KAPL-4823
UC-32
(DOE/TIC-4500-R75)

Monte Carlo Fundamentals

F. B. BROWN and T. M. SUTTON

February 1996

Prepared by
Lockheed Martin Company
KNOLLS ATOMIC POWER LABORATORY
Schenectady, New York

Contract No. DE-AC12-76-SN-00052

Disclaimer

CONTENTS

	Abstract	iv
1.	Introduction	1
2.	Monte Carlo and Transport Equation	7
3.	Monte Carlo and Simulation	11
4.	Random Number Generation	13
5.	Random Sampling	24
6.	Computational Geometry	43
7.	Collision Physics	56
8.	Tallies and Statistical Edits	64
9.	Monte Carlo Eigenvalue Calculations	70
10.	Source Normalization Bias	76
11.	Variance Reduction	79
12.	Multiplied Fixed-Source Calculations	82
13.	Vector and Parallel Monte Carlo	83
14.	Overview of RACER System	104
	References	109

Abstract

This report is composed of the lecture notes from the first half of a 32-hour graduate-level course on Monte Carlo methods offered at KAPL. These notes, prepared by two of the principle developers of KAPL's RACER Monte Carlo code, cover the fundamental theory, concepts, and practices for Monte Carlo analysis methods. In particular, a thorough grounding in the basic fundamentals of Monte Carlo methods is presented, including random number generation, random sampling, the Monte Carlo approach to solving transport problems, computational geometry, collision physics, tallies, and eigenvalue calculations. Furthermore, modern computational algorithms for vector and parallel approaches to Monte Carlo calculations are covered in detail, including fundamental parallel and vector concepts, the event-based algorithm, master/slave schemes, parallel scaling laws, and portability issues.

Introduction

The Monte Carlo method has been used for over 40 years to solve radiation transport problems in high energy physics, nuclear reactor analysis, radiation shielding, medical imaging, oil well-logging, etc. Individual particle histories are simulated using random numbers, highly accurate representations of particle interaction probabilities, and exact models of 3D problem geometry. Monte Carlo methods are sometimes the only viable methods for analyzing complex, demanding particle transport problems.

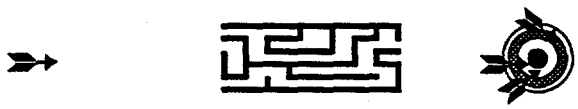
The principal limitation on the Monte Carlo method is computer power. To achieve results with acceptably low statistical uncertainty, it is often necessary to simulate millions of particle histories, consuming many hours or days of supercomputer time. Monte Carlo methods have been successfully adapted to nearly all advanced computer architectures, including vector and parallel computers. Vector Monte Carlo codes were first developed in the early 1980's using an "event-based" algorithm. More recently, parallel Monte Carlo codes have been successful using a "master/slave" approach.

This report is composed of the lecture notes from the first half of a 32-hour graduate-level course on Monte Carlo methods offered at KAPL. The purpose of the course is to provide thorough user education in Monte Carlo methods so that engineers can make more effective use of KAPL supercomputers. These notes, prepared by two of the principle developers of KAPL's RACER Monte Carlo code, cover the fundamental theory, concepts, and practices for Monte Carlo analysis methods. In particular, a thorough grounding in the basic fundamentals of Monte Carlo methods is presented, including random number generation, random sampling, the Monte Carlo approach to solving transport problems, computational geometry, collision physics, tallies, and eigenvalue calculations. While this material is available in standard references, a concise and coherent overview is provided. Furthermore, modern computational algorithms for vector and parallel approaches to Monte Carlo calculations are covered in detail, including fundamental parallel and vector concepts, the event-based algorithm, master/slave schemes, parallel scaling laws, and portability issues.

In addition to the lecture notes, a substantial number of references are provided. Included are general references on Monte Carlo methods for particle transport problems, random number generation and random sampling, and Monte Carlo methods for the solution of reactor eigenvalue problems; as well as open-literature publications on the RACER Monte Carlo code, vector and parallel Monte Carlo, and related Monte Carlo methods

Monte Carlo

Fundamentals



Forrest B. Brown

&

Thomas M. Sutton

Topics

- Introduction
- Monte Carlo & Transport Equation
- Monte Carlo & Simulation
- Random Number Generation
- Random Sampling
- Computational Geometry
- Collision Physics
- Tallies & Statistical Edits
- Monte Carlo Eigenvalue Calculations
- Source Normalization Bias
- Variance Reduction
- Multiplied Fixed-Source Calculations
- Vector & Parallel Monte Carlo
- Overview of RACER System

- **John Von Neumann invented scientific computing in the 1940's**
  - stored programs, "software"
  - algorithms & flowcharts
  - assisted with hardware design as well
  - "ordinary" computers are called "Von Neumann machines"
- **John Von Neumann invented Monte Carlo particle transport in the 1940's**
  - Highly accurate — no essential approximations
  - Expensive — typically "method of last resort"
  - Monte Carlo codes for particle transport have been proven to work effectively on all computers

    SIMD, MIMD, vector, parallel, supercomputers, workstations, clusters of workstations, ...

- **Two basic ways to approach the use of Monte Carlo methods for solving the transport equation:**
  - **mathematical technique for numerical integration**
  - **computer simulation of a physical process**

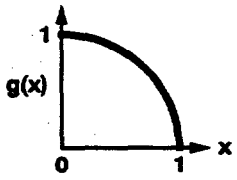
  → Each is "correct"
  - mathematical approach useful for:

    importance sampling, convergence, variance reduction, random sampling techniques, ...
  - simulation approach useful for:

    collision physics, tracking, tallying, ...
- **For Monte Carlo approach, consider the integral form of the Boltzmann equation.**
- **Most theory on Monte Carlo deals with fixed-source problems. Eigenvalue problems are needed for reactor physics calculations**

Simple Monte Carlo Example

Evaluate $G = \int_0^1 g(x)\,dx$, **with** $g(x) = \sqrt{1-x^2}$



- **Mathematical approach:**

  For k = 1, ..., N: choose $\hat{x}_k$ randomly in (0,1)

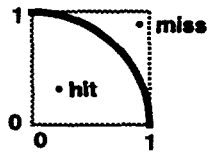
  $G = (1\text{-}0) \cdot$ [average value of g(x)] $\sim \frac{1}{N}\sum_{k=1}^{N} g(\hat{x}_k) = \frac{1}{N}\sum_{k=1}^{N}\sqrt{1-\hat{x}_k^2}$

- **Simulation approach:**

  "darts game"

  For k = 1, ..., N: choose $\hat{x}_k$ and $\hat{y}_k$ randomly in (0,1)

  If $\hat{x}_k^2 + \hat{y}_k^2 \le 1$, tally a "hit"



  G = [area under curve] $\sim (1 \cdot 1) \cdot \frac{\text{number of hits}}{N}$

Monte Carlo is often the method-of-choice for applications with integration over many dimensions

examples: high-energy physics, particle transport

Evaluate $G = \int_{a_1}^{b_1}\int_{a_2}^{b_2}\cdots\int_{a_M}^{b_M} g(r_1, r_2, \ldots, r_M)\,dr_1 dr_2 \ldots dr_M$

where $r_1, r_2, \ldots, r_M$ are all independent variables

For k = 1, ..., N:

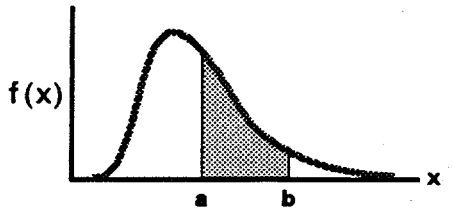
For m = 1, ..., M: choose $R_m^{(k)}$ randomly in (a_m, b_m)

$$G \sim (b_1\text{-}a_1)\cdot\ldots\cdot(b_M\text{-}a_M) \cdot \frac{1}{N}\sum_{k=1}^{N} g\left(R_1^{(k)}, R_2^{(k)}, \ldots, R_M^{(k)}\right)$$

Continuous Probability Density

- f(x)
- $0 \le f(x)$
- Probability$\{ a \le x \le b \} = \int_a^b f(x)\,dx$
- Normalization:

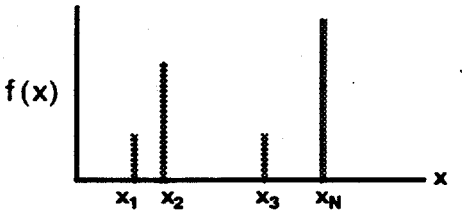
$$\int_{-\infty}^{+\infty} f(x)\,dx = 1$$



Discrete Probability Density

- $\{ f_k \}$, k=1,...,N, where $f_k = f(x_k)$
- $0 \le f_k$
- Probability$\{ x=x_k \} = f_k$
- Normalization:

$$\sum_{k=1}^{N} f_k = 1$$



- **Mean, Average, Expected Value**

$$\bar{x} = \mu = \langle x \rangle = E[x]$$

$$\mu = \int_{-\infty}^{+\infty} x f(x)\,dx \quad \text{[continuous]} \qquad \mu = \sum_{k=1}^{N} x_k f_k \quad \text{[discrete]}$$

- **Variance**

$$\text{var}(x) = \overline{(x-\mu)^2} = \sigma^2 = \langle (x-\mu)^2 \rangle = E\left[(x-\mu)^2\right]$$

$$\sigma^2 = \int_{-\infty}^{+\infty} (x-\mu)^2 f(x)\,dx \qquad \sigma^2 = \sum_{k=1}^{N} (x_k-\mu)^2 f_k$$

- **Standard Deviation**

$$\sigma = \sqrt{\sigma^2}$$

- **Functions of a Random Variable**

Consider g(x), where x is a random variable with density f(x)

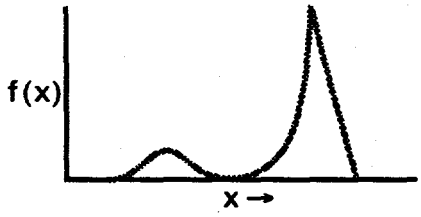
$$E[g(x)] = \int_{-\infty}^{+\infty} g(x) f(x)\,dx \qquad E[g(x)] = \sum_{k=1}^{N} g_k f_k$$

[9]

The key to Monte Carlo methods is the notion of *random sampling.*

- The problem can be stated this way:

  Given a probability density, **f(x)**, produce a sequence of $\hat{x}$'s.
  The $\hat{x}$'s should be distributed in the same manner as **f(x)**.



- The use of random sampling distinguishes Monte Carlo from all other methods
- When Monte Carlo is used to solve the integral Boltzmann transport equation
  - — Random sampling models the outcome of physical events
    (e.g., neutron collisions, fission process, source, ...)
  - — Computational geometry models the arrangement of materials

[10]

Monte Carlo

&

Transport Equation

Boltzmann Transport Equation — Time-Independent, Linear

$$\Psi(\mathbf{r}, \mathbf{v}) = \int\left[\int\Psi(\mathbf{r}', \mathbf{v}')\, C(\mathbf{v}' \rightarrow \mathbf{v}, \mathbf{r}')\, d\mathbf{v}' + Q(\mathbf{r}', \mathbf{v})\right] T(\mathbf{r}' \rightarrow \mathbf{r}, \mathbf{v})\, d\mathbf{r}'$$

where

- $\Psi(\mathbf{r}, \mathbf{v})$ = particle collision density
- $Q(\mathbf{r}', \mathbf{v})$ = source term
- $C(\mathbf{v}' \rightarrow \mathbf{v}, \mathbf{r}')$ = collision kernel, change **velocity** at fixed position
- $T(\mathbf{r}' \rightarrow \mathbf{r}, \mathbf{v})$ = transport kernel, change **position** at fixed velocity

- **Angular Flux** $\psi(\mathbf{r}, \mathbf{v}) = \dfrac{\Psi(\mathbf{r}, \mathbf{v})}{\Sigma(\mathbf{r}, |\mathbf{v}|)}$

- **Scalar Flux** $\Phi(\mathbf{r}, |\mathbf{v}|) = \int_{\vec{\Omega}} \dfrac{\Psi(\mathbf{r}, \mathbf{v})}{\Sigma(\mathbf{r}, |\mathbf{v}|)}\, d\vec{\Omega}, \qquad \mathbf{v} = |\mathbf{v}|\vec{\Omega}$

Source term for the Boltzmann equation

$$Q(\mathbf{r},\mathbf{v}) = \begin{cases} S(\mathbf{r},\mathbf{v}) & \Leftarrow \text{Fixed Source} \\ S(\mathbf{r},\mathbf{v}) + \int \Psi(\mathbf{r},\mathbf{v}')\, F(\mathbf{v}' \rightarrow \mathbf{v},\mathbf{r})\, d\mathbf{v}' & \Leftarrow \text{Fixed Source + Fission} \\ \frac{1}{K} \int \Psi(\mathbf{r},\mathbf{v}')\, F(\mathbf{v}' \rightarrow \mathbf{v},\mathbf{r})\, d\mathbf{v}' & \Leftarrow \text{Eigenvalue} \end{cases}$$

where

- $S(\mathbf{r},\mathbf{v})$ = fixed source
- $F(\mathbf{v}' \rightarrow \mathbf{v},\mathbf{r})$ = creation operator (due to fission), particle at $(\mathbf{r},\mathbf{v}')$ creates particle at $(\mathbf{r},\mathbf{v})$
- K = eigenvalue

$$\Psi(\mathbf{r},\mathbf{v}) = \int \left[\int \Psi(\mathbf{r}',\mathbf{v}')\, C(\mathbf{v}' \rightarrow \mathbf{v},\mathbf{r}')\, d\mathbf{v}' + Q(\mathbf{r}',\mathbf{v}) \right] T(\mathbf{r}' \rightarrow \mathbf{r},\mathbf{v})\, d\mathbf{r}'$$

Assumptions

- static homogeneous medium
- time-independent
- Markovian — next event depends only on current (r,v), not on previous events
- particles do not interact with each other
- neglect relativistic effects
- no long-range forces (particles fly in straight lines between events)
- material properties are not affected by particle reactions
- etc., etc.

→ can use superposition principle

Basis for Monte Carlo Solution Method

Let $\mathbf{p} = (\mathbf{r},\mathbf{v})$ and $R(p' \rightarrow p) = C(\mathbf{v}' \rightarrow \mathbf{v}, \mathbf{r}')\, T(\mathbf{r}' \rightarrow \mathbf{r}, \mathbf{v})$

Expand Ψ into components having 0,1,2,...,k collisions

$$\Psi(p) = \sum_{k=0}^{\infty} \Psi_k(p), \qquad \text{with } \Psi_0(p) = \int Q(\mathbf{r}', \mathbf{v})\, T(\mathbf{r}' \rightarrow \mathbf{r}, \mathbf{v})\, d\mathbf{r}'$$

By definition,

$$\Psi_k(p) = \int \Psi_{k-1}(p')\, R(p' \rightarrow p)\, dp'$$

Note that collision **k** depends only on the results of collision **k-1**, and not on any prior collisions **k-2**, **k-3**, ...

Statistical approach to determining Ψ_k

$$\Psi_k(p) = \int \Psi_{k-1}(p')\, R(p' \rightarrow p)\, dp'$$

- interpret terms in the following manner:

  $\Psi_{k-1}(p')$ = probability density for occurence of $(k\text{-}1)^{st}$ collisions at p'

  $R(p' \rightarrow p)$ = conditional probability that a $(k\text{-}1)^{st}$ collision at p' will result in a $(k)^{th}$ collision at p.

- Monte Carlo method

  (1) Randomly sample p' from $\Psi_{k-1}(p')$

  (2) Given p', randomly sample p from $R(p' \rightarrow p)$

  (3) If p lies within dp_i at p_i, tally 1 in bin i

  → Repeat steps (1), (2), (3) N times,

  then $\bar{\Psi}_k(p_i)\, dp_i \sim \{\text{counts for bin i}\} / N$

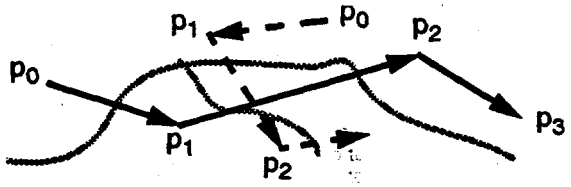
Histories

- After repeated substitution for Ψ_k

$$\Psi_k(p) = \int \Psi_{k-1}(p') R(p' \rightarrow p)\, dp'$$

$$= \int \cdots \int \Psi_0(p_0) R(p_0 \rightarrow p_1) R(p_1 \rightarrow p_2) \cdots R(p_{k-1} \rightarrow p)\, dp_0 \cdots dp_{k-1}$$

- A **"history"** is a sequence of states $(p_0, p_1, p_2, p_3, \ldots)$



For estimates in a given region, tally the occurrences for **each collision** of **each "history"** within a region

$$\Psi_k(p) = \int \cdots \int \Psi_0(p_0) R(p_0 \rightarrow p_1) R(p_1 \rightarrow p_2) \cdots R(p_{k-1} \rightarrow p)\, dp_0 \cdots dp_{k-1}$$

Monte Carlo approach:

- Generate a sequence of states, $(p_0, p_1, \ldots, p_k)$, [i.e., a ***history***] by
  - — Randomly sample from PDF for source: $\Psi_0(p_0)$
  - — Randomly sample from PDF for k^{th} transition: $R(p_{k-1} \rightarrow p_k)$

- Generate estimates of results, by averaging over **M** histories:

$$A = \int A(p) \Psi(p)\, dp \approx \frac{1}{M} \sum_{m=1}^{M} \left(\sum_{k=1}^{\infty} A(p_{k,m}) \right)$$

Monte Carlo

&

Simulation

"Simulation is better than reality"

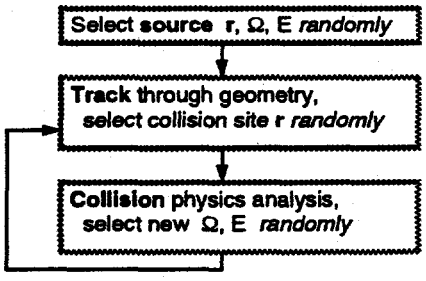
Richard W. Hamming, 1991

Simulation approach to particle transport

- Faithfully simulate the history of a single particle from birth to death

  During the particle history,

  — model collisions using physics equations & cross-section data
  — model free-flight using computational geometry
  — tally the occurrences of events in each region



- Repeat the simulation for many histories, accumulating the tallies

- Fundamental rule:

Think like a neutron

Source

- Random sampling

  E, Ω — analytic, discrete, or piecewise-tabulated PDF's

- Computational geometry

  r — sample from region in 3-D space, or from discrete PDF

Tracking

- Random sampling

  $d_{collide}$ — distance to collision, from mfp & exponential PDF

- Computational geometry

  d_{geom} — distance-to-boundary, ray-tracing, next-region, ...

Collisions

- Random sampling

  E', Ω' — analytic, discrete, or piecewise-tabulated PDF's

- Physics

  Σ, f(μ) — cross-section data, angular PDF's, kinematics, ...

Tallies

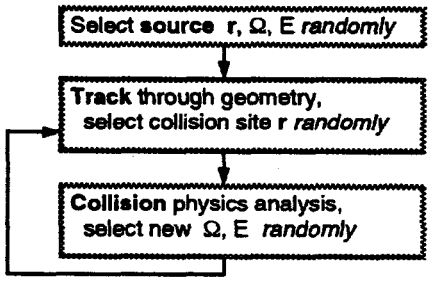
- Statistics

Variance Reduction

- Random sampling

Single particle

- random-walk for particle history
- simulate events, from birth to death
- tally events of interest



Batch of histories ("generation")

- random-walk for many particle histories
- tally the aggregate behavior

Overall

- timesteps
  - — geometry changes
  - — material changes
    - fuel depletion
    - burnable absorbers
    - control rods

```
* Loop over timesteps
•   * Loop over batches
•   •   * Loop over histories
•   •   •   → random walk
•   •   • • •
•   •   → update Keff & reaction rates
•   • • •
•   → compute statistics
•   → update number densities, .....
• • •
```

Random Number Generation

"Randomness is a negative property; it is the absence of any pattern."

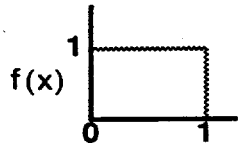
Richard W. Hamming, 1991

"... random numbers should not be generated by a method chosen at random."

D. Knuth, ~1981

Random Number Generators (RNGs)

- **Numbers** are not random; a **sequence** of numbers can be.
- Truly random sequences are generally **not** desired on a computer.
- Pseudo-random sequences:
  - repeatable (deterministic)
  - pass statistical tests for randomness
- RNG — function which generates a sequence of numbers which appear to have been randomly sampled from a uniform distribution on (0,1).
  - — probability density for f(x):



  - — typical usage in codes: `r = ranf()`
  - — also called "pseudo-random number generators" (PRNG)
- All other random sampling is performed using this basic RNG

Most production-level Monte Carlo codes for particle transport use linear congruential random number generators

$$S_{i+1} = [S_i \cdot g + c] \mod 2^m$$

- robust, 40 years of heavy-duty use
- simple, fast
- theory is well-understood (e.g., DE Knuth, Vol 2, 177 pages)
- not the "best" generators, but good enough — RN's are used in unpredictable ways during particle simulation
- To achieve reproducibility of Monte Carlo calculations, despite vectorization or varying numbers of parallel processors, there must be a fast, direct method for skipping ahead (or back) in the random sequence.

Linear Congruential PRNGs

- due to Lehmer, 1949
- most common method, excellent (when not abused)
- Method:

$s_0 \leftarrow$ initial value

$r_k \leftarrow s_k / p$

$s_{k+1} \leftarrow [g \cdot s_k + c] \mod p$

where

s_k, g, c, p = integers, r_k = real

s_k = seed
g = generator, or multiplier
c = increment
p = modulus
r_k = psuedo-random number, $0 \le r_k \le 1$

- "mod p" ⇒ "remainder after division by p", absolutely no roundoff is permitted
- Multiplicative: c = 0
- Mixed: c > 0

Example #1: $s_{k+1} \leftarrow [g \cdot s_k + c] \bmod p$

with $g = 47,\ c = 1,\ s_0 = 1,\ p = 100$

```
s( 0) =  1
s( 1) =  (47x1  + 1) mod 100 =    48 mod 100 =  48
s( 2) =  (47x48 + 1) mod 100 =  2257 mod 100 =  57
s( 3) =  (47x57 + 1) mod 100 =  2680 mod 100 =  80
s( 4) =  (47x80 + 1) mod 100 =  3761 mod 100 =  61
s( 5) =  (47x61 + 1) mod 100 =  2868 mod 100 =  68
s( 6) =  (47x68 + 1) mod 100 =  3197 mod 100 =  97
s( 7) =  (47x97 + 1) mod 100 =  4560 mod 100 =  60
s( 8) =  (47x60 + 1) mod 100 =  2821 mod 100 =  21
s( 9) =  (47x21 + 1) mod 100 =   988 mod 100 =  88
s(10) =  (47x88 + 1) mod 100 =  4137 mod 100 =  37
s(11) =  (47x37 + 1) mod 100 =  1740 mod 100 =  40
s(12) =  (47x40 + 1) mod 100 =  1881 mod 100 =  81
s(13) =  (47x81 + 1) mod 100 =  3808 mod 100 =   8
s(14) =  (47x8  + 1) mod 100 =   377 mod 100 =  77
s(15) =  (47x77 + 1) mod 100 =  3620 mod 100 =  20
s(16) =  (47x20 + 1) mod 100 =   941 mod 100 =  41
s(17) =  (47x41 + 1) mod 100 =  1928 mod 100 =  28
s(18) =  (47x28 + 1) mod 100 =  1317 mod 100 =  17
s(19) =  (47x17 + 1) mod 100 =   800 mod 100 =   0
s(20) =  (47x0  + 1) mod 100 =     1 mod 100 =   1
s(21) =  (47x1  + 1) mod 100 =    48 mod 100 =  48
s(22) =  (47x48 + 1) mod 100 =  2257 mod 100 =  57
 etc.
```


Example #2: $s_{k+1} \leftarrow [g \cdot s_k + c] \bmod p$

with $g = 5,\ c = 1,\ s_0 = 1,\ p = 100$

```
s( 0) =  1
s( 1) =  (5x1  + 1) mod 100 =    6 mod 100 =   6
s( 2) =  (5x6  + 1) mod 100 =   31 mod 100 =  31
s( 3) =  (5x31 + 1) mod 100 =  156 mod 100 =  56
s( 4) =  (5x56 + 1) mod 100 =  281 mod 100 =  81
s( 5) =  (5x81 + 1) mod 100 =  406 mod 100 =   6
s( 6) =  (5x6  + 1) mod 100 =   31 mod 100 =  31
 etc.
```

Example #3: $s_{k+1} \leftarrow [g \cdot s_k + c] \bmod p$

with $g = 5,\ c = 0,\ s_0 = 1,\ p = 100$

```
s( 0) =  1
s( 1) =  (5x1 ) mod 100 =    5 mod 100 =   5
s( 2) =  (5x5 ) mod 100 =   25 mod 100 =  25
s( 3) =  (5x25) mod 100 =  125 mod 100 =  25
s( 4) =  (5x25) mod 100 =  125 mod 100 =  25
 etc.
```

Linear Congruential PRNG - Selecting g, c, p, s_0

$$s_{k+1} \leftarrow [\, g \cdot s_k + c \,] \bmod p$$

- For theoretical & practical considerations in selecting g, c, p, s_0, see D. E. Knuth, The Art of Computer Programming, Vol 2

- Modulus (p):
  - **choose $p = 2^N$**
  - simplifies "mod p"— discard all but the N least significant bits
  - simplifies division by p — shift the "point" left by N bits
  - N should be as large as possible, to maximize the period — N > 35 is best.
  - Usually, choose N to be number of bits in largest positive integer, for efficient computing

- Generator (g), Initial Seed (s_0), & Increment (c) :
  - choose g & c to maximize the period
  - large g is best to reduce serial correlation
  - obviously, g=1 or g=0 are bad
  - **For c = 0 (multiplicative PRNG):**

    choosing (1) g mod 8 = 3 or 5
    (2) s_0 = odd

    results in: period = 2^{N-2}, the maximum possible period.
  - **For c > 0 (mixed PRNG):**

    choosing (1) c relatively prime to p
    (2) (g-1) to be a multiple of every prime factor of p
    (3) (g-1) to be a multiple of 4 if p is a multiple of 4

    results in: period = 2^N, the maximum possible period.

Multiplicative congruential method — Lehmer

$$S_{i+1} = g \cdot S_i + c \mod 2^m, \qquad 0 < S_i < 2^m$$

$$\xi_i = S_i / 2^m, \qquad 0 < \xi < 1$$

Typical parameters

		2^m	period	g	c
RACER	(KAPL)	2^{47}	2^{45}	84,000,335,758,957	0
RCP	(BAPL)	2^{48}	2^{48}	2^9+1	59,482,192,516,946
MORSE	(ORNL)	2^{47}	2^{45}	5^{15}	0
MCNP	(LANL)	2^{48}	2^{46}	5^{19}	0
VIM	(ANL)	2^{48}	2^{46}	5^{19}	0
RANF	(CRAY)	2^{48}	2^{46}	44,485,709,377,909	0
—	(G. Marsaglia)	2^{32}	2^{32}	69069	1

Aside ...

For the multiplicative congruential method,
why is the period limited to a maximum of 2^{N-2} ??

$$s_{k+1} \leftarrow g \cdot s_k \mod p, \qquad s_0 \text{ odd}, \quad g \bmod 8 = 3 \text{ or } 5$$

- All s_k's are odd, g is odd

  $\Rightarrow$ $g \cdot s_k$ will always be odd, reduces period by a factor of two.

- For g mod 8 = 3, trailing bits of g are (...011)

  $g \cdot s_k$ = (...011)•(....11) = (...11)

  or

  $g \cdot s_k$ = (...011)•(....01) = (...01)

  $\Rightarrow$ next-to-last bit of s_k will not change, reduces period by a factor of two.

- For g mod 8 = 5, trailing bits of g are (...101)

  $g \cdot s_k$ = (...101)•(...1x1) = (...1x1)

  or

  $g \cdot s_k$ = (...101)•(...0x1) = (...0x1)

  $\Rightarrow$ third-to-last bit of s_k will not change, reduces period by a factor of two.

Example -- CYBER-205 RANF

$$s_{k+1} \leftarrow [\, g \cdot s_k + c \,] \mod p$$

FORTRAN

```
common /q8ranfc/ seed
r = ranf()
```

META

```
LOD   s_descr, s          *load the seed
EX    g, 84000335758957   *generator
EX    e, 65489            *exponent, 2**-47
MPYL  g, s, s             *mult, keep last 47 bits
STO   s_descr, s          *store new seed
PACK  e, s, r             *insert exponent
ADDN  r,  , r             *normalized result
```

Note: (1) $0 < r < 1$
(2) scalar timing ~320 ns / prn
(3) to vectorize — "unroll" or "replicate", vector timing ~30 ns / rn

How long will the PRNs last ?

		time to generate ALL 2^{45} RNs
Sharp EL-515s		**1 M yr**
CYBER-205,	**scalar**	**4 mos**
CRAY-1,	**vector**	**15 days**
CYBER-205,	**2-pipe vector**	**12 days**
CYBER-205,	**4-pipe vector**	**6 days**
CRAY-XMP/48,	**vector x 4**	**3 days**
CRAY-2,	**vector x 4**	**30 hr**
ETA-10,	**vector x 8**	**13 hr**
cray-c90,	**vector x 16**	**4 hr**

Other PRNGs

- Middle-square method: s_{k+1} = middle digits of s_k^2
- Quadratic-congruential: $s_{k+1} = [\, a \cdot s_k^2 + b s_k + c \,] \bmod p$
- Modified Middle-square: $s_{k+1} = [\, s_k \cdot (s_k + 1) \,] \bmod p$
- Additive: $s_{k+1} = [\, s_k + s_{k-i} \,] \bmod p$
- Additive (or Shift): $s_k = [\, s_{k-j} + s_{k-i} \,] \bmod p$
- Generalized Additive (or Shift): $s_k = [\, a_1 s_{k-1} + a_2 s_{k-2} \ldots a_i s_{k-i} \,] \bmod p$
- Quasi-random sequences ....
- etc., etc., ...

Testing PRNGs

- See Knuth, Vol. 2, pp. 38-113
- Empirical Tests:

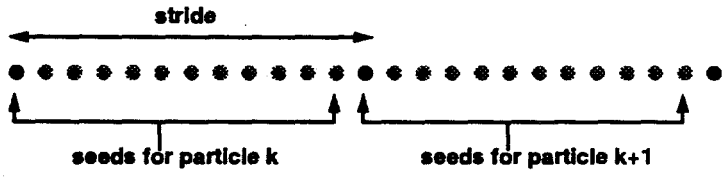
Chi-square test	Kolmogorov-Smirnov test	Frequency test
Serial pair, triplet, ..., distributions	Gap test	Poker tests
Coupon Collector test	Run test	Maximum-of-t test
Collision test	Serial Correlation coefficients	
etc.		

- Theoretical Tests

Spectral test	Serial Correlation (global)	etc.

Reproducibility of a Particle History

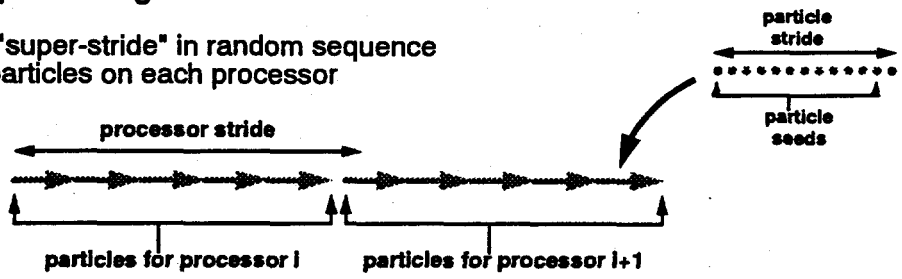
- use separate, distinct random sequence for each particle
- starting seeds for separate particles are separated by "stride"



- stride should be large enough to prevent overlap (for most histories)
  - — 1000 is common for reactor analysis problems
    - splitting & variance reduction not needed for in-core physics
    - reduces total random number usage
  - — 4,297 is the "old" default for MCNP & VIM
  - — 152,917 is the default for MCNP & VIM
    - prepared for lots of splitting & variance reduction
    - potential for lots of secondary particles

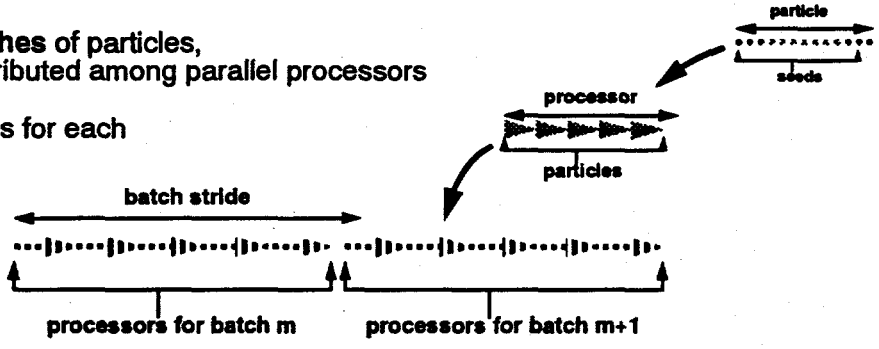
Parallel processing

- take "super-stride" in random sequence for particles on each processor



Eigenvalue Problems

- **batches** of particles, distributed among parallel processors
- seeds for each



To skip ahead *k* steps in the random sequence, [initial seed] → [k^{th} seed]

$$
\begin{aligned}
S_k &= g \cdot S_{k-1} + c \mod 2^m \\
&= g \cdot (g \cdot S_{k-2} + c) + c \mod 2^m \\
&= g(\ldots\ldots g(g(g S_0 + c) + c) + c) \ldots\ldots) + c \mod 2^m \\
&= g^k \cdot S_0 + c \cdot (g^{k-1} + g^{k-2} + \ldots\ldots + g + 1) \mod 2^m \\
&= g^k \cdot S_0 + c \cdot (g^k - 1)/(g-1) \mod 2^m
\end{aligned}
$$

- Periodic sequence:
  negative skip k_n equivalent to positive skip ***(period - k_n)***
- Can skip from any seed directly to any other:
  initial seed → i^{th} seed for j^{th} particle on m^{th} processor in n^{th} batch
  particle i → particle j
  batch i → batch j
- All arithmetic must be performed mod 2^m, without truncation or roundoff

$$S_k = G(k) \cdot S_0 + C(k) \mod 2^m$$

Define $G(k) = g^k \bmod 2^m$

m = 32 or 48 (typical), based on the size of a computer word

$-2^m < k < +2^m$, based on desired "stride"

Denote the j^{th} bit of k by $k_{[j]}$, so that

$$k = 2^{m-1} k_{[m-1]} + 2^{m-2} k_{[m-2]} + \ldots\ldots + 2^1 k_{[1]} + 2^0 k_{[0]}$$

Substituting into G(k) yields

$$G(k) = g^k \bmod 2^m = g^{\sum_{i=0}^{m-1} k_{[i]} 2^i} \bmod 2^m$$

$$= \prod_{j=0}^{m-1} \left(g^{2^j}\right)^{k_{[i]}} \bmod 2^m$$

Efficient algorithms for evaluating G(k) can be formulated using only m steps

Enumerating a few terms of G(k) makes the algorithm obvious

$$G(k) = \left(g^1\right)^{k_{[0]}} \cdot \left(g^2\right)^{k_{[1]}} \cdot \left(g^4\right)^{k_{[2]}} \cdot \left(g^8\right)^{k_{[3]}} \cdots \left(g^{2^{m-1}}\right)^{k_{[m-1]}} \bmod 2^m$$

Note that $k_{[j]}=0$ or $k_{[j]}=1$, so that each term $\left(g^n\right)^{k_{[i]}}$ evaluates to either 1 or g^n

Algorithm G:

$G \leftarrow 1$, $h \leftarrow g$, $i \leftarrow k + 2^m \bmod 2^m$

while $i > 0$

if i = odd: $G \leftarrow G\,h \bmod 2^m$

$h \leftarrow h^2 \bmod 2^m$

$i \leftarrow \lfloor i/2 \rfloor$

Remarks

- Algorithm G terminates after m steps, rather than k steps
- Negative strides are trivial, due to periodicity: $G(-s) = G(2^m - s)$

Define $C(k) = c\left(\frac{g^k - 1}{g - 1}\right) \mod 2^m$

$= c \cdot \left(1 + g + g^2 + g^3 + \ldots + g^{k-1}\right) \mod 2^m$

The series for C(k) can be evaluated recursively, similar to G(k), in **m** steps:

Algorithm C:

$C \leftarrow 0, \quad f \leftarrow c, \quad h \leftarrow g, \quad i \leftarrow k + 2^m \mod 2^m$

while $i > 0$

if i = odd: $\quad C \leftarrow C\,h + f \mod 2^m$

$f \leftarrow f(h + 1) \mod 2^m$

$h \leftarrow h^2 \mod 2^m$

$i \leftarrow \lfloor i/2 \rfloor$

- Since most of the common random number generators use c = 0, Algorithm C is generally not required.
- Algorithm C can be included with Algorithm G, at very little extra cost

Computer Coding

- All integer adds & multiplies must be performed exactly (mod 2^m), without truncation or roundoff
- For $m \le 32$, reasonably portable coding is straightforward in C
- For $m > 32$, or for Fortran coding
  - — split integers into "high" & "low" pieces
  - — perform modular arithmetic — straightforward (but tedious)
    - ⇒ see *Hendricks, Trans ANS 62, 283, 1990* — same techniques can be used for arbitrary skips
  - — For Fortran, reasonably portable coding, except:
    - Sparc2, rs6000, indigo, ...: "double precision"
    - Cray: "real"

R. N. Generator for 32-bit machines (*sparc2, rs6000, indigo, ...*)

$$s \leftarrow 69069 \cdot s + 1 \mod 2^{32}$$

Random Number Generator→

```
static unsigned long    seed_c=1;
static double           norm=(1./4294967296.);

double cranf_( void ) {
    unsigned long   g=69069,  c=1;
    seed_c = g * seed_c + c;
    return  ( (double) seed_c * norm );
}
```

Routine for Arbitrary Skips →

```
void  cranfjump_(   unsigned long   *seed,
                    double          *jump,
                    unsigned long   *newseed ) {

    unsigned long   j, gen=1, inc=0,  g=69069,  c=1;

    if( *jump < 0 )   j = *jump + 4294967296.;
    else              j = *jump;

    for( ; j;  j>>=1 ) {

        if( j&1 ) {
                      inc  = inc*g + c;
                      gen  = gen*g;
        }
        c *= g+1;
        g *= g;
    }
    *newseed  = gen * (*seed)  +  inc;
}
```

Compute:

$gen = g^k$

$inc = c(g^k-1)/(g-1)$

Fortran, 48-bit generator: $g=5^{19}$, c=0, m=48 (VIM & MCNP)

C, 32-bit generator: g=69069, c=1, m=32 (from Marsaglia)

		Sparc2	rs6000/350
C, 32-bit			
random number		1.0 μs	.7 μs
skip forward,	average for $+1...10^5$	7.4 μs	10 μs
skip backward,	average for $-1...-10^5$	4.0 μs	20 μs
Fortran, 48-bit			
random number		3.6 μs	2.3 μs
skip forward,	+152,917	163 μs	78 μs
skip backward,	-152,917	458 μs	215 μs
skip forward,	average for $+1...10^5$	160 μs	75 μs
skip backward,	average for $-1...-10^5$	695 μs	232 μs
skip forward,	+1,152,917	189 μs	90 μs
skip forward,	+1,152,917, **brute force**	4.1 **sec**	2.6 **sec**
skip backward,	-1,152,917	456 μs	210 μs
skip backward,	-1,152,917, **brute force**	8 **year**	5 **year**

- Algorithms for direct skip-ahead in the random sequence are simple, fast, convenient, ..., for modern Monte Carlo codes

- Arbitrary positive or negative strides can be taken, without precomputing or hardwiring specific constants

- Direct skip-ahead simplifies the initialization of random numbers for each particle, especially for parallel processing

- Algorithms described are currently used in:

parallel VIM	— ANL	— Sun, rs6000, SP1, ...
RACER	— KAPL	— Cray, Meiko CS1 & CS2, Sun, SGI, ...
KENO-Va	— CSN (Spain)	— Convex-C3440

Random

Sampling

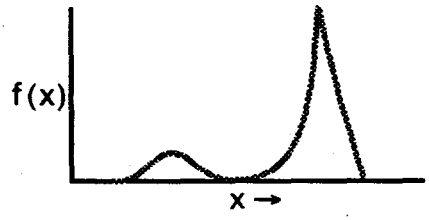
"Anyone who considers arithmetical methods of producing random digits is, of course, in a state of sin."

J. Von Neumann, 1951

Probability Density Function (PDF)

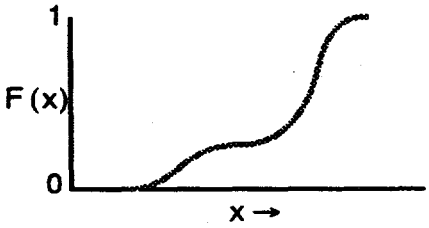
- f(x)
- $0 \le f(x)$
- Probability$\{ a \le x \le b \} = \int_a^b f(x)\,dx$
- Normalization:

$$\int_{-\infty}^{+\infty} f(x)\,dx = 1$$



Cumulative Distribution Function (CDF)

- $F(x) = \int_{-\infty}^{x} f(x')\,dx'$
- $0 \le F(x) \le 1$
- $0 \le \frac{d}{dx} F(x)$
- $F(-\infty) = 0, \quad F(\infty) = 1$



Monte Carlo Codes

Categories of Random Sampling

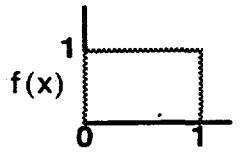
- Random number generator → uniform PDF on (0,1)
- Sampling from analytic PDF's → normal, exponential, Maxwellian, ...
- Sampling from tabulated PDF's → angular PDF's, spectrum, ...

For Monte Carlo Codes...

- Uniform random numbers, ξ, are produced by the R.N. generator on (0,1)
- Non-uniform random variates are produced from the ξ's by
  - — direct inversion
  - — rejection methods
  - — transformations
  - — composition (mixtures)
  - — sums, products, ratios, ...
  - — table lookup + interpolation
  - — lots (!) of other tricks .....
- < 10% of total cpu time (typical)

Pseudo-Random Numbers

- Not strictly "random", but good enough
  - — pass statistical tests
  - — reproducible sequence
- Uniform PDF on (0,1)
- Must be easy to compute



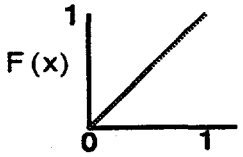
Multiplicative congruential method

- Algorithm

  S_0 = initial seed, odd integer, < M

  $S_k = G \cdot S_{k-1} \mod M, \quad k = 1, 2, ...$

  $\xi_k = S_k / M$



Usage

- In algorithms, usually denote RN uniform on (0,1) by ξ
- In codes, invoke basic RN generator: **r = ranf()**
- Each new usage of ξ or ranf() generates a **new** RN

Direct Sampling

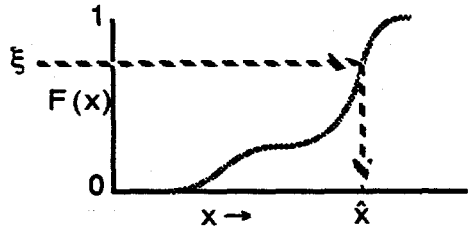
Direct Solution of

$\hat{x} \leftarrow F^{-1}(\xi)$

Solve for $\hat{x}$: $\xi = \int_{-\infty}^{\hat{x}} f(x')\,dx'$

Sampling Procedure:

- Generate ξ
- Determine $\hat{x}$ such that $F(\hat{x}) = \xi$



Advantages

- Straightforward mathematics & coding
- "High-level" approach

Disadvantages

- Often involves complicated functions
- In some cases, F(x) cannot be inverted (e.g., Klein-Nishina)

Rejection Sampling

Von Neumann

"..... it seems objectionable to compute a transcendental function of a random number."

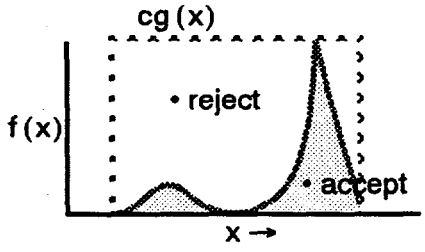
Select a bounding function, g(x), such that

- $c \cdot g(x) \geq f(x)$ for all x
- g(x) is an easy-to-sample PDF

Sampling Procedure:

- sample $\hat{x}$ from g(x): $\hat{x} \leftarrow G^{-1}(\xi_1)$
- test: $\xi_2 \cdot cg(\hat{x}) \leq f(\hat{x})$

  if ***true*** → accept $\hat{x}$, done
  if ***false*** → reject $\hat{x}$, try again



Efficiency

η = % of trials accepted = $\int f(x)\,dx \,/\, \int cg(x)\,dx$

Advantages

- Simple computer operations

Disadvantages

- "Low-level" approach, sometimes hard to understand
- Will be inefficient if cg(x) and f(x) do not have "similar" shapes & ranges

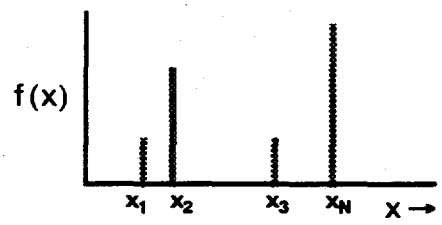
Discrete PDF's

- **Discrete PDF**

$\{f_k\}$, where $f_k = f(x_k)$, $k=1,...,N$

$$f_k \geq 0$$

$$\sum_{j=1}^{N} f_j = 1$$



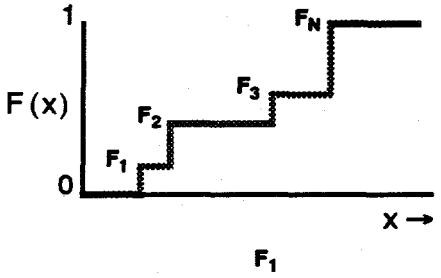
- **Discrete CDF**

$\{F_k\}$, where $F_k = \sum_{j=1}^{k} f_j$, $k=1, ..., N-1$

and

$F_0 = 0$,

$F_N = 1$



F_1

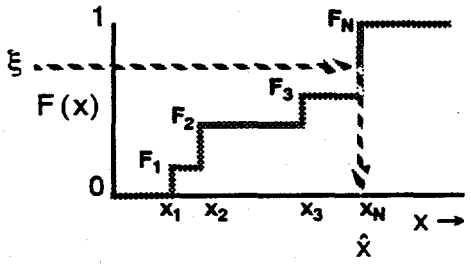
Sampling from Discrete PDF's — Conventional Procedure

Direct Solution of $\hat{x} \leftarrow F^{-1}(\xi)$

(1) Generate ξ

(2) Determine **k** such that $F_{k-1} \leq \xi \leq F_k$

(3) Return $\hat{x} = x_k$



Step (2) requires a **table search**

- **linear** table searches require O(N) time — use when N small
- **binary** table searches require $O(\ln_2 N)$ time — use when N large

For some discrete PDFs, F_k's are not precomputed.

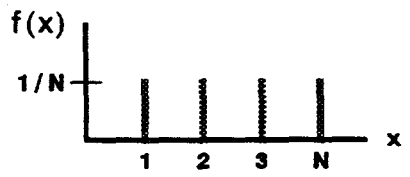
- **linear** search, with F_k's computed on-the-fly as needed

Example — Sampling from Discrete Uniform PDF

Discrete Uniform PDF

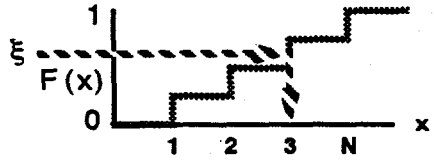
$f_k = 1/N, \quad k = 1, ..., N$

$F_k = k/N, \quad F_0 = 0, \; F_N = 1$



Sampling procedure:

Could use table search method, ...



Easier, for this special case:

$K \leftarrow \lfloor 1 + N\xi \rfloor$, where $\lfloor y \rfloor$ is the "floor" function, largest integer < y

Note: must be sure that $\lfloor 1 + N\xi \rfloor \leq N$

Examples — Sampling from Discrete PDFs

Multigroup scattering

- inelastic scatter from MUFT-group **g** to MUFT-group **g'**
- thermal scatter from THAV-group **g** to THAV-group **g'**

$$f_{g'} = \frac{\sigma_{g \to g'}}{\sum_k \sigma_{g \to k}},$$

where $g' = 1, ..., G_{THAV}$ for thermal,
$g' = g, ..., G_{MUFT}$ for inelastic,

$k = 1, ..., G_{THAV}$ for thermal,
$k = g, ..., G_{MUFT}$ for inelastic

Selection of scattering nuclide for a collision

$$f_k = \frac{N^{(k)} \sigma_s^{(k)}}{\sum_{k'=1}^{K} N^{(k')} \sigma_s^{(k')}}$$

where $k = 1, ..., K$

K = # nuclides in composition

Sampling from Discrete PDF's — Alias Method

Any discrete PDF can be converted into "Alias sampling" form

original PDF: $\{ f_k \}$, $k=1, ..., N$

where f_k = probability of selecting $x = x_k$

aliased PDF: $\{ q_k, i_k, \}$, $k=1, ...,N$

where $\frac{1}{N} \cdot q_k$ = prob. of selecting $\hat{x} = x_k$

$\frac{1}{N} \cdot (1 - q_k)$ = prob. of selecting $\hat{x} = x_{i_k}$

Alias sampling procedure:

Select uniformly for $\hat{k}$: $\hat{k} \leftarrow \lfloor 1 + N\xi_1 \rfloor$

Select either $\hat{k}$ or its "alias" $i_{\hat{k}}$: if $\xi_2 < q_{\hat{k}}$, $\hat{x} \leftarrow x_{\hat{k}}$,

otherwise, $\hat{x} \leftarrow x_{i_{\hat{k}}}$

.....(continued on next page)

Sampling from Discrete PDF's — Alias Method (continued)

Why bother with "alias sampling" ?

- **No table search needed, requires O(1) time**
- **Sampling time is constant & independent of size of PDF**
- **Vectorizes completely & efficiently**
- **Fastest possible way to sample discrete PDFs**
- **Invented by Brown (who later found out Walker did it 3 yr earlier)**

Creating the "aliased PDF" amounts to converting an N-way tree from

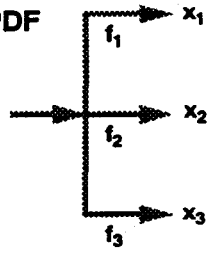
arbitrary branching probabilities with **single** outcomes

to

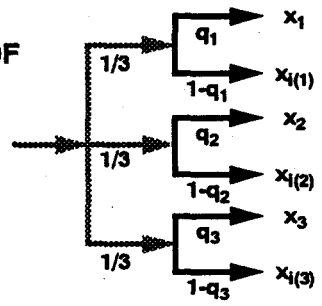
uniform branching probabilities with **dual** outcomes

(See FB Brown & RACER coding for the set up algorithm)

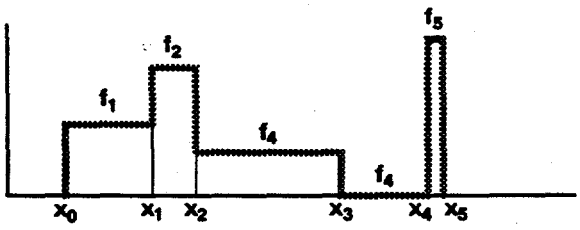
discrete PDF



aliased PDF



Examples — Sampling from "Histogram" PDFs



Two-step sampling: (1) Sample from discrete PDF to select a bin
(2) Sample from uniform PDF within bin

- Discrete PDF: $p_k = f_k \cdot (x_k - x_{k-1}), \quad k = 1, ..., N$

  — generate ξ
  — use table search or alias method to select **K**

- Uniform sampling within bin **K**:

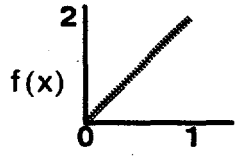
  — generate ξ
  — then,

$$\hat{x} \leftarrow x_{K-1} + (x_K - x_{K-1})\xi$$

Examples — Sampling from Linear PDF on (0,1)

$$f(x) = 2x, \quad 0 \le x \le 1$$

$$F(x) = \int_0^x f(x')\,dx' = \int_0^x 2x'\,dx' = x^2$$

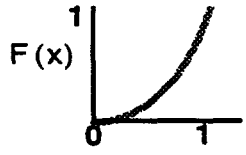


Direct Sampling:

solving $F(\hat{x}) = \xi$ or $\hat{x} \leftarrow F^{-1}(\xi)$

gives:

$$\hat{x} \leftarrow \sqrt{\xi}$$



Examples — Sampling from x^n PDF on (0,1)

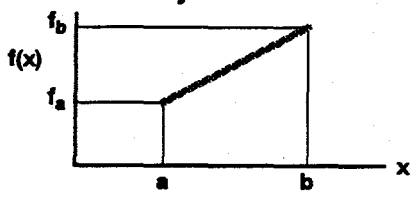
$$f(x) = (n+1)x^n, \quad 0 \le x \le 1$$

$$F(x) = x^{n+1}$$

Solving $F(\hat{x}) = \xi$ gives: $\hat{x} \leftarrow \xi^{\frac{1}{n+1}}$

(Note: only for 0<x<1, does not apply to general intervals !)

Examples — Sampling from Arbitrary Linear PDF



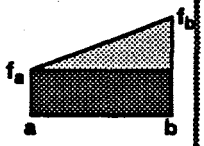
Scheme 1

Decompose into uniform + linear

$p_1 = f_a\,(b\text{-}a)$

$p_2 = (f_b\text{-}f_a)(b\text{-}a)\,/\,2$

$p_1 + p_2 = 1$



$f(x) = p_1 \cdot$ ▭ $+ \quad p_2 \cdot$ ◿

Sampling scheme:

if $\xi_1 < p_1$, $\quad \hat{x} \leftarrow a + (b-a)\,\xi_2$

else $\quad \hat{x} \leftarrow a + (b-a)\,\sqrt{\xi_2}$

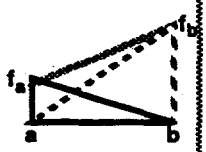
Scheme 2

Decompose into linear + linear

$p_1 = f_a\,(b\text{-}a)\,/\,2$

$p_2 = f_b\,(b\text{-}a)\,/\,2$

$p_1 + p_2 = 1$



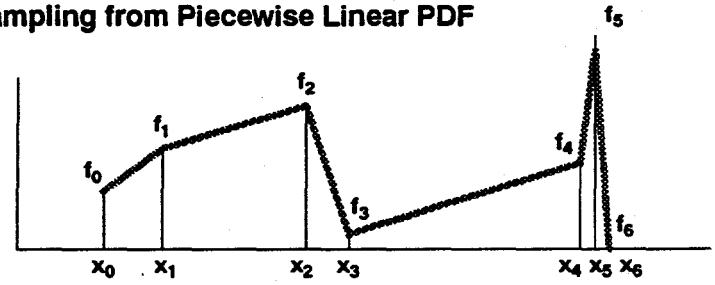
$f(x) = p_1 \cdot$ ◺ $+ \quad p_2 \cdot$ ◿

Sampling scheme:

if $\xi_1 < p_1$, $\quad \hat{x} \leftarrow b - (b-a)\,\sqrt{\xi_2}$

else $\quad \hat{x} \leftarrow a + (b-a)\,\sqrt{\xi_2}$

Examples — Sampling from Piecewise Linear PDF



Two-step sampling: (1) Sample from discrete PDF to select a bin
(2) Sample from linear PDF within bin

- Discrete PDF: $p_k = \dfrac{(f_k + f_{k-1})}{2} \cdot (x_k - x_{k-1})\,, \qquad k = 1, \ldots, N$

  — generate ξ
  — use table search or alias method to select **K**

- Linear sampling within bin **K**:

  — generate ξ
  — then, $\quad$ if $\xi_1 < \dfrac{f_{k-1}}{f_k + f_{k-1}}, \quad \hat{x} \leftarrow x_k - (x_k - x_{k-1})\,\sqrt{\xi_2}$

  otherwise $\quad \hat{x} \leftarrow x_{k-1} + (x_k - x_{k-1})\,\sqrt{\xi_2}$

Examples — Sampling from Exponential PDF

$$f(x) = \lambda e^{-x/\lambda}, \qquad 0 \le x \le \infty$$

$$F(x) = \int_0^x f(x')\,dx' = 1 - e^{-x/\lambda}$$

Direct Sampling:

solving $\xi = 1 - e^{-\hat{x}/\lambda}$ gives: $\hat{x} \leftarrow -\lambda \cdot \ln(1-\xi)$

Although $(1-\xi) \neq \xi$,
both ξ and $(1-\xi)$ are uniformly distributed on (0,1),
so we can use either in the random sampling procedure
(i.e., the *numbers* are different, the *distributions* are the same)

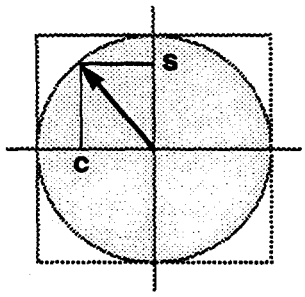
$\hat{x} \leftarrow -\lambda \cdot \ln\xi$

Example — 2D Isotropic

$$f(\hat{\rho}) = \frac{1}{2\pi}, \qquad \hat{\rho} = (u, v)$$

Rejection (*old vim*)

```
      SUBROUTINE AZIRN_VIM( S, C )
      IMPLICIT DOUBLE PRECISION  (A-H, O-Z)
  100 R1=2.*RANF() - 1.
      R1SQ=R1*R1
      R2=RANF()
      R2SQ=R2*R2
      RSQ=R1SQ+R2SQ
      IF(1.-RSQ)100,105,105
  105 S=2.*R1*R2/RSQ
      C=(R2SQ-R1SQ)/RSQ
      RETURN
      END
```



Direct (*racer, new vim*)

```
subroutine azirn_new( s, c )
implicit double precision  (a-h,o-z)
parameter ( twopi = 2.*3.14159265 )
phi = twopi*ranf()
c = cos(phi)
s = sin(phi)
return
end
```

Example — Watt Spectrum

$$f(x) = \frac{2e^{-ab/4}}{\sqrt{\pi a^3 b}} e^{-x/a} \sinh\sqrt{bx}\,, \qquad 0 < x$$

Rejection ***(mcnp)***

- Based on Algorithm **R12** from 3rd Monte Carlo Sampler, Everett & Cashwell
- Define $K = 1 + ab/8$, $L = a\{K + (K^2-1)_{1/2}\}$, $M = L/a - 1$
- Set $x \leftarrow -\log\xi_1$, $y \leftarrow -\log\xi_2$
- If $\{y - M(x+1)\}^2 \le bLx$, accept: return (Lx)
  otherwise, reject

Direct ***(new vim)***

- Sample from Maxwellian in C-of-M, transform to lab

$$w \leftarrow a\left(-\log\xi_1 - \log\xi_2 \cos^2\tfrac{\pi}{2}\xi_3\right)$$

$$x \leftarrow w + \frac{a^2 b}{4} + (2\xi_4 - 1)\sqrt{a^2 bw}$$

(assume isotropic emission from fission fragment moving with constant velocity in C-of-M)

- Unpublished sampling scheme, based on original Watt spectrum derivation

Example — Linear PDF

$$f(x) = 2x\,, \qquad 0 \le x \le 1$$

Rejection

(strictly — this is not "rejection", but has the same flavor)

if $\xi_1 \ge \xi_2$, then $\hat{x} \leftarrow \xi_1$
else $\hat{x} \leftarrow \xi_2$

or

$$\hat{x} \leftarrow \max(\xi_1, \xi_2)$$

or

$$\hat{x} \leftarrow |\xi_1 - \xi_2|$$

Direct

$$F(x) = x^2, \qquad 0 \le x \le 1$$

$$\hat{x} \leftarrow \sqrt{\xi}$$

Probability Density Function	Direct Sampling Method
Linear: $f(x) = 2x, \quad 0 < x < 1$	$x \leftarrow \sqrt{\xi}$
Exponential: $f(x) = e^{-x}, \quad 0 < x$	$x \leftarrow -\log\xi$
2D Isotropic: $f(\vec{\rho}) = \frac{1}{2\pi}, \quad \vec{\rho} = (u, v)$	$u \leftarrow \cos 2\pi\xi_1$ $v \leftarrow \sin 2\pi\xi_1$
3D Isotropic: $f(\vec{\Omega}) = \frac{1}{4\pi}, \quad \vec{\Omega} = (u, v, w)$	$u \leftarrow 2\xi_1 - 1$ $v \leftarrow \sqrt{1-u^2}\cos 2\pi\xi_2$ $w \leftarrow \sqrt{1-u^2}\sin 2\pi\xi_2$
Maxwellian: $f(x) = \frac{2}{T\sqrt{\pi}}\sqrt{\frac{x}{T}}\,e^{-x/T}, \quad 0 < x$	$x \leftarrow T\left(-\log\xi_1 - \log\xi_2\cos^2\frac{\pi}{2}\xi_3\right)$
Watt Spectrum: $f(x) = \frac{2e^{-ab/4}}{\sqrt{\pi a^3 b}}\,e^{-x/a}\sinh\sqrt{bx}, \quad 0 < x$	$w \leftarrow a\left(-\log\xi_1 - \log\xi_2\cos^2\frac{\pi}{2}\xi_3\right)$ $x \leftarrow w + \frac{a^2 b}{4} + (2\xi_4 - 1)\sqrt{a^2 bw}$
Normal: $f(x) = \frac{1}{\sigma\sqrt{2\pi}}\,e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$	$x \leftarrow \mu + \sigma\sqrt{-2\log\xi_1}\cos 2\pi\xi_2$

Machine Considerations

Vector Hardware

- Since ~1980, direct methods have been recommended for vectorization & high performance on ***cray***, ***cyber-205***, ***sx-3***, ***cm-2***, ...

  ➜ Vector concepts apply directly to pipelined RISC cpu's (e.g., ***rs6000***, ***i860***, ***Fujitsu*** ***μ-vp***,...)

Math Libraries

- Many routines in math libraries are now ***table-driven***, hence very fast

  ➜ fast ***sin***, ***cos***, ***sqrt***, ***log***, & ***exp*** functions

RISC + Compiler Technology

- Pipelining, concurrent ops, simple instructions, register-to-register ops, 64-bit hardware, better instruction scheduling, ...

  ➜ fast arithmetic (even for double-precision)

  ➜ Today, the most expensive operations are

  — ***load/store*** (memory access)
  — ***IF...GOTO...*** (flush/fill instruction stack)

Software Considerations

"Rules of thumb" for M.C. algorithm design have changed

- ~~Never take the square root of a random number~~
- ~~Avoid using *sin, cos, log, exp,* .....~~
- ~~Use *IF...GOTO...* to avoid arithmetic~~
- ~~Random numbers are cheap, arithmetic is expensive~~

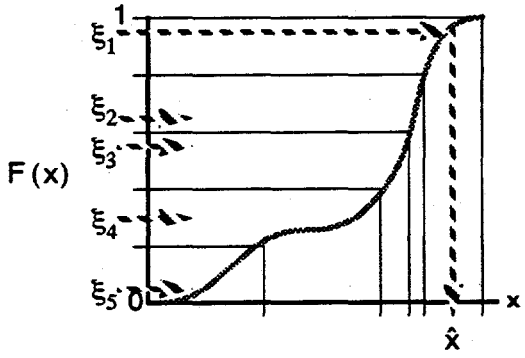
Direct sampling methods have advantages

- Clear, succinct coding — easier to verify & maintain
- *Cpu time is comparable to rejection*
- Direct methods vectorize efficiently

If a specific number of samples, M, is needed from a single distribution:

- Naive approach — repeat the sampling procedure M times

- **Stratified sampling** approach
  - — partition the sample space into M disjoint regions of **equal probability**
  - — produce 1 sample from each region



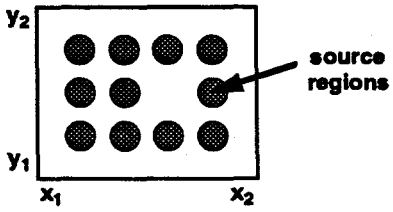
- Stratified sampling considerations
  - — F(x) must be known & easy to partition
  - — The number of partitions, M, must be known in advance
  - — Must be relatively easy to sample **within** each given partition
  - — Stratification improves the "coverage"
  - — Stratified sampling reduces variance, at little or no computing cost

Rejection sampling methods are useful when it is difficult or impossible to invert F(x), or when F(x) is not known

Example — Selection of initial source sites

- select a trial site:

$$\hat{x} \leftarrow x_1 + (x_2 - x_1) \cdot \xi_1$$
$$\hat{y} \leftarrow y_1 + (y_2 - y_1) \cdot \xi_2$$



- if $(\hat{x}, \hat{y})$ is inside shaded region, then **accept** $(\hat{x}, \hat{y})$.
  Otherwise, **reject** $(\hat{x}, \hat{y})$ and repeat.

Rejection methods are generally not used in KAPL Monte Carlo, since they are not well-suited to vectorization.

Exceptions:

- Selection of source sites
  - — every site, for neutrons from fixed-source
  - — sites for initial batch only, for neutrons from fission
- Delta-Tracking

It is sometimes useful to sample from an alternate PDF

$$f(x)\,dx = \left[\frac{f(x)}{g(x)}\right] g(x)\,dx = h(x)\,g(x)\,dx$$

& then "correct" the result via either weight factors or a 2nd sampling stage

Weighted Sampling

- To sample $\hat{x}$ from f(x),
  - — first, sample $\hat{x}$ from g(x)
  - — then, multiply the "weight" assigned to $\hat{x}$ by $\frac{\text{right answer}}{\text{wrong answer}} = \frac{f(\hat{x})}{g(\hat{x})}$
- Note that g(x) must be >0 whenever f(x)>0.
- Also, g(x) must be normalized so that $\int g(x)\,dx = 1$

- **Example — survival biasing of collisions**

  If a collision occurs, $P_{survive} = \frac{\Sigma_S}{\Sigma_T}$ is the probability of surviving.

  Instead of sampling the outcome, always choose survival & multiply the "weight" by $P_{survive}$

- L. Devroye

  Non-Uniform Random Variate Generation, Springer-Verlag, New York (1986)

- C. J. Everett & E. D. Cashwell

  Third Monte Carlo Sampler, LA-9721-MS, Los Alamos National Lab. (1983)

- D. E. Knuth

  The Art of Computer Programming, *Volume 2 — Seminumerical Algorithms,* Addison-Wesley (2nd ed., 1981)

- H. Kahn

  Applications of Monte Carlo, AECU-3259, Rand Corporation (1956)

- J. von Neumann

  Various Techniques Used in Connection with Random Digits, NBS Applied Mathematics Series 12 (1951)

- G. Marsaglia & T. A. Bray

  A Convenient Method for Generating Normal Variables, SIAM Review, 6 (1964)

Weighted Sampling Example — Effective Free-gas Model for Scatter with Bound Hydrogen

- Given a neutron with initial energy E_0, $E_0 > .625$ eV
- For scattering with free hydrogen (target-at-rest), PDF for scatter to E is

$$f_{FREE}(E_0 \to E) = \frac{1}{E_0}, \qquad 0 \le E \le E_0$$

- For scattering with bound hydrogen (free-gas), PDF for scatter to E is

down-scatter: $$f_{BOUND}(E_0 \to E) = \frac{\text{erf}\sqrt{E/kT}}{E_0 - kT/2}, \qquad 0 \le E \le E_0$$

up-scatter: $$f_{BOUND}(E_0 \to E) = \delta(E - E_0), \qquad E > E_0$$

$$P(\text{upscatter}) = \frac{kT}{E_0 + kT/2}$$

- Sampling scheme for $\hat{E}, \hat{\mu}$:

  First, sample $\hat{E}, \hat{\mu}$ using target-at-rest scattering model.
  Then,

  If $\xi < P(\text{upscatter})$, set $\hat{E} \leftarrow E_0$, $\hat{\mu} \leftarrow 1$, then exit

  Otherwise, modify weight by factor

$$\frac{f_{BOUND}(E_0 \to \hat{E})}{f_{FREE}(E_0 \to \hat{E})} = \frac{\text{erf}\sqrt{\hat{E}/kT}}{1 - kT/(2E_0)}$$

  and set $\hat{\mu} \leftarrow \mu_{BOUND}(E_0 \to \hat{E})$

Sampling the free-flight distance, s

- To simulate the free-flight of particles through the problem geometry, need to randomly sample the distance to collision

- PDF for free-flight distance s, along current direction:

$$f(s) = \frac{1}{\Sigma_T(s)} \exp\left(-\int_0^s \Sigma_T(x)\,dx \right)$$

  — If $\Sigma_T(x)$ is constant within a region, the PDF for that region simplifies to $f(s) = \frac{1}{\Sigma_T} \exp(-\Sigma_T \cdot s)$

  — Sampling procedure is then: $\hat{s} \leftarrow -\frac{1}{\Sigma_T} \ln \xi$

- For multiple regions, can stop at each boundary & redetermine $\hat{s}$. Why is this OK?

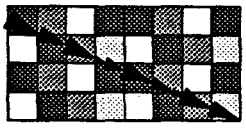
  — Note that probability of traversing distance $\geq s = 1 - \int_0^s \frac{1}{\Sigma} e^{-\Sigma x} dx = e^{-\Sigma s}$

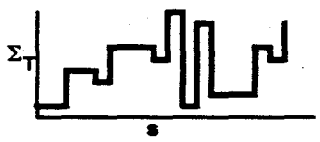
  — For 2 regions, note that

  $e^{-\Sigma x_1} \cdot e^{-\Sigma x_2} = e^{-\Sigma (x_1 + x_2)}$ = probability of traversing both regions

"Regular" Tracking

- Move particles through one region at a time, until collision occurs
- Can be expensive if many regions must be traversed before collision





$$f(s) = \frac{1}{\Sigma_T(s)} \exp\left(-\int_0^s \Sigma_T(x)\,dx \right), \qquad s = \text{distance along flight path}$$

- **"Regular" tracking procedure**, when Σ_T constant within each region:

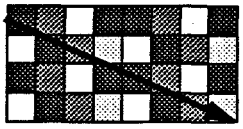
  — Sample a flight distance $\hat{s}$ using Σ_T for current region: $\hat{s} \leftarrow -\frac{1}{\Sigma_T} \ln \xi$

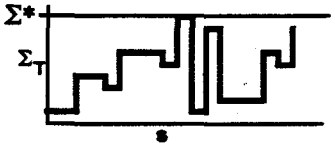
  — If $\hat{s} < d_{boundary}$, move the particle by $\hat{s}$ along current direction, then exit & analyze the collision

  — Otherwise, move the particle by $d_{boundary}$ along current direction, repeat until collision

Delta Tracking

- A type of rejection method for sampling the free-flight distance
- Also called Woodcock tracking, fast tracking, or hole tracking
- Useful when Σ_T varies rapidly over the flight path





$$f(s) = \frac{1}{\Sigma^*}\exp(-\Sigma^* \cdot s), \qquad s = \text{distance along flight path}$$

- For delta tracking, a fictitious cross-section Σ^* is used, rather than $\Sigma_T(s)$
  - — Σ^* should be chosen to be $\geq \Sigma_T(s)$ for all possible points along path
  - — Σ^* may be a function of particle energy, or not
  - — $\Sigma^* = \Sigma_T(s) + \Sigma_\delta(s) = \text{constant}, \qquad \Sigma_\delta(s) \geq 0$ for all s>0

    where $\Sigma_\delta(s)$ = cross-section for "delta-scattering", i.e., *scatter with no change in energy or direction*, a fictitious scattering event, or "pseudo-collision"

Delta Tracking procedure:

- $\Sigma^* = \Sigma_T(s) + \Sigma_\delta(s) = \text{constant}$
- $\Sigma^* \geq \Sigma_T(s)$ and $\Sigma_\delta(s) \geq 0$ for all s>0

- To sample the distance to collision, $\hat{s}$, using delta tracking, play the following rejection game:
  - — set $\hat{s} \leftarrow 0$
  - — repeat the following steps until a (real) collision occurs:
    - sample a flight distance $\hat{x}$ using Σ^*: $\hat{x} \leftarrow -\frac{1}{\Sigma^*}\ln\xi$
    - $\hat{s} \leftarrow \hat{s} + \hat{x}$
    - with probability $\frac{\Sigma_\delta(\hat{s})}{\Sigma^*}$, reject the collision site & repeat.

      otherwise, accept $\hat{s}$ & analyze the collision

Delta tracking can be effective if at least some of the following are true:

- The cost of locating a particle position in the problem geometry is small, relative to the cost of computing many distances
- Σ^* is not too different from the "average" $\Sigma_T(s)$
- $1/\Sigma^*$ is large compared to distances between geometric boundaries

➜ All of these considerations are (usually) true for fast neutrons in grid regions

Delta tracking can be ineffective if a few small regions have very large Σ_T

- Results in $\Sigma^* \gg \Sigma_T(s)$ for most regions, so that efficiency of rejection sampling is low

If the cross-sections are continuously-varying functions of position, then delta-tracking may be the only feasible method for sampling s

- Same delta-tracking procedure is used if cross-sections vary continuously with position &/or energy.
- Can even be used when the cross-sections are not known in advance, so long as the maximum cross-section can be determined.

Proof: **Delta-Tracking is an unbiased method for sampling the free-flight distance**

Consider the probability of traversing a distance s along the flight path without undergoing a (real) collision, P(s).

- $\Sigma^* = \Sigma_T(s) + \Sigma_\delta(s) = \text{constant}, \quad \Sigma^* \geq \Sigma_T(s)$ and $\Sigma_\delta(s) \geq 0$ for all s>0
- For convenience, define optical thicknesses for real scatter & delta scatter:

$$\tau(s) = \int_0^s \Sigma_T(x)\,dx \qquad \tau_\delta(s) = \int_0^s \Sigma_\delta(x)\,dx$$

Note that, by definition, $\Sigma^* s = \tau(s) + \tau_\delta(s)$ and $\Sigma^* s \geq \tau(s)$

- For a particular flight, there could be exactly 0, 1, 2, ..., ∞ pseudo-collisions before a real collision occurs.

• Let $P(s \mid n)$ = probability of traversing distance **s** along the flight path with <u>exactly</u> **n** pseudo-collisions

Then, $P(s) = \sum_{n=0}^{\infty} P(s|n)$

$$P(s|0) = e^{-\Sigma^* s}$$

$$P(s|1) = \int_0^s P(x|0)\,\Sigma_\delta(x)\,P(s-x|0)\,dx = \int_0^s e^{-\Sigma^* x}\Sigma_\delta(x)\,e^{-\Sigma^*[s-x]}\,dx = \tau_\delta(s)\,e^{-\Sigma^* s}$$

$$P(s|2) = \int_0^s P(x|1)\,\Sigma_\delta(x)\,P(s-x|0)\,dx = \int_0^s \tau_\delta(x)\,e^{-\Sigma^* x}\Sigma_\delta(x)\,e^{-\Sigma^*[s-x]}\,dx$$

$$= \int_0^s \tau_\delta(x)\,\Sigma_\delta(x)\,e^{-\Sigma^* s}\,dx = \frac{[\tau_\delta(s)]^2}{2} e^{-\Sigma^* s}$$

$$P(s|n) = \int_0^s P(x|n-1)\,\Sigma_\delta(x)\,P(s-x|0)\,dx = \frac{[\tau_\delta(s)]^n}{n!} e^{-\Sigma^* s}$$

• Then, the total probability of traversing a distance **s** without undergoing a (real) collision is

$$P(s) = \sum_{n=0}^{\infty} P(s|n) = \sum_{n=0}^{\infty} \frac{[\tau_\delta(s)]^n}{n!} e^{-\Sigma^* s} = e^{\tau_\delta(s)} e^{-\Sigma^* s}$$

$$= \exp(\tau_\delta(s) - \Sigma^* s) = \exp[-\tau(s)] = \exp\left(-\int_0^s \Sigma_T(x)\,dx\right)$$

➜ **This is the correct result, identical to the "Regular Tracking" procedure.**

Combined Russian Rouletting & Splitting

- Russian Roulette — kill off some particles, but conserve total weight
  — to save computing time, roulette "unimportant" particles
- Splitting — create extra identical particles, but conserve total weight
  — to reduce variance, split particles if weights "too large"
- Definitions

  wgt = Particle weight

  For the region containing the particle:

  w_{high} = upper bound on weight, if wgt larger — split

  w_{low} = lower bound on weight, if wgt lower — roulette

  w_{ave} = weight to assign survivors, $w_{low} < w_{ave} < w_{high}$

  Then,

  wgt / w_{ave} = probability of surviving split/roulette

  For each region, choose w_{ave} based on region "importance" (using adjoint function, if known). Choose w_{low} & w_{high} 2-5 times lower or higher.

- Combined game for split/roulette:

  if $wgt < w_{low}$ or $w_{high} < wgt$,

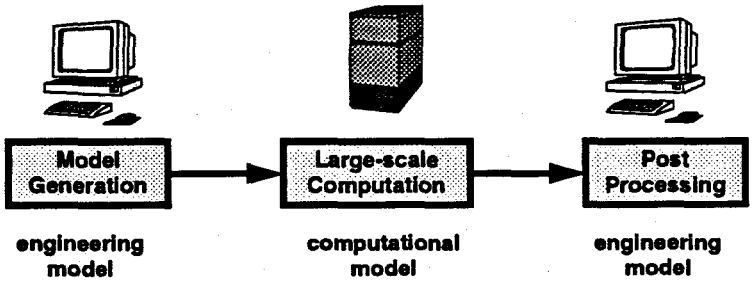
  create n particles of weight w_{ave}, where $n \leftarrow \left\lfloor \frac{wgt}{w_{ave}} + \xi \right\rfloor$

Random Sampling in RACER

- Source
  - — Fixed sites — uniform PDF + rejection
  - — Fission sites — discrete PDF + stratified sampling
  - — Energy — piecewise-linear PDF (binary table search + linear PDF)
  - — Direction — isotropic 3D PDF
- Tracking
  - — free-flight distance — exponential PDF
  - — delta-tracking — rejection sampling of pseudo- & real collisions
- Russian Roulette & Splitting
  - — discrete PDF + weights
- Collisions
  - — Survival biasing — weights
  - — Select phase & nuclide — discrete PDF (on-the-fly)
  - — Epithermal
    - • Scattering angle — equally-probable histograms (uniform discrete PDF + uniform in bin)
    - • Inelastic: energy — discrete PDF (aliased), then uniform within group
      n2n — weights
    - • modified free-gas — weights
  - — Thermal
    - • multigroup — discrete PDF for group-to-group (aliased), linear PDF for μ
    - • S(α,β) — discrete PDF (aliased) & uniform PDF sampling
  - — Direction — polar angle from uniform PDF
  - — Fission bank — discrete PDF + weights

Computational Geometry

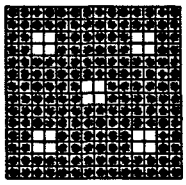
Engineering Model vs. Computational Model



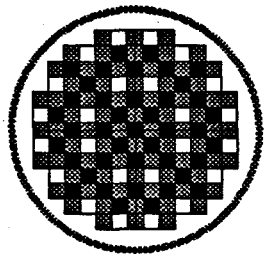
- **Model Generation**
  - — focus on engineering productivity
  - — describes "reality" to computer
  - — interactive, batch, or CAD
- **Large-scale Computation**
  - — focus on efficiency & capabilities
  - — data structure should be compact & regular
  - — computational model often hidden from user
  - — best reference: source coding
- **Post-Processing**
  - — interpretation of results
  - — visualization

- **Element geometry**

- **Elements → Assemblies**



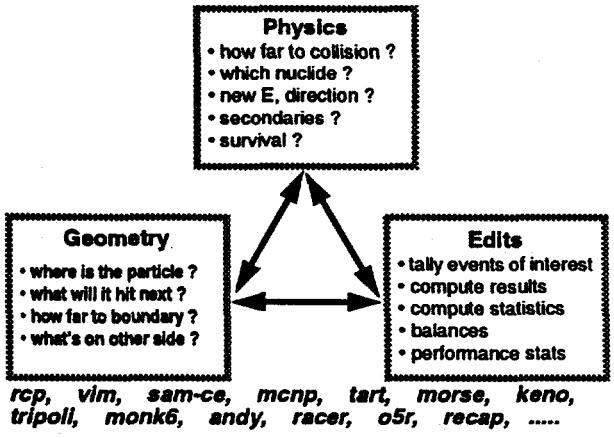
- **Assemblies → Core**



- **Core + peripherals → 3-D Model**

Model Construction

Geometry Computation



rcp, vim, sam-ce, mcnp, tart, morse, keno, tripoli, monk6, andy, racer, o5r, recap, ...

Development of particular capabilities is driven by applications:

- **Shielding & experiment analysis**
  - — irregular geometry
  - — moderate number of regions
  - — few compositions
  - ⇒ ***focus:*** ***convenient, flexible input for arbitrary geometry***
- **Reactor core analysis**
  - — regular geometry
  - — many regions (up to ~10^8)
  - — many compositions (up to ~10^5)
  - ⇒ ***focus:*** ***automated processing for repeating, detailed geometry***

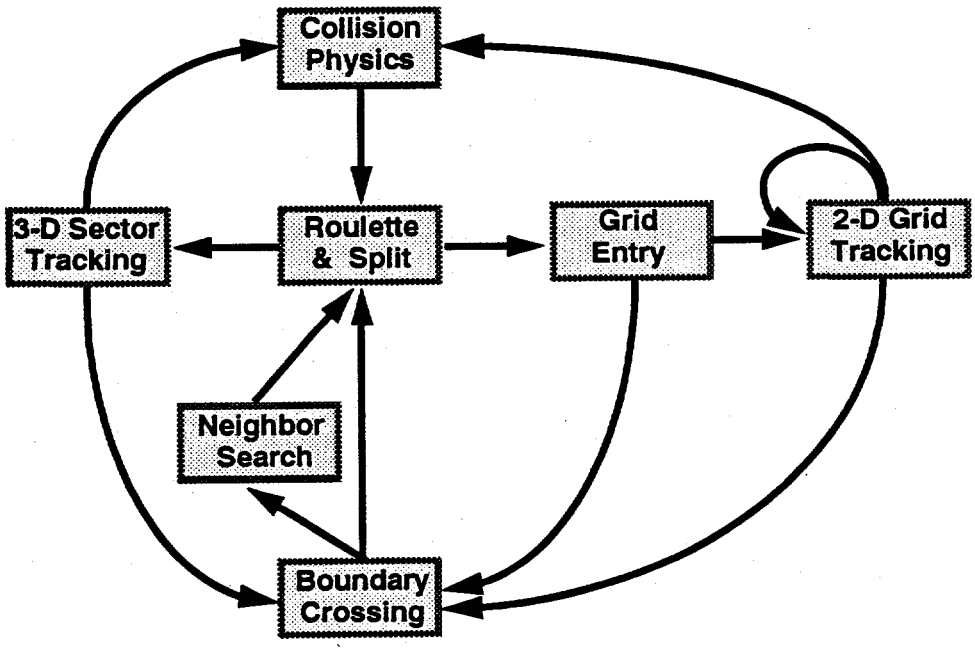
RACER Computational Algorithm — Geometric View

repeat for all **batches**
. repeat for all cross-section **supergroups**
. . repeat until **neutrons** are gone
. . . repeat until **collision**
. . . . repeat for **surfaces** of 3-D region
. . . . . • **distance calculation**
. . . .
. . . . repeat while in **grid**
. . . . . • **distance calculation** ***
. . . . . repeat for **figures** in grid region
. . . . . . repeat for **quadrics** in **figure**
. . . . • **distance calculation** ***
. . . .
. . . . • boundary cross
. . . . • neighbor search
. . . . • roulette
. . . ...
. . . • collide
. . . • roulette
... ...

1 reactor calculation
⇓
~10^9 distance calculations

*** if delta-tracking, "locate" only

RACER Computational Events



RACER — General Geometry Capabilities

3-D general geometry

- allowed **surface** types:

  general planes, sphere, right circular cylinder,
  right elliptic cylinder, ellipsoid, skewed cylinders,
  general 3-D quadratic

- 3-D regions (**sectors**) defined by lists of **surfaces** (with sense & BC)
- Sectors may be combined into **edit-media**
- Neighbor lists built dynamically

Sectors may have embedded detailed grids

- lattice (need not be rectangular)
- general rotation/translation/scaling to/from 3-D geometry
- lattice "boxes" may contain arbitrary sets of quadratics
- detailed lattice regions may be assigned to arbitrary **edit-media**

Delta-tracking or surface-to-surface tracking, variable by energy

- Delta-tracking — also called "Woodcock tracking" or "hole geometry"

large problem example: > 5000 3-D sectors, $>10^7$ detailed regions

RACER 3D Geometry

Surface

- Linear or quadratic polynomial, in absolute coordinates (x,y,z)

$$F(x,y,z) = ax^2 + by^2 + cz^2 + dxy + eyz + fzx + gx + hy + iz + j$$

  — Normalization is arbitrary

  — RACER Convention: factor of leading 2^{nd} order term is positive, usually = 1.0

  — Surface is defined by: $F(x,y,z) = 0$

- Surfaces are infinite in extent

- **Sense:**

  — relationship between a point in space (x_0,y_0,z_0) & a surface $F(x,y,z)$

  — point (x_0,y_0,z_0) is **inside** if $F(x_0,y_0,z_0) < 0$
  outside if $F(x_0,y_0,z_0) > 0$
  on if $F(x_0,y_0,z_0) = 0$

RACER Surface Equations

Type	Description	Equation: F(x,y,z) = 0
1	general plane	$ax + by + cz + d = 0$
2	right cylinder	$a(x+b)^2 + (y+c)^2 + d = 0$
3	cylinder skewed in Z	$a(x+b+cz)^2 + (y+d+ez)^2 + f = 0$
4	sphere	$(x+a)^2 + (y+b)^2 + (z+c)^2 + d = 0$
5	ellipsoid	$a(x+b)^2 + c(y+d)^2 + e(z+f)^2 + g = 0$
6	general 3-D quadric	$ax^2 + by^2 + cz^2 + dxy + eyz + fzx + gx + hy + iz + j = 0$
7	pair of z-independent planes	$(x+by+c)^2 - d^2 = 0$
11	x=constant plane	$x + d = 0$
12	y=constant plane	$y + d = 0$
13	z=constant plane	$z + d = 0$
14	z-independent plane	$x + by + d = 0$
21	pair of planes parallel to YZ	$(x+c)^2 - d^2 = 0$
22	pair of planes parallel to XZ	$(y+c)^2 - d^2 = 0$

RACER 3D Geometry

Side

- half-space, defined by **signed surface number**

  e.g., +7 ➜ { (x,y,z) | $F_7(x,y,z)>0$ }

Sector

- 3-D region, **intersection of sides**
  - — note: sectors are defined only by the intersection of sides; unions of sides are **not** allowed
- defined by list of sides

  e.g., Sector 3 ==> -1, +4, +7, -523, -734

- **Sense**:
  - — relationship between a **point** in space (x_0,y_0,z_0) & a **sector**
  - — point (x_0,y_0,z_0) is **inside** / **outside** if all surface-senses match, otherwise
- attributes:
  - edit medium number
  - importance region number
  - boundary conditions for each surface (R,E,P,N)
  - embedded geometry (grid) — optional

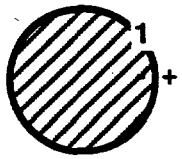
Sector Geometry — Examples

Example 1

Surface 1 : sphere

Sector 1 : - 1

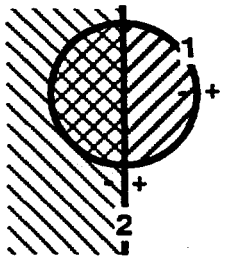
sense — surface #



Example 2

Surface 1 : sphere
Surface 2 : plane

Sector 1 : - 1, - 2



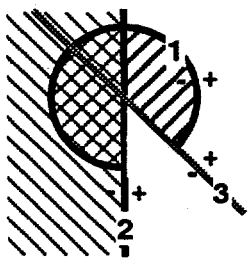
Example 3

Surface 1 : sphere
Surface 2 : plane
Surface 3 : plane

Sector 1 : - 1, - 2
Sector 2 : - 1, + 2, + 3

Edit-medium 1: 1, 2

sectors



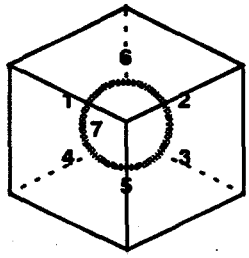
Sector Geometry Example: Box with a Hole

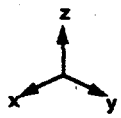
- **Define surfaces:**

planes: 1, 2, 3, 4 (sides)
5, 6 (top, bot)

sphere: 7

(problem origin at lower rear)





- **Define sectors**

Sector 1: inside the sphere -7

Sector 2: inside the box, but outside the sphere +7, + 1, +2, -3, -4, +5, -6

- **Assign properties to each sector**

— define the **boundary conditions** for each surface of the sector

— assign an **edit-medium number** to the sector
— assign a composition (material) to the edit-medium

— assign an **importance region number** to the sector

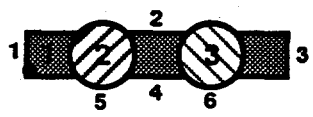
Sector Geometry — Miscellaneous

- Sectors do **not** have to be convex regions

  Sector 1 : +1, -2, -3, +4, +5, +6

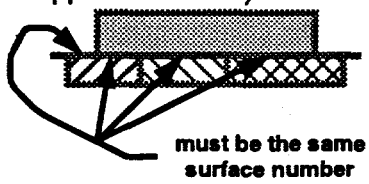
  Sector 2 : -5

  Sector 3 : -6



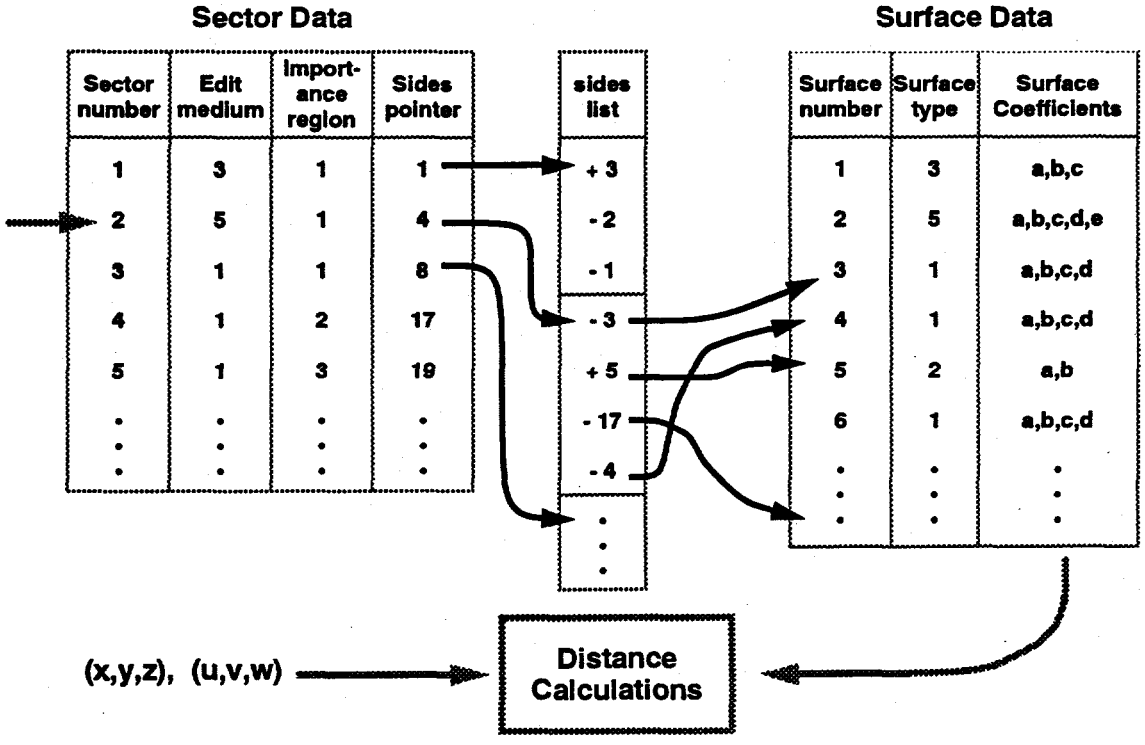
- Sectors may be infinite in extent
  - — For 2-D problems, sectors do not need a "top" & "bottom"

- Adjoining sectors **must** share a common surface (with opposite senses)
  - — No gaps or overlap permitted
  - — "Similar" surfaces must be combined



- Sectors cannot be defined as unions of sides
  - — **Not** a valid sector: 
  - — Note: can do this by combining sectors into edit-media

Sector Geometry — Lists, Logic, Lookups



Distance Calculations in 3D Geometry

3D Surface: $F(x, y, z) = 0$

"linear" ➔ ∇F = constant

"quadratic" ➔ $\nabla F = f(x, y, z)$, $\nabla^2 F$ = constant

Distance Calculation

- s = directed distance from (x_0,y_0,z_0) along (u,v,w) to F(x,y,z)

  = smallest positive root of $F(x_0 + su, y_0 + sv, z_0 + sw) = 0$

- recast as

$$G(0) + \int_0^s \frac{\partial G}{\partial s'} ds' = 0, \qquad \text{where } G(s) = F(x_0 + su, y_0 + sv, z_0 + sw)$$

- then,

$$\left[\frac{1}{2}\frac{\partial^2}{\partial s^2} G(0)\right] s^2 + \left[\frac{\partial}{\partial s} G(0)\right] s + [G(0)] = 0$$

- Or,

$$As^2 + 2Bs + C = 0, \qquad D = B^2 - AC$$

[continued]

Distance Calculations in 3D Geometry [continued]

- $As^2 + 2Bs + C = 0, \qquad D = B^2 - AC$
- 27 combinations of A, B, C >0, <0, =0
- only 12 yield valid solutions:

$$s = -\frac{C}{2B} \quad \text{if} \quad (A=0, C<0, B>0) \quad \text{or} \quad (A=0, C>0, B<0)$$

$$= \frac{-B-\sqrt{D}}{A} \quad \text{if} \quad (A>0, C>0, B<0, D>0) \quad \text{or} \quad (A>0, C<0, B>0, D>0)$$
$$\text{or} \quad (A<0, C>0, B<0, D>0) \quad \text{or} \quad (A<0, C>0, B=0, D>0)$$

$$= \frac{-B+\sqrt{D}}{A} \quad \text{if} \quad (A>0, C<0, B>0, D>0) \quad \text{or} \quad (A>0, C<0, B<0, D>0)$$
$$\text{or} \quad (A>0, C<0, B=0, D>0) \quad \text{or} \quad (A<0, C<0, B>0, D>0)$$
$$\text{or} \quad (A>0, C=0, B<0, D>0) \quad \text{or} \quad (A<0, C=0, B>0, D>0)$$

$$= \infty \quad \text{otherwise}$$

- Use known surface-sense, §, to solve roundoff problems & eliminate tolerance check

$$s' = -\frac{C}{2B} \quad \text{if} \quad (A=0, D>0)$$

$$= \frac{-B-\sqrt{D}}{A} \quad \text{if} \quad (A\neq 0, D>0, \S>0)$$

$$= \frac{-B+\sqrt{D}}{A} \quad \text{if} \quad (A\neq 0, D>0, \S<0)$$

$$= \infty \quad \text{otherwise}$$

$$s = s' \quad \text{if } s'>0$$
$$= \infty \quad \text{otherwise}$$

- Distance to Sector Boundary = minimum s for all surfaces of sector

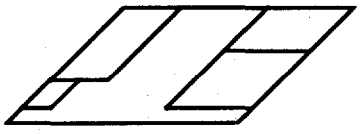
Sector Tracking — outline of calculation

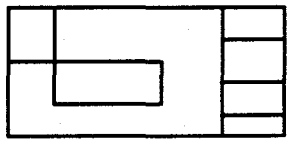
```
IND   ← pull from event-stack                    • get neutron pointers
ISEC  ← ISECSTK( INDX )                          • get sector from neutron-stack

DSECSTK( INDX ) ← infinity                       • initialize distance results
LSECSTK( INDX ) ← 0                              • initialize pointer results

LS1   ← LSID1SEC( ISEC )                         • pointers to 1st side of sectors
LS2   ← LSID2SEC( ISEC )                         • pointers to last side of sectors

Do While any LS1 < LS2
.                                                • SELECT neutrons with more sides
.     INDK ← INDX(    LS1<LS2  )                 • get neutron pointers
.     ISURK← ISURSEC( LS1( LS1<LS2 ) )           • get signed surface numbers
.
.     For Each Surface Type J
.     .                                          • SELECT neutrons with surface type
.     .     INDJ      ← INDK( TYPE(ISURK)=J )    • get neutron pointers
.     .     ISURJ     ← ISURK( TYPE(ISURK)=J )   • get signed surface numbers
.     .
.     .                                          • particular surface type
.     .     X,Y,Z     ← stk( INDJ )              • GATHER neutron position
.     .     U,V,W     ← stk( INDJ )              • GATHER neutron direction
.     .     A,B,...   ← coef( ISURJ )            • GATHER equation coefficients
.     .     DISTJ     ← distance calc            • compute distance, vector
.     .
.     .     DISTK( INDJ ) ← DISTJ                • save distance, temporarily
.     ...
.     where DISTK < DSECSTK( INDK )              • smallest distance, so far ?
.     .     DSECSTK( INDK ) ← DISTK              • SCATTER distance
.     .     LSECSTK( INDK ) ← LS1                • SCATTER pointer
.     ...
.     LS1   ← LS1 + 1                                  • bump pointers to surfaces
...
```


RACER Embedded Lattice Geometry

Grid:

- **embedded within 3-D sector**
- detailed 2-D geometry extruded, **axially uniform**
- 2-D mesh of parallel lines, need not be rectangular





- grid lines are clipped at grid boundaries
- local coordinates — origin at lower left corner
- general transformation to/from 3-D: rotate, reflect, scale, translate
- grid box may contain composition or embedded figures
- very compact storage, much faster tracking

Figure:

- arbitrary types and numbers of 2-D quadrics
- constraint: **quadrics do not intersect within grid box**
- local coordinates
- translate to/from grid coordinates
- clipped at grid box boundaries

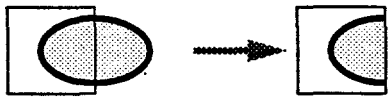
Quadrics:

- general 2nd order surfaces, F(x,y)
- local coordinates
- clipped at grid box boundaries

RACER Grid Geometry

- Within a grid box, quadric surfaces can be nested. Inner surfaces overlay outer ones.

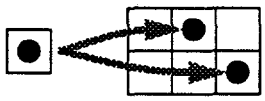
- Quadrics are clipped at grid box boundaries



- Quadrics cannot intersect within a grid box. Extra grid lines must be inserted at intersections.

- Lines which are not parallel to the grid must be represented by infinite ellipses

- A single figure can be repeated via translation to different grid boxes (no rotation)

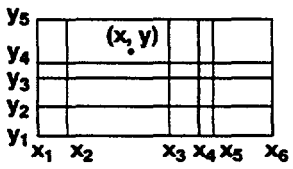


Location & Distances within Grids

- **Location — determine grid-box (i,j)**

  i : binary search to find x-interval containing x

  j : binary search to find y-interval containing y

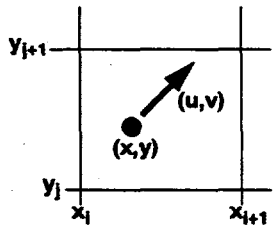


- **Distance to boundary of grid-box**

  — use signs of (u,v) to select the next x-bound & y-bound:

  if u<0, $\bar{x} \leftarrow x_i$, otherwise $\bar{x} \leftarrow x_{i+1}$

  if v<0, $\bar{y} \leftarrow y_j$, otherwise $\bar{y} \leftarrow y_{j+1}$



  — compute x-distance & y-distance

$$d_x = \frac{\bar{x} - x}{u} \qquad d_y = \frac{\bar{y} - y}{v}$$

  — if grid-box contains quadrics, compute q-distance, d_q

  — distance to next grid-boundary = $d_{grid} = \min(\, d_x,\, d_y,\, d_q \,)$

  — $d = \min(\, d_{sector},\, d_{grid} \,)$

Grid Tracking

Grid Entry:
- transform (x,y,z) and (u,v,w) from 3-D to local
- (u,v,w) *not normalized*

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix}_{Grid} = \begin{bmatrix} g_1 & g_2 & g_3 \\ g_5 & g_6 & g_7 \\ g_9 & g_{10} & g_{11} \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}_{Sector} + \begin{bmatrix} g_4 \\ g_8 \\ g_{12} \end{bmatrix} \qquad \begin{bmatrix} u \\ v \\ w \end{bmatrix}_{Grid} = \begin{bmatrix} g_1 & g_2 & g_3 \\ g_5 & g_6 & g_7 \\ g_9 & g_{10} & g_{11} \end{bmatrix} \begin{bmatrix} u \\ v \\ w \end{bmatrix}_{Sector}$$

Grid Locate:
- binary searches of x & y grid-line intercepts

Grid Distance:
- select x & y grid-lines using signs of u & v
- simple distance calculation (if **not** using delta-tracking)

Figures:
- for each figure, check all quadrics

Quadrics:
- translate x & y
- compute distance (if **not** using delta-tracking) & sense

Lookups:
- edit-medium, composition

Grid Exit:
- transform back to 3-D, if collision or grid boundary cross

RACER Boundary Conditions — for Each Surface of Sector

Normal:
- continue neutron flight, no action at boundary

Exit:
- leakage out of problem, tally & kill

Reflect:
- Plane— $ax + by + cz + d = 0$, with $a^2 + b^2 + c^2 = 1$

$$\begin{bmatrix} u \\ v \\ w \end{bmatrix}_{New} = \begin{bmatrix} u \\ v \\ w \end{bmatrix}_{Old} - 2(au + bv + cw) \cdot \begin{bmatrix} a \\ b \\ c \end{bmatrix}$$

- Cylinder— $a(x+b)^2 + (y+c)^2 + d = 0$

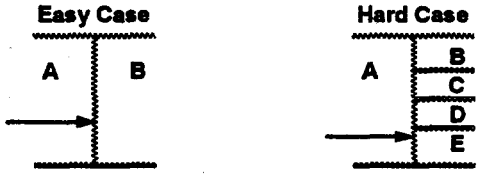
$$\begin{bmatrix} u \\ v \\ w \end{bmatrix}_{New} = \begin{bmatrix} u \\ v \\ w \end{bmatrix}_{Old} - 2\frac{(ux + vy)}{|d|} \cdot \begin{bmatrix} x \\ y \\ 0 \end{bmatrix}$$

Periodic:
- rotate about point $(x_p, y_p, 0)$ by angle θ, with $C_p = \cos\theta$, $S_p = \sin\theta$

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix}_{New} = \begin{bmatrix} C_p & -S_p & 0 \\ S_p & C_p & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x - x_p \\ y - y_p \\ z \end{bmatrix}_{Old} + \begin{bmatrix} x_p \\ y_p \\ 0 \end{bmatrix} \qquad \begin{bmatrix} u \\ v \\ w \end{bmatrix}_{New} = \begin{bmatrix} C_p & -S_p & 0 \\ S_p & C_p & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ w \end{bmatrix}_{Old}$$

Neighbor Search

- When a region boundary is reached, what's on the other side ?



- Most codes build "neighbor lists" during tracking
  - — for each boundary surface of region, **remember** list of neighbors
  - — initially, neighbor lists are empty
  - — check regions having surface in common until one is found satisfying all sense conditions
  - — save it
  - — later, check neighbor lists first, if necessary do search
- Neighbor search is expensive at first, cheap later
- Tracking speeds up as calculation progresses

Miscellaneous Topics

- Axially varying compositions
  - — "axcomps", permuted compositions
- "Strong" sense checking & distance calculations
- "Lost" particles
- Reference points
- Source volumes
- 3-D grid for "site-plot" & fission-matrix
- Importance regions
- "Slice" plots

3-D Geometry Schemes — Variations

Combinatorial Geometry (e.g., MAGI, MORSE)

- pre-defined primitives ("bodies" -- box, sphere, cone, ...)
- Boolean operations on primitives
  - — intersection, union, complement operations on bodies

Constructive Solid Geometry (e.g., CSG algorithms for graphics)

- pre-defined primitives (box, sphere, ...)
- general Boolean operations
  - — intersection, union, complement operations on primitives & objects constructed from primitives

MCNP Geometry

- *user-defined primitives ("cells" -- list of signed surfaces)*
- Boolean operations
  - — intersection & union of surface half-spaces
  - — complement of another cell

Sector Geometry (e.g., O5R, RACER)

- user-defined primitives ("sectors" -- list of signed surfaces)
- Boolean operations
  - — intersection (only) of surface half-spaces
  - — unions of sectors achieved via "edit-media"

Hole Geometry (e.g., MONK, RACER)

- anything & everything imaginable (MONK6)

3-D Geometry Scheme Comparison

	Combinatorial Geometry	RACER 3D Geometry
User-Input Level	• define BODIES rect. parallelepiped,box, sphere,right circ. cyl., right ellipt. cyl.,ellipsoid, trunc. cone, wedge, arbitrary convex polyhedron	• define SURFACES general planes, sphere,right circ. cyl., right ellipt. cyl., ellipsoid, skewed cylinders, general 3-D quadratic
	• define INPUT ZONES composition number importance region number list of BODIES AND, NOT, OR operators	• define SECTORS (AND,NOT) list of SURFACES (with sense & BC) importance region number • define EDIT-MEDIA composition number list of SECTORS (OR operator) list of regions in embedded GRIDS
Code Level	• construct CODE ZONES list of BODIES AND, NOT operators only composition number importance region number	
	• detailed calculations distance to each SURFACE of each BODY in CODE ZONE	• detailed calculations distance to each SURFACE of SECTOR
	• SURFACE definitions implicit, inferred from BODY input	• BODY definitions are implicit, inferred from SECTOR/SURFACE input
Comments	• **"input details" are different** • ***detailed calculations are essentially equivalent***	

Collision

Physics

Epithermal Energy Range

- continuous energy neutrons & cross sections
- elastic scattering
  - assume free nucleus at rest ($E >> k_BT$)
  - conserves kinetic energy and momentum of neutron + nucleus
- scattering from H bound in H_2O
  - accounts for chemical binding
- inelastic scattering
  - forms compound nucleus, kinetic energy is not conserved
  - multigroup treatment

→ no upscattering allowed in epithermal range in RACER

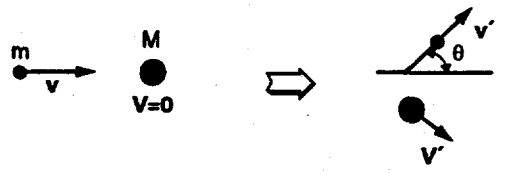
Thermal Energy Range - Multigroup Method

- multigroup neutrons & cross sections
- only collisions with 'moderators' result in an energy change
  - upscattering and downscattering
  - P_1 angular distribution
- scattering from non-moderators is isotropic in the LAB system

Thermal Energy Range - Continuous Energy Method

- continuous energy neutrons & cross sections
- scattering handled explicitly for each type
  - coherent elastic (Bragg)
  - incoherent inelastic ($S(\alpha, \beta)$)
  - incoherent elastic

Elastic Scattering From a Free Nucleus at Rest



pre-collision post-collision

LAB System

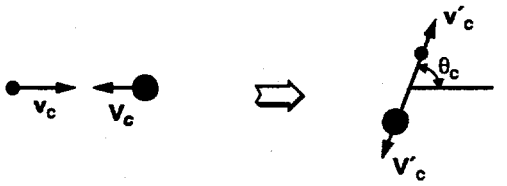
- scattering angle θ_c is sampled in the COM system
- azimuthal angle φ is uniformly sampled between 0 and 2π
- exit energy E' is determined by

$$E' = E\frac{A^2 + 2A\mu_c + 1}{(A+1)^2}$$

$$A = M/m,\ \mu_c = \cos\theta_c$$

- LAB scattering angle is determined by

$$\mu = \frac{1 + A\mu_c}{\sqrt{A^2 + 2A\mu_c + 1}}$$



pre-collision post-collision

COM System

Legendre Expansion of the Scattering Cosine Distribution

$\sigma(\mu_c)$ = cross section dependence on the cosine of the scattering angle in the COM system

$f(\mu_c) = \sigma(\mu_c)/\sigma$ = scattering cosine pdf, where $\sigma = \int_{-1}^{1} d\mu_c \sigma(\mu_c)$

$f_n = \int_{-1}^{1} d\mu_c P_n(\mu_c) f(\mu_c)$ = n^{th} Legendre moment

→ Legendre moments are part of the basic cross section data.

Given the moments, the pdf can be reconstructed, i.e.

$$f(\mu_c) = \sum_{n=0}^{\infty} \frac{2n+1}{2} f_n P_n(\mu_c).$$

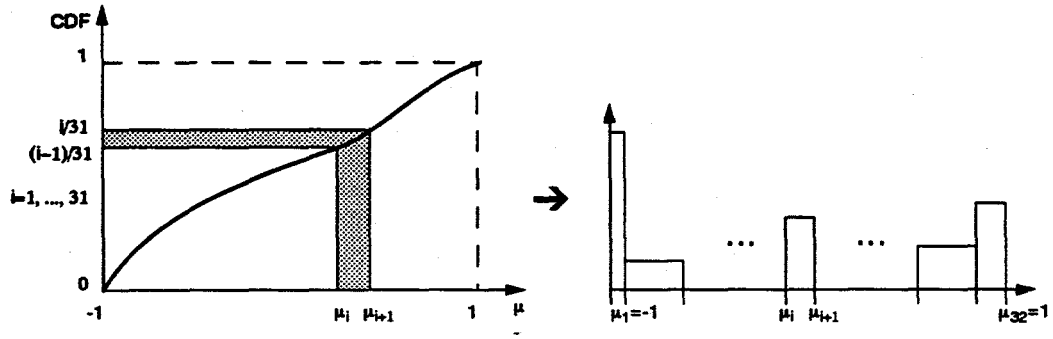
In practice, only a finite number of moments are known, so that

$$f(\mu_c) \approx \sum_{n=0}^{N} \frac{2n+1}{2} f_n P_n(\mu_c).$$

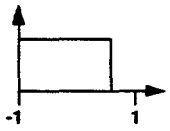
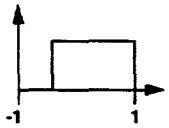
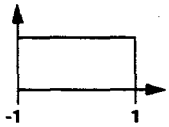
→ RACER can utilize Legendre moments up to the 20^{th}.

Selection of the Center-of-Mass Scattering Cosine

- generate up to (typically) 31 equally-probable step functions
- randomly select one of the 31 angular 'bins'
- uniformly sample scattering cosine between μ_i and μ_{i+1}



Special cases:

P_1:  or  Isotropic: 

Epithermal Scattering From Bound Hydrogen

- for scattering from H in H_2O, the effect of the H-O bond must be accounted for
- Nelkin water model

— the proton is treated as a sum of 5 harmonic oscillators
- one translational mode
- one rotational mode
- three vibrational modes

— the effective temperature is thus given by

$$k_B T_{eff} = \frac{k_B T}{18} + 0.0824 + 0.0176 \coth\left(\frac{0.1025}{k_B T}\right) + 0.0129 \coth\left(\frac{0.03}{k_B T}\right)$$

- Cady prescription

— PDF for exit energy

$$f_{BOUND}(E \rightarrow E') = \delta(E - E'), \qquad E' > E$$

$$f_{BOUND}(E \rightarrow E') = \frac{\mathrm{erf}\left(\sqrt{E'/k_B T_{eff}}\right)}{E - k_B T_{eff}/2}, \qquad E' \le E$$

$$P[E' > E] = \frac{k_B T_{eff}}{E + k_B T_{eff}/2}$$

→ implemented in RACER using weight modification

— scattering cosine in LAB system is set to average value for bound H downscattering

$$\mu \leftarrow \mu_{BOUND} = \mu_{FREE}\left[1 - \frac{k_B T_{eff}}{E'} + 2\sqrt{\frac{k_B T_{eff}}{\pi E'}} \frac{\exp(-E'/k_B T_{eff})}{\mathrm{erf}\left(\sqrt{E'/k_B T_{eff}}\right)}\right]$$

— scattering cosine in LAB system is set to 1 for upscattering

Epithermal Inelastic Scattering

- based on top 25 MUFT groups (need not correspond to supergroups)
- alias sampling from discrete PDF to get exit group ($P_{g \to g'}$, $g' \geq g$)
- if $g' \neq g$, exit energy E' is uniformly sampled between E^g_{bot} and E^g_{top}
- if $g' = g$, E' is uniformly sampled between incident energy E and E^g_{bot}
- history weight is changed to account for (n,2n) reactions
- isotropic in LAB or COM systems (user selectable)

Grp. #	E_{top} (eV)	E_{bot} (eV)
1	20,000,000	7,788,000
2	7,788,000	6,065,000
3	6,065,000	4,724,000
4	4,724,000	3,679,000
5	3,679,000	2,865,000
6	2,865,000	2,231,000
7	2,231,000	1,738,000
8	1,738,000	1,353,000
9	1,353,000	1,054,000
10	1,054,000	820,800
11	820,800	639,600
12	639,600	497,600
13	497,600	387,700
14	387,700	302,000
15	302,000	235,200
16	235,200	183,200
17	183,200	142,600
18	142,600	111,100
19	111,100	86,520
20	86,520	67,380
21	67,380	40,870
22	40,870	24,790
23	24,790	15,030
24	15,030	9,119
25	9,119	5,531

Neutron Thermalization

- neutron energy comparable to k_BT

  → upscattering is important

- neutron energy < chemical binding energy

  → cannot use target-at-rest kinematics

  —scattering from molecules
  - elastic
    - entire molecule recoils
    - neutron can gain or lose energy
  - inelastic scattering
    - change in vibrational and/or rotational quantum states of molecule
    - neutron can gain or lose energy

  —scattering from crystals
  - elastic
    - entire crystal recoils
    - negligible neutron energy change
  - inelastic
    - emission or absorption of phonons
    - neutron can gain or lose energy

- neutron de Broglie wavelength comparable to interatomic spacing

  → quantum interference effects—Bragg scattering from crystalline materials

Multigroup Thermal Treatment

- typically used for E < 0.625 eV
- multigroup method (typically 32)
- all 'isotopes' classified as either moderators or non-moderators
  - — scattering from non-moderators
    - isotropic in LAB system
    - no energy change
  - — scattering from moderators
    - multigroup energy transfer kernels
    - multigroup P_1 angular distributions
- upscattering allowed, but not back into epithermal range

Scattering From Moderators

- assume that the double differential scattering cross section $\sigma(E \rightarrow E', \mu)$ is known
- approximate using a P_1 expansion

$$\sigma(E \rightarrow E', \mu) \approx \frac{1}{4\pi}\sigma_0(E \rightarrow E') + \frac{3}{4\pi}\sigma_1(E \rightarrow E')\mu$$

$$\sigma_0(E \rightarrow E') = 2\pi\int d\mu\,\sigma(E \rightarrow E', \mu)$$

$$\sigma_1(E \rightarrow E') = 2\pi\int d\mu\,\mu\,\sigma(E \rightarrow E', \mu)$$

- average over groups
  - total scattering cross section

$$\sigma^g = \left(\int_{E_{g-1}}^{E_g} dE \int_0^{\infty} dE'\phi(E)\,\sigma_0(E \rightarrow E')\right) \Big/ \left(\int_{E_{g-1}}^{E_g} dE\,\phi(E)\right), \quad \phi = \text{spectral weighting function}$$

  - 0^{th} Legendre moment

$$\sigma_0^{g \rightarrow g'} = \left(\int_{E_{g-1}}^{E_g} dE \int_{E_{g'-1}}^{E_{g'}} dE'\phi(E)\,\sigma_0(E \rightarrow E')\right) \Big/ \left(\int_{E_{g-1}}^{E_g} dE\,\phi(E)\right)$$

  - 1^{st} Legendre moment

$$\sigma_1^{g \rightarrow g'} = \left(\int_{E_{g-1}}^{E_g} dE \int_{E_{g'-1}}^{E_{g'}} dE' \int d\mu\,\phi(E)\,\mu\,\sigma_0(E \rightarrow E', \mu)\right) \Big/ \left(\int_{E_{g-1}}^{E_g} dE\,\phi(E)\right)$$

  → Note: The upper limit of the integrals over E´ are assumed to go to infinity for g´ = 1.

Scattering From Moderators (cont.)

• the P1 approximation to the group-averaged double differential scattering cross section is:

$$\sigma^{g\to g'}(\mu) = \sigma^{g}\left[\frac{\sigma_0^{g\to g'}}{\sigma^{g}}\right]\frac{1}{2\pi}\left[\frac{1}{2}+\frac{3}{2}\frac{\sigma_1^{g\to g'}}{\sigma_0^{g\to g'}}\mu\right]$$

• thus, given a scattering from the moderator in group g:

— sample the exit group from the discrete pdf $\frac{\sigma_0^{g\to g'}}{\sigma^{g}}$, $g' = 1, \ldots, g$

— sample the scattering cosine from the linear pdf $\frac{1}{2}+\frac{3}{2}\frac{\sigma_1^{g\to g'}}{\sigma_0^{g\to g'}}\mu$

— uniformly sample the azimuthal angle on $(0,2\pi)$

Continuous Energy Thermal Treatment

• typically used for E < 0.625 eV

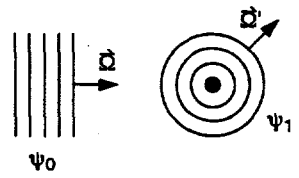
• neutrons and cross sections are continuous in energy

• different scattering types are treated explicitly

— coherent elastic (Bragg)
— incoherent elastic
— incoherent inelastic

• upscattering allowed, but not back into epithermal range

Scattering From a Single Nucleus at Rest



• asymptotic neutron wave function is given by

$$\psi(r) = \frac{1}{(2\pi)^{3/2}}[\psi_0(r) + \psi_1(r)]$$

$\psi_0(r) = e^{ik\cdot r}$ = incident wave function

$\psi_1(r) = f(\vec{\Omega}\cdot\vec{\Omega}')\frac{1}{r}e^{ikr}$ = scattered wave function

$\sigma(\vec{\Omega}\cdot\vec{\Omega}') = |f(\vec{\Omega}\cdot\vec{\Omega}')|^2$

• solve Schrödinger equation

$$\left[-\frac{h^2}{8\pi^2 m_r}\nabla^2 + V(r)\right]\psi(r) = E\psi(r)$$

$$\psi(r) = \frac{1}{(2\pi)^{3/2}}e^{ik\cdot r} - \frac{2\pi m_r}{h^2}\int dr' \frac{e^{ik|r-r'|}}{|r-r'|}V(r')\psi(r'), \qquad m_r = \text{reduced mass},\ k = \left(2\pi\sqrt{2m_r E}/h\right)\vec{\Omega}$$

• make Born approximation

$$\psi(r') \leftarrow \frac{1}{(2\pi)^{3/2}}e^{ik\cdot r'} \text{ in integrand}$$

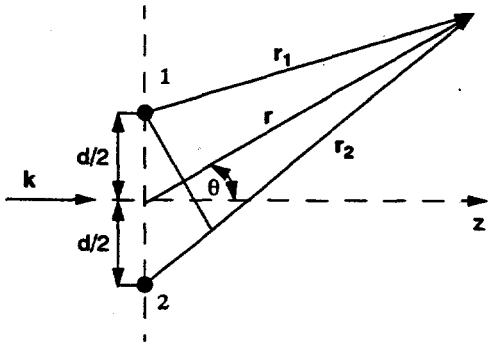
• use Fermi pseudopotential

$$V(r) = \frac{h}{m_r}a\delta(r), \qquad a = \text{scattering length}$$

• final result:

$$\sigma = a^2 = \sigma_0$$

Scattering From Two Nuclei



• wave function

$$\psi(r) = \psi_0(r) + \psi_s(r)$$

• incident wave function

$$\psi_0(r) = e^{ikz}$$

• scattered wave function

$$\psi_s(r) = \psi_1(r) + \psi_2(r)$$

$$\psi_1(r) = -\frac{a_1}{r}e^{ik\left(r - \frac{d}{2}\sin\theta\cos\varphi\right)}$$

$$\psi_2(r) = -\frac{a_2}{r}e^{ik\left(r + \frac{d}{2}\sin\theta\cos\varphi\right)}$$

• scattering cross section

$$\sigma(\theta,\varphi) = \frac{r^2}{2}|\psi_s(r)|^2 = \frac{a_1^2 + a_2^2 - 2a_1a_2}{2} + a_1a_2 2\left[\cos\left(\frac{kd}{2}\sin\theta\cos\varphi\right)\right]^2$$

➔ *interference effects*

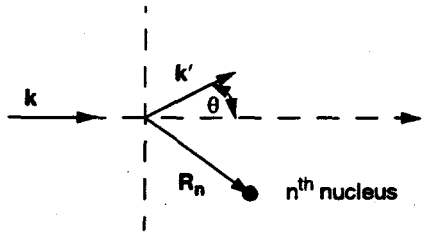
• if $a_1 = a_2$

$$\sigma(\theta,\varphi) = \sigma_0 2\left[\cos\left(\frac{kd}{2}\sin\theta\cos\varphi\right)\right]^2$$

Scattering From N Fixed Nuclei

$$\sigma(\theta,\varphi) = \frac{1}{N}\left|\sum_{n=1}^{N} a_n e^{iK\cdot R_n}\right|^2$$

$$= \frac{1}{N}\sum_{n=1}^{N} a_n^2 + \frac{1}{N}\sum_{n=1}^{N}\sum_{m\neq n}^{N} a_n a_m e^{iK\cdot(R_n - R_m)}$$



$k = k'$

assume a_n are not correlated with position, then

$$\sigma(\theta,\varphi) = \langle a^2\rangle + \frac{1}{N}\langle a\rangle^2 \sum_{n=1}^{N}\sum_{m\neq n}^{N} e^{iK\cdot(R_n - R_m)}$$

$$\langle a\rangle = \frac{1}{N}\sum_{n=1}^{N} a_n \quad \langle a^2\rangle = \frac{1}{N}\sum_{n=1}^{N} a_n^2$$

rearrange

$$\sigma(\theta,\varphi) = \sigma_{inc} + \sigma_{coh}\frac{1}{N}\left|\sum_{n=1}^{N} e^{iK\cdot R_n}\right|^2, \qquad \sigma_{inc} = \frac{1}{4\pi}\left(\langle a^2\rangle - \langle a\rangle^2\right) \qquad \sigma_{coh} = \frac{1}{4\pi}\langle a\rangle^2$$

If $\sigma_{inc} = 0$ → *coherent elastic scattering*

General Case of Scattering From N Nuclei

• in general, we have

$$\sigma(E\to E', \vec{\Omega}\to\vec{\Omega}') = \sigma_{inc}(E\to E', \vec{\Omega}\to\vec{\Omega}') + \sigma_{coh}(E\to E', \vec{\Omega}\to\vec{\Omega}')$$

$$\sigma_{inc}(E\to E', \vec{\Omega}\to\vec{\Omega}') = \frac{\sigma_{inc}}{2h}\sqrt{\frac{E}{E'}}\frac{1}{2\pi}\int_{-\infty}^{\infty} dt\int dr e^{i(K\cdot r - 2\pi\varepsilon t/h)} G_s(r,t)$$

$$\sigma_{coh}(E\to E', \vec{\Omega}\to\vec{\Omega}') = \frac{\sigma_{coh}}{2h}\sqrt{\frac{E}{E'}}\frac{1}{2\pi}\int_{-\infty}^{\infty} dt\int dr e^{i(K\cdot r - 2\pi\varepsilon t/h)} G(r,t)$$

$$G_s(r,t) = \frac{1}{N}\sum_{n=1}^{N}\int dr'\langle\delta(r + R_n(0) - r')\,\delta(r' - R_n(t))\rangle$$

$$G(r,t) = \frac{1}{N}\sum_{n=1}^{N}\sum_{m\neq 1}^{N}\int dr'\langle\delta(r + R_m(0) - r')\,\delta(r' - R_n(t))\rangle$$

$$\varepsilon = E - E', \; K = 2\pi m(v - v')/h$$

$$S_s(K,\varepsilon) = \frac{1}{2\pi}\int_{-\infty}^{\infty} dt\int dr e^{i(K\cdot r - 2\pi\varepsilon t/h)} G_s(r,t), \quad S(K,\varepsilon) = \frac{1}{2\pi}\int_{-\infty}^{\infty} dt\int dr e^{i(K\cdot r - 2\pi\varepsilon t/h)} G(r,t)$$

• for some materials, we may define

$$S_s(\alpha,\beta) = k_B T e^{\beta/2} S_s(K,\varepsilon), \quad S(\alpha,\beta) = k_B T e^{\beta/2} S(K,\varepsilon)$$

$$\alpha = \frac{h^2K^2}{2Mk_BT}, \; \beta = -\varepsilon/k_BT$$

→ *this representation is used for incoherent inelastic scattering*

Tallies

&

Statistical Edits

Tallies

- analog Monte Carlo
  - — each history is assigned a 'weight' of 1
  - — final event estimator
    - follow each history until it terminates by absorption or leakage
    - if termination by leakage, tally 1 leakage
    - if termination by absorption, tally
      - $\Sigma_{a,j}/\Sigma_a$ for all nuclides j in the region where the absorption occurred
      - $\nu\Sigma_f/\Sigma_a$
  - — guarantees that absorptions + leakages = source + (n,2n) exactly
- weighted Monte Carlo
  - — each history is initially assigned a weight (w) of 1
  - — at each collision:
    - employ collision estimator
      - tally $w\Sigma_{x,j}/\Sigma_t$ for all reactions x and nuclides j in the region where the collision occurred
      - collision estimator is usually best for optically thick regions
    - reduce the weight by multiplying by the nonabsorption probability, $w \leftarrow w(1-\Sigma_a/\Sigma_t)$
    - → survival biasing
  - — for each flight (collision-to-collision, boundary-to-collision, etc.)
    - employ path-length estimator
      - tally $ws\Sigma_{x,j}$ for all reactions x and nuclides j in the region (s = path-length of flight)
      - path-length estimator is usually best for optically thin regions
  - — terminate histories using Russian Roulette
  - — better estimator is a linear combination of collision and path-length estimators
    - combining coefficients are chosen to minimize variance
    - combining coefficients are edit-quantity-dependent
  - — absorptions + leakages = source + (n,2n) on the average

Introduction to Statistics

Characterizing Random Data

Given a set of N random values $\{x_1, x_2, ..., x_N\}$, one can calculate:

• sample mean

$$m = \frac{1}{N}\sum_{i=1}^{N} x_i$$

• sample variance

$$s^2 = \frac{1}{N-1}\sum_{i=1}^{N} (x_i - m)^2$$

→ for truly random data with pdf p(x):

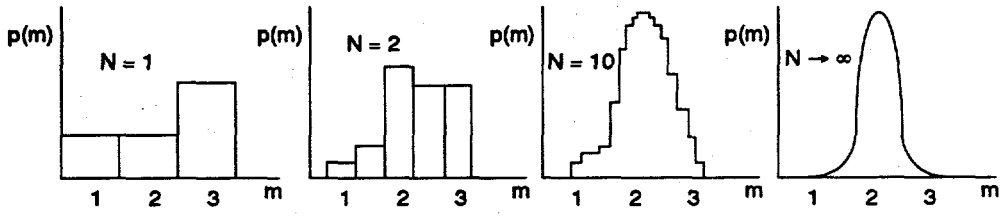
$$\langle m \rangle = E[m] = \mu, \qquad \mu = \int dx x p(x) = \text{mean}$$

$$\langle s^2 \rangle = E[s^2] = \sigma^2, \qquad \sigma^2 = \int dx (x-\mu)^2 p(x) = \text{variance}$$

Introduction to Statistics

Central Limit Theorem

As the sample size N increases, the distribution of the sample mean m of a sample drawn from almost any distribution approaches a normal distribution with mean μ and variance σ^2/N.



Confidence Intervals

• the 95% confidence intervals d_{95} are given by

$d_{95} = t_{N-1} s/\sqrt{N}$, t_{N-1} = Student's t factor for N-1 degrees of freedom

Given a sample mean m, with sample variance s, there is a 95% probability that the population mean μ lies in the interval $[m-d_{95}, m+d_{95}]$.

Test for Normality

• *the skewness coefficient c (<c> = 0 for normal distribution)*

$$c = \frac{1}{s^3}\frac{1}{N-1}\sum_{i=1}^{N} (x_i - m)^3$$

• a confidence interval is doubtful if $c^2 \geq (1.96)^2\left[\frac{6(N-1)(N-2)}{N(N+1)(N+3)}\right]$ (marked with a "*" in RACED)

• a confidence interval is probably meaningless if $c^2 \geq (2.81)^2\left[\frac{6(N-1)(N-2)}{N(N+1)(N+3)}\right]$ (marked with a "?")

RACED Output

Single Estimators

- RACER supplies RACED with one value of a reaction rate, x_i, for each of N non-discarded batches i
- RACED computes
  - — the sample mean: $m = \frac{1}{N}\sum_{i=1}^{N} x_i$
  - — the sample variance: $s^2 = \frac{1}{N-1}\sum_{i=1}^{N} (x_i - m)^2$
  - — the 95% confidence interval: $d_{95} = t_{N-1} s/\sqrt{N}$
  - — the skewness coefficient: $c = \frac{1}{N-1}\sum_{i=1}^{N} \left(\frac{x_i - m}{s}\right)^3$
- RACED outputs the mean along with the 95% confidence interval and an indication of its uncertainty

RACED Output

Double Estimators

- RACER supplies RACED with path-length and collision estimators, x_i and y_i, for each batch i
- RACED computes
  - — the sample means: $m_x = \frac{1}{N}\sum_{i=1}^{N} x_i$ and $m_y = \frac{1}{N}\sum_{i=1}^{N} y_i$
  - — the sample variance: $s_x^2 = \frac{1}{N-1}\sum_{i=1}^{N} (x_i - m_x)^2$ and $s_y^2 = \frac{1}{N-1}\sum_{i=1}^{N} (y_i - m_y)^2$
  - — the covariance: $s_{xy}^2 = \frac{1}{N-1}\sum_{i=1}^{N} (x_i - m_x)(y_i - m_y)$
- these are used to compute the minimum total variance and corresponding mean
  - — mean: $m = \alpha m_x + (1-\alpha) m_y$
  - — minimum variance: $s^2 = \alpha^2 s_x^2 + 2\alpha(1-\alpha) s_{xy}^2 + (1-\alpha)^2 s_y^2$

    $\alpha = \frac{s_y^2 - s_{xy}^2}{s_x^2 - 2s_{xy}^2 + s_y^2}$, but set to 0 if negative and to 1 if > 1
  - — the 95% c. i.: $d_{95} = t_{N-1} s/\sqrt{N}$
  - — the skewness coefficient is obtained in an analogous manner
- RACED outputs the mean along with the 95% confidence interval and an indication of its uncertainty

RACED Output

Combination Edits

- combination edits are sums of other edits, i.e.

$$x_i = \sum_{j \in \text{combination}} x_{ji},\ y_i = \sum_{j \in \text{combination}} y_{ji}$$

x_{ji}, y_{ji} = estimators of quantity j in batch i

- the sums x_i and y_i are analyzed using the usual procedures for single and double estimators

Ratio Edits

- ratio edits are ratios of other edits
- ratio edits use a single estimator only
- let x_i and y_i be batch i single estimator values for the numerator and denominator, resp., of the ratio
  - — RACED computes

$$m_x = \frac{1}{N}\sum_{i=1}^{N} x_i \text{ and } m_y = \frac{1}{N}\sum_{i=1}^{N} y_i$$

$$s_x^2 = \frac{1}{N-1}\sum_{i=1}^{N} (x_i - m_x)^2,\ s_y^2 = \frac{1}{N-1}\sum_{i=1}^{N} (y_i - m_y)^2,\ \&\ s_{xy}^2 = \frac{1}{N-1}\sum_{i=1}^{N} (x_i - m_x)(y_i - m_y)$$

  - — these are used to obtain
    - the mean of the ratio: $m = m_x/m_y$
    - the variance of the ratio: $s^2 = \left(\frac{m_x}{m_y}\right)^2\left[\frac{s_x^2}{m_x^2} - \frac{2s_{xy}^2}{m_x m_y} + \frac{s_y^2}{m_y^2}\right]$
    - the 95% confidence interval: $d_{95} = t_{N-1}s/\sqrt{N}$

RACED Output

Notes:

- **the single estimator procedures are used if:**
  - — *delta tracking* is being used
  - — the *bigdepl* option is being used (estimator is average of path length and collision estimators)
- **for double estimators, separate minimum σ^2 linear combinations are found for each quantity**
  - — energy-integrated values will not in general equal the sum over edit groups
  - — combination edits will not in general equal the sum of the constituent edits
  - — ratio edits will not in general equal the ratio of double estimators

Source Correlation Correction of Confidence Intervals

- For iterated-source problems, the batch-wise results are correlated
  - — starting locations for a batch are sampled from the fission sites of the previous batch
  - — this leads to a bias in the confidence intervals
- For iterated-source problems, RACED optionally applies a correction factor to the confidence intervals
- The **true sample variance** can only be found by analyzing results from independent calculations

  m_j = result from job j (all J jobs are identical except for random number sequences)

$$s_\mu^2 = \frac{1}{J-1}\sum_{j=1}^{J}(m_j-\bar{m})^2,\ \bar{m} = \frac{1}{J}\sum_{j=1}^{J} m_j$$

  - — the expected value of s_μ^2 is σ_μ^2
- The **apparent sample variance** is computed as if the batch-wise results in a single job were uncorrelated

x_{ij} = result from batch i of job j, $\quad m_j = \frac{1}{N}\sum_{i=1}^{N} x_{ij}$

$$s_{\mu j}^2 = \frac{1}{N(N-1)}\sum_{i=1}^{N}(x_{ij}-m_j)^2$$

  - — the expected value of $s_{\mu j}^2$ is $\sigma_{\mu a}^2$, but $\sigma_{\mu a}^2$ is not equal to σ_μ^2
- An approximate relationship between the true and apparent variances is

$$\sigma_\mu^2 = \sigma_{\mu a}^2\left[1+\frac{2}{N}\sum_{k=1}^{N-1}(N-k)\gamma_k\right],\ \text{where } \gamma_k \text{ is the correlation coefficient lag k}$$

- RACED correction:
  - — make the approximation $\gamma_k \approx \gamma_{k-1}(\lambda_1/\lambda_0)$, where λ_1/λ_0 = dominance ratio (default = 0.9)
  - — approximate the true variance by

$$s_\mu^2 = \max\left[s_{\mu j}^2\left[1+\frac{2\hat{\gamma}_1}{1-(\lambda_1/\lambda_0)}\right], s_{\mu j}^2\right]$$

Source Normalization Bias

Source of Bias

- In a real reactor at steady state, the number of neutrons/generation fluctuates about a mean value

- In RACER, the number of starting histories per batch is constant
  - — prevents extinction or overflow of population
  - — simplifies algorithm
  - — leads to a bias for eigenvalue problems, if not corrected, since all starting distributions are (unfairly) given equal weights

Bias Correction in RACED

Define:

$$K = \left(\prod_{i=1}^{N} k_i\right)^{1/N} = \text{geometric average of batch multiplication factors}$$

$$M_1 = 1; \qquad M_i = \left(\prod_{j=1}^{i-1} k_j\right)/K^{i-1} = \prod_{j=1}^{i-1}\frac{k_j}{K},\ i>1$$

Edit the weighted batch averages

$$\bar{x} = \left(\sum_{i=1}^{N} M_i x_i\right)/\left(\sum_{i=1}^{N} M_i\right)$$

- For ratio edits the numerator and denominator are corrected separately
- The source normalization is normally turned on for iterated-source calculations
- It is turned off when the *bigdepl* option is being used
- It can optionally be turned off for any calculation

Propagation of Error

• General Equation

$x = x(y_1, y_2, \ldots, y_n)$

$$\sigma_x^2 = \sum_{i=1}^{n}\sum_{j=1}^{n}\frac{\partial x}{\partial y_i}\frac{\partial x}{\partial y_j}\sigma_{y_i y_j}^2, \qquad \sigma_{y_i y_i}^2 = \sigma_{y_i}^2$$

examples:

— sum of two quantities

$x = y + z$

$$\sigma_x^2 = \sigma_y^2 + 2\sigma_{yz}^2 + \sigma_z^2$$

— product of two quantities

$x = yz$

$$\sigma_x^2 = z^2\sigma_y^2 + 2yz\sigma_{yz}^2 + y^2\sigma_z^2$$

• Equation for Uncorrelated quantities

$x = x(y_1, y_2, \ldots, y_n)$

$$\sigma_x^2 = \sum_{i=1}^{n}\left(\frac{\partial x}{\partial y_i}\right)^2\sigma_{y_i}^2$$

example:

— difference of two estimators

$x = y - z$

$$\sigma_x^2 = \sigma_y^2 + \sigma_z^2$$

→ *These expressions are in terms of σ^2, but similar relations hold for the square of the confidence intervals.*

Monte Carlo

Eigenvalue Calculations

Eigenvalue Problems — Reactor "criticality"

$$\Psi(p) = \int \Psi(p') R(p' \rightarrow p)\, dp' + \frac{1}{K_{eff}} \int \Psi(p') F(p' \rightarrow p)\, dp'$$

$$\Psi = R \bullet \Psi + \frac{1}{K_{eff}} F \bullet \Psi$$

Generation Model

$$\Psi^{(i+1)} = R \bullet \Psi^{(i+1)} + \frac{1}{K_{eff}^{(i)}} F \bullet \Psi^{(i)}$$

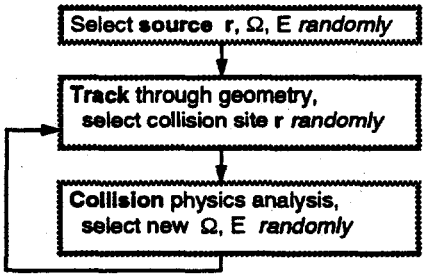
$$\Psi^{(i+1)} = \frac{1}{K_{eff}^{(i)}} [I - R]^{-1} F \bullet \Psi^{(i)} \qquad K_{eff}^{(i)} = \int F \bullet \Psi^{(i)}\, dp dp'$$

Monte Carlo approach:

- Guess $\Psi^{(0)}, K_{eff}^{(0)}$
- Follow a "batch" of histories, estimate $\Psi^{(i)}, K_{eff}^{(i)}$
- Repeat until converged
- Discard initial batches
- Iterate, accumulating scores — until variances are small enough

Single particle

- random-walk for particle history
- simulate events, from birth to death
- tally events of interest



Batch of histories ("generation")

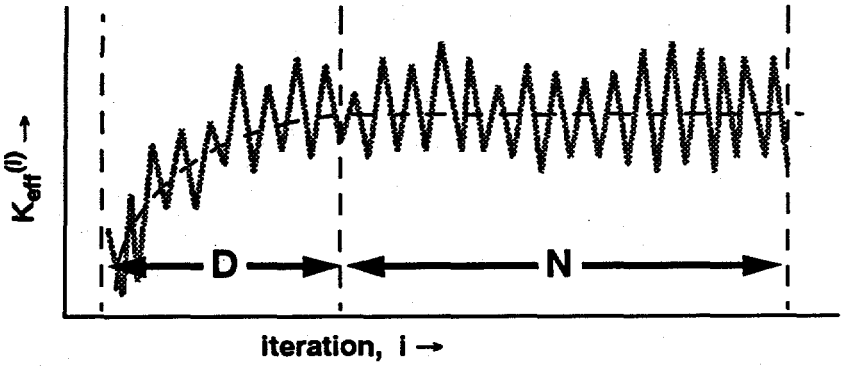
- random-walk for many particle histories
- tally the aggregate behavior

Overall

- timesteps
  - — geometry changes
  - — material changes
    - fuel depletion
    - burnable absorbers
    - control rods

```
* Loop over timesteps
•    * Loop over batches
•    •    * Loop over histories
•    •    •    → random walk
•    •    •••
•    •    → update Keff & reaction rates
•    •••
•    → compute statistics
•    → update number densities, .....
•••
```


Monte Carlo Eigenvalue Calculations



- D = number of initial "contaminated" batches to discard
- N = number of batches to keep, for averaging results

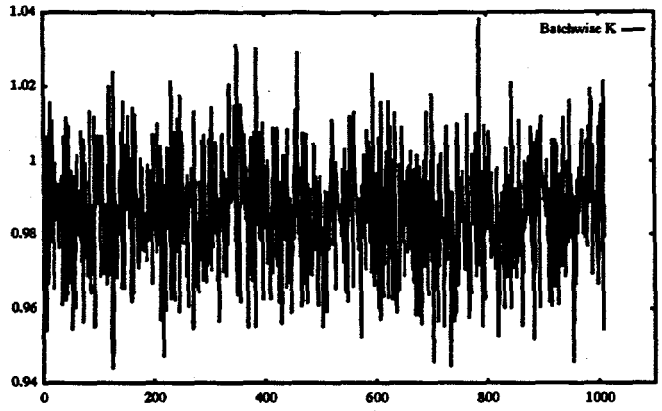
D & N determined by heuristics, not theory:

- should have $\rho^D \ll \sigma_{batch}$, where ρ is the dominance ratio, & σ_{batch} is std. dev. in K_{batch}
- should have $N \gg 25$, so that σ's are small & reliably estimated
- use larger N on weekends, ...

Eigenvalue Calculations — Convergence

[141]

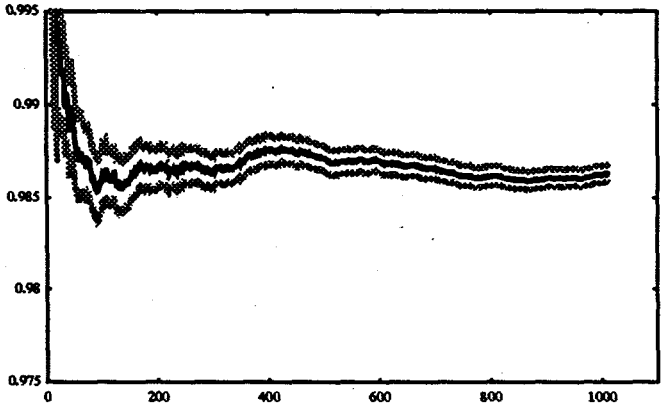
K-effective — batchwise



M = 1,000 histories/batch

D = 10 batches discarded
N = 1,000 batches kept

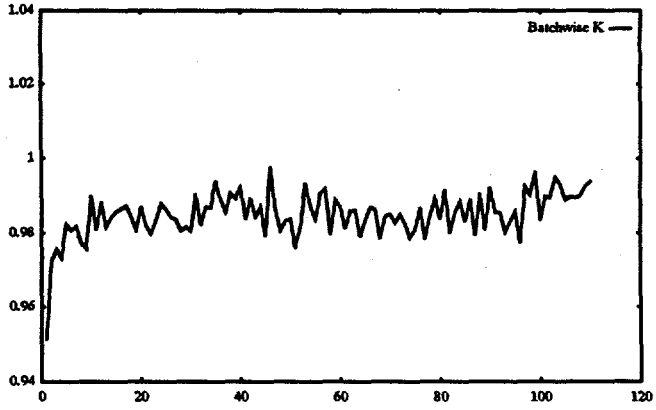
K-effective — cumulative



Eigenvalue Calculations — Convergence

[142]

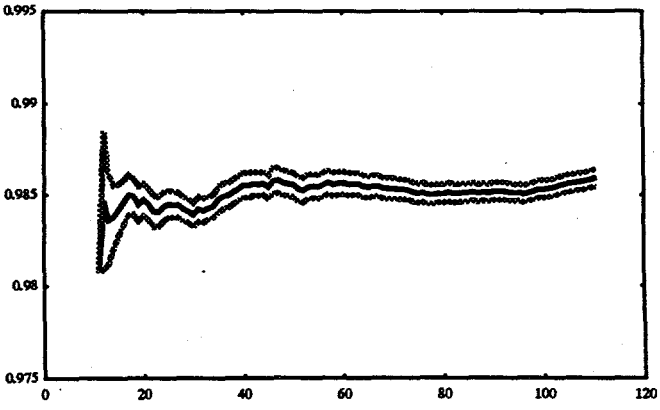
K-effective — batchwise



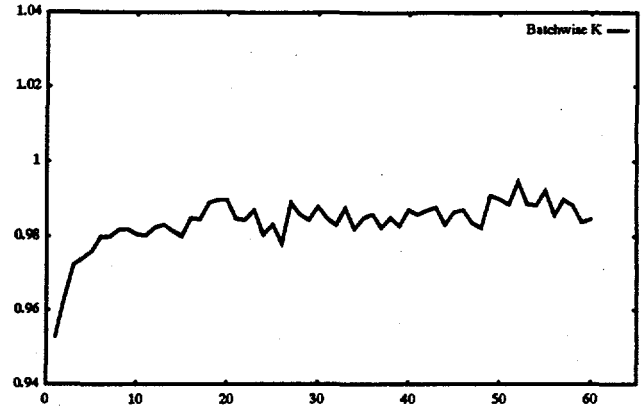
M = 10,000 histories/batch

D = 10 batches discarded
N = 100 batches kept

K-effective — cumulative



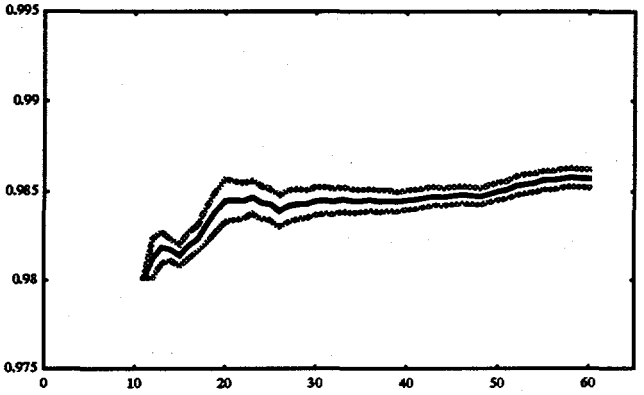
K-effective — batchwise



M = 20,000 histories/batch

D = 10 batches discarded
N = 50 batches kept

K-effective — cumulative



Bias

- K_j = eigenvalue from a single Monte Carlo calculation
- Repeat the calculation ***J*** times (with different random numbers)
- If

$$\lim_{J \to \infty} \frac{1}{J} \sum_{j=1}^{J} K_j = K_{exact}$$

then the Monte Carlo algorithm is ***unbiased***

Eigenvalue Calculations

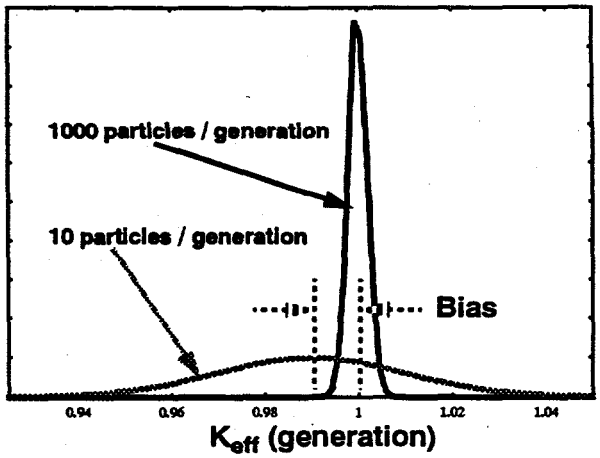
- ***Unaccelerated power method*** $S^{(i+1)}(r) = \frac{1}{K^{(i)}} \int S^{(i)}(r') H(r' \to r)\, dr'$
- Typical Monte Carlo algorithm
  - — follow batch of ***M*** histories — store fission sites in a "bank"
  - — from ***M'*** banked sites, randomly choose ***M*** to start next batch
- Source renormalization [***M'→M***] introduces **bias**
  - — needed to prevent population explosion or extinction

Theory

- Discrete M C: Gelbard & Prael (1974), Brissenden & Garlick (1986),...
- Continuous M C: Sutton & Brown (1991), ...
- bias $= -\frac{\sigma_k^2}{k_{eff}}$ • [sum of correlation coeff's between batch k's]

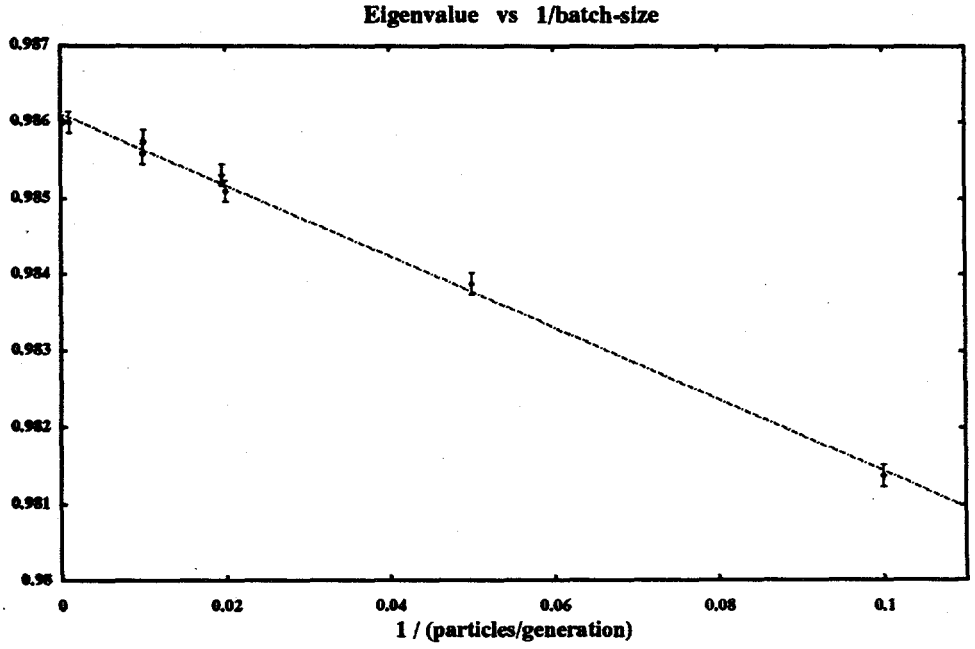
$$= -\frac{\sigma_k^2}{2k_{eff}} \sum_{j=1}^{\infty} R_{kk}^j$$

- Smaller M
  - **M** = histories / generation
  - ➜ larger bias in k_{eff}
  - ➜ larger bias in source



Practice

- bigger, faster computers ➜ larger batches (M >> 1,000)
- negligible bias in K_{eff}



Theory

- Discrete M C: Gelbard, Brown, & Gu (1993)
- Bias in region source $= -\frac{\sigma_k \sigma_s}{k_{eff}}$ • [sum of correlation coeff's between batch k & region source]

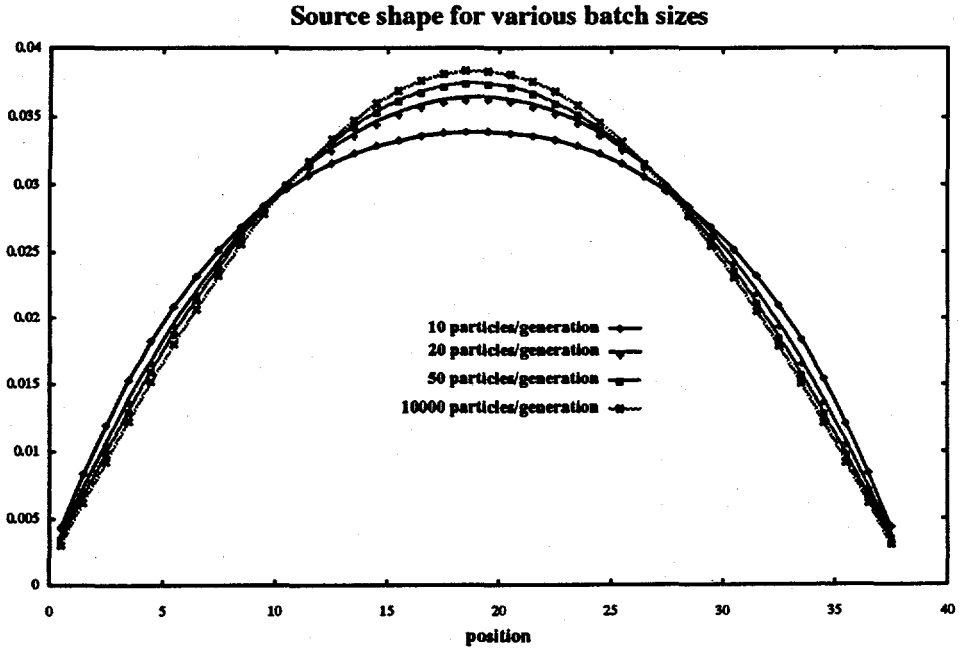
$$= -\frac{\sigma_k \sigma_s}{2k_{eff}} \sum_{j=1}^{\infty} R_{ks}^{j}$$

- Smaller M (**M** = histories / generation)
  - ➜ larger bias in region source
- For reasonable assumptions (verified for many test cases),

$$\left(\frac{\text{bias}}{\sigma}\right)_{\text{Region Source}} < \left(\frac{\text{bias}}{\sigma}\right)_{\text{Eigenvalue}}$$

Practice

- bigger, faster computers ➜ larger batches (**M** >> 1,000)
- negligible bias in eigenvalue ➜ negligible bias in region sources
- similar reasoning should (?) apply to other reaction rates



Source Normalization Bias

Derivation

The transport equation can be written in integral form in terms of the source density as

$$\hat{s}(r) = \frac{1}{\hat{\lambda}}\int dr' \hat{s}(r') H(r' \rightarrow r).$$

Consider a Monte Carlo calculation where

$$S^{(m)}(r) = \sum_{i=1}^{I^{(m)}} w_i^{(m)} \delta\left(r - r_i^{(m)}\right) = \text{fission bank for batch m}$$

$w_i^{(m)}$ = fission production weight at fission site i of batch m

$r_i^{(m)}$ = location of fission site i of batch m.

Now, assume that the Monte Carlo algorithm is such that

$$w_i^{(m)} / \sum_{j=1}^{I^{(m)}} w_j^{(m)} = \text{probability that a particular history in batch m+1 will begin at } r_i^{(m)}$$

and thus

$$S^{(m+1)}(r) = N\frac{\int dr' S^{(m)}(r') H(r' \rightarrow r)}{\int dr' S^{(m)}(r')} + \sqrt{N}\varepsilon^{(m)}(r)$$

where $\varepsilon^{(m)}(r)$ is a random function with zero mean.

Now define a normalized distribution

$$s(r) = \langle S^{(m)}(r)\rangle/N,$$

and define a fluctuating component of the Monte Carlo source distribution so that

$$S^{(m)}(r) = Ns(r) + \sqrt{N}\zeta^{(m)}(r).$$

Substitute into

$$S^{(m+1)}(r) = N\frac{\int dr' S^{(m)}(r') H(r' \to r)}{\int dr' S^{(m)}(r')} + \sqrt{N}\varepsilon^{(m)}(r),$$

expand in powers of $1/\sqrt{N}$, retaining only terms of order 1/N, and ensemble average to obtain

$$s(r) = \frac{1}{\hat{\lambda}}\int dr' s(r') H(r' \to r) - \frac{1}{N\hat{\lambda}}\int dr' A(r', r)\int dr'' \langle \zeta(r')\zeta(r'')\rangle$$

$$\zeta^{(m+1)}(r) = \int dr' \zeta^{(m)}(r') A(r', r) + \varepsilon^{(m)}(r).$$

Now, let

$$s(r) = \hat{s}(r) + \frac{1}{N}s'(r).$$

Substitute, retaining term to 1/N, to get

$$s'(r) = -\frac{1}{\hat{\lambda}}[1-\Gamma]^{-1}\Gamma\int dr' \langle \zeta(r)\zeta(r')\rangle, \text{ where } \Gamma f(r) = \frac{1}{\hat{\lambda}}\int dr' [H(r' \to r) - \hat{s}(r)] f(r').$$

Now, define the eigenvalue bias as

$$\Delta\lambda = \lambda - \hat{\lambda} = \frac{1}{N}\int dr s'(r),$$

and we find that

$$\Delta\lambda = -\frac{1}{N\hat{\lambda}}\int dr [1-\Gamma]^{-1}\Gamma\int dr' \langle \zeta(r)\zeta(r')\rangle$$

$$= -\frac{1}{N\hat{\lambda}}\int dr \sum_{j=1}^{\infty}\Gamma^j\int dr' \langle \zeta(r)\zeta(r')\rangle$$

$$= -\frac{1}{N\hat{\lambda}}\int dr \sum_{j=1}^{\infty}\int dr' \langle \zeta^{(m+j)}(r)\zeta^{(m)}(r')\rangle.$$

Finally, use

$$\frac{1}{N}\int dr S^{(m)}(r) = \frac{1}{N}\int dr \left[Ns(r) + \sqrt{N}\zeta^{(m)}(r)\right]$$

so that

$$\lambda^{(m)} - \hat{\lambda} = \frac{1}{\sqrt{N}}\int dr \zeta^{(m)}(r)$$

and

$$\Delta\lambda = \frac{1}{\hat{\lambda}}\sum_{j=1}^{\infty}\left\langle \left(\lambda^{(m+j)} - \hat{\lambda}\right)\left(\lambda^{(m)} - \hat{\lambda}\right)\right\rangle = \text{(sum of all eigenvalue covariances)/(true eigenvalue)}$$

→ *For RACER, and the number of histories/batch currently used for design calculations, the bias appears to be completely negligible.*

MacMillan's Correction

The Monte Carlo iterative scheme may be represented as

$$S^{(m+1)}(r) = N\frac{\int dr' S^{(m)}(r') H(r' \to r)}{\int dr' S^{(m)}(r')} + \sqrt{N}\varepsilon^{(m)}(r).$$

The ideal (unbiased scheme), would be

$$\hat{S}^{(m+1)}(r) = \frac{\int dr' \hat{S}^{(m)}(r') H(r' \to r)}{\hat{\lambda}} + \sqrt{N}\varepsilon^{(m)}(r),$$

where the number of histories/batch can fluctuate, and where the true eigenvalue $\hat{\lambda}$ is unknown.

Comparing the two operators, we can see that they differ by the factor

$$\frac{\int dr' S^{(m)}(r')/N}{\hat{\lambda}} = \frac{\lambda^{(m)}}{\hat{\lambda}}.$$

MacMillan's scheme approximates this by

$$\frac{\lambda^{(m)}}{\left[\prod_{m=1}^{M} \lambda^{(m)}\right]^{1/M}}.$$

Variance Reduction

Stratified Sampling

Consider the integral

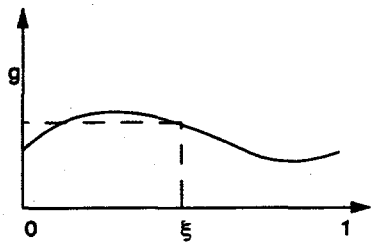
$$G = \int_0^1 dx g(x).$$

The Monte Carlo evaluation is

$$\bar{G} = \frac{1}{N}\sum_{n=1}^{N} g(\xi_n)$$

with variance

$$\sigma^2 = \frac{1}{N}\left[\int_0^1 dx g^2(x) - G^2\right].$$



Stratified sampling:

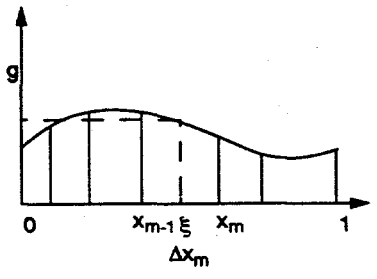
- divide domain into M intervals m
- perform N_m trials in each interval ($\sum_m N_m = N$)
- compute the integral

$$\bar{G}_m = \frac{\Delta x_m}{N_m}\sum_{n=1}^{N_m} g(x_{m-1} + \xi_n \Delta x_m)$$

$$\bar{G} = \sum_{m=1}^{M} \bar{G}_m$$

- the variance is

$$\sum_{m=1}^{M} \frac{1}{N_m}\left[\Delta x_m \int_{x_{m-1}}^{x_m} dx g^2(x) - \bar{G}_m^2\right] \le \sigma^2$$



Importance Sampling

Consider the function g(x) and PDF p(x) defined on the interval [a,b]:

- the mean is

$$\langle g \rangle = \int_a^b dx p(x) g(x)$$

- the variance is

$$\sigma^2 = \langle g^2 \rangle - \langle g \rangle^2$$

$$\langle g^2 \rangle = \int_a^b dx p(x) g^2(x)$$

Now, suppose we use instead the PDF q(x) to chose the random numbers on [a,b]:

- assign each trial a "weight" $w(x) = p(x)/q(x)$
- the mean is unchanged

$$\int_a^b dx q(x) w(x) g(x) = \int_a^b dx p(x) g(x) = \langle g \rangle$$

- the variance is

$$\tilde{\sigma}^2 = \int_a^b dx q(x) w^2(x) g^2(x) - \left[\int_a^b dx q(x) w(x) g(x)\right]^2$$

$$= \int_a^b dx q(x) \left[\frac{p(x)}{q(x)} g(x) - \langle g \rangle\right]^2$$

- if q(x) is chosen to be $p(x) g(x) / \langle g \rangle$ then the variance will be zero (however, $\langle g \rangle$ is not known)

→ *To minimize the variance, choose* q *so that* pg/q *is approximately constant. Sample more in the "important" regions.*

Splitting and Rouletting

Goal: Track histories that will contribute significantly to the desired results, and don't track those that will not—but do so in a way that is unbiased.

- define a minimum (w_{low}), "average" (w_{ave}), and maximum weights (w_{hi}) for each region & energy range

- following each boundary crossing and collision

  — if the history weight (w) is less than w_{low}
  - roulette
  - terminate the history with probability $1 - w/w_{ave}$
  - if it survives, increase the weight to w_{ave}

  — if w is greater than w_{hi}
  - split
  - replace the original history with $\lfloor w/w_{ave} \rfloor$ new histories
  - create yet another new history with probability $(w/w_{ave}) - \lfloor w/w_{ave} \rfloor$
  - assign the new histories each a weight of w_{ave}

- rouletting increases the variance

- splitting reduces the variance

- splitting requires additional storage

- if done properly, the use of splitting and rouletting decreases the cpu time required to achieve a variance

Additional Variance Reduction Methods

• survival biasing

Instead of terminating upon an absorption, always scatter with reduced weight

• exponential transform

modify total cross section as $\Sigma_{ex} = \Sigma_t[1 - p\vec{\Omega} \cdot \vec{\Omega}_0]$

$0 \le p < 1$ = adjustable parameter

$\vec{\Omega}$ = neutron direction

$\vec{\Omega}_0$ = preferred direction

• correlated sampling

maintain some correlation (e.g., random number sequence) between unperturbed & perturbed calcs.

Importance Sampling

Consider the fixed source problem

$$\psi(R) = \int dR' K(R' \to R)\psi(R') + S(R)$$

$$X = \int dR \psi(R) x(R)$$

$\psi(R)$ = collision density at phase-space point R

S(R) = first-flight collision density due to fixed sources

X = desired functional (e.g., reaction rate)

x(R) = contribution to functional due to a collision at R

Decompose the kernel

$$K(R' \to R) = (1 - \alpha(R'))\, k(R' \to R)$$

$\alpha(R)$ = absorption probability at R

$k(R' \to R)$ = normailized kernel

Now, define altered quantities

$$\tilde{\alpha}(R) = x(R)/\psi^\dagger(R)$$

$$\tilde{k}(R' \to R) = k(R' \to R)\psi^\dagger(R) / \int dR k(R' \to R)\psi^\dagger(R)$$

$$\tilde{S}(R) = S(R)\psi^\dagger(R) / \int dR S(R)\psi^\dagger(R)$$

where the adjoint function $\psi^\dagger(R)$ satisfies

$$\psi^\dagger(R) = \int dR' K(R \to R')\psi^\dagger(R') + x(R)$$

→ *A Monte Carlo calculation using the altered quantities (and with appropriate weights) produces the correct result with <u>every</u> history! Of course, the adjoint is never exactly known.*

Multiplied Fixed-Source Calculations

Multiplied Fixed-Source Calculations [162]

Physical Problem: subcritical system with an extraneous source

RACER Method:

- total number of histories/batch = constant = N
- N = S + F
  - S = number of histories from fixed-source
  - F = number of histories from induced fission source
  - S and F may vary from batch to batch
- $\langle F \rangle = kN$
- $\langle S \rangle = (1-k)N$
- k = (expected # of neutrons due to fission)/(total # of neutrons)

RACER Algorithm:

1) first batch: $F_1 = 0$; $S_1 = N$
2) estimate k
   - if in discarding stage, $k_i = F_i/N$
   - otherwise, use cumulative value
3) subsequent batches

$$F_{i+1} = \lfloor k_i N + \xi \rfloor$$

$$S_{i+1} = N - F_{i+1}$$

ξ = random number on (0,1)

- Note: the spectra used for the fixed and fission sources may be different.

Vector & Parallel

Monte Carlo

Trends in Computing Technology

- Commodity chips:
  - — CPU power → ~ 2 x gain / 18 months
  - — Memory density → ~ 2 x gain / 18 months
- Supercomputers: → ~ 4 yr development time
- UNIX operating systems + networking → distributed computing

Need to reduce Monte Carlo uncertainties by 2x ?

- **Wait** 3-4 years for new computer

or

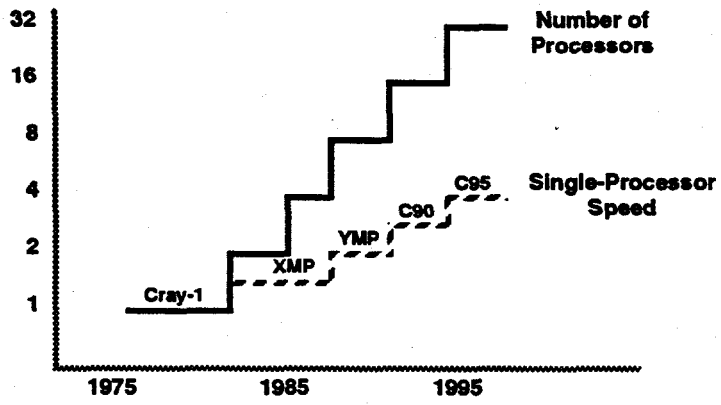
- Use parallel &/or vector processing **now**

Alternatives

- CRAY Supercomputers → **real** supercomputing
- Moderately Parallel computers → **cheap** supercomputing (??)
- Workstation Network → **free** supercomputing (??)

Scientific Computations — State-of-the-Art

Supercomputer Advances:



Single-Processor Speed: 20 yrs ⇒ 4 X

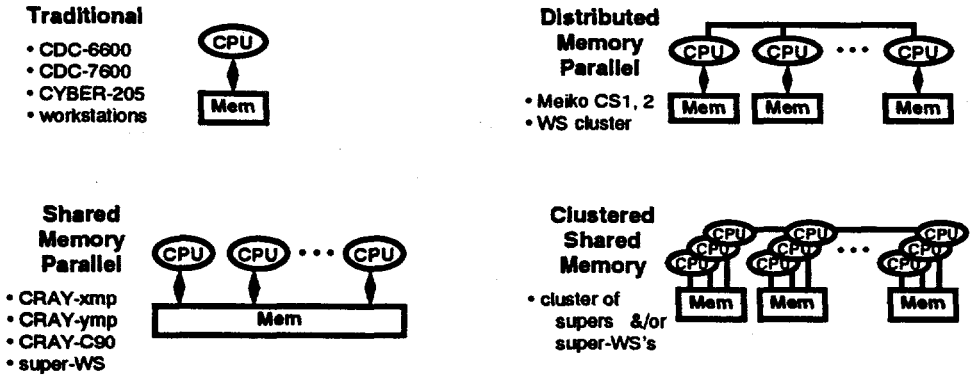
Number of Processors: 20 yrs ⇒ 32-64 X

⇒ **Parallel & vector processing are now "routine" & necessary for high-performance computing**

Characterize computers by:

- CPU: scalar, superscalar, pipelined, vector, RISC, CISC, ...
- Memory: shared, distributed, cache, banks, bandwidth, ...
- Interconnects: bus, switch, ring, grid, ...

Basic types, with examples:



Particle transport Monte Carlo is naturally parallel

- Fixed-source problems: each particle in **problem** is independent
- Eigenvalue problems: each particle in **generation** is independent

⇒ Particle **histories** can be analyzed in parallel

Monte Carlo is often the first use for advanced computers

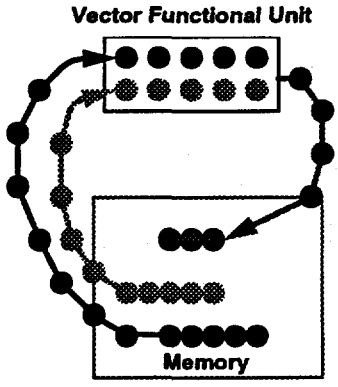
- Easy to port — compact coding, little I/O, simple parallel algorithm
- Flexible — independent histories on each node
- Big payoff — bigger & faster calculations

Computational considerations

- *Expensive — hours / days / weeks of computing*
- Compact — moderate memory size
- CPU-intensive — very little I/O or communications

T(computation) >> T(communications)

Vector Processing



- **Fortran**

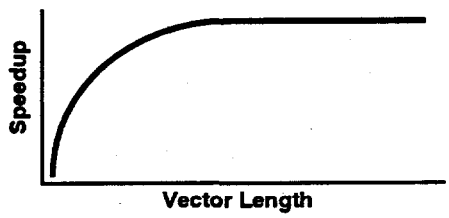
```
do j=1,L
    a(j) = b(j) + c(j)
enddo
```

- **Timing**

$$T_{vector} = t_{startup} + L \cdot t_{operation}$$

- **Speedup**

$$S = T_{scalar} / T_{vector} = \frac{L \cdot t_{scalar\text{-}op}}{t_{startup} + L \cdot t_{vector\text{-}op}} \Rightarrow \frac{t_{scalar\text{-}op}}{t_{vector\text{-}op}}$$



Vector Operations

• **Gather** — form a contiguous vector from data in arbitrary locations

```
do j=1,L
    a(j) = b( i(j) )
enddo
```

```
b():   7,  5,  3,  6,  1,  14
i():   4,  1,  2,  4,  4
a():   6,  7,  5,  6,  6
```

• **Scatter** — disperse vector data to arbitrary locations

```
do j=1,L
    a( i(j) ) = b(j)
enddo
```

```
b():   1,  2,  3,  4
i():   4,  1,  2,  5
a():   2,  3,  ?,  1,  4,  ?, ...
```


Vector Operations

• **Mask** — either/or selection of data from two vectors

```
do j=1,L
    if( test(j) ) then
        c(j) = a(j)
    else
        c(j) = b(j)
    endif
enddo
```

```
test():  T,  T,  F,  F,  T
a():     0,  1,  2,  3,  4
b():     5,  6,  7,  8,  9
c():     0,  1,  7,  8,  4
```

• **Compressed Index Generation** — find the indices of selected items in a vector

```
k = 0
do j=1,L
    if( test(j) ) then
        k = k + 1
        indx(k) = j
    endif
enddo
```

```
test():  T,  T,  F,  F,  T
k:       3
indx():  1,  2,  5
```

The Tally Loop (scalar)

- Indexed accumulation

```
do j=1,L
    sum( i(j) ) = sum( i(j) ) + r(j)
enddo
```

- Used to tally particle scores into bins, for overall results
- Tally operations account for 1-10% of time
- **Not** readily vectorized (some tricks for Cray-C90)

Writing Efficient Vector Coding

- Clean loops — structure & indent (Good-looking code runs faster!)
- Innermost loop should be the vector loop
- Avoid IF tests, unless strictly "either/or"
- Never use "GO TO" statements
- No subroutine calls
- No user-defined function calls
- No recursion (ie, forward-stores or backward-fetches)
- Timing estimates:
  - — Count **all** operations inside loop, including **both** branches for IF's.
  - — Multiply by vector length & clock cycle time.
  - — Measure. If much different from estimate, find out why !

Amdahl's Law

- If a computation has fast (vector) & slow (scalar) components, the overall calculation time will be dominated by the slower component

- Speedup = $\frac{1}{(1-f)+f/R}$, where f = fraction vectorized, R = max speedup from vector

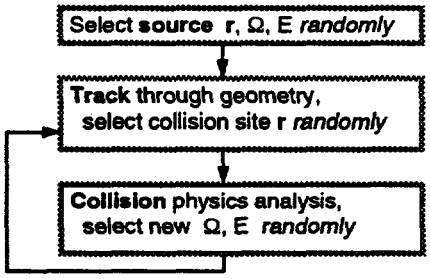
for R = 10				for R = ∞			
f	S	f	S	f	S	f	S
20%	1.2	90%	5.3	20%	1.3	90%	10
40%	1.6	95%	6.9	40%	1.7	95%	20
60%	2.2	99%	9.2	60%	2.5	99%	100
80%	3.6	99.5%	9.6	80%	5	99.5%	200

- For effective vector performance, **must vectorize everything !**

Vectorized Monte Carlo

- **Monte Carlo**

  Simulate neutron behavior by random-walk.



- **Conventional Monte Carlo**

  Analyze many events for one neutron, repeat for other neutrons

- **Vectorized Monte Carlo**

  Analyze many neutrons for one event, repeat for other events

  ⇒ **Event-based algorithms** developed by Kalos, Brown/Martin, Bobrowicz

Monte Carlo is difficult to "vectorize"

- Branching, data retrieval, & arithmetic operations vary for each particle, depending on location, type of collision, code options, etc.
- Typically, ~ 1/3 of essential Fortran statements are IF-tests, which inhibit vectorization
- Not useful: — "automatic vectorizers"
  — syntactic hand-vectorization by programmers

  [In early 1980s, LANL tried each approach with ***mcnp*** → 2X ***slower***]

Method for Vectorizing Monte Carlo

1. Use supercomputer with vector hardware for data-handling
2. Deliberate & careful development
3. Restructure the database
4. Restructure & rewrite the Monte Carlo code

Method for Vectorizing Monte Carlo

1. Use supercomputer with vector hardware for data-handling
   - Only ~40% of operations are floating-point arithmetic (*, +,-, /, sqrt)
   - 40-60% of operations involve data-handling, indexing, selection, ...
   - Must have hardware support for data-handling (gather, scatter, mask, compressed index, ...)
2. Deliberate & careful development
   - Start small, with few options
   - No committees !!!
   - Focus effort on total vectorization
   - Build gradually, restructuring as needed for new features
   - Debugging is extremely difficult — test everything, separately & integrated
3. Restructure the database
   - Unified data formats, with no special cases
   - Arrange for simple & logical direct addressing using vector gather operations
   - Use some new (but equivalent) physics, if necessary
4. Restructure & rewrite the Monte Carlo code
   - "Top-down" development, based on **event-driven algorithm**
   - Use some new (but equivalent) physics, if necessary
   - Avoid rejection methods for random sampling
   - "Vectorize" the IF-tests by data motion, extra computation, or new algorithms

Vectorizing IF-tests

In Monte Carlo codes, IF-tests arise in the context of:

implicit loops, conditional coding, code options

Implicit loops

- Logic of the form "loop UNTIL ....."
- Usually coded as "IF ..... GOTO" & backward branch, instead of "DO ....."
- Number of passes is generally not known in advance
- Some particles satisfy the exit conditions on first pass, others take many passes
- Vectorize by:
  - — Data motion — rearrange the particle data after each pass (eg, event-driven algorithm)
  - — Extra computation — dummy ops on "finished" particles till all are done
  - — Different math/physics — eliminate implicit loop (eg, direct sampling instead of rejection)

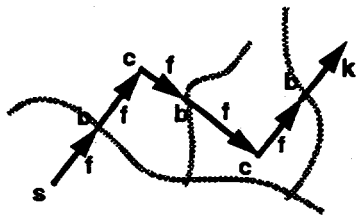
Conditional coding

- Selective operations of some particles, but not others
- Vectorize by:
  - — Gather / Operate / Scatter
  - — Rearrange selective ops into series of "either/or" ops using vector masks
  - — Generalized equations, without special cases

Code options

- Easy — one test/branch for all particles

Monte Carlo — Vectorization



Conventional Algorithm

history	*event sequence*
1	s f c f b f b f c f b k
2	s f b f c f c f b f b f b k
3	s f b f b f b k
4	s f b k
5	s f c f c f c f b f b f c f b f b k

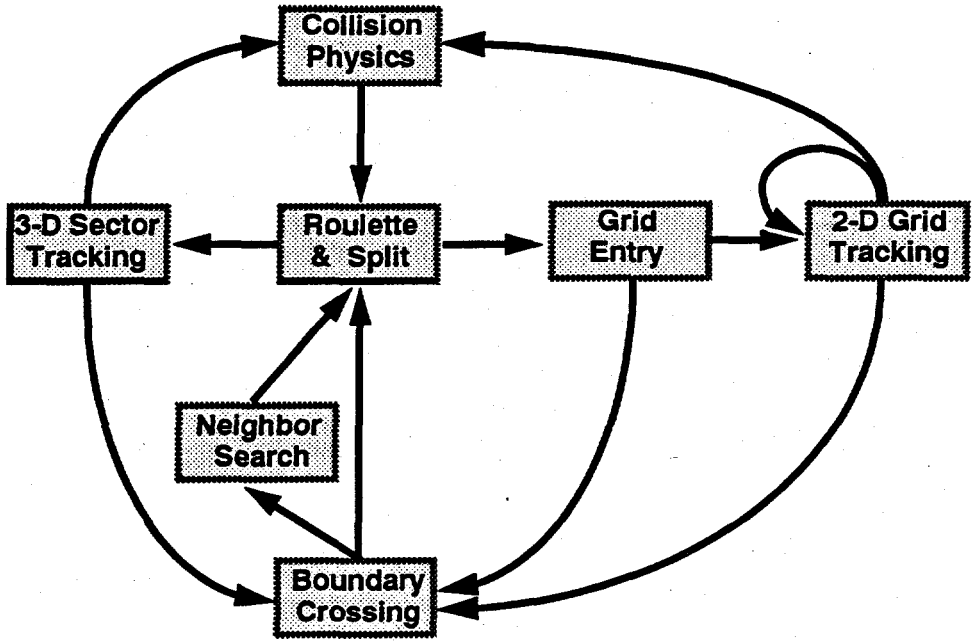
Event-Based Algorithm

history	*event sequence*
1	s f - c f b - f b - f - - - - c f b k
2	s f b - f - c f - c f b f b f - - b k
3	s f b - f b - f b - - - - - - - - - k
4	s f b - - - - - - - - - - - - - - - k
5	s f - c f - c f - c f b f b f c f b - f b k

vector event sequence

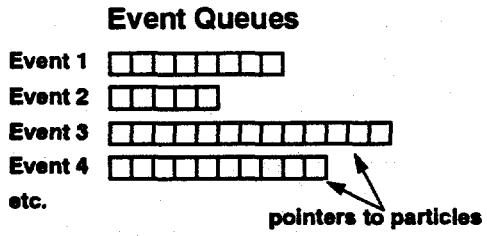
s f b c f b c f b c f b f b f c f b k f b k

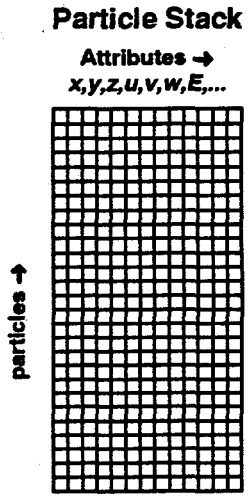
RACER Computational Events



Monte Carlo — Vectorization

Event-Driven Algorithm:

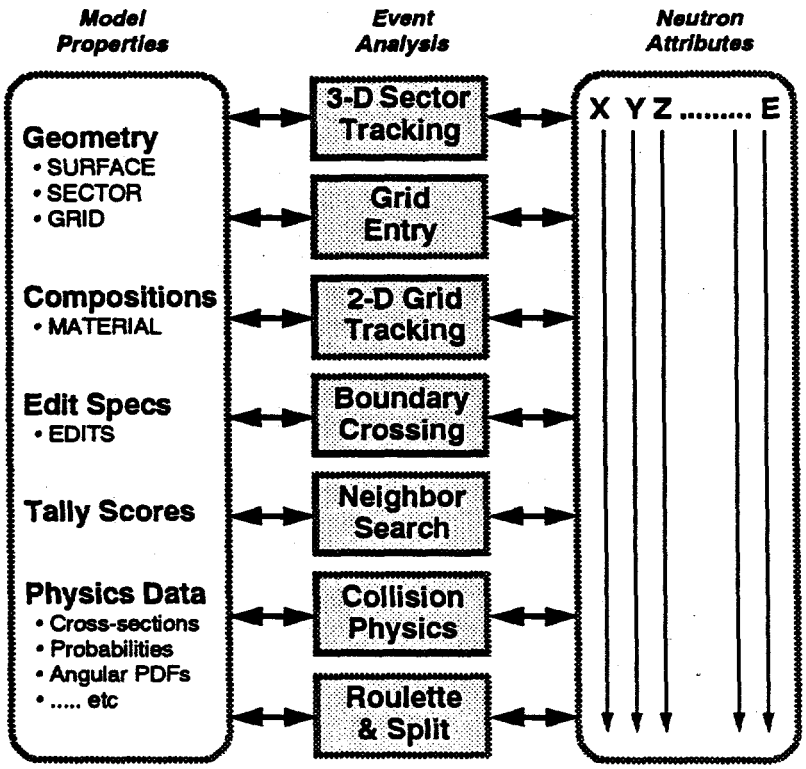




while events are pending:

- — **select event** (1,2,3,...) with largest event-queue
- — **execute the event:**
  - ***Pull*** N ***pointers*** from event-queue
    - ***Gather attributes*** for N particles from stack
      - ***Analyze*** event — vector calculation
    - ***Scatter*** modified ***attributes*** to stack
  - ***Push pointers*** onto next event-queue
- • •

RACER — Code Organization




```
* Loop over timesteps
•   * Loop over batches
•   •   → select starters
•   •   * Loop over edit-groups
•   •   •   → clear edit-group tallies
•   •   •   * Loop over super-groups
•   •   •   •   → get σ's, f(μ)'s, .....
•   •   •   •   → clear event-queues
•   •   •   •   → push to event-queues:          pointers to neutrons in supergroup
•   •   •   •   * Loop until event-queues are empty
•   •   •   •   •   → select event with longest pending queue
•   •   •   •   •   → pull from event-queue:     pointers to neutrons
•   •   •   •   •   → gather:                    needed neutron attributes
•   •   •   •   •   → analyze event & tally
•   •   •   •   •   → scatter:                   updated neutron attributes
•   •   •   •   •   → push to next-event queue:  pointers to neutrons
•   •   •   •   •••
•   •   •   •••
•   •   •••
•   •   → update eigenvalue, results, stats
•   •••
•   → depletion
•••
```

Monte Carlo — Vectorization

Status

- Vectorized Monte Carlo, with event-driven algorithms, was proven to work effectively in ~1980 on Cray-1 & Cyber-205
- **Large speedups** (20x or more) were demonstrated in production Monte Carlo codes, on real problems
- Relatively easy to migrate to MIMD or mixed MIMD/SIMD architectures
- **Very few** large production codes have adopted this approach

What's the problem ?

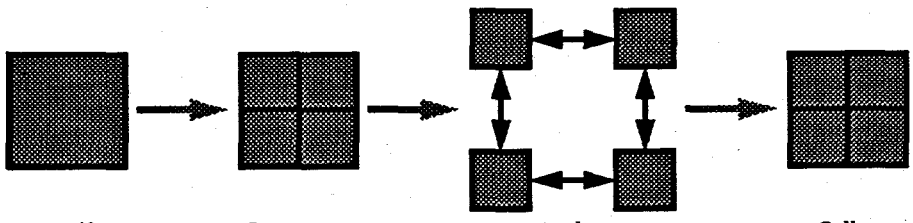
- Must restructure the entire database & rewrite the entire code
- Large amount of people-time → expensive
- "Why change something that works?" → QA work

Parallel Programming

SIMD Machines

- Fine-grained parallelism, low-level
- Vector algorithms & programming

MIMD Machines & Distributed Systems



- Coarse-grained parallelism, high-level
- Ideal for loosely-coupled machines & message-passing libraries

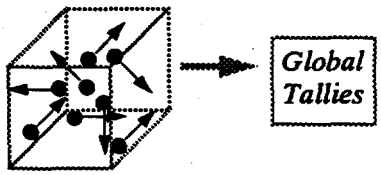
pvm, p4, MPI, express, lam, parmacs, ...

Physical Problem

- 3-D geometry, continuous-energy physics
- ***vim*** — ANL, ***racer*** — KAPL, ***mcnp*** — LANL

Conventional Solution Algorithm

- Random walk for neutrons
- Tally events of interest

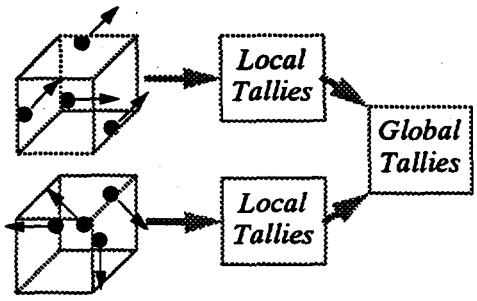


Parallel Algorithm

- Distribute neutrons to different processors
- Local tallies on each processor
- Combine local tallies into global results

Status

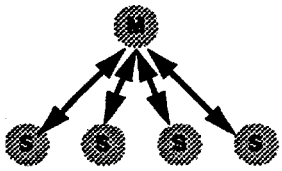
- ***racer*** — in production use,
  Cray-C90, Cray-YMP, Meiko-CS1
- ***vim*** — in testing,
  workstation network & IBM-SP1
- ***mcnp*** — in production use,
  workstation network, Cray-YMP



Monte Carlo — Parallel

Master-slave approach

- Master
  - — Control
  - — All I/O
- Slaves
  - — computations



Master process

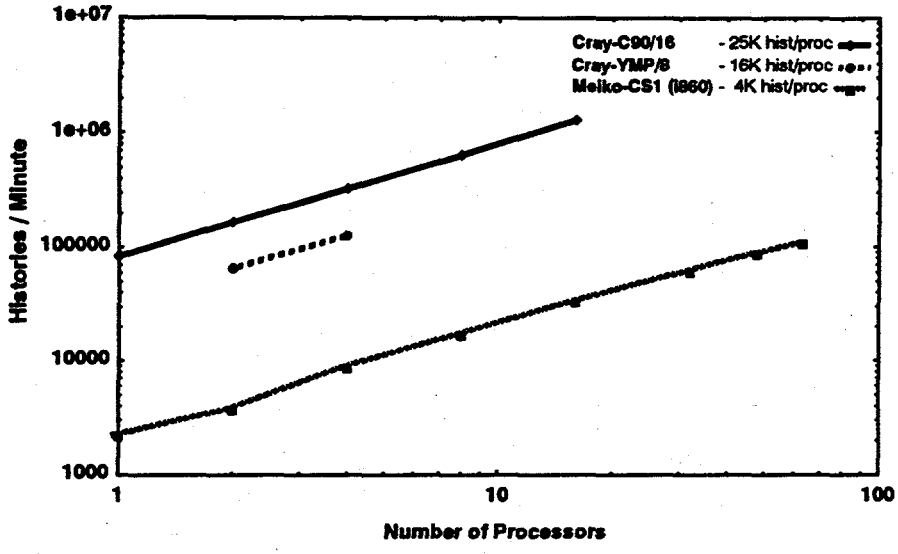
```
* Loop over timesteps
•    * Loop over batches
•    •    → select starters
•    •    * Loop over chunks
•    •    •    → find a ready slave
•    •    •    → send starters to slave
•    •    •••
•    •    * Loop over slaves
•    •    •    → collect slave tallies
•    •    •••
•    •    → update eigenvalue, results, stats
•    •••
```

Slave process

```
* Loop forever
•    → initialize tallies
•    * Loop over chunks, till done
•    •    → tell master: ready for work
•    •    → receive starters from master
•    •    * Loop over histories in chunk
•    •    •    → random walk for 1 history
•    •    •••
•    •••
•    → send tallies to master
•••
```

racer — Parallel / Vector Performance

- Measured performance → histories / minute
- 3-D full-core PWR test problem
- Fixed number of histories/processor
- **Range in performance spans 1000x** [8 hr Cray-C90] ~ [1 yr workstation]



Vector & Parallel Monte Carlo — Issues

- **Hierarchical parallelism**
- **Shared memory vs distributed memory**
- **Parallel speedup & scaling**
- **Software & Portability**
- **Reproducibility**

Scalar Monte Carlo

Loop over ***batches***
•
• Loop over ***particles***
• •
• • **analyze many events**
• • **for 1 particle history**
• •••
•••

Parallel Scalar Monte Carlo

Loop over ***batches***
•
• Loop over ***particles***, N at a time
• •
• • **scalar scalar ... scalar**
• • **cpu-1 cpu-2 ... cpu-N**
• •••
•••

parallel ⇒ 1 particle per CPU,
scalar analysis

high-level: parallel
low-level: scalar

Vector Monte Carlo

Loop over ***batches***
•
• Loop over ***events***
• •
• • **vector analysis of 1 event**
• • **for many particle histories**
• •••
•••

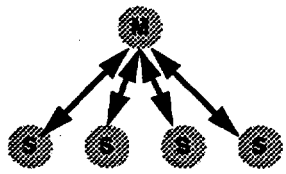
Parallel Vector Monte Carlo

Loop over ***batches***
•
• Loop over ***events***
• •
• • **vector vector ... vector**
• • **cpu-1 cpu-2 ... cpu-N**
• •••
•••

parallel ⇒ many particles per CPU
vector ⇒ events on each CPU

high-level: parallel
low-level: vector

RACER parallel algorithm



✔ **High-level:**

- independent tasks + **message-passing**
- distribute histories among processors
- Master / Slave algorithm
  - — Master: control, distribute work, collect results
  - — Slaves: compute particle histories, no communication with other slaves

(next) **Mid-level:**

- independent tasks + shared memory
- **"macro-tasking"**
- several slaves share memory, take turns on "critical regions"

✘ **Low-level:**

- **"microtasking"**
- split each DO-loop into pieces, compute, synchronize

✔ **Low-level:**

- **vectorization**, within each slave process
- Event-based algorithm (Brown/Martin, 1981)
  - — vectorize events independently (collision, 3D flight, boundary, ...)
  - — create & manage queues of particles waiting for each event

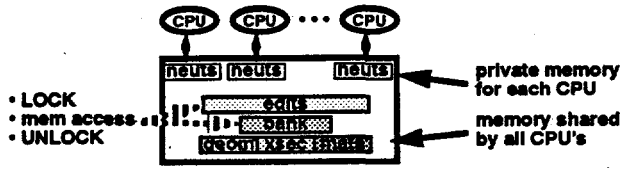
Shared memory usage

Algorithm decisions: Private data vs. Shared data

Issues: overall memory size, data coherency, memory contention, lock/unlock overhead, portability

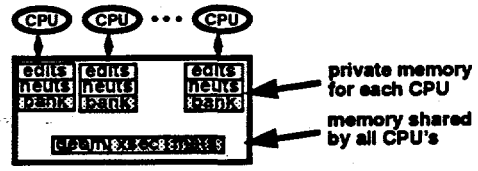
RACER — 1986

- benchmarks
- Cray-XMP



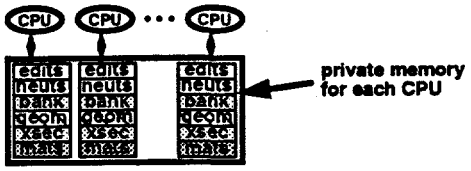
RACER — 1987

- Cray-XMP



RACER — 1988...today

- Cray-YMP
- Cray-C90
- super-WS



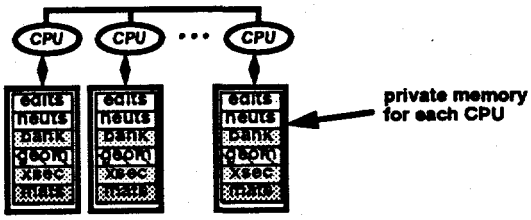
Distributed memory usage

Algorithm decisions: private data (only)

Issues: local memory size

RACER — 1989...today

- Meiko CS1, 2
- WS cluster



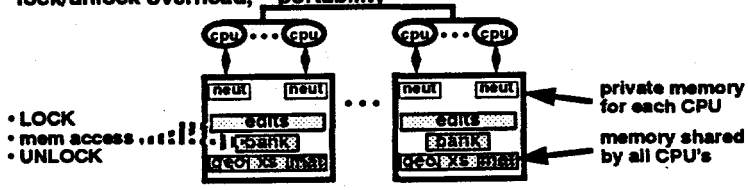
Clustered Shared-Memory

Algorithm decisions: Private data vs. Shared data

Issues: overall memory size, data coherency, memory contention, lock/unlock overhead, portability

RACER — (soon)

- Next super
- Cray-C90
- cluster of super-WS
- cluster of anything



Performance Measurements

- **Metrics**

  Speedup $S_N = T_1 / T_N$ N = # processors

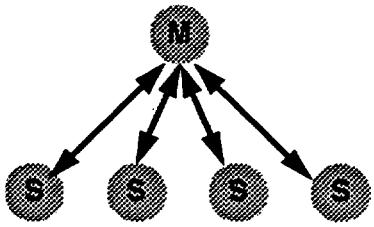
  Efficiency $\eta_N = S_N / N$

- **Fixed Overall Work**
  - — Efficiency decreases with **N**
  - — Speedup (eventually) drops
  - — Example: constant — # histories/batch
    variable — # histories/processor (~1/N)

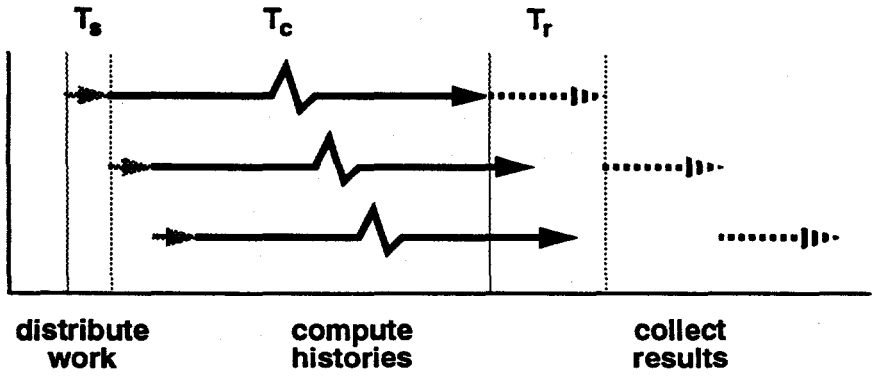
- **Fixed Work per Processor**
  - — Efficiency approx. constant with **N**
  - — Speedup approx. linear with **N**
  - — Example: variable — # histories/batch (~N)
    constant — # histories/processor

    ➜ called "*scaled speedup*"

Master / Slave Algorithm

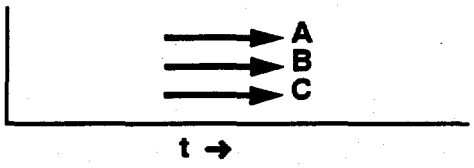


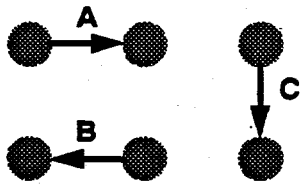
- **Master**
  - — control
  - — distribute work
  - — collect results
- **Slaves**
  - — compute particle histories
  - — no communication with other **slaves**



"Parallel" Message Passing

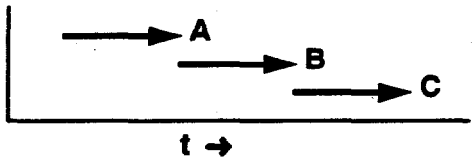
Most parallel computers support concurrent message-passing between separate pairs of nodes

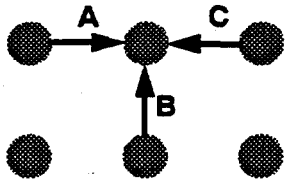




"Serial" Message Passing

But, multiple messages to a single node are (almost always) handled sequentially





For a given physical problem,

- computation time ~ number of histories (M)
- Define:

M_1 = number of histories in job (fix.src.) or batch (eig.) for calculation using 1 slave

N = number of slave processes

M_N = number of histories per slave in job (fix.src) or batch (eig.) using N slaves

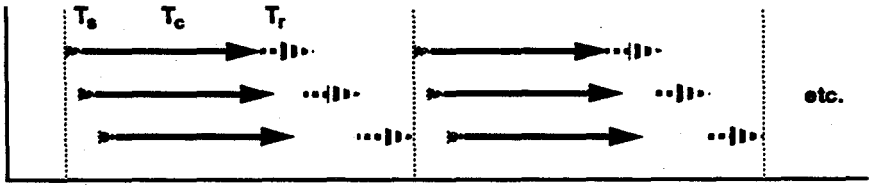
T_N = total time required for M_N histories = $T_s + T_c + T_r$

"Fixed Size" Problem: $M_N = M_1 / N$

- Constant number of particles in job (fix.src.) or batch (eig.)
- Goal of parallel calculation: ***same work, less time***

"Scaled" Problem: $M_N = M_1$

- Constant number of particles per slave
- Goal of parallel calculation: ***more work, same time***



t_h = cpu time per history,
depends on: physics, geometry, Fortran compiler, machine speed & architecture, computer coding, random straggling, etc.

L = amount of tally data per slave, proportional to # regions

$T_s \approx 0$ negligible — send coordinates to slaves

$T_r \approx s + L/r$ s = latency, r = streaming rate

$T_c \approx M_N t_h$

$T_c^{fix} = M_1 t_h / N$ (fixed size)

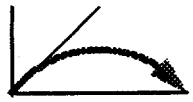
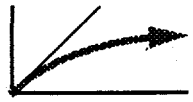
$T_c^{scale} = M_1 t_h$ (scaled)

Scaling Models for Parallel Eigenvalue Calculations

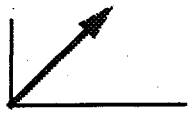
Problem Size	*Message Passing*	*Parallel Speedup*	
fixed		$S_{fix} = T_1 / T_N^{fix}$	
"	serial	$S_{fix,ser} = T_1 / (0 + T_1/N + N T_r)$	$= N / (1 + cN^2)$
"	parallel	$S_{fix,par} = T_1 / (0 + T_1/N + T_r)$	$= N / (1 + cN)$
scaled		$S_{scale} = N T_1 / T_N^{scale}$	
"	serial	$S_{scale,ser} = N T_1 / (0 + T_1 + N T_r)$	$= N / (1 + cN)$
"	parallel	$S_{scale,par} = N T_1 / (0 + T_1 + T_r)$	$= N / (1 + c)$

$$c = (s + L/r) / (M_1 t_h)$$

Scaling Models for Parallel Eigenvalue Calculations

problem size	*communications*	*Speedup*	
fixed	serial	$S = N / (1 + cN^2)$	
fixed	parallel	$S = N / (1 + cN)$	
scaled	serial	$S = N / (1 + cN)$	
scaled	parallel	$S = N / (1 + c)$	

Scaling Models for Parallel Fixed-source Calculations

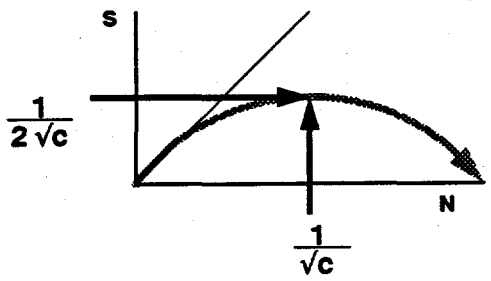
for long calculations: $S \sim N$ 

N = number of slaves
$c = (s + L/r) / (M_1 t_h)$

Parallel Eigenvalue Calculations

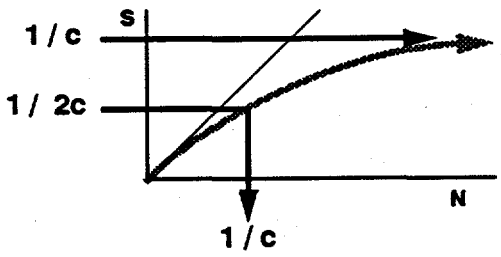
Fixed size, serial messages

$S = N / (1 + cN^2)$



Scaled size, serial messages

$S = N / (1 + cN)$



N = number of slaves
$c = (s + L/r) / (M_1 t_h)$

Parallel Eigenvalue Calculations — scaled size, serial messages

$$S = N / (1 + cN)$$

$$S_{max} = 1 / c$$

$$N_{1/2} = 1 / c$$

N = number of slaves
$c = (s + L/r) / (M_1 t_h)$

Examples:

- **VIM, TREAT problem**

Sun Sparc2 workstation cluster	c=.0043	$S_{max} = 233$
rs6000/350 workstation cluster	c=.011	$S_{max} = 93$
SP1, using ethernet	c=.014	$S_{max} = 70$
SP1, using EUIH comm.	c=.00134	$S_{max} = 748$

- **RACER, "typical" large problem**

100 K histories/min, 20 K histories/slave
32 MB tally data, r ~ 1800 MB/sec

Cray-C90, using SSD for messages (16 processors, max)	c~.001	$S_{max} \sim 1000$

Software & Portability Issues

Portability

- Best bets (for now) → Fortran-77 + C + message-passing
  or → Fortran-77 + C++ + message-passing
- Maybe → Fortran-90 + C
- Gamble → vendor-specific languages, new languages
- Not likely → "automatic" parallelizers

"Standard" message-passing packages

- ***pvm*** → from Oak Ridge & Univ. Tennessee
- ***mpi*** → "Message-Passing Interface", draft standard
- ***p4*** → from Argonne
- ***express*** → commercial product, Parasoft

Performance using distributed computing

- Minimize communications
- Minimize disk I/O (master only ???)

Parallel Monte Carlo — Reproducibility

Eigenvalue Calculations

- **Power iteration**
  - ➜ Histories **within** a batch are independent— analyze in parallel
  - ➜ Successive batches are **not** independent— analyze sequentially
- **Large batch size**
  - ➜ maximize parallel efficiency & performance
  - ➜ Side effect: **reduced bias** in M.C. eigenvalue & shapes

Reproducibility

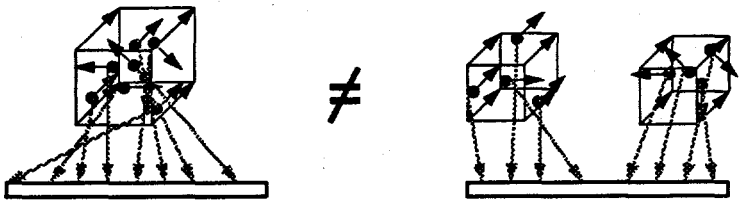
- Must get identical results (bit-for-bit) using 1 or many processors
- A solved problem, see

  "Reproducibility & Monte Carlo Eigenvalue Problems,"
  F. B. Brown & T. M. Sutton, *Trans. Am. Nucl. Soc.* 65, 235 (June 1992)

  - ➜ special random number generators
  - ➜ reorder fission-site bank at end of batch

Reproducibility & Monte Carlo Eigenvalue Calculations

- Difficulties: — Random number usage by each particle
  — Ordering of particles in "fission bank"

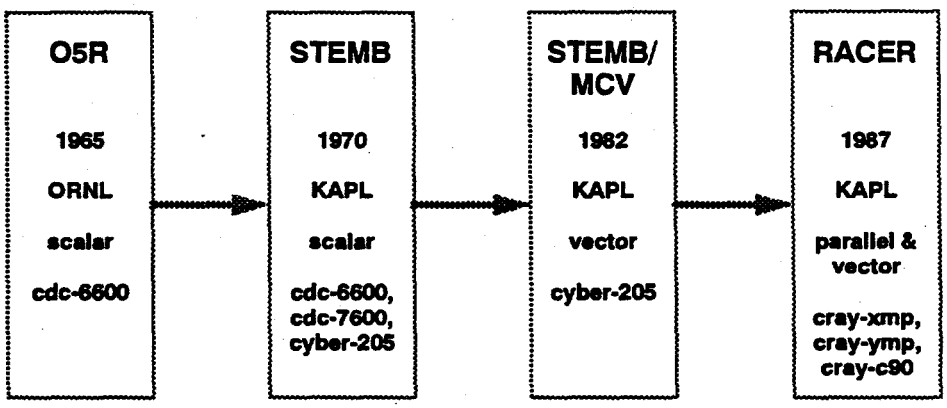


- Require ***identical*** results if calculation repeated
  (with same random numbers)
  - — with different ***number of processors***
  - — with different ***vector length*** limits
  - — with different ***"supergrouping"*** of cross-section data
  - ➜ ***use a separate random seed for each particle***

    | ← Seeds for particle k → | ← Seeds for particle k+1 → | ← Seeds .....
  - ➜ ***unique reordering of fission bank at end of each batch***

    Brown & Sutton, Trans. Am. Nucl. Soc. 65, 235 (1992)

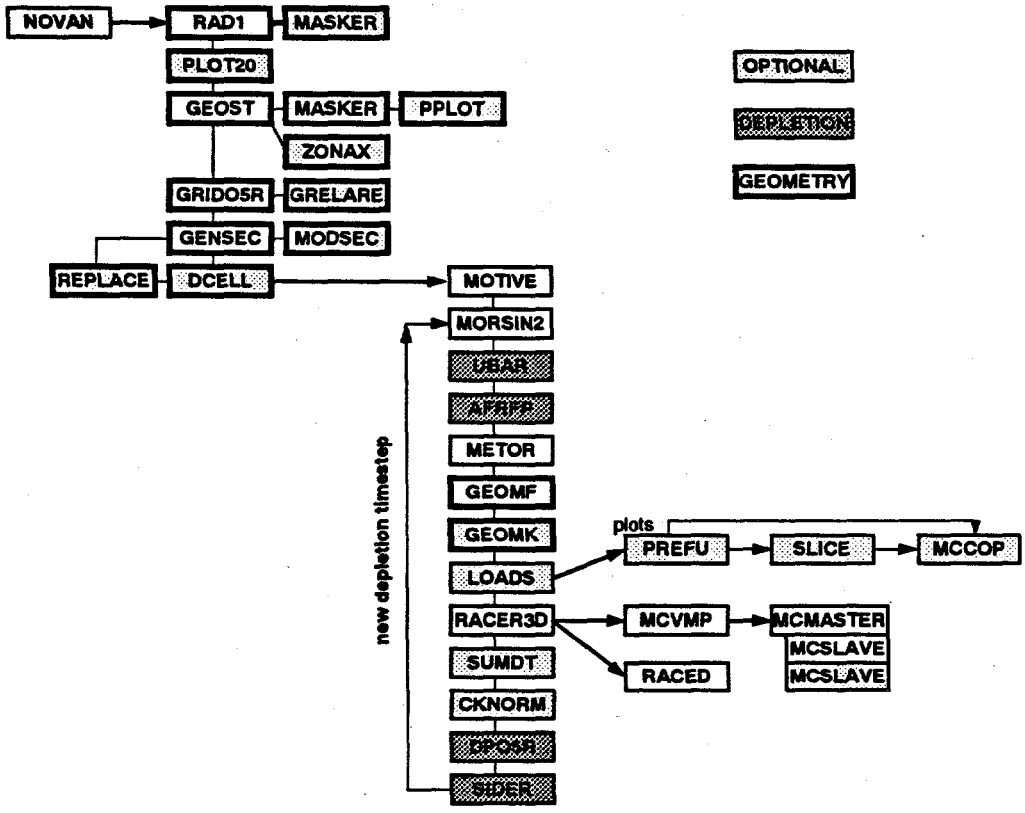
RACER Monte Carlo ⇒ 3-D reactor physics analysis



- Extensively benchmarked vs experiment & real reactors
- Most-used KAPL code: ~30-40% of all Cray time

Focus for RACER development:

⇒ very large & detailed 3-D reactor physics problems



Module Development History (dates are very approximate)

Datapool:	1965	Kazmierczak, Warrington, ..., Austin	
Datatran:	1965	Kopp, Selengut, Schilling, ... (Cecil invented it)	
O5R:	1965,	Coveyou, Irving, Freestone, Kam (ORNL) (**scalar** M.C.)	
RAD1:	1970	Beam, Kopp, Crawford,.....	
EVDF:	1970	Reynolds, Lubitz, Stieglitz, ...	
STEMB:	1972,	Speers, Tuecke, Ellis, MacMillan, Bischoff, Kalos, Cady, Nelkin, Mendelson, Howe, ..., Austin, Kudlacik, Muro, Hurley, Schindler, ...	
MCV:	1982,	Brown	(2-D, grid, **vector** M.C.)
MCVLS:	1983,	Brown	(multiple 1-D problems, LF/SSF)
RACER2D:	1984,	Brown	(cross-sections & depletion)
RACER3D:	1986,	Brown	(3-D general geometry)
RACER:	1988,.....	Brown, Bischoff, Sutton, Ballinger, Schindler, Kelly	

NOVAN — initialize Datapool & read thermal multigroup cross-sections

RAD1 — 2-D geometry setup

PLOT20 — 2-D pictures

GEOST — construct 3-D geometry from 2-D RAD1 model (MASKER, PPLOT, ZONAX)

GRIDO5R — process 2-D RAD1 geometry into "grids" (GRELARE)

GENSEC — 3-D surface equations & sector definitions (MODSEC, REPLACE)

DCELL — subdivide 2-D M.C. geometry & edits to match NOVA

MOTIVE — prepare description of cross-section data, or TAPE9/TAPE3 libraries

MORSIN2 — assign edit media, compositions, importances, source, ...

UBAR — linkage from NOVA depletion to Monte Carlo

AFRFP — setup lumped multigroup background fission product cross-sections

METOR — setup energy mesh, edits (isotopic, comb, ratio, ...), edit groups, & misc. input

GEOMF — reformat 3-D geometry data

GEOMK — compute areas & volumes

RACER Monte Carlo System — Modules

[209]

LOADS	— edit material loadings
PREFU	— collect data for plotting
SLICE	— 2-D slice plots, line drawings
MCCOP	— 2-D slice plots, color
RACER3D	— collect & reformat data for Monte Carlo
MCVMP	— start the Monte Carlo programs
MCMASTER	— master process for Monte Carlo
MCSLAVE	— slave process for Monte Carlo
RACED	— edit the results from Monte Carlo
CNVRG	— convergence plots, selected data vs batch
SUMDT	— retrieve selected results into Datatran lists
CKNORM	— detailed comparison of Monte Carlo & diffusion results
DPO5R	— linkage from Monte Carlo to NOVA depletion
SIDER	— depletion

Conclusions

[210]

- **Parallel Monte Carlo codes (mcnp4a, keno, vim, racer, ...) are now running on many parallel computers (cray, meiko, intel, convex, cm5, ...) & workstation networks**

- **Master/slave algorithms are simple, easy to implement, & scale well for eigenvalue problems.**

- **Communications bottlenecks at the master process are <u>not</u> a problem today:**

  - **— ethernet is fast enough for 10's of slaves**
  - **— FDDI, EUIH, & other schemes should permit 100's of slaves**

- **The major limitation on parallel Monte Carlo today appears to be memory size — each node must contain entire problem & tallies**

Vector & parallel computing have side-benefits:

- Larger batches + more batches → • reduces convergence problems
  • reduces bias
  • better correlation corrections
- "Unthinkable" problems become routine

Proven Monte Carlo algorithms exist for

- SIMD, MIMD, & mixed MIMD/SIMD supercomputers & MP computers
- MIMD distributed processing on a network of machines

Parallel algorithms which scale to 1,000's or 1,000,000's of nodes

- "master-slave" algorithm on 1,000,000 Internet nodes could take ~ 1 year to start
- UNIX socket connections — limits master to 10's or 100's of nodes

→ develop hierarchical parallelism, "clusters-of-clusters"
→ parallel histories + geometric decomposition (?)

Algorithms with load-balancing & fault-tolerance

- "Virtual supercomputer" can change continuously

→ recover from lost nodes & hardware failures
→ dynamic load balancing
→ cooperate with distributed queuing systems

Acceleration for eigenvalue calculations

→ automated procedures for discarding initial batches
→ importance sampling or "fission matrix"

Methods for eliminating bias

Improved methods for estimating variance, corrections for correlation

References

General References on Monte Carlo Methods for Particle Transport Problems

[1] L. L. Carter and E. D. Cashwell, Particle Transport Simulation with the Monte Carlo Method, ERDA Critical Review Series, TID-26607, National Technical Information Service, Springfield MA (1975).

[2] E. D. Cashwell and C. J. Everett, A Practical Manual on the Monte Carlo Method for Random Walk Problems, Pergamon Press, London (1959).

[3] J. J. Duderstadt and W. R. Martin, Transport Theory, John Wiley & Sons, NY (1979).

[4] G. Goertzel and M. H. Kalos, "Monte Carlo Methods in Transport Problems," in *Progress in Nuclear Energy, Series I, Physics and Mathematics*, Vol. 2, (1958).

[5] J. H. Halton, "A Retrospective and Prospective Survey of the Monte Carlo Method," *SIAM Rev.* **12**, 1, (1970).

[6] J. M. Hammersley and D. C. Handscomb, Monte Carlo Methods, John Wiley & Sons, NY (1964).

[7] M. H. Kalos and P. A. Whitlock, Monte Carlo Methods, Volume I: Basics, John Wiley & Sons, NY (1986).

[8] E. E. Lewis and W. F. Miller, Jr., Computational Methods of Neutron Transport, American Nuclear Society, Inc., LaGrange Park, IL (1993).

[9] I. Lux and L. Koblinger, Monte Carlo Particle Transport Methods: Neutron and Photon Calculations, CRC Press, Ann Arbor, MI (1991).

[10] S. Nakamura, Computational Methods in Engineering and Science, R. E. Krieger Pub. Company, Malabar, FL (1986).

[11] R. Y. Rubinstein, Simulation and the Monte Carlo Method, John Wiley & Sons, NY (1981).

[12] Y. A. Schreider, The Monte Carlo Method, Pergamon Press, NY (1966).

[13] J. Spanier and E. M. Gelbard, Monte Carlo Principles and Neutron Transport Problems, Addison-Wesley, Reading, MA (1969).

[14] J. Wood, Computational Methods in Reactor Shielding, Pergamon Press, Oxford (1982).

General References on Random Number Generation and Random Sampling Methods

[15] L. Devroye, Non-Uniform Random Variate Generation, Springer-Verlag, NY (1986).

[16] C. J. Everett and E. D. Cashwell, "A Third Monte Carlo Sampler," LA9721-MS, Los Alamos National Laboratory, Los Alamos, NM (1983).

[17] H. Kahn, "Applications of Monte Carlo," AECU-3259, Rand Corporation, Santa Monica, CA (1954).

[18] D. E. Knuth, The Art of Computer Programming, Vol. 2: Semi-numerical Algorithms, Addison-Wesley, Reading, MA (1981).

[19] E. J. McGrath and D. C. Irving, "Techniques for Efficient Monte Carlo Simulation," ORNL-RSIC-38, Vols. I-III, Oak Ridge National Laboratory, Oak Ridge, TN (1975).

General References on Monte Carlo Methods for the Solution of Reactor Eigenvalue Problems

[20] R. C. Gast and N. R. Candelore, "Monte Carlo Eigenfunction Strategies and Uncertainties," in *Proc. NEACRP Meeting of a Monte Carlo Study Group*, ANL-75-2, Argonne National Laboratory, Argonne, IL (1974).

[21] D. C. Irving, R. M. Freestone, and F. B. K. Kam, "O5R, A General Purpose Neutron Monte Carlo Code," ORNL-3622, Oak Ridge National Laboratory (1965).

[22] M. H. Kalos, F. R. Nakache, and J. Celnik, "Monte Carlo Methods in Reactor Computations," in Computing Methods in Reactor Physics, Gordon and Breach, NY (1968).

[23] J. Lieberoth, "A Monte Carlo Technique to Solve the Static Eigenvalue Problem of the Boltzmann Transport Equation," *Nukleonik* 11, 213 (1968).

[24] M. R. Mendelson, "Monte Carlo Criticality Calculations for Thermal Reactors," *Nucl. Sci Eng.* **32**, 319-331 (1968).

[25] H. Rief and H. Kschwendt, "Reactor Analysis by Monte Carlo," *Nucl. Sci. Eng.*, 30, 395 (1967).

[26] E. R. Woodcock, et al, "Techniques Used in the GEM Code for Monte Carlo Neutronics Calculations in Reactors and Other Systems of Complex Geometry," in *Proc. Conf. Applications of Computing Methods to Reactor Problems*, ANL-7050, Argonne National Laboratory, Argonne, IL (1965).

Open-Literature Publications on the RACER Monte Carlo Code, Vector and Parallel Monte Carlo, and Related Monte Carlo Methods

[27] D. J. Kelly, "Depletion of a BWR Lattice Using the RACER Continuous-Energy Monte Carlo Code," Proceedings of the ANS *Intl. Conf. on Mathematics & Computation, Reactor Physics, & Environ. Analyses*, April 30-May 4, Portland, Oregon (1995).

[28] C. T. Ballinger, "The Direct S(α,β) Method for Thermal Neutron Scattering," Proceedings of the ANS *Intl. Conf. on Mathematics & Computation, Reactor Physics, & Environ. Analyses*, April 30-May 4, Portland, Oregon (1995).

[29] F. B. Brown, "Random Number Generation with Arbitrary Strides," *Trans. Am. Nucl. Soc.* **71**, 202 (1994).

[30] S. Matsuura, F. B. Brown, and R. N. Blomquist, "Parallel Monte Carlo Eigenvalue Calculations," *Trans. Am. Nucl. Soc.* **71**, 199 (1994).

[31] T. M. Sutton and F. B. Brown, "Parallel Monte Carlo for Reactor Calculations," Proceedings of the ANS Topical Meeting on *Advances in Reactor Physics*, April 11-15, 1994, Knoxville, TN (April 1994).

[32] F. B. Brown, K. L. Derstine, and R. N. Blomquist, "Distributed Computing and Nuclear Reactor Analysis," Proceedings of the ANS Topical Meeting on *Advances in Reactor Physics*, April 11-15, 1994, Knoxville, TN (April 1994).

[33] R. N. Blomquist and F.B. Brown, "Parallel Monte Carlo Reactor Neutronics," Proceedings of the Society for Computer Simulation Meeting on *High Performance Computing '94*, April 11-15, 1994, La Jolla, CA (April 1994).

[34] F. B. Brown and J. L. Vujic, "Comparison of Direct and Rejection Sampling Methods," *Trans. Am. Nucl. Soc.* **69**, 223 (Nov. 1993).

[35] E. M. Gelbard, F. B. Brown, and A. G. Gu, "Estimation of Fission Source Bias in Monte Carlo Eigenvalue Calculations," *Trans. Am. Nucl. Soc.* **69**, 201 (Nov. 1993).

[36] F. B. Brown, "Monte Carlo Neutronics Simulations on Parallel and Distributed Computers," invited paper presented at the SIAM Conference on *Simulation and Monte Carlo Methods*, San Francisco (Aug. 1993).

[37] F. B. Brown and T. M. Sutton, "Reproducibility and Monte Carlo Eigenvalue Calculations," *Trans. Am. Nucl. Soc.* **65**, 235 (June 1992).

[38] R. G. Gamino, F. B. Brown, and M. R. Mendelson, "A Monte Carlo Green's Function Method for 3-D Neutron Transport," *Trans. Am. Nucl. Soc* **65**, 237 (June 1992).

[39] W. R. Martin, J. A. Rathkopf, and F. B. Brown, "The Impact of Advances in Computer Technology on Particle Transport Monte Carlo," Proceedings of the ANS Topical Meeting on *New Horizons in Radiation Protection and Shielding*, Richland WA (April 1992).

[40] F. B. Brown, W. R. Martin, and J. A. Rathkopf, "Particle Transport Monte Carlo and Parallel Computers," Proceedings of the *Argonne Theory Institute on Parallel Monte Carlo Simulations*, Argonne National Laboratory (August 1991).

[41] W. R. Martin and F. B. Brown, "Monte Carlo Methods for Particle Transport," *Trans. Am. Nucl. Soc.* **60**, 336 (1989).

[42] F. B. Brown and F. G. Bischoff, "Computational Geometry for Reactor Applications," *Trans. Am. Nucl. Soc.* **57**, 112 (1988).

[43] F. B. Brown, "Present Status of Vectorized Monte Carlo," *Trans. Am. Nucl. Soc.* **55**, 323 (1987).

[44] W. R. Martin and F. B. Brown, "Present Status of Vectorized Monte Carlo for Particle Transport Analysis," *International Journal of Supercomputer Applications*, Vol. 1, No. 2, 11-32 (June 1987).

[45] F. B. Brown, "Vectorization of 3-D General-Geometry Monte Carlo," *Trans. Am. Nucl. Soc.* **53**, 283 (1986).

[46] F. B. Brown and W. R. Martin, "Monte Carlo Methods for Vector Computers," *J. Progress in Nuclear Energy*, Vol. 14, No. 3, 269-299 (1984).

[47] F. B. Brown and M. R. Mendelson, "Vectorized Monte Carlo Applications in Reactor Physics Analysis," *Trans. Am. Nucl. Soc.* **46**, 727 (June 1984).

[48] F. B. Brown, "Vectorized Monte Carlo Methods for Reactor Lattice Analysis," Proceedings ANS Topical Meeting on *Advances in Reactor Computations*, Salt Lake City, Utah, 108-123 (March 1983).

[49] F. B. Brown, "Vectorized Monte Carlo Methods for Reactor Lattice Analysis," KAPL-4163, Knolls Atomic Power Laboratory, Schenectady, NY (March 1983).

[50] F. B. Brown, "Development of Vectorized Monte Carlo Algorithms for Reactor Lattice Analysis," *Trans. Am. Nucl. Soc.* **43**, 377 (1982).

[51] F. B. Brown, "Vectorized Monte Carlo," Ph. D. dissertation, University of Michigan, Ann Arbor, Michigan (1981).

[52] F. B. Brown, W. R. Martin, and D. A. Calahan, "Investigation of Vectorized Monte Carlo Algorithms," *Trans. Am. Nucl. Soc.* **39**, 755 (1981).

[53] F. B. Brown, W. R. Martin, and D. A. Calahan, "A Discrete Sampling Method for Vectorized Monte Carlo Algorithms," *Trans. Am. Nucl. Soc.* **38**, 354 (1981).

[54] F. B. Brown, D. A. Calahan, W. R. Martin, et al, "Investigation of Vectorized Monte Carlo Algorithms" working paper presented at the *DOE Conference on High Speed Computing*, Gleneden Beach, Oregon (April 1981).

[55] C. L. Ellis and D. B. MacMillan, "O5R Users Manual," KAPL-M-6741, National Technical Information Service, 5285 Port Royal Road, Springfield, VA 22161 (1967).

DISCLAIMER