FUDGE and GIDI+ support for GNDS

Presented at OECD/NEA/WPEC EG-GNDS

Bret Beck

Nov 2020

LLNL-PRES-817411

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC

Lawrence Livermore National Laboratory

Outline

- Overview of LLNL GNDS support
- map, multi-group and flux files
- FUDGE
- EMU
- GIDI+
  - PoPI
  - GIDI
  - MCGIDI
- To do
- Code releases

Overview

- FUDGE is used to convert ENDF-6 and LLNL ENDL files into GNDS, and GNDS to ENDF-6.
- FUDGE is used to modify and plot GNDS data.
- We are developing EMU to sample realizations.
- FUDGE is used to process GNDS for Monte Carlo and multi-group transport (deterministic).
- GIDI is used by our transport codes to read GNDS data.
- GIDI is used by our deterministic transport codes to access multi-group data.
- MCGIDI is used by our Monte Carlo transport codes to lookup and sample data.

Overview II

- FUDGE and GIDI/MCGIDI work on an individual projectile/target/evaluation file.
  - Called a protare in GIDI and MCGIDI
    - PROjectile + TARget + Evaluation
- A list of protares are referenced in a “map” file to create a library.
- We developed multi-group and flux formats based on GNDS containers that are used for processing (i.e., grouping) and collapsing.
- FUDGE and GIDI work with LLNL’s GNDS 1.10 and are almost compatible with 2.0. Note, the official version of GNDS is 1.9.
- In addition to GNDS and PoPs files, FUDGE and GIDI support map, multi-group and flux files.

Map file -- test.map

```
<map library=”test22" format="0.2">

<protare projectile="n"  target=”O16"  evaluation=”fromJoe”
                         path=”fromJoe/n-008_O_016.xml"/>
<protare projectile="n"  target=”U235"  evaluation=”fromJoe”
                         path=”fromJoe/n-092_U_235.xml"/>

<protare projectile="n"  target=”U235"  evaluation=”Ian”
                         path=”fromIan/n-092_U_235.xml"/>
<protare projectile="n"  target=”U238"  evaluation=”Ian”
                         path=”fromIan/n-092_U_238.xml"/>

<TNSL projectile="n"  target="OinBeO"  evaluation="ENDF/B-8.0”
                         path="tsl/tsl-OinBeO.xml">
                         standardTarget="O16" standardEvaluation="ENDF/B-8.0”/>

<import production.map></map>
```

New multi-group boundaries and flux file

- Multi-group boundaries format uses GNDS <group> node to store the label and boundaries for a group.

```
<group label="LLNL_gid_23">
  <grid index="0" label="energy" unit="MeV" style="boundaries">
   <values>2.0908e-6 2.0908e-4 1.8817e-3 .010245 .07002 0.27097 .7527 15.754</values></grid></group>
```

- Flux stored as f(T,E,mu) using a GNDS 3d function.

```
<XYs3d label="LLNL_fid_1">
  <axes>
   <axis index="3" label="temperature" unit="MeV/k"/>
   <axis index="2" label="energy_in" unit="MeV"/>
   <axis index="1" label="mu" unit=""/>
   <axis index="0" label="flux" unit="1/s"/></axes>
  <XYs2d outerDomainValue="0.0">
   <Legendre outerDomainValue="0.0"><values>85</values></Legendre>
   <Legendre outerDomainValue="21.0"><values>85</values></Legendre></XYs2d></XYs3d>
```

FUDGE

- For Updating Data and Generating Evaluations
- User interface in python.
- FUDGE is like a toolkit.
- Python makes the interface easier for the users.
- Computationally intensive stuff written in C and C++.
  - E.g., heating cross sections, multi-grouping distributions.
  - Users do not interact directly with the C and C++ codes, the python interface handles that.
- Next released version
  - will be for python 3.6+.
  - will use pip for installation.
- Currently working on version that can read/write GNDS 1.10 and 2.0(?) but uses 2.0 internally. That is, user must think 2.0 when using next version of FUDGE.

FUDGE scripts

- We are developing scripts to make it easier to examine and process GNDS files.

- Some examples include:
  - processProtare.py: Processes for Monte Carlo and multi-group transport
  - checkGNDS.py: Runs all FUDGE physics tests on a GNDS file
  - peek.py: Prints each reaction and brief information about each reaction's products.
  - diffGNDS.py: Does a partial diff of two GNDS files
  - buildMapFile.py: Creates a map file from a list of GNDS files
  - temperatures.py: List all temperature data in a GNDS file

```
(venv-3.7.2) # ./temperatures.py ~/GIDI_plus/GIDI/Test/Data/MG_MC3Ts/neutrons/n-008_O_016.xml
/g/g16/beck6/Git/GIDI_plus/GIDI/Test/Data/MG_MC3Ts/neutrons/n-008_O_016.xml
  temperature 0.0 K: eval
        temperature [K]         heated           griddedCrossSection    URR_probabilityTables      heatedMultiGroup       SnElasticUpScatter
    ------------------------------------------------------------------------------------------------------------------------------------------
                  300.1       heated_000           MonteCarlo_000                                  MultiGroup_000
                1.16e+04       heated_001           MonteCarlo_001                                  MultiGroup_001
                1.16e+06       heated_002           MonteCarlo_002                                  MultiGroup_002
```

FUDGE processing example for transport codes

- FUDGE command to process a GNDS file for deterministic and Monte Carlo transport at 3 temperatures

```
bin/processProtare.py -mc -mg -t 2.58522e-08 -t 1e-07 -t 1e-4 gnds.file.xml
```

- File contains
  - evaluation data
  - Reconstructed cross sections (if needed)
  - Coulomb + nuclear elastic μ cutoff (if needed)
  - average product data
  - pdf/cdf data for distributions (e.g., P(E'|E))
  - heated cross sections (at 3 temperatures in this case)
  - Common grid heated cross section for Monte Carlo (at 3 temperatures in this case)
  - multi-group data (at 3 temperatures in this case)
  - TNSL data, if present, are processed

URR processing is done separately (computationally intensive).

EMU

- Evaluations with Means and Uncertainties
  - Replaces KIWI which handles LLNL ENDL formatted data.
  - Main developer is Kyle Wendt.
- Written in Python.
- Uses mean and covariance data to create a realization.
- Currently a separate package from FUDGE but uses FUDGE to read/write and manipulate GNDS data.
- We will be releasing this also but probably not with the next FUDGE release.

GIDI+ main user packages

- PoPI
  - Properties of Particle Interface
  - C++ API to read and allow access to GNDS PoPs data
- GIDI
  - General Interaction Data Interface
  - C++ API to read and access to GNDS data
  - Developed to give access for transport codes
  - Follows outline of GNDS
  - Has Map and Protare classes
- MCGIDI
  - Monte Carlo General Interaction Data Interface
  - C++ API for use in Monte Carlo transport codes
  - Extracts data from a GIDI::Protare
  - Stores data in more suitable way for Monte Carlo transport
  - Cross section look up by temperature and projectile energy for host code
  - Samples a reaction for a protare
  - Samples outgoing particle data for a reaction

Directory structure of packages GIDI3, MCGIDI and PoPI

```
Makefile
Doc/          # Documentation
Speeds/       # Extra
Src/          # All *.hpp and *.cpp files
Test/         # A suite of tests run with "make check"
bin/          # Some useful executables  and their sources
include/      # "make" puts needed user *.hpp files here
lib/          # "make" puts needed user library files here
```

Example of executable in GIDI/bin:

```
directory     # Displays a list of all protares matching projectileID, targetID and evaluation.
```

Examples of a PoPI file, a map file and various protare files are found in the Test directory.

GIDI+ (or gidiplus)

- To use GIDI and MCGIDI requires additional packages. GIDI, MCGIDI and these additional packages are dubbed GIDI+.
  - pugixml-1.8
    - Third party XML parser
    - Written in C++
  - statusMessageReporting
    - Handles message passing between C packages
    - Written in C
  - numericalFunctions
    - Supports 1d numerical functions including addition, multiplication
    - Written in C
  - PoPI
  - GIDI
  - MCGIDI

statusMessageReporting and numericalFunctions are also used by FUDGE.

PoPI C++ API

- Property of Particles Interface (PoPI)
- Implements the PoPs part of GNDS
- Uses strings for particle IDs as defined in GNDS
  - (e.g., “O16”, “n”, “U235”, “u235_e6”)
- Current LLNL PoPs files
  - pops.xml (currently only defines ground state nuclei)
  - metastables_alias.xml (e.g., “Am242_m1” for “Am242_e2”)
  - LLNL_alias.xml (e.g., “92235” for “U235”)

Simple PoPI example

```
PoPI::Database pops( "pops.xml" );
pops.addFile( "metastables_alias.xml" )

PoPI::Particle const &O16 = pops.get<PoPI::Particle>( "O16" );
            //                O16 = pops.particle( "O16" );

std::cout << "O16 -> " << O16_3.ID( ) << std::endl;
std::cout << "mass = " << O16_3.massValue( "amu" ) << std::endl;
```

```
O16 -> O16
mass = 15.9949
```

GIDI C++ API

- General Interaction Data Interface (GIDI)
- A C++ API for reading a GNDS reactionSuite.
  - Uses PoPI to read the PoPs part.
- Retrieving and collapsing multi-group data for use in deterministic codes (or Monte Carlo but that is better handled by MCGIDI).
- Protare is a virtual class. Actual classes are ProtareSingleton, ProtareComposite and ProtareTNSL.
- Support reading/writing GNDS 1.10 and 2.0(?) but, like FUDGE, uses 2.0 internally.

Simple GIDI example

```
GIDI::Map map( “test.map”, pops );

GIDI::Construction::Settings construction( GIDI::Construction::e_all,
                                           GIDI::Construction::e_nuclearAndAtomic );
GIDI::Protare *protare = map.protare( construction, pops, PoPI::IDs::neutron, "O16" );

GIDI::Styles::TemperatureInfos temperatures = protare->temperatures( );

GIDI::Settings::MG settings( protare->projectile( ).ID( ), label, true );
GIDI::Vector crossSection = protare->multiGroupCrossSection( settings, particles );
```

typedef std::vector<Styles::TemperatureInfo> TemperatureInfos;

```
(venv-3.7.2) # temperatures.py ~/GIDI_plus/GIDI/Test/Data/MG_MC3Ts/neutrons/n-008_O_016.xml
/g/g16/beck6/Git/GIDI_plus/GIDI/Test/Data/MG_MC3Ts/neutrons/n-008_O_016.xml
  temperature 0.0 K: eval
       temperature [K]         heated           griddedCrossSection    URR_probabilityTables      heatedMultiGroup       SnElasticUpScatter
   ------------------------------------------------------------------------------------------------------------------------------------------
                 300.1       heated_000           MonteCarlo_000                                  MultiGroup_000
               1.16e+04       heated_001           MonteCarlo_001                                  MultiGroup_001
               1.16e+06       heated_002           MonteCarlo_002                                  MultiGroup_002
```

Script `temperatures.py`

```
temperature [K]     heated  griddedCrossSection   heatedMultiGroup
 ---------------------------------------------------------------
         300.1  heated_000       MonteCarlo_000     MultiGroup_000
      1.16e+04  heated_001       MonteCarlo_001     MultiGroup_001
      1.16e+06  heated_002       MonteCarlo_002     MultiGroup_002
```

```
▼<reaction label="n + H1" ENDF_MT="2">
 ▼<crossSection>
  ▶<XYs1d label="eval">
    ...
   </XYs1d>
  ▶<XYs1d label="heated_000">
    ...
   </XYs1d>
  ▶<Ys1d label="MonteCarlo_000">
    ...
   </Ys1d>
  ▶<gridded1d label="MultiGroup_000">
    ...
   </gridded1d>
  ▶<XYs1d label="heated_001">
    ...
   </XYs1d>
  ▶<Ys1d label="MonteCarlo_001">
    ...
   </Ys1d>
  ▶<gridded1d label="MultiGroup_001">
    ...
   </gridded1d>
  ▶<XYs1d label="heated_002">
    ...
   </XYs1d>
  ▶<Ys1d label="MonteCarlo_002">
    ...
   </Ys1d>
  ▶<gridded1d label="MultiGroup_002">
    ...
   </gridded1d>
```

MCGIDI C++ API for GNDS

- Monte Carlo GIDI (MCGIDI)
- A C++ API for Monte Carlo transport codes
- Can do LLNL model A and B (MCNP) upscatter for outgoing particles
- Supports broadcasting for MPI and GPUs
- Implemented in LLNL's Monte Carlo transport code Mercury

Simple MCGIDI example

```
double energy = 14.1, crossSection, reactionCrossSection;

MCGIDI::DomainHash domainHash( 4000, 1e-8, 10 );
MCGIDI::Protare *MCProtare = MCGIDI::protareFromGIDIProtare( *protare, pops, MC
                        particles, domainHash, temperatures, reactionsToExclude );

int hashIndex = domainHash.index( energy );
crossSection = MCProtare->crossSection( URR_infos, hashIndex, temperature, energy );
reactionCrossSection = MCProtare->reactionCrossSection( ir, URR_infos, hashIndex,
                        temperature, energy );

int reactionIndex = MCProtare->sampleReaction( URR_infos, hashIndex,
                                temperature, energy, crossSection, rng, rngState );
reaction->sampleProducts( MCProtare, energy, input, products );
```

To do

- Finish GNDS 2.0 support
- Expand PoPs database (e.g., add excited states)
- HAPI (Hierarchical API)
  - Will support GNDS in other “metalanguages” (XML, HDF5, XML/HDF5, etc.)
  - Need to also speed up load time (90% time spent converting double string to binary
- Add unit support to GIDI+
- Speed up MCGIDI and improve GPU support
- Update codes for GPUs where possible and beneficial (e.g., heating cross sections, calculating multi-group transfer matrices)

Code releases

- We are releasing all codes under https://github.com/LLNL
  - FUDGE
    - Named “fudge”
    - Version 4.2.3
      - Python 2.7
    - Next version
      - Python 3.6+
      - Pip install
    - BSD license (probably will switch to MIT license)
  - GIDI+
    - Named “gidiplus”
    - Version 3.18.129
    - GNDS 1.10 (internal LLNL version)

- Releasing all codes under MIT license, except currently FUDGE

We need to release a GNDS 2.0? version soon.