)

30. D. F. Hollenbach, H. L. Dodds, and L. M. Petrie, "Improvements in the Vectorization of the KENO-V.a Criticality Safety Code", Trans. Am. Nuc. Soc. (65), 241 (1992)

APPENDIXES

Appendix A

Listing of the Vectorized Version of the Subroutine TRACK

```
      subroutine track(neut,mixc,impc,mat,imp,igeom,
     * xx,kbnds1,kbnds2,lba,deltx,delty,deltz,fisden,
     * wt,tme,u,v,w,x,y,z,x1,y1,z1,etausd,nunum,kcol,
     * kcore,nbx,nby,nbz,k,ig,nhole,narrin,nholin,
     * nsmult,kft,lls,nhl,nar,stacka,stackh,rfbank,
     * ifbank,fwrr,lim,tp,tu,th,ta,ipap,vinv,mwa,sigt,
     * fnap,fnfp,fap,chi,twodee,mal,a,alb,wtavg,flux,
     * fleak,fmabs,fmfis,kboxc,plim,cpol,spol,mwxsc,
     * mwalb,probx,proba,iabsg,iregc,x1d,ndxlba,irega,
     * indxx,indxy,indxz,nbxmx,nbymx,nbzmx,holx,holy,
     * holz,lholu,khole,jgeom,ifh,ilh,xtra,nsct1,lensg,
     * lreg,nscnt,nfiss,lgtyp)
c.....................................................
      common /albnam/rnames(6)
      character*8 rnames
      common /albdat/ intr(6),idalb(6),nbxl(6),iface,
     * nnsig,nalb,nang,ng,awt,lprbx,lprba,nagsg,nag1
c.....................................................
c.....................................................
      common /dimen/tmax,tbtch,dwtav,wthigh,wtlow,big,
     1     nba, npb, nskip, nbas, nrstrt, numx1d,
     2    nbank,  nxnbk,  nfbnk,  nxfbk,
     3    lbank,  lfbnk, lnubnk, lnextr, lnfbnk,
     4      ngp,   ngp1, ntypst, nmat, matt, nmix,
     5  kmax, krefm, nbox, nboxt, ngblu, nucom,
     6  nbxmax, nbymax, nbzmax, nglobl, matdim,  nacom,
     7  maxmix, maximp, mang, nimp, numids, ncs,ntsets,
     8     nsg, nchigp, nsct, mix, mixt,  npl, pbxs,
     9  maxara, lbalng,  ndelx,  ndely,  ndelz,
     a  numara, nalvls, lnstka,  lstka, lfstka,  iaa,
     b  numhol, nhlvls, lnstkh,  lstkh, lfstkh,  ihh,
     c  mult
c
c  tmax   is time allowed for the problem
c  tbtch  is time allowed for a generation
c  dwtavg is the default value of avg. wt.
c  wthigh is the wt. at which splitting occurs
c         (wthigh*wtavg)
c  wtlow  is the russian roulette threshold(wtlow*wtavg)
c  big    is the maximum path in a void
c  nba    is number of generations
c  npb    is number per generation
c  nskip  is number of generations skipped
c  nbas   is the starting generation
c  nrstrt is generations between restarts
c  numx1d is no. of extra 1-d xsecs
c  nbank  is no. of positions in neutron bank
c  nxnbk  is extra positions per neutron in neutron bank
```

```
c  nfbnk  is no. of positions in fission bank
c  nxfbk  is extra positions per neutron in fission bank
c  lbank  is total positions per neutron in the integer
c         neutron bank
c  lfbnk  is total positions per neutron in the integer
c         fission bank
c  lnubnk is the maximum length of the neutron bank
c  lnextr is length of extra data from subroutine extra
c  lnfbnk is the maximum length of the fission bank
c  ngp    is number of energy groups
c  ngp1   is no. of energy groups + 1
c  ntypst is start type
c  nmat   is number of mixtures on ice tape
c  matt   is no. of mixtures in the problem
c  nmix   is no. of mixing table entries
c  kmax   is number of geometry regions used
c  krefm  is no. of geometry cards read
c  nbox   is the largest unit number specified in the
c         input data
c  nboxt  is the largest unit number in the problem
c  ngblu  is the global unit number
c  nucom  is the number of units having comments in
c         the geometry
c  nbxmax is number of units in the x dir.
c  nbymax is number of units in the y dir
c  nbzmax is number of units in the z direction
c  nglobl is the global array number
c  maxmix is largest mixture no. in geometry
c  maximp is largest biasing region no. in geometry
c  mang   is no. of angles on xsec tape
c  nimp   is no. of biasing regions
c  numids is the number of bias id's requested
c  ncs    is the number of sets of weights read from
c         cards
c  ntsets is the total number of weight group
c         structures read from cards
c  matdim is nbgbl*nbybgl*nbzgbl + 1
c  nacom  is the number of arrays with comments in the
c         array data
c  nsg    is the number of supergroups
c  nchigp is the no. of groups the fission spectrum
c        varies with
c  nsct   is the number of scattering angles
c  mix    is the no. of different mixtures to be mixed
c  mixt   is the maximum mixture no. to be mixed
c  npl    is the order of legendre coefficients + 1
c  pbxs   is the threshold for printing messages for
c         errors found in processing the angles and
c         probabilities
c  maxara is the maximum array number
c  lbalng is the length of the composite lba array
c  ndelx  is the length of the composite delx array
```

```
c  ndely  is the length of the composite dely array
c  ndelz  is the length of the composite delz array
c  numara is the number of arrays
c  nalvls is the max. depth of nesting for arrays
c  lnstka is the length of stacka (array stack)
c  lstka  is the pointer to stacka within the neutron bank
c  lfstka is the pointer to stacka within the fission bank
c  iaa    is the location in the fission bank of the
c         pointer to last level in the stacka array
c         extra(ias)
c  numhol is the number of holes
c  nhlvls is the max. depth of nesting for holes
c  lnstkh is the length of stackh (hole stack)
c  lstkh  is the pointer to stackh within the neutron bank
c  lfstkh is the pointer to stackh within the fission bank
c  ihh    is the location in the fission bank of the
c         pointer to the last level in the stackh array
c         extra(ihs)
c  mult   is the variable setting the number of storage
c         locations used by a variable typed double
c         precision
c.....................................................
c.....................................................
      common/final/dtime,diin,diii,igen,iostrt
c.....................................................
c.....................................................
      common /holary/ iholes, iarrin, iholin, ismult,
     *                ikapos, ikunit, ikhole, ikarry
c     iholes is the length of nhole
c     iarrin is the length of narrin
c     iholin is the length of nholin
c     ismult is the length of nsmult
c     ikapos is the length of kft
c     ikunit is the length of lls
c     ikhole is the length of nhl
c     ikarry is the length of nar
c.....................................................
c.....................................................
      common /lifetm/ sorwt,tofis,fleakt,sleak,sqleak,
     * fmabst,smabs,
     1 sqabs,fmfist,sfmfis,sqfis,tfisn,gfis,nubar,nusqr,
     2 timg,tmgs,timl,tmls,sqkef,skbar,akbar,rakbar,age,
     3 tlf,gbar,gsqr,self,slfsq,tself,awtsum,awtsq,lvel,
     4 lprnt,indxfs
      double precision sorwt,tofis,fleakt,sleak,sqleak,
     1 fmabst,smabs,sqabs,fmfist,sfmfis,sqfis,tfisn,
     2 gfis,nubar,nusqr,timg,tmgs,timl,tmls,sqkef,
     3 skbar,akbar,rakbar,age,tlf,gbar,gsqr,self,slfsq,
     4 tself,awtsum,awtsq
c.....gbar   = sum of the average energy group where
c              fission occurs
c.....gsqr   = sum of the squares of the group where
```

```
c                fission occurs
c.....nubar   = sum of the average value of nu
c.....nusqr   = sum of the squares of nu
c.....self    = avg. self multiplication of the units
c.....slfsq   = sum of the squares of the self
c                multiplication
c.....indxfs = position of the fission xsec in the extra
c                1-d xsec array
c.....tfisn   = avg. fission cross section
c.....gfis    = fission weighted energy group
c.....................................................
c.....................................................
      common /logic/lrun,lplot,nflx,nfden,lmult,newbar,
     1 lunit,   lcku, pmunit, larpos,   lckp, pmapos,
     2 lmhole,   lckh, pmhole,  lhhgh,
     3 lmarry,   lcka, pmarry,  lahgh,
     4 lpaxs,   prt1,  prtp0,  prtap, prtchi,  prtex,
     5 lfa, lcorsp, lpwt,  lgeom, ldbug, ltrk, nadj,
     6 lstgen, lxtra, lsgun, lsav, lstop, mflag, lreed,
     7 lnxx,  lsavf, lrnd, lsavu, lsavp, lsavh, lsava,
     8 mbox,  exrfl,  nesta,  nesth, ppmsg
       logical nflx,nfden,nadj,lpaxs,prtap,prt1,prtp0
       logical lfa,lunit,larpos,lmhole,lmarry,lsgun,lsav
       logical pmapos,pmunit,pmhole,pmarry
       logical lstop,mflag,lreed
       logical prtchi, prtex
       logical ldbug,ltrk
       logical lnxx,lahgh,lhhgh
       logical lsavf,lrnd,lsavu,lsavp,lsavh,lsava,mbox,
       logical lpwt,lgeom,exrfl
       logical lmult,newbar,nesta,nesth,lrun,lplot,ppmsg
       logical lcorsp,lcku,lckp,lckh,lcka,lxtra(2),lstgen
c.....lrun is the execution flag.  a value of false
c          suppresses execution.  default is true.
c.....nadj is the adjoint flag.  default is false.
c.....nflx is the flag for collecting and printing
c          fluxes.  default is false.
c.....nfden is the flag for collecting and printing
c           fission densities.  default is false.
c.....lfa is the flag for printing fissions and
c         absorptions by region.  default is false.
c.....lmult is true if the average self multiplication
c           of a unit is to be calculated.
c.....newbar is true if nu-bar and the average fission
c            group are to be calculated
c.....lunit is the logical key for collecting matrix
c           information by unit type.
c.....lcku is the logical flag for calculating cofactor
c          k-effective by unit type
c.....pmunit is the flag for printing the fission
c            probability matrix by unit type.
c.....larpos is the logical key for collecting matrix
```

```
c            information by array position  (location
c            within the array).
c.....lckp is the logical flag for calculating cofactor
c          k-effective by unit location or position in
c          the array.
c.....pmapos is the flag for printing the fission
c            probability matrix by unit location or
c            position in the array.
c.....lmhole is the logical key for collecting matrix
c            information by hole.
c.....lckh is the logical flag for calculating cofactor
c          k-effective by hole.
c.....pmhole is the flag for printing the fission
c            probability matrix by hole.
c.....lmarry is the logical key for collecting matrix
c            information by array.
c.....lcka is the logical flag for calculating cofactor
c          k-effective by array.
c.....pmarry is the flag for printing the fission
c            probability matrix by array.
c.....lpaxs is the print flag to turn on all mixture
c           cross section print options.  if any options
c           are entered in addition to lpaxs, they
c           override lpaxs
c.....prt1 is the print flag for the 1-d cross sections
c.....prtp0 is the print flag for the 2-d cross sections
c           (p0)
c.....prtap is the print flag for the angles and
c           probabilities
c.....prtchi is the print flag for chi, the fission
c            spectrum
c.....prtex is the print flag for the extra 1-d cross
c           sections
c.....lcorsp is true for printing xsec-albedo
c            correspondence arrays
c.....lpwt is logical key for printing weight average
c.....ldbug keys input debug print
c.....ltrk keys debug tracking print
c.....lgeom is print flag for input geometry data in
c           kenog & rdbox
c.....lplot is the logical flag indicating whether a
c           picture plot is to be generated.  default
c           value is false.
c.....lahgh is true if matrix information by array is
c           collected at the highest level and false if
c           at the lowest level.
c.....lhhgh is true if matrix information by hole is
c           collected at the highest level and false if
c           at the lowest level.
c.....lxtra holds space for logical input parameters
c.....lstgen is set true upon completion of the last
c            generation that is run.  it is used for
```

```
c            restart in the search routines.
c.....lsgun is set true for a single unit problem.
c.....lsav is false if fissions and absorptions by
c          region were written on restart and rstrt=
c          wstrt and the restarted problem says to
c          suppress fissions and absorptions by region.
c          a warning message is written, fissions and
c          absorptions by region are calculated but not
c          printed
c.....lstop is used to bypass data reading in datain if
c           an end data was encountered before going to
c           datain.
c.....mflag is set true when an error is found, so
c           execution can be terminated when data
c           processing is completed.
c.....lreed is true if a read flag was read when the
c           end param flag was expected.
c.....lnxx is true if an albedo option is to be exercised
c.....lsavf is false if fluxes were written on restart
c           and rstrt=wstrt and the restarted problem
c           says to suppress the fluxes.  a warning
c           message is written and the fluxes are
c           calculated but not printed.
c.....lrnd is true if the restarted problem is to
c          continue with the original random sequence.
c          to change random sequence, enter a random
c          number as a parameter
c.....lsavu is false if matrix information by unit type
c           was written on the restart unit and rstrt=
c           wstrt and the restarted problem specifies
c           suppressing it.  a warning message is printed
c           and the data is calculated but not printed.
c.....lsavp is false if matrix information by unit
c           position was written on the restart unit and
c           rstrt=wstrt and the restarted problem
c           specifies suppressing it.  a warning message
c           is printed and the data is calculated but
c           not printed.
c.....lsavh is false if matrix information by hole was
c           written on the restart unit and rstrt=wstrt
c           and the restarted problem specifies
c           suppressing it.  a warning message is printed
c           and the data is calculated but not printed.
c.....lsava is false if matrix information by array was
c           written on the restart unit and rstrt=wstrt
c           and the restarted problem specifies
c           suppressing it.  a warning message is printed
c           and the data is calculated but not printed.
c.....mbox is true if multiple units are specified
c.....exrfl is true if a global reflector is present
c.....nesta is logical flag for nested arrays.
c           default=false
```

```
c.....nesth is logical flag for nested holes.
c           default=false
c.....ppmsg is the flag for printing posit error messages.
c     ppmsg is false while drawing a picture and true
c           while tracking.
c.....................................................
c.....................................................
      common /lowbnd/ 101,102,103,104,105,106,107,108,
     * 109,110,111,112,113,114,115,116,117,118,119,120
c     101 is the lower bound for numhol and nhole
c     102 is the lower bound for nalb
c     103 is the lower bound for nang
c     104 is the lower bound for lnextr
c     105 is the lower bound for lbalng
c     106 is the lower bound for numara and maxara
c     107 is the lower bound for nalvls and narrin
c     108 is the lower bound for nhlvls and nholin
c     109 is the lower bound for numx1d
c     110 is the lower bound for nsct1
c     111 is the lower bound for nsct
c     112 is the lower bound for ndelx,ndely,ndelz
c     113 is the lower bound for lprbx
c     114 is the lower bound for lprba
c     115 is the lower bound for nsmult
c     116 is the lower bound for kft
c     117 is the lower bound for lls
c     118 is the lower bound for nhl
c     119 is the lower bound for nar
c.....................................................
c.....................................................
      common /matrx/ nbxgbl,nbygbl,nbzgbl,kglobl,kgore,
     * ngx,ngy,ngz,ng01
c     nbxgbl,nbygbl,nbzgbl are nbxmax,nbymax,nbzmax in
c     the global array
c     kglobl is kcore of the global array
c     ngx,ngy,ngz define the current position in the
c     global array
c     kgore is kcore of the first array level at that
c     position
c     ng01 is the lower index on the global lba
c.....................................................
c.....................................................
      common /titl/ title(20),rdwrd
      character*4 title,rdwrd
c      title is the problem title
c      rdwrd is the block identifier specified on the
c      first data block
c.....................................................
c.....................................................
      common /unit/ inpt, outpt, icexs, albdo, wts,
     1 skrt, rstrt, wstrt, ampxs, direct(3), i0, i1,
     2 i2, i3, i4, nspare(4)
```

```
      integer outpt,albdo,skrt,rstrt,wstrt,direct,wts,
     * icexs,ampxs
c.....................................................
c******rfbank contains x,y,z**************************
c******ifbank contains nbx,nby,nbz,k,energy gp (ig) ***
      dimension mwa(3,lensg,matt), twodee(ipap,3)
      dimension wt(nbank),tme(nbank),u(nbank),v(nbank),
     * w(nbank),x(nbank),y(nbank),z(nbank),x1(nbank),
     * y1(nbank),z1(nbank),etausd(nbank),nunum(nbank),
     * kcol(nbank),kcore(nbank),nbx(nbank),nby(nbank),
     * nbz(nbank),k(nbank),ig(nbank),nhole(101:iholes),
     * narrin(107:iarrin),lhole(101:iholes),
     * lsthol(101:iholes),lhic(101:iholes),
     * nholin(108:iholin),nsmult(115:ismult),
     * kft(116:ikapos),lls(117:ikunit),nhl(118:ikhole),
     * nar(119:ikarry),stacka(nbank,107:lnstka),
     * stackh(nbank,108:lnstkh)
      integer   stacka, stackh
      dimension xx(7,krefm),fap(lensg,matt),mat(krefm),
     * kboxc(krefm),fwrr(nfbnk),rfbank(nfbnk,3),
     * ifbank(nfbnk,lfbnk),chi(ngp,nchigp,matt),
     * fnap(lensg,matt),fnfp(lensg,matt)
      dimension fmabs(lensg,lreg),fmfis(lensg,lreg),
     * fleak(lensg),igeom(krefm),kbnds1(nboxt),
     * kbnds2(nboxt),lim(nsg,3),wtavg(lensg,nimp),
     * sigt(lensg,matt),vinv(lensg),deltx(112:ndelx),
     * delty(112:ndely),deltz(112:ndelz),mixc(maxmix),
     * impc(maximp),imp(krefm),flux(lensg,kmax),
     * fisden(kmax,3)
      dimension tp(matdim,matdim),tu(nboxt,nboxt),
     * th(101:numhol,101:numhol),
     * ta(106:numara,106:numara),
     * mal(3,103:nang,lensg),alb(102:iabsg,102:nalb),
     * a(103:nang,102:iabsg,102:nalb)
      dimension plim(103:nang,102:nalb),
     * cpol(103:nang,102:nalb)
      dimension spol(103:nang,102:nalb)
      dimension mwxsc(ngp,3),mwalb(102:ng,3)
      dimension probx(113:lprbx),proba(114:lprba)
      dimension iregc(krefm),xtra(104:lnextr)
      dimension x1d(lensg,matt,109:numx1d)
      dimension ndx1ba(106:numara),irega(106:maxara),
     * indxx(106:numara),indxy(106:numara),
     * indxz(106:numara),nbxmx(106:maxara),
     * nbymx(106:maxara),nbzmx(106:maxara)
c     ndx1ba holds the starting location of each 1ba array
c     irega is the array correspondence array
c     indxx,indxy,indxz are offsets into deltx,delty,deltz
c     nbxmx,nbymx,nbzmx contain nbxmax,nbymax,nbzmax for
c     each array
      dimension sold(3),snew(3),s1old(3)
      dimension  holx(101:numhol), holy(101:numhol),
```

```
     *           holz(101:numhol)
      dimension khole(101:numhol),lholu(101:numhol)
      dimension ifh(krefm),ilh(krefm)
c     sold is the beginning point of the track
c     snew is the crossing point
c     slold is the end point of the track
c     holx, holy, holz are the x,y&z offset of the hole
c     lholu is the unit within the hole
c     khole is the region that contains the hole
c     ifh and ilh are the first and last holes in a region
      logical logper,lalb,lhic,matrix
      logical larray,lic,ltar1,lsplit,lrghol,lhole,
     *        lnmara,lnmhol
c     lrghol logical flag is true if the region contains
c            holes
c     lhole  logical flag is true if you are in a hole
c     lnmhol logical flag is true if the problem
c            utilizes holes
c     lnmara logical flag is true if the problem
c            utilizes arrays
c     matrix logical flag is true if collecting matix
c            information
      integer lba(105:lbalng)
      double precision fleak,fmabs,fmfis
c** the new arrays for the vector version of track*****
      dimension ncrosi(nbank),ncroso(nbank),nff(nbank),
     * ncoll(nbank),nkill(nbank),itar1(nfbnk),
     * itar2(nfbnk),itar3(nfbnk),itar4(nbank),
     * itar5(nbank),itar6(nbank),itar7(nbank),
     * itar8(nbank),itar9(nbank),itar10(nbank),
     * itar11(nbank),itar12(nbank),itar13(nbank),
     * ltar1(nbank)
      dimension rtar1(nbank),rtar2(nbank),rtar3(nbank),
     * rtar4(nbank),rtar5(nbank),rtar6(nbank),
     * rtar7(nbank),rtar8(nbank),rtar9(nbank),
     * rtar10(nbank),rtar11(nbank),rtar12(nbank),
     * rtar13(nbank),rtar14(nbank),rtar15(nbank),
     * rtar16(nbank),rtar17(nbank),rtar18(nbank),
     * rtar19(nbank),rtar20(nbank),rtar21(nbank),
     * rtar22(nbank),rtar23(nbank),rtar24(nbank)
      dimension ll(nbank),kr(nbank),ki(nbank),
     * igeo(nbank),now(nbank),k1(nbank),k2(nbank),
     * rpth(nbank),pth(nbank),jgeom(10),ngeom(2,10),
     * lic(nbank),m(nbank),fisw(nbank),r(nfbnk),
     * fwr(nbank),logper(nbank),larray(nbank),
     * etamin(nbank)
c
      parameter (twopi=6.28318530717958647692)
      logical lxhcyy,lxhcyz,lyhcyx,lyhcyz,lzhcyx,lzhcyy
      logical lhsphx,lhsphy,lhsphz,lmir,lper,ldif
      common /lhemi/ lxhcyy, lxhcyz, lyhcyx, lyhcyz,
     *       lzhcyx, lzhcyy, lhsphx, lhsphy, lhsphz
```

```
c
c********************** start fstart ******************
      if(ldbug) call ldwrt(kmax,kmax1,lim(1,3),neut,
     *  nsct,nsct1,lensg,matt,mwa,fap,fnap,fnfp,sigt)
       lnmara = numara.gt.0
       lnmhol = numhol.gt.0
       isc    = 0
       lself  = 0
       lcoll  = 0
       lcrsi  = 0
       lcrso  = 0
       lkil   = nbank - neut
       lff    = neut
       lmir = .false.
       lper = .false.
       ldif = .false.
       do 100 neut = 1, lff
         nff(neut)  = nunum(neut)
         ll(neut)   = kboxc(k(neut))
         kr(neut)   = mat(k(neut))
         ki(neut)   = imp(k(neut))
         nkill(neut) = 0
         rpth(neut) = 0.0
         larray(neut) = .not.nbx(neut).eq.0
 100   continue
       do 101 neut = 1, lff
       k1(neut)     = kbnds1(ll(neut))
       k2(neut)     = kbnds2(ll(neut))
       lic(neut)   = .true.
       ki(neut)     = impc(ki(neut))
       if(kr(neut) .gt. 0) kr(neut) = mixc(kr(neut))
 101   continue
       do 102 neut = 1, lff
         if(k(neut).le.k1(neut) .and. igeom(k(neut)).gt.0
     *     .and. ilh(k(neut)).le.0) lic(neut) = .false.
  102  continue
       do 103 neut = 1, 6
         if(intr(neut).eq.1) lmir = .true.
         if(intr(neut).eq.2) lper = .true.
         if(intr(neut).eq.3) ldif = .true.
  103  continue
       do 104 neut = 1, lff
         if(kcore(neut).gt.0) then
           now(neut) = mat(kcore(neut))
         else
           now(neut) = nglobl
         endif
  104  continue
       do 105 neut = 1, lkil
         nkill(neut) = nbank + 1 - neut
  105  continue
       if(lnmara) then
```

```
        maxnbx = nbxmx(1)
        maxnby = nbymx(1)
        maxnbz = nbzmx(1)
      do 106 neut = 2, maxara
        maxnbx = max(maxnbx,nbxmx(neut))
        maxnby = max(maxnby,nbymx(neut))
        maxnbz = max(maxnbz,nbzmx(neut))
  106 continue
      if(nesta) then
        do 107 neut = 1, lff
          if(.not.larray(neut)) narrin(neut) = -3
  107   continue
      endif
      endif
      if(lnmhol) then
        do 108 neut = 1, lff
          lhole(neut) = nhole(neut).gt.0
          lhic(neut) = .false.
  108   continue
        do 109 neut = 1, lff
          if(lic(neut) .and. k(neut).le.k1(neut))
     *      lhic(neut) = .true.
  109   continue
      endif
      matrix = larpos .or. lunit .or. lmhole .or. lmarry
      if(ltrk)call trkwrt('fstart  ',wt,tme,u,v,w,x,y,z,
     *  kcore,nbx,nby,nbz,k,ig,nff,kr,ki,ll,k1,k2,now,
     *  igeo,nbank,lff)
      go to 1003
c********************** end fstart ******************
c
c******************start control section***************
 1001 levent = 0
      nevent = 1
      if(lff .gt. 0) then
        levent = lff
        nevent = 2
      endif
      if(lcrsi .gt. levent) then
        levent = lcrsi
        nevent = 3
      endif
      if(lcrso .gt. levent) then
        levent = lcrso
        nevent = 4
      endif
      if(lcol .gt. levent) then
        levent = lcol
        nevent = 5
      endif
      go to (1000,1002,1004,1005,1006) nevent
c********************end control section****************
```

```
c*******calculate new fission neutron energy group*****
 1000 igchi = 1
      do 121 neut = 1, nfiss
        r(neut) = ranf()
        itar1(neut) = mat(ifbank(neut,5))
  121 continue
      do 122 neut = 1, nfiss
        itar1(neut) = mixc(itar1(neut))
  122 continue
      do 123 neut = 1, nfiss
        do 124 igt = 1, ngp
          if(r(neut).le.chi(igt,igchi,itar1(neut)))
     *    go to 129
  124   continue
        igt = ngp
  129   ifbank(neut,6) = igt
  123 continue
      return
c******end fission neutron energy group calculation****
c
c********** start russian roulette ********************
 1002 do 200 neut = 1, lff
        itar4(neut) = k(nff(neut))
  200 continue
      do 222  neut = 1, lff
        itar5(neut) = imp(itar4(neut))
  222 continue
      do 201 neut = 1, lff
        itar1(neut) = ig(nff(neut))
        itar2(neut) = impc(itar5(neut))
        ki(nff(neut)) = impc(itar5(neut))
        itar3(neut) = nff(neut)
  201 continue
      do 202 neut = 1, lff
        rtar3(neut) = wtavg(itar1(neut),itar2(neut))
        rtar4(neut) = wt(nff(neut))
  202 continue
      if(ltrk)call
trkwrt('rus-roul',wt,tme,u,v,w,x,y,z,kcore,nbx,nby,
     *  nbz,k,ig,nff,kr,ki,ll,k1,k2,now,igeo,nbank,lff)
      ltemp = lff
      lff   = 0
      lffl  = 0
      lffh  = 0
      do 203 neut = 1, ltemp
        if(rtar4(neut) .lt. rtar3(neut)*wtlow) then
          lffl = lffl + 1
          itar4(lffl) = itar3(neut)
          rtar1(lffl) = rtar3(neut)
        else if(rtar4(neut).gt.rtar3(neut)*wthigh)then
          lffh = lffh + 1
          itar5(lffh) = itar3(neut)
```

```
          rtar2(lffh) = rtar3(neut)
        else
          lff = lff + 1
          nff(lff) = itar3(neut)
        endif
  203 continue
      if(lffl .gt. 0) then
cdir$ ivdep
      do 204 neut = 1, lffl
        if(wt(itar4(neut)).lt.ranf()*rtar1(neut))then
          lkil = lkil + 1
          nkill(lkil) = itar4(neut)
          tlf = tlf + wt(itar4(neut))*tme(itar4(neut))
        else
          lff = lff + 1
          nff(lff) = itar4(neut)
          wt(itar4(neut)) = rtar1(neut)
        endif
  204 continue
      endif
      if(lffh .gt. 0) then
        lkil = lkil - lffh
cdir$  ivdep
        do 205 neut = 1, lffh
          lff = lff + 1
          nff(lff) = itar5(neut)
          itar1(neut) = nkill(lkil+neut)
          wt(itar5(neut)) = wt(itar5(neut))/2.0
  205   continue
cdir$ ivdep
  230   do 206 neut = 1, lffh
          lff = lff + 1
          nff(lff)           = itar1(neut)
          wt(itar1(neut))    = wt(itar5(neut))
          tme(itar1(neut))   = tme(itar5(neut))
          u(itar1(neut))     = u(itar5(neut))
          v(itar1(neut))     = v(itar5(neut))
          w(itar1(neut))     = w(itar5(neut))
          x(itar1(neut))     = x(itar5(neut))
          y(itar1(neut))     = y(itar5(neut))
          z(itar1(neut))     = z(itar5(neut))
          rpth(itar1(neut))  = 0.0
          k(itar1(neut))     = k(itar5(neut))
          ll(itar1(neut))    = ll(itar5(neut))
          k1(itar1(neut))    = k1(itar5(neut))
          k2(itar1(neut))    = k2(itar5(neut))
          ig(itar1(neut))    = ig(itar5(neut))
          ki(itar1(neut))    = ki(itar5(neut))
          kcore(itar1(neut)) = kcore(itar5(neut))
          nbx(itar1(neut))   = nbx(itar5(neut))
          nby(itar1(neut))   = nby(itar5(neut))
          nbz(itar1(neut))   = nbz(itar5(neut))
```

```
          lic(itar1(neut))    = lic(itar5(neut))
          now(itar1(neut))    = now(itar5(neut))
          larray(itar1(neut))= larray(itar5(neut))
  206   continue
        if(lnmhol) then
cdir$ ivdep
          do 207 neut = 1, lffh
            lhic(itar1(neut))  = lhic(itar5(neut))
            lhole(itar1(neut)) = lhole(itar5(neut))
            nhole(itar1(neut)) = nhole(itar5(neut))
  207     continue
        endif
        if(nesta) then
cdir$ ivdep
          do 208 neut = 1, lffh
            narrin(itar1(neut)) = narrin(itar5(neut))
  208     continue
          do 209 ik = 1, lnstka
cdir$ ivdep
            do 210 neut = 1, lffh
              stacka(itar1(neut),ik) =
     *        stacka(itar5(neut),ik)
  210       continue
  209     continue
        endif
        if(nesth) then
cdir$ ivdep
          do 211 neut = 1, lffh
            nholin(itar1(neut)) = nholin(itar5(neut))
  211     continue
          do 212 ik = 1, lnstkh
cdir$ ivdep
            do 213 neut = 1, lffh
              stackh(itar1(neut),ik) =
     *        stackh(itar5(neut),ik)
  213       continue
  212     continue
        endif
        if(lmult) then
cdir$ ivdep
          do 214 neut = 1, lffh
            nsmult(itar1(neut)) = nsmult(itar5(neut))
  214     continue
        endif
        if(matrix) then
          if(lunit) then
cdir$ ivdep
            do 215 neut = 1, lffh
              lls(itar1(neut)) = lls(itar5(neut))
  215       continue
          endif
          if(lmhole) then
```

```
cdir$ ivdep
            do 216 neut = 1, lffh
              nhl(itar1(neut)) = nhl(itar5(neut))
  216       continue
          endif
          if(lmarry) then
cdir$ ivdep
            do 217 neut = 1, lffh
              nar(itar1(neut)) = nar(itar5(neut))
  217       continue
          endif
          if(larpos) then
cdir$ ivdep
            do 218 neut = 1, lffh
              kft(itar1(neut)) = kft(itar5(neut))
  218       continue
          endif
        endif
        ltemp = 0
        do 219 neut = 1, lffh
          if(wt(itar5(neut)).gt.rtar2(neut)*wthigh)then
            ltemp = ltemp + 1
            itar6(ltemp) = itar5(neut)
            rtar1(ltemp) = rtar2(neut)
          endif
  219   continue
        do 220 neut = 1, lffh
          if(wt(itar1(neut)).gt.rtar2(neut)*wthigh)then
            ltemp = ltemp + 1
            itar6(ltemp) = itar1(neut)
            rtar1(ltemp) = rtar2(neut)
          endif
  220   continue
        if(ltemp .le. 0) go to 1003
        lkil = lkil - ltemp
        lffh = ltemp
cdir$ ivdep
        do 221 neut = 1, lffh
          itar1(neut) = nkill(lkil+neut)
          itar5(neut) = itar6(neut)
          rtar2(neut) = rtar1(neut)
          wt(itar6(neut)) = wt(itar6(neut))/2.0
  221   continue
        go to 230
      endif
c**************** end russian roulette ***************
c
c****************** start free flight *****************
cdir$ ivdep
 1003 do 301 neut = 1, lff
        if(rpth(nff(neut)).le.0.0) rpth(nff(neut)) =
     *    -log(ranf())
```

```
  301 continue
c     if(igen.eq.1) then
c       write(outpt,10003)
c       do 300 neut = 1, lff
c         ii = nff(neut)
c         write(outpt,10000)ii,x(ii),y(ii),z(ii),u(ii),
c    *    v(ii),w(ii),k(ii),ll(ii),kcore(ii),nbx(ii),
c    *    nby(ii),nbz(ii),nhole(ii)
c 300   continue
c     endif
      do 302 neut = 1, lff
        itar1(neut) = k(nff(neut))
  302 continue
      do 303 neut = 1, lff
        itar2(neut) = mat(itar1(neut))
        itar3(neut) = igeom(itar1(neut))
  303 continue
cdir$ ivdep
      do 304 neut = 1, lff
        if(itar2(neut).le.0 .or. itar3(neut).le.0) then
          kr(nff(neut)) = 0
        else
          kr(nff(neut)) = mixc(itar2(neut))
        endif
  304 continue
      do 305 neut = 1, lff
        itar1(neut) = ig(nff(neut))
        itar2(neut) = kr(nff(neut))
  305 continue
cdir$ ivdep
      do 306 neut = 1, lff
        if(itar2(neut).eq.0) then
          pth(nff(neut)) = big
        else
          pth(nff(neut))=rpth(nff(neut))/
     *    sigt(itar1(neut),itar2(neut))
        endif
  306 continue
      do 307 neut = 1, lff
      x1(nff(neut)) = x(nff(neut)) +
     *                pth(nff(neut))*u(nff(neut))
      y1(nff(neut)) = y(nff(neut)) +
     *                pth(nff(neut))*v(nff(neut))
      z1(nff(neut)) = z(nff(neut)) +
     *                pth(nff(neut))*w(nff(neut))
  307 continue
      if(ltrk)call trkwrt('free-flt',wt,tme,u,v,w,x,y,z,
     *  kcore,nbx,nby,nbz,k,ig,nff,kr,ki,ll,k1,k2,now,
     *  igeo,nbank,lff)
      ltmp = lcol
      do 308 neut = 1, ltmp
        if(x(nff(neut)) .eq. x1(nff(neut)) .and.
```

```
     *    y(nff(neut)) .eq. y1(nff(neut)) .and.
     *    z(nff(neut)) .eq. z1(nff(neut))) then
            lcol = lcol + 1
            ncol1(lcol) = nff(neut)
          endif
  308   continue
        if(ltmp .eq. lcol) go to 333
        ltmp = 0
        do 309 neut = 1, lff
          if(x(nff(neut)) .ne. x1(nff(neut)) .or.
     *    y(nff(neut)) .ne. y1(nff(neut)) .or.
     *    z(nff(neut)) .ne. z1(nff(neut))) then
            ltmp = ltmp + 1
            itar1(ltmp) = nff(neut)
          endif
  309   continue
        do 312 neut = 1, ltmp
          nff(neut) = itar1(neut)
  312   continue
        lff = ltmp
  333   do 310 neut = 1, lff
          if(lic(nff(neut))) then
            lcrsi = lcrsi + 1
            ncrosi(lcrsi) = nff(neut)
          else
            lcrso = lcrso + 1
            ncroso(lcrso) = nff(neut)
          endif
  310   continue
        do 311 neut = 1, lff
          nff(neut) = 0
  311   continue
        lff = 0
        go to 1001
c********************** end free flight ***************
c
c********************** start crossing ****************
c****************** begin inward crossing *************
 1004   n = -1
c       if(igen.eq.1) then
c         write(outpt,10004)
c         do 491 neut = 1, lcrsi
c           ii = ncrosi(neut)
c           write(outpt,10000)ii,x(ii),y(ii),z(ii),u(ii),
c    *      v(ii),w(ii),k(ii),ll(ii),kcore(ii),nbx(ii),
c    *      nby(ii),nbz(ii),nhole(ii)
c 491     continue
c       endif
        if(lnmara) then
          ltemp = lcrsi
          lara = 0
          lcrsi = 0
```

```
      do 401 neut = 1, ltemp
        if(igeom(k(ncrosi(neut))).eq.-1) then
          lara = lara + 1
          itar9(lara) = ncrosi(neut)
        else
          lcrsi = lcrsi + 1
          ncrosi(lcrsi) = ncrosi(neut)
        endif
401   continue
    endif
    if(lnmhol) then
      ltemp = lcrsi
      lhol = 0
      lcrsi = 0
      do 402 neut = 1, ltemp
        if(lhic(ncrosi(neut))) then
          lhol = lhol + 1
          itar8(lhol) = ncrosi(neut)
        else
          lcrsi = lcrsi + 1
          ncrosi(lcrsi) = ncrosi(neut)
        endif
402   continue
      lhol1 = lcrsi
      if(lhol.gt.0) then
        do 418 neut = 1, lhol
          lhol1 = lhol1 + 1
          ncrosi(lhol1) = itar8(neut)
418     continue
      endif
    endif
    do 403 neut = 1, lcrsi
      itar1(neut) = ncrosi(neut)
      itar2(neut) = k(ncrosi(neut)) - 1
403 continue
    do 404 neut = 1, lcrsi
      igeo(neut) = iabs(igeom(itar2(neut)))
404 continue
    call clear(ngeom,20)
    lgtl = 0
    do 405 lgt = 1,lgtyp
      go to (41,42,43,41,45,46,47,48,49,40)jgeom(lgt)
 41   ngeom(1,1) = lgtl + 1
      do 406 neut = 1, lcrsi
        if (igeo(neut) .eq. 1) then
          lgtl = lgtl + 1
          ncrosi(lgtl) = itar1(neut)
        endif
406   continue
      if(lgtl.ge.ngeom(1,1)) ngeom(2,1) = lgtl
      go to 405
 42   ngeom(1,2) = lgtl + 1
```

```
      do 407 neut = 1, lcrsi
        if (igeo(neut).eq.2) then
          lgtl = lgtl + 1
          ncrosi(lgtl) = itar1(neut)
        endif
407   continue
      if(lgtl.ge.ngeom(1,2)) ngeom(2,2) = lgtl
      go to 405
 43   ngeom(1,3) = lgtl + 1
      do 408 neut = 1, lcrsi
        if (igeo(neut).eq.3) then
          lgtl = lgtl + 1
          ncrosi(lgtl) = itar1(neut)
        endif
408   continue
      if(lgtl.ge.ngeom(1,3)) ngeom(2,3) = lgtl
      go to 405
 45   ngeom(1,5) = lgtl + 1
      do 409 neut = 1, lcrsi
        if (igeo(neut).eq.5) then
          lgtl = lgtl + 1
          ncrosi(lgtl) = itar1(neut)
        endif
409   continue
      if(lgtl.ge.ngeom(1,5)) ngeom(2,5) = lgtl
      go to 405
 46   ngeom(1,6) = lgtl + 1
      do 410 neut = 1, lcrsi
        if (igeo(neut).eq.6) then
          lgtl = lgtl + 1
          ncrosi(lgtl) = itar1(neut)
        endif
410   continue
      if(lgtl .ge. ngeom(1,5)) ngeom(2,6) = lgtl
      go to 405
 47   ngeom(1,7) = lgtl + 1
      do 411 neut = 1, lcrsi
        if(igeo(neut).eq.7) then
          lgtl = lgtl + 1
          ncrosi(lgtl) = itar1(neut)
        endif
411   continue
      if(lgtl.ge.ngeom(1,7)) ngeom(2,7) = lgtl
      go to 405
 48   ngeom(1,8) = lgtl + 1
      do 412 neut = 1, lcrsi
        if(igeo(neut).ge.12.and.igeo(neut).le.15)then
          lgtl = lgtl + 1
          ncrosi(lgtl) = itar1(neut)
        endif
412   continue
      if(lgtl.ge.ngeom(1,8)) ngeom(2,8) = lgtl
```

```
      go to 405
 49   ngeom(1,9) = lgtl + 1
      do 413 neut = 1, lcrsi
        if(igeo(neut).ge.16.and.igeo(neut).le.19)then
          lgtl = lgtl + 1
          ncrosi(lgtl) = itar1(neut)
        endif
413   continue
      if(lgtl.ge.ngeom(1,9)) ngeom(2,9) = lgtl
      go to 405
 40   ngeom(1,10) = lgtl + 1
      do 414 neut = 1, lcrsi
        if(igeo(neut).ge.8.and.igeo(neut).le.11)then
          lgtl = lgtl + 1
          ncrosi(lgtl) = itar1(neut)
        endif
414   continue
      if(lgtl.ge.ngeom(1,9)) ngeom(2,10) = lgtl
      go to 405
405 continue
    lcrsi = lgtl
    do 415 neut = 1, nbank
      etausd(neut) = 0.0
      etamin(neut) = 1.0
415 continue
    if(lnmhol) then
      lcrsh = 0
      do 720 neut = 1, lhol1
        if(ilh(k(ncrosi(neut))).gt.0) then
          lcrsh = lcrsh + 1
          itar1(lcrsh) = ncrosi(neut)
        endif
720   continue
      do 416 neut = 1, lcrsh
        rtar1(itar1(neut)) = x(itar1(neut))
        rtar2(itar1(neut)) = y(itar1(neut))
        rtar3(itar1(neut)) = z(itar1(neut))
416   continue
      do 417 neut = 1, nbank
        ltar1(neut) = .false.
        itar4(neut) = 0
417   continue
    endif
    if(nflx .or. lnmhol) then
      lkflx = max(lcrsi,lhol1)
      do 419 neut = 1, lkflx
        itar7(ncrosi(neut)) = k(ncrosi(neut))
419   continue
    endif
    if(ltrk)call trkwrt('inward  ',wt,tme,u,v,w,x,y,
   *  z,kcore,nbx,nby,nbz,k,ig,ncrosi,kr,ki,ll,k1,
   *  k2,now,igeo,nbank,lcrsi)
```

```
      call cros(ncrosi,x,y,z,x1,y1,z1,etausd,k,xx,
     *  lcrsi,m,n,igeom,ngeom,jgeom,lgtyp,itar10,
     *  itar11,itar12,rtar10,rtar11,rtar12,rtar13,
     *  rtar14,rtar15,rtar16,rtar17,rtar18,rtar19,
     *  rtar20,rtar21,rtar22,rtar23,rtar24)
c****** m=0 means a crossing did not occur
c****** m=1 means a crossing did occur
c*********************begin hole in**********************
      if(lnmhol) then
cdir$ ivdep
        do 713 neut = 1, lcrsi
          if(m(neut).eq.1) then
            etamin(ncrosi(neut)) = etausd(ncrosi(neut))
            k(ncrosi(neut)) = k(ncrosi(neut)) - 1
            ltar1(ncrosi(neut)) = .true.
          endif
  713   continue
        lcrsi = lhol1
        do 721 ik = 1, krefm
          if(lcrsh.le.0) go to 799
          if(ilh(ik).le.0) go to 721
          lhol1 = lcrsh
          ltemp = 0
          lcrsh = 0
          do 722 neut = 1, lhol1
            if(k(itar1(neut)).eq.ik) then
              ltemp = ltemp + 1
              itar2(ltemp) = itar1(neut)
            else
              lcrsh = lcrsh + 1
              itar1(lcrsh) = itar1(neut)
            endif
  722     continue
          if(ltemp.le.0) go to 721
          do 723 ih = ifh(ik), ilh(ik)
            lhol1 = 0
            do 724 neut = 1, ltemp
              if(lsthol(itar2(neut)).ne.ih) then
                lhol1 = lhol1 + 1
                itar5(lhol1) = itar2(neut)
              endif
  724       continue
            if(lhol1.le.0) go to 723
            call clear(ngeom,20)
            khi = kbnds2(lholu(ih))
            khgeo = iabs(igeom(khi))
            if(khgeo.ge.12 .and. khgeo.le.15) khgeo = 8
            if(khgeo.ge.16 .and. khgeo.le.19) khgeo = 9
            if(khgeo.ge.8  .and. khgeo.le.11) khgeo = 10
            ngeom(1,khgeo) = 1
            ngeom(2,khgeo) = lhol1
            do 725 neut = 1, lhol1
```

```
              rtar7(itar5(neut)) = rtar1(itar5(neut)) -
     *                             holx(ih)
              rtar8(itar5(neut)) = rtar2(itar5(neut)) -
     *                             holy(ih)
              rtar9(itar5(neut)) = rtar3(itar5(neut)) -
     *                             holz(ih)
              rtar4(itar5(neut)) = x1(itar5(neut)) -
     *                             holx(ih)
              rtar5(itar5(neut)) = y1(itar5(neut)) -
     *                             holy(ih)
              rtar6(itar5(neut)) = z1(itar5(neut)) -
     *                             holz(ih)
              itar8(itar5(neut))  = khi + 1
              etausd(itar5(neut)) = 0.0
              m(neut) = 0
  725       continue
            call cros(itar5,rtar7,rtar8,rtar9,rtar4,
     *        rtar5,rtar6,etausd,itar8,xx,lhol1,m,n,
     *        igeom,ngeom,jgeom,lgtyp,itar10,itar11,
     *        itar12,rtar10,rtar11,rtar12,rtar13,rtar14,
     *        rtar15,rtar16,rtar17,rtar18,rtar19,rtar20,
     *        rtar21,rtar22,rtar23,rtar24)
cdir$ ivdep
            do 726 neut = 1, lhol1
              if(etausd(itar5(neut)).lt.etamin(itar5(neut))
     *         .and.m(neut).eq.1) then
                etamin(itar5(neut)) = etausd(itar5(neut))
                k(itar5(neut)) = itar8(itar5(neut)) - 1
                itar4(itar5(neut)) = ih
                ltar1(itar5(neut)) = .true.
                x(itar5(neut)) = rtar7(itar5(neut))
                y(itar5(neut)) = rtar8(itar5(neut))
                z(itar5(neut)) = rtar9(itar5(neut))
              endif
  726       continue
  723     continue
  721   continue
  799   continue
        do 734 neut = 1, lcrsi
          lsthol(ncrosi(neut)) = 0
          lhic(ncrosi(neut)) = .false.
  734   continue
        ltemp = lff + 1
        do 727 neut = 1, lcrsi
          if(ltar1(ncrosi(neut))) then
            lff = lff + 1
            nff(lff) = ncrosi(neut)
          else
            lcrso = lcrso + 1
            ncroso(lcrso) = ncrosi(neut)
          endif
  727   continue
```

```
      if(ltemp.gt.lff) go to 1044
      do 728 neut = ltemp, lff
        ll(nff(neut)) = kboxc(k(nff(neut)))
728   continue
      do 735 neut = ltemp, lff
        k1(nff(neut)) = kbnds1(ll(nff(neut)))
        k2(nff(neut)) = kbnds2(ll(nff(neut)))
735   continue
      if(nesth) then
        do 729 neut = ltemp, lff
          itar8(neut) = nholin(nff(neut)) + 1
729     continue
        do 730 neut = ltemp, lff
          if(lhole(nff(neut)) .and. itar4(nff(neut))
     *      .gt.0) then
            nholin(nff(neut)) = itar8(neut)
            stackh(nff(neut),itar8(neut)) =
     *        nhole(nff(neut))
          endif
730     continue
      endif
      do 731 neut = ltemp, lff
        if(itar4(nff(neut)).gt.0) then
          nhole(nff(neut)) = itar4(nff(neut))
          lhole(nff(neut)) = .true.
        endif
731   continue
      do 732 neut = ltemp, lff
        if(ilh(k(nff(neut))).le.0.and.k(nff(neut))
     *    .le.k1(nff(neut)))lic(nff(neut)) = .false.
732   continue
      do 733 neut = ltemp, lff
        if(ilh(k(nff(neut))).gt.0.and.k(nff(neut))
     *    .le.k1(nff(neut)))lhic(nff(neut)) = .true.
733   continue
      go to 1045
    endif
c************************end hole in*********************
      ltemp = lff + 1
      do 441 neut = 1, lcrsi
        if(m(neut).eq.1) then
          lff = lff + 1
          nff(lff) = ncrosi(neut)
          etamin(ncrosi(neut)) = etausd(ncrosi(neut))
        else
          lcrso = lcrso + 1
          ncroso(lcrso) = ncrosi(neut)
        endif
441   continue
      if(ltemp.gt.lff) go to 1044
cdir$ ivdep
      do 442 neut = ltemp, lff
```

```
        k(nff(neut)) = k(nff(neut)) - 1
  442 continue
      do 445 neut = ltemp, lff
        if(k(nff(neut)).le.k1(nff(neut)))
     *    lic(nff(neut)) = .false.
  445 continue
 1045 continue
cdir$ ivdep
      do 443 neut = ltemp, lff
        tme(nff(neut)) = tme(nff(neut))+pth(nff(neut))*
     *  etamin(nff(neut))*vinv(ig(nff(neut)))
        if(kr(nff(neut)).gt.0) rpth(nff(neut)) =
     *  rpth(nff(neut))*(1.0 - etamin(nff(neut)))
  443 continue
      if(nflx) then
        do 444 neut = ltemp, lff
          flux(ig(nff(neut)),iregc(itar7(nff(neut)))) =
     *    flux(ig(nff(neut)),iregc(itar7(nff(neut)))) +
     *    wt(nff(neut))*pth(nff(neut))*etamin(nff(neut))
  444   continue
      endif
c*******************begin findbx**********************
 1044 if(lnmara) then
        lcrsi = 0
        if(lff.ge.ltemp) then
        do 451 neut = ltemp, lff
          if(igeom(k(nff(neut))).eq.-1) then
            lcrsi = lcrsi + 1
            ncrosi(lcrsi) = nff(neut)
          endif
  451   continue
        endif
        if(lara.gt.0) then
        do 452 neut = 1, lara
          lff = lff + 1
          lcrsi = lcrsi + 1
          nff(lff) = itar9(neut)
          ncrosi(lcrsi) = itar9(neut)
  452   continue
        endif
        if(lcrsi.eq.0) go to 499
        if(ltrk)call trkwrt('findbx-b',wt,tme,u,v,w,x,
     *    y,z,kcore,nbx,nby,nbz,k,ig,ncrosi,kr,ki,ll,
     *    k1,k2,now,igeo,nbank,lcrsi)
        if(nesta) then
          do 453 neut = 1, lcrsi
            itar6(neut) = narrin(ncrosi(neut)) + 4
  453     continue
cdir$ ivdep
          do 454 neut = 1, lcrsi
            if(larray(ncrosi(neut))) then
              narrin(ncrosi(neut)) = itar6(neut)
```

```
              stacka(ncrosi(neut),itar6(neut))    =
     *        kcore(ncrosi(neut))
              stacka(ncrosi(neut),itar6(neut)+1) =
     *        nbx(ncrosi(neut))
              stacka(ncrosi(neut),itar6(neut)+2) =
     *        nby(ncrosi(neut))
              stacka(ncrosi(neut),itar6(neut)+3) =
     *        nbz(ncrosi(neut))
            endif
  454     continue
        endif
        do 455 neut = 1, lcrsi
          lic(ncrosi(neut)) = .true.
          larray(ncrosi(neut)) = .true.
          itar1(neut) = k(ncrosi(neut))
  455   continue
        do 456 neut = 1, lcrsi
          now(ncrosi(neut)) = mat(itar1(neut))
          itar2(neut) = mat(itar1(neut))
  456   continue
        do 457 neut = 1, lcrsi
          itar7(ncrosi(neut)) = 2
          itar8(ncrosi(neut)) = 4
          itar9(ncrosi(neut)) = 6
          rtar1(ncrosi(neut)) = 0.0
          rtar2(ncrosi(neut)) = 0.0
          rtar3(ncrosi(neut)) = 0.0
          kcore(ncrosi(neut)) = itar1(neut)
          itar3(neut) = irega(itar2(neut))
  457   continue
        lara = 0
cdir$ ivdep
        do 458 neut = 1, lcrsi
          if(x(ncrosi(neut)).eq.xx(1,itar1(neut))) then
            nbx(ncrosi(neut)) = nbxmx(itar2(neut))
            itar7(ncrosi(neut)) = 1
          else if(x(ncrosi(neut)).eq.xx(2,itar1(neut)))then
            nbx(ncrosi(neut)) = 1
          else
            lara = lara + 1
            itar6(lara) = ncrosi(neut)
            itar5(lara) = itar1(neut)
            itar4(lara) = indxx(itar3(neut))
            rtar1(ncrosi(neut)) = x(ncrosi(neut)) -
     *                            xx(2,itar1(neut))
          endif
  458   continue
        if(lara.eq.0) go to 1041
        do 459 ii = 2, maxnbx+1
          ltemp = lara
          lara = 0
cdir$ ivdep
```

```
          do 460 neut = 1, ltemp
            if(rtar1(itar6(neut)).lt.deltx(itar4(neut)+
     *      ii)) then
              nbx(itar6(neut)) = ii - 1
              rtar1(itar6(neut)) = rtar1(itar6(neut)) -
     *                           deltx(itar4(neut)+ii-1)
            else
              lara = lara + 1
              itar6(lara) = itar6(neut)
              itar4(lara) = itar4(neut)
            endif
  460     continue
          if(lara.eq.0) go to 1041
  459   continue
 1041   lara = 0
cdir$ ivdep
        do 461 neut = 1, lcrsi
          if(y(ncrosi(neut)).eq.xx(3,itar1(neut))) then
            nby(ncrosi(neut)) = nbymx(itar2(neut))
            itar8(ncrosi(neut)) = 3
          else if(y(ncrosi(neut)).eq.xx(4,itar1(neut)))then
            nby(ncrosi(neut)) = 1
          else
            lara = lara + 1
            itar6(lara) = ncrosi(neut)
            itar5(lara) = itar1(neut)
            itar4(lara) = indxy(itar3(neut))
            rtar2(ncrosi(neut)) = y(ncrosi(neut)) -
     *                            xx(4,itar1(neut))
          endif
  461   continue
        if(lara.eq.0) go to 1042
        do 462 ii = 2, maxnby+1
          ltemp = lara
          lara = 0
cdir$ ivdep
          do 463 neut = 1, ltemp
            if(rtar2(itar6(neut)).lt.delty(itar4(neut)+
     *      ii)) then
              nby(itar6(neut)) = ii - 1
              rtar2(itar6(neut)) = rtar2(itar6(neut)) -
     *                           delty(itar4(neut)+ii-1)
            else
              lara = lara + 1
              itar6(lara) = itar6(neut)
              itar4(lara) = itar4(neut)
          endif
  463     continue
          if(lara.eq.0) go to 1042
  462   continue
 1042   lara = 0
cdir$ ivdep
```

```
        do 464 neut = 1, lcrsi
          if(z(ncrosi(neut)).eq.xx(5,itar1(neut))) then
            nbz(ncrosi(neut)) = nbzmx(itar2(neut))
            itar9(ncrosi(neut)) = 5
          else if(z(ncrosi(neut)).eq.xx(6,itar1(neut)))then
            nbz(ncrosi(neut)) = 1
          else
            lara = lara + 1
            itar6(lara) = ncrosi(neut)
            itar5(lara) = itar1(neut)
            itar4(lara) = indxz(itar3(neut))
            rtar3(ncrosi(neut)) = z(ncrosi(neut)) -
     *                            xx(6,itar1(neut))
          endif
  464   continue
        if(lara.eq.0) go to 1043
        do 465 ii = 2, maxnbz+1
          ltemp = lara
          lara = 0
cdir$ ivdep
          do 466 neut = 1, ltemp
            if(rtar3(itar6(neut)).lt.deltz(itar4(neut)+
     *      ii)) then
              nbz(itar6(neut)) = ii - 1
              rtar3(itar6(neut)) = rtar3(itar6(neut)) -
     *                             deltz(itar4(neut)+ii-1)
            else
              lara = lara + 1
              itar6(lara) = itar6(neut)
              itar4(lara) = itar4(neut)
            endif
  466     continue
          if(lara.eq.0) go to 1043
  465   continue
 1043   continue
        if(mbox) then
          do 467 neut = 1, lcrsi
            ll(ncrosi(neut)) = lba(ndxlba(itar3(neut)) - 1
+
     *      nbx(ncrosi(neut)) + nbxmx(itar2(neut))*
     *      (nby(ncrosi(neut)) - 1 + nbymx(itar2(neut))*
     *      (nbz(ncrosi(neut)) - 1)))
  467     continue
        else
          do 472 neut = 1, lcrsi
            ll(ncrosi(neut)) = 1
  472     continue
        endif
        do 468 neut = 1, lcrsi
          itar6(neut) = ll(ncrosi(neut))
  468   continue
        do 469 neut = 1, lcrsi
```

```
          k1(ncrosi(neut)) = kbnds1(itar6(neut))
          k2(ncrosi(neut)) = kbnds2(itar6(neut))
          k(ncrosi(neut))  = kbnds2(itar6(neut))
          itar1(neut)      = kbnds2(itar6(neut))
          itar2(neut)      = itar7(ncrosi(neut))
          itar3(neut)      = itar8(ncrosi(neut))
          itar4(neut)      = itar9(ncrosi(neut))
  469   continue
        do 470 neut = 1, lcrsi
        x(ncrosi(neut)) = rtar1(ncrosi(neut))+
     *                    xx(itar2(neut),itar1(neut))
        y(ncrosi(neut)) = rtar2(ncrosi(neut))+
     *                    xx(itar3(neut),itar1(neut))
        z(ncrosi(neut)) = rtar3(ncrosi(neut))+
     *                    xx(itar4(neut),itar1(neut))
  470   continue
        if(ltrk)call trkwrt('findbx-e',wt,tme,u,v,w,x,
     *  y,z,kcore,nbx,nby,nbz,k,ig,ncrosi,kr,ki,ll,
     *  k1,k2,now,igeo,nbank,lcrsi)
       endif
c**********************end findbx**********************
  499 do 471 neut = 1, nbank
        ncrosi(neut) = 0
  471 continue
      lcrsi = 0
c     if(igen.eq.1) then
c       write(outpt,10043)
c       do 492 neut = 1, lff
c         ii = nff(neut)
c         write(outpt,10000)ii,x(ii),y(ii),z(ii),u(ii),
c    *    v(ii),w(ii),k(ii),ll(ii),kcore(ii),nbx(ii),
c    *    nby(ii),nbz(ii),nhole(ii)
c 492   continue
c       write(outpt,10045)
c       do 493 neut = 1, lcrso
c         ii = ncroso(neut)
c         write(outpt,10000)ii,x(ii),y(ii),z(ii),u(ii),
c    *    v(ii),w(ii),k(ii),ll(ii),kcore(ii),nbx(ii),
c    *    nby(ii),nbz(ii),nhole(ii)
c 493   continue
c     endif
      go to 1001
c******************** end inward crossing *************
c
c**********************start posit*********************
 1005 do 500 neut = 1, lcrso
        itar1(neut) = ncroso(neut)
        itar2(neut) = k(ncroso(neut))
  500 continue
c     if(igen.eq.1) then
c       write(outpt,10005)
c       do 551 neut = 1, lcrso
```

```
c          ii = ncroso(neut)
c          write(outpt,10000)ii,x(ii),y(ii),z(ii),u(ii),
c     *    v(ii),w(ii),k(ii),ll(ii),kcore(ii),nbx(ii),
c     *    nby(ii),nbz(ii),nhole(ii)
c 551    continue
c      endif
       do 502 neut = 1, lcrso
         igeo(neut) = iabs(igeom(itar2(neut)))
  502  continue
       if(ltrk)call trkwrt('posit   ',wt,tme,u,v,w,x,y,
      *  z,kcore,nbx,nby,nbz,k,ig,ncroso,kr,ki,ll,k1,k2,
      *  now,igeo,nbank,lcrso)
       call clear(ngeom,20)
       lgtl = 0
       do 503 lgt = 1,lgtyp
         go to (51,52,53,51,55,56,57,58,59,50) jgeom(lgt)
    51   ngeom(1,1) = lgtl + 1
         do 504 neut = 1, lcrso
           if (igeo(neut) .eq. 1) then
             lgtl = lgtl + 1
             ncroso(lgtl) = itar1(neut)
           endif
  504    continue
         if(lgtl .ge. ngeom(1,1)) ngeom(2,1) = lgtl
         go to 503
    52   ngeom(1,2) = lgtl + 1
         do 505 neut = 1, lcrso
           if (igeo(neut).eq.2) then
             lgtl = lgtl + 1
             ncroso(lgtl) = itar1(neut)
           endif
  505    continue
         if(lgtl .ge. ngeom(1,2)) ngeom(2,2) = lgtl
         go to 503
    53   ngeom(1,3) = lgtl + 1
         do 507 neut = 1, lcrso
           if (igeo(neut).eq.3) then
             lgtl = lgtl + 1
             ncroso(lgtl) = itar1(neut)
           endif
  507    continue
         if(lgtl .ge. ngeom(1,3)) ngeom(2,3) = lgtl
         go to 503
    55   ngeom(1,5) = lgtl + 1
         do 509 neut = 1, lcrso
           if (igeo(neut).eq.5) then
             lgtl = lgtl + 1
             ncroso(lgtl) = itar1(neut)
           endif
  509    continue
         if(lgtl .ge. ngeom(1,5)) ngeom(2,5) = lgtl
         go to 503
```

```
 56   ngeom(1,6) = lgtl + 1
      do 511 neut = 1, lcrso
        if (igeo(neut).eq.6) then
          lgtl = lgtl + 1
          ncroso(lgtl) = itar1(neut)
        endif
511   continue
      if(lgtl .ge. ngeom(1,6)) ngeom(2,6) = lgtl
      go to 503
 57   ngeom(1,7) = lgtl + 1
      do 508 neut = 1, lcrso
        if(igeo(neut).eq.7) then
          lgtl = lgtl + 1
          ncroso(lgtl) = itar1(neut)
        endif
508   continue
      if(lgtl .ge. ngeom(1,7)) ngeom(2,7) = lgtl
      go to 503
 58   ngeom(1,8) = lgtl + 1
      do 510 neut = 1, lcrso
        if(igeo(neut).ge.12 .and. igeo(neut).le.15) then
          lgtl = lgtl + 1
          ncroso(lgtl) = itar1(neut)
        endif
510   continue
      if(lgtl .ge. ngeom(1,8)) ngeom(2,8) = lgtl
      go to 503
 59   ngeom(1,9) = lgtl + 1
      do 512 neut = 1, lcrso
        if(igeo(neut).ge.16 .and. igeo(neut).le.19) then
          lgtl = lgtl + 1
          ncroso(lgtl) = itar1(neut)
        endif
512   continue
      if(lgtl .ge. ngeom(1,9)) ngeom(2,9) = lgtl
      go to 503
 50   ngeom(1,10) = lgtl + 1
      do 506 neut = 1, lcrso
        if(igeo(neut).ge.8 .and. igeo(neut).le.11)then
          lgtl = lgtl + 1
          ncroso(lgtl) = itar1(neut)
        endif
506   continue
      if(lgtl .ge. ngeom(1,10)) ngeom(2,10) = lgtl
    go to 503
503 continue
    do 570 neut = 1, lcrso
      itar1(neut) = k(ncroso(neut))
570 continue
    lcrso = 0
    do 520 lgt = 1, lgtyp
    go to (61,62,63,61,65,66,67,68,69,60) jgeom(lgt)
```

```
c************ begin cuboid check **********************
   61 if(ngeom(2,1).eq.0) go to 520
      ltmp = lcrso + 1
      do 521 neut = ngeom(1,1), ngeom(2,1)
        if(x1(ncroso(neut)).le.xx(1,itar1(neut)).and.
     *    x1(ncroso(neut)).ge.xx(2,itar1(neut)).and.
     *    y1(ncroso(neut)).le.xx(3,itar1(neut)).and.
     *    y1(ncroso(neut)).ge.xx(4,itar1(neut)).and.
     *    z1(ncroso(neut)).le.xx(5,itar1(neut)).and.
     *    z1(ncroso(neut)).ge.xx(6,itar1(neut))) then
          lcol = lcol + 1
          ncoll(lcol) = ncroso(neut)
        else
          lcrso = lcrso + 1
          itar4(lcrso) = ncroso(neut)
        endif
  521 continue
        ngeom(2,1) = 0
        if(lcrso .ge. ltmp) ngeom(2,1) = lcrso
      go to 520
c**************** end cuboid ***************************
c********** begin z-cylinder check ********************
   62 if(ngeom(2,2).eq.0) go to 520
      ltmp = lcrso + 1
      do 522 neut = ngeom(1,2), ngeom(2,2)
        rtar1(neut)=(x1(ncroso(neut))-xx(5,itar1(neut)))**2
     *  + (y1(ncroso(neut))-xx(6,itar1(neut)))**2
        if(z1(ncroso(neut)).le.xx(2,itar1(neut)).and.
     *    z1(ncroso(neut)).ge.xx(3,itar1(neut)).and.
     *    rtar1(neut).le.xx(4,itar1(neut))) then
          lcol = lcol + 1
          ncoll(lcol) = ncroso(neut)
        else
          lcrso = lcrso + 1
          itar4(lcrso) = ncroso(neut)
        endif
  522 continue
      ngeom(1,2) = ltmp
      ngeom(2,2) = 0
      if(lcrso .ge. ltmp) ngeom(2,2) = lcrso
      go to 520
c**************** end z-cylinder ***********************
c************** begin sphere check ********************
   63 if(ngeom(2,3).eq.0) go to 520
      ltmp = lcrso + 1
      do 523 neut = ngeom(1,3), ngeom(2,3)
        rtar1(neut)=(x1(ncroso(neut))-xx(4,itar1(neut)))**2
     *  + (y1(ncroso(neut))-xx(5,itar1(neut)))**2 +
     *    (z1(ncroso(neut))-xx(6,itar1(neut)))**2
  523 continue
      do 538 neut = ngeom(1,3), ngeom(2,3)
        if(rtar1(neut).le.xx(2,itar1(neut))) then
```

```
            lcol = lcol + 1
            ncoll(lcol) = ncroso(neut)
          else
            lcrso = lcrso + 1
            itar4(lcrso) = ncroso(neut)
          endif
  538 continue
      ngeom(1,3) = ltmp
      ngeom(2,3) = 0
      if(lcrso .ge. ltmp) ngeom(2,3) = lcrso
      go to 520
c*************** end sphere ***************************
c********** begin x-cylinder check ********************
   65 if(ngeom(2,5).eq.0) go to 520
      ltmp = lcrso + 1
      do 524 neut = ngeom(1,5), ngeom(2,5)
        rtar1(neut)=(y1(ncroso(neut))-xx(5,itar1(neut)))**2
     *  + (z1(ncroso(neut))-xx(6,itar1(neut)))**2
        if(x1(ncroso(neut)).le.xx(2,itar1(neut)) .and.
     *    x1(ncroso(neut)).ge.xx(3,itar1(neut)) .and.
     *    rtar1(neut).le.xx(4,itar1(neut))) then
          lcol = lcol + 1
          ncoll(lcol) = ncroso(neut)
        else
          lcrso = lcrso + 1
          itar4(lcrso) = ncroso(neut)
        endif
  524 continue
      ngeom(1,5) = ltmp
      ngeom(2,5) = 0
      if(lcrso .ge. ltmp) ngeom(2,5) = lcrso
      go to 520
c*************** end x-cyliner ************************
c******************* begin y-cyliner check
**********************
   66 if(ngeom(2,6).eq.0) go to 520
      ltmp = lcrso + 1
      do 525 neut = ngeom(1,6), ngeom(2,6)
        rtar1(neut)=(x1(ncroso(neut))-xx(5,itar1(neut)))**2
     *  + (z1(ncroso(neut))-xx(6,itar1(neut)))**2
        if(y1(ncroso(neut)).le.xx(2,itar1(neut)) .and.
     *    y1(ncroso(neut)).ge.xx(3,itar1(neut)) .and.
     *    rtar1(neut).le.xx(4,itar1(neut))) then
          lcol = lcol + 1
          ncoll(lcol) = ncroso(neut)
        else
          lcrso = lcrso + 1
          itar4(lcrso) = ncroso(neut)
        endif
  525 continue
      ngeom(1,6) = ltmp
      ngeom(2,6) = 0
```

```
      if(lcrso .ge. ltmp) ngeom(2,6) = lcrso
      go to 520
c******************end y-cylinder *********************
c************* begin hemisphere check ****************
   67 if(ngeom(2,7).eq.0) go to 520
      ltmp = lcrso + 1
      do 526 neut = ngeom(1,7), ngeom(2,7)
        rtar3(neut) = sign(1.0,xx(3,itar1(neut)))
        itar3(neut) = (1 + abs(xx(3,itar1(neut))))/2
        rtar1(neut)=(x1(ncroso(neut))-xx(4,itar1(neut)))**2
     *  + (y1(ncroso(neut))-xx(5,itar1(neut)))**2 +
     *  (z1(ncroso(neut))-xx(6,itar1(neut)))**2
  526 continue
      if(lhsphx) then
      do 527 neut = ngeom(1,7), ngeom(2,7)
        if(itar3(neut) .eq. 1) rtar2(neut) =
     *  rtar3(neut)*(x1(ncroso(neut))-xx(4,itar1(neut)))
  527 continue
      endif
      if(lhsphy) then
      do 575 neut = ngeom(1,7), ngeom(2,7)
        if(itar3(neut) .eq. 2) rtar2(neut) =
     *  rtar3(neut)*(y1(ncroso(neut))-xx(5,itar1(neut)))
  575 continue
      endif
      if(lhsphz) then
      do 576 neut = ngeom(1,7), ngeom(2,7)
        if(itar3(neut) .eq. 3) rtar2(neut) =
     *  rtar3(neut)*(z1(ncroso(neut))-xx(6,itar1(neut)))
  576 continue
      endif
      do 528 neut = ngeom(1,7), ngeom(2,7)
        if(rtar1(neut) .le. xx(2,itar1(neut)) .and.
     *    rtar2(neut) .ge. -xx(7,itar1(neut))) then
          lcol = lcol + 1
          ncoll(lcol) = ncroso(neut)
        else
          lcrso = lcrso + 1
          itar4(lcrso) = ncroso(neut)
        endif
  528 continue
      ngeom(1,7) = ltmp
      ngeom(2,7) = 0
      if(lcrso .ge. ltmp) ngeom(2,7) = lcrso
      go to 520
c*************** end hemisphere ***********************
c********** begin x-hemicylinder check ***************
   68 if(ngeom(2,8).eq.0)go to 520
      ltmp = lcrso + 1
      do 532 neut = ngeom(1,8), ngeom(2,8)
        rtar1(neut)=(y1(ncroso(neut))-xx(5,itar1(neut)))**2
     *  + (z1(ncroso(neut))-xx(6,itar1(neut)))**2
```

```
        rtar3(neut) = 2*mod(igeom(itar1(neut)),2)-1
        itar2(neut) = (igeom(itar1(neut))-10)/2
  532 continue
      if(lxhcyz) then
      do 533 neut = ngeom(1,8), ngeom(2,8)
        if(itar2(neut) .eq. 1) rtar4(neut) =
     *   rtar3(neut)*(z1(ncroso(neut))-xx(6,itar1(neut)))
  533 continue
      endif
      if(lxhcyy) then
      do 577 neut = ngeom(1,8), ngeom(2,8)
        if(itar2(neut) .eq. 2) rtar4(neut) =
     *    rtar3(neut)*(y1(ncroso(neut))-xx(5,itar1(neut)))
  577 continue
      endif
      do 534 neut = ngeom(1,8), ngeom(2,8)
        if(rtar4(neut).ge.-xx(7,itar1(neut)) .and.
     *     rtar1(neut).le.xx(4,itar1(neut))  .and.
     *     x1(ncroso(neut)).le.xx(2,itar1(neut)) .and.
     *     x1(ncroso(neut)).ge.xx(3,itar1(neut))) then
          lcol = lcol + 1
          ncol1(lcol) = ncroso(neut)
        else
          lcrso = lcrso + 1
          itar4(lcrso) = ncroso(neut)
        endif
  534 continue
      ngeom(1,8) = ltmp
      ngeom(2,8) = 0
      if(lcrso .ge. ltmp) ngeom(2,8) = lcrso
      go to 520
c**************** end x-hemicylinder ******************
c************ begin y-hemicylinder check **************
   69 if(ngeom(2,9).eq.0) go to 520
      ltmp = lcrso + 1
      do 535 neut = ngeom(1,9), ngeom(2,9)
        rtar1(neut)=(x1(ncroso(neut))-xx(5,itar1(neut)))**2
     *  + (z1(ncroso(neut))-xx(6,itar1(neut)))**2
        rtar3(neut) = 2*mod(igeom(itar1(neut)),2)-1
        itar2(neut) = (igeom(itar1(neut))-14)/2
  535 continue
      if(lyhcyz) then
      do 536 neut = ngeom(1,9), ngeom(2,9)
        if(itar2(neut) .eq. 1) rtar4(neut) =
     *    rtar3(neut)*(z1(ncroso(neut))-xx(6,itar1(neut)))
  536 continue
      endif
      if(lyhcyx) then
      do 578 neut = ngeom(1,9), ngeom(2,9)
        if(itar2(neut) .eq. 2) rtar4(neut) =
     *    rtar3(neut)*(x1(ncroso(neut))-xx(5,itar1(neut)))
  578 continue
```

```
      endif
      do 537 neut = ngeom(1,9), ngeom(2,9)
        if(rtar4(neut).ge.-xx(7,itar1(neut)) .and.
     *     rtar1(neut).le.xx(4,itar1(neut))  .and.
     *     y1(ncroso(neut)).le.xx(2,itar1(neut)) .and.
     *     y1(ncroso(neut)).ge.xx(3,itar1(neut))) then
          lcol = lcol + 1
          ncoll(lcol) = ncroso(neut)
        else
          lcrso = lcrso + 1
          itar4(lcrso) = ncroso(neut)
        endif
  537 continue
      ngeom(1,9) = ltmp
      ngeom(2,9) = 0
      if(lcrso .ge. ltmp) ngeom(2,9) = lcrso
      go to 520
c************** end y-hemicylinder ********************
c********** begin z-hemicylinder check ****************
   60 if(ngeom(2,10).eq.0) go to 520
      ltmp = lcrso + 1
      do 529 neut = ngeom(1,10), ngeom(2,10)
        rtar1(neut)=(x1(ncroso(neut))-xx(5,itar1(neut)))**2
     *  + (y1(ncroso(neut))-xx(6,itar1(neut)))**2
        rtar3(neut) = 2*mod(igeom(itar1(neut)),2)-1
        itar2(neut) = (igeom(itar1(neut))-6)/2
  529 continue
      if(lzhcyx) then
      do 530 neut = ngeom(1,10), ngeom(2,10)
        if(itar2(neut) .eq. 1) rtar4(neut) =
     *    rtar3(neut)*(x1(ncroso(neut))-xx(5,itar1(neut)))

  530 continue
      endif
      if(lzhcyy) then
      do 579 neut = ngeom(1,10), ngeom(2,10)
        if(itar2(neut) .eq. 2) rtar4(neut) =
     *    rtar3(neut)*(y1(ncroso(neut))-xx(6,itar1(neut)))
  579 continue
      endif
      do 531 neut = ngeom(1,10), ngeom(2,10)
        if(rtar4(neut).ge.-xx(7,itar1(neut)) .and.
     *     rtar1(neut).le.xx(4,itar1(neut))  .and.
     *     z1(ncroso(neut)).le.xx(2,itar1(neut)).and.
     *     z1(ncroso(neut)).ge.xx(3,itar1(neut))) then
          lcol = lcol + 1
          ncoll(lcol) = ncroso(neut)
        else
          lcrso = lcrso + 1
          itar4(lcrso) = ncroso(neut)
        endif
  531 continue
```

```
      ngeom(1,10) = ltmp
      ngeom(2,10) = 0
      if(lcrso .ge. ltmp) ngeom(2,10) = lcrso
      go to 520
c************** end z-hemicylinder ********************
  520 continue
c********************end posit*************************
c************** begin outward crossing ****************
      if(lcrso .eq. 0) go to 1001
      n = 1
      do 550 neut = 1, lcrso
        ncroso(neut) = itar4(neut)
        ltar1(neut)  = .false.
  550 continue
      if(ltrk)call trkwrt('outward ',wt,tme,u,v,w,x,y,
     *  z,kcore,nbx,nby,nbz,k,ig,ncroso,kr,ki,ll,k1,k2,
     *  now,igeo,nbank,lcrso)
      call cros(ncroso,x,y,z,x1,y1,z1,etausd,k,xx,lcrso,
     *  m,n,igeom,ngeom,jgeom,lgtyp,itar10,itar11,itar12,
     *  rtar10,rtar11,rtar12,rtar13,rtar14,rtar15,rtar16,
     *  rtar17,rtar18,rtar19,rtar20,rtar21,rtar22,rtar23,
     *  rtar24)
cdir$ ivdep
      do 561 neut = 1, lcrso
        if(kr(ncroso(neut)).gt.0) rpth(ncroso(neut)) =
     *    rpth(ncroso(neut))*( 1.0 - etausd(ncroso(neut)))
        tme(ncroso(neut)) = tme(ncroso(neut)) +
     *  pth(ncroso(neut))*etausd(ncroso(neut))*
     *  vinv(ig(ncroso(neut)))
  561 continue
      if(nflx) then
        do 562 neut = 1, lcrso
          flux(ig(ncroso(neut)),iregc(k(ncroso(neut)))) =
     *      flux(ig(ncroso(neut)),iregc(k(ncroso(neut)))) +

     *      wt(ncroso(neut))*pth(ncroso(neut))*
     *      etausd(ncroso(neut))
  562   continue
      endif
      ltemp = lcrso
      lcrso = 0
      lara = lff + 1
cdir$ ivdep
      do 567 neut = 1, ltemp
        if(k(ncroso(neut)).lt.k2(ncroso(neut))) then
          lff = lff + 1
          nff(lff) = ncroso(neut)
          lic(ncroso(neut)) = .false.
          k(ncroso(neut)) = k(ncroso(neut)) + 1
        else
          lcrso = lcrso + 1
          ncroso(lcrso) = ncroso(neut)
```

```
        endif
  567 continue
      if(lmult) then
cdir$ ivdep
        do 569 neut = 1, lcrso
          if(ll(ncroso(neut)) .eq. nsmult(ncroso(neut)))
     *    nsmult(ncroso(neut)) = -nsmult(ncroso(neut))
  569   continue
      endif
      if(lnmhol) then
        do 568 neut = lara, lff
          if(ilh(k(nff(neut))).gt.0) then
            lic(nff(neut)) = .true.
            lhic(nff(neut)) = .true.
          endif
  568   continue
      endif
      if(lcrso.eq.0) go to 1051
c*********************begin hole out*********************
      if(lnmhol) then
        ltemp = lcrso
        lara = lff + 1
        lcrso = 0
        do 901 neut = 1, ltemp
          ltar1(neut) = lhole(ncroso(neut))
          itar1(neut) = ll(ncroso(neut))
          itar2(neut) = lholu(nhole(ncroso(neut)))
  901   continue
        do 902 neut = 1, ltemp
          if(ltar1(neut) .and. itar1(neut).eq.
     *    itar2(neut)) then
            lff = lff + 1
            nff(lff) = ncroso(neut)
            itar3(lff) = nhole(ncroso(neut))
          else
            lcrso = lcrso + 1
            ncroso(lcrso) = ncroso(neut)
          endif
  902   continue
        if(lara .gt. lff) go to 1091
cdir$ ivdep
        do 903 neut = lara, lff
          k(nff(neut)) = khole(itar3(neut))
          lhole(nff(neut)) = .false.
          lsthol(nff(neut)) = nhole(nff(neut))
          lic(nff(neut)) = .false.
          lhic(nff(neut)) = .false.
          x(nff(neut)) = x(nff(neut)) + holx(itar3(neut))
          y(nff(neut)) = y(nff(neut)) + holy(itar3(neut))
          z(nff(neut)) = z(nff(neut)) + holz(itar3(neut))
  903   continue
        do 904 neut = lara, lff
```

```
          ll(nff(neut)) = kboxc(k(nff(neut)))
          nhole(nff(neut)) = 0
  904   continue
        do 905 neut = lara, lff
          k1(nff(neut)) = kbnds1(ll(nff(neut)))
          k2(nff(neut)) = kbnds2(ll(nff(neut)))
  905   continue
        do 908 neut = lara, lff
          if(k(nff(neut)).gt.k1(nff(neut)) .or.
     *    ilh(k(nff(neut))).gt.0) lic(nff(neut)) = .true.
          if(k(nff(neut)).le.k1(nff(neut)))
     *    lhic(nff(neut)) = .true.
  908   continue
        if(nesth) then
          do 906 neut = lara, lff
            itar1(neut) = nholin(nff(neut))
  906     continue
cdir$ ivdep
          do 907 neut = lara, lff
            if(itar1(neut).gt.0) then
              nholin(nff(neut)) = nholin(nff(neut)) - 1
              lhole(nff(neut)) = .true.
              nhole(nff(neut)) =
     *        stackh(nff(neut),itar1(neut))
            endif
  907     continue
        endif
      endif
c**********************end hole out*********************
c**********************begin out array*****************
 1091 if(lnmara) then
        ltemp = lcrso
        lcrso = 0
        lara = 0
        do 911 neut = 1, ltemp
          if(larray(ncroso(neut))) then
            lara = lara + 1
            itar1(lara) = ncroso(neut)
          else
            lcrso = lcrso + 1
            ncroso(lcrso) = ncroso(neut)
          endif
  911   continue
        if(lara.eq.0) go to 800
        do 912 neut = 1, lara
          itar2(neut) = k(itar1(neut))
          itar4(neut) = 0
          itar5(neut) = 0
          itar6(neut) = 0
          ltar1(neut) = .false.
  912   continue
      if(ltrk)call trkwrt('array   ',wt,tme,u,v,w,x,y,
```

```
     *  z,kcore,nbx,nby,nbz,k,ig,itar1,kr,ki,ll,k1,k2,
     *  now,igeo,nbank,lara)
c**********determine particle crossing face
cdir$ ivdep
        do 921 neut = 1, lara
          if(x(itar1(neut)).eq.xx(1,itar2(neut)) .and.
     *       u(itar1(neut)).gt.0.0) then
            itar4(neut) = 2
            nbx(itar1(neut)) = nbx(itar1(neut)) + 1
          endif
  921   continue
cdir$ ivdep
        do 922 neut = 1, lara
          if(x(itar1(neut)).eq.xx(2,itar2(neut)) .and.
     *       u(itar1(neut)).lt.0.0) then
            itar4(neut) = 1
            nbx(itar1(neut)) = nbx(itar1(neut)) - 1
          endif
  922   continue
cdir$ ivdep
        do 923 neut = 1, lara
          if(y(itar1(neut)).eq.xx(3,itar2(neut)) .and.
     *       v(itar1(neut)).gt.0.0) then
            itar5(neut) = 4
            nby(itar1(neut)) = nby(itar1(neut)) + 1
          endif
  923   continue
cdir$ ivdep
        do 924 neut = 1, lara
          if(y(itar1(neut)).eq.xx(4,itar2(neut)) .and.
     *       v(itar1(neut)).lt.0.0) then
            itar5(neut) = 3
            nby(itar1(neut)) = nby(itar1(neut)) - 1
          endif
  924   continue
cdir$ ivdep
        do 925 neut = 1, lara
          if(z(itar1(neut)).eq.xx(5,itar2(neut)) .and.
     *       w(itar1(neut)).gt.0.0) then
            itar6(neut) = 6
            nbz(itar1(neut)) = nbz(itar1(neut)) + 1
          endif
  925   continue
cdir$ ivdep
        do 926 neut = 1, lara
          if(z(itar1(neut)).eq.xx(6,itar2(neut)) .and.
     *       w(itar1(neut)).lt.0.0) then
            itar6(neut) = 5
            nbz(itar1(neut)) = nbz(itar1(neut)) - 1
          endif
  926   continue
        do 927 neut = 1, lara
```

```
          if(nbx(itar1(neut)).gt.nbxmx(now(itar1(neut)))
     *    .or.nbx(itar1(neut)).le.0) ltar1(neut) = .true.
          if(nby(itar1(neut)).gt.nbymx(now(itar1(neut)))
     *    .or.nby(itar1(neut)).le.0) ltar1(neut) = .true.
          if(nbz(itar1(neut)).gt.nbzmx(now(itar1(neut)))
     *    .or.nbz(itar1(neut)).le.0) ltar1(neut) = .true.
  927   continue
        ltemp = lara
        lara = 0
        lout = 0
        do 928 neut = 1, ltemp
          if(ltar1(neut)) then
            lout = lout + 1
            itar8(lout) = itar1(neut)
          else
            lff = lff + 1
            lara = lara + 1
            nff(lff)    = itar1(neut)
            itar7(lara) = itar1(neut)
            itar4(lara) = itar4(neut)
            itar5(lara) = itar5(neut)
            itar6(lara) = itar6(neut)
            lic(itar1(neut)) = .true.
          endif
  928   continue
        if(lara.eq.0) go to 940
c**********shift coordinate system of neutrons in array
        if(mbox) then
          do 929 neut = 1, lara
            itar2(neut) = now(itar7(neut))
  929     continue
          do 930 neut = 1, lara
            itar3(neut) = nbx(itar7(neut)) - 1 +
     *      nbxmx(itar2(neut))*(nby(itar7(neut)) - 1 +
     *      nbymx(itar2(neut))*(nbz(itar7(neut)) - 1))
  930     continue
          do 931 neut = 1, lara
            itar2(neut) = irega(itar2(neut))
  931     continue
          do 932 neut = 1, lara
            itar1(neut) = itar3(neut)+ndxlba(itar2(neut))
  932     continue
          do 933 neut = 1, lara
            itar3(neut) = lba(itar1(neut))
            itar2(neut) = k(itar7(neut))
  933     continue
          do 934 neut = 1, lara
            ll(itar7(neut)) = itar3(neut)
            k1(itar7(neut)) = kbnds1(itar3(neut))
            k2(itar7(neut)) = kbnds2(itar3(neut))
            k(itar7(neut))  = kbnds2(itar3(neut))
  934     continue
```

```
          do 935 neut = 1, lara
            itar3(neut) = k(itar7(neut))
  935     continue
cdir$ ivdep
          do 936 neut = 1, lara
            x(itar7(neut)) = x(itar7(neut)) -
     *      (xx(2,itar2(neut)) - xx(2,itar3(neut)))
            y(itar7(neut)) = y(itar7(neut)) -
     *      (xx(4,itar2(neut)) - xx(4,itar3(neut)))
            z(itar7(neut)) = z(itar7(neut)) -
     *      (xx(6,itar2(neut)) - xx(6,itar3(neut)))
  936     continue
        else
          do 937 neut = 1, lara
           itar3(neut) = k(itar7(neut))
  937     continue
        endif
        do 938 neut = 1, lara
          if(itar4(neut).gt.0) x(itar7(neut)) =
     *    xx(itar4(neut),itar3(neut))
          if(itar5(neut).gt.0) y(itar7(neut)) =
     *    xx(itar5(neut),itar3(neut))
          if(itar6(neut).gt.0) z(itar7(neut)) =
     *    xx(itar6(neut),itar3(neut))
  938   continue
        do 939 neut = 1, lara
          if(k(itar7(neut)).le.k1(itar7(neut)) .and.
     *    igeom(k(itar7(neut))).ne.-1)
     *    lic(itar7(neut)) = .false.
  939   continue
        if(lnmhol) then
cdir$ ivdep
          do 950 neut = 1, lara
            if(lic(itar7(neut))) then
              lhic(itar7(neut)) = .false.
            else
              lhic(itar7(neut)) = .true.
            endif
  950     continue
          do 945 neut = 1, lara
            if(ilh(itar3(neut)).gt.0)
     *      lic(itar7(neut)) = .true.
  945     continue
        endif
c**********process neutrons exiting the array
  940   if(lout.eq.0) go to 800
        lara = 0
        do 941 neut = 1, lout
          if(now(itar8(neut)).eq.nglobl .and.
     *    .not.exrfl) then
            lcrso = lcrso + 1
            ncroso(lcrso) = itar8(neut)
```

```
          else
            lff = lff + 1
            lara = lara + 1
            nff(lff) = itar8(neut)
            itar1(lara) = itar8(neut)
          endif
  941   continue
        if(lara.eq.0) go to 800
        do 942 neut = 1, lara
          lic(itar1(neut)) = .false.
          itar2(neut) = kcore(itar1(neut))
          itar3(neut) = now(itar1(neut))
          itar6(neut) = k(itar1(neut))
  942   continue
        do 943 neut = 1, lara
          itar4(neut) = irega(itar3(neut))
          k(itar1(neut)) = itar2(neut)
          ll(itar1(neut)) = kboxc(itar2(neut))
          itar5(neut)     = kboxc(itar2(neut))
  943   continue
        do 944 neut = 1, lara
          k1(itar1(neut)) = kbnds1(itar5(neut))
          k2(itar1(neut)) = kbnds2(itar5(neut))
  944   continue
cdir$ ivdep
        do 946 neut = 1, lara
          if(nbx(itar1(neut)).lt.1) then
            x(itar1(neut)) = xx(2,itar2(neut))
          else if(nbx(itar1(neut)).gt.
     *    nbxmx(now(itar1(neut)))) then
            x(itar1(neut)) = xx(1,itar2(neut))
          else
            x(itar1(neut)) = x(itar1(neut)) -
     *      xx(2,itar6(neut)) + deltx(nbx(itar1(neut)) +
     *      indxx(itar4(neut))) + xx(2,itar2(neut))
          endif
  946   continue
cdir$ ivdep
        do 947 neut = 1, lara
          if(nby(itar1(neut)).lt.1) then
            y(itar1(neut)) = xx(4,itar2(neut))
          else if(nby(itar1(neut)).gt.
     *    nbymx(now(itar1(neut)))) then
            y(itar1(neut)) = xx(3,itar2(neut))
          else
            y(itar1(neut)) = y(itar1(neut)) -
     *      xx(4,itar6(neut)) + delty(nby(itar1(neut)) +
     *      indxy(itar4(neut))) + xx(4,itar2(neut))
          endif
  947   continue
cdir$ ivdep
        do 948 neut = 1, lara
```

```
          if(nbz(itar1(neut)).lt.1) then
            z(itar1(neut)) = xx(6,itar2(neut))
          else if(nbz(itar1(neut)).gt.
     *    nbzmx(now(itar1(neut)))) then
            z(itar1(neut)) = xx(5,itar2(neut))
          else
            z(itar1(neut)) = z(itar1(neut)) -
     *      xx(6,itar6(neut)) + deltz(nbz(itar1(neut)) +
     *      indxz(itar4(neut))) + xx(6,itar2(neut))
          endif
  948   continue
        if(nesta) then
        do 952 neut = 1, lara
          itar3(neut) = narrin(itar1(neut))
  952   continue
cdir$ ivdep
        do 953 neut = 1, lara
          if(itar3(neut).gt.0) then
            narrin(itar1(neut)) = narrin(itar1(neut)) - 4
            kcore(itar1(neut))  =
     *      stacka(itar1(neut),itar3(neut))
            nbx(itar1(neut))    =
     *      stacka(itar1(neut),itar3(neut)+1)
            nby(itar1(neut))    =
     *      stacka(itar1(neut),itar3(neut)+2)
            nbz(itar1(neut))    =
     *     stacka(itar1(neut),itar3(neut)+3)
            now(itar1(neut))    = nglobl
          else
            larray(itar1(neut)) = .false.
            kcore(itar1(neut))  = 0
            nbx(itar1(neut))    = 0
            now(itar1(neut))    = 0
          endif
  953   continue
        do 954 neut = 1, lara
         if(kcore(itar1(neut)).ne.0) now(itar1(neut)) =
     *     mat(kcore(itar1(neut)))
  954   continue
        else
        do 949 neut = 1, lara
          kcore(itar1(neut)) = 0
          nbx(itar1(neut))   = 0
          now(itar1(neut))   = 0
          larray(itar1(neut)) = .false.
  949   continue
        endif
      endif
c************************end out array*****************
c*********************albedo treatment*****************
  800 if(lnxx) then
        do 810 neut = 1, lcrso
```

```
          logper(neut) = .false.
  810   continue
      if(ltrk)call trkwrt('albedo  ',wt,tme,u,v,w,x,y,
     *  z,kcore,nbx,nby,nbz,k,ig,ncroso,kr,ki,ll,k1,k2,
     *  now,igeo,nbank,lcrso)
c*******periodic albedo treatment
        if(lper) then
          if(intr(1).eq.2) then
cdir$ ivdep
            do 811 neut = 1, lcrso
              if(x(ncroso(neut)).eq.xx(1,k2(ncroso(neut)))
     *        .and. u(ncroso(neut)).gt.0.0) then
                logper(neut) = .true.
                nbx(ncroso(neut)) = nbxl(1)
                x(ncroso(neut)) = xx(2,k2(ncroso(neut)))
              endif
  811       continue
          endif
          if(intr(2).eq.2) then
cdir$ ivdep
            do 812 neut = 1, lcrso
              if(x(ncroso(neut)).eq.xx(2,k2(ncroso(neut)))
     *        .and. u(ncroso(neut)).lt.0.0) then
                logper(neut) = .true.
                nbx(ncroso(neut)) = nbxl(2)
                x(ncroso(neut)) = xx(1,k2(ncroso(neut)))
              endif
  812       continue
          endif
          if(intr(3).eq.2) then
cdir$ ivdep
            do 813 neut = 1, lcrso
              if(y(ncroso(neut)).eq.xx(3,k2(ncroso(neut)))
     *        .and. v(ncroso(neut)).gt.0.0) then
                logper(neut) = .true.
                nbx(ncroso(neut)) = nbxl(3)
                y(ncroso(neut)) = xx(4,k2(ncroso(neut)))
              endif
  813       continue
          endif
          if(intr(4).eq.2) then
cdir$ ivdep
            do 814 neut = 1, lcrso
              if(y(ncroso(neut)).eq.xx(4,k2(ncroso(neut)))
     *        .and. v(ncroso(neut)).lt.0.0) then
                logper(neut) = .true.
                nbx(ncroso(neut)) = nbxl(4)
                y(ncroso(neut)) = xx(3,k2(ncroso(neut)))
              endif
  814       continue
          endif
          if(intr(5).eq.2) then
```

```
cdir$ ivdep
            do 815 neut = 1, lcrso
              if(z(ncroso(neut)).eq.xx(5,k2(ncroso(neut)))
     *        .and. w(ncroso(neut)).gt.0.0) then
                logper(neut) = .true.
                nbx(ncroso(neut)) = nbx1(5)
                z(ncroso(neut)) = xx(6,k2(ncroso(neut)))
              endif
  815       continue
          endif
          if(intr(6).eq.2) then
cdir$ ivdep
            do 816 neut = 1, lcrso
              if(z(ncroso(neut)).eq.xx(6,k2(ncroso(neut)))
     *        .and. w(ncroso(neut)).lt.0.0) then
                logper(neut) = .true.
                nbx(ncroso(neut)) = nbx1(6)
                z(ncroso(neut)) = xx(5,k2(ncroso(neut)))
              endif
  816       continue
          endif
          if(.not.exrfl) then
            if(mbox) then
              do 817 neut = 1,lcrso
                if(logper(neut)) ll(ncroso(neut)) =
     *          lba(ndxlba(irega(now(ncroso(neut)))) - 1 +
     *          nbx(ncroso(neut))+nbxmx(now(ncroso(neut)))*
     *          (nby(ncroso(neut)) - 1 +
     *          nbymx(now(ncroso(neut)))*
     *          (nbz(ncroso(neut)) - 1)))
  817         continue
            endif
          endif
          do 819 neut = 1, lcrso
            if(logper(neut)) then
              k1(ncroso(neut)) = kbnds1(ll(ncroso(neut)))
              k2(ncroso(neut)) = kbnds2(ll(ncroso(neut)))
            endif
  819     continue
          do 827 neut = 1, lcrso
            itar5(neut) = k(ncroso(neut))
            itar6(neut) = k2(ncroso(neut))
  827     continue
cdir$ ivdep
          do 828 neut = 1, lcrso
            if(logper(neut)) then
              x(ncroso(neut)) = xx(2,itar6(neut)) +
     *          (x(ncroso(neut)) - xx(2,itar5(neut)))
              y(ncroso(neut)) = xx(4,itar6(neut)) +
     *          (y(ncroso(neut)) - xx(4,itar5(neut)))
              z(ncroso(neut)) = xx(6,itar6(neut)) +
     *          (z(ncroso(neut)) - xx(6,itar5(neut)))
```

```
            endif
  828     continue
          do 829 neut = 1, lcrso
            if(logper(neut)) k(ncroso(neut)) = itar5(neut)
  829     continue
        endif
c*******mirror albedo treatment
        if(lmir) then
          if(intr(1).eq.1) then
cdir$ ivdep
            do 821 neut = 1, lcrso
              if(x(ncroso(neut)).eq.
     *        xx(1,k2(ncroso(neut)))) then
                logper(neut) = .true.
                u(ncroso(neut)) = -u(ncroso(neut))
                nbx(ncroso(neut)) = nbxl(2)
              endif
  821       continue
          endif
          if(intr(2).eq.1) then
cdir$ ivdep
            do 822 neut = 1, lcrso
              if(x(ncroso(neut)).eq.
     *        xx(2,k2(ncroso(neut)))) then
                logper(neut) = .true.
                u(ncroso(neut)) = -u(ncroso(neut))
                nbx(ncroso(neut)) = nbxl(1)
              endif
  822       continue
          endif
          if(intr(3).eq.1) then
cdir$ ivdep
            do 823 neut = 1, lcrso
              if(y(ncroso(neut)).eq.
     *        xx(3,k2(ncroso(neut)))) then
                logper(neut) = .true.
                v(ncroso(neut)) = -v(ncroso(neut))
                nby(ncroso(neut)) = nbxl(4)
              endif
  823       continue
          endif
          if(intr(4).eq.1) then
cdir$ ivdep
            do 824 neut = 1, lcrso
              if(y(ncroso(neut)).eq.
     *        xx(4,k2(ncroso(neut)))) then
                logper(neut) = .true.
                v(ncroso(neut)) = -v(ncroso(neut))
                nby(ncroso(neut)) = nbxl(3)
              endif
  824       continue
          endif
```

```
          if(intr(5).eq.1) then
cdir$ ivdep
            do 825 neut = 1, lcrso
              if(z(ncroso(neut)).eq.
     *        xx(5,k2(ncroso(neut)))) then
                logper(neut) = .true.
                w(ncroso(neut)) = -w(ncroso(neut))
                nbz(ncroso(neut)) = nbxl(6)
              endif
  825       continue
          endif
          if(intr(6).eq.1) then
cdir$ ivdep
            do 826 neut = 1, lcrso
              if(z(ncroso(neut)).eq.
     *        xx(6,k2(ncroso(neut)))) then
                logper(neut) = .true.
                w(ncroso(neut)) = -w(ncroso(neut))
                nbz(ncroso(neut)) = nbxl(5)
              endif
  826       continue
          endif
        endif
c*******differential albedo treatment
        if(ldif) then
          if(intr(1).eq.3) then
            do 831 neut = 1, lcrso
              if(x(ncroso(neut)).eq.
     *        xx(1,k2(ncroso(neut)))) then
                it = ncroso(neut)
                iface = 1
                logper(neut) = .true.
                call albedo(mal,alb,a,plim,cpol,spol,
     *          mwxsc,mwalb,probx,proba,ngp,iabsg,u(it),
     *          v(it),w(it),wt(it),ig(it))
                nbx(ncroso(neut)) = nbxl(2)
                rpth(ncroso(neut)) = 0.0
              endif
  831       continue
          endif
          if(intr(2).eq.3) then
            do 832 neut = 1, lcrso
              if(x(ncroso(neut)).eq.
     *        xx(2,k2(ncroso(neut)))) then
                it = ncroso(neut)
                iface = 2
                logper(neut) = .true.
                call albedo(mal,alb,a,plim,cpol,spol,
     *          mwxsc,mwalb,probx,proba,ngp,iabsg,u(it),
     *          v(it),w(it),wt(it),ig(it))
                nbx(ncroso(neut)) = nbxl(1)
                rpth(ncroso(neut)) = 0.0
```

```
            endif
832       continue
          endif
          if(intr(3).eq.3) then
            do 833 neut = 1, lcrso
              if(y(ncroso(neut)).eq.
     *        xx(3,k2(ncroso(neut)))) then
                it = ncroso(neut)
                iface = 3
                logper(neut) = .true.
                call albedo(mal,alb,a,plim,cpol,spol,
     *          mwxsc,mwalb,probx,proba,ngp,iabsg,
     *          v(it),w(it),u(it),wt(it),ig(it))
                nby(ncroso(neut)) = nbxl(4)
                rpth(ncroso(neut)) = 0.0
              endif
833       continue
          endif
          if(intr(4).eq.3) then
            do 834 neut = 1, lcrso
              if(y(ncroso(neut)).eq.
     *        xx(4,k2(ncroso(neut)))) then
                it = ncroso(neut)
                iface = 4
                logper(neut) = .true.
                call albedo(mal,alb,a,plim,cpol,spol,
     *          mwxsc,mwalb,probx,proba,ngp,iabsg,
     *          v(it),w(it),u(it),wt(it),ig(it))
                nby(ncroso(neut)) = nbxl(3)
                rpth(ncroso(neut)) = 0.0
              endif
834       continue
          endif
          if(intr(5).eq.3) then
            do 835 neut = 1, lcrso
              if(z(ncroso(neut)).eq.
     *        xx(5,k2(ncroso(neut)))) then
                it = ncroso(neut)
                iface = 5
                logper(neut) = .true.
                call albedo(mal,alb,a,plim,cpol,spol,
     *          mwxsc,mwalb,probx,proba,ngp,iabsg,
     *          w(it),u(it),v(it),wt(it),ig(it))
                nbz(ncroso(neut)) = nbxl(6)
                rpth(ncroso(neut)) = 0.0
              endif
835       continue
          endif
          if(intr(6).eq.3) then
            do 836 neut = 1, lcrso
              if(z(ncroso(neut)).eq.
     *        xx(6,k2(ncroso(neut)))) then
```

```
                it = ncroso(neut)
                iface = 6
                logper(neut) = .true.
                call albedo(mal,alb,a,plim,cpol,spol,
     *          mwxsc,mwalb,probx,proba,ngp,iabsg,
     *          w(it),u(it),v(it),wt(it),ig(it))
                nbz(ncroso(neut)) = nbxl(5)
                rpth(ncroso(neut)) = 0.0
              endif
  836       continue
          endif
        endif
        ltemp = lcrso
        lcrso = 0
        lara = lff + 1
        do 841 neut = 1, ltemp
          if(logper(neut)) then
            lff = lff + 1
            nff(lff) = ncroso(neut)
          else
            lcrso = lcrso + 1
            ncroso(lcrso) = ncroso(neut)
          endif
  841   continue
        if(lff.ge.lara) then
          do 842 neut = lara, lff
            if(k(nff(neut)).gt.k1(nff(neut)) .or.
     *      igeom(k(nff(neut))).lt.0)
     *      lic(nff(neut)) = .true.
  842     continue
        endif
      endif
c*******************end albedo treatment***************
      do 583 neut = 1, lcrso
        lkil = lkil + 1
        nkill(lkil) = ncroso(neut)
        tlf = tlf + wt(ncroso(neut))*tme(ncroso(neut))
  583 continue
      if(igen .gt. nskip) then
        do 584 neut = 1, lcrso
          itar4(neut) = ig(ncroso(neut))
  584   continue
        do 574 neut = 1, lcrso
          fleak(itar4(neut)) = fleak(itar4(neut)) +
wt(ncroso(neut))
  574   continue
      endif
 1051 do 582 neut = 1, nbank
        ncroso(neut) = 0
        m(neut)      = 0
  582 continue
      lcrso = 0
```

```
      go to 1001
c****************** end outward crossing **************
c********************** end crossing ******************
c
c*********************start collision******************
 1006 do 601 neut = 1, lcol
        itar1(neut) = ig(ncoll(neut))
        itar2(neut) = kr(ncoll(neut))
  601 continue
c     if(igen.eq.1) then
c       write(outpt,10006)
c       do 691 neut = 1, lcol
c         ii = ncoll(neut)
c         write(outpt,10000)ii,x(ii),y(ii),z(ii),u(ii),
c    *    v(ii),w(ii),k(ii),ll(ii),kcore(ii),nbx(ii),
c    *    nby(ii),nbz(ii),nhole(ii)
c 691   continue
c     endif
      do 602 neut = 1, lcol
        x(ncoll(neut)) = x1(ncoll(neut))
        y(ncoll(neut)) = y1(ncoll(neut))
        z(ncoll(neut)) = z1(ncoll(neut))
  602 continue
      if(nflx) then
        do 631 neut = 1, lcol
          flux(ig(ncoll(neut)),iregc(k(ncoll(neut)))) =
     *      flux(ig(ncoll(neut)),iregc(k(ncoll(neut)))) +
     *      wt(ncoll(neut))*pth(ncoll(neut))
  631   continue
      endif
      do 603 neut = 1, lcol
        rpth(ncoll(neut))   = 0.0
        rtar15(neut) = wt(ncoll(neut))*
     *                 fap(itar1(neut),itar2(neut))
        fisw(neut) = wt(ncoll(neut))*
     *               fnfp(itar1(neut),itar2(neut))
  603 continue
cdir$ ivdep
      do 604 neut = 1, lcol
        wt(ncoll(neut)) = wt(ncoll(neut))*
     *                    fnap(itar1(neut),itar2(neut))
        tme(ncoll(neut)) = tme(ncoll(neut)) +
     *                pth(ncoll(neut))*vinv(itar1(neut))
  604 continue
      do 605 neut = 1, lcol
        tlf    = tlf+rtar15(neut)*tme(ncoll(neut))
        tofis  = tofis+fisw(neut)
        age    = age+fisw(neut)*tme(ncoll(neut))
  605 continue
      if(ltrk)call trkwrt('collison',wt,tme,u,v,w,x,y,
     *  z,kcore,nbx,nby,nbz,k,ig,ncoll,kr,ki,ll,k1,k2,
     *  now,igeo,nbank,lcol)
```

```
    if(lmult) then
      do 679 neut = 1, lcol
        if(nsmult(ncoll(neut)) .gt. 0) tself = tself +
   *    fisw(neut)
679   continue
    endif
    if(matrix) then
      if(lunit) then
        do 680 neut = 1, lcol
          tu(lls(ncoll(neut)),ll(ncoll(neut))) =
   *      tu(lls(ncoll(neut)),ll(ncoll(neut)))+fisw(neut)
680     continue
      endif
      if(lmhole) then
        do 681 neut = 1, lcol
          itar5(neut) = nhole(ncoll(neut))
681     continue
        if(lhhgh .and. nesth) then
          do 682 neut = 1, lcol
            if(nholin(ncoll(neut)).gt.0) itar5(neut) =
   *        stackh(ncoll(neut),1)
682       continue
        endif
        do 683 neut = 1, lcol
          if(lhole(ncoll(neut)) .and.
   *      nhl(ncoll(neut)).gt.0)
   *        th(nhl(ncoll(neut)),itar5(neut)) =
   *        th(nhl(ncoll(neut)),itar5(neut)) + fisw(neut)
683     continue
      endif
      if(lmarry) then
        do 684 neut = 1, lcol
          itar5(neut) = irega(now(ncoll(neut)))
684     continue
        if(lahgh .and. nesta) then
          do 685 neut = 1, lcol
            if(narrin(ncoll(neut)).gt.0) itar5(neut) =
   *        irega(mat(stacka(ncoll(neut),1)))
685       continue
        endif
        do 686 neut = 1, lcol
          if(larray(ncoll(neut)) .and.
   *      nar(ncoll(neut)).gt.0)
   *      ta(nar(ncoll(neut)),itar5(neut)) =
   *      ta(nar(ncoll(neut)),itar5(neut)) + fisw(neut)
686    continue
      endif
      if(larpos) then
        do 687 neut = 1, lcol
          itar5(neut) = now(ncoll(neut))
          itar6(neut) = nbx(ncoll(neut))
          itar7(neut) = nby(ncoll(neut))
```

```
          itar8(neut) = nbz(ncoll(neut))
687       continue
          if(nesta) then
            do 688 neut = 1, lcol
              if(narrin(ncoll(neut)) .gt. 0) then
                itar5(neut) = mat(stacka(ncoll(neut),1))
                itar6(neut) = stacka(ncoll(neut),2)
                itar7(neut) = stacka(ncoll(neut),3)
                itar8(neut) = stacka(ncoll(neut),4)
              endif
688         continue
          endif
          do 689 neut = 1, lcol
            if(nglobl .eq. itar5(neut)) then
              itar9(neut) = nbxgbl*nbygbl*(itar8(neut)-1) +
     *        nbxgbl*(itar7(neut)-1) + itar6(neut)
            else
              itar9(neut) = matdim
            endif
689       continue
          do 690 neut = 1, lcol
            tp(kft(ncoll(neut)),itar9(neut)) =
     *      tp(kft(ncoll(neut)),itar9(neut)) + fisw(neut)
690       continue
        endif
      endif
      if (newbar) then
        do 629 neut = 1, lcol
          rtar14(neut) = wt(ncoll(neut))*
     *    x1d(itar1(neut),itar2(neut),indxfs)
629     continue
        do 630 neut = 1, lcol
          rtar13(neut) = ig(ncoll(neut))*rtar14(neut)
630     continue
        do 606 neut = 1, lcol
          tfisn  = tfisn + rtar14(neut)
          gfis   = gfis + rtar13(neut)
606     continue
      end if
      if (igen.le.nskip) go to 1060
      if (lfa) then
        do 607 neut = 1, lcol
          itar13(neut) = iregc(k(ncoll(neut)))
607     continue
      else
        do 608 neut = 1, lcol
          itar13(neut) = 1
608     continue
      endif
      do 609 neut = 1, lcol
        fmabs(itar1(neut),itar13(neut)) =
     *  fmabs(itar1(neut),itar13(neut)) + rtar15(neut)
```

```
        fmfis(itar1(neut),itar13(neut)) =
     *  fmfis(itar1(neut),itar13(neut)) + fisw(neut)
  609 continue
      if (nfden) then
        do 611 neut = 1, lcol
          fisden(iregc(k(ncoll(neut))),1) =
     *    fisden(iregc(k(ncoll(neut))),1) + fisw(neut)
  611   continue
      endif
 1060 continue
c******calculate a new energy and scattering angle*****
      do 612 neut = 1, lcol
        itar12(neut) = mwa(1,itar1(neut),itar2(neut)) +
     *  ranf()*mwa(3,itar1(neut),itar2(neut))
  612 continue
      do 613 neut = 1, lcol
        rtar16(neut) = twodee(itar12(neut),1)
  613 continue
      do 614 neut = 1, lcol
        if(ranf() .le. rtar16(neut)) then
          rtar18(neut) = twodee(itar12(neut),2)
        else
          rtar18(neut) = twodee(itar12(neut),3)
        endif
  614 continue
c******itar13 is the new energy group for the particle
      do 615 neut = 1, lcol
        itar13(neut) = abs(rtar18(neut))
  615 continue
c*****check for anisotropic scattering*****************
      do 616 neut = 1, lcol
        rtar1(neut) = twopi*ranf()
        rtar21(neut) = 0.0
        rtar20(neut) = 0.0
  616 continue
      do 617 neut = 1, lcol
        rtar2(neut) = 2.0*ranf() - 1.0
        rtar3(neut) = sin(rtar1(neut))
        rtar4(neut) = cos(rtar1(neut))
  617 continue
      do 618 neut = 1, lcol
      if (rtar18(neut) .lt. 0.0) then
        rtar1(neut)    = sqrt(1.0-rtar2(neut)**2)
        u(ncoll(neut)) = rtar4(neut)*rtar1(neut)
        v(ncoll(neut)) = rtar3(neut)*rtar1(neut)
        w(ncoll(neut)) = rtar2(neut)
      else
        rtar21(neut) = rtar3(neut)
        rtar20(neut) = rtar4(neut)
      endif
  618 continue
      do 619 neut = 1, lcol
```

```
        rtar17(neut) = 2.0*(rtar18(neut)-itar13(neut))-1.0
  619 continue
      if (nsct1 .eq. 0) then
        do 620 neut = 1, lcol
          rtar1(neut) = 3.0 * rtar17(neut)
          r(neut) = 2.0*ranf() - 1.0
  620   continue
        do 621 neut = 1, lcol
          if(abs(rtar1(neut)) .le. 1.0) then
            rtar17(neut) = (-1.0 +
sqrt(1.0+2.0*rtar1(neut)*
     *      r(neut)+rtar1(neut)**2))/rtar1(neut)
          else
            rtar17(neut) = rtar17(neut)+r(neut)*
     *      (1.0-abs(rtar17(neut)))
          endif
  621   continue
      end if
      do 622 neut = 1, lcol
        if(rtar18(neut) .lt. 0.0) rtar17(neut) = 1.0
        rtar12(neut) = sqrt(1.0-rtar17(neut)*rtar17(neut))
        rtar24(neut) = sqrt(v(ncoll(neut))**2 +
     *                 w(ncoll(neut))**2)
  622 continue
      do 623 neut = 1, lcol
        if(rtar24(neut).le.0.0) rtar23(neut) = 1.0
        if(rtar24(neut).le.0.0) rtar22(neut) = 0.0
        if(rtar24(neut).gt.0.0) rtar23(neut) =
     *    v(ncoll(neut))/rtar24(neut)
        if(rtar24(neut).gt.0.0) rtar22(neut) =
     *    w(ncoll(neut))/rtar24(neut)
  623 continue
      do 624 neut = 1, lcol
        rtar19(neut) = u(ncoll(neut))*rtar20(neut)*
     *                 rtar12(neut)
  624 continue
      do 625 neut = 1, lcol
        rtar1(neut) = u(ncoll(neut))
        rtar2(neut) = v(ncoll(neut))
        rtar3(neut) = w(ncoll(neut))
        itar1(neut) = k(ncoll(neut))
  625 continue
      do 626 neut = 1, lcol
        v(ncoll(neut)) = rtar2(neut)*rtar17(neut) +
     *  rtar23(neut)*rtar19(neut) - rtar22(neut)*
     *  rtar12(neut)*rtar21(neut)
        w(ncoll(neut)) = rtar3(neut)*rtar17(neut) +
     *  rtar22(neut)*rtar19(neut) + rtar23(neut)*
     *  rtar12(neut)*rtar21(neut)
        u(ncoll(neut)) = rtar1(neut)*rtar17(neut) -
     *  rtar20(neut)*rtar12(neut)*rtar24(neut)
  626 continue
```

```
      do 627 neut = 1, lcol
        lff = lff + 1
        nff(lff) = ncoll(neut)
        ig(ncoll(neut)) = itar13(neut)
  627 continue
      do 628 neut = 1, lcol
        if(itar1(neut).gt.k1(ncoll(neut)) .or.
igeom(itar1(neut)).lt.0)
     *    lic(ncoll(neut)) = .true.
        if(itar1(neut).le.k1(ncoll(neut)) .and.
ilh(itar1(neut)).gt.0)
     *    lhic(ncoll(neut)) = .true.
  628 continue
c**********************end collision*********************
c
c******************** start fission ********************
c***** this section loads the fission bank
      lgtl1 = 1
 1067 lgtl = lgtl1 - 1
      do 641 neut = lgtl1, lcol
        r(neut) = ranf()
  641 continue
      do 642 neut = lgtl1, lcol
        fwr(neut) = fisw(neut)/r(neut)
  642 continue
      do 643 neut = lgtl1, lcol
        if(fwr(neut).gt.rakbar) then
          lgtl = lgtl + 1
          ncoll(lgtl) = ncoll(neut)
          fisw(lgtl)  = fisw(neut)
          fwr(lgtl)   = fwr(neut)
          r(lgtl)     = r(neut)
        endif
  643 continue
      if (lgtl .eq. 0) go to 1200
      lcol = lgtl
      do 644 neut = lgtl1, lcol
        if(fisw(neut).gt.rakbar) then
          lgtl = lgtl + 1
          fwr(neut) = rakbar/r(neut)
          ncoll(lgtl) = ncoll(neut)
          fisw(lgtl) = fisw(neut) - rakbar
        endif
  644 continue
      lgtl1 = lcol + 1
      lcol  = lgtl
      if(lcol .ge. lgtl1) go to 1067
      if (lcol .gt. nfbnk-nfiss) lgtl = nfbnk - nfiss
      if (nfiss .eq. nfbnk) go to 1071
      do 645 neut = 1, lgtl
        fwrr(nfiss+neut) = fwr(neut)
        rfbank(nfiss+neut,1) = x(ncoll(neut))
```

```
        rfbank(nfiss+neut,2) = y(ncoll(neut))
        rfbank(nfiss+neut,3) = z(ncoll(neut))
        ifbank(nfiss+neut,1) = kcore(ncoll(neut))
        ifbank(nfiss+neut,2) = nbx(ncoll(neut))
        ifbank(nfiss+neut,3) = nby(ncoll(neut))
        ifbank(nfiss+neut,4) = nbz(ncoll(neut))
        ifbank(nfiss+neut,5) = k(ncoll(neut))
        ifbank(nfiss+neut,6) = ig(ncoll(neut))
  645 continue
      ij = 7
      if(numhol.gt.0) then
        do 646 neut = 1, lgtl
          ifbank(nfiss+neut,ij) = nhole(ncoll(neut))
  646   continue
        ij = ij + 1
      endif
      if(nesta) then
        do 647 neut = 1, lgtl
          ifbank(nfiss+neut,ij) = narrin(ncoll(neut))
  647   continue
        do 648 ik = 1, lnstka
          do 649 neut = 1, lgtl
            ifbank(nfiss+neut,ij+ik) =
     *      stacka(ncoll(neut),ik)
  649     continue
  648   continue
        ij = ij + 1 + lnstka
      endif
      if(nesth) then
        do 650 neut = 1, lgtl
          ifbank(nfiss+neut,ij) = nholin(ncoll(neut))
  650   continue
        do 651 ik = 1, lnstkh
          do 652 neut = 1, lgtl
            ifbank(nfiss+neut,ij+ik) =
     *      stackh(ncoll(neut),ik)
  652     continue
  651   continue
      endif
      nfiss = nfiss + lgtl
      if(lgtl .eq. lcol) go to 1200
 1071 do 658 neut = lgtl + 1, lcol
        ifl = 1
        rlow = fwrr(1)
        do 659 ijk = 2, nfiss
          if(fwrr(ijk) .lt. rlow) then
            ifl = ijk
            rlow = fwrr(ijk)
          endif
  659   continue
        if(fwr(neut) .gt. rlow) then
          fwrr(ifl) = fwr(neut)
```

```
            rfbank(ifl,1) = x(ncoll(neut))
            rfbank(ifl,2) = y(ncoll(neut))
            rfbank(ifl,3) = z(ncoll(neut))
            ifbank(ifl,1) = kcore(ncoll(neut))
            ifbank(ifl,2) = nbx(ncoll(neut))
            ifbank(ifl,3) = nby(ncoll(neut))
            ifbank(ifl,4) = nbz(ncoll(neut))
            ifbank(ifl,5) = k(ncoll(neut))
            ifbank(ifl,6) = ig(ncoll(neut))
            ij = 7
            if(numhol.gt.0) then
              ifbank(ifl,ij) = nhole(ncoll(neut))
              ij = ij + 1
            endif
            if(nesta) then
              ifbank(ifl,ij) = narrin(ncoll(neut))
              do 660 ik = 1, lnstka
                ifbank(ifl,ij+ik) = stacka(ncoll(neut),ik)
  660         continue
              ij = ij + 1 + lnstka
            endif
            if(nesth) then
              ifbank(ifl,ij) = nholin(ncoll(neut))
              do 661 ik = 1, lnstkh
                ifbank(ifl,ij+ik) = stackh(ncoll(neut),ik)
  661         continue
            endif
          endif
  658   continue
 1200   continue
        do 662 neut = 1, nbank
          ncoll(neut) = 0
  662   continue
        lcol = 0
        go to 1001
c********************* end fission ********************
10003   format('entering free-flight' /
     * 1x,'hist',6x,'x',11x,'y',11x,'z',10x,'u',9x,'v',9x,
     * 'w',8x,'k',3x,'ll',1x,'kcor',2x,'nbx',2x,'nby',2x,
     * 'nbz',1x,'nhole')
10004   format('entering inward-crossing' /
     * 1x,'hist',6x,'x',11x,'y',11x,'z',10x,'u',9x,'v',9x,
     * 'w',8x,'k',3x,'ll',1x,'kcor',2x,'nbx',2x,'nby',2x,
     * 'nbz',1x,'nhole')
10005   format('entering outward-crossing' /
     * 1x,'hist',6x,'x',11x,'y',11x,'z',10x,'u',9x,'v',9x,
     * 'w',8x,'k',3x,'ll',1x,'kcor',2x,'nbx',2x,'nby',2x,
     * 'nbz',1x,'nhole')
10006   format('entering collision' /
     * 1x,'hist',6x,'x',11x,'y',11x,'z',10x,'u',9x,'v',9x,
     * 'w',8x,'k',3x,'ll',1x,'kcor',2x,'nbx',2x,'nby',2x,
     * 'nbz',1x,'nhole')
```

```
10043 format('exiting inward crossing -- free-flight' /
     * 1x,'hist',6x,'x',11x,'y',11x,'z',10x,'u',9x,'v',9x,
     * 'w',8x,'k',3x,'ll',1x,'kcor',2x,'nbx',2x,'nby',2x,
     * 'nbz',1x,'nhole')
10045 format('exiting inward crossing -- outward-crossing'/
     * 1x,'hist',6x,'x',11x,'y',11x,'z',10x,'u',9x,'v',9x,
     *  'w',8x,'k',3x,'ll',1x,'kcor',2x,'nbx',2x,'nby',2x,
     * 'nbz',1x,'nhole')
10000 format(i5,3e12.5,3f10.5,7i5)
      return
      end
```

Appendix B

Listing of Vectorized Version of the Subroutine CROS

```
      subroutine cros(ncros,x,y,z,x1,y1,z1,etausd,k,xx,
     *  lcrs,m,n,igeom,ngeom,jgeom,lgtyp,iar,igeo,nhemi,r,
     *  s,t,tx,ty,tz,tx1,ty1,tz1,f,p,q,dis,dtausd,sgn)
      double precision r,s,t,tx,ty,tz,tx1,ty1,tz1,f,p,
     *  q,dis,dtausd,zero
c.....
c.....
      common /dimen/ tmax, batch, dwtav,wthigh,wtlow,big,
     1          nba, npb,  nskip,   nbas, nrstrt, numx1d,
     2        nbank,  nxnbk,  nfbnk,  nxfbk,
     3        lbank,  lfbnk, lnubnk, lnextr, lnfbnk,
     4          ngp, ngp1, ntypst,   nmat,   matt,   nmix,
     5         kmax, krefm,  nbox,  nboxt,  ngblu,  nucom,
     6       nbxmax, nbymax,nbzmax, nglobl, matdim, nacom,
     7       maxmix, maximp, mang, nimp,numids,ncs,ntsets,
     8          nsg, nchigp, nsct, mix,  mixt,  npl, pbxs,
     9       maxara, lbalng,  ndelx,  ndely,  ndelz,
     a       numara, nalvls, lnstka,lstka, lfstka,  iaa,
     b       numhol, nhlvls, lnstkh,  lstkh, lfstkh, ihh,
     c       mult
      dimension x(nbank),y(nbank),z(nbank),x1(nbank),
     *  y1(nbank),z1(nbank),etausd(nbank),k(nbank),
     *  xx(7,krefm),ncros(nbank),m(nbank),igeom(krefm),
     *  ngeom(2,10),iar(nbank),igeo(nbank),nhemi(nbank),
     *  jgeom(10)
      dimension r(nbank),s(nbank),t(nbank),tx(nbank),
     *  ty(nbank),tz(nbank),tx1(nbank),ty1(nbank),
     *  tz1(nbank),f(nbank),p(nbank),q(nbank),dis(nbank),
     *  dtausd(nbank),sgn(nbank)
c.....
      logical lxhcyy,lxhcyz,lyhcyx,lyhcyz,lzhcyx,lzhcyy
      logical lhsphx,lhsphy,lhsphz
      common /lhemi/ lxhcyy, lxhcyz, lyhcyx, lyhcyz,
     *       lzhcyx, lzhcy, lhsphx, lhsphy, lhsphz
c.....
      common /unit/ inpt, outpt, icexs, albdo, wts, skrt,
     * rstrt, wstrt, ampxs, direct(3), i0, i1, i2, i3, i4,
     * nspare(4)
      integer outpt,albdo,skrt,rstrt,wstrt,direct,wts,
     * icexs,ampxs
c.....
c
c     igeo=1  cube or cuboid
c     igeo=2  cylinder
c     igeo=3  sphere
c     igeo=4  reflector following a core boundary card
c     igeo=5  x-cylinder
c     igeo=6  y-cylinder
```

```
c      igeo=7  hemisphere
c      igeo=8  z-hemicylinder in the -x direction
c      igeo=9  z-hemicylinder in the +x direction
c      igeo=10 z-hemicylinder in the -y direction
c      igeo=11 z-hemicylinder in the +y direction
c      igeo=12 x-hemicylinder in the -z direction
c      igeo=13 x-hemicylinder in the +z direction
c      igeo=14 x-hemicylinder in the -y direction
c      igeo=15 x-hemicylinder in the +y direction
c      igeo=16 y-hemicylinder in the -z direction
c      igeo=17 y-hemicylinder in the +z direction
c      igeo=18 y-hemicylinder in the -x direction
c      igeo=19 y-hemicylinder in the +x direction
c      igeo=20 unit or box type
c      igeo=-1 core boundary
c      igeo=21 hole
c      igeo=22 end geometry
c      igeo=23 comment
c
       zero = 0.0
       do 100 neut = 1, lcrs
         dtausd(neut) = zero
         m(neut) = 0
  100  continue
       if(n .lt. 0) then
         do 101 neut = 1, lcrs
           iar(neut) = k(ncros(neut)) - 1
  101    continue
       endif
       if(n .gt. 0) then
         do 102 neut = 1, lcrs
           iar(neut) = k(ncros(neut))
  102    continue
       endif
       do 999 lgt = 1, lgtyp
         go to (901,902,903,901,905,906,907,908,909,900)
     *   jgeom(lgt)
c**********************cuboid package*******************
  901    continue
         if(ngeom(2,1) .eq. 0) go to 999
         lstart = ngeom(1,1)
         lend   = ngeom(2,1)
         do 110 neut = lstart, lend
           r(neut) = x1(ncros(neut))-x(ncros(neut))
           s(neut) = y1(ncros(neut))-y(ncros(neut))
           t(neut) = z1(ncros(neut))-z(ncros(neut))
  110    continue
         if (n.gt.0) go to 150
c*******inward crossing
c******* x - face crossing
         do 111 neut = lstart, lend
           if(xx(1,iar(neut)).le.x(ncros(neut)) .and.
```

```
     *       xx(1,iar(neut)).ge.x1(ncros(neut)) .and.
     *       r(neut).ne.0.0) then
             dtausd(neut) = (xx(1,iar(neut))-
     *                       x(ncros(neut)))/r(neut)
             tx1(neut) = xx(1,iar(neut))
             m(neut) = -1
           endif
  111    continue
         do 112 neut =  lstart, lend
           if(xx(2,iar(neut)).ge.x(ncros(neut)) .and.
     *       xx(2,iar(neut)).le.x1(ncros(neut)) .and.
     *       r(neut).ne.0.0) then
             dtausd(neut) = (xx(2,iar(neut))-
     *                       x(ncros(neut)))/r(neut)
             tx1(neut) = xx(2,iar(neut))
             m(neut) = -1
           endif
  112    continue
         do 113 neut = lstart, lend
           if(m(neut) .eq. -1) then
           ty1(neut)=dtausd(neut)*s(neut)+y(ncros(neut))
           tz1(neut)=dtausd(neut)*t(neut)+z(ncros(neut))
           endif
  113    continue
cdir$ ivdep
         do 114 neut = lstart, lend
           if(ty1(neut).le.xx(3,iar(neut)) .and.
     *        ty1(neut).ge.xx(4,iar(neut)) .and.
     *        tz1(neut).le.xx(5,iar(neut)) .and.
     *        tz1(neut).ge.xx(6,iar(neut)) .and.
     *        m(neut).eq.-1) m(neut) = 1
  114    continue
c******* y - face crossing
         do 121 neut = lstart, lend
           if(xx(3,iar(neut)).le.y(ncros(neut)) .and.
     *       xx(3,iar(neut)).ge.y1(ncros(neut)) .and.
     *       s(neut).ne.0.0 .and. m(neut).le.0) then
             dtausd(neut) = (xx(3,iar(neut))-
     *                       y(ncros(neut)))/s(neut)
             ty1(neut) = xx(3,iar(neut))
             m(neut) = -2
           endif
  121    continue
         do 122 neut = lstart, lend
           if(xx(4,iar(neut)).ge.y(ncros(neut)) .and.
     *       xx(4,iar(neut)).le.y1(ncros(neut)) .and.
     *       s(neut).ne.0.0 .and. m(neut).le.0) then
             dtausd(neut) = (xx(4,iar(neut))-
     *                       y(ncros(neut)))/s(neut)
             ty1(neut) = xx(4,iar(neut))
             m(neut) = -2
           endif
```

```
  122   continue
        do 123 neut = lstart, lend
          if(m(neut) .eq. -2) then
          tx1(neut)=dtausd(neut)*r(neut)+x(ncros(neut))
          tz1(neut)=dtausd(neut)*t(neut)+z(ncros(neut))
          endif
  123   continue
cdir$ ivdep
        do 124 neut = lstart, lend
          if(tx1(neut).le.xx(1,iar(neut)) .and.
     *       tx1(neut).ge.xx(2,iar(neut)) .and.
     *       tz1(neut).le.xx(5,iar(neut)) .and.
     *       tz1(neut).ge.xx(6,iar(neut)) .and.
     *       m(neut).eq.-2) m(neut) = 1
  124   continue
c******* z - face crossing
        do 131 neut = lstart, lend
          if(xx(5,iar(neut)).le.z(ncros(neut)) .and.
     *      xx(5,iar(neut)).ge.z1(ncros(neut)) .and.
     *      t(neut).ne.0.0 .and. m(neut).le.0) then
            dtausd(neut)=(xx(5,iar(neut))-
     *                   z(ncros(neut)))/t(neut)
            tz1(neut) = xx(5,iar(neut))
            m(neut) = -3
          endif
  131   continue
        do 132 neut = lstart, lend
          if(xx(6,iar(neut)).ge.z(ncros(neut)) .and.
     *      xx(6,iar(neut)).lt.z1(ncros(neut)) .and.
     *      t(neut).ne.0.0 .and. m(neut).le.0) then
            dtausd(neut)=(xx(6,iar(neut))-
     *                   z(ncros(neut)))/t(neut)
            tz1(neut) = xx(6,iar(neut))
            m(neut) = -3
          endif
  132   continue
        do 133 neut = lstart, lend
          if(m(neut) .eq. -3) then
            tx1(neut) = dtausd(neut)*r(neut) +
     *                  x(ncros(neut))
            ty1(neut) = dtausd(neut)*s(neut) +
     *                  y(ncros(neut))
          endif
  133   continue
cdir$ ivdep
        do 134 neut = lstart, lend
          if(ty1(neut).le.xx(3,iar(neut)) .and.
     *       ty1(neut).ge.xx(4,iar(neut)) .and.
     *       tx1(neut).le.xx(1,iar(neut)) .and.
     *       tx1(neut).ge.xx(2,iar(neut)) .and.
     *       m(neut).eq.-3) m(neut) = 1
  134   continue
```

```
        do 135 neut = lstart, lend
          if(m(neut).eq.1) then
            etausd(ncros(neut)) = dtausd(neut)
            x(ncros(neut)) = tx1(neut)
            y(ncros(neut)) = ty1(neut)
            z(ncros(neut)) = tz1(neut)
          endif
  135   continue
        go to 999
c*******end inward crossing
c*******outward crossing
  150 continue
c******* x - face crossing
        do 151 neut = lstart, lend
          if(xx(1,iar(neut)).le.x1(ncros(neut)) .and.
     *      r(neut).ne.0.0) then
            dtausd(neut)=(xx(1,iar(neut))-x(ncros(neut)))/
     *      r(neut)
            tx1(neut) = xx(1,iar(neut))
            m(neut) = -1
          endif
  151   continue
        do 152 neut =  lstart, lend
          if(xx(2,iar(neut)).ge.x1(ncros(neut)) .and.
     *      r(neut).ne.0.0) then
            dtausd(neut)=(xx(2,iar(neut))-
     *                  x(ncros(neut)))/r(neut)
            tx1(neut) = xx(2,iar(neut))
            m(neut) = -1
          endif
  152   continue
        do 153 neut = lstart, lend
          if(m(neut) .eq. -1) then
          ty1(neut)=dtausd(neut)*s(neut)+y(ncros(neut))
          tz1(neut)=dtausd(neut)*t(neut)+z(ncros(neut))
          endif
  153   continue
cdir$ ivdep
        do 154 neut = lstart, lend
          if(ty1(neut).le.xx(3,iar(neut)) .and.
     *       ty1(neut).ge.xx(4,iar(neut)) .and.
     *       tz1(neut).le.xx(5,iar(neut)) .and.
     *       tz1(neut).ge.xx(6,iar(neut)) .and.
     *       m(neut).eq.-1) m(neut) = 1
  154   continue
c******* y - face crossing
        do 161 neut = lstart, lend
          if(xx(3,iar(neut)).le.y1(ncros(neut)) .and.
     *      s(neut).ne.0.0 .and. m(neut).le.0) then
            dtausd(neut)=(xx(3,iar(neut))-
     *                  y(ncros(neut)))/s(neut)
            ty1(neut) = xx(3,iar(neut))
```

```
            m(neut) = -2
          endif
  161   continue
        do 162 neut = lstart, lend
          if(xx(4,iar(neut)).ge.y1(ncros(neut)) .and.
     *      s(neut).ne.0.0 .and. m(neut).le.0) then
          dtausd(neut)=(xx(4,iar(neut))-
     *                  y(ncros(neut)))/s(neut)
            ty1(neut) = xx(4,iar(neut))
            m(neut) = -2
          endif
  162   continue
        do 163 neut = lstart, lend
          if(m(neut) .eq. -2) then
          tx1(neut)=dtausd(neut)*r(neut)+x(ncros(neut))
          tz1(neut)=dtausd(neut)*t(neut)+z(ncros(neut))
          endif
  163   continue
cdir$ ivdep
        do 164 neut = lstart, lend
          if(tx1(neut).le.xx(1,iar(neut)) .and.
     *       tx1(neut).ge.xx(2,iar(neut)) .and.
     *       tz1(neut).le.xx(5,iar(neut)) .and.
     *       tz1(neut).ge.xx(6,iar(neut)) .and.
     *       m(neut).eq.-2) m(neut) = 1
  164   continue
c******* z - face crossing
        do 171 neut = lstart, lend
          if(xx(5,iar(neut)).le.z1(ncros(neut)) .and.
     *      t(neut).ne.0.0 .and. m(neut).le.0) then
          dtausd(neut)=(xx(5,iar(neut))-
     *                  z(ncros(neut)))/t(neut)
            tz1(neut) = xx(5,iar(neut))
          endif
  171   continue
        do 172 neut = lstart, lend
          if(xx(6,iar(neut)).ge.z1(ncros(neut)) .and.
     *      t(neut).ne.0.0 .and. m(neut).le.0) then
          dtausd(neut)=(xx(6,iar(neut))-
     *                  z(ncros(neut)))/t(neut)
            tz1(neut) = xx(6,iar(neut))
          endif
  172   continue
        do 173 neut = lstart, lend
          if(m(neut) .le. 0) then
            tx1(neut) = dtausd(neut)*r(neut) +
     *                  x(ncros(neut))
            ty1(neut) = dtausd(neut)*s(neut) +
     *                  y(ncros(neut))
            m(neut) = 1
          endif
  173   continue
```

```
cdir$ ivdep
        do 174 neut = lstart, lend
            etausd(ncros(neut)) = dtausd(neut)
            x(ncros(neut)) = tx1(neut)
            y(ncros(neut)) = ty1(neut)
            z(ncros(neut)) = tz1(neut)
  174   continue
        go to 999
c*******end outward crossing
c*********************end cuboid package****************
c*********************sphere package******************
  903   if(ngeom(2,3) .eq. 0) go to 999
        lstart = ngeom(1,3)
        lend   = ngeom(2,3)
        do 211 neut = lstart, lend
          r(neut)  = x1(ncros(neut))-x(ncros(neut))
          s(neut)  = y1(ncros(neut))-y(ncros(neut))
          t(neut)  = z1(ncros(neut))-z(ncros(neut))
          tx(neut) = x(ncros(neut))-xx(4,iar(neut))
          ty(neut) = y(ncros(neut))-xx(5,iar(neut))
          tz(neut) = z(ncros(neut))-xx(6,iar(neut))
  211   continue
        do 212 neut = lstart, lend
          q(neut) = r(neut)*tx(neut)+s(neut)*ty(neut) +
     *              t(neut)*tz(neut)
          p(neut) = r(neut)**2 + s(neut)**2 + t(neut)**2
  212   continue
        do 213 neut = lstart, lend
          dis(neut) = q(neut)**2 - p(neut)*(tx(neut)**2 +
     *    ty(neut)**2 + tz(neut)**2 - xx(2,iar(neut)))
  213   continue
        if(n.gt.0)     go to 215
c*******inward crossing
        do 214 neut = lstart, lend
          if(dis(neut).gt.0.0) dtausd(neut) =
     *      max((-q(neut)-sqrt(dis(neut)))/p(neut),zero)
  214   continue
cdir$ ivdep
        do 217 neut = lstart, lend
          if(dtausd(neut).le.1.0.and.q(neut).lt.0.0)then
           etausd(ncros(neut)) = dtausd(neut)
           x(ncros(neut))=dtausd(neut)*r(neut)+
     *                    x(ncros(neut))
           y(ncros(neut))=dtausd(neut)*s(neut)+
     *                    y(ncros(neut))
           z(ncros(neut))=dtausd(neut)*t(neut)+
     *                    z(ncros(neut))
           m(neut) = 1
          endif
  217   continue
        go to 999
c*******end inward crossing
```

```
c*******outward crossing
  215   do 216 neut = lstart, lend
          if(dis(neut) .gt. 0.0) dtausd(neut) =
     *      max((-q(neut)+sqrt(dis(neut)))/p(neut),zero)
  216   continue
cdir$ ivdep
        do 218 neut = lstart, lend
           etausd(ncros(neut)) = dtausd(neut)
           x(ncros(neut))=dtausd(neut)*r(neut)+
     *                    x(ncros(neut))
           y(ncros(neut))=dtausd(neut)*s(neut)+
     *                    y(ncros(neut))
           z(ncros(neut))=dtausd(neut)*t(neut)+
     *                    z(ncros(neut))
  218   continue
        go to 999
c*******end outward crossing
c*********************end spherical Package*************
c*********************Hemispherical Package*************
  907   if(ngeom(2,7).eq.0) go to 999
        lstart = ngeom(1,7)
        lend   = ngeom(2,7)
        do 221 neut = lstart, lend
          nhemi(neut) = (abs(xx(3,iar(neut))) + 1)/2
          sgn(neut)   = sign(1.0,xx(3,iar(neut)))
  221   continue
        if(lhsphx) then
        do 222 neut = lstart, lend
          if(nhemi(neut) .eq. 1) then
            tx(neut)  = y(ncros(neut))-xx(5,iar(neut))
            ty(neut)  = z(ncros(neut))-xx(6,iar(neut))
            tz(neut)  = x(ncros(neut))-xx(4,iar(neut))
            tx1(neut) = y1(ncros(neut))-xx(5,iar(neut))
            ty1(neut) = z1(ncros(neut))-xx(6,iar(neut))
            tz1(neut) = x1(ncros(neut))-xx(4,iar(neut))
          endif
  222   continue
        endif
        if(lhsphy) then
        do 223 neut = lstart, lend
          if(nhemi(neut) .eq. 2) then
            tx(neut)  = z(ncros(neut))-xx(6,iar(neut))
            ty(neut)  = x(ncros(neut))-xx(4,iar(neut))
            tz(neut)  = y(ncros(neut))-xx(5,iar(neut))
            tx1(neut) = z1(ncros(neut))-xx(6,iar(neut))
            ty1(neut) = x1(ncros(neut))-xx(4,iar(neut))
            tz1(neut) = y1(ncros(neut))-xx(5,iar(neut))
          endif
  223   continue
        endif
        if(lhsphz) then
        do 224 neut = lstart, lend
```

```
            if(nhemi(neut) .eq. 3) then
              tx(neut) = x(ncros(neut))-xx(4,iar(neut))
              ty(neut) = y(ncros(neut))-xx(5,iar(neut))
              tz(neut) = z(ncros(neut))-xx(6,iar(neut))
              tx1(neut) = x1(ncros(neut))-xx(4,iar(neut))
              ty1(neut) = y1(ncros(neut))-xx(5,iar(neut))
              tz1(neut) = z1(ncros(neut))-xx(6,iar(neut))
            endif
  224     continue
          endif
          do 225 neut = lstart, lend
            r(neut)    = tx1(neut) - tx(neut)
            s(neut)    = ty1(neut) - ty(neut)
            t(neut)    = tz1(neut) - tz(neut)
  225     continue
          do 226 neut = lstart, lend
            q(neut) = r(neut)*tx(neut)+s(neut)*ty(neut) +
                      t(neut)*tz(neut)
            p(neut) = r(neut)**2 + s(neut)**2 + t(neut)**2
            f(neut) = tx(neut)**2+ty(neut)**2+tz(neut)**2-
     *              xx(2,iar(neut))
  226     continue
          do 227 neut = lstart, lend
            dis(neut) = q(neut)**2 - p(neut)*f(neut)
  227     continue
          if(n.gt.0) go to 250
c*******inward crossing
c****hemi-surface crossing check
          do 231 neut = lstart, lend
            if(tz(neut)*sgn(neut).le.-xx(7,iar(neut)).and.
     *        tz1(neut)*sgn(neut).ge.-xx(7,iar(neut)))then
              dtausd(neut) = (-sgn(neut)*xx(7,iar(neut))-
     *                       tz(neut))/t(neut)
              m(neut) = 1
            endif
  231     continue
          do 232 neut = lstart, lend
            if(m(neut).eq.1) then
            tx1(neut) = tx(neut) + r(neut)*dtausd(neut)
            ty1(neut) = ty(neut) + s(neut)*dtausd(neut)
            tz1(neut) = -sgn(neut)*xx(7,iar(neut))
            endif
  232     continue
          do 233 neut = lstart, lend
            if(tx1(neut)**2+ty1(neut)**2+tz1(neut)**2.ge.
     *        xx(2,iar(neut)) .and. m(neut).eq.1) then
              m(neut) = 0
              dtausd(neut) = zero
            endif
  233     continue
c****curved surface crossing check
          do 234 neut = lstart, lend
```

```
          if(m(neut).eq.0 .and. dis(neut).gt.0.0)
     *      dtausd(neut) = max((-q(neut) -
     *      sqrt(dis(neut)))/p(neut),zero)
  234   continue
        do 235 neut = lstart, lend
          if(m(neut).eq.0)tz1(neut)=
     *      tz(neut)+t(neut)*dtausd(neut)
  235   continue
        do 236 neut = lstart, lend
          if(m(neut).eq.0.and.dtausd(neut).le.1.0
     *    .and.q(neut).lt.0.0.and.tz1(neut)*sgn(neut)
     *    .ge.-xx(7,iar(neut)))then
            tx1(neut) = tx(neut) + r(neut)*dtausd(neut)
            ty1(neut) = ty(neut) + s(neut)*dtausd(neut)
            m(neut)   = 1
          endif
  236   continue
        go to 260
c*******end inward crossing
c*******outward crossing
c****curved surface crossing check
  250   do 251 neut = lstart, lend
          if(dis(neut).gt.0.0) dtausd(neut) =
     *      max((-q(neut)+sqrt(dis(neut)))/p(neut),zero)
  251   continue
        do 252 neut = lstart, lend
            tx1(neut) = tx(neut) + r(neut)*dtausd(neut)
            ty1(neut) = ty(neut) + s(neut)*dtausd(neut)
            tz1(neut) = tz(neut) + t(neut)*dtausd(neut)
  252   continue
        do 253 neut = lstart, lend
          if(sgn(neut)*tz1(neut).ge.-xx(7,iar(neut)))
     *      m(neut) = 1
  253   continue
c****hemi-surface crossing check
        do 254 neut = lstart, lend
          if(m(neut).eq.0) dtausd(neut) = -(sgn(neut)*
     *     xx(7,iar(neut))+tz(neut))/t(neut)
  254   continue
        do 255 neut = lstart, lend
          if(m(neut).eq.0) then
            tx1(neut) = tx(neut) + r(neut)*dtausd(neut)
            ty1(neut) = ty(neut) + s(neut)*dtausd(neut)
            tz1(neut) = -sgn(neut)*xx(7,iar(neut))
            m(neut) = 1
          endif
  255   continue
cdir$ ivdep
  260   if(lhsphx) then
        do 261 neut = lstart, lend
          if(nhemi(neut).eq.1 .and. m(neut).eq.1) then
            x(ncros(neut)) = tz1(neut)+xx(4,iar(neut))
```

```
            y(ncros(neut)) = tx1(neut)+xx(5,iar(neut))
            z(ncros(neut)) = ty1(neut)+xx(6,iar(neut))
            etausd(ncros(neut)) = dtausd(neut)
          endif
  261   continue
        endif
cdir$ ivdep
        if(lhsphy) then
        do 262 neut = lstart, lend
          if(nhemi(neut).eq.2 .and. m(neut).eq.1) then
            x(ncros(neut)) = ty1(neut)+xx(4,iar(neut))
            y(ncros(neut)) = tz1(neut)+xx(5,iar(neut))
            z(ncros(neut)) = tx1(neut)+xx(6,iar(neut))
            etausd(ncros(neut)) = dtausd(neut)
          endif
  262   continue
        endif
cdir$ ivdep
        if(lhsphz) then
        do 263 neut = lstart, lend
          if(nhemi(neut).eq.3 .and. m(neut).eq.1) then
            x(ncros(neut)) = tx1(neut)+xx(4,iar(neut))
            y(ncros(neut)) = ty1(neut)+xx(5,iar(neut))
            z(ncros(neut)) = tz1(neut)+xx(6,iar(neut))
            etausd(ncros(neut)) = dtausd(neut)
          endif
  263   continue
        endif
      go to 999
c*******end outward crossing
c****************end hemispherical package*************
c*******************Cylinder package********************
c*******start z - cylinder
  902   assign 332 to num
        if(ngeom(2,2) .eq. 0) go to 999
        lstart = ngeom(1,2)
        lend   = ngeom(2,2)
        do 301 neut = lstart, lend
          tx(neut)  = x(ncros(neut)) - xx(5,iar(neut))
          ty(neut)  = y(ncros(neut)) - xx(6,iar(neut))
          tz(neut)  = z(ncros(neut))
          tz1(neut) = z1(ncros(neut))
          r(neut)   = x1(ncros(neut)) - x(ncros(neut))
          s(neut)   = y1(ncros(neut)) - y(ncros(neut))
          t(neut)   = z1(ncros(neut)) - z(ncros(neut))
  301   continue
        go to 339
c*******start x - cylinder
  905   assign 335 to num
        if(ngeom(2,5) .eq. 0) go to 999
        lstart = ngeom(1,5)
        lend   = ngeom(2,5)
```

```
        do 303 neut = lstart, lend
          tx(neut)   = y(ncros(neut)) - xx(5,iar(neut))
          ty(neut)   = z(ncros(neut)) - xx(6,iar(neut))
          tz(neut)   = x(ncros(neut))
          tz1(neut)  = x1(ncros(neut))
          r(neut)    = y1(ncros(neut)) - y(ncros(neut))
          s(neut)    = z1(ncros(neut)) - z(ncros(neut))
          t(neut)    = x1(ncros(neut)) - x(ncros(neut))
  303   continue
        go to 339
c*******start y - cylinder
  906   assign 336 to num
        if(ngeom(2,6) .eq. 0) go to 999
        lstart = ngeom(1,6)
        lend   = ngeom(2,6)
        do 305 neut = lstart, lend
          tx(neut)   = z(ncros(neut)) - xx(6,iar(neut))
          ty(neut)   = x(ncros(neut)) - xx(5,iar(neut))
          tz(neut)   = y(ncros(neut))
          tz1(neut)  = y1(ncros(neut))
          r(neut)    = z1(ncros(neut)) - z(ncros(neut))
          s(neut)    = x1(ncros(neut)) - x(ncros(neut))
          t(neut)    = y1(ncros(neut)) - y(ncros(neut))
  305   continue
  339   if (n.gt.0)     go to 340
c*******inner crossing check
c****end surface crossing check
        do 311 neut = lstart, lend
          if(t(neut).ne.0.0.and.tz(neut).ge.
     *    xx(2,iar(neut)).and.tz1(neut).le.
     *    xx(2,iar(neut))) then
            dtausd(neut) = (xx(2,iar(neut))-
     *                     tz(neut))/t(neut)
            tz1(neut) = xx(2,iar(neut))
            m(neut) = -1
          endif
  311   continue
        do 312 neut = lstart, lend
          if(t(neut).ne.0.0 .and. tz(neut).le.
     *    xx(3,iar(neut)).and.tz1(neut).ge.
     *    xx(3,iar(neut))) then
            dtausd(neut) = (xx(3,iar(neut))-
     *                     tz(neut))/t(neut)
            tz1(neut) = xx(3,iar(neut))
            m(neut) = -1
          endif
  312   continue
        do 313 neut = lstart, lend
          if(m(neut).eq.-1) then
            tx1(neut) = dtausd(neut)*r(neut) + tx(neut)
            ty1(neut) = dtausd(neut)*s(neut) + ty(neut)
          endif
```

```
  313   continue
        do 314 neut = lstart, lend
          if(m(neut).eq.-1 .and. tx1(neut)**2+
     *    ty1(neut)**2.le.xx(4,iar(neut))) m(neut) = 1
  314   continue
c****curved surface crossing check
        do 321 neut = lstart, lend
          if(m(neut).le.0) then
            q(neut) = tx(neut)*r(neut)+ty(neut)*s(neut)
            f(neut) = xx(4,iar(neut)) - tx(neut)**2 -
     *                ty(neut)**2
            p(neut) = r(neut)**2 + s(neut)**2
          endif
  321   continue
        do 322 neut = lstart, lend
          if(m(neut).le.0) dis(neut) =
q(neut)**2+p(neut)*f(neut)
  322   continue
        do 323 neut = lstart, lend
          if(m(neut).le.0.and.dis(neut).gt.0.0)then
            dtausd(neut) = max((-q(neut)-
     *      sqrt(dis(neut)))/p(neut),zero)
            m(neut) = -2
          endif
  323   continue
        do 324 neut = lstart, lend
          if(m(neut).eq. -2 .and. q(neut).lt.0.0 .and.
     *      dtausd(neut).le.1.0) then
            tz1(neut) = dtausd(neut)*t(neut) + tz(neut)
            m(neut) = -3
          endif
  324   continue
        do 325 neut = lstart, lend
          if(m(neut).eq. -3 .and. tz1(neut).le.
     *    xx(2,iar(neut)) .and. tz1(neut).ge.
     *    xx(3,iar(neut))) then
            tx1(neut) = dtausd(neut)*r(neut) + tx(neut)
            ty1(neut) = dtausd(neut)*s(neut) + ty(neut)
            m(neut)  = 1
          endif
  325   continue
        go to 399
c*******end inner crossing check
c*******outer crossing check
c****top surface crossing check
  340   do 341 neut = lstart, lend
          if(t(neut).ne.0.0.and.tz1(neut).gt.
     *    xx(2,iar(neut)))then
            dtausd(neut) = (xx(2,iar(neut))-
     *                     tz(neut))/t(neut)
            tz1(neut) = xx(2,iar(neut))
            m(neut) = 1
```

```
        endif
  341   continue
        do 342 neut = lstart, lend
          if(t(neut).ne.0.0.and.tz1(neut).lt.
     *    xx(3,iar(neut)))then
            dtausd(neut) = (xx(3,iar(neut))-
     *                     tz(neut))/t(neut)
            tz1(neut) = xx(3,iar(neut))
            m(neut) = 1
          endif
  342   continue
        do 343 neut = lstart, lend
          if(m(neut).eq.1) then
            tx1(neut) = dtausd(neut)*r(neut) + tx(neut)
            ty1(neut) = dtausd(neut)*s(neut) + ty(neut)
          endif
  343   continue
        do 344 neut = lstart, lend
          if(m(neut).eq.1 .and. tx1(neut)**2+
     *    ty1(neut)**2.gt.xx(4,iar(neut))) then
            m(neut) = 0
            dtausd(neut) = zero
          endif
  344   continue
c****curved surface crossing check
        do 351 neut = lstart, lend
            q(neut) = tx(neut)*r(neut)+ty(neut)*s(neut)
            f(neut) = xx(4,iar(neut)) - tx(neut)**2 -
     *                ty(neut)**2
            p(neut) = r(neut)**2 + s(neut)**2
  351   continue
        do 352 neut = lstart, lend
          dis(neut) = q(neut)**2 + p(neut)*f(neut)
  352   continue
        do 353 neut = lstart, lend
          if(m(neut).eq.0 .and. dis(neut).gt.0.0)
     *    dtausd(neut) = max((-q(neut)+sqrt(dis(neut)))/
     *                   p(neut),zero)
  353   continue
        do 354 neut = lstart, lend
          if(m(neut).le.0) then
            tx1(neut) = dtausd(neut)*r(neut) + tx(neut)
            ty1(neut) = dtausd(neut)*s(neut) + ty(neut)
            tz1(neut) = dtausd(neut)*t(neut) + tz(neut)
            m(neut) = 1
          endif
  354   continue
c*******end outward crossing
  399   go to num, (332,335,336)
  332   do 391 neut = lstart, lend
          if(m(neut).eq.1) then
            x(ncros(neut)) = tx1(neut) + xx(5,iar(neut))
```

```
            y(ncros(neut)) = ty1(neut) + xx(6,iar(neut))
            z(ncros(neut)) = tz1(neut)
            etausd(ncros(neut)) = dtausd(neut)
          endif
  391   continue
        go to 999
  336   do 392 neut = lstart, lend
          if(m(neut).eq.1) then
            x(ncros(neut)) = ty1(neut) + xx(5,iar(neut))
            y(ncros(neut)) = tz1(neut)
            z(ncros(neut)) = tx1(neut) + xx(6,iar(neut))
            etausd(ncros(neut)) = dtausd(neut)
          endif
  392   continue
        go to 999
  335   do 393 neut = lstart, lend
          if(m(neut).eq.1) then
            x(ncros(neut)) = tz1(neut)
            y(ncros(neut)) = tx1(neut) + xx(5,iar(neut))
            z(ncros(neut)) = ty1(neut) + xx(6,iar(neut))
            etausd(ncros(neut)) = dtausd(neut)
          endif
  393   continue
        go to 999
c******************end cylinder package***************
c******************Hemicylinder package***************
c*******start x - hemicylinder
  908   assign 488 to num
        if(ngeom(2,8) .eq. 0) go to 999
        lstart = ngeom(1,8)
        lend   = ngeom(2,8)
        do 405 neut = lstart, lend
          igeo(neut) = igeom(iar(neut))
  405   continue
        do 401 neut = lstart, lend
          nhemi(neut) = (igeo(neut)-6)/2
          sgn(neut)   = 2*mod(igeo(neut),2) - 1
          tz(neut)    = x(ncros(neut))
          tz1(neut)   = x1(ncros(neut))
          t(neut)     = x1(ncros(neut)) - x(ncros(neut))
  401   continue
        if(lxhcyz) then
        do 402 neut = lstart, lend
          if(nhemi(neut).eq.3) then
            tx(neut) = z(ncros(neut)) - xx(6,iar(neut))
            ty(neut) = y(ncros(neut)) - xx(5,iar(neut))
            r(neut)  = z1(ncros(neut)) - z(ncros(neut))
            s(neut)  = y1(ncros(neut)) - y(ncros(neut))
          endif
  402   continue
        endif
        if(lxhcyy) then
```

```
        do 403 neut = lstart, lend
          if(nhemi(neut).eq.4) then
            tx(neut) = y(ncros(neut)) - xx(5,iar(neut))
            ty(neut) = z(ncros(neut)) - xx(6,iar(neut))
            r(neut)  = y1(ncros(neut)) - y(ncros(neut))
            s(neut)  = z1(ncros(neut)) - z(ncros(neut))
          endif
  403   continue
        endif
        go to 430
c*******start y - hemicylinder
  909   assign 489 to num
        if(ngeom(2,9) .eq. 0) go to 999
        lstart = ngeom(1,9)
        lend   = ngeom(2,9)
        do 415 neut = 1, lcrs
          igeo(neut) = igeom(iar(neut))
  415   continue
        do 411 neut = lstart, lend
          nhemi(neut) = (igeo(neut)-6)/2
          sgn(neut)   = 2*mod(igeo(neut),2) - 1
          tz(neut)    = y(ncros(neut))
          tz1(neut)   = y1(ncros(neut))
          t(neut)     = y1(ncros(neut)) - y(ncros(neut))
  411   continue
        if(lyhcyz) then
        do 412 neut = lstart, lend
          if(nhemi(neut).eq.5) then
            tx(neut) = z(ncros(neut)) - xx(6,iar(neut))
            ty(neut) = x(ncros(neut)) - xx(5,iar(neut))
            r(neut)  = z1(ncros(neut)) - z(ncros(neut))
            s(neut)  = x1(ncros(neut)) - x(ncros(neut))
          endif
  412   continue
        endif
        if(lyhcyx) then
          do 413 neut =lstart, lend
            if(nhemi(neut).eq.6) then
              tx(neut) = x(ncros(neut)) - xx(5,iar(neut))
              ty(neut) = z(ncros(neut)) - xx(6,iar(neut))
              r(neut)  = x1(ncros(neut)) - x(ncros(neut))
              s(neut)  = z1(ncros(neut)) - z(ncros(neut))
            endif
  413     continue
        endif
        go to 430
c*******start z - hemicylinder
  900   assign 490 to num
        if(ngeom(2,10) .eq. 0) go to 999
        lstart = ngeom(1,10)
        lend   = ngeom(2,10)
        do 425 neut = 1, lcrs
```

```
          igeo(neut) = igeom(iar(neut))
  425   continue
        do 421 neut = lstart, lend
          nhemi(neut) = (igeo(neut)-6)/2
          sgn(neut)   = 2*mod(igeo(neut),2) - 1
          tz(neut)    = z(ncros(neut))
          tz1(neut)   = z1(ncros(neut))
          t(neut)     = z1(ncros(neut)) - z(ncros(neut))
  421   continue
        if(lzhcyx) then
        do 422 neut = lstart, lend
          if(nhemi(neut).eq.1) then
            tx(neut) = x(ncros(neut)) - xx(5,iar(neut))
            ty(neut) = y(ncros(neut)) - xx(6,iar(neut))
            r(neut)  = x1(ncros(neut)) - x(ncros(neut))
            s(neut)  = y1(ncros(neut)) - y(ncros(neut))
          endif
  422   continue
        endif
        if(lzhcyy) then
        do 423 neut = lstart, lend
          if(nhemi(neut).eq.2) then
            tx(neut) = y(ncros(neut)) - xx(6,iar(neut))
            ty(neut) = x(ncros(neut)) - xx(5,iar(neut))
            r(neut)  = y1(ncros(neut)) - y(ncros(neut))
            s(neut)  = x1(ncros(neut)) - x(ncros(neut))
          endif
  423   continue
        endif
  430   if (n.gt.0)     go to 450
c*******inner crossing check
c****end surface crossing check
        do 431 neut = lstart, lend
          if(t(neut).ne.0.0 .and. tz(neut).ge.
     *    xx(2,iar(neut)).and.tz1(neut).le.
     *    xx(2,iar(neut))) then
            dtausd(neut) = (xx(2,iar(neut))-
     *                     tz(neut))/t(neut)
            tz1(neut) = xx(2,iar(neut))
            m(neut) = -1
          endif
  431   continue
        do 432 neut = lstart, lend
          if(t(neut).ne.0.0 .and. tz(neut).le.
     *    xx(3,iar(neut)).and.tz1(neut).ge.
     *    xx(3,iar(neut))) then
             dtausd(neut) = (xx(3,iar(neut))-
     *                      tz(neut))/t(neut)
             tz1(neut) = xx(3,iar(neut))
             m(neut) = -1
          endif
  432   continue
```

```
      do 433 neut = lstart, lend
        if(m(neut).eq.-1) then
          tx1(neut) = dtausd(neut)*r(neut) + tx(neut)
          ty1(neut) = dtausd(neut)*s(neut) + ty(neut)
        endif
433   continue
      do 434 neut = lstart, lend
        if(m(neut).eq.-1 .and. tx1(neut)**2+
     *  ty1(neut)**2.le.xx(4,iar(neut)).and.sgn(neut)*
     *  tx1(neut).ge.-xx(7,iar(neut))) m(neut) = 1
434   continue
c****curved surface crossing check
      do 438 neut = lstart, lend
          q(neut) = tx(neut)*r(neut) + ty(neut)*s(neut)
          f(neut) = xx(4,iar(neut)) - tx(neut)**2 -
     *              ty(neut)**2
          p(neut) = r(neut)**2 + s(neut)**2
438   continue
      do 439 neut = lstart, lend
        dis(neut) = q(neut)**2 + p(neut)*f(neut)
439   continue
      do 440 neut = lstart, lend
        if(m(neut).le.0 .and. dis(neut).gt.0.0) then
          dtausd(neut) = max((-q(neut)-
     *    sqrt(dis(neut)))/p(neut),zero)
          m(neut) = -2
        endif
440   continue
      do 441 neut = lstart, lend
        if(m(neut).eq. -2 .and. q(neut).lt.0.0 .and.
     *     dtausd(neut).le.1.0) then
          tx1(neut) = dtausd(neut)*r(neut) + tx(neut)
          ty1(neut) = dtausd(neut)*s(neut) + ty(neut)
          tz1(neut) = dtausd(neut)*t(neut) + tz(neut)
          m(neut) = -3
        endif
441   continue
      do 442 neut = lstart, lend
        if(m(neut).eq.-3.and.tz1(neut).le.
     *  xx(2,iar(neut)).and.tz1(neut).ge.
     *  xx(3,iar(neut)).and.sgn(neut)*tx1(neut).ge.
     *  -xx(7,iar(neut))) m(neut) = 1
442   continue
c****hemi-surface crossing check
      do 444 neut = lstart, lend
        if(m(neut).le.0 .and. sgn(neut)*tx(neut).le.
     *     -xx(7,iar(neut)) .and. r(neut).ne.0) then
          m(neut) = -4
          dtausd(neut) = -(sgn(neut)*xx(7,iar(neut))+
     *                    tx(neut))/r(neut)
        endif
444   continue
```

```
        do 445 neut = lstart, lend
          if(m(neut).eq.-4.and.dtausd(neut).ge.0.0.and.
     *    dtausd(neut) .le.1.0) then
            tx1(neut) = -sgn(neut)*xx(7,iar(neut))
            ty1(neut) = ty(neut) + dtausd(neut)*s(neut)
            tz1(neut) = tz(neut) + dtausd(neut)*t(neut)
            m(neut) = -5
          endif
  445   continue
        do 446 neut = lstart, lend
          if(m(neut).eq.-5.and.tz1(neut).le.
     *    xx(2,iar(neut)).and.tz1(neut).ge.
     *    xx(3,iar(neut)).and.tx1(neut)**2+ty1(neut)**2
     *   .le.xx(4,iar(neut))) m(neut) = 1
  446   continue
        go to 499
c*******end inner crossing check
c*******outer crossing check
c****end surface crossing check
  450   do 451 neut = lstart, lend
          if(t(neut).ne.0.0.and.tz1(neut).gt.
     *    xx(2,iar(neut)))then
            dtausd(neut) = (xx(2,iar(neut))-
     *                     tz(neut))/t(neut)
            tz1(neut) = xx(2,iar(neut))
            m(neut) = -1
          endif
  451   continue
        do 452 neut = lstart, lend
          if(t(neut).ne.0.0.and.tz1(neut).lt.
     *    xx(3,iar(neut)))then
            dtausd(neut) = (xx(3,iar(neut))-
     *                     tz(neut))/t(neut)
             tz1(neut) = xx(3,iar(neut))
            m(neut) = -1
          endif
  452   continue
        do 453 neut = lstart, lend
          if(m(neut).eq.-1) then
            tx1(neut) = dtausd(neut)*r(neut) + tx(neut)
            ty1(neut) = dtausd(neut)*s(neut) + ty(neut)
          endif
  453   continue
        do 454 neut = lstart, lend
          if(m(neut).eq.-1.and.tx1(neut)**2+ty1(neut)**2
     *    .le.xx(4,iar(neut)).and.sgn(neut)*tx1(neut)
     *    .ge.-xx(7,iar(neut))) m(neut) = 1
  454   continue
c****curved surface and hemi-surface crossing check
        do 461 neut = lstart, lend
          if(m(neut).le.0) then
            q(neut) = tx(neut)*r(neut)+ty(neut)*s(neut)
```

```
            f(neut) = xx(4,iar(neut)) - tx(neut)**2 -
     *                ty(neut)**2
            p(neut) = r(neut)**2 + s(neut)**2
          endif
  461   continue
        do 462 neut = lstart, lend
          if(m(neut).le.0) dis(neut) = q(neut)**2 +
     *                     p(neut)*f(neut)
  462   continue
        do 463 neut = lstart, lend
          if(m(neut).le.0 .and. dis(neut).gt.0.0)
     *    dtausd(neut) = max((-q(neut)+sqrt(dis(neut)))/
     *                   p(neut),zero)
  463   continue
        do 464 neut = lstart, lend
          if(m(neut).le.0) tx1(neut) = dtausd(neut)*
     *                     r(neut)+tx(neut)
  464   continue
        do 465 neut = lstart, lend
          if(m(neut).le.0 .and. sgn(neut)*tx1(neut).lt.
     *       -xx(7,iar(neut))) then
            dtausd(neut) = -(tx(neut)+sgn(neut)*
     *      xx(7,iar(neut)))/r(neut)
            tx1(neut) = -sgn(neut)*xx(7,iar(neut))
          endif
  465   continue
        do 466 neut = lstart, lend
          if(m(neut).le.0) then
            ty1(neut) = ty(neut) + dtausd(neut)*s(neut)
            tz1(neut) = tz(neut) + dtausd(neut)*t(neut)
            m(neut) = 1
          endif
  466   continue
c*******end outward crossing
  499 go to num, (488,489,490)
  488   continue
        if(lxhcyz) then
        do 481 neut = lstart, lend
          if(m(neut).eq.1 .and. nhemi(neut).eq.3) then
            x(ncros(neut)) = tz1(neut)
            y(ncros(neut)) = ty1(neut) + xx(5,iar(neut))
            z(ncros(neut)) = tx1(neut) + xx(6,iar(neut))
            etausd(ncros(neut)) = dtausd(neut)
          endif
  481   continue
        endif
        if(lxhcyy) then
        do 482 neut = lstart, lend
          if(m(neut).eq.1 .and. nhemi(neut).eq.4) then
            x(ncros(neut)) = tz1(neut)
            y(ncros(neut)) = tx1(neut) + xx(5,iar(neut))
            z(ncros(neut)) = ty1(neut) + xx(6,iar(neut))
```

```
            etausd(ncros(neut)) = dtausd(neut)
          endif
  482     continue
          endif
          go to 999
  489     continue
          if(lyhcyz) then
          do 483 neut = lstart, lend
            if(m(neut).eq.1 .and. nhemi(neut).eq.5) then
              x(ncros(neut)) = ty1(neut) + xx(5,iar(neut))
              y(ncros(neut)) = tz1(neut)
              z(ncros(neut)) = tx1(neut) + xx(6,iar(neut))
              etausd(ncros(neut)) = dtausd(neut)
            endif
  483     continue
          endif
          if(lyhcyx) then
          do 484 neut = lstart, lend
            if(m(neut).eq.1 .and. nhemi(neut).eq.6) then
              x(ncros(neut)) = tx1(neut) + xx(5,iar(neut))
              y(ncros(neut)) = tz1(neut)
              z(ncros(neut)) = ty1(neut) + xx(6,iar(neut))
              etausd(ncros(neut)) = dtausd(neut)
            endif
  484     continue
          endif
        go to 999
  490     continue
          if(lzhcyx) then
          do 485 neut = lstart, lend
            if(m(neut).eq.1 .and. nhemi(neut).eq.1) then
              x(ncros(neut)) = tx1(neut) + xx(5,iar(neut))
              y(ncros(neut)) = ty1(neut) + xx(6,iar(neut))
              z(ncros(neut)) = tz1(neut)
              etausd(ncros(neut)) = dtausd(neut)
            endif
  485     continue
          endif
          if(lzhcyy) then
          do 486 neut = lstart, lend
            if(m(neut).eq.1 .and. nhemi(neut).eq.2) then
              x(ncros(neut)) = ty1(neut) + xx(5,iar(neut))
              y(ncros(neut)) = tx1(neut) + xx(6,iar(neut))
              z(ncros(neut)) = tz1(neut)
              etausd(ncros(neut)) = dtausd(neut)
            endif
  486     continue
          endif
c*******************end hemicylinder package************
  999   continue
        return
        end
```

Appendix C

Input Listing of the Benchmark Problems

```
=csas25
sample problem 1  case 2c8 bare
27group inf
uranium  1 den=18.76 1 293 92235 93.2 92238 5.6 92234
1.0 92236 0.2 end
end comp
sample problem 1  case 2c8 bare
read parameters   flx=yes fdn=yes far=yes  end parameters
read geometry
cylinder 1 1 5.748 5.3825 -5.3825
cuboid  0 1 6.87 -6.87 6.87 -6.87 6.505 -6.505
end geometry
read array  nux=2 nuy=2 nuz=2  end array
end data
end
=csas25
problem 2  2c8 bare with 8 unit types matrix calculation
27group inf
uranium  1 den=18.76 1 293 92235 93.2 92238 5.6 92234
1.0 92236 0.2 end
end comp
problem 2  2c8 bare with 8 unit types matrix calculation
read param              flx=yes fdn=yes
mku=yes fmu=yes mkp=yes fmp=yes
end param  read geometry  unit 1
cylinder  1 1 5.748 5.3825 -5.3825
cuboid  0 1 6.87 -6.87 6.87 -6.87 6.505 -6.505
unit 2
cylinder  1 1 5.748 5.3825 -5.3825
cuboid  0 1 6.87 -6.87 6.87 -6.87 6.505 -6.505
unit 3  cylinder  1 1 5.748 5.3825 -5.3825 cuboid  0 1
6.87 -6.87 6.87  -6.87 6.505 -6.505
box type 4  cylinder  1 1 5.748 5.3825 -5.3825
cuboid  0 1 6.87 -6.87 6.87 -6.87 6.505 -6.505
box type  5
cylinder  1 1 5.748 5.3825 -5.3825
cuboid  0 1 6.87 -6.87 6.87 -6.87 6.505 -6.505
unit 6
cylinder  1 1 5.748 5.3825 -5.3825  cuboid  0 1 6.87 -6.87
6.87 -6.87  6.505 -6.505 unit 7  cylinder  1 1 5.748
5.3825 -5.3825
cuboid  0 1 6.87 -6.87 6.87 -6.87 6.505 -6.505
unit 8
cylinder  1 1 5.748 5.3825 -5.3825
cuboid  0 1 6.87 -6.87 6.87 -6.87 6.505 -6.505 end geom
read array  nux=2 nuy=2 nuz=2  loop  10*1 3*2 7*1  3 1 1
1 2 2 1 1 1 1  4 2 2 1 2 2 1 1 1 1  5 6*1 2 2 1  6 2 2 1
1 1 1 2 2 1 7  1 1 1 2 2 1 2 2 1  8 2 2 1 2 2 1 2 2 1
```

```
end array
end data
end
=csas25
sample problem 3  2c8  15.24 cm paraffin refl
27group inf
uranium  1 den=18.76 1 293 92235 93.2 92238 5.6 92234
1.0 92236 0.2 end
para(h2o) 2 end
end comp
sample problem 3  2c8  15.24 cm paraffin refl
read param     flx=yes fdn=yes        pwt=yes end param
read array nux=2 nuy=2 nuz=2 end array
read geom
cylinder  1 1 5.748 5.3825 -5.3825
cuboid  0 1 11.74 -11.74 11.74 -11.74 11.375 -11.375
core  0 1 -23.48 -23.48 -22.75
cuboid  2 2 26.48 -26.48 26.48 -26.48 25.75 -25.75
cuboid  2 3 29.48 -29.48 29.48 -29.48 28.75 -28.75
cuboid  2 4 32.48 -32.48 32.48 -32.48 31.75 -31.75
cuboid  2 5 35.48 -35.48 35.48 -35.48 34.75 -34.75
cuboid  2 6 38.72 -38.72 38.72 -38.72 37.99 -37.99
end geom
read bias  id=400 2 6  end bias  end data
end
=csas25
sample problem 4  2c8 15.24 cm paraffin refl automatic refl
27group inf
uranium  1 den=18.76 1 293 92235 93.2 92238 5.6 92234 1.0
92236 0.2 end
para(h2o) 2 end
end comp
sample problem 4  2c8 15.24 cm paraffin refl automatic refl
read param    pwt=yes          flx=yes fdn=yes end param
read geometry  cylinder  1 1 5.748 5.3825 -5.3825
cuboid  0 1 11.74 -11.74 11.74 -11.74 11.375 -11.375
core  0 1 -23.48 -23.48 -22.75
reflector  2 2 6*3.0 5
reflector  2 7 6*.24 1
end geom
read arra  nux=2 nuy=2 nuz=2 end array
read bias  id=400 2 7  end bias
end data
end
=csas25
sample problem 5  2c8 12 inch paraffin albedo reflector
27group inf
uranium  1 den=18.76 1 293 92235 93.2 92238 5.6 92234
1.0 92236 0.2 end
end comp
sample problem 5  2c8 12 inch paraffin albedo reflector
read para
```

```
 flx=yes far=yes fdn=yes rnd=e85631374579    end para
read array  nux=2 nuy=2 nuz=2 end array
read bounds  all=paraffin  end bounds
read geom  cylinder  1 1 5.748 5.3825 -5.3825
cuboid  0 1 11.74 -11.74 11.74 -11.74 11.375 -11.375
end geom
end data
end
=csas25
sample problem 6  one 2c8 unit (single unit)
27group inf
uranium  1 den=18.76 1 293 92235 93.2 92238 5.6 92234
1.0 92236 0.2 end
end comp
sample problem 6  one 2c8 unit (single unit)
read para   flx=yes fdn=yes far=yes         end para
read geometry  cylinder  1 1 5.748 5.3825 -5.3825
end geometry  end data
end
=csas25
sample problem 7  bare 2c8 using specular reflection
27group inf
uranium  1 den=18.76 1 293 92235 93.2 92238 5.6 92234
1.0 92236 0.2 end
end comp
sample problem 7  bare 2c8 using specular reflection
read para      flx=yes fdn=yes  far=yes  end parameters
read geom  cylinder  1 1 5.748 5.3825 -5.3825
cuboid  0 1 6.87 -6.87 6.87 -6.87 6.505 -6.505
end geom
read bounds +fc=specular end bounds  end data
end
=csas25
sample problem 8  infinitely long cylinder from 2c8 unit
27group inf
uranium  1 den=18.76 1 293 92235 93.2 92238 5.6 92234
1.0 92236 0.2 end
end comp
sample problem 8  infinitely long cylinder from 2c8 unit
read param                        end param
read geometry  cylinder  1 1 5.748 10.0 -10.0
cuboid  0 1 6.87 -6.87 6.87 -6.87 10.0 -10.0
end geometry
read bounds zfc=mirror  end bounds
end data
end
=csas25
sample problem 9  infinite array of 2c8 units
27group inf
uranium  1 den=18.76 1 293 92235 93.2 92238 5.6 92234
1.0 92236 0.2 end
end comp
```

```
sample problem 9  infinite array of 2c8 units
read param  rnd=e85631374579     gen=33 end param
read boun  all=mir  end boun
read geom
cylinder  1 1 5.748 5.3825 -5.3825  cuboid  0 1 6.87
-6.87 6.87 -6.87 6.505 -6.505  end geom  end data
end
=csas25
sample problem 10  case 2c8 bare  write restart
27group inf
uranium  1 den=18.76 1 293 92235 93.2 92238 5.6 92234
1.0 92236 0.2 end
end comp
sample problem 10  case 2c8 bare  write restart
read parameters  flx=yes fdn=yes far=yes  res=5 wrs=94
end parameters
read geometry
cylinder 1 1 5.748 5.3825 -5.3825
cuboid  0 1 6.87 -6.87 6.87 -6.87 6.505 -6.505
end geometry
read array  nux=2 nuy=2 nuz=2  end array
end data
end
=kenova
sample problem 11  2c8 bare  read restart data
read param   beg=51           rst=94 res=0 end param
end data
end
=csas25
sample problem 12 4 aqueous 4 metal mixed units
27group inf
uranium  1 den=18.76 1 293 92235 93.2 92238 5.6 92234
1.0 92236 0.2 end
solnuo2(no3)2  2 415 9.783-3 spg=1.555 1.0 293 92235
92.6 92238 5.9 92234 1.0  92236 0.5 end
plexiglas 3 end
end comp
sample problem 12 4 aqueous 4 metal mixed units
read param  flx=yes fdn=yes    nub=yes smu=yes mkp=yes
mku=yes fmp=yes fmu=yes end param
read geom
box type 1
cylinder  2 1 9.525 8.89 -8.89
cylinder  3 1 10.16 9.525 -9.525
cuboid  0 1 10.875 -10.875 10.875 -10.875 10.24 -10.24
box type  2
cylinder  1 1 5.748 5.3825 -5.3825
cuboid  0 1 6.59 -15.16 6.59 -15.16 6.225 -14.255
box type  3
cylinder  1 1 5.748 5.3825 -5.3825
cuboid  0 1 6.59 -15.16 15.16 -6.59 6.225 -14.255
box type  4
```

```
cylinder  1 1 5.748 5.3825 -5.3825
cuboid  0 1 6.59 -15.16 6.59 -15.16 14.255 -6.225
box type  5
cylinder  1 1 5.748 5.3825 -5.3825
cuboid  0 1 6.59 -15.16 15.16 -6.59 14.255 -6.225
end geom  read array  nux=2 nuy=2 nuz=2  loop
1 3r2 1 2 1 1 2 1  2 9r1  3 3r1 2 2 1 3r1
4 6r1 2 2 1  5 3r1 2 2 1 2 2 1
end array
end data
end
=csas25
sample problem 13  two cuboids in a cylindrical annulus
27group inf
uranium  1 den=18.69 1 293  92235 93.2 92238 5.6 92234
1.0 92236 0.2 end
end comp
sample problem 13  two cuboids in a cylindrical annulus
read param              end param
read geom
unit 1
cuboid  1 1 6.35 -6.35 6.35 -6.35 7.62 0.0
cylinder  0 1 13.97 7.62 0.0 orig -6.0934 0.0
cylinder  1 1 19.05 7.62 0.0 orig -6.0934 0.0
cuboid  0 1 12.9566 -25.1434 19.05 -19.05 7.62 0.0
unit 2
cuboid  1 1 6.35 -6.35 6.35 -6.35 8.56 0.0
cylinder  0 1 13.97 8.56 0.0 origin 6.0934 0.0
cylinder  1 1 19.05 8.56 0.0 origin 6.0934 0.0
cuboid  0 1 25.1434 -12.9566 19.05 -19.05 8.56 0.0
unit 3
cuboid  1 1 6.35 -6.35 6.35 -6.35 2.616 0.0
cuboid  0 1 25.1434 -12.9566 19.05 -19.05 2.616 0.0
end geom
read array nux=1 nuy=1 nuz=3  fill 1 2 3 t end array
end data
end
=csas25
sample problem 14  u metal cylinder in an annulus
27group inf
uranium  1 den=18.69 1 293  92235 93.2 92238 5.6 92234
1.0 92236 0.2 end
end comp
sample problem 14  u metal cylinder in an annulus
read param                  end param
read geom
cylinder  1 1 8.89 10.109 0.0 orig 5.0799 0.0
cylinder  0 1 13.97 10.109 0.0
cylinder  1 1 19.05 10.109 0.0
end geom
end data
end
```

```
=csas25
problem 15  small water reflected sphere on plexiglas
collar
27group inf
uranium  1 den=18.794 1 293 92235 97.67 92238 0.98
         92234 1.17  92236 0.20 end
plexiglas 2 end
h2o 3 end
end comp
problem 15  small water reflected sphere on plexiglas
collar
read param rnd=e85631374579        flx=yes fdn=yes end
param
read geom
unit 1
hemisphe-z 1 1 6.5537 chord -5.09066
cylinder  3 1 4.1275 -5.09066 -7.63065
cylinder  2 1 12.7   -5.09066 -7.63065
cuboid  3 1 4p12.7 -5.09066 -7.63065
unit 2
hemisphe+z 1 1 6.5537 chord 5.09066
cuboid  3 1 4p12.7 6.5537 -5.09066
core  0 1 -12.7 -12.7 -7.092175
cylinder  3 1 17.97 2p7.0922
replicate 3 2 3*3.0 5
end geom
read bias  id=500 2 6 end bias
read array nux=1 nuy=1 nuz=2  fill 1 2 end array
read plot ttl='x-z slice through the center of the sphere'
lpi=6
xul=-20.0 zul=10.0 yul=0.0  xlr=20.0 ylr=0.0 zlr=-10.0
uax=1.0 wdn=-1.0 nax=130    nch=' *0-'
end plot
end data
end
=csas25
sample problem 16 uo2f2 infinite slab k-infinity
27group inf
solnuo2f2       1 578.7 0 1.0 293 92235 93.2 92238 6.8 end
solnuo2f2       3 578.7 0 1.0 293 92235 93.2 92238 6.8 end
pyrex 2 end
boron           3 den=.0266 end
end comp
sample problem 16 uo2f2 infinite slab k-infinity
read parameters   amx=yes xap=no     end parameters
read geometry
cuboid 1 1 2.479 -2.479 100 -100 100 -100
cuboid 2 1 3.749 -3.749 100 -100 100 -100
cuboid 3 1 17.479 -17.479 100 -100 100 -100
end geom
read bounds  all=mirror  end bounds    read array  end
array
```

```
end data
end
=csas25
problem 17 93% uo2f2 solution sphere  adjoint calculation
27group inf
solnuo2f2  1 133 0 1.0 293 92235 93.0 92238 7 end
end comp
problem 17 93% uo2f2 solution sphere  adjoint calculation
read parameters  nbk=600    adj=yes amx=yes xap=no
end parameters
read geometry
sphere 1 1 16.0
end geom
end data
end
=csas25
sample problem 18   1f27 demonstration of options problem
27group inf
solnuo2(no3)2  1 415 9.783-3 spg=1.555 1.0 293 92235
92.6 92238 5.9
         92234 1.0  92236 0.5 end
plexiglas 2 end
para(h2o) 3 end
h2o 4 1-9 end
end comp
sample problem 18   1f27 demonstration of options problem
read para   gen=53 npg=350 fdn=yes nub=yes
  mku=yes fmu=yes mkh=yes fmh=yes mka=yes fma=yes
  rnd=e95631374579
  pwt=yes far=yes flx=yes amx=yes pax=yes pgm=yes
 end para
read bounds   -zb= h2o   end bounds
read geom unit 1 cylinder 1 1 9.52 8.7804 -8.7804
 cylinder 0 1 9.52 8.9896 -8.7804
 cylinder 2 1 10.16 9.6296 -9.4204
 cuboid 4 1 18.45 -18.45 18.45 -18.45 17.8946 -17.6854
 unit 2 array 1 3*0.0
 unit 3 array 2 3*0.0
 unit 4 array 3 3*0.0
 unit 5 array 4 3*0.0
 global
 unit 6 cuboid 4 1 55.3501 -55.3501 55.3501 -55.3501
53.3701 -53.3701
  hole 2 -55.35   -18.45   -17.79
  hole 3 -55.35   -18.45   -53.3701
  hole 4  18.4501 -18.45   -53.3701
  hole 5 -55.3501 -55.3501 -53.3701
 replicate 3 2 6*3 5  replicate 3 7 6*0.24 1  end geom
 read bias  id=400 2 7  end bias
read array
 ara=1 nux=2 nuy=2 nuz=2 fill f1 end fill
 ara=2 nux=2 nuy=2 nuz=1 fill f1 end fill
```

```
 ara=3 nux=1 nuy=2 nuz=3 fill f1 end fill
 ara=4 nux=3 nuy=1 nuz=3 fill f1 end fill
 end array
 read start nst=6 tfx=0.0 tfy=0.0 tfz=0.0
 lnu=350 ps6=yes
 end start
read plot          plt=yes lpi=6
ttl=?  1f27 xy plot at z=0.0 ?  xul=-71.0 yul=71.0 zul=0.0
 xlr=71.0 ylr=-71.0 zlr=0.0 uax=1 vdn=-1 nax=130 nch=?.*-3
?
 run=yes end      ttl=?unit map 1f27 xy plot at z=0.0?
 pic=unit nch=? 123456? end plot    end data
end
=csas25
problem 19 4 aqueous 4 metal array of arrays (samp prob 12)
27group inf
uranium  1 den=18.76 1 293 92235 93.2 92238 5.6 92234
1.0 92236 0.2 end
solnuo2(no3)2  2 415 9.783-3 spg=1.555 1.0 293 92235
92.6 92238 5.9
   92234 1.0  92236 0.5 end
plexiglas 3 end
end comp
problem 19 4 aqueous 4 metal array of arrays (samp prob 12)
read param    flx=yes fdn=yes    nub=yes smu=yes mkp=yes
mku=yes fmp=yes fmu=yes end param
read geom
unit 1
com='uranyl nitrate solution in a plexiglas container'
cylinder  2 1 9.525 2p8.89
cylinder  3 1 10.16 2p9.525
cuboid  0 1 4p10.875 2p10.24
unit 2
com='uranium metal cylinder'
cylinder  1 1 5.748 2p5.3825
cuboid  0 1 4p6.59 2p6.225
unit 3
com='1x2x2 array of solution units'
array 1 3*0.0
unit 4
com='1x2x2 array of metal units padded to match solution
array'
array 2 3*0.0
replicate 0 1 2*0.0 2*8.57 2*8.03 1
end geom
read array ara=1 nux=1 nuy=2 nuz=2 fill f1 end fill
ara=2 nux=1 nuy=2 nuz=2 fill f2 end fill gbl=3 ara=3 nux=2
nuy=1 nuz=1
com='composite array of solution and metal units'
fill 4 3 end fill
end array
end data
```

```
end
=csas25
sample problem 20 triangular pitched array
27group inf
solnuo2f2  1 576.87 0 1.0 293 92235 93.2 92238 6.8 end
al 2 end
end comp
sample problem 20 triangular pitched array
read param              end param
read geom
unit 1
cylinder    1 1 10.16 18.288 0
cylinder    2 1 10.312 18.288 -.152
unit 2
cuboid      0 1 4p50 50 -.152
hole        1 3r0
hole        1 21.006 2r0
hole        1 -21.006 2r0
hole        1 10.503 18.192 0
hole        1 -10.503 18.192 0
hole        1 10.503 -18.192 0
hole        1 -10.503 -18.192 0
end geom
read array nux=1 nuy=1 nuz=1 fill 2 end fill end array
read plot ttl='hex array' pic=mix lpi=6
xul=0 yul=100 zul=10
xlr=100 ylr=0 zlr=10 uax=1 vdn=-1 nax=120
nch=' 12' end plot
end data
end
=csas25
sample problem 21  partially filled sphere
27group inf
solnuo2f2  1 494 0 spg= 1.56 1 293 92235 4.89 92238
95.09 92234 0.02 end
al 2 end
end comp
sample problem 21  partially filled sphere
read param rnd=e85631374579     end param
read geom
hemisphe-z  1 1 34.6   chord 30.
sphere      0 1 34.6
sphere      2 1 34.759
end geom
end data
end
=csas25
sample problem 22   case 2c8 bare with 3 nested holes,
each is equal volume
27group inf
uranium  1 den=18.76 1 293 92235 93.2 92238 5.6 92234
1.0 92236 0.2 end
```

```
end comp
sample problem 22   case 2c8 bare with 3 nested holes,
each is equal volume
read parameters   flx=yes fdn=yes far=yes  end parameters
read geometry

unit 1
cylinder 1 1 3.621 2p3.3907

unit 2
cylinder 1 1 4.5622 2p4.2721
hole 1 3*0.0

unit 3
cylinder 1 1 5.2224 2p4.8903
hole 2 3*0.0

unit 4
cylinder 1 1 5.748 5.3825 -5.3825
hole 3 3*0.0
cuboid  0 1 6.87 -6.87 6.87 -6.87 6.505 -6.505

end geometry
read array  nux=2 nuy=2 nuz=2 fill f4 end fill end array
end data
end
=csas25
sample problem 23  case 2c8 bare as mixed zhemicylinders
27group inf
uranium  1 den=18.76 1 293 92235 93.2 92238 5.6 92234
1.0 92236 0.2 end
end comp
sample problem 23  case 2c8 bare as mixed zhemicylinders
read parameters   fdn=yes           end parameters
read geometry
unit 1
com='-x half of unit 3'
zhemicyl-x 1 1 5.748 5.3825 -5.3825
cuboid  0 1 0.0  -6.87 6.87 -6.87 6.505 -6.505
unit 2
com='+x half of unit 3'
zhemicyl+x 1 1 5.748 5.3825 -5.3825
cuboid  0 1 6.87  0.0  6.87 -6.87 6.505 -6.505
unit 3
com='cylinder composed of equal halves (zhemicylinders with

x radii)'
array 1 3*0.0
unit 4
com='-x portion (more than half) of unit 6'
zhemicyl-x 1 1 5.748 5.3825 -5.3825 chord 3.0
cuboid  0 1 3.0  -6.87 6.87 -6.87 6.505 -6.505
```

```
unit 5
com='+x portion (less than half) of unit 6'
zhemicyl+x 1 1 5.748 5.3825 -5.3825 chord -3.0
cuboid  0 1 6.87  3.0  6.87 -6.87 6.505 -6.505
unit 6
com='cylinder composed of unequal halves (zhemicylinders
with x radii)'
array 2 3*0.0
unit 7
com='cylinder of a single zhemicylinder in the -x
direction'
zhemicyl-x 1 1 5.748 5.3825 -5.3825 chord 5.748
cuboid  0 1 6.87 -6.87 6.87 -6.87 6.505 -6.505
unit 8
com='cylinder of a single zhemicylinder in the +x
direction'
zhemicyl+x 1 1 5.748 5.3825 -5.3825 chord 5.748
cuboid  0 1 6.87 -6.87 6.87 -6.87 6.505 -6.505
unit 9
com='-y half of unit 11'
zhemicyl-y 1 1 5.748 5.3825 -5.3825
cuboid  0 1 6.87 -6.87 0.0  -6.87 6.505 -6.505
unit 10
com='+y half of unit 11'
zhemicyl+y 1 1 5.748 5.3825 -5.3825
cuboid  0 1 6.87 -6.87 6.87  0.0  6.505 -6.505
unit 11
com='cylinder composed of equal halves (zhemicylinders with

z radii)'
array 3 3*0.0
unit 12
com='-y portion (more than half) of unit 14'
zhemicyl-y 1 1 5.748 5.3825 -5.3825 chord 3.0
cuboid  0 1 6.87 -6.87 3.0  -6.87 6.505 -6.505
unit 13
com='+y portion (less than half) of unit 14'
zhemicyl+y 1 1 5.748 5.3825 -5.3825 chord -3.0
cuboid  0 1 6.87 -6.87 6.87  3.0  6.505 -6.505
unit 14
com='cylinder composed of unequal halves (zhemicylinders
with z radii)'
array 4 3*0.0
unit 15
com='cylinder of a single zhemicylinder in the -y
direction'
zhemicyl-y 1 1 5.748 5.3825 -5.3825 chord 5.748
cuboid  0 1 6.87 -6.87 6.87 -6.87 6.505 -6.505
unit 16
com='cylinder of a single zhemicylinder in the +y'
zhemicyl+y 1 1 5.748 5.3825 -5.3825 chord 5.748
cuboid  0 1 6.87 -6.87 6.87 -6.87 6.505 -6.505
```

```
end geometry
read array
com='array 1 defines unit 3 (zhemicylinders with x radii)'
ara=1 nux=2 nuy=1 nuz=1 fill 1 2 end fill
com='array 2 defines unit 6 (zhemicylinders with x radii)'
ara=2 nux=2 nuy=1 nuz=1 fill 4 5 end fill
com='array 3 defines unit 11 (zhemicylinders with y radii)'
ara=3 nux=1 nuy=2 nuz=1 fill 9 10 end fill
com='array 4 defines unit 14 (zhemicylinders with y radii)'
ara=4 nux=1 nuy=2 nuz=1 fill 12 13 end fill
com='array 5 defines the total 2c8 problem'
gbl=5 ara=5 nux=2 nuy=2 nuz=2 fill 3 7 6 8 11 15 14 16
end fill
end array
end data
end
=csas25
sample problem 24  case 2c8 bare as mixed xhemicylinders
27group inf
uranium  1 den=18.76 1 293 92235 93.2 92238 5.6 92234
1.0 92236 0.2 end
end comp
sample problem 24  case 2c8 bare as mixed xhemicylinders
read parameters   fdn=yes            end parameters
read geometry
unit 1
com='-y half of unit 3'
xhemicyl-y 1 1 5.748 5.3825 -5.3825
cuboid  0 1 6.505 -6.505 0.0 -6.87 6.87 -6.87
unit 2
com='+y half of unit 3'
xhemicyl+y 1 1 5.748 5.3825 -5.3825
cuboid  0 1 6.505 -6.505 6.87 0.0  6.87 -6.87
unit 3
com='cylinder composed of equal halves (xhemicylinders with

y radii)'
array 1 3*0.0
unit 4
com='-y portion (more than half) of unit 6'
xhemicyl-y 1 1 5.748 5.3825 -5.3825 chord 3.0
cuboid  0 1 6.505 -6.505 3.0 -6.87 6.87 -6.87
unit 5
com='+y portion (less than half) of unit 6'
xhemicyl+y 1 1 5.748 5.3825 -5.3825 chord -3.0
cuboid  0 1 6.505 -6.505 6.87 3.0  6.87 -6.87
unit 6
com='cylinder composed of unequal halves (xhemicylinders
with y radii)'
array 2 3*0.0
unit 7
com='cylinder of a single xhemicylinder in the -y
```

```
direction'
xhemicyl-y 1 1 5.748 5.3825 -5.3825 chord 5.748
cuboid  0 1 6.505 -6.505 6.87 -6.87 6.87 -6.87
unit 8
com='cylinder of a single xhemicylinder in the +y
direction'
xhemicyl+y 1 1 5.748 5.3825 -5.3825 chord 5.748
cuboid  0 1 6.505 -6.505 6.87 -6.87 6.87 -6.87
unit 9
com='-z half of unit 11'
xhemicyl-z 1 1 5.748 5.3825 -5.3825
cuboid  0 1 6.505 -6.505 6.87 -6.87 0.0 -6.87
unit 10
com='+z half of unit 11'
xhemicyl+z 1 1 5.748 5.3825 -5.3825
cuboid  0 1 6.505 -6.505 6.87 -6.87 6.87 0.0
unit 11
com='cylinder composed of equal halves (xhemicylinders with

z radii)'
array 3 3*0.0
unit 12
com='-z portion (more than half) of unit 14'
xhemicyl-z 1 1 5.748 5.3825 -5.3825 chord 3.0
cuboid  0 1 6.505 -6.505 6.87 -6.87 3.0 -6.87
unit 13
com='+z portion (less than half) of unit 14'
xhemicyl+z 1 1 5.748 5.3825 -5.3825 chord -3.0
cuboid  0 1 6.505 -6.505 6.87 -6.87 6.87 3.0
unit 14
com='cylinder composed of unequal halves (xhemicylinders
with z radii)'
array 4 3*0.0
unit 15
com='cylinder of a single xhemicylinder in the -z
direction'
xhemicyl-z 1 1 5.748 5.3825 -5.3825 chord 5.748
cuboid  0 1 6.505 -6.505 6.87 -6.87 6.87 -6.87
unit 16
com='cylinder of a single xhemicylinder in the +z
direction'
xhemicyl+z 1 1 5.748 5.3825 -5.3825 chord 5.748
cuboid  0 1 6.505 -6.505 6.87 -6.87 6.87 -6.87
end geometry
read array
com='array 1 defines unit 3 (xhemicylinders with y radii)'
ara=1 nux=1 nuy=2 nuz=1 fill 1 2 end fill
com='array 2 defines unit 6 (xhemicylinders with y radii)'
ara=2 nux=1 nuy=2 nuz=1 fill 4 5 end fill
com='array 3 defines unit 11 (xhemicylinders with z radii)'
ara=3 nux=1 nuy=1 nuz=2 fill 9 10 end fill
com='array 4 defines unit 14 (xhemicylinders with z radii)'
```

```
ara=4 nux=1 nuy=1 nuz=2 fill 12 13 end fill
com='array 5 defines the total 2c8 problem'
gbl=5 ara=5 nux=2 nuy=2 nuz=2 fill 3 7 6 8 11 15 14 16
end fill
end array
end data
end
=csas25
sample problem 25  case 2c8 bare as mixed yhemicylinders
27group inf
uranium  1 den=18.76 1 293 92235 93.2 92238 5.6 92234
1.0 92236 0.2 end
end comp
sample problem 25  case 2c8 bare as mixed yhemicylinders
read parameters              fdn=yes  end parameters
read geometry
unit 1
com='-x half of unit 3'
yhemicyl-x 1 1 5.748 5.3825 -5.3825
cuboid  0 1 0.0  -6.87 6.505 -6.505 6.87 -6.87
unit 2
com='+x half of unit 3'
yhemicyl+x 1 1 5.748 5.3825 -5.3825
cuboid  0 1 6.87  0.0  6.505 -6.505 6.87 -6.87
unit 3
com='cylinder composed of equal halves (yhemicylinders with

x radii)'
array 1 3*0.0
unit 4
com='-x portion (more than half) of unit 6'
yhemicyl-x 1 1 5.748 5.3825 -5.3825 chord 3.0
cuboid  0 1 3.0  -6.87 6.505 -6.505 6.87 -6.87
unit 5
com='+x portion (less than half) of unit 6'
yhemicyl+x 1 1 5.748 5.3825 -5.3825 chord -3.0
cuboid  0 1 6.87  3.0  6.505 -6.505 6.87 -6.87
unit 6
com='cylinder composed of unequal halves (yhemicylinders
with x radii)'
array 2 3*0.0
unit 7
com='cylinder of a single yhemicylinder in the -x
direction'
yhemicyl-x 1 1 5.748 5.3825 -5.3825 chord 5.748
cuboid  0 1 6.87 -6.87 6.505 -6.505 6.87 -6.87
unit 8
com='cylinder of a single yhemicylinder in the +x
direction'
yhemicyl+x 1 1 5.748 5.3825 -5.3825 chord 5.748
cuboid  0 1 6.87 -6.87 6.505 -6.505 6.87 -6.87
unit 9
```

```
com='-z half of unit 11'
yhemicyl-z 1 1 5.748 5.3825 -5.3825
cuboid  0 1 6.87 -6.87 6.505 -6.505 0.0 -6.87
unit 10
com='+z half of unit 11'
yhemicyl+z 1 1 5.748 5.3825 -5.3825
cuboid  0 1 6.87 -6.87 6.505 -6.505 6.87 0.0
unit 11
com='cylinder composed of equal halves (yhemicylinders with

z radii)'
array 3 3*0.0
unit 12
com='-z portion (more than half) of unit 14'
yhemicyl-z 1 1 5.748 5.3825 -5.3825 chord 3.0
cuboid  0 1 6.87 -6.87 6.505 -6.505 3.0 -6.87
unit 13
com='+z portion (less than half) of unit 14'
yhemicyl+z 1 1 5.748 5.3825 -5.3825 chord -3.0
cuboid  0 1 6.87 -6.87 6.505 -6.505 6.87 3.0
unit 14
com='cylinder composed of unequal halves (yhemicylinders
with z radii)'
array 4 3*0.0
unit 15
com='cylinder of a single yhemicylinder in the -z
direction'
yhemicyl-z 1 1 5.748 5.3825 -5.3825 chord 5.748
cuboid  0 1 6.87 -6.87 6.505 -6.505 6.87 -6.87
unit 16
com='cylinder of a single yhemicylinder in the +z
direction'
yhemicyl+z 1 1 5.748 5.3825 -5.3825 chord 5.748
cuboid  0 1 6.87 -6.87 6.505 -6.505 6.87 -6.87
end geometry
read array
com='array 1 defines unit 3 (yhemicylinders with x radii)'
ara=1 nux=2 nuy=1 nuz=1 fill 1 2 end fill
com='array 2 defines unit 6 (yhemicylinders with x radii)'
ara=2 nux=2 nuy=1 nuz=1 fill 4 5 end fill
com='array 3 defines unit 11 (yhemicylinders with z radii)'
ara=3 nux=1 nuy=1 nuz=2 fill 9 10 end fill
com='array 4 defines unit 14 (zhemicylinders with z radii)'
ara=4 nux=1 nuy=1 nuz=2 fill 12 13 end fill
com='array 5 defines the total 2c8 problem'
gbl=5 ara=5 nux=2 nuy=2 nuz=2 fill 3 7 6 8 11 15 14 16
end fill
end array
end data
end
```

Appendix D

Input Listing of the Spent Fuel Storage Cask Problem

```
=csas25
material specifications for pwr castor v/21 cask
criticality
27groupndf4 latticecell
uo2 1 .96 293 92235 3.7 92238 96.3 end
zr 2 1.0 end
h2o 3 1.0 end
boron 4 .028 end
ss304 4 .972 end
ss304 5 1.0 end
arbm-ciron 7.150 3 0 0 0 6012 3.25 14000 1.35 26000 95.4 6
end
end comp
squarepitch 1.43 .9322 1 3 1.0718 2 .9498 0 end
case 1 pwr castor v/21 storage cask 4/85
fn=llnl.cntl(pwrcskv1)
read param flx=yes nub=yes fdn=yes plt=no npg=300 gen=103
end param
read geom
unit 1
com='geometry for fuel pin and spacing'
cylinder 1 1 .4661 174.88 -190.88
cylinder 0 1 .4749 190.88 -190.88
cylinder 2 1 .5359 190.88 -190.88
cylinder 5 1 .5359 2p192.53
cuboid   3 1 4p.715 2p192.53
unit 2
com='geometry for water box in fuel assm.'
cuboid 3 1 4p.715 2p192.53
unit 3
com='geometry for fuel asm.'
array 1 -10.725 -10.725 -192.53
cuboid 3 1 4p11.0 2p192.54
unit 4
cuboid 4 1 2p.5 2p11.0 2p192.54
unit 5
cuboid 4 1 2p11.0 2p.5 2p192.54
unit 6
cuboid 4 1 2p0.5 2p23.5 2p192.54
unit 7
cuboid 4 1 2p23.0 2p0.5 2p192.54
unit 8
cuboid 4 1 2p11.5 2p.25 2p192.54
unit 9
cuboid 4 1 2p.25 2p11.0 2p192.54
unit 10
cuboid 4 1 2p1.0 2p31.6 2p192.54
unit 11
```

```
cuboid 4 1 2p30.6 2p1.0 2p192.54
global
unit 100
cylinder 3 1 76.2 2p193.05
hole 3    0.0      0.0      0.0
hole 3    -34.0    0.0      0.0
hole 3    -63.2    0.0      0.0
hole 4    -11.5    0.0      0.0
hole 4    -22.5    0.0      0.0
hole 4    -45.501  0.0      0.0
hole 4    -51.7    0.0      0.0
hole 3    0.0      -34.0    0.0
hole 3    0.0      -63.2    0.0
hole 5    0.0      -11.5    0.0
hole 5    0.0      -22.5    0.0
hole 5    0.0      -45.501  0.0
hole 5    0.0      -51.7    0.0
hole 3    -28.65   -28.65   0.0
hole 3    -28.65   -51.65   0.0
hole 3    -51.65   -28.65   0.0
hole 10   -12.0    -42.6    0.0
hole 11   -43.6    -12.0    0.0
hole 6    -17.15   -40.15   0.0
hole 7    -40.65   -17.15   0.0
hole 8    -51.65   -39.9    0.0
hole 9    -39.9    -51.65   0.0
hole 5    -28.65   -40.15   0.0
hole 5    -28.65   -63.15   0.0
hole 4    -40.15   -28.65   0.0
hole 4    -63.15   -28.65   0.0
hole 3    34.0     0.0      0.0
hole 3    63.2     0.0      0.0
hole 3    0.0      34.0     0.0
hole 3    0.0      63.2     0.0
hole 3    28.65    -28.65   0.0
hole 3    28.65    -51.65   0.0
hole 3    51.65    -28.65   0.0
hole 3    -28.65   28.65    0.0
hole 3    -28.65   51.65    0.0
hole 3    -51.65   28.65    0.0
hole 3    28.65    28.65    0.0
hole 3    28.65    51.65    0.0
hole 3    51.65    28.65    0.0
hole 4    11.5     0.0      0.0
hole 4    22.5     0.0      0.0
hole 4    45.501   0.0      0.0
hole 4    51.7     0.0      0.0
hole 4    40.15    -28.65   0.0
hole 4    63.15    -28.65   0.0
hole 4    -40.15   28.65    0.0
hole 4    -63.15   28.65    0.0
hole 4    40.15    28.65    0.0
```

```
hole 4    63.15    28.65    0.0
hole 5    0.0      11.5     0.0
hole 5    0.0      22.5     0.0
hole 5    0.0      45.501   0.0
hole 5    0.0      51.7     0.0
hole 5    28.65    -40.15   0.0
hole 5    28.65    -63.15   0.0
hole 5    -28.65   40.15    0.0
hole 5    -28.65   63.15    0.0
hole 5    28.65    40.15    0.0
hole 5    28.65    63.15    0.0
hole 6    17.15    -40.15   0.0
hole 6    -17.15   40.15    0.0
hole 6    17.15    40.15    0.0
hole 7    40.65    -17.15   0.0
hole 7    -40.65   17.15    0.0
hole 7    40.65    17.15    0.0
hole 8    51.65    -39.9    0.0
hole 8    -51.65   39.9     0.0
hole 8    51.65    39.9     0.0
hole 9    39.9     -51.65   0.0
hole 9    -39.9    51.65    0.0
hole 9    39.9     51.65    0.0
hole 10   12.0     -42.6    0.0
hole 10   -12.0    42.6     0.0
hole 10   12.0     42.6     0.0
hole 11   43.6     -12.0    0.0
hole 11   -43.6    12.0     0.0
hole 11   43.6     12.0     0.0
cylinder 3 1 76.35 2p207.7
cylinder 6 1 116.35 2p207.7
cylinder 5 1 116.35 2p244.3
end geom
read start
nst=1
xsm=53.0 xsp=-53.0 ysm=53.0 ysp=-53.0 zsm=174.0 zsp=-190.0
end start
read array
com='array 1 describes one 15x15 fuel assembly'
ara=1 nux=15 nuy=15 nuz=1 fill
32r1 2 2r1 2 3r1 2 2r1 2 9r1 2 11r1 2 5r1 2 6r1 2 9r1 2
20r1 2 3r1
2 1b112 end fill
end array
read plot ttl='x-y slice through y=0.0'
xul=-80.0 zul=0.0 yul=80.0 xlr=80.0 ylr=-80.0 zlr=0.0
uax=1.0 vdn=-1.0 nax=390 nch=' 12345678'
end plot
end data
end
```

Appendix E

Input Listing of the Spent Fuel Shipping Cask Problem

```
=kenova
'    ***** begin data for plane #1 *****
keno input for 31 element burnup credit cask, load
31.5gwd/mtu
read parm
run=yes gen=203 lib=41 npg=1000 nb8=500 tme=92 lng=300000
end parm
read mixt  sct=3
mix=1
  40302 4.251-2
mix=4
  9824304  1.743-2
  9825055  1.736-3
  9826304  5.936-2
  9828304  7.721-3
mix=5
  9824304  1.691-2
  9825055  1.684-3
  9826304  5.758-2
  9828304  7.489-3
  985010   7.836-4
  985011   3.181-3
mix=6
  9826304  7.356-2
  986012   1.166-2
  9814028  2.07-3
mix=7
  981001  6.675-2
  988016  3.338-2
'  buc nuclides -  lumped fission product
'  isotopics for na1c5, burnup=23 gwd/mtu
mix=109
  3092234  3.8385e-06
  3092235  2.3828e-04
  3092236  9.1495e-05
  3092238  2.0576e-02
'   93237  1.0753e-05
    94238  3.1827e-06
  3094239  1.3961e-04
  3094240  4.4975e-05
  3094241  3.1487e-05
    94242  8.5062e-06
    95241  8.5658e-07
'   95243  1.8031e-06
'   95242  2.0922e-08
'   96242  2.1005e-07
'   96244  4.5032e-07
     8016  4.3747e-02
```

```
'    36083  2.5676e-06
'    40093  3.9041e-05
     42095  3.4840e-05
     43099  3.9701e-05
'    44101  3.8036e-05
'    44103  1.6208e-06
     45103  2.2414e-05
'    45105  8.1268e-14
'    46105  1.8320e-05
'    46108  6.0144e-06
'    47109  2.9661e-06
'    53135  0.0000e+00
'    54131  1.6820e-05
'    54135  0.0000e+00
     55133  4.3371e-05
'    55134  4.2653e-06
     55135  1.4168e-05
'    59141  3.8858e-05
     60143  3.0061e-05
     60145  2.3690e-05
'    60147  4.8425e-08
'    60148  1.2475e-05
'    61147  6.5090e-06
'    61148  1.1290e-09
'    61149  6.1869e-12
     62147  2.1573e-06
     62149  1.7234e-07
     62150  1.1694e-05
     62151  6.3655e-07
     62152  4.2374e-06
     63153  3.5278e-06
'    63154  1.2328e-06
'    63155  4.9346e-07
     64155  9.1070e-09
     99999  1.3989e-03
'  buc nuclides -  lumped fission product
'  isotopics for na1c5, burnup=34 gwd/mtu
mix=209
   3092234  3.8385e-06
   3092235  2.3828e-04
   3092236  9.1495e-05
   3092238  2.0576e-02
'    93237  1.0753e-05
     94238  3.1827e-06
   3094239  1.3961e-04
   3094240  4.4975e-05
   3094241  3.1487e-05
     94242  8.5062e-06
     95241  8.5658e-07
'    95243  1.8031e-06
'    95242  2.0922e-08
'    96242  2.1005e-07
```

```
'    96244   4.5032e-07
      8016   4.3747e-02
'    36083   2.5676e-06
'    40093   3.9041e-05
     42095   3.4840e-05
     43099   3.9701e-05
'    44101   3.8036e-05
'    44103   1.6208e-06
     45103   2.2414e-05
'    45105   8.1268e-14
'    46105   1.8320e-05
'    46108   6.0144e-06
'    47109   2.9661e-06
'    53135   0.0000e+00
'    54131   1.6820e-05
'    54135   0.0000e+00
     55133   4.3371e-05
'    55134   4.2653e-06
     55135   1.4168e-05
'    59141   3.8858e-05
     60143   3.0061e-05
     60145   2.3690e-05
'    60147   4.8425e-08
'    60148   1.2475e-05
'    61147   6.5090e-06
'    61148   1.1290e-09
'    61149   6.1869e-12
     62147   2.1573e-06
     62149   1.7234e-07
     62150   1.1694e-05
     62151   6.3655e-07
     62152   4.2374e-06
     63153   3.5278e-06
'    63154   1.2328e-06
'    63155   4.9346e-07
     64155   9.1070e-09
     99999   1.3989e-03
'  buc nuclides -  lumped fission product
'  isotopics for na1c5, burnup=20 gwd/mtu
mix=309
   3092234   3.8385e-06
   3092235   2.3828e-04
   3092236   9.1495e-05
   3092238   2.0576e-02
'    93237   1.0753e-05
     94238   3.1827e-06
   3094239   1.3961e-04
   3094240   4.4975e-05
   3094241   3.1487e-05
     94242   8.5062e-06
     95241   8.5658e-07
'    95243   1.8031e-06
```

```
'    95242  2.0922e-08
'    96242  2.1005e-07
'    96244  4.5032e-07
      8016  4.3747e-02
'    36083  2.5676e-06
'    40093  3.9041e-05
     42095  3.4840e-05
     43099  3.9701e-05
'    44101  3.8036e-05
'    44103  1.6208e-06
     45103  2.2414e-05
'    45105  8.1268e-14
'    46105  1.8320e-05
'    46108  6.0144e-06
'    47109  2.9661e-06
'    53135  0.0000e+00
'    54131  1.6820e-05
'    54135  0.0000e+00
     55133  4.3371e-05
'    55134  4.2653e-06
     55135  1.4168e-05
'    59141  3.8858e-05
     60143  3.0061e-05
     60145  2.3690e-05
'    60147  4.8425e-08
'    60148  1.2475e-05
'    61147  6.5090e-06
'    61148  1.1290e-09
'    61149  6.1869e-12
     62147  2.1573e-06
     62149  1.7234e-07
     62150  1.1694e-05
     62151  6.3655e-07
     62152  4.2374e-06
     63153  3.5278e-06
'    63154  1.2328e-06
'    63155  4.9346e-07
     64155  9.1070e-09
     99999  1.3989e-03
end mixt
read geom
' ***** begin data for plane #1 *****
' *** height contains an extra 16 cm for the gas plenum ***
' ***** first define 17 different fuel pins *****
unit 0101
cylinder 0101 1 .41783 34.29 0.0
cylinder   0 1 .41783 34.29 0.0
cylinder   1 1 .47498 34.29 0.0
cuboid   7 1 2p.62992 2p.62992 34.29 0.0
unit 0102
cylinder 0102 1 .41783 34.29 0.0
cylinder   0 1 .41783 34.29 0.0
```

```
cylinder   1 1 .47498 34.29 0.0
cuboid   7 1 2p.62992 2p.62992 34.29 0.0
unit 0103
cylinder 0103 1 .41783 34.29 0.0
cylinder   0 1 .41783 34.29 0.0
cylinder   1 1 .47498 34.29 0.0
cuboid   7 1 2p.62992 2p.62992 34.29 0.0
unit 0104
cylinder 0104 1 .41783 34.29 0.0
cylinder   0 1 .41783 34.29 0.0
cylinder   1 1 .47498 34.29 0.0
cuboid   7 1 2p.62992 2p.62992 34.29 0.0
unit 0105
cylinder 0105 1 .41783 34.29 0.0
cylinder   0 1 .41783 34.29 0.0
cylinder   1 1 .47498 34.29 0.0
cuboid   7 1 2p.62992 2p.62992 34.29 0.0
unit 0106
cylinder 0106 1 .41783 34.29 0.0
cylinder   0 1 .41783 34.29 0.0
cylinder   1 1 .47498 34.29 0.0
cuboid   7 1 2p.62992 2p.62992 34.29 0.0
unit 0107
cylinder 0107 1 .41783 34.29 0.0
cylinder   0 1 .41783 34.29 0.0
cylinder   1 1 .47498 34.29 0.0
cuboid   7 1 2p.62992 2p.62992 34.29 0.0
unit 0108
cylinder 0108 1 .41783 34.29 0.0
cylinder   0 1 .41783 34.29 0.0
cylinder   1 1 .47498 34.29 0.0
cuboid   7 1 2p.62992 2p.62992 34.29 0.0
unit 0109
cylinder 0109 1 .41783 34.29 0.0
cylinder   0 1 .41783 34.29 0.0
cylinder   1 1 .47498 34.29 0.0
cuboid   7 1 2p.62992 2p.62992 34.29 0.0
unit 0110
cylinder 0110 1 .41783 34.29 0.0
cylinder   0 1 .41783 34.29 0.0
cylinder   1 1 .47498 34.29 0.0
cuboid   7 1 2p.62992 2p.62992 34.29 0.0
unit 0111
cylinder 0111 1 .41783 34.29 0.0
cylinder   0 1 .41783 34.29 0.0
cylinder   1 1 .47498 34.29 0.0
cuboid   7 1 2p.62992 2p.62992 34.29 0.0
unit 0112
cylinder 0112 1 .41783 34.29 0.0
cylinder   0 1 .41783 34.29 0.0
cylinder   1 1 .47498 34.29 0.0
cuboid   7 1 2p.62992 2p.62992 34.29 0.0
```

```
unit 0113
cylinder 0113 1 .41783 34.29 0.0
cylinder   0 1 .41783 34.29 0.0
cylinder   1 1 .47498 34.29 0.0
cuboid   7 1 2p.62992 2p.62992 34.29 0.0
unit 0114
cylinder 0114 1 .41783 34.29 0.0
cylinder   0 1 .41783 34.29 0.0
cylinder   1 1 .47498 34.29 0.0
cuboid   7 1 2p.62992 2p.62992 34.29 0.0
unit 0115
cylinder 0115 1 .41783 34.29 0.0
cylinder   0 1 .41783 34.29 0.0
cylinder   1 1 .47498 34.29 0.0
cuboid   7 1 2p.62992 2p.62992 34.29 0.0
unit 0116
cylinder 0116 1 .41783 34.29 0.0
cylinder   0 1 .41783 34.29 0.0
cylinder   1 1 .47498 34.29 0.0
cuboid   7 1 2p.62992 2p.62992 34.29 0.0
unit 0117
cylinder 0117 1 .41783 34.29 0.0
cylinder   0 1 .41783 34.29 0.0
cylinder   1 1 .47498 34.29 0.0
cuboid   7 1 2p.62992 2p.62992 34.29 0.0
'    ***** define special pins *****
'    ***** guide tube *****
unit 0161
cylinder 7 1 .57150 34.29 0.0
cylinder   1 1 .61214 34.29 0.0
cuboid   7 1 2p.62992 2p.62992 34.29 0.0
'    ***** fresh bp pin *****  '
unit 0162
cylinder   0 1 .21400 34.29 0.0
cylinder   2 1 .22924 34.29 0.0
cylinder   3 1 .43688 34.29 0.0
cylinder   2 1 .48387 34.29 0.0
cylinder 7 1 .57150 34.29 0.0
cylinder   1 1 .61214 34.29 0.0
cuboid   7 1 2p.62992 2p.62992 34.29 0.0
'    ***** depleted bp pin *****  (water hole for now)'
unit 0163
cylinder 7 1 .57150 34.29 0.0
cylinder   1 1 .61214 34.29 0.0
cuboid   7 1 2p.62992 2p.62992 34.29 0.0
'*now define 40 individual assembly units (2 per assembly)*
'two types of assembly units are defined for the 31 element
'burnup credit cask - type i assembly units are xxxx1 and '
'have 0.5cm borated steel on sides and top, and type i
'assembly units are xxxx2 and have 0.5cm borated steel on
'sides and bottom.
'    ***** no bps except as noted *****
```

```
unit 01311
array     0101 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 3r0.5 3r0.0 1
unit 01321
array     0102 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 3r0.5 3r0.0 1
unit 01331
array     0103 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 3r0.5 3r0.0 1
unit 01341
array     0104 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 3r0.5 3r0.0 1
unit 01351
array     0105 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 3r0.5 3r0.0 1
unit 01361
array     0106 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 3r0.5 3r0.0 1
'    ***** 12 depleted bps *****
unit 01371
array    0107 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 3r0.5 3r0.0 1
unit 01381
array     0108 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 3r0.5 3r0.0 1
unit 01391
array     0109 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 3r0.5 3r0.0 1
unit 01401
array     0110 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 3r0.5 3r0.0 1
unit 01411
array     0111 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 3r0.5 3r0.0 1
unit 01421
array     0112 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 3r0.5 3r0.0 1
unit 01431
array     0113 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
```

```
reflector 5 1 3r0.5 3r0.0 1
unit 01441
array     0114 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 3r0.5 3r0.0 1
unit 01451
array     0115 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 3r0.5 3r0.0 1
unit 01461
array     0116 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 3r0.5 3r0.0 1
'    ***** fresh fuel   no bps *****
unit 01471
array     0117 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 3r0.5 3r0.0 1
'    ***** fresh fuel   16 fresh bps *****
unit 01481
array     0118 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 3r0.5 3r0.0 1
'    ***** fresh fuel   20 fresh bps *****
unit 01491
array     0119 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 3r0.5 3r0.0 1
'    ***** fresh fuel   12 depleted bps *****
unit 01501
array     0120 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 3r0.5 3r0.0 1
unit 01312
array     0101 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 2r0.5 0.0 0.5 2r0.0 1
unit 01322
array     0102 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 2r0.5 0.0 0.5 2r0.0 1
unit 01332
array     0103 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 2r0.5 0.0 0.5 2r0.0 1
unit 01342
array     0104 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 2r0.5 0.0 0.5 2r0.0 1
unit 01352
array     0105 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
```

```
reflector 5 1 2r0.5 0.0 0.5 2r0.0 1
unit 01362
array     0106 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 2r0.5 0.0 0.5 2r0.0 1
'    ***** 12 depleted bps *****
unit 01372
array     0107 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 2r0.5 0.0 0.5 2r0.0 1
unit 01382
array     0108 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 2r0.5 0.0 0.5 2r0.0 1
unit 01392
array     0109 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 2r0.5 0.0 0.5 2r0.0 1
unit 01402
array     0110 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 2r0.5 0.0 0.5 2r0.0 1
unit 01412
array     0111 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 2r0.5 0.0 0.5 2r0.0 1
unit 01422
array     0112 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 2r0.5 0.0 0.5 2r0.0 1
unit 01432
array     0113 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 2r0.5 0.0 0.5 2r0.0 1
unit 01442
array     0114 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 2r0.5 0.0 0.5 2r0.0 1
unit 01452
array     0115 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 2r0.5 0.0 0.5 2r0.0 1
unit 01462
array     0116 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 2r0.5 0.0 0.5 2r0.0 1
'    ***** fresh fuel   no bps *****
unit 01472
array     0117 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 2r0.5 0.0 0.5 2r0.0 1
'    ***** fresh fuel   16 fresh bps *****
```

```
unit 01482
array     0118 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 2r0.5 0.0 0.5 2r0.0 1
'    ***** fresh fuel   20 fresh bps *****
unit 01492
array     0119 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 2r0.5 0.0 0.5 2r0.0 1
'    ***** fresh fuel   12 depleted bps *****
unit 01502
array     0120 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 2r0.5 0.0 0.5 2r0.0 1
'    ***** end of data for plane #1 ****
'    ***** begin data for plane #2 *****
'**** height contains an extra 16 cm for the gas plenum
****
'    ***** first define 17 different fuel pins *****
unit 0201
cylinder 0201 1 .41783 297.18 0.0
cylinder   0 1 .41783 297.18 0.0
cylinder   1 1 .47498 297.18 0.0
cuboid   7 1 2p.62992 2p.62992 297.18 0.0
unit 0202
cylinder 0202 1 .41783 297.18 0.0
cylinder   0 1 .41783 297.18 0.0
cylinder   1 1 .47498 297.18 0.0
cuboid   7 1 2p.62992 2p.62992 297.18 0.0
unit 0203
cylinder 0203 1 .41783 297.18 0.0
cylinder   0 1 .41783 297.18 0.0
cylinder   1 1 .47498 297.18 0.0
cuboid   7 1 2p.62992 2p.62992 297.18 0.0
unit 0204
cylinder 0204 1 .41783 297.18 0.0
cylinder   0 1 .41783 297.18 0.0
cylinder   1 1 .47498 297.18 0.0
cuboid   7 1 2p.62992 2p.62992 297.18 0.0
unit 0205
cylinder 0205 1 .41783 297.18 0.0
cylinder   0 1 .41783 297.18 0.0
cylinder   1 1 .47498 297.18 0.0
cuboid   7 1 2p.62992 2p.62992 297.18 0.0
unit 0206
cylinder 0206 1 .41783 297.18 0.0
cylinder   0 1 .41783 297.18 0.0
cylinder   1 1 .47498 297.18 0.0
cuboid   7 1 2p.62992 2p.62992 297.18 0.0
unit 0207
cylinder 0207 1 .41783 297.18 0.0
cylinder   0 1 .41783 297.18 0.0
```

```
cylinder   1 1 .47498 297.18 0.0
cuboid   7 1 2p.62992 2p.62992 297.18 0.0
unit 0208
cylinder 0208 1 .41783 297.18 0.0
cylinder   0 1 .41783 297.18 0.0
cylinder   1 1 .47498 297.18 0.0
cuboid   7 1 2p.62992 2p.62992 297.18 0.0
unit 0209
cylinder 0209 1 .41783 297.18 0.0
cylinder   0 1 .41783 297.18 0.0
cylinder   1 1 .47498 297.18 0.0
cuboid   7 1 2p.62992 2p.62992 297.18 0.0
unit 0210
cylinder 0210 1 .41783 297.18 0.0
cylinder   0 1 .41783 297.18 0.0
cylinder   1 1 .47498 297.18 0.0
cuboid   7 1 2p.62992 2p.62992 297.18 0.0
unit 0211
cylinder 0211 1 .41783 297.18 0.0
cylinder   0 1 .41783 297.18 0.0
cylinder   1 1 .47498 297.18 0.0
cuboid   7 1 2p.62992 2p.62992 297.18 0.0
unit 0212
cylinder 0212 1 .41783 297.18 0.0
cylinder   0 1 .41783 297.18 0.0
cylinder   1 1 .47498 297.18 0.0
cuboid   7 1 2p.62992 2p.62992 297.18 0.0
unit 0213
cylinder 0213 1 .41783 297.18 0.0
cylinder   0 1 .41783 297.18 0.0
cylinder   1 1 .47498 297.18 0.0
cuboid   7 1 2p.62992 2p.62992 297.18 0.0
unit 0214
cylinder 0214 1 .41783 297.18 0.0
cylinder   0 1 .41783 297.18 0.0
cylinder   1 1 .47498 297.18 0.0
cuboid   7 1 2p.62992 2p.62992 297.18 0.0
unit 0215
cylinder 0215 1 .41783 297.18 0.0
cylinder   0 1 .41783 297.18 0.0
cylinder   1 1 .47498 297.18 0.0
cuboid   7 1 2p.62992 2p.62992 297.18 0.0
unit 0216
cylinder 0216 1 .41783 297.18 0.0
cylinder   0 1 .41783 297.18 0.0
cylinder   1 1 .47498 297.18 0.0
cuboid   7 1 2p.62992 2p.62992 297.18 0.0
unit 0217
cylinder 0217 1 .41783 297.18 0.0
cylinder   0 1 .41783 297.18 0.0
cylinder   1 1 .47498 297.18 0.0
cuboid   7 1 2p.62992 2p.62992 297.18 0.0
```

```
'    ***** define special pins *****
'    ***** guide tube *****
unit 0261
cylinder 7 1 .57150 297.18 0.0
cylinder   1 1 .61214 297.18 0.0
cuboid   7 1 2p.62992 2p.62992 297.18 0.0
'    ***** fresh bp pin *****  '
unit 0262
cylinder   0 1 .21400 297.18 0.0
cylinder   2 1 .22924 297.18 0.0
cylinder   3 1 .43688 297.18 0.0
cylinder   2 1 .48387 297.18 0.0
cylinder 7 1 .57150 297.18 0.0
cylinder   1 1 .61214 297.18 0.0
cuboid   7 1 2p.62992 2p.62992 297.18 0.0
'    ***** depleted bp pin *****  (water hole for noow)'
unit 0263
cylinder 7 1 .57150 297.18 0.0
cylinder   1 1 .61214 297.18 0.0
cuboid   7 1 2p.62992 2p.62992 297.18 0.0
'*now define 40 individual assembly units (2 per assembly)*
'two types of assembly units are defined for the 31 element
'burnup credit cask - type i assembly units are xxxx1 and
'have 0.5cm borated steel on sides and top, and type i
'assembly units are xxxx2 and have 0.5cm borated steel on
'sides and bottom.
'    ***** no bps except as noted *****
unit 02311
array     0201 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 3r0.5 3r0.0 1
unit 02321
array     0202 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 3r0.5 3r0.0 1
unit 02331
array     0203 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 3r0.5 3r0.0 1
unit 02341
array     0204 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 3r0.5 3r0.0 1
unit 02351
array     0205 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 3r0.5 3r0.0 1
unit 02361
array     0206 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 3r0.5 3r0.0 1
'    ***** 12 depleted bps *****
```

```
unit 02371
array    0207 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 3r0.5 3r0.0 1
unit 02381
array     0208 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 3r0.5 3r0.0 1
unit 02391
array     0209 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 3r0.5 3r0.0 1
unit 02401
array     0210 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 3r0.5 3r0.0 1
unit 02411
array     0211 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 3r0.5 3r0.0 1
unit 02421
array     0212 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 3r0.5 3r0.0 1
unit 02431
array     0213 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 3r0.5 3r0.0 1
unit 02441
array     0214 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 3r0.5 3r0.0 1
unit 02451
array     0215 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 3r0.5 3r0.0 1
unit 02461
array     0216 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 3r0.5 3r0.0 1
'    ***** fresh fuel   no bps *****
unit 02471
array     0217 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 3r0.5 3r0.0 1
'    ***** fresh fuel   16 fresh bps *****
unit 02481
array     0218 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 3r0.5 3r0.0 1
'    ***** fresh fuel   20 fresh bps *****
unit 02491
```

```
array     0219 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 3r0.5 3r0.0 1
'    ***** fresh fuel   12 depleted bps *****
unit 02501
array     0220 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 3r0.5 3r0.0 1
unit 02312
array     0201 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 2r0.5 0.0 0.5 2r0.0 1
unit 02322
array     0202 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 2r0.5 0.0 0.5 2r0.0 1
unit 02332
array     0203 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 2r0.5 0.0 0.5 2r0.0 1
unit 02342
array     0204 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 2r0.5 0.0 0.5 2r0.0 1
unit 02352
array     0205 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 2r0.5 0.0 0.5 2r0.0 1
unit 02362
array     0206 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 2r0.5 0.0 0.5 2r0.0 1
'    ***** 12 depleted bps *****
unit 02372
array     0207 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 2r0.5 0.0 0.5 2r0.0 1
unit 02382
array     0208 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 2r0.5 0.0 0.5 2r0.0 1
unit 02392
array     0209 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 2r0.5 0.0 0.5 2r0.0 1
unit 02402
array     0210 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 2r0.5 0.0 0.5 2r0.0 1
unit 02412
array     0211 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
```

```
reflector 5 1 2r0.5 0.0 0.5 2r0.0 1
unit 02422
array     0212 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 2r0.5 0.0 0.5 2r0.0 1
unit 02432
array     0213 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 2r0.5 0.0 0.5 2r0.0 1
unit 02442
array     0214 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 2r0.5 0.0 0.5 2r0.0 1
unit 02452
array     0215 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 2r0.5 0.0 0.5 2r0.0 1
unit 02462
array     0216 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 2r0.5 0.0 0.5 2r0.0 1
'    ***** fresh fuel   no bps *****
unit 02472
array     0217 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 2r0.5 0.0 0.5 2r0.0 1
'    ***** fresh fuel   16 fresh bps *****
unit 02482
array     0218 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 2r0.5 0.0 0.5 2r0.0 1
'    ***** fresh fuel   20 fresh bps *****
unit 02492
array     0219 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 2r0.5 0.0 0.5 2r0.0 1
'    ***** fresh fuel   12 depleted bps *****
unit 02502
array     0220 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 2r0.5 0.0 0.5 2r0.0 1
'    ***** end of data for plane #2 ****
'    ***** begin data for plane #3 *****
'*** height contains an extra 16 cm for the gas plenum ***
'     ***** first define 17 different fuel pins ****
unit 0301
cylinder 0301 1 .41783 34.29 0.0
cylinder   0 1 .41783 50.29 0.0
cylinder   1 1 .47498 50.29 0.0
cuboid   7 1 2p.62992 2p.62992 50.29 0.0
unit 0302
cylinder 0302 1 .41783 34.29 0.0
```

```
cylinder   0 1 .41783 50.29 0.0
cylinder   1 1 .47498 50.29 0.0
cuboid   7 1 2p.62992 2p.62992 50.29 0.0
unit 0303
cylinder 0303 1 .41783 34.29 0.0
cylinder   0 1 .41783 50.29 0.0
cylinder   1 1 .47498 50.29 0.0
cuboid   7 1 2p.62992 2p.62992 50.29 0.0
unit 0304
cylinder 0304 1 .41783 34.29 0.0
cylinder   0 1 .41783 50.29 0.0
cylinder   1 1 .47498 50.29 0.0
cuboid   7 1 2p.62992 2p.62992 50.29 0.0
unit 0305
cylinder 0305 1 .41783 34.29 0.0
cylinder   0 1 .41783 50.29 0.0
cylinder   1 1 .47498 50.29 0.0
cuboid   7 1 2p.62992 2p.62992 50.29 0.0
unit 0306
cylinder 0306 1 .41783 34.29 0.0
cylinder   0 1 .41783 50.29 0.0
cylinder   1 1 .47498 50.29 0.0
cuboid   7 1 2p.62992 2p.62992 50.29 0.0
unit 0307
cylinder 0307 1 .41783 34.29 0.0
cylinder   0 1 .41783 50.29 0.0
cylinder   1 1 .47498 50.29 0.0
cuboid   7 1 2p.62992 2p.62992 50.29 0.0
unit 0308
cylinder 0308 1 .41783 34.29 0.0
cylinder   0 1 .41783 50.29 0.0
cylinder   1 1 .47498 50.29 0.0
cuboid   7 1 2p.62992 2p.62992 50.29 0.0
unit 0309
cylinder 0309 1 .41783 34.29 0.0
cylinder   0 1 .41783 50.29 0.0
cylinder   1 1 .47498 50.29 0.0
cuboid   7 1 2p.62992 2p.62992 50.29 0.0
unit 0310
cylinder 0310 1 .41783 34.29 0.0
cylinder   0 1 .41783 50.29 0.0
cylinder   1 1 .47498 50.29 0.0
cuboid   7 1 2p.62992 2p.62992 50.29 0.0
unit 0311
cylinder 0311 1 .41783 34.29 0.0
cylinder   0 1 .41783 50.29 0.0
cylinder   1 1 .47498 50.29 0.0
cuboid   7 1 2p.62992 2p.62992 50.29 0.0
unit 0312
cylinder 0312 1 .41783 34.29 0.0
cylinder   0 1 .41783 50.29 0.0
cylinder   1 1 .47498 50.29 0.0
```

```
cuboid   7 1 2p.62992 2p.62992 50.29 0.0
unit 0313
cylinder 0313 1 .41783 34.29 0.0
cylinder   0 1 .41783 50.29 0.0
cylinder   1 1 .47498 50.29 0.0
cuboid   7 1 2p.62992 2p.62992 50.29 0.0
unit 0314
cylinder 0314 1 .41783 34.29 0.0
cylinder   0 1 .41783 50.29 0.0
cylinder   1 1 .47498 50.29 0.0
cuboid   7 1 2p.62992 2p.62992 50.29 0.0
unit 0315
cylinder 0315 1 .41783 34.29 0.0
cylinder   0 1 .41783 50.29 0.0
cylinder   1 1 .47498 50.29 0.0
cuboid   7 1 2p.62992 2p.62992 50.29 0.0
unit 0316
cylinder 0316 1 .41783 34.29 0.0
cylinder   0 1 .41783 50.29 0.0
cylinder   1 1 .47498 50.29 0.0
cuboid   7 1 2p.62992 2p.62992 50.29 0.0
unit 0317
cylinder 0317 1 .41783 34.29 0.0
cylinder   0 1 .41783 50.29 0.0
cylinder   1 1 .47498 50.29 0.0
cuboid   7 1 2p.62992 2p.62992 50.29 0.0
'    ***** define special pins *****
'    ***** guide tube *****
unit 0361
cylinder 7 1 .57150 50.29 0.0
cylinder   1 1 .61214 50.29 0.0
cuboid   7 1 2p.62992 2p.62992 50.29 0.0
'    ***** fresh bp pin *****  '
unit 0362
cylinder   0 1 .21400 34.29 0.0
cylinder   2 1 .22924 34.29 0.0
cylinder   3 1 .43688 34.29 0.0
cylinder   2 1 .48387 34.29 0.0
cylinder 7 1 .57150 50.29 0.0
cylinder   1 1 .61214 50.29 0.0
cuboid   7 1 2p.62992 2p.62992 50.29 0.0
'    ***** depleted bp pin *****  (water hole for now)'
unit 0363
cylinder 7 1 .57150 50.29 0.0
cylinder   1 1 .61214 50.29 0.0
cuboid   7 1 2p.62992 2p.62992 50.29 0.0
'*now define 40 individual assembly units (2 per assembly)*
'two types of assembly units are defined for the 31 element
'burnup credit cask - type i assembly units are xxxx1 and
'have  0.5cm borated steel on sides and top, and type i
'assembly units are xxxx2 and have 0.5cm borated steel on
'sides and bottom.
```

```
'    ***** no bps except as noted *****
unit 03311
array     0301 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 3r0.5 3r0.0 1
unit 03321
array     0302 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 3r0.5 3r0.0 1
unit 03331
array     0303 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 3r0.5 3r0.0 1
unit 03341
array     0304 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 3r0.5 3r0.0 1
unit 03351
array     0305 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 3r0.5 3r0.0 1
unit 03361
array     0306 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 3r0.5 3r0.0 1
'    ***** 12 depleted bps *****
unit 03371
array    0307 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 3r0.5 3r0.0 1
unit 03381
array     0308 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 3r0.5 3r0.0 1
unit 03391
array     0309 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 3r0.5 3r0.0 1
unit 03401
array     0310 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 3r0.5 3r0.0 1
unit 03411
array     0311 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 3r0.5 3r0.0 1
unit 03421
array     0312 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 3r0.5 3r0.0 1
unit 03431
array     0313 2r-10.70864 0.0
```

```
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 3r0.5 3r0.0 1
unit 03441
array     0314 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 3r0.5 3r0.0 1
unit 03451
array     0315 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 3r0.5 3r0.0 1
unit 03461
array     0316 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 3r0.5 3r0.0 1
'    ***** fresh fuel   no bps *****
unit 03471
array     0317 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 3r0.5 3r0.0 1
'    ***** fresh fuel   16 fresh bps *****
unit 03481
array     0318 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 3r0.5 3r0.0 1
'    ***** fresh fuel   20 fresh bps *****
unit 03491
array     0319 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 3r0.5 3r0.0 1
'    ***** fresh fuel   12 depleted bps *****
unit 03501
array     0320 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 3r0.5 3r0.0 1
unit 03312
array     0301 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 2r0.5 0.0 0.5 2r0.0 1
unit 03322
array     0302 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 2r0.5 0.0 0.5 2r0.0 1
unit 03332
array     0303 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 2r0.5 0.0 0.5 2r0.0 1
unit 03342
array     0304 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 2r0.5 0.0 0.5 2r0.0 1
unit 03352
array     0305 2r-10.70864 0.0
```

```
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 2r0.5 0.0 0.5 2r0.0 1
unit 03362
array     0306 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 2r0.5 0.0 0.5 2r0.0 1
'    ***** 12 depleted bps *****
unit 03372
array     0307 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 2r0.5 0.0 0.5 2r0.0 1
unit 03382
array     0308 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 2r0.5 0.0 0.5 2r0.0 1
unit 03392
array     0309 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 2r0.5 0.0 0.5 2r0.0 1
unit 03402
array     0310 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 2r0.5 0.0 0.5 2r0.0 1
unit 03412
array     0311 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 2r0.5 0.0 0.5 2r0.0 1
unit 03422
array     0312 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 2r0.5 0.0 0.5 2r0.0 1
unit 03432
array     0313 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 2r0.5 0.0 0.5 2r0.0 1
unit 03442
array     0314 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 2r0.5 0.0 0.5 2r0.0 1
unit 03452
array     0315 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 2r0.5 0.0 0.5 2r0.0 1
unit 03462
array     0316 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 2r0.5 0.0 0.5 2r0.0 1
'    ***** fresh fuel   no bps *****
unit 03472
array     0317 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 2r0.5 0.0 0.5 2r0.0 1
```

```
'    ***** fresh fuel   16 fresh bps *****
unit 03482
array     0318 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 2r0.5 0.0 0.5 2r0.0 1
'    ***** fresh fuel   20 fresh bps *****
unit 03492
array     0319 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 2r0.5 0.0 0.5 2r0.0 1
'    ***** fresh fuel   12 depleted bps *****
unit 03502
array     0320 2r-10.70864 0.0
reflector 7 1 4r0.29136 2r0.0 1
reflector 5 1 2r0.5 0.0 0.5 2r0.0 1
'    ***** end of data for plane #3 ****
'  build the seven units that will be used as holes in

'  the global unit
unit 32
array 32 -11.5 2r0.0
reflector 5 1 3r0.5 3r0.0 1
unit 33
array 33 -57.5 2r0.0
reflector 5 1 3r0.5 3r0.0 1
unit 34
array 34 -69.0 2r0.0
reflector 5 1 3r0.5 3r0.0 1
unit 35
array 35 -80.5 2r0.0
reflector 5 1 3r0.50 1.00 2r0.0 1
unit 36
array 36 -69.0 2r0.0
reflector 5 1 2r0.5 0.0 0.5 2r0.0 1
unit 37
array 37 -57.5 2r0.0
reflector 5 1 2r0.5 0.0 0.5 2r0.0 1
unit 38
array 38 -11.5 2r0.0
reflector 5 1 2r0.5 0.0 0.5 2r0.0 1
'define global model unit for 31 element burnup-credit cask
global unit 31
cylinder 7 1 83.25 381.76 0.0
hole 32 0.0 58 0.0
hole 33 0.0 35 0.0
hole 34 0.0 12 0.0
hole 35 0.0 -11.0 0.0
hole 36 0.0 -34.5 0.0
hole 37 0.0 -57.5 0.0
hole 38 0.0 -80.5 0.0
reflector 4 1 0.15 0.0 0.0 1
reflector 7 1 0.0 32.14 0.0 1
```

```
reflector 4 1 0.0 38.00 0.0 1
reflector 6 1 33.0 0.0 34.0 1
end geom
'****** end of global data **********
' build the 7 arrays for the 31 element cask
'  array numbers 32-38 correspond to rows ca-cg
'  type i units are used in rows ca-cd (32-35)
'  type ii units are used in rows ce-cg (36-38)
read array
ara=32 nux=1 nuy=1 nuz=3   fill
 1391
 2391
 3391
 end fill
ara=33 nux=5 nuy=1 nuz=3   fill
 1391 1391 1391 1391 1391
 2391 2391 2391 2391 2391
 3391 3391 3391 3391 3391
 end fill
ara=34 nux=6 nuy=1 nuz=3   fill
 1391 1391 1391 1391 1391 1391
 2391 2391 2391 2391 2391 2391
 3391 3391 3391 3391 3391 3391
 end fill
ara=35 nux=7 nuy=1 nuz=3   fill
 1391 1391 1391 1391 1391 1391 1391
 2391 2391 2391 2391 2391 2391 2391
 3391 3391 3391 3391 3391 3391 3391
 end fill
ara=36 nux=6 nuy=1 nuz=3   fill
 1392 1392 1392 1392 1392 1392
 2392 2392 2392 2392 2392 2392
 3392 3392 3392 3392 3392 3392
 end fill
ara=37 nux=5 nuy=1 nuz=3   fill
 1392 1392 1392 1392 1392
 2392 2392 2392 2392 2392
 3392 3392 3392 3392 3392
 end fill
ara=38 nux=1 nuy=1 nuz=3   fill
 1392
 2392
 3392
 end fill
' array data for fuel assemblies 101-120
'    ***** array data for plane #1   *****
'    *** position h8    xsec batch #5    no bps ***
ara=101   nux=17 nuy=17 nuz=1     fill  f101
   a40 161   a43 161   a46 161   a55 161   a65 161
  a88 161   a91 161   a94 161   a97 161  a100 161
 a139 161  a142 161  a145 161  a148 161  a151 161
 a190 161  a193 161  a196 161  a199 161  a202 161
```

```
 a225 161  a235 161  a244 161  a247 161  a250 161  end fill
'    *** position f8    xsec batch #1    no bps ***
ara=102   nux=17 nuy=17 nuz=1     fill  f102
   a40 161   a43 161   a46 161   a55 161   a65 161
  a88 161   a91 161   a94 161   a97 161  a100 161
 a139 161  a142 161  a145 161  a148 161  a151 161
 a190 161  a193 161  a196 161  a199 161  a202 161
 a225 161  a235 161  a244 161  a247 161  a250 161  end fill
'    *** position d8    xsec batch #5    no bps ***
ara=103   nux=17 nuy=17 nuz=1     fill  f103
   a40 161   a43 161   a46 161   a55 161   a65 161
  a88 161   a91 161   a94 161   a97 161  a100 161
 a139 161  a142 161  a145 161  a148 161  a151 161
 a190 161  a193 161  a196 161  a199 161  a202 161
 a225 161  a235 161  a244 161  a247 161  a250 161  end fill
'    *** position b8    xsec batch #5    no bps ***
ara=104   nux=17 nuy=17 nuz=1     fill  f104
   a40 161   a43 161   a46 161   a55 161   a65 161
  a88 161   a91 161   a94 161   a97 161  a100 161
 a139 161  a142 161  a145 161  a148 161  a151 161
 a190 161  a193 161  a196 161  a199 161  a202 161
 a225 161  a235 161  a244 161  a247 161  a250 161  end fill
'    *** position a8    xsec batch #4    no bps ***
ara=105   nux=17 nuy=17 nuz=1     fill  f105
   a40 161   a43 161   a46 161   a55 161   a65 161
  a88 161   a91 161   a94 161   a97 161  a100 161
 a139 161  a142 161  a145 161  a148 161  a151 161
 a190 161  a193 161  a196 161  a199 161  a202 161
 a225 161  a235 161  a244 161  a247 161  a250 161  end fill
'    *** position g9    xsec batch #2    no bps ***
ara=106   nux=17 nuy=17 nuz=1     fill  f106
   a40 161   a43 161   a46 161   a55 161   a65 161
  a88 161   a91 161   a94 161   a97 161  a100 161
 a139 161  a142 161  a145 161  a148 161  a151 161
 a190 161  a193 161  a196 161  a199 161  a202 161
 a225 161  a235 161  a244 161  a247 161  a250 161  end fill
'    *** position e9    xsec batch #5   12 depleted bps ***
ara=107   nux=17 nuy=17 nuz=1     fill  f107
   a40 163   a43 161   a46 163   a55 161   a65 161
  a88 163   a91 161   a94 163   a97 161  a100 163
 a139 161  a142 163  a145 161  a148 163  a151 161
 a190 163  a193 161  a196 163  a199 161  a202 163
 a225 161  a235 161  a244 163  a247 161  a250 163  end fill
'    *** position c9    xsec batch #4    no bps ***
ara=108   nux=17 nuy=17 nuz=1     fill  f108
   a40 161   a43 161   a46 161   a55 161   a65 161
  a88 161   a91 161   a94 161   a97 161  a100 161
 a139 161  a142 161  a145 161  a148 161  a151 161
 a190 161  a193 161  a196 161  a199 161  a202 161
 a225 161  a235 161  a244 161  a247 161  a250 161  end fill
'    *** position a9    xsec batch #3    no bps ***
ara=109   nux=17 nuy=17 nuz=1     fill  f109
```

```
   a40 161   a43 161   a46 161   a55 161   a65 161
  a88 161   a91 161   a94 161   a97 161  a100 161
 a139 161  a142 161  a145 161  a148 161  a151 161
 a190 161  a193 161  a196 161  a199 161  a202 161
 a225 161  a235 161  a244 161  a247 161  a250 161  end fill
'    *** position f10    xsec batch #5    no bps ***
ara=110   nux=17 nuy=17 nuz=1     fill  f110
   a40 161   a43 161   a46 161   a55 161   a65 161
  a88 161   a91 161   a94 161   a97 161  a100 161
 a139 161  a142 161  a145 161  a148 161  a151 161
 a190 161  a193 161  a196 161  a199 161  a202 161
 a225 161  a235 161  a244 161  a247 161  a250 161  end fill
'    *** position d10    xsec batch #5    no bps ***
ara=111   nux=17 nuy=17 nuz=1     fill  f111
   a40 161   a43 161   a46 161   a55 161   a65 161
  a88 161   a91 161   a94 161   a97 161  a100 161
 a139 161  a142 161  a145 161  a148 161  a151 161
 a190 161  a193 161  a196 161  a199 161  a202 161
 a225 161  a235 161  a244 161  a247 161  a250 161  end fill
'    *** position b10    xsec batch #5    no bps ***
ara=112   nux=17 nuy=17 nuz=1     fill  f112
   a40 161   a43 161   a46 161   a55 161   a65 161
  a88 161   a91 161   a94 161   a97 161  a100 161
 a139 161  a142 161  a145 161  a148 161  a151 161
 a190 161  a193 161  a196 161  a199 161  a202 161
 a225 161  a235 161  a244 161  a247 161  a250 161  end fill
'    *** position e11    xsec batch #5    no bps ***
ara=113   nux=17 nuy=17 nuz=1     fill  f113
   a40 161   a43 161   a46 161   a55 161   a65 161
  a88 161   a91 161   a94 161   a97 161  a100 161
 a139 161  a142 161  a145 161  a148 161  a151 161
 a190 161  a193 161  a196 161  a199 161  a202 161
 a225 161  a235 161  a244 161  a247 161  a250 161  end fill
'    *** position b11    xsec batch #3    no bps ***
ara=114   nux=17 nuy=17 nuz=1     fill  f114
   a40 161   a43 161   a46 161   a55 161   a65 161
  a88 161   a91 161   a94 161   a97 161  a100 161
 a139 161  a142 161  a145 161  a148 161  a151 161
 a190 161  a193 161  a196 161  a199 161  a202 161
 a225 161  a235 161  a244 161  a247 161  a250 161  end fill
'    *** position d12    xsec batch #4    no bps ***
ara=115   nux=17 nuy=17 nuz=1     fill  f115
   a40 161   a43 161   a46 161   a55 161   a65 161
  a88 161   a91 161   a94 161   a97 161  a100 161
 a139 161  a142 161  a145 161  a148 161  a151 161
 a190 161  a193 161  a196 161  a199 161  a202 161
 a225 161  a235 161  a244 161  a247 161  a250 161  end fill
'    *** position c12    xsec batch #1    no bps ***
ara=116   nux=17 nuy=17 nuz=1     fill  f116
   a40 161   a43 161   a46 161   a55 161   a65 161
  a88 161   a91 161   a94 161   a97 161  a100 161
 a139 161  a142 161  a145 161  a148 161  a151 161
```

```
 a190 161  a193 161  a196 161  a199 161  a202 161
 a225 161  a235 161  a244 161  a247 161  a250 161  end fill
'*position c11 only   xsec batch #6 (fresh fuel)  no bps*
ara=117   nux=17 nuy=17 nuz=1     fill  f117
   a40 161   a43 161   a46 161   a55 161   a65 161
  a88 161   a91 161   a94 161   a97 161  a100 161
 a139 161  a142 161  a145 161  a148 161  a151 161
 a190 161  a193 161  a196 161  a199 161  a202 161
 a225 161  a235 161  a244 161  a247 161  a250 161  end fill
'*position g8+c8+f9+c10+d11  (fresh fuel #6)  16 fresh bps*
ara=118   nux=17 nuy=17 nuz=1     fill  f117
   a40 162   a43 161   a46 162   a55 162   a65 162
  a88 162   a91 161   a94 162   a97 161  a100 162
 a139 161  a142 162  a145 161  a148 162  a151 161
 a190 162  a193 161  a196 162  a199 161  a202 162
 a225 162  a235 162  a244 162  a247 161  a250 162  end fill
'*** position e8+d9+e10  (fresh fuel #6)  20 fresh bps ***
ara=119   nux=17 nuy=17 nuz=1     fill  f117
   a40 162   a43 162   a46 162   a55 162   a65 162
  a88 162   a91 161   a94 162   a97 161  a100 162
 a139 162  a142 162  a145 161  a148 162  a151 162
 a190 162  a193 161  a196 162  a199 161  a202 162
 a225 162  a235 162  a244 162  a247 162  a250 162  end fill
'*** position b9 only  (fresh fuel #6)  12 depleted bps ***
ara=120   nux=17 nuy=17 nuz=1     fill  f117
   a40 163   a43 161   a46 163   a55 161   a65 161
  a88 163   a91 161   a94 163   a97 161  a100 163
 a139 161  a142 163  a145 161  a148 163  a151 161
 a190 163  a193 161  a196 163  a199 161  a202 163
 a225 161  a235 161  a244 163  a247 161  a250 163  end fill
'    ***** end array data for plane #1 *****
' array data for fuel assemblies 101-120
'    ***** array data for plane #2   *****
'    *** position h8    xsec batch #5    no bps ***
ara=201   nux=17 nuy=17 nuz=1     fill  f201
   a40 261   a43 261   a46 261   a55 261   a65 261
  a88 261   a91 261   a94 261   a97 261  a100 261
 a139 261  a142 261  a145 261  a148 261  a151 261
 a190 261  a193 261  a196 261  a199 261  a202 261
 a225 261  a235 261  a244 261  a247 261  a250 261  end fill
'    *** position f8    xsec batch #1    no bps ***
ara=202   nux=17 nuy=17 nuz=1     fill  f202
   a40 261   a43 261   a46 261   a55 261   a65 261
  a88 261   a91 261   a94 261   a97 261  a100 261
 a139 261  a142 261  a145 261  a148 261  a151 261
 a190 261  a193 261  a196 261  a199 261  a202 261
 a225 261  a235 261  a244 261  a247 261  a250 261  end fill
'    *** position d8    xsec batch #5    no bps ***
ara=203   nux=17 nuy=17 nuz=1     fill  f203
   a40 261   a43 261   a46 261   a55 261   a65 261
  a88 261   a91 261   a94 261   a97 261  a100 261
 a139 261  a142 261  a145 261  a148 261  a151 261
```

```
 a190 261  a193 261  a196 261  a199 261  a202 261
 a225 261  a235 261  a244 261  a247 261  a250 261  end fill
'    *** position b8    xsec batch #5    no bps ***
ara=204   nux=17 nuy=17 nuz=1     fill  f204
   a40 261   a43 261   a46 261   a55 261   a65 261
  a88 261   a91 261   a94 261   a97 261  a100 261
 a139 261  a142 261  a145 261  a148 261  a151 261
 a190 261  a193 261  a196 261  a199 261  a202 261
 a225 261  a235 261  a244 261  a247 261  a250 261  end fill
'    *** position a8    xsec batch #4    no bps ***
ara=205   nux=17 nuy=17 nuz=1     fill  f205
   a40 261   a43 261   a46 261   a55 261   a65 261
  a88 261   a91 261   a94 261   a97 261  a100 261
 a139 261  a142 261  a145 261  a148 261  a151 261
 a190 261  a193 261  a196 261  a199 261  a202 261
 a225 261  a235 261  a244 261  a247 261  a250 261  end fill
'    *** position g9    xsec batch #2    no bps ***
ara=206   nux=17 nuy=17 nuz=1     fill  f206
   a40 261   a43 261   a46 261   a55 261   a65 261
  a88 261   a91 261   a94 261   a97 261  a100 261
 a139 261  a142 261  a145 261  a148 261  a151 261
 a190 261  a193 261  a196 261  a199 261  a202 261
 a225 261  a235 261  a244 261  a247 261  a250 261  end fill
'  *** position e9    xsec batch #5   12 depleted bps ***
ara=207   nux=17 nuy=17 nuz=1     fill  f207
   a40 263   a43 261   a46 263   a55 261   a65 261
  a88 263   a91 261   a94 263   a97 261  a100 263
 a139 261  a142 263  a145 261  a148 263  a151 261
 a190 263  a193 261  a196 263  a199 261  a202 263
 a225 261  a235 261  a244 263  a247 261  a250 263  end fill
'    *** position c9    xsec batch #4    no bps ***
ara=208   nux=17 nuy=17 nuz=1     fill  f208
   a40 261   a43 261   a46 261   a55 261   a65 261
  a88 261   a91 261   a94 261   a97 261  a100 261
 a139 261  a142 261  a145 261  a148 261  a151 261
 a190 261  a193 261  a196 261  a199 261  a202 261
 a225 261  a235 261  a244 261  a247 261  a250 261  end fill
'    *** position a9    xsec batch #3    no bps ***
ara=209   nux=17 nuy=17 nuz=1     fill  f209
   a40 261   a43 261   a46 261   a55 261   a65 261
  a88 261   a91 261   a94 261   a97 261  a100 261
 a139 261  a142 261  a145 261  a148 261  a151 261
 a190 261  a193 261  a196 261  a199 261  a202 261
 a225 261  a235 261  a244 261  a247 261  a250 261  end fill
'    *** position f10    xsec batch #5    no bps ***
ara=210   nux=17 nuy=17 nuz=1     fill  f210
   a40 261   a43 261   a46 261   a55 261   a65 261
  a88 261   a91 261   a94 261   a97 261  a100 261
 a139 261  a142 261  a145 261  a148 261  a151 261
 a190 261  a193 261  a196 261  a199 261  a202 261
 a225 261  a235 261  a244 261  a247 261  a250 261  end fill
'    *** position d10    xsec batch #5    no bps ***
```

```
ara=211   nux=17 nuy=17 nuz=1     fill  f211
   a40 261   a43 261   a46 261   a55 261   a65 261
  a88 261   a91 261   a94 261   a97 261  a100 261
 a139 261  a142 261  a145 261  a148 261  a151 261
 a190 261  a193 261  a196 261  a199 261  a202 261
 a225 261  a235 261  a244 261  a247 261  a250 261  end fill
'    *** position b10    xsec batch #5    no bps ***
ara=212   nux=17 nuy=17 nuz=1     fill  f212
   a40 261   a43 261   a46 261   a55 261   a65 261
  a88 261   a91 261   a94 261   a97 261  a100 261
 a139 261  a142 261  a145 261  a148 261  a151 261
 a190 261  a193 261  a196 261  a199 261  a202 261
 a225 261  a235 261  a244 261  a247 261  a250 261  end fill
'    *** position e11    xsec batch #5    no bps ***
ara=213   nux=17 nuy=17 nuz=1     fill  f213
   a40 261   a43 261   a46 261   a55 261   a65 261
  a88 261   a91 261   a94 261   a97 261  a100 261
 a139 261  a142 261  a145 261  a148 261  a151 261
 a190 261  a193 261  a196 261  a199 261  a202 261
 a225 261  a235 261  a244 261  a247 261  a250 261  end fill
'    *** position b11    xsec batch #3    no bps ***
ara=214   nux=17 nuy=17 nuz=1     fill  f214
   a40 261   a43 261   a46 261   a55 261   a65 261
  a88 261   a91 261   a94 261   a97 261  a100 261
 a139 261  a142 261  a145 261  a148 261  a151 261
 a190 261  a193 261  a196 261  a199 261  a202 261
 a225 261  a235 261  a244 261  a247 261  a250 261  end fill
'    *** position d12    xsec batch #4    no bps ***
ara=215   nux=17 nuy=17 nuz=1     fill  f215
   a40 261   a43 261   a46 261   a55 261   a65 261
  a88 261   a91 261   a94 261   a97 261  a100 261
 a139 261  a142 261  a145 261  a148 261  a151 261
 a190 261  a193 261  a196 261  a199 261  a202 261
 a225 261  a235 261  a244 261  a247 261  a250 261  end fill
'    *** position c12    xsec batch #1    no bps ***
ara=216   nux=17 nuy=17 nuz=1     fill  f216
   a40 261   a43 261   a46 261   a55 261   a65 261
  a88 261   a91 261   a94 261   a97 261  a100 261
 a139 261  a142 261  a145 261  a148 261  a151 261
 a190 261  a193 261  a196 261  a199 261  a202 261
 a225 261  a235 261  a244 261  a247 261  a250 261  end fill
'* position c11 only   xsec batch #6 (fresh fuel)  no bps *
ara=217   nux=17 nuy=17 nuz=1     fill  f217
   a40 261   a43 261   a46 261   a55 261   a65 261
  a88 261   a91 261   a94 261   a97 261  a100 261
 a139 261  a142 261  a145 261  a148 261  a151 261
 a190 261  a193 261  a196 261  a199 261  a202 261
 a225 261  a235 261  a244 261  a247 261  a250 261  end fill
'*position g8+c8+f9+c10+d11  (fresh fuel #6)  16 fresh bps*
ara=218   nux=17 nuy=17 nuz=1     fill  f217
   a40 262   a43 261   a46 262   a55 262   a65 262
  a88 262   a91 261   a94 262   a97 261  a100 262
```

```
 a139 261  a142 262  a145 261  a148 262  a151 261
 a190 262  a193 261  a196 262  a199 261  a202 262
 a225 262  a235 262  a244 262  a247 261  a250 262  end fill
'* position e8+d9+e10  (fresh fuel #6)  20 fresh bps *
ara=219   nux=17 nuy=17 nuz=1     fill  f217
   a40 262   a43 262   a46 262   a55 262   a65 262
  a88 262   a91 261   a94 262   a97 261  a100 262
 a139 262  a142 262  a145 261  a148 262  a151 262
 a190 262  a193 261  a196 262  a199 261  a202 262
 a225 262  a235 262  a244 262  a247 262  a250 262  end fill
'* position b9 only  (fresh fuel #6)  12 depleted bps *
ara=220   nux=17 nuy=17 nuz=1     fill  f217
   a40 263   a43 261   a46 263   a55 261   a65 261
  a88 263   a91 261   a94 263   a97 261  a100 263
 a139 261  a142 263  a145 261  a148 263  a151 261
 a190 263  a193 261  a196 263  a199 261  a202 263
 a225 261  a235 261  a244 263  a247 261  a250 263  end fill
'    ***** end array data for plane #2 *****
' array data for fuel assemblies 101-120
'    ***** array data for plane #3   *****
'    *** position h8    xsec batch #5    no bps ***
ara=301   nux=17 nuy=17 nuz=1     fill  f301
   a40 361   a43 361   a46 361   a55 361   a65 361
  a88 361   a91 361   a94 361   a97 361  a100 361
 a139 361  a142 361  a145 361  a148 361  a151 361
 a190 361  a193 361  a196 361  a199 361  a202 361
 a225 361  a235 361  a244 361  a247 361  a250 361  end fill
'    *** position f8    xsec batch #1    no bps ***
ara=302   nux=17 nuy=17 nuz=1     fill  f302
   a40 361   a43 361   a46 361   a55 361   a65 361
  a88 361   a91 361   a94 361   a97 361  a100 361
 a139 361  a142 361  a145 361  a148 361  a151 361
 a190 361  a193 361  a196 361  a199 361  a202 361
 a225 361  a235 361  a244 361  a247 361  a250 361  end fill
'    *** position d8    xsec batch #5    no bps ***
ara=303   nux=17 nuy=17 nuz=1     fill  f303
   a40 361   a43 361   a46 361   a55 361   a65 361
  a88 361   a91 361   a94 361   a97 361  a100 361
 a139 361  a142 361  a145 361  a148 361  a151 361
 a190 361  a193 361  a196 361  a199 361  a202 361
 a225 361  a235 361  a244 361  a247 361  a250 361  end fill
'    *** position b8    xsec batch #5    no bps ***
ara=304   nux=17 nuy=17 nuz=1     fill  f304
   a40 361   a43 361   a46 361   a55 361   a65 361
  a88 361   a91 361   a94 361   a97 361  a100 361
 a139 361  a142 361  a145 361  a148 361  a151 361
 a190 361  a193 361  a196 361  a199 361  a202 361
 a225 361  a235 361  a244 361  a247 361  a250 361  end fill
'    *** position a8    xsec batch #4    no bps ***
ara=305   nux=17 nuy=17 nuz=1     fill  f305
   a40 361   a43 361   a46 361   a55 361   a65 361
  a88 361   a91 361   a94 361   a97 361  a100 361
```

```
 a139 361  a142 361  a145 361  a148 361  a151 361
 a190 361  a193 361  a196 361  a199 361  a202 361
 a225 361  a235 361  a244 361  a247 361  a250 361  end fill
'    *** position g9    xsec batch #2    no bps ***
ara=306   nux=17 nuy=17 nuz=1     fill  f306
   a40 361   a43 361   a46 361   a55 361   a65 361
  a88 361   a91 361   a94 361   a97 361  a100 361
 a139 361  a142 361  a145 361  a148 361  a151 361
 a190 361  a193 361  a196 361  a199 361  a202 361
 a225 361  a235 361  a244 361  a247 361  a250 361  end fill
'    *** position e9    xsec batch #5   12 depleted bps ***
ara=307   nux=17 nuy=17 nuz=1     fill  f307
   a40 363   a43 361   a46 363   a55 361   a65 361
  a88 363   a91 361   a94 363   a97 361  a100 363
 a139 361  a142 363  a145 361  a148 363  a151 361
 a190 363  a193 361  a196 363  a199 361  a202 363
 a225 361  a235 361  a244 363  a247 361  a250 363  end fill
'    *** position c9    xsec batch #4    no bps ***
ara=308   nux=17 nuy=17 nuz=1     fill  f308
   a40 361   a43 361   a46 361   a55 361   a65 361
  a88 361   a91 361   a94 361   a97 361  a100 361
 a139 361  a142 361  a145 361  a148 361  a151 361
 a190 361  a193 361  a196 361  a199 361  a202 361
 a225 361  a235 361  a244 361  a247 361  a250 361  end fill
'    *** position a9    xsec batch #3    no bps ***
ara=309   nux=17 nuy=17 nuz=1     fill  f309
   a40 361   a43 361   a46 361   a55 361   a65 361
  a88 361   a91 361   a94 361   a97 361  a100 361
 a139 361  a142 361  a145 361  a148 361  a151 361
 a190 361  a193 361  a196 361  a199 361  a202 361
 a225 361  a235 361  a244 361  a247 361  a250 361  end fill
'    *** position f10    xsec batch #5    no bps ***
ara=310   nux=17 nuy=17 nuz=1     fill  f310
   a40 361   a43 361   a46 361   a55 361   a65 361
  a88 361   a91 361   a94 361   a97 361  a100 361
 a139 361  a142 361  a145 361  a148 361  a151 361
 a190 361  a193 361  a196 361  a199 361  a202 361
 a225 361  a235 361  a244 361  a247 361  a250 361  end fill
'    *** position d10    xsec batch #5    no bps ***
ara=311   nux=17 nuy=17 nuz=1     fill  f311
   a40 361   a43 361   a46 361   a55 361   a65 361
  a88 361   a91 361   a94 361   a97 361  a100 361
 a139 361  a142 361  a145 361  a148 361  a151 361
 a190 361  a193 361  a196 361  a199 361  a202 361
 a225 361  a235 361  a244 361  a247 361  a250 361  end fill
'    *** position b10    xsec batch #5    no bps ***
ara=312   nux=17 nuy=17 nuz=1     fill  f312
   a40 361   a43 361   a46 361   a55 361   a65 361
  a88 361   a91 361   a94 361   a97 361  a100 361
 a139 361  a142 361  a145 361  a148 361  a151 361
 a190 361  a193 361  a196 361  a199 361  a202 361
 a225 361  a235 361  a244 361  a247 361  a250 361  end fill
```

```
'    *** position e11    xsec batch #5    no bps ***
ara=313   nux=17 nuy=17 nuz=1     fill  f313
   a40 361   a43 361   a46 361   a55 361   a65 361
  a88 361   a91 361   a94 361   a97 361  a100 361
 a139 361  a142 361  a145 361  a148 361  a151 361
 a190 361  a193 361  a196 361  a199 361  a202 361
 a225 361  a235 361  a244 361  a247 361  a250 361  end fill
'    *** position b11    xsec batch #3    no bps ***
ara=314   nux=17 nuy=17 nuz=1     fill  f314
   a40 361   a43 361   a46 361   a55 361   a65 361
  a88 361   a91 361   a94 361   a97 361  a100 361
 a139 361  a142 361  a145 361  a148 361  a151 361
 a190 361  a193 361  a196 361  a199 361  a202 361
 a225 361  a235 361  a244 361  a247 361  a250 361  end fill
'    *** position d12    xsec batch #4    no bps ***
ara=315   nux=17 nuy=17 nuz=1     fill  f315
   a40 361   a43 361   a46 361   a55 361   a65 361
  a88 361   a91 361   a94 361   a97 361  a100 361
 a139 361  a142 361  a145 361  a148 361  a151 361
 a190 361  a193 361  a196 361  a199 361  a202 361
 a225 361  a235 361  a244 361  a247 361  a250 361  end fill
'    *** position c12    xsec batch #1    no bps ***
ara=316   nux=17 nuy=17 nuz=1     fill  f316
   a40 361   a43 361   a46 361   a55 361   a65 361
  a88 361   a91 361   a94 361   a97 361  a100 361
 a139 361  a142 361  a145 361  a148 361  a151 361
 a190 361  a193 361  a196 361  a199 361  a202 361
 a225 361  a235 361  a244 361  a247 361  a250 361  end fill
'* position c11 only   xsec batch #6 (fresh fuel)  no bps *
ara=317   nux=17 nuy=17 nuz=1     fill  f317
   a40 361   a43 361   a46 361   a55 361   a65 361
  a88 361   a91 361   a94 361   a97 361  a100 361
 a139 361  a142 361  a145 361  a148 361  a151 361
 a190 361  a193 361  a196 361  a199 361  a202 361
 a225 361  a235 361  a244 361  a247 361  a250 361  end fill
'*position g8+c8+f9+c10+d11  (fresh fuel #6)  16 fresh bps*
ara=318   nux=17 nuy=17 nuz=1     fill  f317
   a40 362   a43 361   a46 362   a55 362   a65 362
  a88 362   a91 361   a94 362   a97 361  a100 362
 a139 361  a142 362  a145 361  a148 362  a151 361
 a190 362  a193 361  a196 362  a199 361  a202 362
 a225 362  a235 362  a244 362  a247 361  a250 362  end fill
'* position e8+d9+e10  (fresh fuel #6)  20 fresh bps *
ara=319   nux=17 nuy=17 nuz=1     fill  f317
   a40 362   a43 362   a46 362   a55 362   a65 362
  a88 362   a91 361   a94 362   a97 361  a100 362
 a139 362  a142 362  a145 361  a148 362  a151 362
 a190 362  a193 361  a196 362  a199 361  a202 362
 a225 362  a235 362  a244 362  a247 362  a250 362  end fill
'*** position b9 only  (fresh fuel #6)  12 depleted bps ***
ara=320   nux=17 nuy=17 nuz=1     fill  f317
   a40 363   a43 361   a46 363   a55 361   a65 361
```

```
  a88 363    a91 361    a94 363    a97 361   a100 363
 a139 361   a142 363   a145 361   a148 363   a151 361
 a190 363   a193 361   a196 363   a199 361   a202 363
 a225 361   a235 361   a244 363   a247 361   a250 363   end fill
'     ***** end array data for plane #3 *****
end array
end data
end
```

Appendix F

Abbreviated Output Listing for the Timing Study Problems

sample problem 2 2c8 bare with 8 unit types matrix calculation

lifetime = 1.02642E-08 + or - 6.60315E-11 generation time = 7.03563E-09 + or - 6.56818E-11

no. of initial generations skipped	*average k-effective*	*deviation*	*67 per cent confidence interval*	*95 per cent confidence interval*	*99 per cent confidence interval*	*number of histories*
3	*1.00868*	*+ or - 0.00381*	*1.00486 to 1.01249*	*1.00105 to 1.01630*	*0.99724 to 1.02011*	*30000*
4	*1.00912*	*+ or - 0.00383*	*1.00529 to 1.01294*	*1.00147 to 1.01677*	*0.99764 to 1.02059*	*29700*
5	*1.00908*	*+ or - 0.00386*	*1.00521 to 1.01294*	*1.00135 to 1.01681*	*0.99748 to 1.02067*	*29400*
6	*1.00942*	*+ or - 0.00389*	*1.00553 to 1.01330*	*1.00164 to 1.01719*	*0.99775 to 1.02108*	*29100*
7	*1.00942*	*+ or - 0.00393*	*1.00549 to 1.01335*	*1.00156 to 1.01728*	*0.99763 to 1.02121*	*28800*
8	*1.00960*	*+ or - 0.00397*	*1.00564 to 1.01357*	*1.00167 to 1.01754*	*0.99770 to 1.02151*	*28500*
9	*1.00944*	*+ or - 0.00401*	*1.00543 to 1.01345*	*1.00143 to 1.01745*	*0.99742 to 1.02146*	*28200*
10	*1.00908*	*+ or - 0.00403*	*1.00505 to 1.01311*	*1.00101 to 1.01714*	*0.99698 to 1.02118*	*27900*
11	*1.00854*	*+ or - 0.00404*	*1.00450 to 1.01258*	*1.00046 to 1.01662*	*0.99642 to 1.02066*	*27600*
12	*1.00854*	*+ or - 0.00408*	*1.00446 to 1.01263*	*1.00038 to 1.01671*	*0.99629 to 1.02080*	*27300*
17	*1.01033*	*+ or - 0.00416*	*1.00617 to 1.01449*	*1.00201 to 1.01865*	*0.99785 to 1.02281*	*25800*
22	*1.01162*	*+ or - 0.00426*	*1.00736 to 1.01588*	*1.00310 to 1.02013*	*0.99884 to 1.02439*	*24300*
27	*1.00935*	*+ or - 0.00431*	*1.00505 to 1.01366*	*1.00074 to 1.01797*	*0.99643 to 1.02228*	*22800*
32	*1.00973*	*+ or - 0.00447*	*1.00526 to 1.01420*	*1.00079 to 1.01867*	*0.99632 to 1.02314*	*21300*
37	*1.01185*	*+ or - 0.00451*	*1.00734 to 1.01635*	*1.00283 to 1.02086*	*0.99833 to 1.02537*	*19800*
42	*1.01031*	*+ or - 0.00466*	*1.00565 to 1.01497*	*1.00099 to 1.01962*	*0.99633 to 1.02428*	*18300*
47	*1.00809*	*+ or - 0.00479*	*1.00330 to 1.01287*	*0.99851 to 1.01766*	*0.99372 to 1.02245*	*16800*
52	*1.01046*	*+ or - 0.00502*	*1.00544 to 1.01548*	*1.00041 to 1.02051*	*0.99539 to 1.02553*	*15300*
57	*1.01253*	*+ or - 0.00533*	*1.00720 to 1.01786*	*1.00187 to 1.02319*	*0.99654 to 1.02852*	*13800*
62	*1.01399*	*+ or - 0.00539*	*1.00860 to 1.01938*	*1.00322 to 1.02477*	*0.99783 to 1.03015*	*12300*
67	*1.01337*	*+ or - 0.00584*	*1.00753 to 1.01922*	*1.00169 to 1.02506*	*0.99585 to 1.03090*	*10800*
72	*1.01555*	*+ or - 0.00645*	*1.00911 to 1.02200*	*1.00266 to 1.02844*	*0.99621 to 1.03489*	*9300*
77	*1.01652*	*+ or - 0.00692*	*1.00959 to 1.02344*	*1.00267 to 1.03036*	*0.99575 to 1.03729*	*7800*
82	*1.02037*	*+ or - 0.00771*	*1.01266 to 1.02808*	*1.00495 to 1.03579*	*0.99723 to 1.04350*	*6300*
87	*1.02247*	*+ or - 0.00969*	*1.01278 to 1.03216*	*1.00309 to 1.04185*	*0.99341 to 1.05154*	*4800*
92	*1.02194*	*+ or - 0.01158*	*1.01036 to 1.03352*	*0.99878 to 1.04510*	*0.98720 to 1.05668*	*3300*

sample problem 20 triangular pitched array

lifetime = 4.84650E-06 + or - 7.74914E-08 *generation time = 5.33937E-06 + or - 5.54191E-08*

no. of initial generations skipped	*average k-effective*	*deviation*	*67 per cent confidence inuerval*	*95 per cent confidence inuerval*	*99 per cent confidence inuerval*	*number of histories*
3	*1.00014*	*+ or - 0.00615*	*0.99399 to 1.00629*	*0.98784 to 1.01244*	*0.98169 to 1.01859*	*30000*
4	*1.00044*	*+ or - 0.00621*	*0.99423 to 1.00664*	*0.98803 to 1.01285*	*0.98182 to 1.01905*	*29700*
5	*1.00019*	*+ or - 0.00626*	*0.99392 to 1.00645*	*0.98766 to 1.01271*	*0.98140 to 1.01898*	*29400*
6	*1.00052*	*+ or - 0.00632*	*0.99420 to 1.00684*	*0.98788 to 1.01316*	*0.98156 to 1.01948*	*29100*
7	*0.99966*	*+ or - 0.00633*	*0.99333 to 1.00598*	*0.98701 to 1.01231*	*0.98068 to 1.01863*	*28800*
8	*0.99946*	*+ or - 0.00639*	*0.99307 to 1.00585*	*0.98668 to 1.01224*	*0.98029 to 1.01863*	*28500*
9	*0.99962*	*+ or - 0.00646*	*0.99316 to 1.00608*	*0.98671 to 1.01253*	*0.98025 to 1.01899*	*28200*
10	*1.00000*	*+ or - 0.00651*	*0.99349 to 1.00651*	*0.98697 to 1.01303*	*0.98046 to 1.01954*	*27900*
11	*0.99931*	*+ or - 0.00655*	*0.99276 to 1.00586*	*0.98622 to 1.01241*	*0.97967 to 1.01896*	*27600*
12	*0.99928*	*+ or - 0.00662*	*0.99266 to 1.00590*	*0.98604 to 1.01252*	*0.97942 to 1.01914*	*27300*
17	*0.99861*	*+ or - 0.00690*	*0.99171 to 1.00551*	*0.98481 to 1.01240*	*0.97791 to 1.01930*	*25800*
22	*1.00260*	*+ or - 0.00703*	*0.99557 to 1.00964*	*0.98854 to 1.01667*	*0.98151 to 1.02370*	*24300*
27	*1.00280*	*+ or - 0.00724*	*0.99556 to 1.01004*	*0.98832 to 1.01728*	*0.98108 to 1.02451*	*22800*
32	*1.00210*	*+ or - 0.00764*	*0.99446 to 1.00974*	*0.98682 to 1.01739*	*0.97917 to 1.02503*	*21300*
37	*1.00087*	*+ or - 0.00808*	*0.99279 to 1.00895*	*0.98471 to 1.01704*	*0.97662 to 1.02512*	*19800*
42	*0.99796*	*+ or - 0.00836*	*0.98961 to 1.00632*	*0.98125 to 1.01468*	*0.97289 to 1.02303*	*18300*
47	*0.99736*	*+ or - 0.00892*	*0.98844 to 1.00629*	*0.97951 to 1.01521*	*0.97059 to 1.02413*	*16800*
52	*0.99414*	*+ or - 0.00948*	*0.98466 to 1.00362*	*0.97518 to 1.01310*	*0.96570 to 1.02258*	*15300*
57	*0.99408*	*+ or - 0.00987*	*0.98421 to 1.00394*	*0.97434 to 1.01381*	*0.96448 to 1.02368*	*13800*
62	*0.99488*	*+ or - 0.01069*	*0.98419 to 1.00556*	*0.97350 to 1.01625*	*0.96282 to 1.02694*	*12300*
67	*0.99338*	*+ or - 0.01186*	*0.98152 to 1.00523*	*0.96966 to 1.01709*	*0.95781 to 1.02894*	*10800*
72	*0.99665*	*+ or - 0.01309*	*0.98357 to 1.00974*	*0.97048 to 1.02283*	*0.95739 to 1.03591*	*9300*
77	*0.99248*	*+ or - 0.01408*	*0.97840 to 1.00656*	*0.96432 to 1.02064*	*0.95024 to 1.03472*	*7800*
82	*0.99552*	*+ or - 0.01649*	*0.97903 to 1.01201*	*0.96254 to 1.02850*	*0.94605 to 1.04499*	*6300*
87	*0.99064*	*+ or - 0.02026*	*0.97038 to 1.01090*	*0.95012 to 1.03117*	*0.92986 to 1.05143*	*4800*
92	*0.99092*	*+ or - 0.02564*	*0.96528 to 1.01656*	*0.93964 to 1.04221*	*0.91400 to 1.06785*	*3300*

sample problem 21 partially filled sphere

lifetime = 1.02979E-04 + or - 6.34500E-07 *generation time = 9.53661E-05 + or - 5.01877E-07*

no. of initial generations skipped	*average k-effective*	*deviation*	*67 per cent confidence interval*	*95 per cent confidence interval*	*99 per cent confidence interval*	*number of histories*
3	0.99146	+ or - 0.00314	0.98832 to 0.99460	0.98518 to 0.99774	0.98205 to 1.00087	30000
4	0.99187	+ or - 0.00314	0.98873 to 0.99501	0.98558 to 0.99815	0.98244 to 1.00130	29700
5	0.99227	+ or - 0.00315	0.98912 to 0.99541	0.98597 to 0.99856	0.98282 to 1.00171	29400
6	0.99297	+ or - 0.00310	0.98987 to 0.99607	0.98677 to 0.99917	0.98367 to 1.00228	29100
7	0.99299	+ or - 0.00313	0.98985 to 0.99612	0.98672 to 0.99925	0.98359 to 1.00239	28800
8	0.99361	+ or - 0.00310	0.99051 to 0.99671	0.98741 to 0.99982	0.98430 to 1.00292	28500
9	0.99364	+ or - 0.00314	0.99050 to 0.99677	0.98737 to 0.99991	0.98423 to 1.00305	28200
10	0.99322	+ or - 0.00314	0.99008 to 0.99636	0.98694 to 0.99950	0.98379 to 1.00264	27900
11	0.99317	+ or - 0.00318	0.99000 to 0.99635	0.98682 to 0.99952	0.98365 to 1.00270	27600
12	0.99287	+ or - 0.00320	0.98967 to 0.99606	0.98648 to 0.99926	0.98328 to 1.00245	27300
17	0.99327	+ or - 0.00331	0.98997 to 0.99658	0.98666 to 0.99989	0.98336 to 1.00319	25800
22	0.99549	+ or - 0.00326	0.99223 to 0.99875	0.98896 to 1.00202	0.98570 to 1.00528	24300
27	0.99572	+ or - 0.00343	0.99229 to 0.99915	0.98885 to 1.00259	0.98542 to 1.00602	22800
32	0.99530	+ or - 0.00357	0.99174 to 0.99887	0.98817 to 1.00243	0.98460 to 1.00600	21300
37	0.99674	+ or - 0.00361	0.99313 to 1.00036	0.98952 to 1.00397	0.98591 to 1.00758	19800
42	0.99544	+ or - 0.00381	0.99162 to 0.99925	0.98781 to 1.00307	0.98399 to 1.00688	18300
47	0.99711	+ or - 0.00395	0.99316 to 1.00105	0.98922 to 1.00500	0.98527 to 1.00894	16800
52	0.99836	+ or - 0.00404	0.99432 to 1.00240	0.99027 to 1.00644	0.98623 to 1.01049	15300
57	1.00027	+ or - 0.00422	0.99604 to 1.00449	0.99182 to 1.00872	0.98760 to 1.01294	13800
62	1.00152	+ or - 0.00423	0.99729 to 1.00574	0.99307 to 1.00997	0.98884 to 1.01419	12300
67	0.99982	+ or - 0.00438	0.99545 to 1.00420	0.99107 to 1.00858	0.98669 to 1.01296	10800
72	0.99859	+ or - 0.00492	0.99368 to 1.00351	0.98876 to 1.00842	0.98385 to 1.01334	9300
77	0.99727	+ or - 0.00521	0.99206 to 1.00247	0.98686 to 1.00768	0.98165 to 1.01288	7800
82	0.99723	+ or - 0.00586	0.99137 to 1.00309	0.98551 to 1.00895	0.97965 to 1.01481	6300
87	0.99588	+ or - 0.00751	0.98838 to 1.00339	0.98087 to 1.01090	0.97336 to 1.01841	4800
92	0.99635	+ or - 0.00911	0.98724 to 1.00546	0.97813 to 1.01457	0.96902 to 1.02368	3300

case 1 pwr castor v/21 storage cask 4/85 fn=llnl.cnul(pwrcskv1)

lifetime = 4.76561E-05 + or - 4.67486E-07 *generation time = 3.07807E-05 + or - 2.22178E-07*
nu bar = 4.64618E+00 + or - 7.30395E-03 *average fission group = 2.00266E+01 + or - 2.76291E-02*

no. of initial generations skipped	*average k-effective*	*deviation*	*67 per cent confidence interval*	*95 per cent confidence interval*	*99 per cent confidence interval*	*number of histories*
3	*0.91565*	*+ or - 0.00451*	*0.91114 to 0.92016*	*0.90663 to 0.92467*	*0.90212 to 0.92918*	*29400*
4	*0.91618*	*+ or - 0.00457*	*0.91161 to 0.92076*	*0.90704 to 0.92533*	*0.90246 to 0.92990*	*28800*
5	*0.91675*	*+ or - 0.00464*	*0.91211 to 0.92138*	*0.90748 to 0.92602*	*0.90284 to 0.93065*	*28200*
6	*0.91704*	*+ or - 0.00473*	*0.91232 to 0.92177*	*0.90759 to 0.92650*	*0.90286 to 0.93123*	*27600*
7	*0.91753*	*+ or - 0.00481*	*0.91272 to 0.92234*	*0.90791 to 0.92714*	*0.90310 to 0.93195*	*27000*
8	*0.91715*	*+ or - 0.00490*	*0.91225 to 0.92205*	*0.90734 to 0.92696*	*0.90244 to 0.93186*	*26400*
9	*0.91789*	*+ or - 0.00496*	*0.91293 to 0.92285*	*0.90796 to 0.92781*	*0.90300 to 0.93278*	*25800*
10	*0.91706*	*+ or - 0.00501*	*0.91205 to 0.92208*	*0.90704 to 0.92709*	*0.90203 to 0.93210*	*25200*
11	*0.91701*	*+ or - 0.00514*	*0.91188 to 0.92215*	*0.90674 to 0.92728*	*0.90161 to 0.93242*	*24600*
12	*0.91691*	*+ or - 0.00526*	*0.91164 to 0.92217*	*0.90638 to 0.92743*	*0.90111 to 0.93270*	*24000*
17	*0.91886*	*+ or - 0.00581*	*0.91305 to 0.92466*	*0.90724 to 0.93047*	*0.90144 to 0.93628*	*21000*
22	*0.91995*	*+ or - 0.00630*	*0.91366 to 0.92625*	*0.90736 to 0.93254*	*0.90107 to 0.93884*	*18000*
27	*0.91929*	*+ or - 0.00726*	*0.91204 to 0.92655*	*0.90478 to 0.93381*	*0.89752 to 0.94106*	*15000*
32	*0.92106*	*+ or - 0.00800*	*0.91306 to 0.92906*	*0.90506 to 0.93707*	*0.89706 to 0.94507*	*12000*
37	*0.92104*	*+ or - 0.00917*	*0.91187 to 0.93021*	*0.90271 to 0.93937*	*0.89354 to 0.94854*	*9000*
42	*0.92751*	*+ or - 0.01238*	*0.91513 to 0.93990*	*0.90275 to 0.95228*	*0.89036 to 0.96466*	*6000*
47	*0.94835*	*+ or - 0.01845*	*0.92990 to 0.96679*	*0.91145 to 0.98524*	*0.89300 to 1.00369*	*3000*

31 element burnup credit cask, load 31.5gwd/mtu

lifetime = 3.30313E-05 + or - 2.54446E-07 *generation time = 2.83174E-05 + or - 8.27375E-08*

no. of initial generations skipped	*average k-effective*	*deviation*	*67 per cent confidence interval*	*95 per cent confidence interval*	*99 per cent confidence interval*	*number of histories*
3	*0.80457*	*+ or - 0.00146*	*0.80311 to 0.80603*	*0.80164 to 0.80749*	*0.80018 to 0.80895*	*200000*
4	*0.80503*	*+ or - 0.00141*	*0.80362 to 0.80645*	*0.80220 to 0.80786*	*0.80079 to 0.80927*	*196000*
5	*0.80524*	*+ or - 0.00143*	*0.80381 to 0.80667*	*0.80239 to 0.80810*	*0.80096 to 0.80953*	*192000*
6	*0.80532*	*+ or - 0.00146*	*0.80387 to 0.80678*	*0.80241 to 0.80824*	*0.80096 to 0.80969*	*188000*
7	*0.80577*	*+ or - 0.00142*	*0.80435 to 0.80719*	*0.80294 to 0.80860*	*0.80152 to 0.81002*	*184000*
8	*0.80591*	*+ or - 0.00144*	*0.80447 to 0.80735*	*0.80303 to 0.80879*	*0.80159 to 0.81024*	*180000*
9	*0.80627*	*+ or - 0.00143*	*0.80484 to 0.80770*	*0.80342 to 0.80913*	*0.80199 to 0.81055*	*176000*
10	*0.80694*	*+ or - 0.00129*	*0.80564 to 0.80823*	*0.80435 to 0.80952*	*0.80306 to 0.81082*	*172000*
11	*0.80751*	*+ or - 0.00119*	*0.80632 to 0.80870*	*0.80514 to 0.80988*	*0.80395 to 0.81107*	*168000*
12	*0.80780*	*+ or - 0.00118*	*0.80662 to 0.80898*	*0.80544 to 0.81016*	*0.80426 to 0.81134*	*164000*
17	*0.80799*	*+ or - 0.00130*	*0.80669 to 0.80929*	*0.80540 to 0.81059*	*0.80410 to 0.81189*	*144000*
22	*0.80887*	*+ or - 0.00133*	*0.80755 to 0.81020*	*0.80622 to 0.81153*	*0.80489 to 0.81286*	*124000*
27	*0.80852*	*+ or - 0.00138*	*0.80714 to 0.80990*	*0.80576 to 0.81128*	*0.80438 to 0.81266*	*104000*
32	*0.80924*	*+ or - 0.00131*	*0.80792 to 0.81055*	*0.80661 to 0.81186*	*0.80529 to 0.81318*	*84000*
37	*0.80887*	*+ or - 0.00164*	*0.80723 to 0.81051*	*0.80559 to 0.81215*	*0.80395 to 0.81379*	*64000*
42	*0.80848*	*+ or - 0.00230*	*0.80618 to 0.81078*	*0.80388 to 0.81308*	*0.80158 to 0.81538*	*44000*
47	*0.81020*	*+ or - 0.00089*	*0.80931 to 0.81109*	*0.80843 to 0.81198*	*0.80754 to 0.81287*	*24000*

VITA

Daniel F. Hollenbach was born in Columbia, Missouri on April 13, 1956. He lived the first 13 years of his life in Milwaukee, Wisconsin and moved to Charlotte, North Carolina at the age of 13 where he graduated from Charlotte Catholic High School in June 1972. After attending one year of college at North Carolina State University, he served 4 years in the U.S. Army as a Russian Linguist. Upon being honorably discharged, he returned to North Carolina State University where he received a B.S. in Nuclear Engineering in May, 1982. The following September he entered the University of Washington where he received an M.S. in Nuclear Engineering in December, 1983. In November 1983 he began working as a Reactor Inspector for the U. S. Nuclear Regulatory Commission. In February, 1985, he resigned from the NRC and worked in the Refueling Division at Puget Sound Naval Shipyard. Finally, in August 1988 he left Puget Sound Naval Shipyard and enrolled in the University of Tennessee. In August 1992 he received a Doctor of Philosophy in Nuclear Engineering from the University of Tennessee.

He has accepted a position with Martin Marietta Energy Systems in the Computing and Telecommunications Division at Oak Ridge National Laboratory.