Monte Carlo simulation: Optimization of computer time and memory

Sachin Shet[1] and K.V.Subbaiah[2]
[1]Junior Research Fellow, [2]Consultant Professor
Manipal Centre for Natural Sciences
Manipal University
Manipal-576104, Karnataka
Email: iamsachinshet@gmail.com

Technical Report

Ref:Stream/ Basic Science/BS-RF005

Work is done under the project:

DAE-BRNS Project Ref. No. :36(4)/14/40/2015/36007

Presentation Layout

- **Monte Carlo Method**
  - **What is Monte Carlo Method**
  - **Various Applications**
  - **Estimation of Finite integral**
  - **Estimation of ‘π’ with Buffon’s needle experiment**
  - **Basic elements**
- **Pseudo Random number generator**
- **Dynamic Allocation**
- **Unionized energy Grid**
- **Searching Algorithms**
- **Conclusions**
- **References**

Monte-Carlo Method

What is Monte-Carlo Method

Monte Carlo methods are a class of computational algorithms for simulating the behavior of various physical and mathematical systems, in which observations are randomly generated from the model.

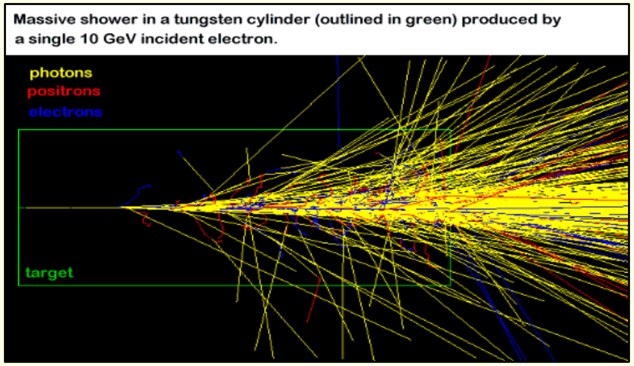
History

In 1777s : Estimation of Pi - Buffon
In 1940s : Named as Monte Carlo by Enricho Fermi, Stan Ulam and Van Neumann.

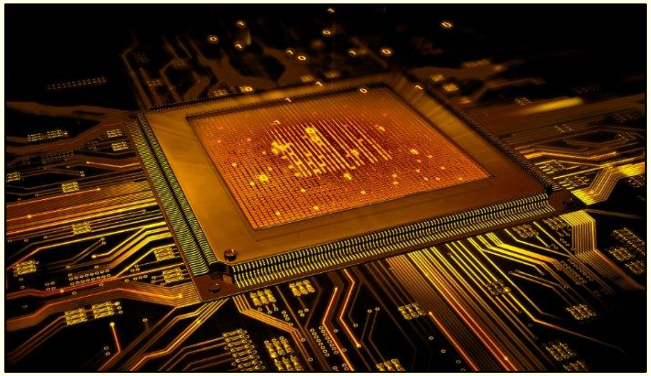
Monte-Carlo Method

Applications of Monte-Carlo Method

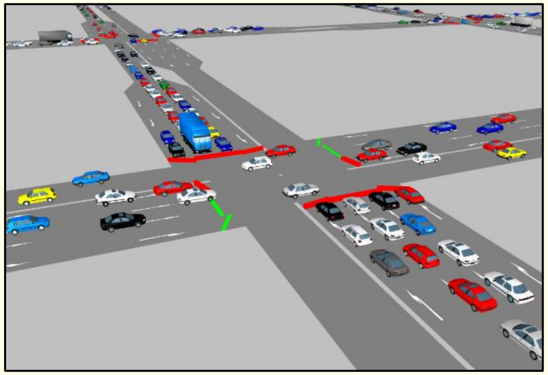
This method is used for solving problems in many branches of science, they are :



Radiation transport problems

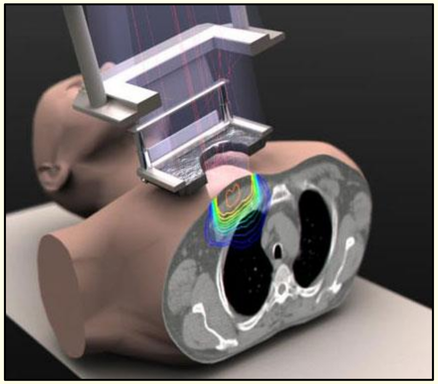


VLSI design



Traffic flow

Dow-Jones weather forecasting



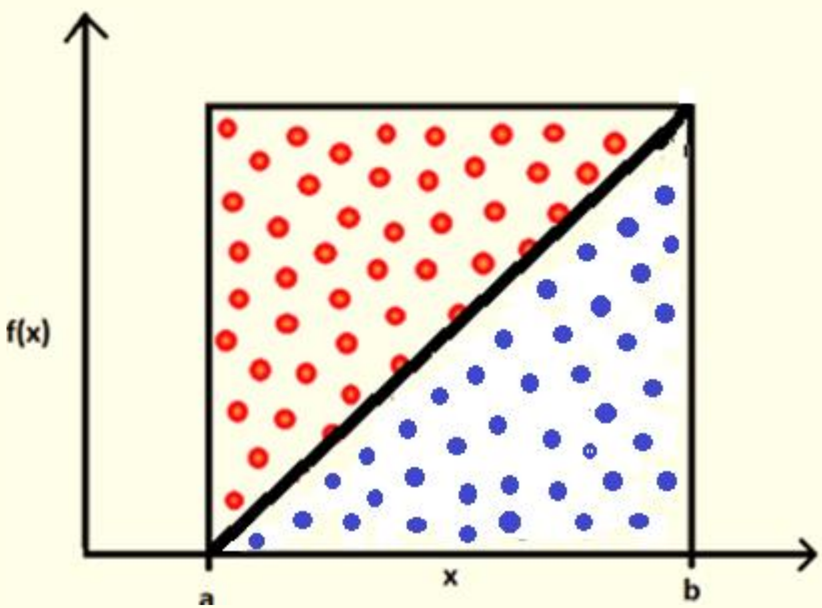
Radiation therapy

Monte-Carlo Method

Estimation of a finite Integral: Deterministic Method

- *Deterministic system : In which the later states of the system are determined by the earlier ones.*
- This method enables artificial construction of a probabilistic model.

$$\int_a^b f(x)\,dx \approx \max(f(x))\,(b-a)\,\frac{\textit{fraction of points under the line}}{\textit{Total number of points}}$$



f(x) = x

$$\int_0^1 \mathbf{x\,dx} = \mathbf{0.5}$$

Number of trials	Points enclosed by the curve	Value of the integral
10	6	0.600
100	56	0.56
1000	497	0.497
10000	5001	0.5001

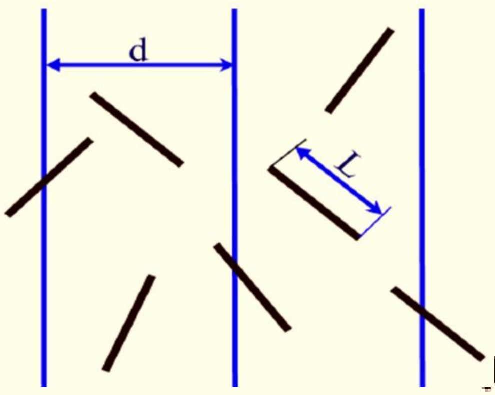
Monte-Carlo Method

Buffon's needle experiment [1777s]: Estimation of ' π '

$$\int_{\theta=0}^{\pi/2} \int_{x=0}^{\left(\frac{L}{2}\right)\sin\theta} \frac{4}{d\pi}\, dx\, d\theta = 2L/\pi d$$

$$P = \frac{\#\, needles\ crossed\ the\ line}{\#\, total\ needles} = 2L/\pi d$$

$$\pi = \frac{2L}{Pd}$$



Geometry of needle position in experiment

π = 3.141592

In 1901
Italianan
Mathematician
Mario Lazzarini
Tossed needle 3408 time
Pi = 3.14159292

Number of trials	Value of Pi
100	3.26086957
1000	3.06122449
10000	3.12434909
100000	3.14348005
1000000	3.13580546
10000000	3.14196728
100000000	3.14205106

Basic elements of Monte Carlo method:

- Random number generator
- Sampling of Probability distribution functions
- Scoring /tally specification
- Error estimation ($1/\sqrt{N}$, N = number of trials)
- Parallelization and vectorization

Pseudo Random number Generator :

Optimization of Computer time.

- **Random numbers are the back bone of all the Monte Carlo methods .**
- **Monte Carlo simulation make use of random numbers generated by computer algorithms .**

Required properties of Pseudo Random number generators

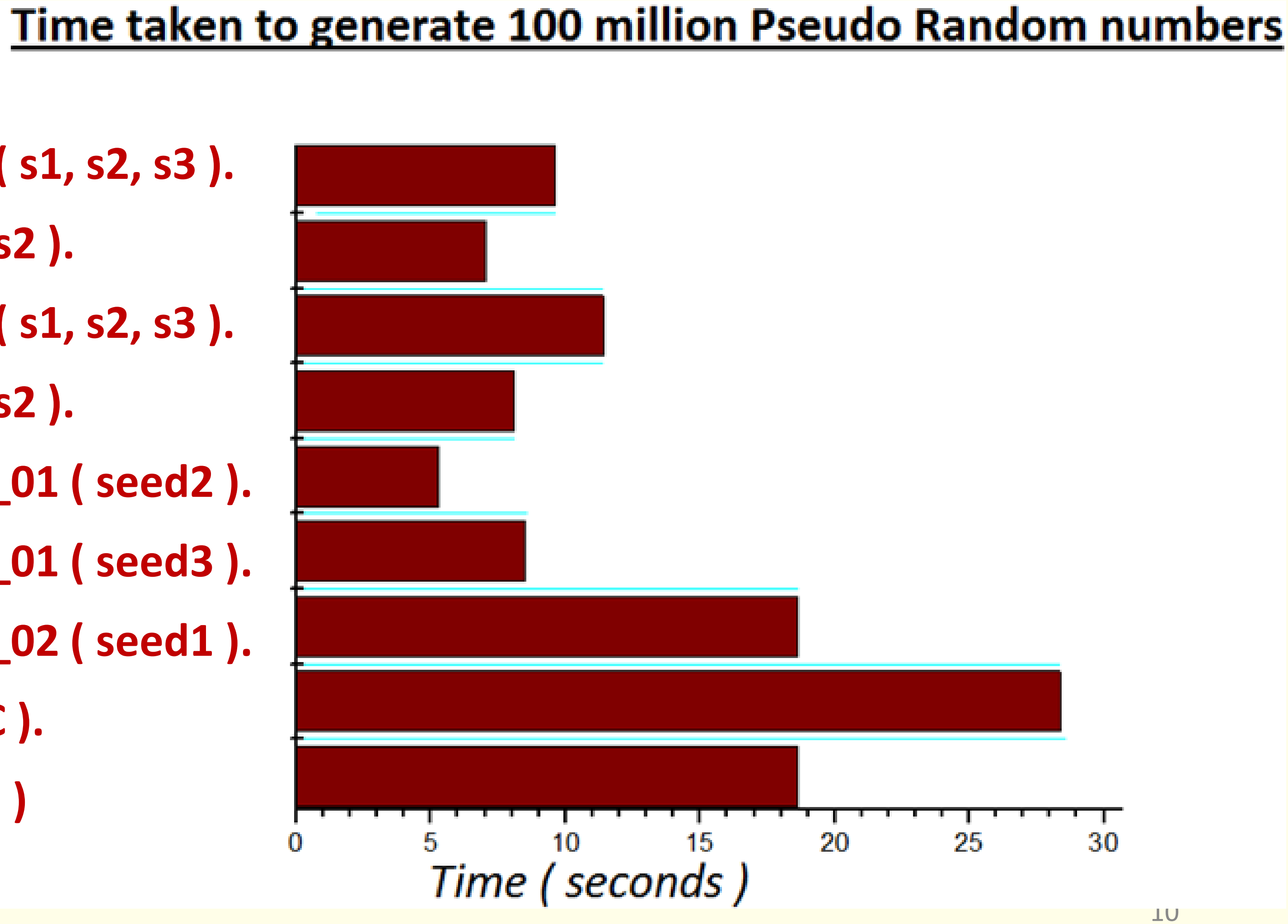
- **Uniformity**
- **Long Period**
- **Reproducibility**
- **Minimum number of Arithmetic steps**

Description	Seeds	Period	Average
1. **r4_random (s1, s2, s3).** Linear Congruential 32 bit generator.[1]	**No of Seeds: 3** 1< S1, S2, S3 < 30000	6.9×10^{12}	0.4998
2. **r4_uni (s1, s2).** Linear Congruential 32 bit generator.[1]	**No of Seeds: 2** 1< S1,S2 <2147483562	2.3×10^{18}	0.5000
3. **r8_random (s1, s2, s3).** Linear Congruential 64 bit generator.[1]	**No of Seeds: 3** 1 < S1, S2, S3 < 30000	6.9×10^{12}	0.4998
4. **r8_uni (s1, s2).** Linear Congruential 64 bit generator.[1]	**No of Seeds:2** 1< S1,S2 <2147483562	2.3×10^{18}	0.5000
5. **r4_uniform_01 (seed2).** Multiplicative pseudo random number generator.[2] [3] [4] [5] 32 bit generator	**No of Seeds: 1** 1< S1 <2147483562	NA	0.4997
6. **r8_uniform_01 (seed3).** Multiplicative pseudo random number generator.[2] [3] [4] [5] 64 bit generator	**No of Seeds: 1** 1< S1 <2147483562	NA	0.4997
7. **r8_uniform_02 (seed1).** Multiplicative pseudo random number generator.[2] [3] [4] [5] 64 bit generator	**No of Seeds:1** 1< S1 <2147483562	NA	0.4997
8. **ranC(idumC).** Multiplicative pseudo random number generator.[6] It is a 32 bit generator, and it uses intrinsic MIN MAX functions.	**No of Seeds:1** S1 = Negative integer	$>2\times10^{18}$	0.4999
9. **ranE(idumE)** Multiplicative pseudo random number generator. [6] Uses absolute value function and is a 32 bit generator.	**No of Seeds:1** S1 = Negative integer	$>2\times10^{18}$	0.5003

Pseudo Random number Generator :

Results of 9 Pseudo Random number generators:

1. **r4_random (s1, s2, s3).**
2. **r4_uni (s1, s2).**
3. **r8_random (s1, s2, s3).**
4. **r8_uni (s1, s2).**
5. **r4_uniform_01 (seed2).**
6. **r8_uniform_01 (seed3).**
7. **r8_uniform_02 (seed1).**
8. **ranC(idumC).**
9. **ranE(idumE)**

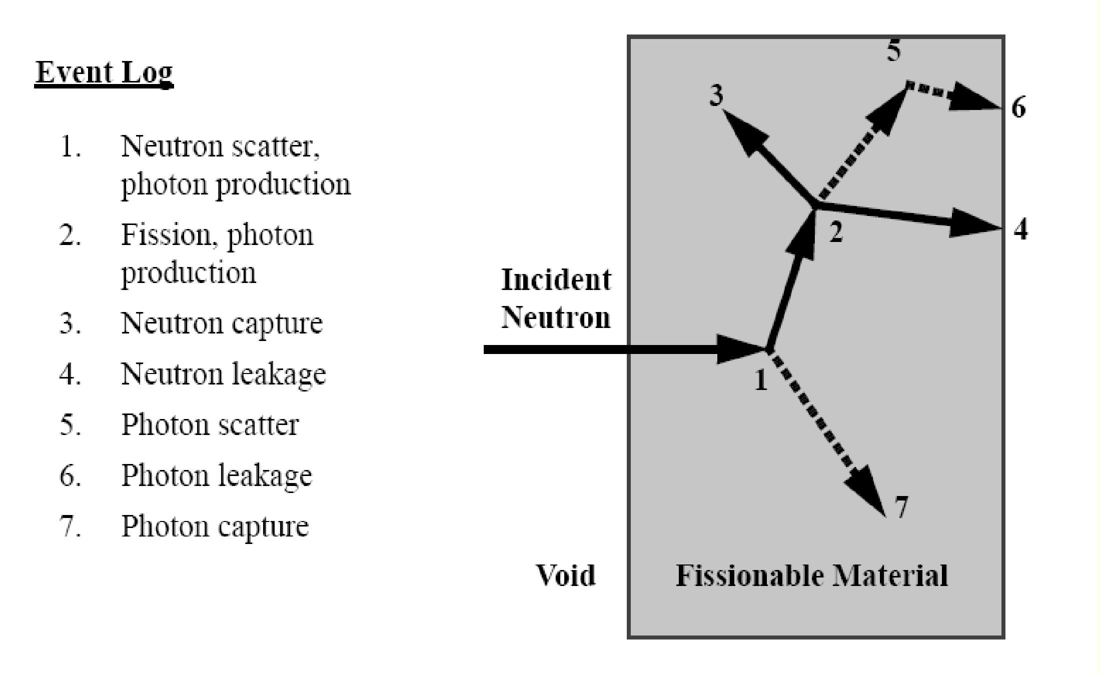


Memory Allocation:

Optimization of memory in radiation transport problem :

- **Monte Carlo simulation requires the interaction probabilities of radiation with matter**

For ***Neutron*** transport problem:



The data table of nuclides contains large number of data points in it.

Data files

Hydrogen	**Uranium**
.	. . . . .
Few thousands	**Few Millions**

Memory Allocation:

Static Allocation :

Size of the memory allocated is fixed, can not be altered during the execution.

Fortran: -------------- Allocate (Array(3,3))----------------

Hydrogen *Allocated Memory*

Uranium

Dynamic Allocation :

The size and the amount of storage can be altered during the execution of the program

-------- Real , Dimension (: , :) , Allocatable : : Array -----

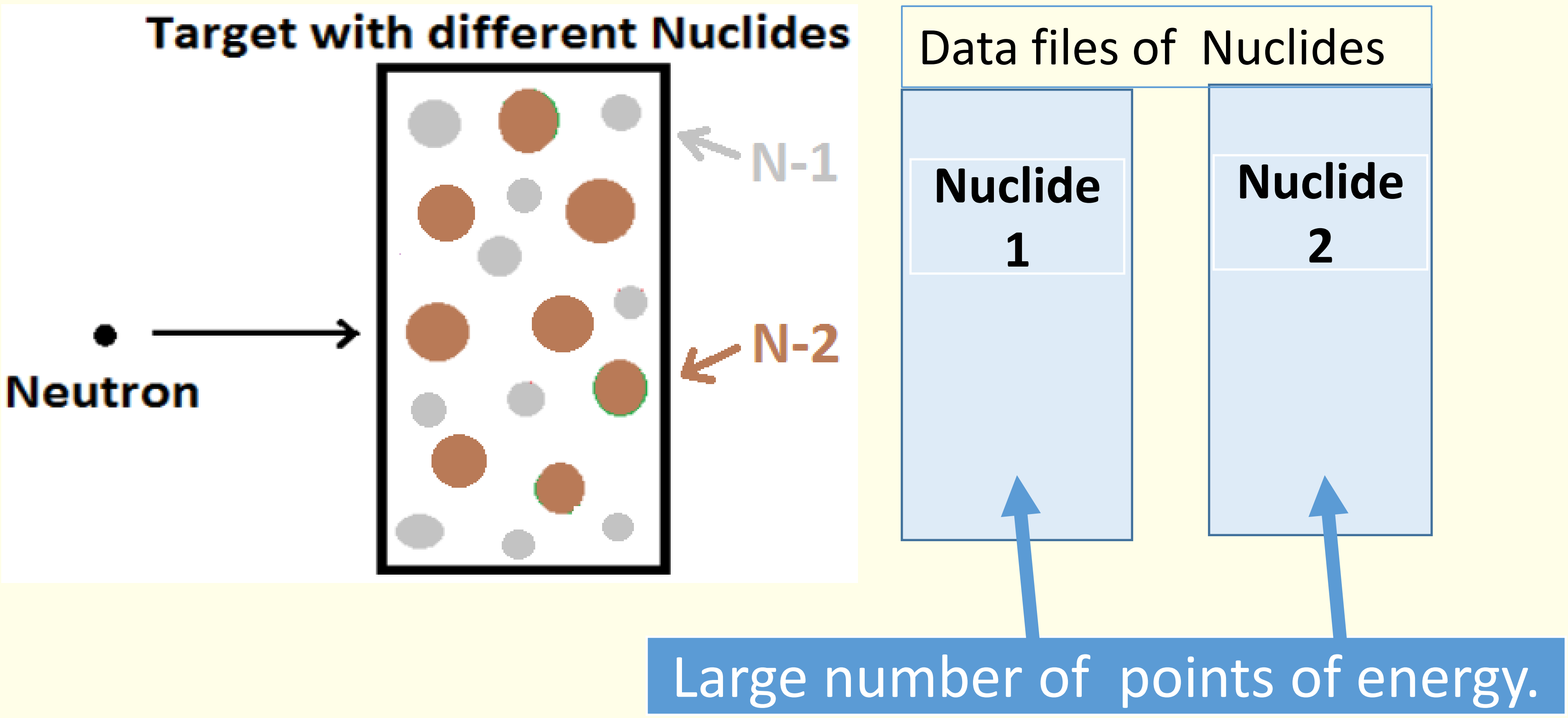
***Fortran:* --------- Allocate (Array (N , N)) --------**

Deallocate ***Free Memory***

Free Memory

Radiation transport problem:

more than one nuclide in target



Unionized energy grid:

<u>If two energy grids are</u> **Overlapping**

n(N1)+n(N2)-n(N1∩N2)

Nuclide 1		Nuclide 2
E-1		E-3
E-2		E-4
E-3	U	E-5
E-4		E-6
E-5		E-7

[n(N1) = 5] ***[n(N2) = 5]***

Unionized Energy

E-1
E-2
E-3
E-4
E-5
E-6
E-7

n(N1 U N2) = 7

Unionized energy grid:

If two energy grids are Identical

$n(N1 \cup N2) = n(N1)$

Nuclide 1		Nuclide 2	Unionized Energy
E-1		E-1	E-1
E-2		E-2	E-2
E-3	U	E-3	E-3
E-4		E-4	E-4
E-5		E-5	E-5

[n(N1) = 5] *[n(N2) = 5]* *n(N1 U N2) = 5*

Unionized energy grid:

<u>If two energy grids are</u> **Disjoint**

n(N1∪N2) = (N1)+n(N2)

Nuclide 1		Nuclide 2
E-1		E-6
E-2		E-7
E-3	U	E-8
E-4		E-9
E-5		E-10

Unionized Energy

E-1	E-6
E-2	E-7
E-3	E-8
E-4	E-9
E-5	E-10

[n(N1) = 5] ***[n(N2) = 5]*** ***n(N1 U N2) = 10***

Searching Algorithm:

Binary Search:

In the given Array A[ ], We have to search for the index i such that A[i] = Search value

a[lo] ≤ Search value ≤ **a[hi].**

6	**13**	**14**	**25**	**33**	**43**	**51**	**53**	**64**	**72**	**84**	**93**	**95**	**96**	**97**
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
↑ lo							↑ mid							↑ hi

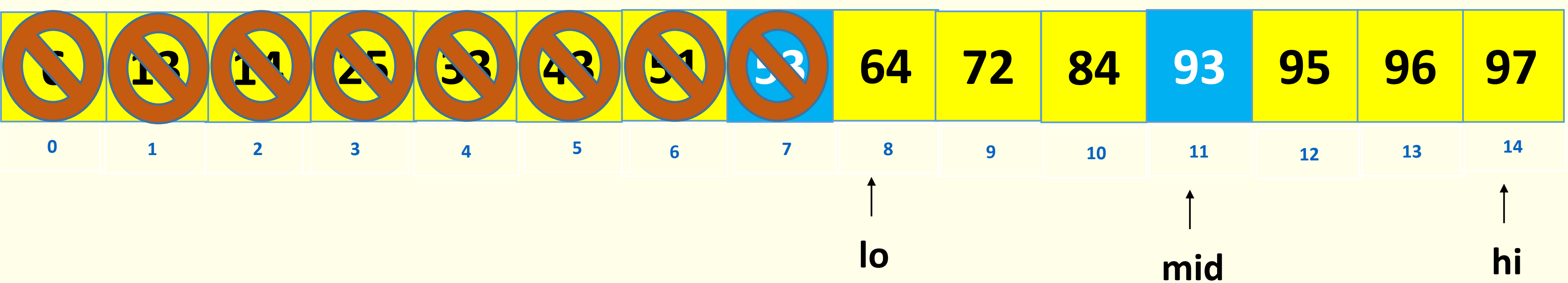
- Ex. ***Binary search for 95.***

Searching Algorithm:

Binary Search:

In the given Array A[ ], We have to search for the index i such that A[i] = Search value

a[lo] ≤ Search value ≤ **a[hi].**

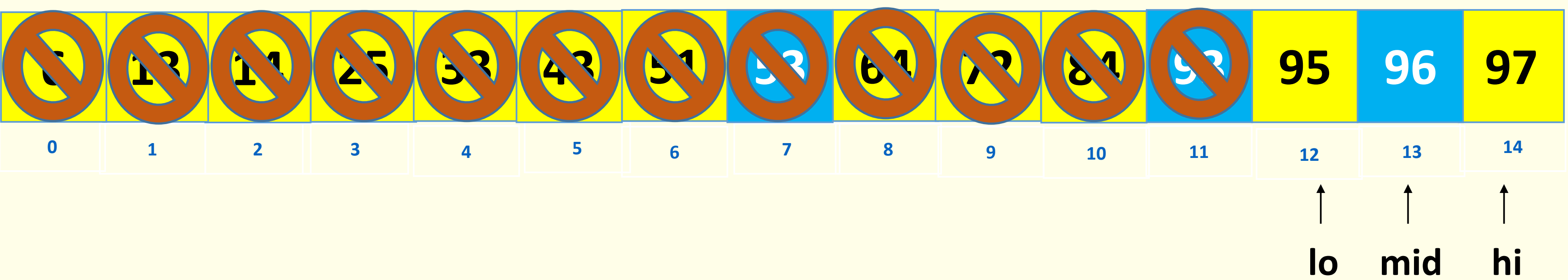


- Ex. ***Binary search for 95.***

Searching Algorithm:

Binary Search:

In the given Array A[ ], We have to search for the index i
such that A[i] = Search value
a[lo] ≤ Search value ≤ **a[hi].**



- Ex. ***Binary search for 95.***

Searching Algorithm:

Binary Search:

In the given Array A[ ], We have to search for the index i such that A[i] = Search value

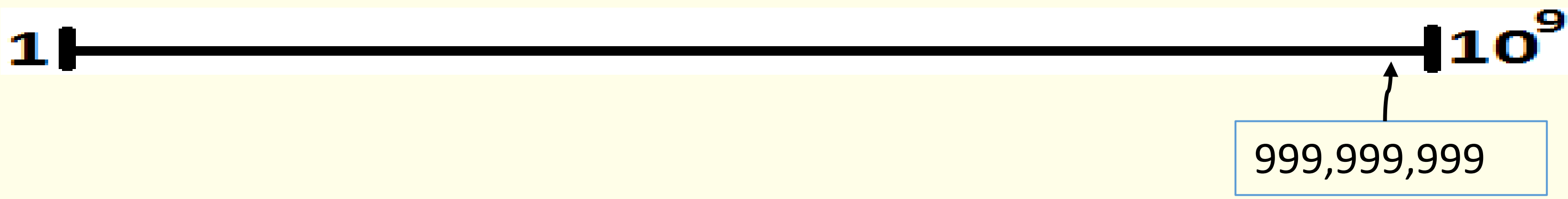
a[lo] ≤ Search value ≤ **a[hi].**

- Ex. ***Binary search for 95.***

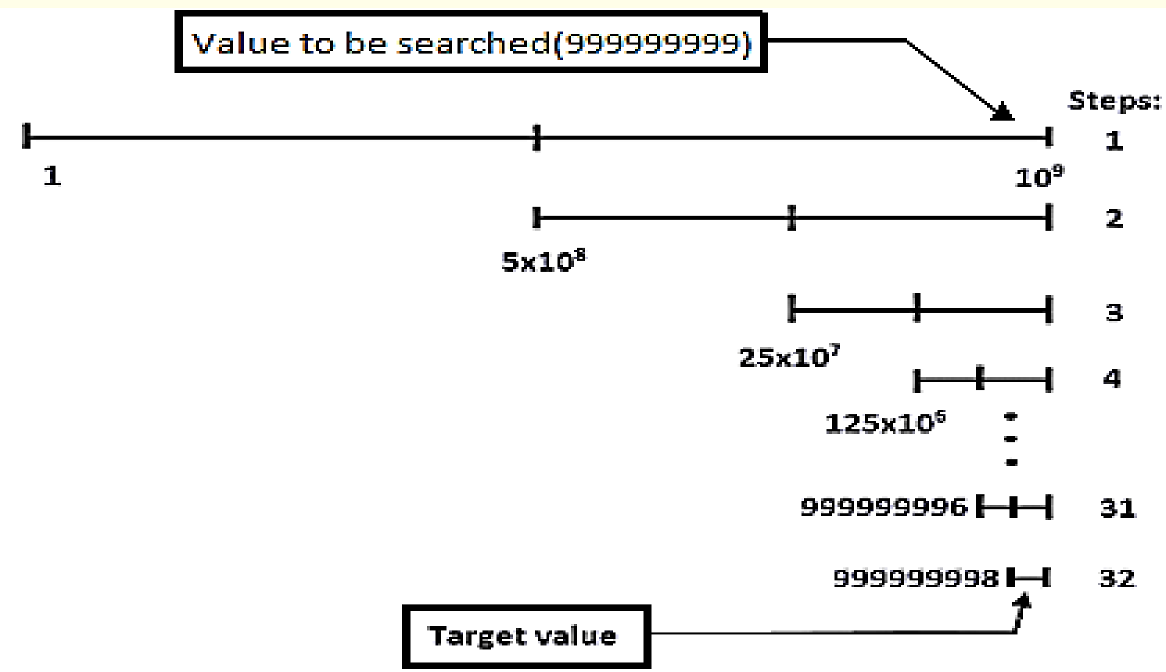
Searching Algorithms:

Comparison of search algorithms:

Sequential Search: *Number of Steps : N (total number of elements in search)*



Binary Search: ***Number of Steps : log_2N***



Sequential	Binary
999,999,999 steps	32 steps
1000 seconds	32 µs

Conclusion

Monte Carlo simulation is very time consuming, and requires huge memory storage.

With usage of

- ✓ Good Random number generating function
- ✓ Dynamic allocation of the array
- ✓ Unionized energy grid of nuclides and
- ✓ Different searching algorithms

the time and the storage space required for the Monte Carlo simulation can be optimized considerably

Acknowledgements

- Authors express sincere thanks to Dr. Kapil Deo Singh and Dr. S. Anand for the training and for the constant encouragement. And I am thankful for the guidance given by other supervisor as well as the panels especially in our project.

References

- Particle-transport simulation with the Monte Carlo method
  Carter, L.L. ; Cashwell, E.D.
- Monte Carlo N–Particle Transport Code System
  Contributed by: Los Alamos National Laboratory, Los Alamos, New Mexico

- Monte Carlo methods
  Malvin H. Kalos

- Bennett Fox,
  Algorithm 647:
  Implementation and Relative Efficiency of Quasirandom Sequence Generators,
  ACM Transactions on Mathematical Software,
  Volume 12, Number 4, December 1986, pages 362-376.

- Pierre L'Ecuyer,
  Random Number Generation,
  in Handbook of Simulation,
  edited by Jerry Banks,
  Wiley, 1998,
  ISBN: 0471134031,
  LC: T57.62.H37.

- Peter Lewis, Allen Goodman, James Miller,
  A Pseudo-Random Number Generator for the System/360,
  IBM Systems Journal,
  Volume 8, 1969, pages 136-143.

- Numerical Recipes in Fortran
  The art of scientific computing
  William H. Press, Saul A. Teukolsky
  William T. Vetterling, Brian P. Flannery

Thank You

Development of Monte Carlo code for the estimation of K_{eff}

Sachin.
JFR, MCNS.

This work carried out under BRNS Project:

Development of indigenous Monte Carlo code for estimation of k_{eff} in nuclear fuel cycle facilities.(Project Ref. No. :36(4)/14/40/2015/36007)

Outline

- Monte Carlo method
- Neutron multiplication factor of a system (k_{eff})
- Introduction and Explanation of developed code
- Solved problems and results
- Plan of further development

Monte Carlo

Probability of getting head or tail is $\frac{1}{2} = \frac{\textit{favorable events}}{\textit{total events}}$

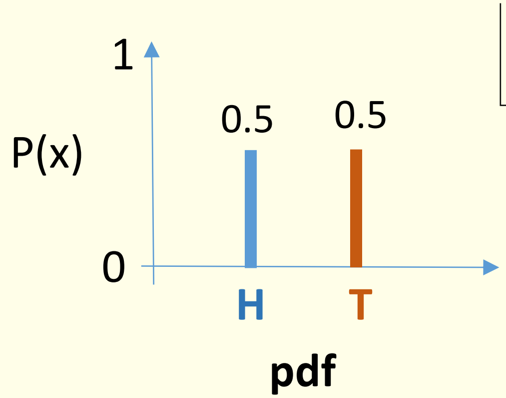
Tossing a coin:

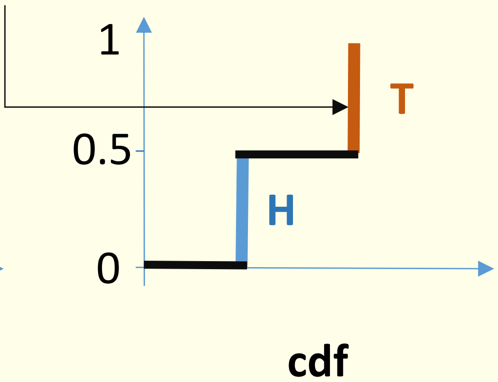
Let's toss a coin for 10 times

T H T H H T T H T H

Randomly

Random numbers (0 to 1)



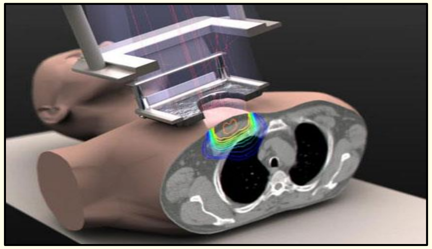


Trials	Number of heads	Expected number
10	6	5
20	11	10
50	28	25

Monte Carlo

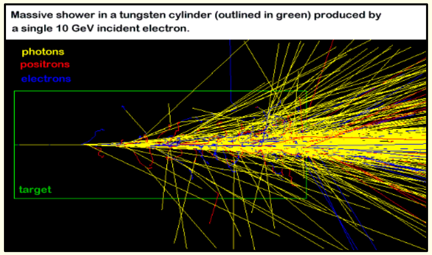
Monte Carlo methods are a *class of computational algorithms* for simulating the behavior of various physical and mathematical systems, in which observations are *randomly generated* from the model.

weather forecasting



Radiation therapy

Traffic flow

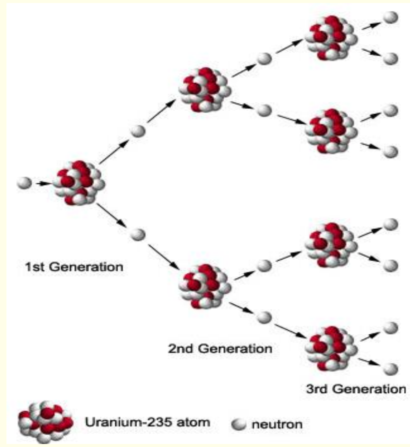


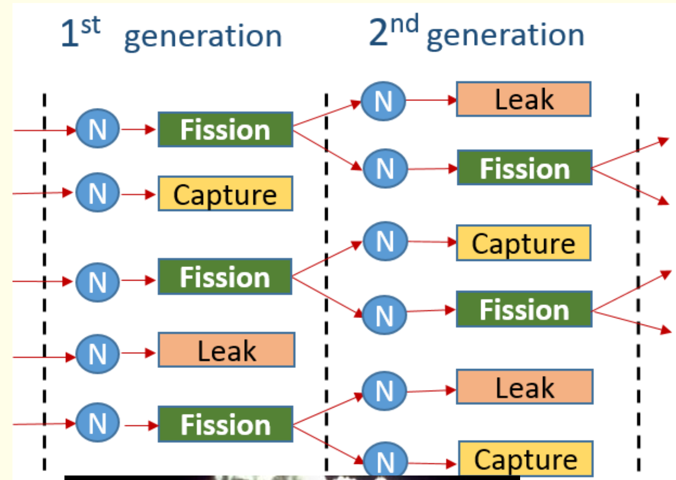
Radiation transport problems

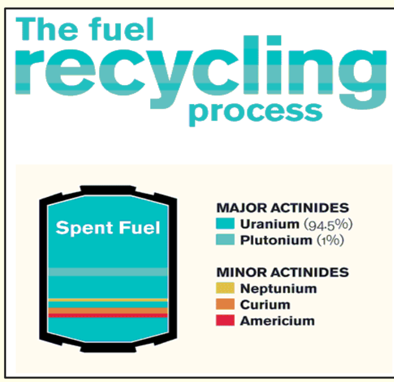
Neutron multiplication factor of a system

Neutron multiplication factor (K_{eff})

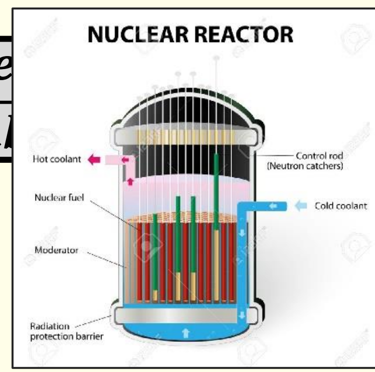
Nuclear Fission
Chain reaction







$K_{eff} < 1$



$K_{eff} = 1$

$K_{eff} > 1$

Reason for development

- **Licensed Monte Carlo Criticality codes are not available in India**
- **Not much significant organized effort for situations outside reactor geometry**
- **To ensure safety, a proper tool is required for analysis of complex fissile systems.**

Therefore, development of a Monte Carlo code indigenously , which is applicable to complex geometries is highly desirable.

Program flow

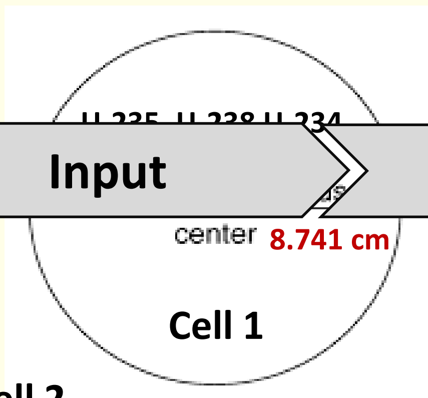
Input

1. Processing user input (Geometry and Materials)
2. Reading nuclide cross-section data

MODULE INPUT

Shows Warning
Execution stops with fatal error

Godiva Reactor (Benchmark Problem)



Cell 2

Input | **Simulate all particles** | **output**

ULE CROSS_SECTION

the *cross section data of all nuclide*

Reads the cross section of all nuclide from ace formatted file and stores them in computer memory for further usage

```
1    1    -18.74  -1    imp:n=1  $  enriched uranium sphere (godiva)
2    0             1    imp:n=0  $  all space outside the sphere
1    so    8.741                 $ radius of the Godiva sphere
m1   92235 -93.71   92238 -5.27   92234 -1.02
```

Cells

Surfaces

Materials

Program flow

Input → **Simulate all particles** → output

1. Random number Generation

2. Required Physics

3. Computational Geometry

Requirement of
huge quantity of
random Numbers
with
good quality

How far to collision ?
Which Nuclide ?
New Energy & Direction ?
Any secondary ?
Is particle alive ?

In which cell a particle is ?
How far to boundary ?
What is after the boundary?
Does particle leaking out ?

Program flow

Input | **Simulate all particles** | output

1. Random number generation

Requirement of **HUGE** quantity (millions) of random numbers

Required properties

Long period – capable of generating huge quantity of random numbers without repetition

Uniformity – should generate random numbers uniformly from 0 to 1

Time – should generate random numbers in optimum time

MODULE RANDOM

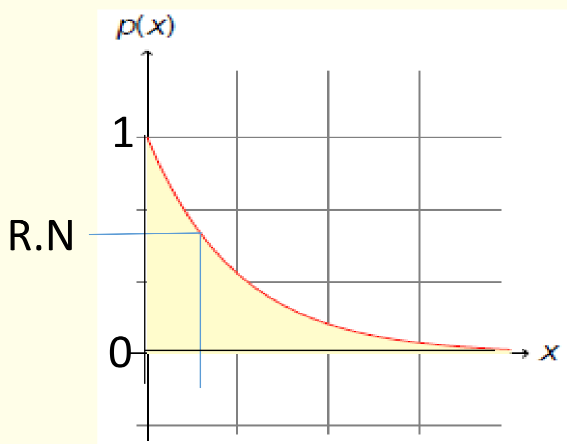
Gives random numbers

Description	Seeds	Period	Average	Time for 100 million(sec)
1. r4_random (s1, s2, s3)	1< S1, S2, S3 < 30000	$>6.9\times10^{12}$	0.4998	9.6250
2. r4_uni (s1, s2)	1< S1,S2 <2147483562	$>2.3\times10^{18}$	0.5000	7.0156
3. r8_random (s1, s2, s3)	1 < S1, S2, S3 < 30000	$>6.9\times10^{12}$	0.4998	11.3906
4. r8_uni (s1, s2)	1< S1,S2 <2147483562	$>2.3\times10^{18}$	0.5000	8.0625
5. r4_uniform_01 (seed2)	1< S1 <2147483562	NA	0.4997	5.2655
6. r8_uniform_01 (seed3)	1< S1 <2147483562	NA	0.4997	8.5156
7. ranC(idumC)	S1 = Negative integer	$>2\times10^{18}$	0.4999	28.4062
8. ranE(idumE)	S1 = Negative integer	$>2\times10^{18}$	0.5003	18.6094
9. Prn(s)	S1= positive integer	$>2.3\times10^{18}$	0.5001	5.0186

2. Required Physics

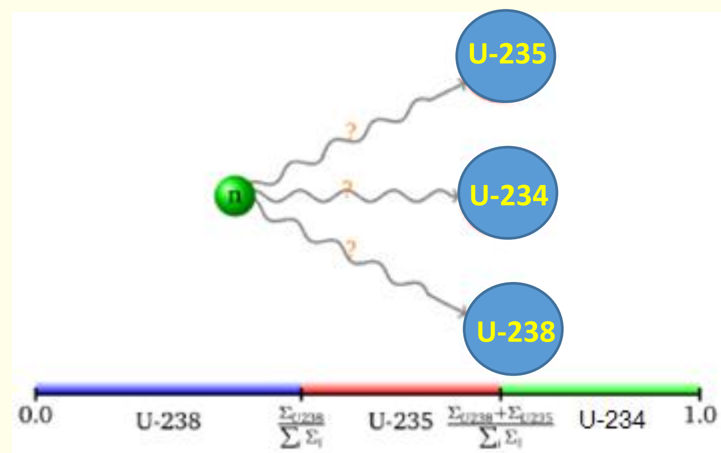
Distance to collision ?

- The probability of collision for a particle between *l* and *l* + *dl* along its line of flight is given by
- $p(x)dx = e^{-\Sigma_t l}\,\Sigma_t dx$
- $x = -\frac{1}{\Sigma_t}\ln(\xi)$



Which nuclide ?

- We compare cross-section to determine which nuclide a neutron collides with
- Normalize the Nuclide cross-sections for sampling



$$p_k = \frac{N^{(k)}\sigma_T^{(k)}}{\Sigma_T}$$

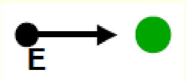
Program flow

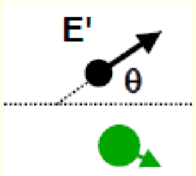
Input | **Simulate all particles** | output

2. Required Physics

- What is the particle **direction** and **energy** after the interaction ?

(if particle is alive)





- What is the **number of neutron** after the **fission** ?

(if fission event occurs)

MODULE physics

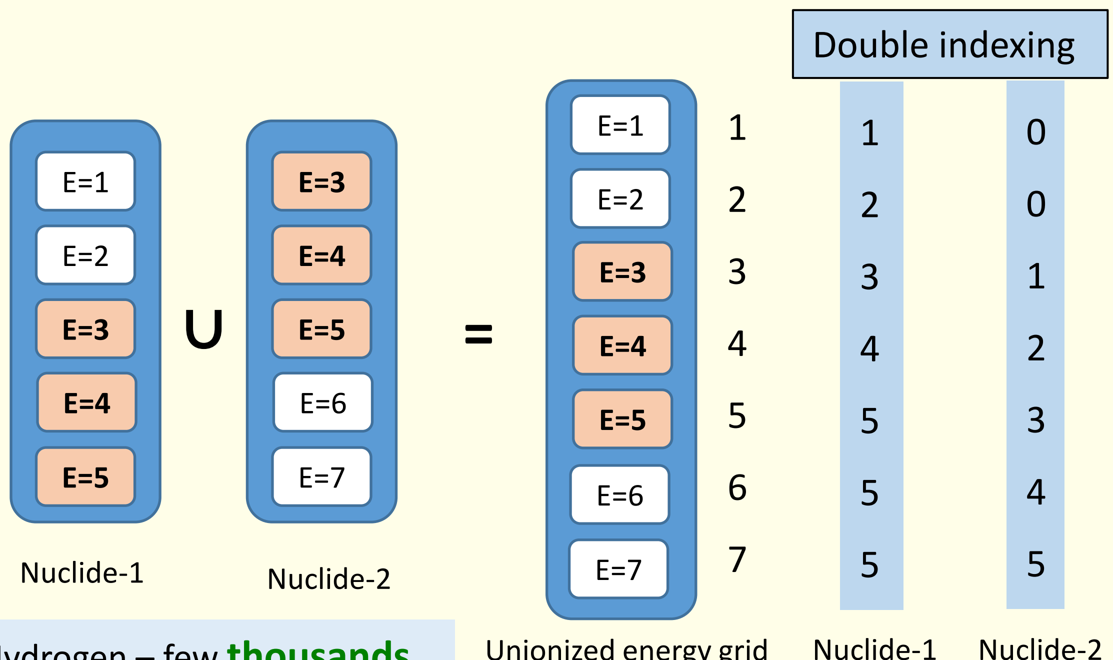
- Gives which nuclide a neutron collides with
- Gives the type of interaction
- Tells Whether the neutron is alive
- Gives the exit energy and direction
- Gives the number of neutron produced from the fission

Program flow

Input | **Simulate all particles** | output

For each collision of neutron having energy E,
it is required to search the energy and in the ACE file of that collision nuclide

Unionized energy grid method



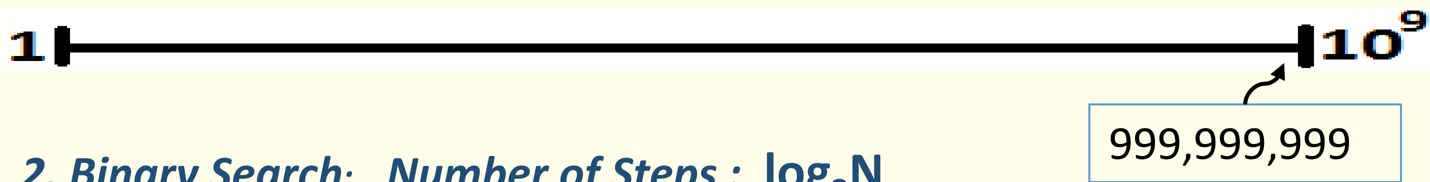
Hydrogen – few **thousands**
Uranium – few **millions**

Program flow

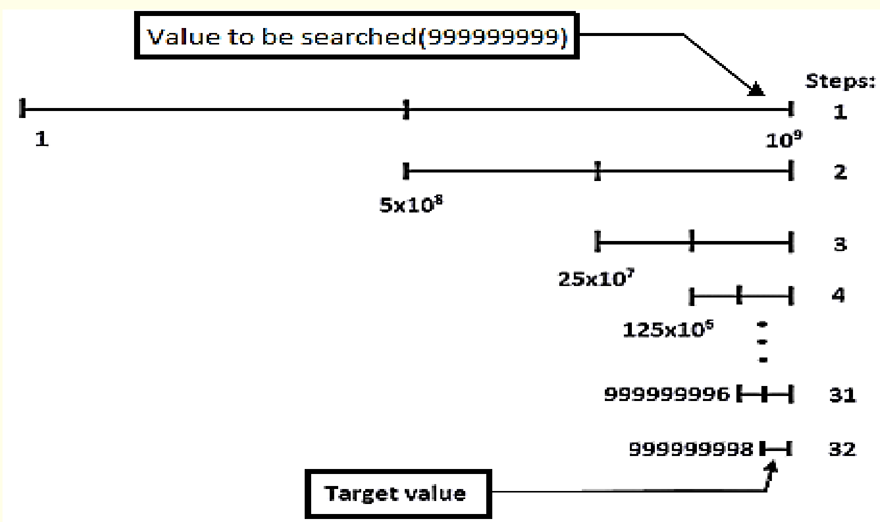
Input | **Simulate all particles** | output

Comparison of search algorithms

1. Sequential Search: ***Number of Steps : N (total number of elements in search)***



2. Binary Search: ***Number of Steps :*** $\log_2 N$



Sequential	Binary
999,999,999 steps	32 steps
1000 seconds (16 min)	32 µs

3. Computational Geometry

Surfaces

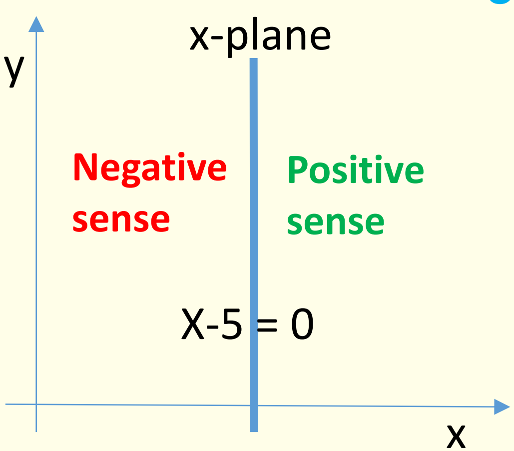
For a given point in space, (x,y,z), and surface equation, F(x’,y’,z’)=0,

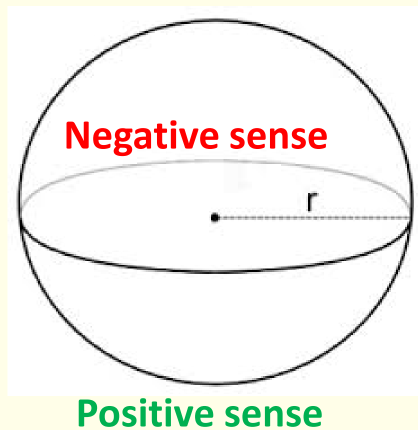
the **sense** of the point with respect to the surface is defined as:

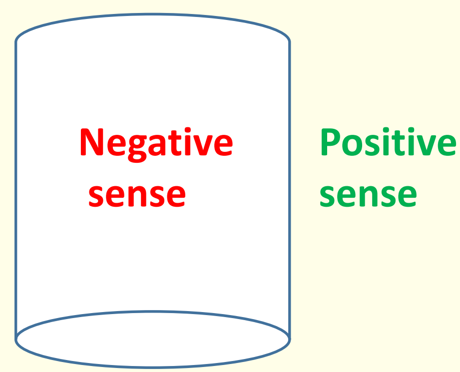
Inside the surface, sense < 0 if F(x,y,z) < 0

Outside the surface, sense > 0 if F(x,y,z) > 0

On the surface, sense = 0 if F(x,y,z) = 0







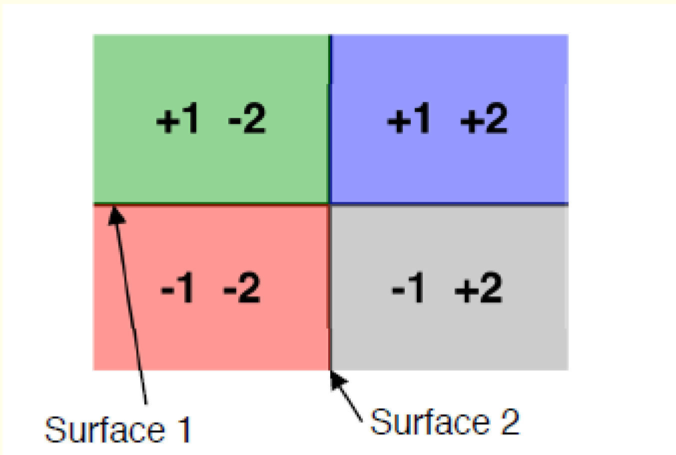
3. Computational Geometry

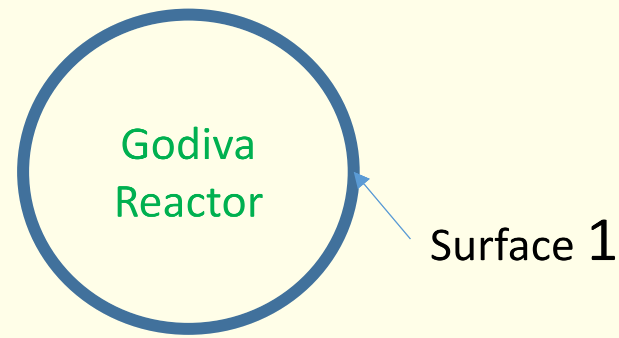
Surfaces

Mnemonic	Type	Description	Equation
P	Plane	General	$Ax + By + Cz - D = 0$
PX		Normal to X-axis	$x - D = 0$
PY		Normal to Y-axis	$y - D = 0$
PZ		Normal to Z-axis	$z - D = 0$
SO	Sphere	Centered at Origin	$x^2 + y^2 + z^2 - R^2 = 0$
S		General	$(x-\bar{x})^2 + (y-\bar{y})^2 + (z-\bar{z})^2 - R^2 = 0$
SX		Centered on X-axis	$(x-\bar{x})^2 + y^2 + z^2 - R^2 = 0$
SY		Centered on Y-axis	$x^2 + (y-\bar{y})^2 + z^2 - R^2 = 0$
SZ		Centered on Z-axis	$y^2 + y^2 + (z-\bar{z})^2 - R^2 = 0$
C/X	Cylinder	Parallel to X-axis	$(y-\bar{y})^2 + (z-\bar{z})^2 - R^2 = 0$
C/Y		Parallel to Y-axis	$(x-\bar{x})^2 + (z-\bar{z})^2 - R^2 = 0$
C/Z		Parallel to Z-axis	$(x-\bar{x})^2 + (y-\bar{y})^2 - R^2 = 0$
CX		On X-axis	$y^2 + z^2 - R^2 = 0$
CY		On Y-axis	$x^2 + z^2 - R^2 = 0$
CZ		On Z-axis	$x^2 + y^2 - R^2 = 0$

3. Computational Geometry

Create cell with intersection of spaces






```
1    1    -18.74  -1     imp:n=1  $  enriched uranium sphere (godiva)
2    0             1     imp:n=0  $  all space outside the sphere

1    so   8.741                     $ radius of the godiva sphere
```

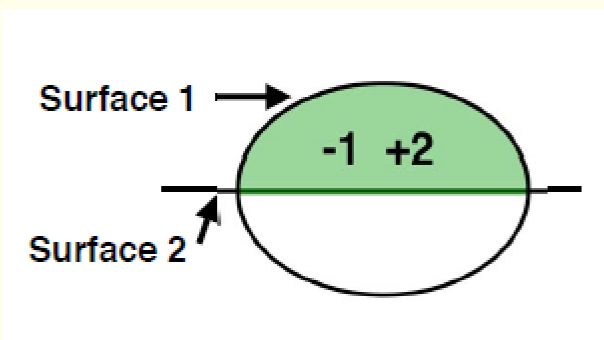
3. Computational Geometry

<u>Finding cell for a co-ordinate</u>

For all cells and each surfaces of a cell

(**substitute particle co-ordinates in surface equation**)

(*if any one surface doesn't satisfy the user defined geometry specification* – **particle is outside**)



<u>Distance to boundary</u>

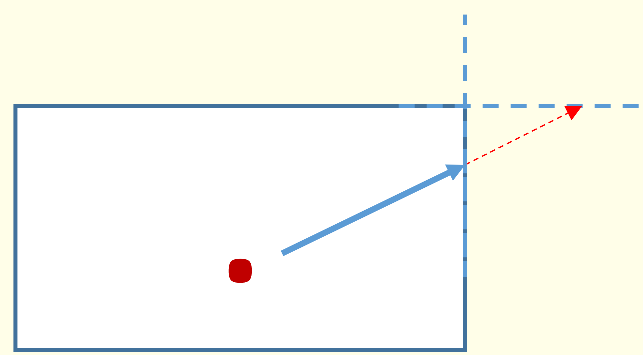
For a cell and each surfaces

Evaluate:

D_{surf} = smallest positive root of

$F_{surf}(x, y, z) = 0$

$d = \min (d, D_{surf})$



3. Computational Geometry

- Cross surface -- Handles surface crossings
- Cell contains -- to check whether particle in a cell
- Sense -- checks the sense of all type of surfaces with a given co-ordinate
- Neighbor list – which is used in surface crossing

Module geometry

Status of a particle and various other parameters of a particle in user defined geometry specification

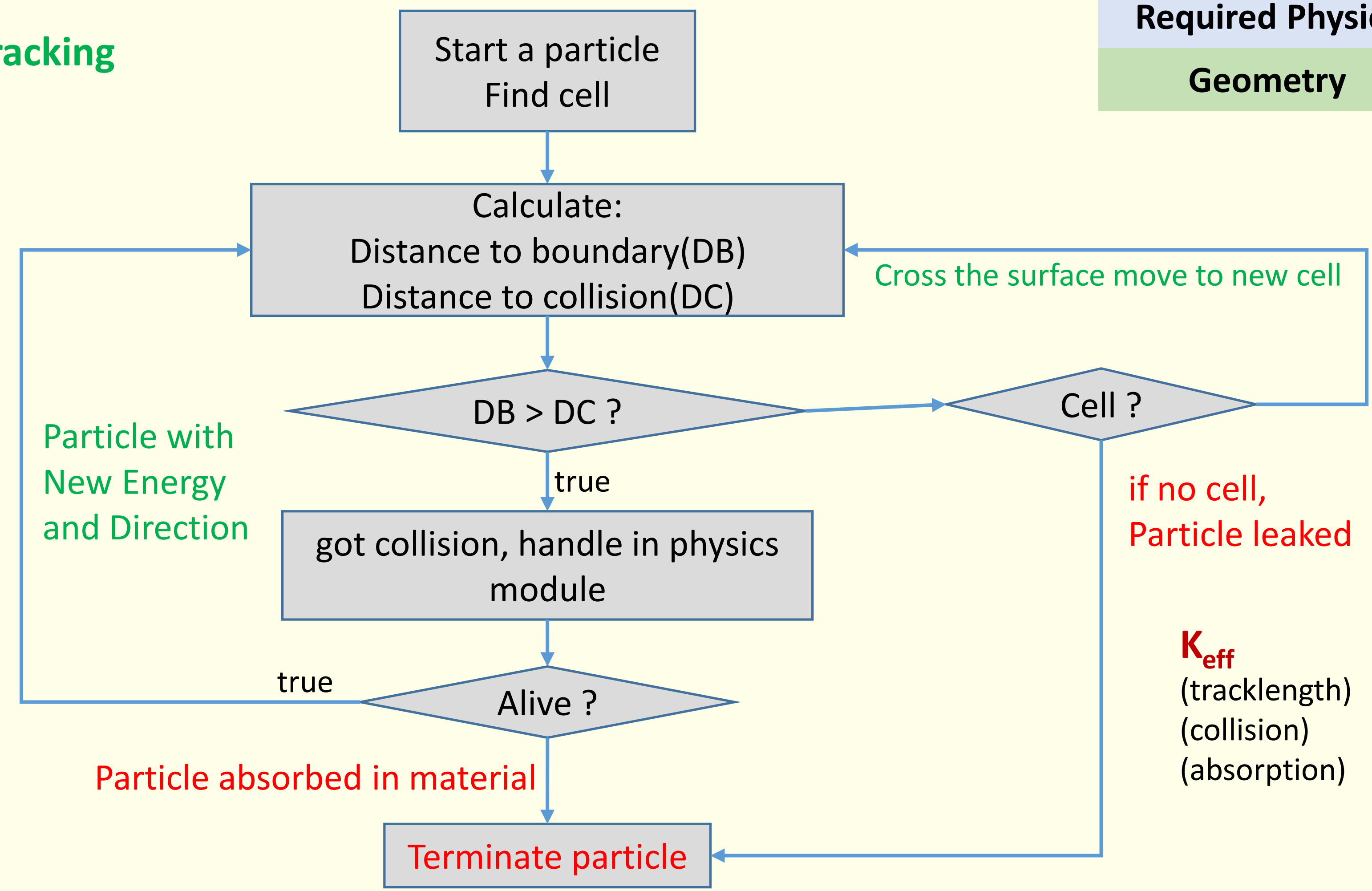
Program flow

Input | **Simulate all particles** | output

Simulate a particle (tracking)

Random numbers
Required Physics
Geometry

Module tracking



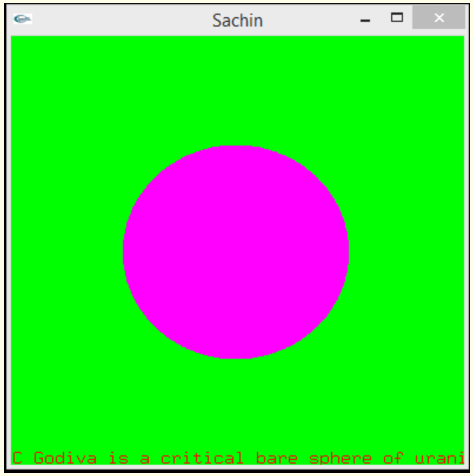
Results

1. Godiva (Benchmark problem)

```
1     -18.74  -1     imp:n=1   $ enriched uranium sphere (godiva)
2       0          1    imp:n=0  $ all space outside the sphere

1     so    8.741       $ radius of the godiva sphere

m1    92235.72c -93.71  92238.72c -5.27  92234.72c -1.02
```



codes	Keff
Benchmark Keff	1.000 ± 0.0010
MCNP	0.9995 ± 0.0005
This code	1.0070 ± 0.0006

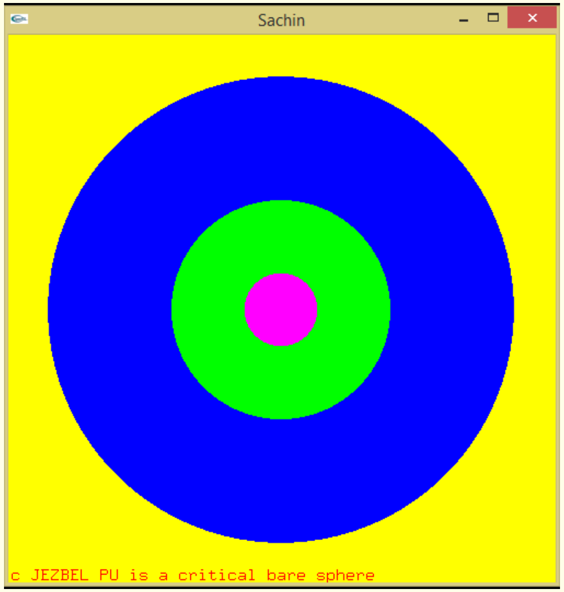
Results

2.Jezbel (Benchmark problem)

```
1   1  .038918  -1       imp:n=1
2   1  .038918  1 -2     imp:n=1
3   1  .038918  2 -3     imp:n=1
4      0          3        imp:n=0

1  so  1.0
2  so  3.0
3      so  6.385

m1 94239.72c  0.03705  94240.72c 0.001751
       94241.72c 0.000117
m2 94239.72c 1.0
m3 94240.72c 1.0
m4 94241.72c 1.0
```



code	Keff
Benchmark Keff	1.000 ±0.0010
MCNP	0.99902 ±0.00057
This code	0.99691±0.00062

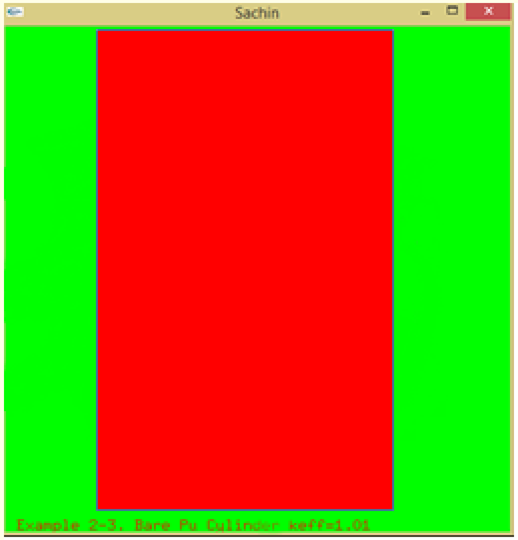
Results

3.Bare Pu-239 cylinder (Benchmark problem)

```
1   1   -15.8   -1  2  -3  imp:n=1
2      0    1:-2:3      imp:n=0

1   cz   4.935
2   Pz   0
3   pz   17.273

m1 94239.66c 1.0
```



code	Keff
Benchmark Keff	≈ 1.01
MCNP	1.01403 ±0.00066
This code	1.023241± 0.0007

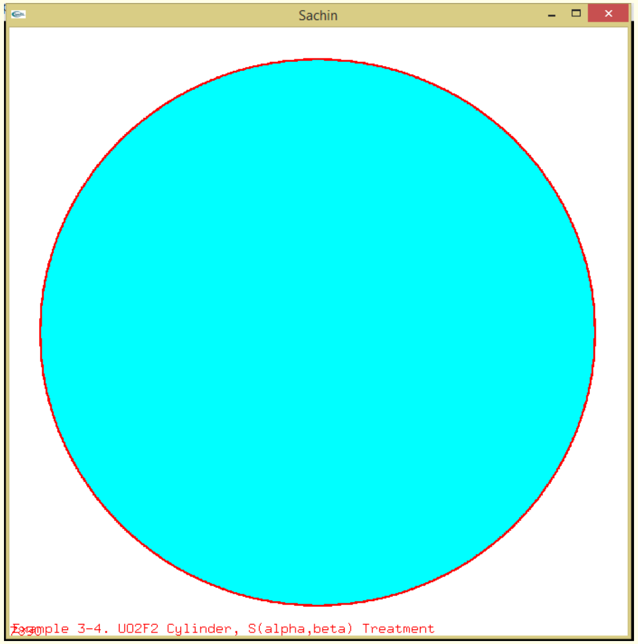
Results

4. UO2F2 Cylinder (Benchmark problem)

```
1   1   9.6586E-2   -1 3 -4   imp:n=1
2   0  -1  4 -5   imp:n=1
3   2  -2.7   -2 -3 6   imp:n=1
4   2  -2.7    1 -2 -5 3   imp:n=1
5   0    2:5:-6    imp:n=0

1 cz 20.12
2 cz 20.2787
3 pz 0.0
4 pz 100.0
5 pz 110.0
6 pz -0.1587

m1 1001.62c 5.7058e-2 8016.62c 3.2929e-2"
      9019.62c 4.3996e-3 92238.66c 2.0909e-3"
      92235.66c 1.0889e-4"
mt1  hh2o.71t  "
m2 13027.62c 1
```



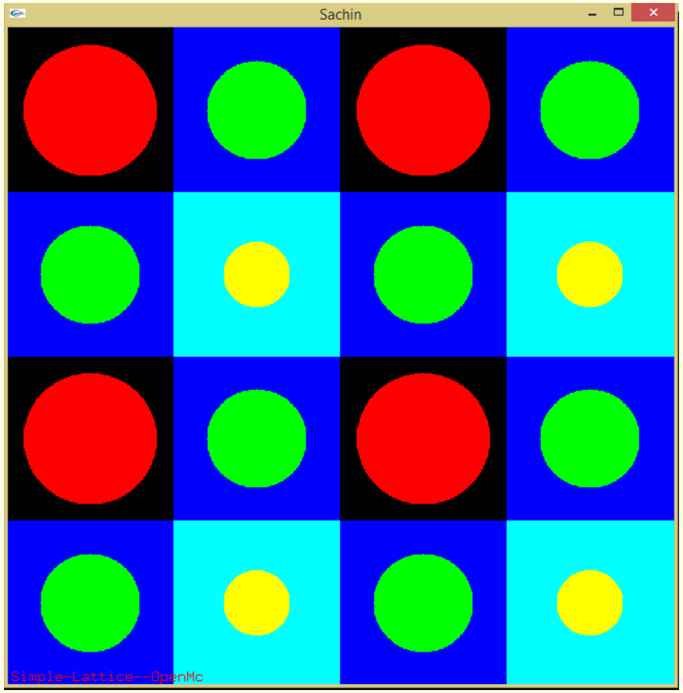
code	Keff
Benchmark Keff	≈ 1.000
MCNP	0.99842± 0.00077
This code	0.996797 ± 0.0008

Results

5. Simple lattice

```
1       0    fill=5   1 -2 3 -4  imp:n=1
101   1  -4.5 -5 u=1 imp:n=1
102   2  -1.0  5 u=1 imp:n=1
201   1  -4.5 -6 u=2 imp:n=1
202   2  -1.0  6 u=2 imp:n=1
301   1  -4.5 -7 u=3 imp:n=1
302   2  -1.0  7 u=3 imp:n=1
3      0   -1:2:-3:4   imp:n=0
2       0  1 -2 3 -4  imp:n=1 lat=1 u=5 fill=0:3 0:3 0:0
                    1 2 1 2
                    2 3 2 3
                    1 2 1 2
                    2 3 2 3
1 px -2.0
2 px  2.0
3 py -2.0
4 py  2.0
5 cz  0.4
6 cz  0.3
7 cz  0.2

m1  92235.72c 1.0
m2   1001.72c 2.0 8016.72c 1.0
mt2  hh2o.71t
```



code	Keff
OpenMC	0.05983 ± 0.0001
This code	0.072534 ± 0.0002

To do

- Incorporate input feature (include other surface like cone ellipsoid)
- lattice – hexagonal
- nested lattice
- complex cell (union and complement operators)
- write code to for parallel mode
- much more

References

- Numerical Recipes in Fortran The art of scientific computing
  William H. Press, Saul A. Teukolsky William T. Vetterling, Brian P. Flannery
- Monte Carlo methods, Malvin H. Kalos
- Monte Carlo N–Particle Transport Code System Contributed by: Los Alamos National Laboratory, Los Alamos, New Mexico
- Roger Brewer, "Criticality Calculations with MCNP5: A Primer", Los Alamos National Laboratory,LA-UR-09-00380, January 2009
- A Monte Carlo Primer, Stephen A Dupree and Stanley K. Fraley
- http:// www.stackoverflow.com http://gcc.gnu.org http://github.com

Conference papers

1. "***Advances in Monte Carlo methods in Radiation Transport***" Sachin Shet, K.V.Subbaiah
   Indian Association for Radiation Protection International Conference – 2018,
2. "***Monte Carlo simulation: Optimization of computer time and memory***", Sachin Shet, K.V.Subbaiah
   Manipal Research Colloquium – 2017
3. **"Comparison of measured and calculated dose rates of Am-Be source with Monte Carlo simulation."**
   Sachin Shet, K V Subbaiah, 21st National Symposium on Radiation Physics-2018

Thank you . . .

cxs
normalize_ao
logarithmic_grid
calculate_xs
calculate_nuclide_xs
calculate_urr_xs
read_xs
read_ace_table
read_esz
read_nu_data
read_reactions
read_angular_dist
read_energy_dist
get_energy_dist
length_energy_dist
read_unr_res
generate_nu_fission
read_thermal_data
get_int
get_real
is_fission
is_disappearance
is_scatter
reaction_name
calculate_sab_xs

declarations
contains all type and class declarations required

common
write_fatalerror
write_warning
write_message
int_read_write
binary_search
int_to_str
interpolate_tab1
real_to_str
free_memory

Input
split_string
Tokenize
to_lower
to_upper
is_number
string_contains
generate_rpn
already_contains
neighbor_list
input_check
id_index
input_print
input_read
input_block_process
input_samples
input_file_read

geometry
cell_contains
simple_cell_contains
complex_cell_contains
find_cell
cross_surface
cross_lattice
distance_to_boundary
Sense
handle_lost_particle

Eigenvalue
[illegible]
[illegible]
initialize_batch
initialize_generation
finalize_generation
finalize_batch
synchronize_bank
reset_result

Graphics
Plot
position_rgb
Plotter
Display

Graphics
C-binding modules
Opengl_kinds
Opengl_gl
Opengl_glu
Opengl_glut

Particle_header
deallocate_coord
initialize_particle
clear_particle

[illegible]cg
Prn
initialize_prng
set_particle_seed

Source
initialize_source
sample_external_source
get_source_particle
copy_source_attributes

Tracking
transport

Physics
Collision
sample_reaction
sample_nuclide
sample_fission
Absorption
Scatter
elastic_scatter
sab_scatter
sample_target_velocity
sample_cxs_target_velocity
create_fission_sites
sample_fission_energy
inelastic_scatter
sample_angle
rotate_angle
sample_energy
maxwell_spectrum
watt_spectrum
nu_total
nu_prompt
nu_delayed

constants
Contains all constant parameter required for the code

13 k lines

cxs
normalize_ao
logarithmic_grid
calculate_xs
calculate_nuclide_xs
calculate_urr_xs
read_xs
read_ace_table
read_esz
read_nu_data
read_reactions
read_angular_dist
read_energy_dist
get_energy_dist
length_energy_dist
read_unr_res
generate_nu_fission
read_thermal_data
get_int
get_real
is_fission
is_disappearance
is_scatter
reaction_name
calculate_sab_xs

declarations
contains all type and class declarations required

common
write_fatalerror
write_warning
write_message
int_read_write
binary_search
int_to_str
interpolate_tab1
real_to_str
free_memory

Input
split_string
Tokenize
to_lower
to_upper
is_number
string_contains
generate_rpn
already_contains
neighbor_list
input_check
id_index
input_print
input_read
input_block_process
input_samples
input_file_read

geometry
cell_contains
simple_cell_contains
complex_cell_contains
find_cell
cross_surface
cross_lattice
distance_to_boundary
Sense
handle_lost_particle

Eigenvalue
run_eigenvalue
output_print
initialize_batch
initialize_generation
finalize_generation
finalize_batch
synchronize_bank
reset_result

Graphics
Plot
position_rgb
Plotter
Display

Graphics C-binding modules
Opengl_kinds
Opengl_gl
Opengl_glu
Opengl_glut

Particle_header
deallocate_coord
initialize_particle
clear_particle

Random_lcg
Prn
initialize_prng
set_particle_seed

Source
initialize_source
sample_external_source
get_source_particle
copy_source_attributes

Tracking
transport

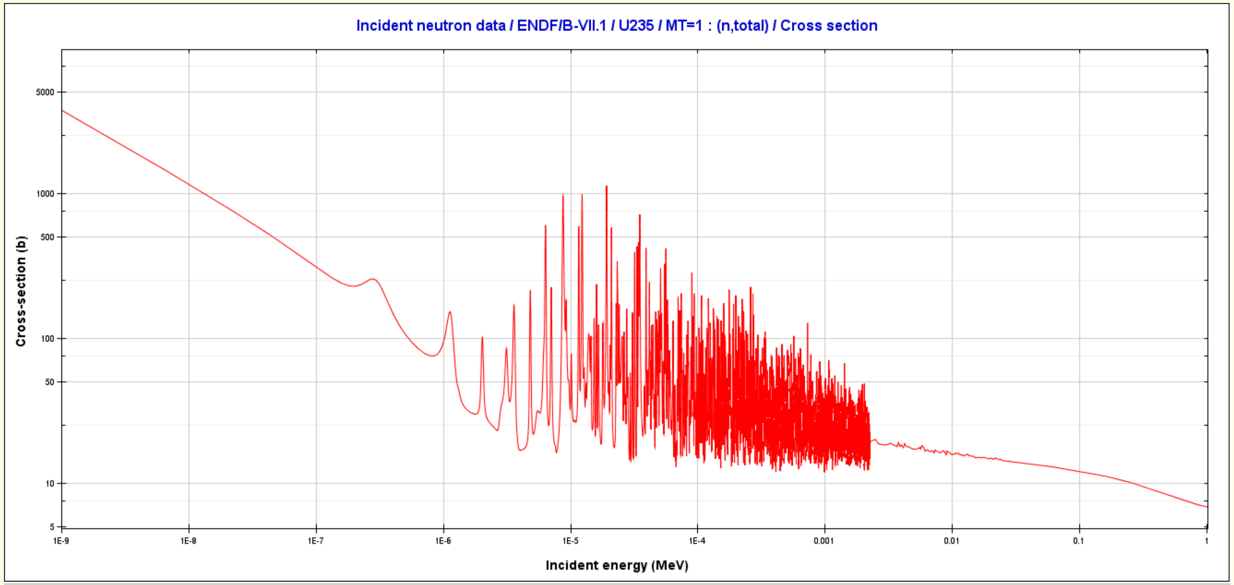
Physics
Collision
sample_reaction
sample_nuclide
sample_fission
Absorption
Scatter
elastic_scatter
sab_scatter
sample_target_velocity
sample_cxs_target_velocity
create_fission_sites
sample_fission_energy
inelastic_scatter
sample_angle
rotate_angle
sample_energy
maxwell_spectrum
watt_spectrum
nu_total
nu_prompt
nu_delayed

constants
Contains all constant parameter required for the code

Thank you for your attention . . .

2.cross-section data visualization

- By using the NJOY code the evaluated neutron cross section data of many nuclides are processed from ENDF-6 format to ACE format.



Plotted using **JANIS-4.0** (Java-based Nuclear Data Information System)

MODULE cxs

Reads all nuclide cross-section data and store it in system memory

Tools used for the code

- Compiler – gfortran windows binaries 64 bit
  Mingw-gfortran 32 bit
- Debugger -- gdb (by Gnu)
- IDE(Integrated-development env) – codeblocks
- Written with fortran 2003 features
- Graphics – open graphics library (**C** libraries)

Monte Carlo

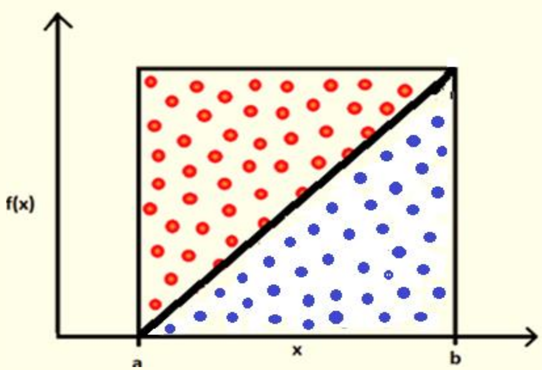
Estimation of a finite Integral: Deterministic Method

- Deterministic system : In which the later states of the system are determined by the earlier ones.
- This method enables artificial construction of a probabilistic model.

f(x) = x

$$\int_0^1 \mathbf{x}\,\mathbf{dx} = \mathbf{0.5}$$

$$\int_a^b f(x)\,dx \approx \max(f(x))\,(b-a)\,\frac{\textit{fraction of points under the line}}{\textit{Total number of points} \quad +}$$



Number of trials	Points enclosed by the curve	Value of the integral
10	6	0.600
100	56	0.56
1000	497	0.497
10000	5001	0.5001

Keff estimators

- Pathlength estimator for Keff

$$K_{path} = \left(\sum_{\substack{\text{all} \\ \text{flights}}} wgt_j \cdot d_j \cdot \nu\Sigma_F \right) \Big/ W$$

W = total weight starting each cycle

- Collision estimator for Keff

$$K_{collision} = \left(\sum_{\substack{\text{all} \\ \text{collisions}}} wgt_j \cdot \frac{\nu\Sigma_F}{\Sigma_T} \right) \Big/ W$$

- Absorption estimator for Keff

$$K_{absorption} = \left(\sum_{\substack{\text{all} \\ \text{absorptions}}} wgt_j \cdot \frac{\nu\Sigma_F}{\Sigma_A} \right) \Big/ W$$

Program flow

Input | Simulate all particles | output

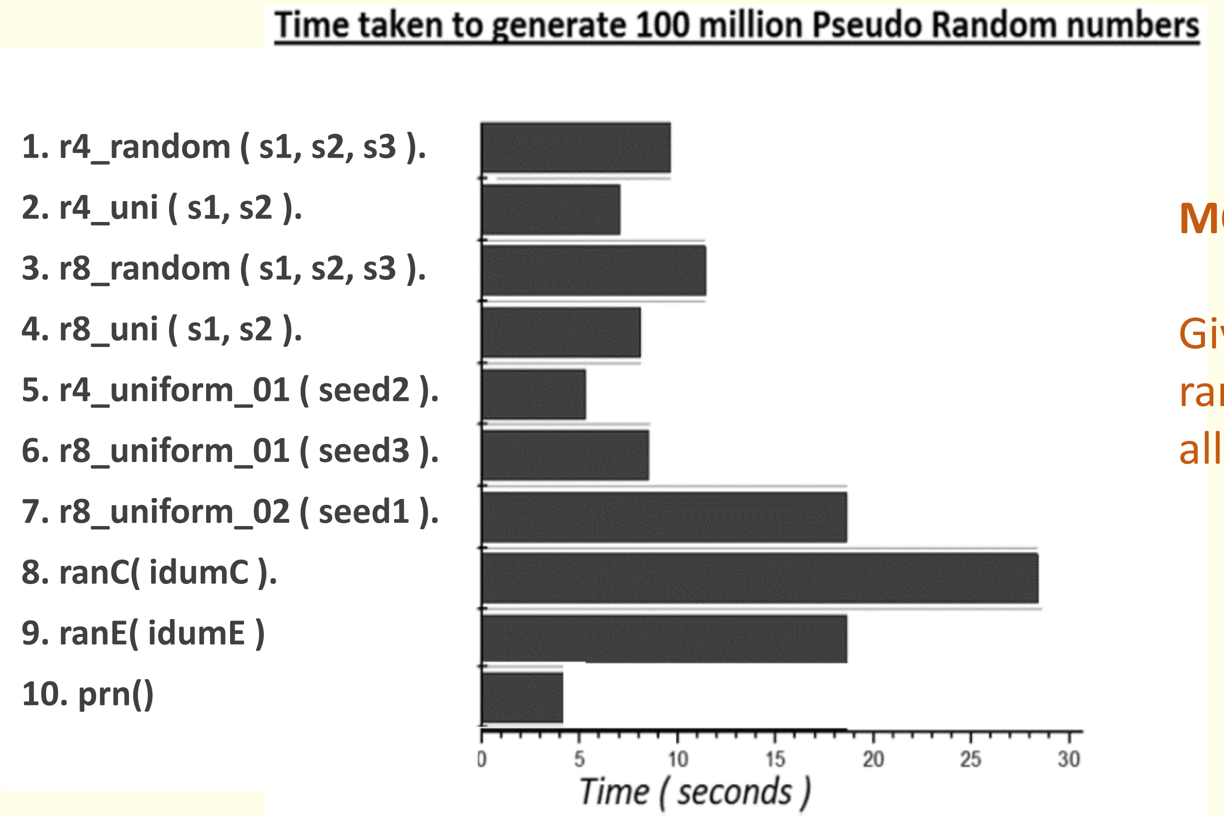
1. Random number generation

Description	Seeds	Period	Average
1. r4_random (s1, s2, s3). Linear Congruential 32 bit generator.[1]	No of Seeds: 3 1< S1, S2, S3 < 30000	6.9×10^{12}	0.4998
2. r4_uni (s1, s2). Linear Congruential 32 bit generator.[1]	No of Seeds: 2 1< S1,S2 <2147483562	2.3×10^{18}	0.5000
3. r8_random (s1, s2, s3). Linear Congruential 64 bit generator.[1]	No of Seeds: 3 1 < S1, S2, S3 < 30000	6.9×10^{12}	0.4998
4. r8_uni (s1, s2). Linear Congruential 64 bit generator.[1]	No of Seeds:2 1< S1,S2 <2147483562	2.3×10^{18}	0.5000
5. r4_uniform_01 (seed2). Multiplicative pseudo random number generator.[2] [3] [4] [5] 32 bit generator	No of Seeds: 1 1< S1 <2147483562	NA	0.4997
6. r8_uniform_01 (seed3). Multiplicative pseudo random number generator.[2] [3] [4] [5] 64 bit generator	No of Seeds: 1 1< S1 <2147483562	NA	0.4997
7. r8_uniform_02 (seed1). Multiplicative pseudo random number generator.[2] [3] [4] [5] 64 bit generator	No of Seeds:1 1< S1 <2147483562	NA	0.4997
8. ranC(idumC). Multiplicative pseudo random number generator.[6] It is a 32 bit generator, and it uses intrinsic MIN MAX functions.	No of Seeds:1 S1 = Negative integer	$>2 \times 10^{18}$	0.4999
9. ranE(idumE) Multiplicative pseudo random number generator. [6] Uses absolute value function and is a 32 bit generator.	No of Seeds:1 S1 = Negative integer	$>2 \times 10^{18}$	0.5003
10. Prn(seed) Linear congruential generator with the use of bit masking	No of Seeds:1 S1 = positive integer	$>6.9 \times 10^{12}$	0.5002

Program flow

Input | Simulate all particles | output

1. Random number generation

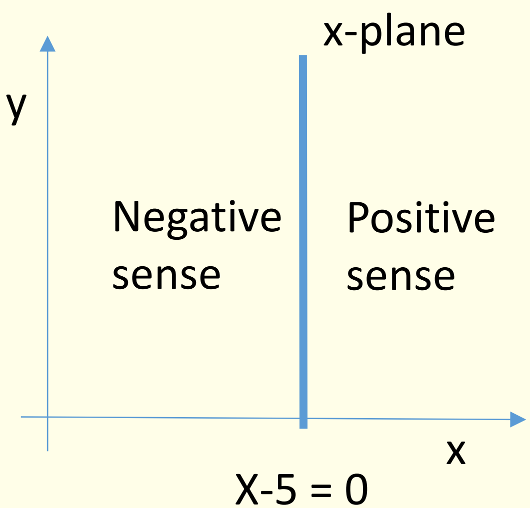


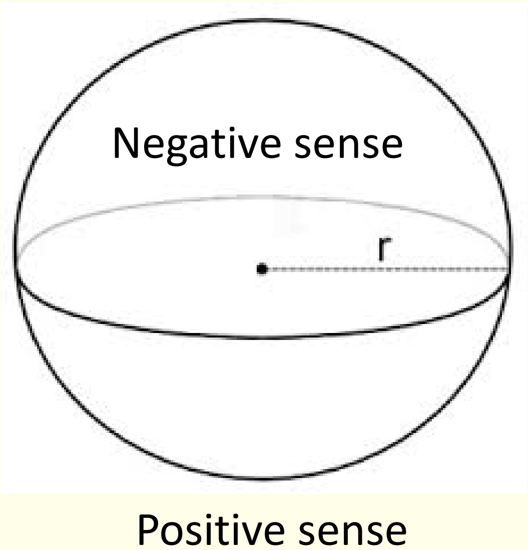
MODULE random_lcg

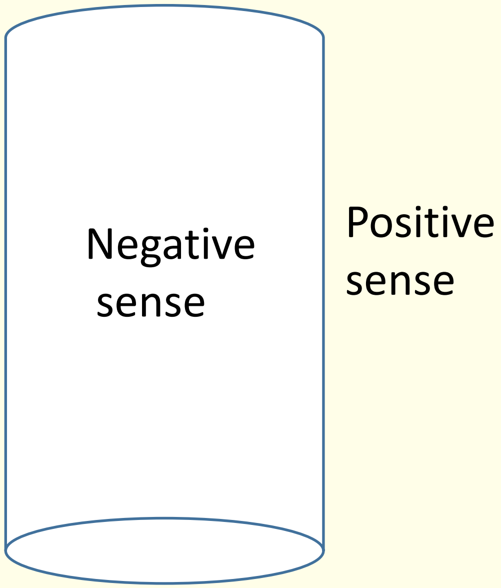
Gives huge quantity of random numbers with all required properties

3. Computational Geometry

Surfaces



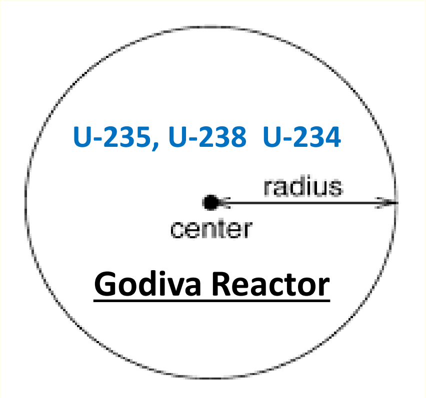





```
1     so     8.741                    $ radius of the godiva sphere
```

Program flow

Input | Simulate all particles | output

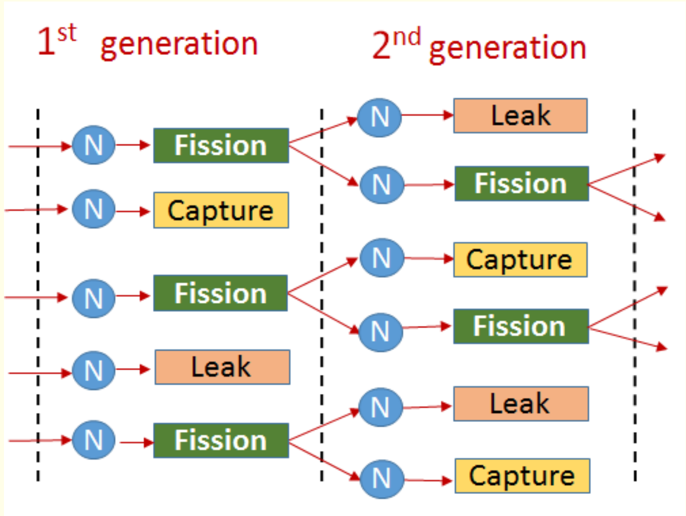


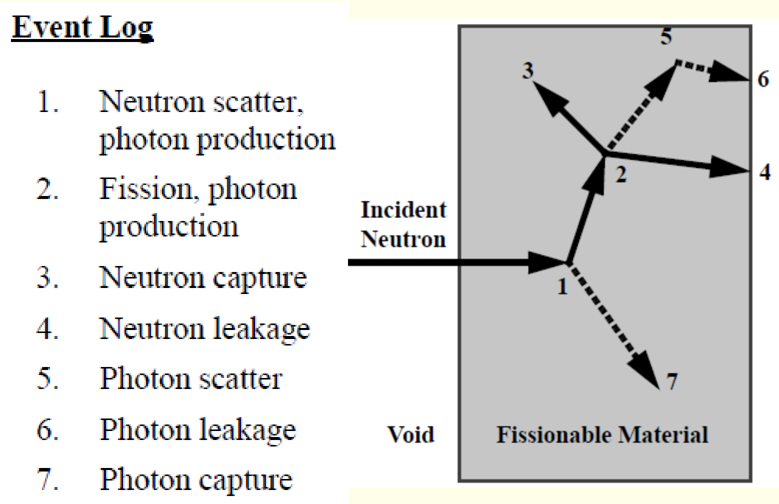
Available in **endf-6** format by

ENDF, JEFF and JENDL etc

NJOY code [available]

Getting K_{eff} with Monte Carlo method





Over all batches (generations)

Over all particles

Simulate(transport) the behavior of each particle in **user defined geometry** and **material specification.** **(Monte Carlo)**