

# Monte Carlo Project

22.901 Introduction to Computer Programming for Nuclear Engineers

January 30 - February 2, 2012

# What is Monte Carlo?

- Is a stochastic method to solve PDEs
- Useful for simulating random events
- Probability distributions capture physics
- Results are reported with a mean and variance

# Neutron Transport Equation

- In its most detailed form (time-independent)

$$\underbrace{\hat{\Omega} \cdot \nabla \varphi(\vec{r}, E, \hat{\Omega})}_{\text{leakage}} + \underbrace{\Sigma_t(\vec{r}, E) \varphi(\vec{r}, E, \hat{\Omega})}_{\text{interactions}} =$$

$$\underbrace{\int_0^{4\pi} d^2\Omega' \int_0^\infty dE' \Sigma_s(\vec{r}, E' \rightarrow E, \hat{\Omega}' \rightarrow \hat{\Omega}) \phi(\vec{r}, E')}_{\text{scattering production}} +$$

$$\underbrace{\int_0^{4\pi} d^2\Omega' \int_0^\infty dE' \nu \Sigma_f(\vec{r}, E' \rightarrow E, \hat{\Omega}' \rightarrow \hat{\Omega}) \phi(\vec{r}, E')}_{\text{fission production}} + \underbrace{q(\vec{r}, E, \hat{\Omega})}_{\text{external source}}$$

- Simplified: 1-D homogeneous slab, 1 energy group, isotropic scattering in LAB, no fission, isotropic uniform source

$$\mu \frac{\partial \varphi}{\partial x} + \Sigma_t \varphi(x, \mu) = \Sigma_s \int_{-1}^1 d\mu' \varphi(x, \mu' \rightarrow \mu) + \frac{Q}{2X}$$

# Neutron Simulation

- Monte Carlo neutronic analyses involve simulating neutrons 1-by-1
- So what happens in the life of a neutron
  - 1 Neutron is born at some location and traveling angle
  - 2 It then transports to location - what can happen?
    - Cross a surface - go back to 2
    - Leak out of system - kill neutron
    - Collide with an isotope
  - 3 If it collides get a collision type
    - Absorbed - kill neutron
    - Scattered - get new angle in LAB
  - 4 If particle is not dead go back to 2

That's it!

# Source Code Layout

- `main.f90` - The main program file
- `execute.f90` - A module where the neutrons life is simulated
- `global.f90` - A module that contains all of the global vars
- `geometry.f90` - A module that contains geometry information and procedures
- `material.f90` - A module that contains material information and procedures
- `particle.f90` - A module that contains particle information and procedures
- `tally.f90` - A module that contains tally information and procedures

Make sure you copy **OBJECTS, MAKEFILE, DEPENDENCIES** and **timing.f90** to your local source directory

## Module - geometry.f90

- define geometry type with the following attributes
  - Number of sub-slabs
  - Length of slab
  - Length of sub-slab
- Contains one procedure - read\_geometry
  - pass it a geometry type
  - Read in two out of the 3 attributes
  - calculate the 3rd attribute

## Module - material.f90

- define material type with the following attributes
  - totalxs - total macroscopic cross section
  - absxs - absorption macroscopic cross section
  - scattxs - scattering macroscopic cross section

$$\Sigma_t = \Sigma_a + \Sigma_s$$

- Contains one procedure - `read_material`
  - pass it a material type
  - Read in each cross section from user

## Module - tally.f90

- define tally type with the following attributes (initialize to 0)
  - c1 - collision accumulator
  - c2 - square of collision accumulator
  - s1 - path accumulator
  - s2 - square of path accumulator
  - smean - mean for tracklength est
  - cmean - mean for collision est
  - svar - variance for tracklength est
  - cvar - variance for collision est
  - track - temp. track var
  - coll - temp. collision var
- Contains the following procedures
  - sub: tally\_reset with argument of a tally\_type
  - sub: bank\_tally with argument of a tally\_type
  - sub: perform\_statistics with arguments of a tally\_type, number of histories, sub-slab width



# Module - particle.f90

- use pdfs (in the static library)
- define particle type with the following attributes
  - slab - the slab id number
  - xloc - the x location of the particle
  - mu - the angle cosine of travel
  - alive - a logical to indicate if the particle is alive
- Contains one procedure - `particle_init`
  - pass it a `particle_type` and length of slab

## Module - global.f90

- use all of the types listed on previous slides
- make sure you indicate that you want to save the variables
- define a geometry (geo), material (mat) and particle (neutron) type
- define an allocatable, 1-D, tally type called tal (it will be n\_slabs in length)
- define a variables for number of histories
- define a timer type see timing.f90
- this module contains two procedures
  - sub: allocate\_problem - no args
  - sub: free\_memory - no args

# Module - execute.f90

- eventually will include a bunch of use commands
- no module-data here
- contains the following procedures
  - sub: run\_problem - no args
  - sub: transport - no args
  - sub: interaction - no args
  - fun: get\_slab\_id - no args
  - sub: reset\_tallies - no args
  - sub: bank\_tallies - no args
  - sub: print\_tallies - no args

# Program - `main.f90`

- Put appropriate **use** commands - will be obvious from below
- Read in number of particles to simulate from user
- Read in geometry - call `read_geometry`
- Read in material - call `read_material`
- begin a timer
- allocate the problem - call `allocate_problem`
- run the problem - call `run_problem`
- stop the timer
- print results - call `printtallies`
- free memory - call `free_memory`
- terminate the program

# Milestone 1

- Develop the file structure for the code
- Follow description of modules on previous slides
- Determine which procedures should be public and which should be private
- Make sure you do PARTIAL INCLUSION
- Goal is to read in the data and print it back out in print\_tallies routine
- Code will go into all subroutines but won't do anything
- Make sure timer works

If you have any trouble getting the code to work come see me at NW12-234 as we will be moving fast!