LA-UR-20-24025
Approved for public release; distribution is unlimited.

Title: Processing MCNP Elemental Edit Outputs

Author(s): Mehta, Vedant Kiritkumar
Armstrong, Jerawan Chudoung

Intended for: Report

Issued: 2020-06-02

Processing MCNP Elemental Edit Outputs

Vedant Mehta and Jerawan Armstrong
Los Alamos National Laboratory
Los Alamos, NM, 87545

1 Introduction

The Los Alamos National Laboratory's (LANL) Monte Carlo N-Particle® (MCNP®) [1] transport code version 6 has the capability for tracking particles on unstructured mesh (UM) geometry models [2, 3]. The MCNP UM feature was originally designed for the UM models created by the Abaqus/CAE software product [1]. The MCNP version 6.2.0 and later recognizes the UM models read from Abaqus INP files or MCNPUM files converted from Abaqus INP files. Other finite element analysis software packages may generate UM models and then convert these models into Abaqus INP formats.

A geometry has to be defined in a MCNP input file in order to perform a MCNP simulatuon. The MCNP geometry is organized into cells bounded by surfaces. This geometry is known as a constructive solid geometry or (CSG). It is difficult and time-consuming to build the CSG models for complicated geometry problems. For some applications, the users may spend several months to build the CSG geometries. The MCNP UM feature has been developed for performing calculations of complex geometry models. This capability is for tracking particles on the hybrid geometries where the

[1] MCNP® and Monte Carlo N-Particle® are registered trademarks owned by Triad National Security, LLC, manager and operator of Los Alamos National Laboratory for the U.S. Department of Energy under contract number 89233218CNA000001. Any third party use of such registered marks should be properly attributed to Triad National Security, LLC, including the use of the ®designation as appropriate. Any questions regarding licensing, proper use, and/or proper attribution of Triad National Security, LLC marks should be directed to trademark@lanl.gov. For the purposes of visual clarity, the registered trademark symbol is assumed for all references to MCNP within the remainder of this report.

finite element meshes. are embedded into the CSG cells. Both structured or unstructured meshes can be embedded into the CSG cells. The unstructured mesh is the collection of elements and thier nodes. The computer programs are used to generate the mesh geometries from the solid geometries. MCNP is not a mesh generation code. The mesh generation software packages should be used to generate the unstructured meshes for use in MCNP simulations.

The MCNP UM capability was originally developed for thermal mechanical analysis applications, but this feature has increasingly used for other applications including reactor calculations. We have modified the MCNP code for microreactor multiphysics calculations [4]. Microreactors are state-of-the-art reactor concept with power level rated between 1 kWe and 10 MWe. Using metallic fuel, metallic moderator, and heat pipes makes microreactors compact, truck-transportable, and suitable for several high-temperature civilian and military applications. However, there are many roadblocks that needs to be crossed before a successful deployment of moderated microreactor concept. Metal Hydrides are a great moderator for nuclear reactor as they improve neutron economy significantly. However, all hydrides share the same flaw. Under high temperature gradients, the hydrogen in the metallic bonds of hydride dissociates and migrates from hot zones to cold zones. In addition, under high temperature environment, hydrogen escapes the reactor core thus impacting the reactor operation. This migration of hydrogen during the reactor operation results in local changes in moderator material properties such as neutron cross sections, core power generation, thermal/mechanical feedback. Thus, high fidelity multiphysics modeling is required to properly understand the reactor dynamics during its operation. To do so, MCNP, a Monte-Carlo particle transport code, and ABAQUS, a finite-element engineering simulation software are used to develop a multiphysics framework called MARM for high temperature microreactor analysis. MARM is a MCNP ABAQUS based Reactor Multiphysics software package being developed at Los Alamos National Laboratory for microreactor applications. This suite of software is aimed to provide steady-state and transient coupled multiphysics analysis for microreactors including neutronics, heat transfer, mechanical stresses, mass diffusion, and extensions capability for additional user-defined physics. The main goal of this project is to enable high quality multiphysics analysis for engineering length scale microreactor applications.

The MCNP UM calculations are computationally expensive. Typically, the calculations are performed on the supercomputers. After the MCNP calcu-

lation is finished, the python script is used to process the MCNP outputs to write the new Abaqus input file for finite element analysis calculation. The Abaqus calculation is then performed where the outputs are written into the ODB file. The OBD file is the Abaqus binary output file format. We have developed a python code for processing a MCNP elemental edit output (EEOUT) file to produce a heat flux file for Abaqus calculations. This python code is presented in Appendix A.

References

[1] Abaqus/CAE. www.3ds.com/simulia.

[2] R. L. Martz and D. L. Crane. The MCNP6 Book on Unstructured Mesh Geometry: Foundation. Technical Report LA-UR 12-25478 Rev 1, Los Alamos National Laboratory, 2014.

[3] R. L. Roger. The MCNP6 Book On Unstructured Mesh Geometry: User's Guide For MCNP 6.2. Technical Report LA-UR 17-22442, Los Alamos National Laboratory, 2017.

[4] H. Trellue, S. Vogel, A. Long, V. Mehta, J. Armstrong, G. McKinney, A. Shivprasad, E. Luther, J. Payne, M. Cooper, T. Carver, B. Wilkerson, J. Wermer, and J. Bull. Demonstration of advances experimental and theoretical characterization of hydrogen dynamics and associated behavior in advanced reactors. Technical Report LA-UR-20-22974, Los Alamos National Laboratory, 2020.

Appendix A Python Code

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
  Code:        eeout_to_inp.py

  Authors:     Vedant Mehta
               NEN-5: Systems Design and Analysis
               Los Alamos National Laboratory

               Jerawan Armstrong
               XCP-3: Monte Carlo Codes
               Los Alamos National Laboratory

  Copyright (c) 2020 Triad National Security, LLC.  All rights reserved.

  Permission is hereby granted, free of charge, to any person obtaining a copy
```

```
  of this software and associated documentation files (the "Software"), to deal
  in the Software without restriction, including without limitation the rights
  to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
  copies of the Software, and to permit persons to whom the Software is
  furnished to do so, subject to the following conditions:

  The above copyright notice and this permission notice shall be included in all
  copies or substantial portions of the Software.

  THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
  IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
  FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
  AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
  LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
  OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
  SOFTWARE.
 """
import os, sys
import numpy as np

help_info = """
This script processes MCNP eeout file and produces body heat flux for ABAQUS.

HOW TO RUN:
python eeout_to_inp.py <eeout filename> <reactor power level in Watts>

Produces:
heatFlux.inp (FOR ABAQUS)

Add the resulting file in ABAQUS after '*Heat Transfer' as
'''
*INCLUDE, INPUT=heatFlux.inp
'''
"""

#-------------------------------------------------------------
def extract_eeout_data(fileIn,tally_num):

    fileIN = os.path.isabs(fileIn)

    if not os.path.isfile(fileIn):
        raise IOError('MCNP EEOUT file not found :: check directory path')

    # number of items in identificaiton segment line in EEOUT file
    # MCNP writes this data line to describe the numbers of various data in the
    # current data group. MCNP writes 6 items in this line
    num_id_segments = 6

     # read eeout file
    fr   = open(fileIn,"r")
    line = fr.readline()  # 1st line in EEOUT file is the title line

    # initialize vectors
    energy      = [] # MeV/g
    density     = [] # g/cm**3
    volume      = [] # cm**3

    ins_id         = [] # instance element ID
    start_element  = [] # start element number for each element ID
    end_element    = [] # end element number for each element ID

    # extract enery from this tally type
    e6_tally_id    = 'ENERGY_{:d}'.format(tally_num)

    # ========== Begin extract EEOUT data ====================
    indx     = 0
    found_e6 = False

    while True:
        line = fr.readline()
        if not line: break
        #print(line.strip())

        id_segments = line.split()
        if len(id_segments) != num_id_segments:
            print(">>>Check this invalid line:")
```

```
    print(line)
    raise ValueError('invalid identification segments in EEOUT file; check EEOUT file')

id_segments          = np.int_(id_segments)
num_char_in_title    = id_segments[0] # number of chracter in the title line
num_records          = id_segments[1] # number of records in the data-set after the title record
data_type            = id_segments[2] # data type:0=no data, 1=character, 2=integer, 3=real
num_items_per_record = id_segments[4] # number of items in each record
num_items_per_line   = id_segments[5] # length of each record
#print(id_segments)
#print(num_char_in_title)
#print(num_records)
#print(data_type)
#print(num_items_per_record)
#print(num_items_per_line)

if num_char_in_title > 0:
    title_line = fr.readline().strip()
else:
    title_line = ''
#print(title_line.strip()+'\n')

#if len(title_line) > 0:
#    print(line.strip())
#    print(title_line.strip())
#    print('\n')

if num_records > 0:
    if data_type == 1: # charecter data type
        if num_items_per_line != 1:
            print(">>>Check this invalid line:")
            print(line)
            raise ValueError('invalid number of iterms for chacter data type in id segment line')
        else:
            num_of_lines = num_records


    elif data_type==2 or data_type==3: # integer or real data type
        if num_records > 1 and num_items_per_record == 1:
            num_of_lines = num_records
        else:
            num_data = num_records * num_items_per_record
            if np.mod(num_data,num_items_per_line) == 0:
                num_of_lines = int(num_data/num_items_per_line)
            else:
                num_of_lines = int(num_data/num_items_per_line) + 1
    else:
        if data_type != 0:
            print(">>>Check this invalid line:")
            print(line)
            raise ValueError('invalid data type in id segment line')
else:
     num_of_lines = 0

if title_line.startswith('INSTANCE ELEMENT NAMES'):
    if data_type != 1: # must be character
        raise ValueError('invalid data type for INSTANCE ELEMENT NAMES; check EEOT file')

    for i in range(0,num_of_lines):
        line = fr.readline().split()
        for l in line: ins_id.append(l)

elif  title_line.startswith('INSTANCE ELEMENT TYPE TOTALS'):
    if data_type != 2: # must be integer
        raise ValueError('invalid data type for INSTANCE ELEMENT TYPE TOTALS; check EEOT file')

    # element number start end end for each element types
    # there are 6 element tyoes: tet, pent, hex tet2, pent2, hex2
    if num_items_per_line != 12:
        raise ValueError('invalid record length for INSTANCE ELEMENT TYPE TOTALS; check EEOUT file')

    for i in range(0,num_of_lines):
        spl  = np.int_(fr.readline().split())
        locs = []
        loce = []
```

```
            for i in [0,2,4,6,8,10]:
                if spl[i]==0: continue
                locs.append(spl[i])

            for i in [1,3,5,7,9,11]:
                if spl[i]==0: continue
                loce.append(spl[i])

            start_element.append(locs)
            end_element.append(loce)

    elif title_line.startswith('DATA OUTPUT PARTICLE') and e6_tally_id in title_line:
        found_e6 = True

    elif title_line.startswith('DATA SETS RESULT') and found_e6:
        found_e6 = False

        if num_records != 1: # must be 1 record
            raise ValueError('invalid number of records for DATA SETS RESULT; check EEOT file')

        if data_type != 3: # must be real
            raise ValueError('invalid data type for DATA SETS RESULT; check EEOT file')

        for i in range(0,num_of_lines):
            line = fr.readline().split()
            for l in line: energy.append(l)


    elif title_line.startswith('DENSITY'):
        if num_records != 1: # must be 1 record
            raise ValueError('invalid number of records for DENSITY; check EEOT file')

        if data_type != 3: # must be real
            raise ValueError('invalid data type for DENSITY; check EEOT file')

        for i in range(0,num_of_lines):
            line = fr.readline().split()
            for l in line: density.append(l)


    elif title_line.startswith('VOLUMES'):
        if num_records != 1: # must be 1 record
            raise ValueError('invalid number of records for DENSITY; check EEOT file')

        if data_type != 3: # must be real
            raise ValueError('invalid data type for DENSITY; check EEOT file')

        for i in range(0,num_of_lines):
            line = fr.readline().split()
            for l in line: volume.append(l)

    else:
        if num_of_lines > 0:
            for i in range(0,num_of_lines): fr.readline()


# ==============End Extract EEOUT data ==============
fr.close()

if len(ins_id) != len(start_element) or len(start_element) != len(end_element):
    raise ValueError('invalid INSTANCE ELEMENT NAMES and INSTANCE ELEMENT TYPE TOTALS; check EEOUT file')

for locs, loce in zip(start_element,end_element):
    for s,e in zip(locs,loce):
        if s >= e:
            print(s)
            print(e)
            raise ValueError('invalid INSTANCE ELEMENT TYPE TOTALS; check EEOUT file')

# MCNP write the 1st record of energy to 0.0
if len(energy)-1 != len(density):
    print('len energy : ',len(energy))
    print('len density : ',len(density))
    print(" READING ERROR :: length of energy and density vectors not consistent \n \n")
    print(" INCORRECT INTERPOLATION :: DON'T TRUST DATA \n \n")
```

```
        raise ValueError('invalid energy and density data points')

    # MCNP Energy in MeV/g by default
    density = np.float_(density)
    volume  = np.float_(volume)
    energy  = np.float_(energy)

    return ins_id, start_element, end_element, density, volume, energy


#-------------------------------------------------------------
def write_heat_flux(fileIn, fileOut, Watt, amp, flux, tally_num):
    ins_id, start_element, end_element, density, volume, energy = extract_eeout_data(fileIn, tally_num)

    el_energy = density*energy[1:]            # MeV/cc
    esum      = np.sum(volume*el_energy)      # MeV
    power     = el_energy/esum                # /cc

    # write dflux file
    fw = open(fileOut,"w")

    fw.write(amp+'\n')
    fw.write('0, '+str(Watt)+'\n')
    fw.write('1, '+str(Watt)+'\n')
    fw.write('1e+33, '+str(Watt)+'\n')
    fw.write(flux+'\n')

    for cid, locs, loce in zip(ins_id,start_element,end_element):
        j = 0
        for s, e in zip(locs,loce):
            for m in np.arange(s-1,e):
                j = j+1
                fw.write(cid+'.'+str(j)+', bf, '+str(power[m])+'\n')
    fw.close()

    print(" ABAQUS thermal input file generated :: "+fileOut+'\n')

    return None


#-------------------------------------------------------------
if __name__ == '__main__':
    import argparse
    import time

    parser = argparse.ArgumentParser(prog='eeout_to_inp',
                                     description='This script processes MCNP eeout file \
and produces body heat flux for ABAQUS.')

    parser.add_argument('eeout_filename',
                        type=str,
                        help="MCNP EEOUT file name")

    parser.add_argument('power_level_in_watts',
                        type=float,
                        help="reactor power level in Watts")

    parser.add_argument("-o", "--output", metavar="<heatFlux.inp>",
                        default='heatFlux.inp', type=str,
                        help="heat flux file name")

    parser.add_argument('-em', '--embee',  metavar='num',
                        default=6, type=int,
                        help='number in EMBEE card')

    args = parser.parse_args()

    if args.eeout_filename is not None and args.power_level_in_watts is not None:
        fileIn    = args.eeout_filename
        Watt      = float(args.power_level_in_watts)
        fileOut   = args.output
        tally_num = args.embee

        amp     = '*amplitude, name=thermalAmplitude'
        flux    = '*dflux, amplitude=thermalAmplitude'
```

```
        print(help_info)
        print('>>> Caution :: Assuming DEFAULT EMBEE6 units from MCNP \n')
        print('    Reading File :: '+fileIn+'\n')
        print('    Power set to :: {:f}  watts\n'.format(Watt))

        start_time = time.time()
        write_heat_flux(fileIn, fileOut, Watt, amp, flux, tally_num)
        time_s = round(time.time() - start_time, 5)
        time_m = round(time_s/60., 1)
        time_h = round(time_s/3600., 2)

        message = '\n>>>Code completed in {:f}  seconds [{:f} minutes \
|| {:f} hours]\n'.format(time_s, time_m, time_h)
        print(message)
```