VISUAL SIMULATION STEERING FOR A 3D NEUTRON TRANSPORT AGENT CODE SYSTEM

Jovana Knežević

Technische Universität München
Computation in Engineering
Arcisstr. 21 80290, Munich, Germany

knezevic@bv.tu-muenchen.de

Hermilo Hernández

The University of Utah
Nuclear Engineering Program
50 S. Central Campus Dr, Salt Lake City, UT, 84112, USA
Hermilo.Hernandez@utah.edu

Thomas Fogal

The University of Utah
Scientific Computing and Imaging Institute
72 S. Central Campus Dr, Salt Lake City, UT, 84112, USA
tfogal@sci.utah.edu

Tatjana Jevremovic

The University of Utah
Nuclear Engineering Program
50 S. Central Campus Dr, Salt Lake City, UT, 84112, USA
Tatjana.Jevremovic@utah.edu

ABSTRACT

We have integrated a framework for computational steering developed at the Technische Universität München [1] into the AGENT [2] codebase. This framework allows for the arbitrary interruption of any simulation system for the purpose of modifying internal state. Once the simulation is interrupted, the solver, meshing and resolution parameters, and convergence variables can all be modified to more appropriate values. Depending which settings have been modified, remeshing and refined resolution are then performed, and values are copied from the old set of resolution parameters onto the new set. Then the simulation restarts at whatever state - for example, timestep - it reached previously, without requiring a resubmission into the job control system. The utility of this computational steering process is exemplified during the 3D simulation of the University of Utah's 100 kW TRIGA reactor (UUTR) core with the AGENT code. In this paper we present an example in which initially the mesh resolution of the UUTR core input set up with a wide ray separation of 1.2 cm that as expected does not converge to the specified criteria after 600 iterations; the AGENT code version we used to demonstrate this new feature does not initiate any of other existing iteration acceleration techniques. Thus, using the aforementioend computational steering framework, the simulation is interrupted after 249 iterations. The ray separation is then changed to a smaller value and after 351 iterations the solution meets the required convergence criteria. The second example illustrates how the user can accelerate the convergence: starting from a coarse mesh resolution an interruption is performed after a few iterations to refine the initial resolution. There is a significant reduction in the CPU time (~ 22.0 %) following this procedure compared to finding the solution using only the refined resolution.

1. INTRODUCTION

We use the AGENT simulation framework for modeling reactor physics in configurations common to standard research reactors, current power reactors and any future reactor designs. AGENT solves the 3D Boltzmann transport equation using the method of characteristics (MOC). A number of rays are generated for each azimuthal and polar direction, and intersections are found for each surface within the domain. These form ray segments, which provide the basis for sampling a discretized Boltzmann equation. In three dimensions a 2D version may be employed with a 1D factor to account for axial leakage between reactor core slices.

This method of simulation requires a large number of parameters to be configured by the user. The user must choose which simulation method to utilize, the 2D/1D coupled model with MOC or nodal expansion method (NEM) used at the axial direction, inclusive of a variety of meshing parameters, as well as a set of convergence variables. Meshing and resolution parameters include the number of azimuthal and polar angles, the number of rays (representing neutron tracks of motion), meshing of the

zones, and the number of boundary edges along each side of the reactor core assemblies. Convergence parameters include *k*-values and scalar neutron flux.

Improper settings of these parameters can lead to excessive computational or memory requirements, leading to less properly converged neutron transport solution. However, in many cases proper values are not easily understood and by nature of the Method of Characteristics (MOC) based codes require for the user to complete a detailed survey toward the best estimate solution. This unnecessarily exacerbates the computational cost of simulating particular phenomena. To resolve this, we employ a computational steering approach. Computational steering is a powerful concept that allows scientists to interactively control a computational process during its execution in order to gain insight concerning parameters, algorithmic behavior, and opportunities for optimization.

2. THE AGENT CODE

AGENT (Arbitrary GEometry Neutron Transport) is a deterministic code that is used for 2D and 3D detailed, accurate and yet computationally efficient nuclear reactor modeling. The multi-group neutron transport equation is solved with the MOC. Compared to other deterministic and the MOC codes, the main AGENT advantage lies on the use of the so called *R*-functions to describe the reactor geometry and generate all geometrical data needed for the straightforward MOC solution. We have published a number of conference and journal papers on AGENT code physics, structure and accuracy in last more than ten years, but we usually refer to a paper listed as Ref [2] as the paper that gives the best overview of all unique aspects of the AGENT code. Thus, in this paper we just point in brief on the main aspects of the code philosophy that are of interest to simulation steering.

The AGENT code is optimized for robustness, simplicity, accuracy, and efficiency while supporting a full treatment of neutron transport in highly heterogeneous reactors' designs. The robustness of the *R*-function based geometrical module is assured through the sequential generation of the geometry and automatic sub-meshing of complex reactor lattices. The simplicity of reactor geometry description and selection of the parameters for accurate treatment of neutron propagation is achieved through the hierarchical organization of geometrical primitives (that could be pre-meshed into even smaller material zones) into more complex shapes (to be meshed based on the mesh of the primitives or to be meshed independently) using the *R*-function formula to describe the domain areas of all involved geometrical shapes. The accuracy is comparable to the Monte Carlo codes and is obtained by following neutron propagation through defined real geometrical domains subdivided into as many small material zones as user would like to select. The efficiency is maintained through a set of acceleration techniques introduced in all important calculation levels.

The *R*-functions modeler [3] used to represent complex domains through the combination of simple primitives into a single analytical equation allows great flexibility in hierarchical organization of any type of reactor geometry. The modeler is equally general as Monte Carlo combinatorial geometry based approach, but incomparably simpler and importantly faster. AGENT modeler permits automatic generation of flat-flux zones that is required for accurate MOC calculation. Graphic user interface is available at any of currently available computer platforms including mobile technologies, [4].

3. SIMULATION STEERING

Computational steering allows scientists to interactively control a computational process during its execution in order to gain insight concerning parameters, algorithmic behavior, and opportunities for optimization [5]. The first challenge introduced by the simulation steering concept is the matter of *instantly* communicating the desired changes to the simulation program at runtime. There are two primary state-of-the-art approaches for doing this: *checkpointing* and process interruption.

3.1 *Checkpoints* implementation

The first approach introduces the idea of *checkpoints* inserted at fixed places in the code, testing if anything has changed on the user's side. Such an approach has several disadvantages. First, it involves major code modifications, making the implementation of such an approach tedious for the end-user (researcher) writing the simulation code. As a consequence, despite the advantages that computational steering may offer, researchers become reluctant to apply the concept in their applications. The second disadvantage is the difficulty in making the right decision as to *where* to insert these checkpoints, since the answer to that question is entirely simulation dependent, and, moreover, problem-size dependent. In order to guarantee the *immediate* response of the computational model to user modifications, one would have to estimate this in advance, based on the experience gained by previous program runs. Another disadvantage is the inevitable trade-off between frequent polling (at considerable computational cost) and responsiveness to new user input. In addition to the insertion of checkpoints, a common approach is assigning one thread per simulation process *exclusively* to check for updates from the user, but this presents logistical issues in coordinating access to shared data (Figure 1).

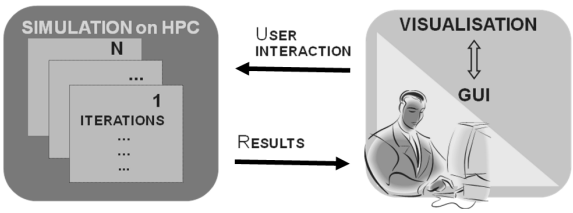

Figure 1. *Typical use case scenario.*

3.1 Interruption implementation

We chose to use an elegant and straightforward integration framework, previously proven to give excellent results in other engineering applications [1,7]. This framework provides instant responses user interaction with only minimal code modifications to the simulation. The main idea of the framework is to exploit user-generated interrupts, i.e. *signals*, to interfere with the regular course of the simulation.

Threads in any application programs execute sequentially if every instruction runs properly. In case of an error or other anomaly during execution, the operating system utilizes signals to notify the program. There are many scenarios in which the operating system will generate a signal, and one of them is when the user sends an interrupt to the program. In common UNIX-based environments, this consists of a Ctrl-C key combination. The default action for such a signal is to terminate the process, however this framework uses a signal handler to override that operation and provide the user with the opportunity to modify the state of the running simulation.

The user may send the signal at any point during program execution. However, this means that the program could be executing any arbitrary piece of code at that point in time; if the simulation was interrupted after acquiring a resource, we do not want to alter the program state such that the resource is never released. To workaround this problem, instead of directly modifying important state, the signal handler simply rewrites loop indices so that inner loops are considered 'done'. When the signal handler returns, control is resumed from whatever point of execution was suspended by entering the signal handler. This ensures any required cleanup occurs whilst avoiding the heavy cost of finishing the current iteration at the old data values. Finally, the simulation code is instrumented at its outermost loop to check and see if a user interruption has occurred, and if so it updates the relevant simulation variables.

For this work, updates may refer to changing the convergence criteria parameters, resolution parameters (such as the number of polar and azimuthal angles), maximal number of iterations, etc. Depending on the nature of the modification, appropriate re-initialization steps for the data have to be taken at the beginning of the new iteration, to ensure the correct execution of the program.

Illustration: The whole computation is intended to be restarted by manipulating the iteration vector i = (idx_1, idx_2, ... idx_n), i. e. the loop indices idx_i of all loops by setting it for each loop index to be some value out of its actual range, as shown in the following pseudo code:

```
for (t ← T0 to TN) do          // iterations over time
  reinitialize_data()          // (re)initialize MAX1, MAX2
                               // and other necessary data
  for (idx1 ← 1 to MAX1)       // in the case of interrupt, MAX1
                               //and MAX2 are both set to -1
    for (idx2 ← 1 to MAX2)
      process(data[idx1][ idx2])  // can be interrupted at
                                  // any point
```

4. VISUALIZATION

To make an informed decision on the new parameter set, a user must be able to understand the efficacy of the current solution. Thus the simulation periodically outputs the current state to disk. Visualization software converts that data into a representable form, and the results are displayed for the user's evaluation.

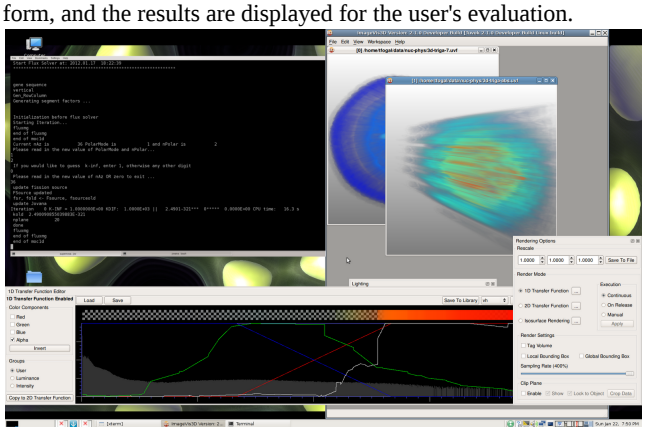

Figure 2. *Instrumented AGENT session: the user runs AGENT on a remote server (terminal, top left); as iterations complete, visualizations come available and appear in the locally-running ImageVis3D client*

A common mode of operation for simulations is to run the simulation code on a remote computing resource, such as a supercomputer. This presents a problem for visualization software, which typically runs on the user's desktop and therefore cannot directly access the data. To solve this problem, we have implemented a client/server solution: a simple daemon process runs on the server and brokers access to the data to clients. Client applications are presented with a list of available iterations of the currently running simulation, which is dynamically updated as the simulation progresses. Requesting a data set sends it over to the client machine and automatically loads up the *ImageVis3D* volume rendering tool [6] to visualize the data. Figure 2 shows the use of ImageVis3D client to visualize the main UUTR core parameters after AGENT multiple iterations have been completed.

5. RESULTS

Since the AGENT code solves the neutron transport equation using an iterative process, two criteria are applied to indicate whether the calculation has converged after each iteration, or did not. The first criterion is the relative difference for the multiplication factor between the iterations, defined as

$$k_{diff} = \left| \frac{k_{new} - k}{k} \right| \qquad (1)$$

The second criterion is the maximum relative difference of zone flux for all zones, which is defined as

$$\varphi_{diff,\max} = \max\left(\left|\frac{\varphi_{i,new}-\varphi_i}{\varphi_i}\right| \text{ for all zone } i\right) \quad (2)$$

When $k_{diff} < \varepsilon_1$ and $\phi_{diff,max} < \varepsilon_2$, the calculation is considered converged and will stop; otherwise, it will continue on to a subsequent iteration until convergence or until reaching a maximum (user defined) number of iterations. For the UUTR core the selected values are: $\varepsilon_1 = 0.00001$ and $\varepsilon_2 = 0.0001$.

The AGENT code MOC parameters required to solve the transport equation for heterogeneous reactor cores are:

a. *Number of polar angles (for the UUTR simulations two polar angles are chosen, with weights based on the study of Leonard and McDaniel [8]. AGENT also permits the use of more than two polar angles by using the CACTUS quadrature approximation [9].*
b. *Number of azimuthal angles (n_θ).*
c. *Ray separation (δA).*
d. *Number of boundary edges (n_b).*

The following examples illustrate an initial study focused on the utility of the computational steering integration framework applied to the deterministic code AGENT.

5.1 Monitoring the AGENT live simulation

In the case the user started the simulation with an incorrect initial parameters (for example, n_θ =4, n_b = 22 and $\boldsymbol{\delta A}$ =1.2 cm), the solution converges for k_{diff}, but giving an inaccurate value for k. Meanwhile, the convergence towards ε_2 follows a slow trend. If we interrupt the simulation (we selected to interrupt at 249th iteration) by conserving the same mesh but refining the MOC resolution parameters as δA=0.9 cm, the solution will improve toward best estimate but in a shorter computation time.

Figure 3 illustrates the value of k_{diff} as a function of the number of iterations. The peak at the iteration number of 250 is due to the value of k monotonically increasing from its initial value at the uninterrupted case.

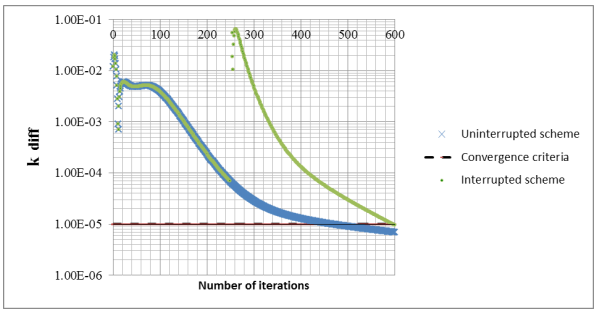

Figure 3. *k_{diff} as a function of number of iterations by refining the MOC resolution parameters*

Similarly, Figure 4 shows the values for $\phi_{diff,max}$ (scalar neutron flux). At the iteration number 600, the interrupted scheme reaches the convergence criteria.

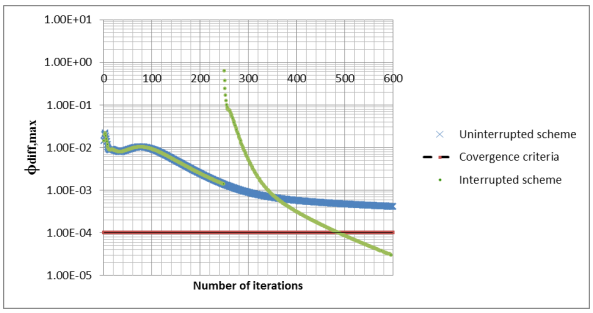

Figure 4. *$\phi_{diff,max}$ as a function of number of iterations by refining the MOC resolution parameters*

5.2 Interrupting AGENT live simulation achieving higher accuracy and reducing total CPU time

In this example we show the effect of the interruption early on during the AGENT simulation of the University of Utah TRIGA research reactor. The AGENT iterative process starts with the low-resolution parameters: n_θ =8, δA=1.2 cm and n_b=22 (in accordance with the previous section these values by themselves do not meet the convergence criteria but creates an initial solution estimate). The interruption is introduced after 82.5 seconds, at the 41st iteration (when k > 1.0). Two independent cases based on MOC resolution were analyzed after the interruption:

(a) *Medium resolution with the new values of n_θ =18, δA=0.45 cm and n_b =22.*
(b) *Higher resolution with the new values of n_θ =32, δA=0.1 cm and n_b =22.*

Table 1 shows the time and number of iterations it takes for AGENT to solve the transport equation for the UUTR. It can be observed that the low-to-medium interruption procedure consumes ~ 84.5% of the CPU-time spent solving the transport equation at medium-resolution level. This gain in time increases with the low-to-high interruption procedure which consumes ~ 78% of the CPU-time with respect to a high-resolution level

computation. At this early stage of computation steering module development, we observe the time savings in the order of 30%.

Table 1. AGENT code performance vs MOC resolution parameters

Modeling the UUTR	Total number of iterations	CPU-time (s)
Medium-Resolution	157	1,000
Low-to-Medium Resolution	159	843
High-Resolution	157	8,511
Low-to-High Resolution	158	6,634

6. CONCLUSIONS

We have introduced a new feature into the AGENT code based on the utility of dynamically-adjusted resolution parameters in a simulation framework. Although in its early stage of the development, this streaming simulation key provides a unique code feature non-existent in neutron transport codes based on deterministic methods. This new framework provides a number of advanced ways in running AGENT code allowing user to monitor the solution for the best estimate in a shorter computation time. For example, by starting AGENT solution with a coarse resolution and by interrupting the solution early on, users can accelerate the overall time-to-convergence of the entire simulation run. Also, at runtime, the user can correct initial mesh and resolution parameters increasing the accuracy of the entire simulation. Computation steering is therefore a remarkable new feature of the AGENT code providing a new way of monitoring live simulations toward best estimate optimization of reactor core modeling; in addition, the computation steering is well utilized as a tool for validation and verification of the code simulation. Our next development includes live monitoring of the graphical representations of the main solution parameters such as neutron flux and current distributions in 2D and 3D solution modalities.

7. REFERENCES

[1] Jovana Knežević, Ralf-Peter Mundani, and Ernst Rank, "Interactive Computing−Virtual Planning of Hip Joint Surgeries with Real-Time Structure Simulations," *International Journal of Modeling and Optimization*, Vol 1 (4), 2011, p 308.

[2] Mathieu Hursin, Shanjie Xiao, Tatjana Jevremovic, "Synergism of the method of characteristic, *R*-functions and diffusion solution for accurate representation of 3D neutron interactions in research reactors using the AGENT code system." *Annals of Nuclear Energy*, Vol. 33, 2006, p 1116.

[3] V. L. Rvachev, "Theory of R-functions and Some Applications," Naukova Dumka, 1982. (in Russian)

[4] Tatjana Jevremovic, Thomas Fogal, Dong-OK Choe, Haori Yang, Jens Krüger, "The Role of Virtual Engineering and EMERGING Visualization Tools in Nuclear Engineering Education and Training at the University of Utah", *NEST Conference, ENS*, Prague, April, 2011

[5] Mulder, J. D., van Wijk, J. J., van Liere, R: A Survey of Computational Steering Environments. *Future Generation Computer Systems*, Vol. 15(1), 1999, p 119.

[6] Knezevic, J., Frisch, J., Mundani, R.-P. and Rank, E. "Interactive computing framework for engineering applications." *J. Comput. Sci.*, Vol. 7 (5), 2011, p 591.

[7] Thomas Fogal, Jens Krüger, "*Tuvok*, an Architecture for Large Scale Volume Rendering." *Proceedings of the 15th International Workshop on Vision, Modeling, and Visualization*, 2010.

[8] A. Leonard, C.T. McDaniel, "Optimal polar angles and weights". *Trans. Am. Nucl. Soc*., Vol. 73,1995, p. 171

[9] M. Halsall, "*CACTUS*, A Characteristics Solution to the Neutron Transport Equation in Complicated Geometries, *AEEW-R1291*, UKAEA, Winfrith, 1980.