CEM and LAQGSM Event Generators: Modern Software Development

Leslie Kerby, Ph.D [kerbles@isu.edu]
Chase Juneau [junechas@isu.edu]

XSCRC2019: Cross Sections for Cosmic Rays @ CERN
CERN, Geneva, Switzerland
November 14, 2019

Contents

Contents

Introduction: What is GSM?

- The Generalized Spallation Model (GSM)[1] is an <u>event generator.</u>
  - Simulates high-energy hadron-nucleus and nucleus-nucleus collisions

- Rooted in the Cascade Exciton Model (CEM)[2] and the Los-Alamos Quark Gluon String Model (LAQGSM)[3]

- Latest develops
  - CEM03.03-F (2015)
  - LAQGSM03.03-F_noGPL (2012)

Introduction: What is GSM? (cont'd)

- The CEM[2] simulates spallation events for…
  - Incident neutrons, protons, pions, and photons
  - Incident energies ranging from $\approx 100\ MeV$ to $\approx 4.5\ GeV$
  - Targets larger than ^{4}He

- The LAQGSM[3] simulates spallation events for…
  - Any incident particle (hadron or nucleus)
  - Incident energies ranging from $\approx 5\ GeV$ to $\approx 1\ TeV$, per nucleon
  - Any target nucleus

Introduction: What is GSM? (cont'd)

- GSM predicts emission of particles ranging from neutrons and protons up to ^{28}Mg fragments, excluding residual nuclei

- GSM was built using the pre-existing CEM event generator code
  - Incorporated LAQGSM cascade for appropriate reactions

Introduction: What is GSM? (cont'd)

- The CEM and LAQGSM in MCNPX/6 model progeny creation from high-energy interactions when data is not available

- Consider GSM as "CEM+LAQGSM extended"

- Generalized for…
  - Incident hadron or nucleus ("projectile")
  - Incident energy
  - Target nucleus
  - Software client usage

Introduction: What is GSM? (cont'd)

- Note that particle transport codes rely heavily on tracking progeny creation and on cross section tables

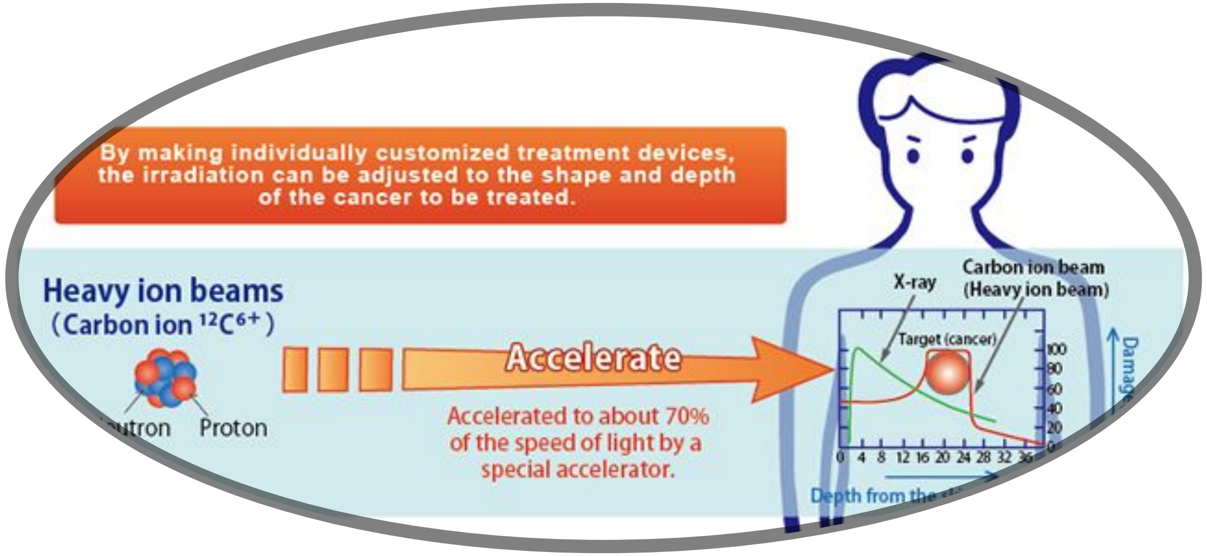
- Particle transport codes primarily use event generators to:
  - Predict spallation progeny for a given collision
  - Predict interaction probabilities above those tabulated

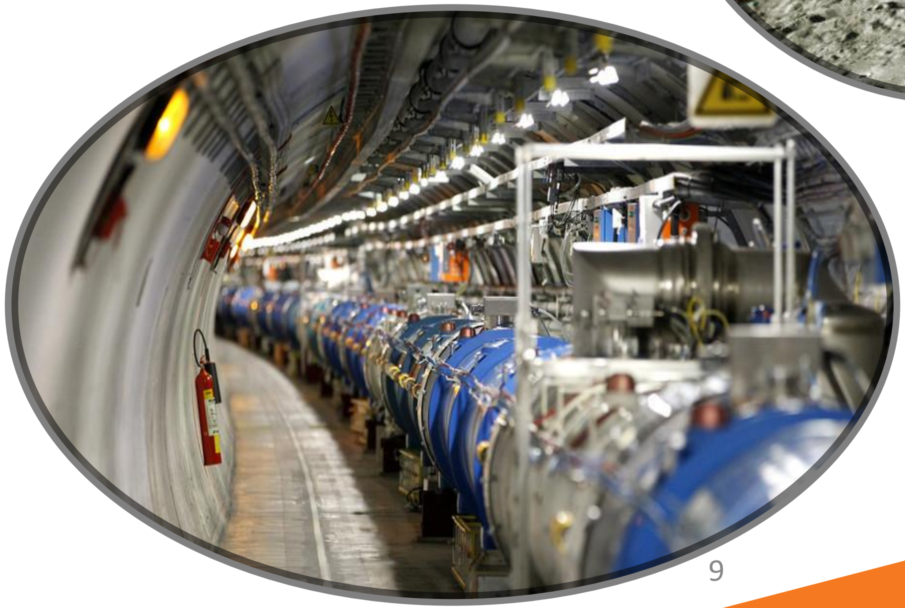
Idaho State University

CAES
Center for Advanced Energy Studies

Introduction: What is GSM? (cont'd)

- Applications are traditionally limited to…
  - Accelerator facilities
  - Medical treatments
  - Space analyses





ROAR

Introduction: Why GSM?

- GSM motivated by an effort to deprecate the LAQGSM[4]
  - LAQGSM written with legacy Fortran standards and is similar to CEM
  - GSM demonstrated potential to deprecate the LAQGSM
- HPC compatibility for scaling and cluster computation
- Work focused on making the GSM current.
  - Reduced technical debt
  - Migrate to a robust software library with explicit functionality

Introduction: Why GSM? (cont'd)

- Modernization emphasized…
  - Removal of implicit variables
  - Usage of modern language syntax and declarations
  - Modular units of functionality
  - Containerization
  - Flexibility
  - Portability
  - Migration to an object-oriented architecture
  - Robust API creation
  - Object/container optimization

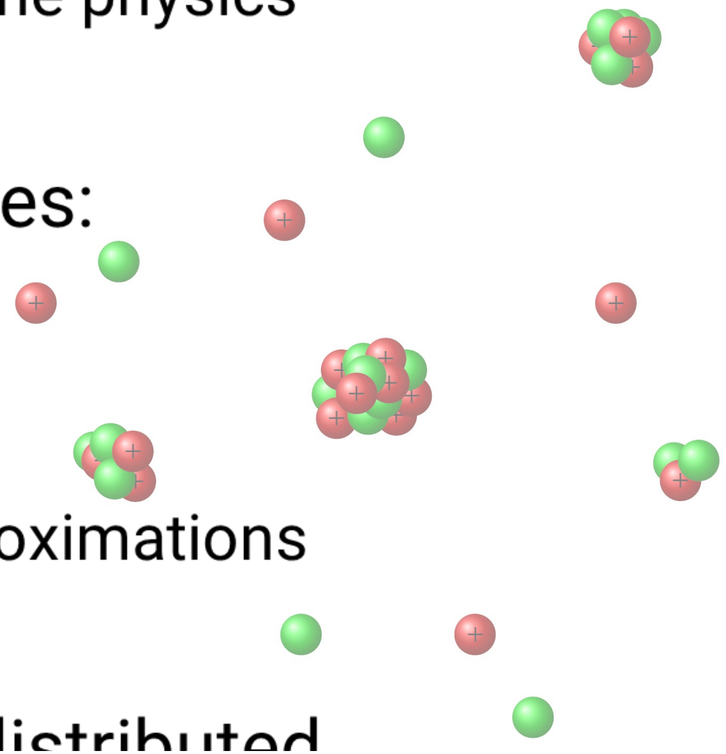
Contents

Spallation Physics Overview

- Spallation is a highly energy-dependent process
  - Large incident energies add complexity to the physics

- GSM[1] considers spallation in three phases:
  - Fast ($\gtrsim 100$ MeV)
  - Intermediate (unstable energies)
  - Compound ($\gtrsim 30\ MeV$)
  - Energy regimes are based on physical approximations

- Events are independent and identically distributed



Spallation Physics Overview (cont'd)

- Fast stage consists of the IntraNuclear Cascade (INC)[1]
  - Standard and Modified Dubna Cascade Models (DCM) available

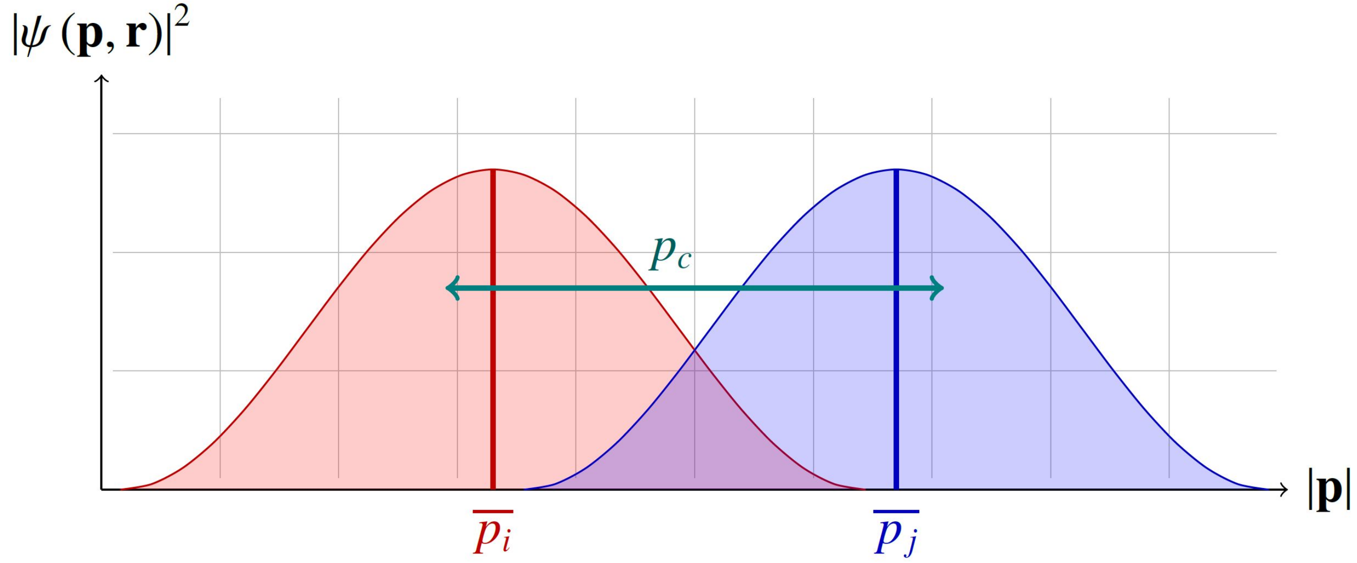
- Intermediate stage consists of a preequilibrium model[5]
  - Equilibration of the INC process's residual nucleus

- Compound stage models evaporation and fission of the compound nucleus[6]

Spallation Physics Overview (cont'd)

- High energy requirements are a function of the physical assumptions made

- Coalescence[7] of only INC progeny is also modeled



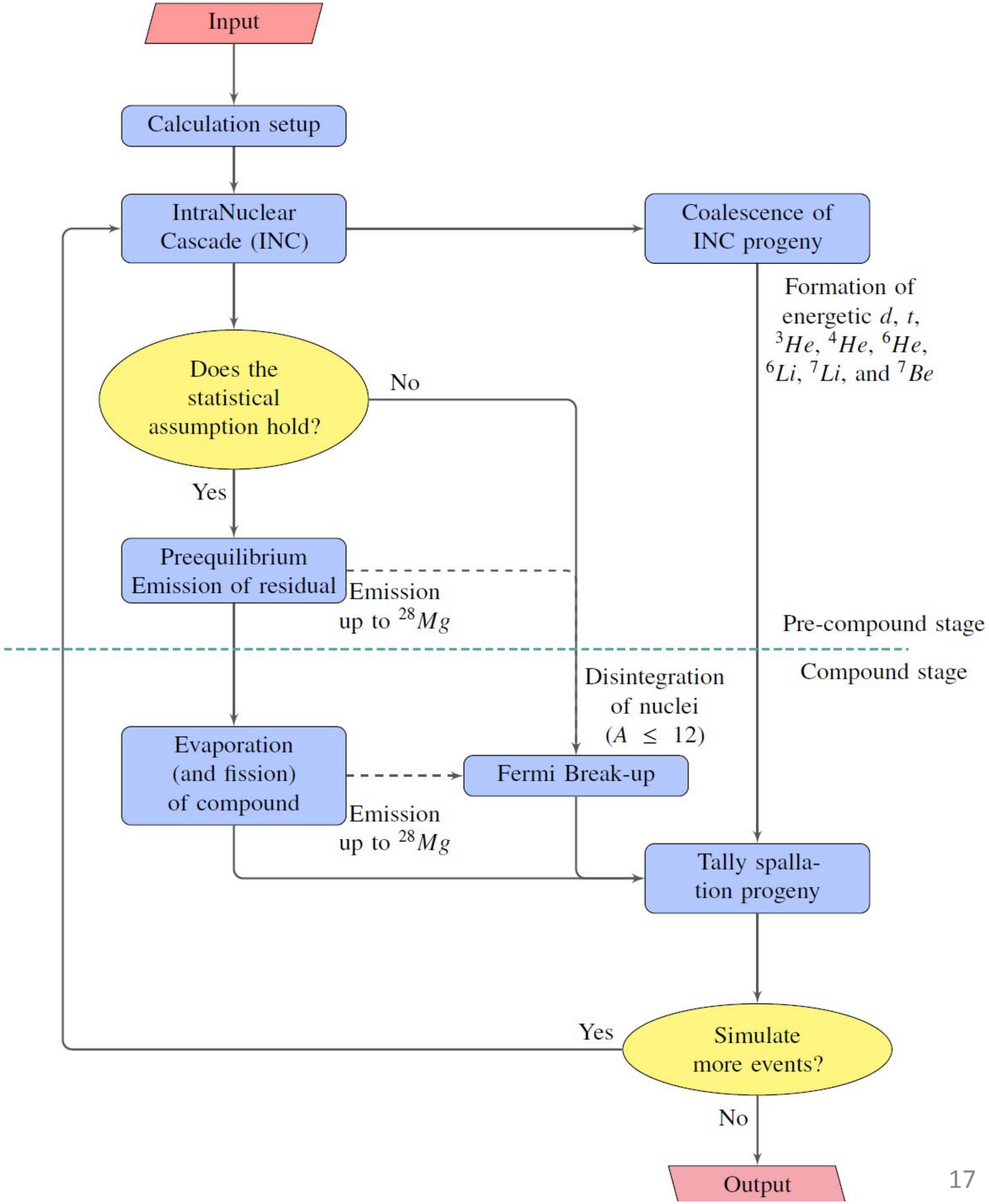
Spallation Physics Overview (cont'd)

- Preequilibrium and evaporation models assume "sufficient" nucleons for statistical invariance[8]
  - Particle emittance estimated via exciton transitions and decay widths, respectively

- Nuclear disintegration is modeled when this assumption is not satisfied ($A_T \leq 12$)[9]
  - Similar to the statistical multi-fragmentation model

Spallation Physics Overview (cont'd)

Spallation simulation flow chart of the GSM



Spallation Physics Overview (cont'd)

- GSM utilizes the Dubna Cascade Model (DCM) for its INC simulations, the "fast" phase

- The DCM in GSM has two forms:
  - Time-<u>independent</u> DCM (standard DCM [sDCM])
  - Time-<u>dependent</u> DCM (modified DCM [mDCM])

Spallation Physics *(continued)*

- The DCM, in general, assumes:
  - Large incident energies ($\gtrsim 100\ MeV$)
    - All collisions are two-bodied collisions occurring between the nucleons of the target and the projectile particle (or the nucleons of a projectile nucleus)
  - Intranuclear collisions occur in less time than that required between successive collisions
    - All collisions are thus independent and identically distributed (IID)

19

Spallation Physics *(continued)*

- The time-independent DCM:
  - Assumes the target nucleus has seven “zones” of constant nuclear densities and Fermi-gas energies
  - Considers nucleon-nucleon, pion-nucleon, and photon-nucleon collisions
  - Utilizes the Pauli Exclusion Principle to verify allowed INC end-states

20

Spallation Physics *(continued)*

- The time-dependent DCM:
  - Utilizes continuous nuclear densities and Fermi-gas energies for the target nucleus
  - Considers nucleon-nucleon, pion-nucleon, and photon-nucleon collisions
  - Utilizes the Pauli Exclusion Principle to verify allowed INC end-states
  - Considers nuclear trawling (*i.e.* depletion)

21

Spallation Physics *(continued)*

- The intermediate phase in GSM is characterized by a preequilibrium model:
  - Considers exciton transitions and subsequent particle emissions
  - Based on the modified exciton model (MEM)
  - Probability of an emission is based on a ratio of simulated exciton information
  - Applicable when a statistical assumption is met
    - $A > 12$

22

Spallation Physics *(continued)*

- Many event generators couple the fast and intermediate phases of the reaction
  - The residual's equilibration is tied into the event generator's INC model (such as in the INCL++)
  - This may be chosen to help mitigate the effects of a "hard" transition from the fast to intermediate phases
  - The DCM, and thus the GSM, considers an optical potential for the transition

23

Spallation Physics *(continued)*

- The slow phase in GSM is characterized by a coupled evaporation-fission model:
  - The Generalized Evaporation Model (GEM) is utilized by GSM to characterize this process
  - GEM2 utilizes <u>many</u> empirical subsets of data
    - Fission is <u>not</u> modeled as releasing any neutrons
  - Considers primarily decay widths for predicting particle emission

24

Idaho State University

CAES Center for Advanced Energy Studies

Spallation Physics *(continued)*

- GSM utilizes a Fermi Break-up model to predict decay of small residual and compound nuclei ($A_T \leq 12$)
  - Similar to the statistical multi-fragmentation model, but for smaller nuclei
  - The nucleons' binding energy is negligible to the particle's excitation energy, thus disintegrating it

25

ROAR

Spallation Physics *(continued)*

- Fermi Break-up is necessitated when the statistical assumption ($A \leq 12$) used in the preequilibrium and evaporation models is no longer satisfied

- GSM, and the preequilibrium and evaporation models, allows the Fermi Breakup model to determine when it should be used

26
ROAR

Spallation Physics *(continued)*

- Secondary nucleons emitted <u>during</u> the INC process may coalesce to form compound nuclei:
  - GSM allows secondary compounding based on a quantum overlap approximation
  - Compounding of particle pairs is modeled (*n+n, n+p, etc.*)
    - D, T and ^3He, ^4He, ^6He and ^6Li, ^7Li and ^7Be, respectively, may be formed

27

Idaho State University

CAES
Center for Advanced Energy Studies

Contents

ROAR

Modernization and API

- Modernization emphasized…
  - Removal of implicit variables
  - Usage of modern language syntax and declarations
  - Modular units of functionality
  - Containerization
  - Flexibility
  - Portability
  - Migration to an object-oriented architecture
  - Robust API creation
  - Object/container optimization

Modernize code

Modernize structure

Optimization

ROAR

Code Modernization

- Prior to this work, GSM was primarily legacy Fortran code (*i*.e. Fortran66/77)

- At present, GSM and all of its sub-models, except the time-dependent DCM, are “modernized”

30

Code Modernization *(continued)*

- "Modernization" here refers to:
  - Modern syntax for current compilers (Fortran2008)
  - Parameters (no *data* statements)
  - Argument intent
  - Modules
  - Object-oriented framework
  - Procedure pointers
  - etc.

31

Idaho State University

CAES
Center for Advanced Energy Studies

Code Modernization (*continued*)

```fortran
      subroutine chabs(l, ine, ne1, ne2, a, z, r1)

      implicit real*8 (a-h, o-z), integer (i-n)

      data one /1.d0/


      .
      .
      .

      return
      end
```

sDCM “chabs” procedure (prior)

32

ROAR

Code Modernization (*continued*)

```
subroutine chabs(sDCM, l, ine, ne1, ne2, a, z, r1)

   use, intrinsic :: iso_fortran_env, only: int32, real64
   use standardDCMParams, only: one

   implicit none
   class(StandardDCM), intent(inout) :: sDCM
   integer(int32), intent(in   ) :: l
   integer(int32), intent(in   ) :: ine
   integer(int32), intent(inout) :: ne1
   integer(int32), intent(inout) :: ne2
   real(real64),   intent(in   ) :: a
   real(real64),   intent(in   ) :: z
   real(real64),   intent(in   ) :: r1

   real(real64) :: t1, t2, temp, temp1

   .
   .
   .

   return
end subroutine chabs
```

sDCM “chabs” procedure (post)

33

Code Modernization (*continued*)

- Modernization here also touches on code <u>readability</u>
  - GSM was difficult for developers to understand

- GSM previously grouped variables in a global scope via *common* blocks, each with eight or less characters (typically 4 or less)
  - Modernization grouped them into derived types with clearly named members

34

Code Modernization (*continued*)

- Why modernize?
  - Migration from “monolithic” to modular
  - Less technical debt
  - Scalability
  - Efficiently utilize HPC resources
  - (Preliminary) parallelization

- These benefits are all HUGE!

Code Modernization *(continued)*

- Migration to an object-oriented approach
  - Creates further modulation (non-monolithic)
  - Dependencies are better known

- GSM and its sub-models utilize many “container” data types
  - GSM itself is a collection of sub-model objects

36

Code Modernization (*continued*)

- GSM and its sub-models utilize various data types:
  - I/O handler
  - Simulation options (behaviors, numerics)
  - Data object(s), where applicable
  - Random number generator (RNG) procedure pointer (for MC methods)
  - During a simulation, end-state and interim result objects are passed among member procedures
    - Allows greater scalability and parallelization
  - Construction flag (protection)
  - Other physics models' interface procedures <u>may</u> be utilized
    - Clients <u>may</u> consider photon emission following particle emission

37

Code Modernization *(continued)*

```
! Import all object-dependencies (data objects, other models, etc.):
!     For example: Molnix, FissionBarrier, PreequilibriumData, FermiBreakUp
use someModelClass, only: ModelData

type, public:: GeneralObject
    ! ==============================================
    ! Object member-variables
    ! ==============================================
    private

    ! Flags if the object was constructed
    logical, private :: constructed = .FALSE.

    ! I/O object for the model:
    type(generalIO), private :: io

    ! Controller for model behavior and numerics:
    type(generalOptions), private :: options

    ! Data objects:
    type(ModelData), private :: data
    procedure(RANDOM), private, pointer :: rang => NULL()

    ! For Preequilibrium, and Evaporation (GSM uses global to all instances):
    logical, private :: usePhotonEmission = .FALSE.
    procedure(PHOTOEMISSION), private, pointer :: photonEmission => NULL()

 contains
```

Object member-variables

38

Code Modernization (*continued*)

```
contains
  ! ================================================
  ! Object member-procedures
  ! ================================================
  private

  ! Simulate procedures, for example:
  procedure, public  :: simulate    ! Interfaces to "startSimulation"
  procedure, public  :: start => simulate

  ! Setter/getter-type methods, for example:
  procedure, public  :: properlyConstructed
  procedure, public  :: setOptions
  procedure, public  :: queryOptions

  ! Internal members for the simulation (highly model-specific):
  procedure, private :: startSimulation
  procedure, private :: someOtherProcedure
  .
  .
  .

end type GeneralObject
```

Object member-procedures

39

Code Modernization *(continued)*

- Some client benefits:
  - Additional features
  - Significant client control
  - Simple client setup
  - Highly reusable
  - Highly scalable
  - Highly parallel
  - *etc.*

40

Code Modernization (*continued*)

- Developer benefits:
  - Fast deployment
    - Introduce hotfixes and new features and improvements quickly
  - Modify code without affecting client utilization
  - Highly decoupled

41
ROAR

Code Modernization (*continued*)

```
# Build GSM
mkdir ./build && cd ./build
cmake ../ && make

# Move to the simulation directory:
cd ../my_GSM

# See what arguments are available:
./xgsm1 -help

# Perform a simulation:
./xgsm1 -v=2 -i=myInputFile.inp
```

Building and running GSM

- CMake and command line options

42

API Development

- The Application Programming Interface (API) acts as the “contract” between the client and the object being utilized

- Provides well defined methods to utilize the object and on utilizing the object and all generated data

43

API Development (continued)

- An API acts as the “seams” between a client and a model or server
- Flexible implementation
- Follows a service-oriented architecture (SOA)
  - “Black-box” utilization, self-contained, specified focus, *etc.*

44

API Development (continued)

- Provides much abstraction
- Localizes complexity
- Each object-oriented model may be its own API
  - The object's constructor and various accessible procedures comprise the API

45

API Development (continued)

- The API of an object-oriented model may be easily consumed and tested

- The GSM API requires:
  - Data initialization and construction, object construction, and then simulation (as needed)
  - Simulation results are contained within a single object for client ease

46

```fortran
subroutine clientGSMDataInitialization( clientProcedure )

   use generalizedSpallationData, only: &
        & initializeGSMData, &   ! Subroutine to initialize data
        & gsmDataInitialized     ! Logical flag indicating if data was
            initialized

   implicit none
   procedure(IOHANDLER), intent(in   ), pointer :: clientProcedure

   ! Verify data was not already initialized:
   if ( gsmDataInitialized ) then
      write(*,*) "The GSM data was already successfully initialized."
      return
   end if

   ! Initialize data using default arguments:
   call initializeGSMData()

   ! Initialize data using optional arguments (with default values):
   call initializeGSMData( &
        & clientRndmType = 1,              &   ! Random Number Generator
        & gammaFile = "channel1.tab", gammaUnit = 17, &   ! sDCM Defaults
        & photoFile = "channel1.tab", photoUnit = 17, &   ! mDCM Defaults (
            photon file)
        & decayFile = "atab.dat",     decayUnit = 18, &   ! mDCM Defaults (
            decay file)
        & massFile  = "mass.tbl",     massUnit  = 19, &   ! Evaporation
            Defaults (mass file)
        & levelFile = "level.tbl",    levelUnit = 20, &   ! Evaporation
            Defaults (level file)
        & shellFile = "shell.tbl",    shellUnit = 21, &   ! Evaporation
            Defaults (shell file)
        & clientIO = clientProcedure                  &   ! I/O for message
            handling
        & )

   ! Check if initialization was successful:
   if( .not.gsmDataInitialized ) then
      write(*,*) "The GSM data failed to initialize."
   end if

   return
end subroutine clientGSMDataInitialization
```

API Development (continued)

- GSM data initialization

47

API Development (continued)

```fortran
subroutine constructGSM( thisRNG, thisGSMOptions, thisIO )

   use gsmClass, only: &
        & GSM,          & ! GSM class/object type
        & newGSM,       & ! GSM constructor interface
        & gsmOptions      ! Options for the GSM simulation

   implicit none
   procedure(RANDOM),    intent(in   ), pointer :: thisRNG
   type(gsmOptions),     intent(inout)          :: thisGSMOptions
   procedure(IOHANDLER), intent(in   ), pointer :: thisIO

   ! Declare a GSM object:
   type(GSM) :: gsmObj


   ! Construction of GSM with default arguments:
   gsmObj = newGSM()

   ! Construction of GSM with all optional arguments:
   gsmObj = newGSM(                             &
        & clientRNG     = thisRNG,              &
        & clientOptions = thisGSMOptions,       &
        & clientIO      = thisIO                &
        & )

   ! Update client's options object to reflect those used:
   thisGSMOptions = gsmObj%queryOptions()

   return
end subroutine constructGSM
```

General GSM construction

48

Idaho State University

CAES Center for Advanced Energy Studies

API Development (continued)

```
subroutine constructGSMResults( bankSize )

   use gsmClass, only:  &
        & GSMProgeny,   &    ! Progeny tracked in GSM simulation
        & GSMResults,   &    ! GSM class/object type
    & newGSMResults        ! GSM constructor interface

   implicit none
   integer(int32), intent(in   ) :: bankSize

   ! Declare progeny array:
   type(GSMProgeny), dimension( bankSize ) :: progenyBank


   ! Declare results object and construct it:
   type(GSMResults) :: results
   results = newGSMResults( progenyBank )

   return
end subroutine constructGSMResults
```

gsmResults constructor

49

ROAR

Idaho State University

CAES Center for Advanced Energy Studies

API Development (continued)

```
subroutine sample_GSM_API(projectilePbj, targetObj)
! ==================================================================
! A composite sample API for software clients of the GSM
! ==================================================================
   use, intrinsic:: iso_fortran_env, only: int32, real64
   use gsmClass, only:    &
        & GSM,            &    ! GSM Object
        & newGSM,         &    ! GSM Constructor
        & gsmOptions,     &    ! Options controller
        & gsmVerbose,     &    ! Controls all object's verbosity
        & gsmProjectile,  &    ! Projectile object
        & gsmTarget,      &    ! Target object
        & gsmProgeny      &    ! Progeny tracked by GSM
        & gsmResults,     &    ! Results object
        & newGSMResults         ! Results object constructor

   type(gsmProjectile), intent(inout) :: projectileObj
   type(gsmTarget),     intent(inout) :: targetObj

   ! ==============================================================
   ! Set options:
   type(gsmOptions) :: options
   options%nucleonTransitionE = 1000.0_real64 ! Reduce INC transition to 1 GeV

```

Sample API (1)

50

ROAR

Idaho State University

CAES Center for Advanced Energy Studies

API Development (continued)

```
    options%nucleonTransitionE = 1000.0_real64 ! Reduce INC transition to 1 GeV

    ! Construct the  object:
    type(GSM) :: gsmObj
    gsmObj = newGSM(clientOptions = options)

    ! ==============================================================
    ! Create and construct the results object
    type(gsmProgeny), dimension(150):: progenyBnk
    type(gsmResults):: results
    results = newGSMResults(progenyBnk)   ! Object construction

    ! Simulate a collision:
    gsmObj%collide(projectileObj, targetObj)

    ! Interface to the results, for example
    write(*, 1000) results%numProgeny
    if(results%simState /= 0_int32 ) then
        write(*, 1100) results%simState
    end if
    return
! ==================================================================
1000 format("There were ", i3, " progeny produced during the simulation.")
1100 format("Warning: the GSM simulation ended with a warning or error (", i3,
     ").")
! ==================================================================
end subroutine sample_GSM_API
```

Sample API (2)

51

API Development (continued)

Member Name	Scope	Data Type	Default	Description
particleName	Public	character(6)	*prot*	Name of the particle
numBaryons	Public	integer(int32)	1	Number of nucleons in the nucleus
numProtons	Public	integer(int32)	1	Number of protons in the nucleus
decayNumber	Public	integer(int32)	0	Quantum decay number
kinEnergy	Public	real(real64)	1.0	Kinetic energy of the incident nucleus [GeV]
kinEnergyMax	Public	real(real64)	$1E9$	Maximum kinetic energy of the incident nucleus [GeV]
dKinEnergy	Public	real(real64)	-50.0	Step size of incident energy [MeV] (≤ 0 unused)
restMass	Public	real(real64)	-1.0	Rest mass of the nucleus [GeV/c^2]
afMultiplier	Public	real(real64)	-1.0	A_f multiplier within the fission model
czMultiplier	Public	real(real64)	-1.0	C_Z multiplier within the fission model
particleFlag	Public	integer(int32)	1	Flags the incident particle type as a nucleus (0), mono energetic photon (1), bremsstrahlung photon (2), or a pion (3)
system	Private	integer(int32)	1	Flags the system used during the INC calculation

The "gsmProjectile" object

52

Idaho State University

CAES Center for Advanced Energy Studies

API Development (continued)

Member Name	Scope	Data Type	Default	Description
particleName	Public	character(6)	*Pb-208*	Name of the target nucleus
numBaryons	Public	real(real64)	208.0	Number of nucleons in the nucleus
numProtons	Public	real(real64)	82.0	Number of protons in the nucleus
restMass	Public	real(real64)	−1.0	Rest mass of the nucleus [GeV/c^2]
afMultiplier	Public	real(real64)	−1.0	A_f multiplier within the fission model
czMultiplier	Public	real(real64)	−1.0	C_Z multiplier within the fission model

The “gsmTarget” object

53
ROAR

API Development (continued)

Member Name	Scope	Data Type	Description
initialProj	Public	GSMProjectile	Pointer to the initial projectile nucleus
initialTarg	Public	GSMTarget	Pointer to the initial target nucleus
projRes	Public	GSMResidual	Residual projectile nucleus at the end of the simulation
targRes	Public	GSMResidual	Residual target nucleus at the end of the simulation
projExc	Public	excitonData	Projectile exciton information at the end of the simulation
targExc	Public	excitonData	Target exciton information at the end of the simulation
progenyBnk	Public	GSMProgeny	Array pointer to the client-defined progeny array
numProgeny	Public	integer(int32)	Number of progeny existing the in the progeny bank
maxProgeny	Private	integer(int32)	Size of the progeny array
maxProgenyM1	Private	integer(int32)	Size of the progeny array minus one
numElasticEvents	Public	integer(int32)	Number of elastic events that occurred prior to the occurence of an inelastic event
modelUsage	Public	GSMModelUsage	Provides information on the sub-model usage within GSM
restarts	Public	eventRestarts	Specifies the number of times the event was restarted due to various detected errors
simState	Public	integer(int32)	Flags the end-state of the simulation
constructed	Private	logical	Flags whether or not the GSM object was constructed

The “gsmResults” object

54

Idaho State University

CAES Center for Advanced Energy Studies

API Development (continued)

Member Name	Type	Description
numBaryons	real64	Total number of nucleons (i.e. baryons)
numProtons	real64	Total number of protons (i.e. baryons)
kinEnergy	real64	Kinetic energy of particle [GeV]
restMass	real64	Rest mass of particle [GeV/c^2]
phi	real64	ϕ component of particle's motion
theta	real64	θ component of particle's motion
sinTheta	real64	$\sin\theta$
cosTheta	real64	$\cos\theta$
typeID	real64	ID of the particle, being one up to nine for n, p, d, t, ^{3}He, ^{4}He, π^-, π^0, π^+, or $= 1000 * Z + N$
prodMech	real64	Mechanism by which progeny was produced

The "gsmProgeny" object

55

ROAR

API Development (continued)

Member Name	Type	Description
numBaryons	real64	Total number of nucleons (i.e. baryons)
numProtons	real64	Total number of protons (i.e. baryons)
kinEnergy	real64	Kinetic energy of particle [GeV]
linearMom	real64	3-dimensional (x, y, z) linear momentum [GeV/c]
angularMom	real64	3-dimensional (x, y, z) angular momentum [GeV (unit length)/c]

The “gsmResidual” object

Idaho State University

CAES Center for Advanced Energy Studies

API Development (continued)

Procedure Name	Type	Description
For Establishing the GSM State (without the Constructor):		
updateOptions	Subroutine	Sets the *gsmOptions* type used by GSM for its simulations
updateRNG	Subroutine	Sets the RNG procedure pointer utilized by GSM
updateMessageHandler	Subroutine	Sets the message handling procedure utilized by GSM when messages are generated
For Querying the GSM State:		
properlyConstructed	Function	Returns logical flag for GSM construction state
queryOptions	Function	Returns the `GSMOptions` object utilized by GSM for its simulations
queryRNG	Function	Returns the RNG procedure pointer used by GSM
queryPhotoEmissionUse	Function	Returns a logical flag for whether or not photon emission is used in GSM
For Simulating A Single Event:		
simulateEvent	Subroutine	Simulates a single spallation event
performSimulation	Subroutine	See the *simulateEvent* procedure for details
collide	Subroutine	See the *simulateEvent* procedure for details
interact	Subroutine	See the *simulateEvent* procedure for details
simulate	Subroutine	See the *simulateEvent* procedure for details
execute	Subroutine	See the *simulateEvent* procedure for details
start	Subroutine	See the *simulateEvent* procedure for details

Public GSM procedures (1)

57

ROAR

Idaho State University

CAES Center for Advanced Energy Studies

API Development (continued)

For Generating an Output File:		
generateOutput	Subroutine	Generates an output file
output	Subroutine	See the *generateOutput* procedure for details
simulate	Subroutine	See the *generateOutput* procedure for details
execute	Subroutine	See the *generateOutput* procedure for details
start	Subroutine	See the *generateOutput* procedure for details
Physics Interfaces:		
standardDCMInterface	Subroutine	Interface to the standard DCM sub-library
setupMDCM	Subroutine	Establishes the modified DCM nuclei data for a given simulation
modifiedDCMInterface	Subroutine	Interface to the modified DCM sub-library
coalescenceInterface	Subroutine	Interface to the coalescence sub-library
fermiBreakUpInterface	Subroutine	Interface to the Fermi break-up sub-library
preequilibriumInterface	Subroutine	Interface to the preequilibrium sub-library
evaporationInterface	Subroutine	Interface to the evaporation sub-library
simulateDecay	Subroutine	Interfaces to a residual's preequilibrium and evaporation decay scheme
Other Available Procedures:		
formNuclei	Subroutine	Establishes all default fields for a given

Public GSM procedures (2)

58

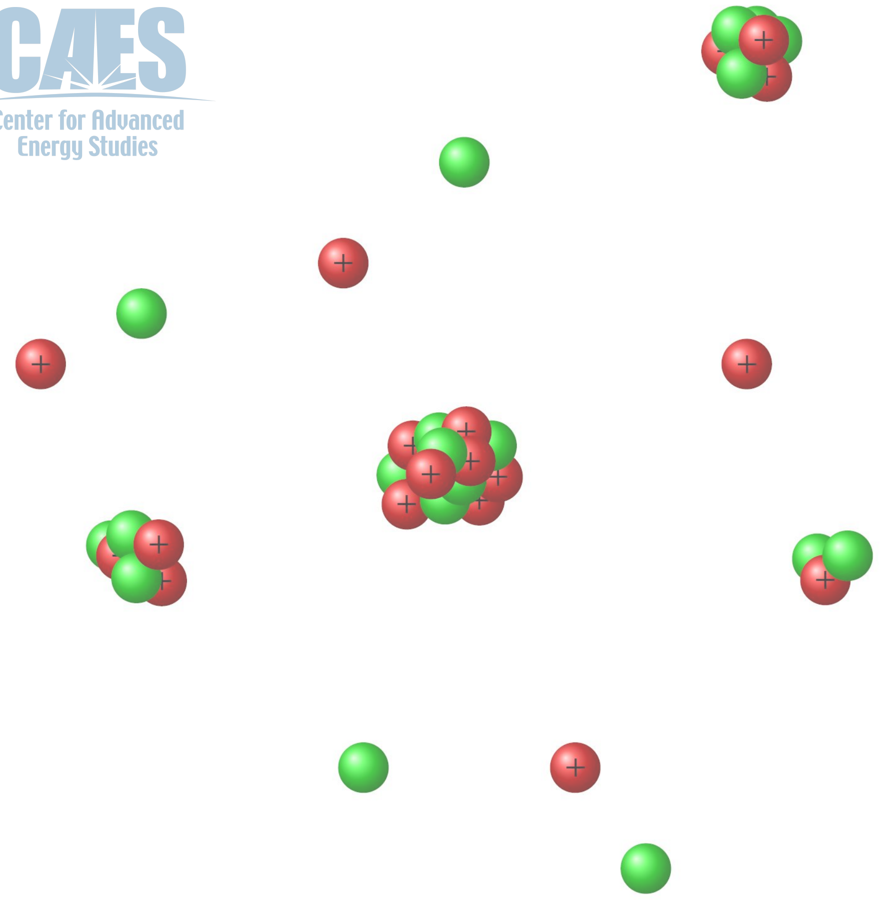
ROAR

API Development (continued)

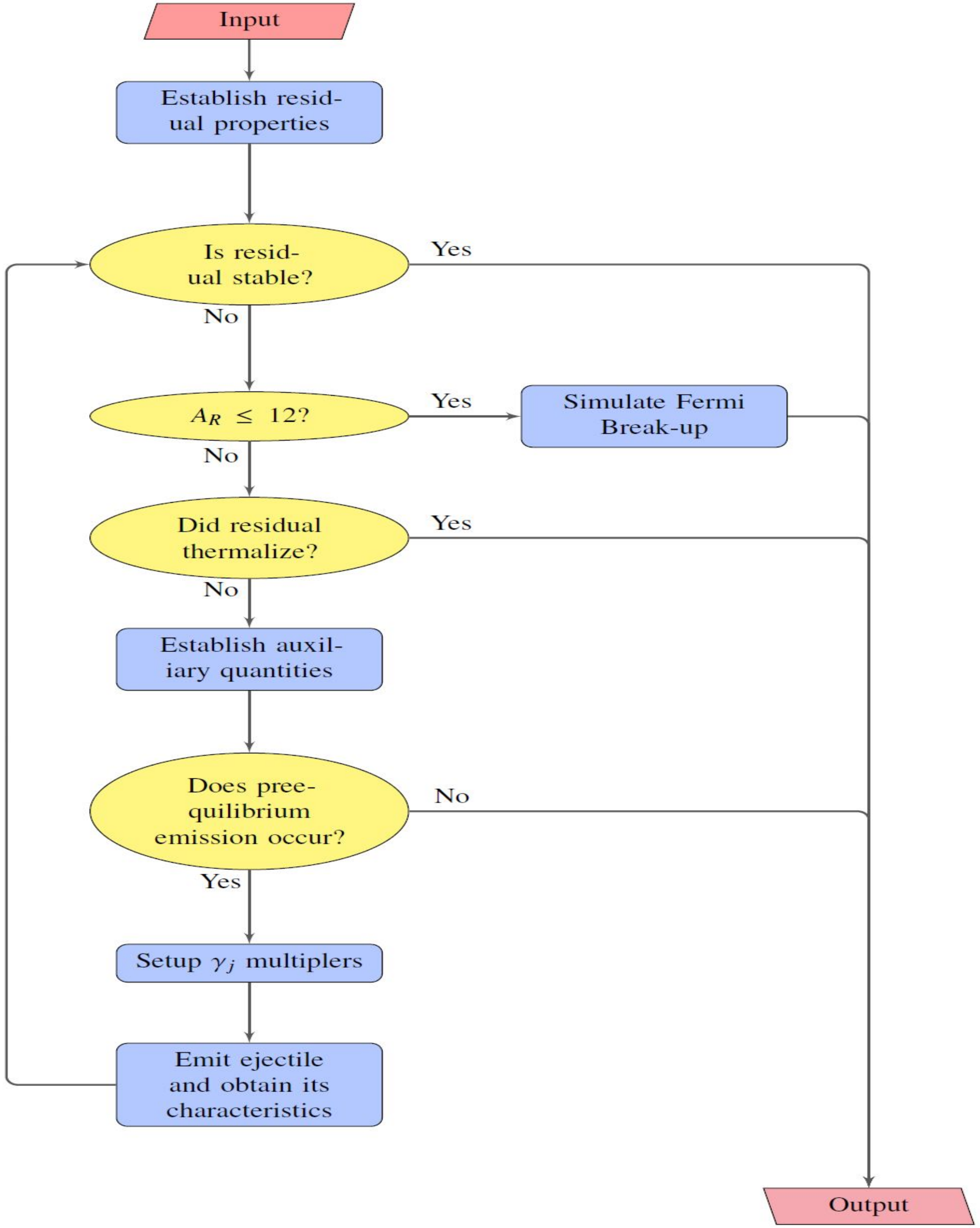
Procedure Name	Type	Description
		projectile and target nucleus
buildNuclei	Subroutine	See the *formNuclei* procedure for details
inquireINCModel	Function	Determines the INC model to be used given a projectile and target
sampleEnergy	Function	Returns an energy directly sampled from a Gaussian distribution around a provided mean and standard deviation, when GSM uses continuous transitions around the INC transition energy
determineRestMass	Function	Returns the GSM approximation for a nucleus's rest mass [GeV/c^2]

Public GSM procedures (3)

59



CASE STUDY: THE PREEQUILIBRIUM MODEL AND CODE



Preequilibrium
(continued)

- Flow chart of preequilibrium physics

61
ROAR

Preequilibrium (continued)

```
! Preequilibrium Class
type, public :: Preequilibrium
   private

   ! Flags if the object was constructed or not:
   logical, private :: constructed = .FALSE.


   ! For all message handling:
   type(preequilibriumIO), private :: io


   ! Class options/numerics/behaviors:
   type(preequilibriumOptions), private :: options


   ! For the required data:
   ! (non-reaction specific)
   type(Molnix),             private, pointer :: molEnergy    => NULL()
   type(FissionBarrier),     private, pointer :: fissBarr     => NULL()
   ! (reaction specific)
   type(PreequilibriumData), private, pointer :: preeqData    => NULL()


   ! Random Number Generator:
   procedure(RANDOM),        private, pointer, nopass :: rng => NULL()


   ! To allow client to simulate photon emission:
   procedure(PHOTOEMISSION), private, pointer, nopass :: photonEmission    => NULL()
   logical,                  private                 :: usePhotonEmission =  .FALSE.
```

- The preequilibrium member variables

62

Preequilibrium (continued)

- Preequilibrium class data types
  - Similar implementation for GSM's other sub-models

```
! For all message handling:
type, private :: preequilibriumIO
   private
   character(LEN=512),   private                     :: message  =  ""
   procedure(IOHANDLER), private, pointer, nopass :: print    => printPreeq
end type preequilibriumIO


! Contains options available to the preequilibrium simulation (ALL user specifiable)
type, public  :: preequilibriumOptions
   private
   real(real64),    public  :: r0Mult         = defaultR0Multiplier     != Radius multip
   integer(int32), public  :: numPreeqType   = defaultNumPreeqType     != Number of pre
   integer(int32), public  :: levelDenParam  = defaultLevelDenParam    != Flag for whic
   real(real64),    public  :: emissionWidth  = defaultEmissionWidth    != Standard devi
   integer(int32), public  :: excludePreeq   = defaultExcludePreeq     != Flags to NOT
end type preequilibriumOptions
```


63

Preequilibrium (continued)

- Preequilibrium data class (member variables and types)

```
! Preequilibrium data class
type, public :: PreequilibriumData
   private

   ! To ensure data class was constructed before use:
   logical, private :: constructed = .FALSE.


   ! For message handling:
   type(preequilibriumDataIO), private :: io


   ! Non-Specific Data:
   type(Molnix),          private, pointer  :: molEnergy => NULL()   ! For energies
   type(FissionBarrier), private, pointer  :: fissBarr  => NULL()   ! For the fissi


   ! Reaction Specific Data:
   type(preequilibriumCompound), private :: compound
   type(compoundEnergies),       private :: compEnergy   ! Ground state and fission
   type(GammaJ),                 public  :: gammaJObj    ! For F_j coefficients
```


64

Preequilibrium (continued)

- Preequilibrium data class (member procedures)

```
contains
  private

  ! To obtain information on the compound nucleus that was used to establish the data:
  procedure, public  :: numBaryons
  procedure, public  :: numProtons
  procedure, public  :: kinEnergy


  ! To obtain ground state and fissin barrier energies of the compound:
  procedure, public :: auxl


  ! To obtain the values of the ground state and fission barrier arrays:
  procedure, private :: checkIndex   ! Verifies that the requested valued of 'eb' or 'egs' exists
  procedure, public  :: eb
  procedure, public  :: egs


  ! Various ways clients can access the information here:
  procedure, public  :: properlyConstructed
  procedure, public  :: getMolnix
  procedure, public  :: getFissionBarrier

end type PreequilibriumData
```


65

Preequilibrium (continued)

- Preequilibrium results object

```
type, public :: preequilibriumResults
   private

   ! Flags to the object that progeny array is NOT associated with any location in memc
   logical, private :: constructed = .FALSE.


   ! Progeny information:
   integer(int32), public  :: numProgeny = 0_int32
   integer(int32), public  :: maxProgeny = 0_int32
   type(preequilibriumFragment), public, dimension(:), pointer :: progenyBnk => NULL()


   ! Nucleus information:
   type(residualNucleus), public :: initResidual
   type(residualNucleus), public :: residual


   ! Indicator of how the simulation ended:
   integer(int32), public :: simState = 0_int32

end type preequilibriumResults
```


66

Preequilibrium (continued)

- Preequilibrium construction

```
! Construct Reaction-specific preequilibrium object:
preeqObj = newPreequilibrium(preeqData, gsmObj%rang, &
        & gsmObj%options%preeq, clientIO = gsmObj%io%print)
```

- Note that an optional photon emission procedure pointer is <u>not</u> utilized
  - GSM does not include this model, but the preequilibrium object allows clients to specify one

67

Preequilibrium (continued)

- Usage of the preequilibrium model requires:
  - Construction of a results object
  - Passing in a preequilibrium residual nucleus

```
! Construct results object:
results = newPreequilibriumResults( progenyBnk )


! Simulate preequilibrium physics
call preeqObj%simulate(excitedResidual, n0, np0, nh0, npz0, results)
```


68

Modernization and API (cont'd)

- The internal state of the object is further containerized
  - Low mutability with data hiding and API
  - Abstraction
  - Flexibility

- Object-oriented structure easily scales with problem size and processing power

Modernization and API (cont'd)

- Object-oriented structures provides a robust API
  - Provides an implicit contract of <u>what</u> the software client can and cannot do
- API provides simple methods for controlling and using the model
  - Simple setup, pre-processing, usage, and post-processing
  - Creates many simple layers with simple interfaces
- Simple implementation into software clients, *e.g.* MCNP6, Geant4, *etc.*

Modernization and API (cont'd)

TABLE I. Some API Procedures of the GSM Event Generator

Name	Description
updateOptions	Modifies the internal simulation option object utilized by the GSM
properlyConstructed	Returns a logical flag indicating the construction state of the GSM object
simulateEvent	Simulates a single spallation event
generateOutput	Generates an output file based on the provided output options object
fermiBreakUpInterface	Interface to the Fermi breakup model
formNuclei	Establishes default fields of the projectile and target nuclei objects

- Currently pursuing an open-source license with LANL and ISU
  - GSM may be obtained through LANL or one of its authors presently

Contents

Model Verification (continued)

- GSM significantly reduces the technical debt of its predecessors, CEM and LAQGSM
  - Less code to maintain and better modulation

- GSM provides significant abstraction
  - Simple to add new features, controls, *etc.*

73

Model Verification (continued)

- Conservation of mass, charge, momentum, and energy are checked at each event
  - Momentum conservation is verified after INC simulations
    - On total (default) or on average (approximation, faster)
  - Energy conservation verified after the INC stage
  - Mass and charge conservation verified after INC stage and collision end-state

74

Model Verification (continued)

- Profiling GSM can reveal where simulations differ from those of CEM and LAQGSM
  - GSM matches the profiles of CEM and LAQGSM for neutron, proton, photon, pion, and some light-ion induced events
  - There have been discrepancies between the timing of the modified DCM in GSM and in LAQGSM for heavy-ion and some light-ion induced events

75

Idaho State University

CAES Center for Advanced Energy Studies

Code Validation

- High-level checking
  - Does GSM do what it is requested of it?
- Primarily black-box regression and unit testing
  - Simulation results compared to that of CEM, LAQGSM, and experimental data
- Note validation for EGs is considered “good” if within an order of magnitude

76
ROAR

Code Validation (continued)

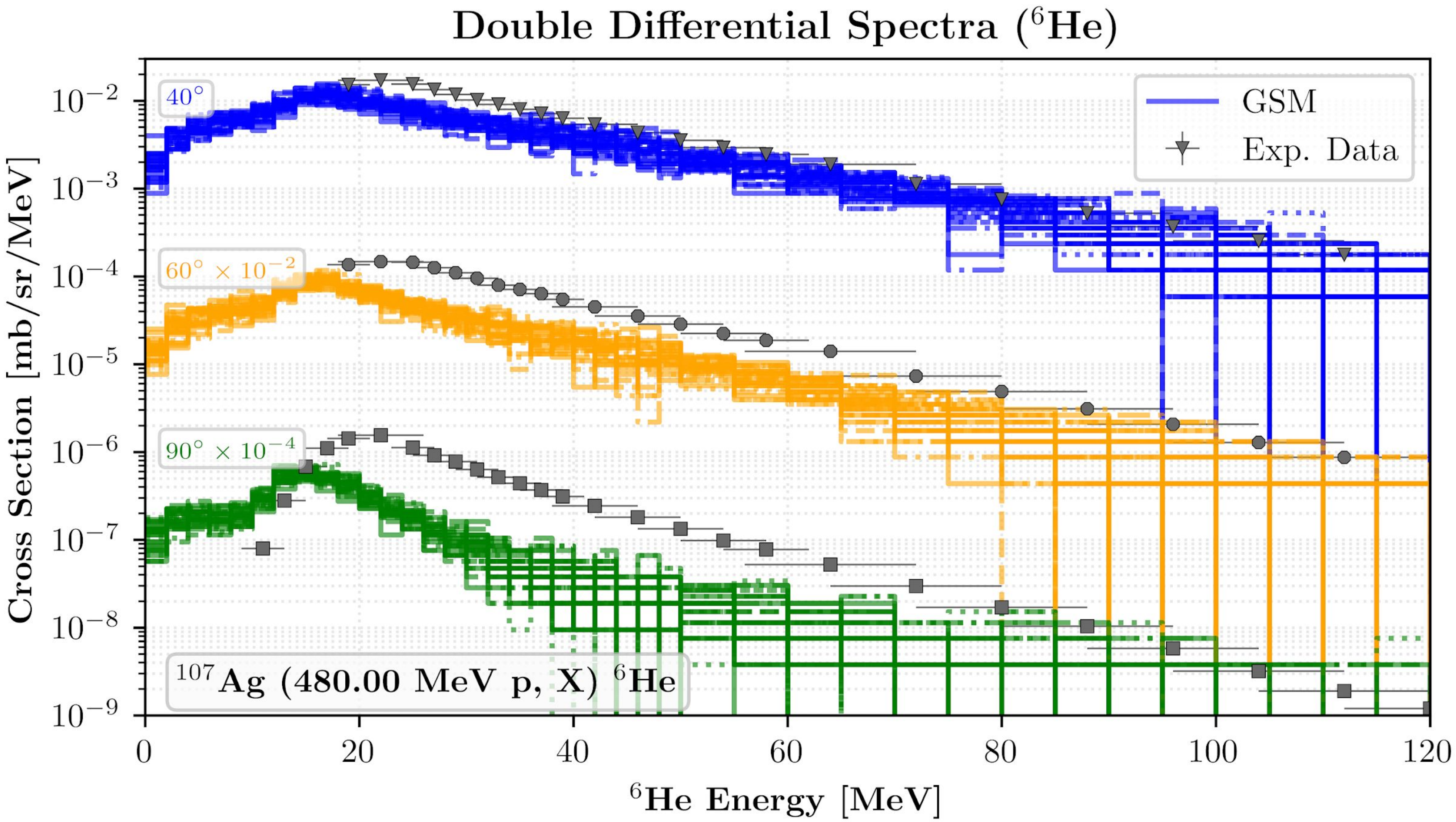
- Recommended to improve the current CMake build system
  - "CMake is an open-source, cross-platform family of tools designed to build, test and package software" (cmake.org)
  - Generated Makefiles are used to build the software without needing to know how to build it
- Aids in portability between systems
  - *e.g.* Linux, Mac, Windows, *etc.*

77
ROAR

Results and Analysis

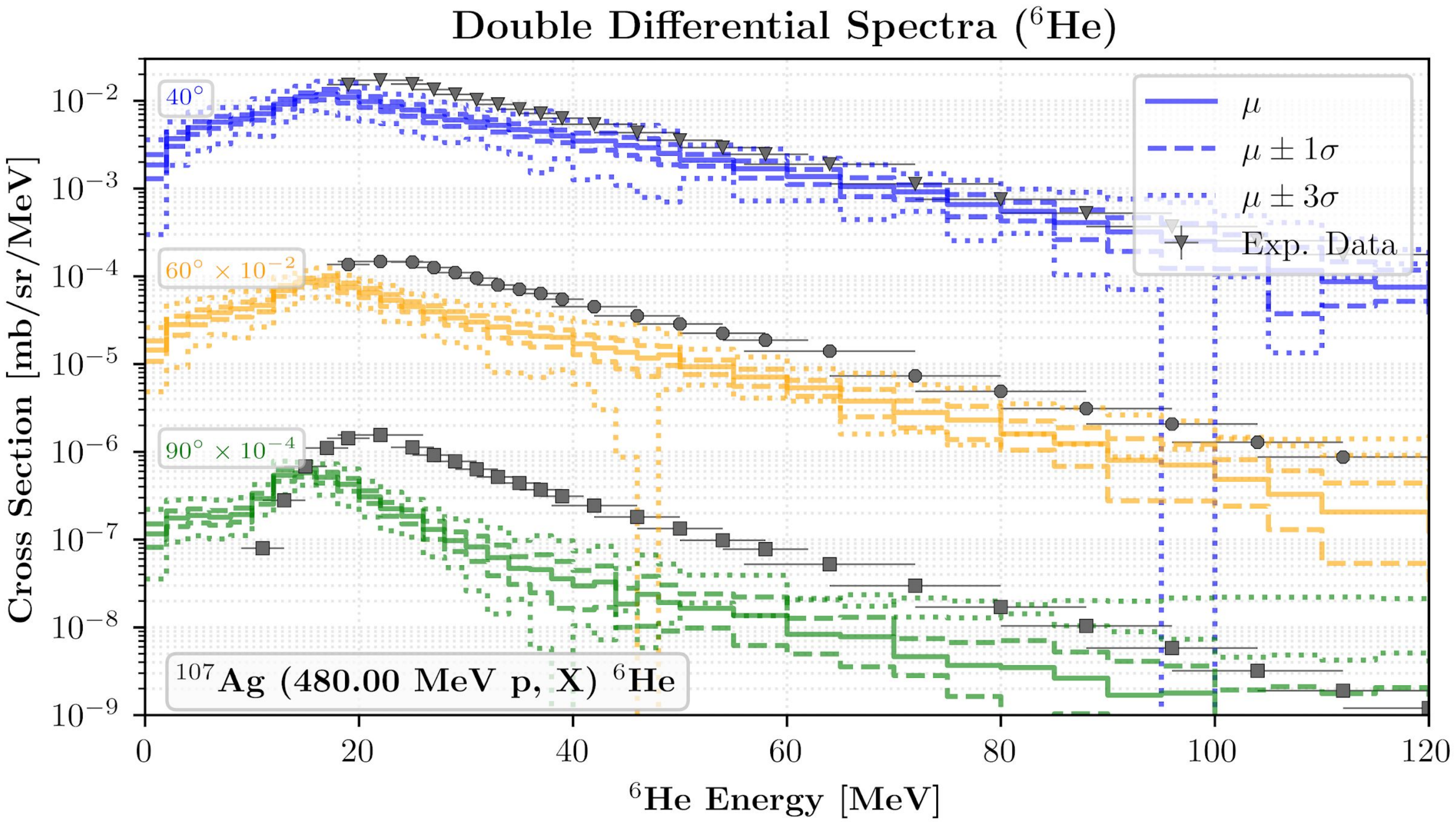
- Extensible software application
- Simple setup, pre-processing, usage, and post-processing

Results and Analysis (cont'd)[10]



10. Exp. Data from R. Green, R. Korteling, and K. Jackson. "Inclusive Production of Isotopically Resolved Li through Mg Fragments by 480 MeV p + Ag Reactions". In: Physical Review, Part C, Nuclear Physics 29 (1984), p. 1806. DOI: 10.1103/PhysRevC.29.1806. URL: http://dx.doi.org/10.1103/PhysRevC.29.1806.

Results and Analysis (cont'd)[10]

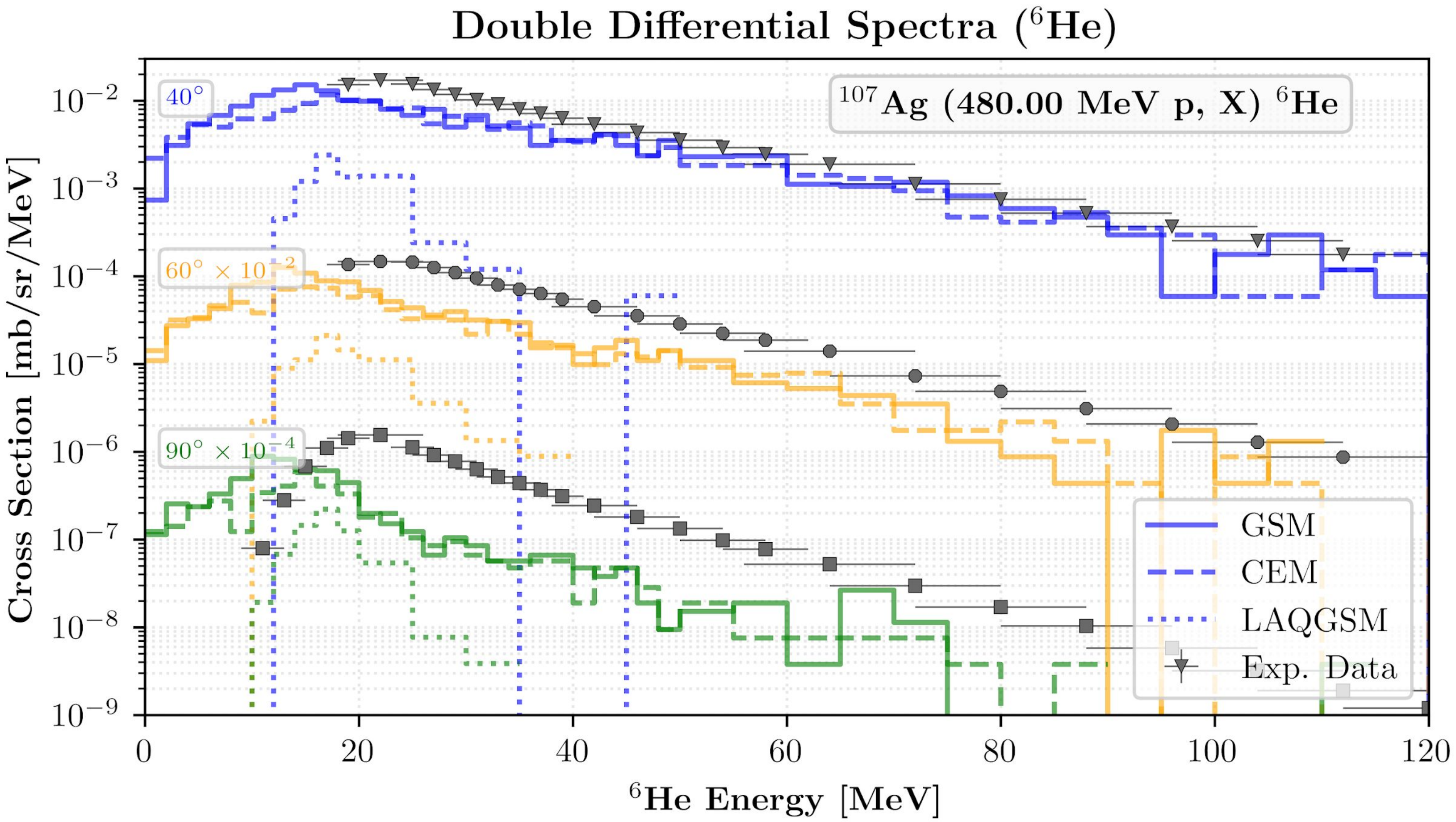


10. Exp. Data from R. Green, R. Korteling, and K. Jackson. "Inclusive Production of Isotopically Resolved Li through Mg Fragments by 480 MeV p + Ag Reactions". In: Physical Review, Part C, Nuclear Physics 29 (1984), p. 1806. DOI: 10.1103/PhysRevC.29.1806. URL: http://dx.doi.org/10.1103/PhysRevC.29.1806.

Idaho State University

CAES Center for Advanced Energy Studies

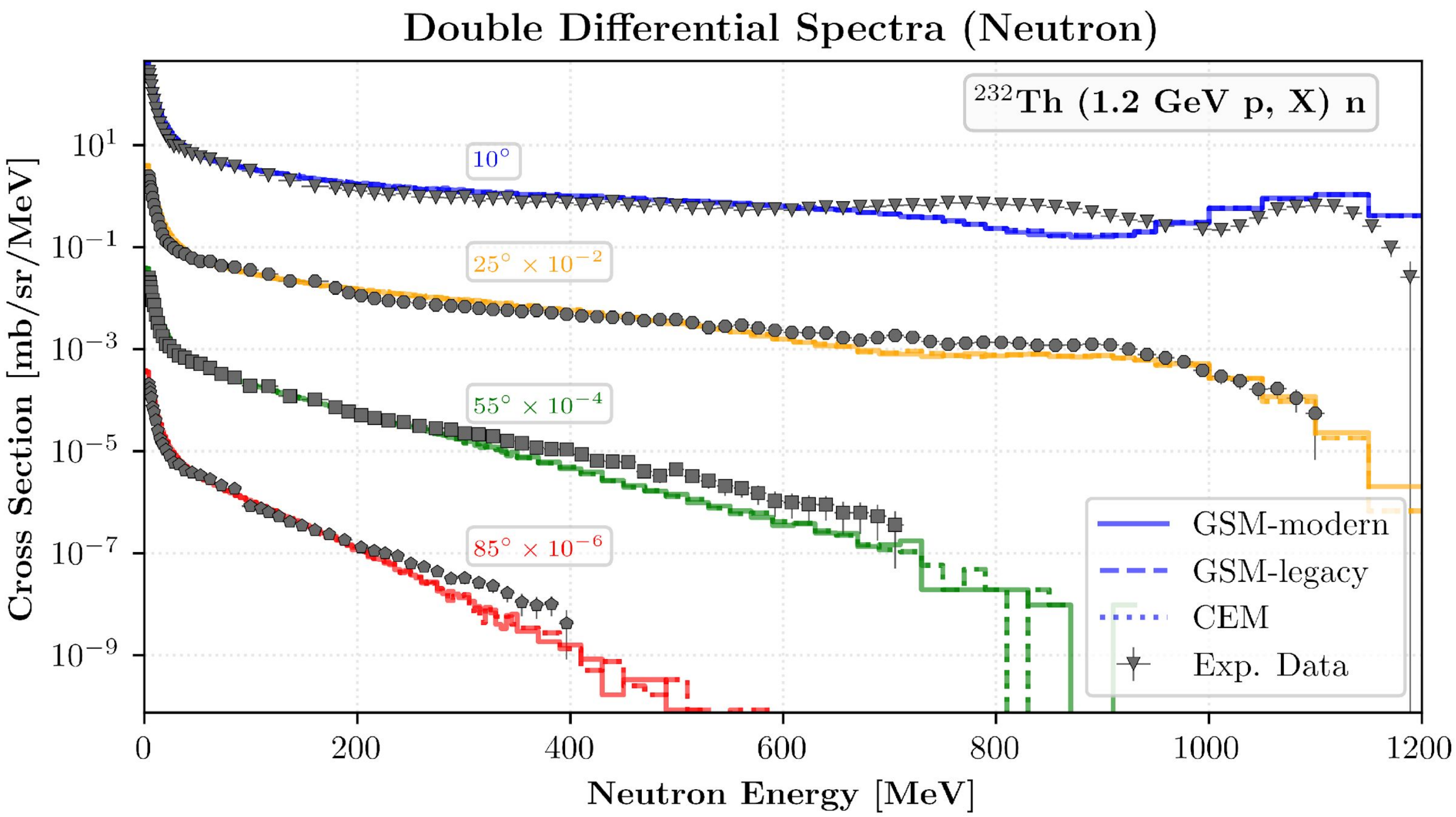
Results and Analysis (cont'd)[10]



10. Exp. Data from R. Green, R. Korteling, and K. Jackson. "Inclusive Production of Isotopically Resolved Li through Mg Fragments by 480 MeV p + Ag Reactions". In: Physical Review, Part C, Nuclear Physics 29 (1984), p. 1806. DOI: 10.1103/PhysRevC.29.1806. URL: http://dx.doi.org/10.1103/PhysRevC.29.1806.

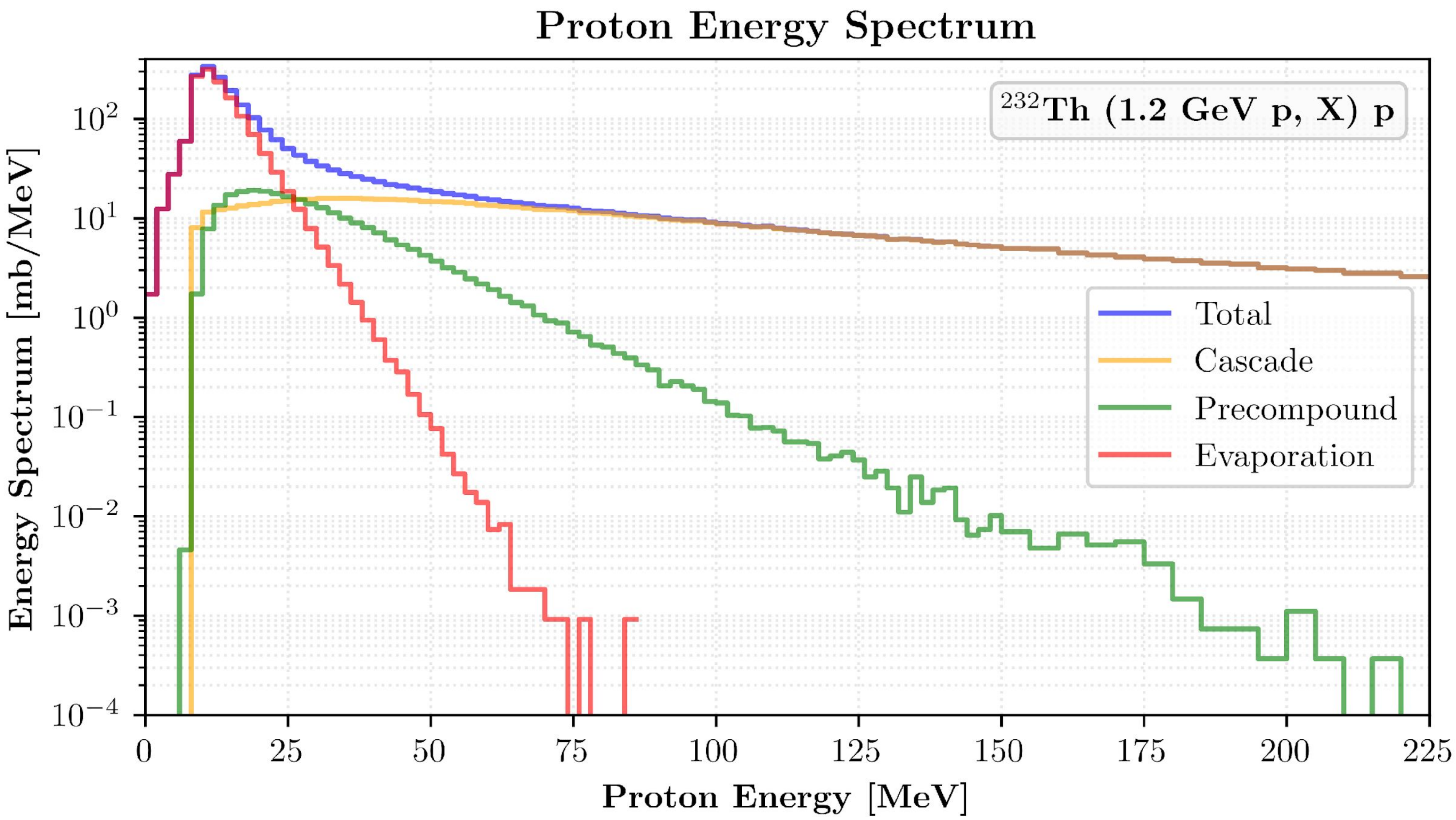
ROAR

Results and Analysis (cont'd)[11]

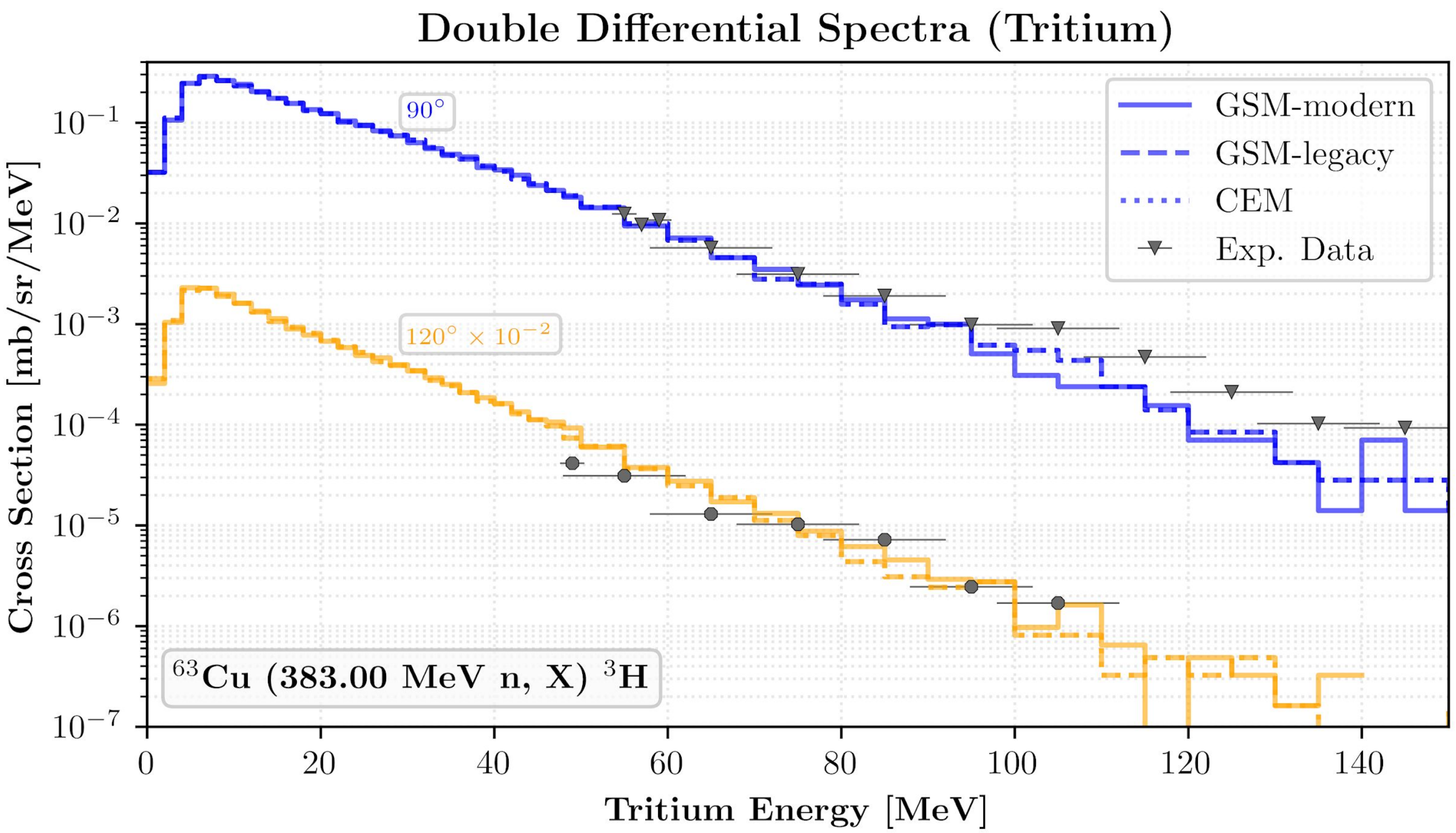


11. Exp. Data from S. Leray et al. "Spallation Neutron Production by 0.8, 1.2, and 1.6 GeV Protons on Various Targets". In: Physical Review, Part C, Nuclear Physics 65 (2002), p. 044621. DOI: 10.1103/PhysRevC.65.044621. URL: http://dx.doi.org/10.1103/PhysRevC.65.044621.

Results and Analysis (cont'd)

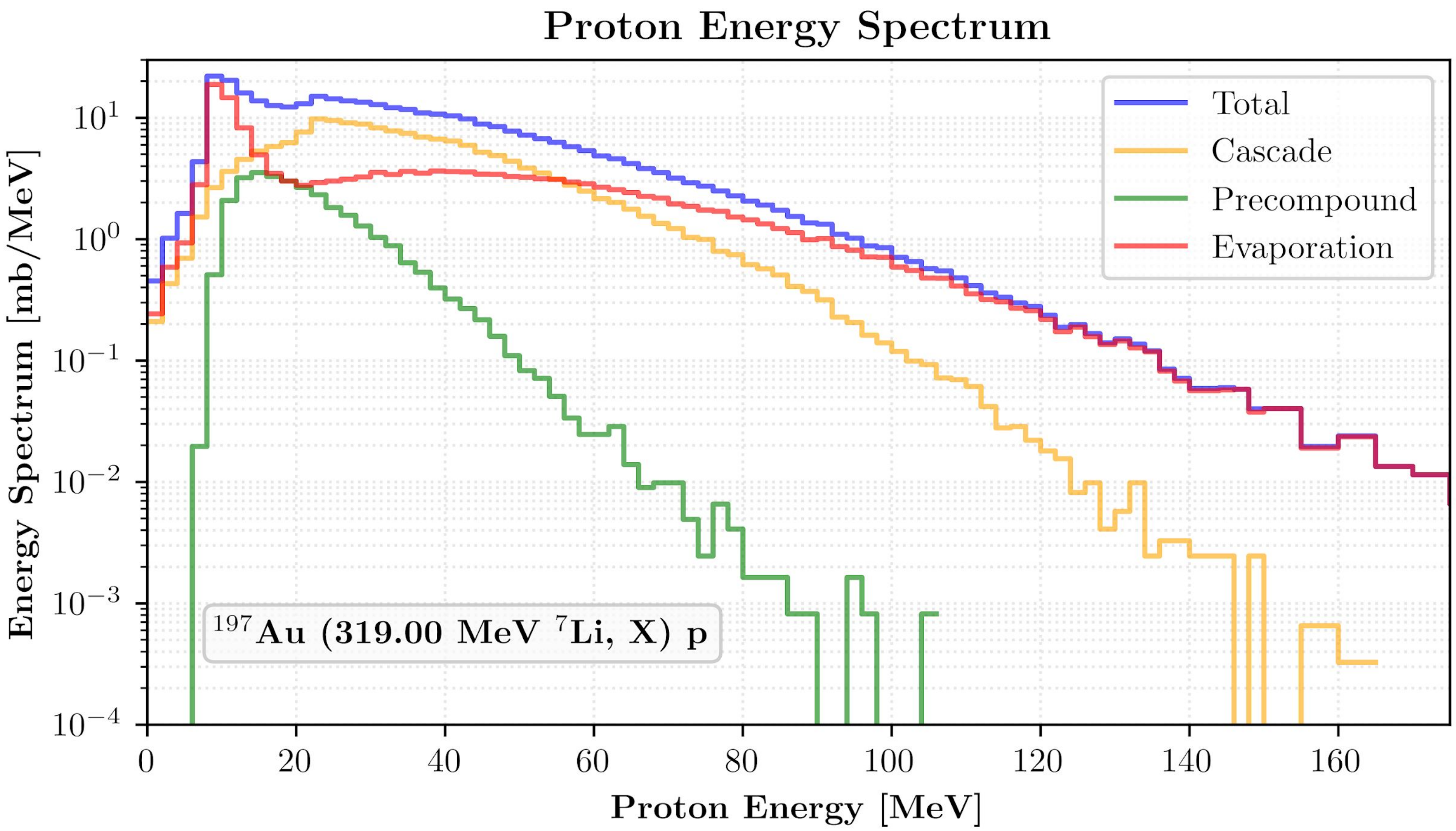


Results and Analysis (cont'd)[12]

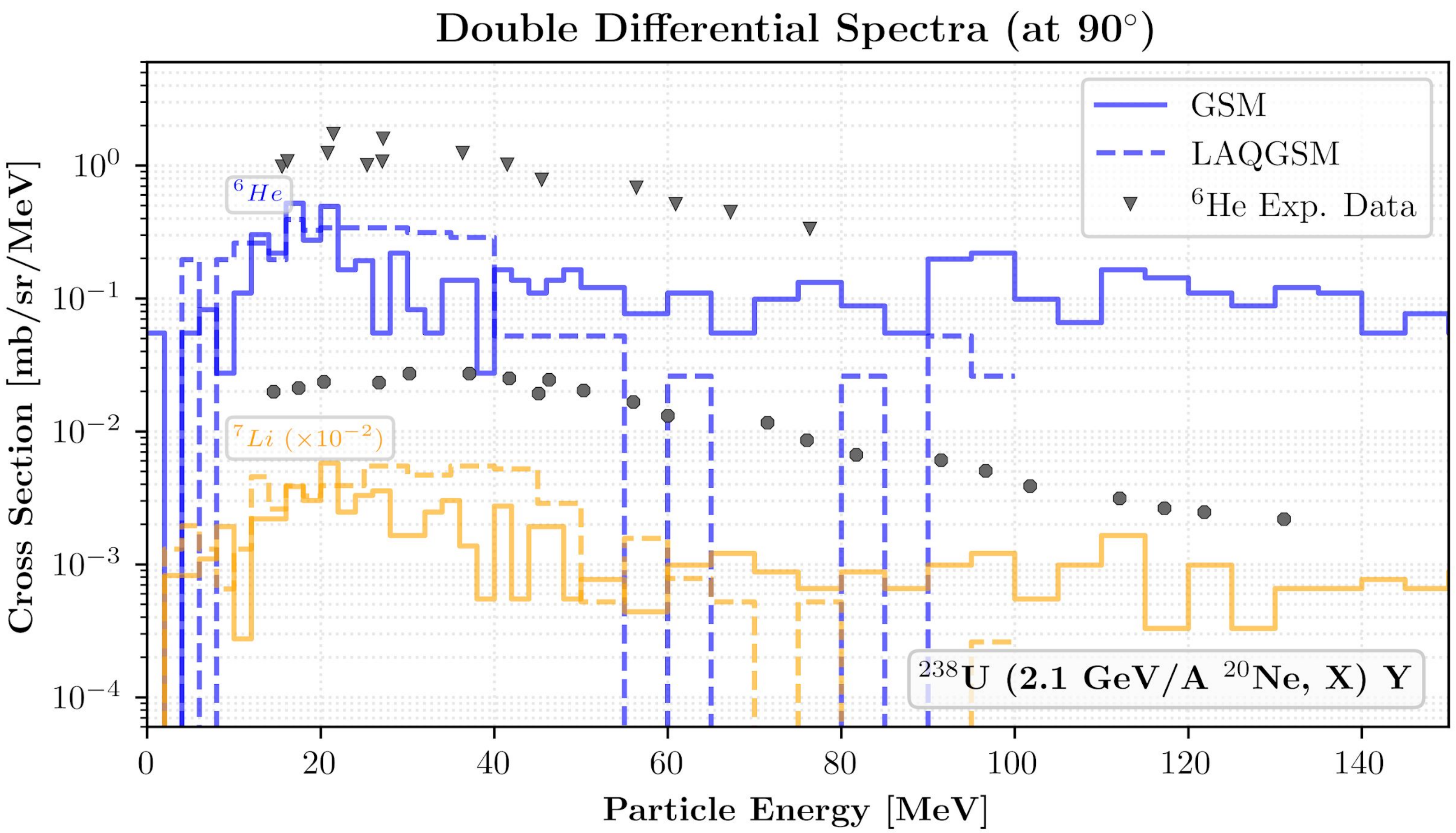


12. Exp. Data from J. Franz et al. "Neutron-Induced Production of Protons, Deuterons and Tritons on Copper and Bismuth". In: Nuclear Physics, Section A 510 (1990), p. 774. DOI: 10.1016/0375-9474(90)90360-X. URL: http://dx.doi.org/10.1016/0375-9474(90)90360-X.

Results and Analysis (cont'd)



Results and Analysis (cont'd)[13]



13. Exp. Data from J. Gosset et al. "Central Collisions of Relativistic Heavy Ions". In: Physical Review, Part C, Nuclear Physics 16 (1977), p. 629. DOI: 10 . 1103 / PhysRevC . 16 . 629. URL: http://dx.doi.org/10.1103/PhysRevC.16.629.

Results and Analysis (cont'd)

- Concisely put, GSM…
  - Models the CEM regime well
    - Incident protons, neutrons, pions, and photons
  - Produces results like LAQGSM with more predictive power in <u>most</u> cases
  - Physics could benefit from…
    - An improved Coulomb barrier
    - An improved Evaporation model
    - Tuned parameterizations

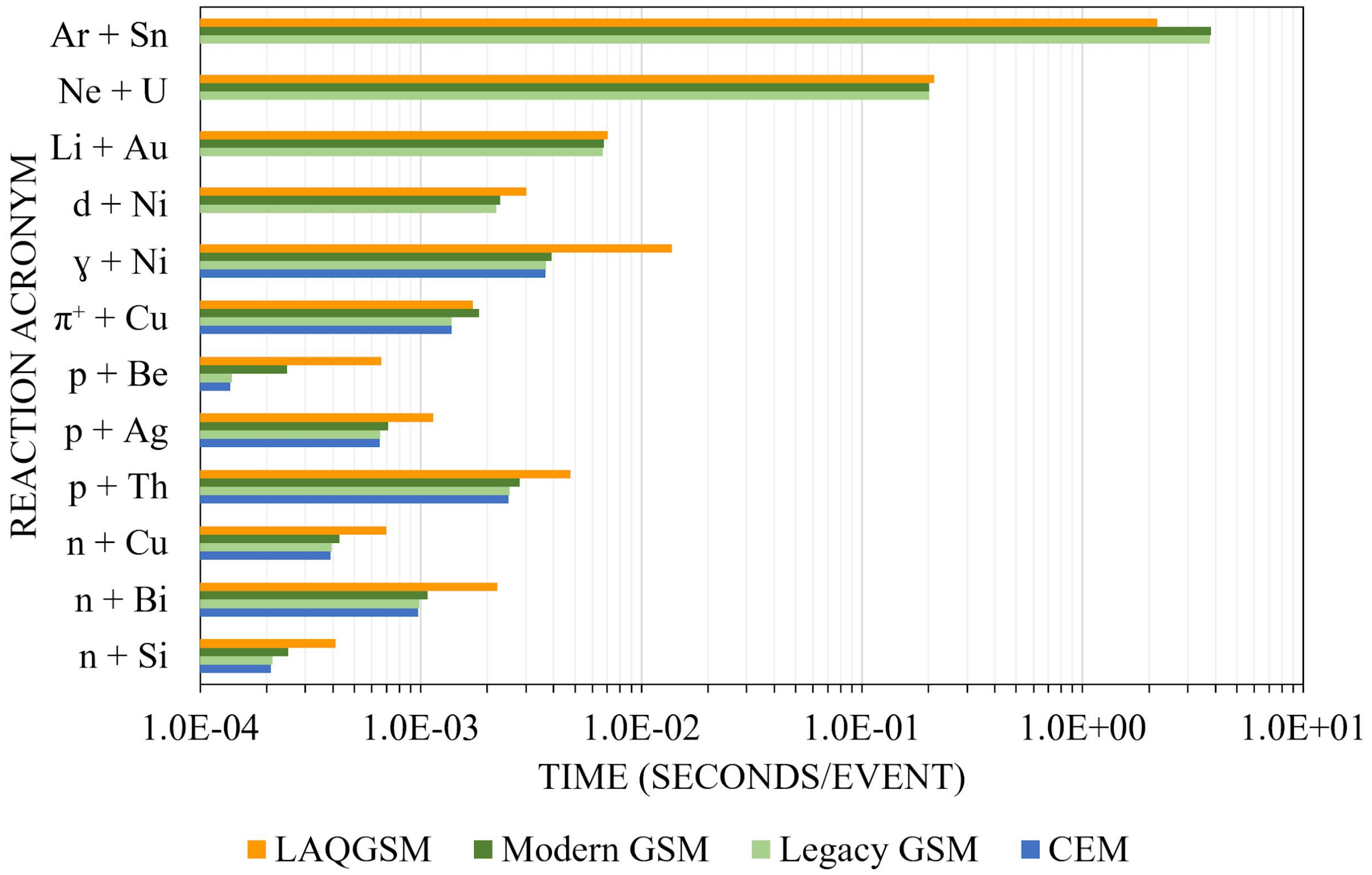
- More validation for incident light- and heavy-ions is desired

Results and Analysis (cont'd)

- Computation times
  - As-is, varies by target size, incident energy, and simulation verbosity
    - Low verbosity computation times are similar to those of legacy GSM
- Similar times to the CEM while incurring little penalty for the benefits of a modern object-oriented structure.
- Typically smaller than those of the LAQGSM

Results and Analysis (cont'd)

Computation times of the CEM, GSM, and LAQGSM event generators



Contents

ROAR

Summary

- The modernized GSM provides a major improvement from its predecessors, the CEM and the LAQGSM event generators.
- Complex physics models are abstracted and provide simple setup, pre-processing, usage, and post-processing to software clients and end-users.

TABLE I. Some API Procedures of the GSM Event Generator

Name	Description
updateOptions	Modifies the internal simulation option object utilized by the GSM
properlyConstructed	Returns a logical flag indicating the construction state of the GSM object
simulateEvent	Simulates a single spallation event
generateOutput	Generates an output file based on the provided output options object
fermiBreakUpInterface	Interface to the Fermi breakup model
formNuclei	Establishes default fields of the projectile and target nuclei objects

Summary (cont’d)

- GSM provides reusable and portable modular software components
- Improved reliability of the model
- Good predictive power for light incident particles
  - Improvement to physics desired for better light- and heavy-ion validation

Contents

ROAR

Future Work

- Finish modernization of the GSM and its sub-models
- Further improve structure of the GSM and each object utilized
- Physics improvements
- Parallelization via OpenMP and MPI
  - What are the scaling efficiencies for the GSM?

Future Work

- Improvements to the output data

- Implementation in software clients, *e.g.* MCNP, Geant4, etc.
  - Deprecation of the CEM and LAQGSM event generators in MCNP and others
  - Test the GSM API robustness

Contents

- Introduction: What is GSM?
- Spallation Physics Overview
- Modernization and API
- Results and Analysis
- Summary
- Future Work
- Acknowledgments

ROAR

Acknowledgments

- We would like to thank Dr. Stepan Mashnik, Dr. Arnold Sierk, and Dr. Konstantin Gudima for their thorough understanding of the multitude of complex physics that is discussed in this document.

- This work was carried out with funding provided by the Idaho State University and with funding provided by the Los Alamos National Laboratory, under Idaho State University subcontract number 385443 and through the advanced simulation and computing program.

Thank you!

Questions?

References

1. C. Juneau. The Development of the Generalized Spallation Model. Master thesis. Idaho State University, 2019
2. S. Mashnik and A. Sierk. CEM03.03 User Manual. Tech. rep. LA-UR-12-01364. Los Alamos National Laboratory, 2012.
3. K. Gudima, S. Mashnik, and A. Sierk. User Manual for the Code LAQGSM. Tech. rep. LA-UR-01-6804. Los Alamos National Laboratory, 2001.
4. Development of the Generalized Spallation Model. Vol. 117. Computational Tools for Radiation Protection and Shielding. ANS, 2017, pp. 1182–1185.

References

5. L. Kerby. “Precompound Emission of Energetic Light Fragments in Spallation Reactions”. PhD thesis. University of Idaho, 2015.
6. S. Furihata. “The GEM Code Version 2 Users Manual”. In: Mitsubishi Research Institute, Inc., Tokyo, Japan (2001).
7. K. Gudima et al. “The Coalescence Model and Pauli Quenching in High-Energy Heavy-Ion Collisions”. In: Joint Institute for Nuclear Research Report JINR-E2-83-101, Dubna (1983), pp. 458–460.
8. V. Weisskopf. “Statistics and Nuclear Reactions”. In: Physical Review 52.4 (1937), p. 295.
9. E. Fermi. “High Energy Nuclear Events”. In: Progress of theoretical physics 5.4 (1950), pp. 570–583.

References

10. R. Green, R. Korteling, and K. Jackson. “Inclusive Production of Isotopically Resolved Li through Mg Fragments by 480 MeV p + Ag Reactions”. In: Physical Review, Part C, Nuclear Physics 29 (1984), p. 1806. DOI: 10.1103/PhysRevC.29.1806. URL: http://dx.doi.org/10.1103/PhysRevC.29.1806.
11. S. Leray et al. “Spallation Neutron Production by 0.8, 1.2, and 1.6 GeV Protons on Various Targets”. In: Physical Review, Part C, Nuclear Physics 65 (2002), p. 044621. DOI: 10.1103/PhysRevC.65.044621. URL: http://dx.doi.org/10.1103/PhysRevC.65.044621.
12. J. Franz et al. “Neutron-Induced Production of Protons, Deuterons and Tritons on Copper and Bismuth”. In: Nuclear Physics, Section A 510 (1990), p. 774. DOI: 10.1016/0375-9474(90)90360-X. URL: http://dx.doi.org/10.1016/0375-9474(90)90360-X.

References

13. J. Gosset et al. “Central Collisions of Relativistic Heavy Ions”. In: Physical Review, Part C, Nuclear Physics 16 (1977), p. 629. DOI: 10 . 1103 / PhysRevC . 16 . 629. URL: http://dx.doi.org/10.1103/PhysRevC.16.629.