Available online at www.sciencedirect.com

ScienceDirect

Procedia Engineering 82 (2014) 351 – 363

Procedia Engineering

www.elsevier.com/locate/procedia

23rd International Meshing Roundtable (IMR23)

Generating Unstructured Nuclear Reactor Core Meshes in Parallel

Rajeev Jain*[a], Timothy J. Tautges[b]

[a]*Argonne National Laboratory, Argonne, IL 60439, U.S.A.*
[b]*CD-Adapco, Austin, TX 78747, U.S.A*

Abstract

Recent advances in supercomputers and parallel solver techniques have enabled users to run large simulations problems using millions of processors. Techniques for multiphysics nuclear reactor core simulations are under active development in several countries. Most of these techniques require large unstructured meshes that can be hard to generate in a standalone desktop computers because of high memory requirements, limited processing power, and other complexities. We have previously reported on a hierarchical lattice-based approach for generating reactor core meshes. Here, we describe efforts to exploit coarse-grained parallelism during reactor assembly and reactor core mesh generation processes. We highlight several reactor core examples including a very high temperature reactor, a full-core model of the Korean MONJU reactor, a ¼ pressurized water reactor core, the fast reactor Experimental Breeder Reactor-II core with a XX09 assembly, and an advanced breeder test reactor core. The times required to generate large mesh models, along with speedups obtained from running these problems in parallel, are reported. A graphical user interface to the tools described here has also been developed.

Peer-review under responsibility of organizing committee of the 23rd International Meshing Roundtable (IMR23)

Keywords: nuclear reactor core mesh, parallel mesh generation, MeshKit, RGG, AssyGen, CoreGen

1. Introduction

Mesh generation is a challenging field of science that has made rapid advancement over the years. New discretization techniques, improvements in computer hardware, and robust solver methods have aided to the research in this field. Fully detailed multiphysics reactor core analysis can be performed on hundreds of thousands of CPU cores, and such simulations require large meshes that are often hard or impossible to generate on a standalone desktop computer.

A nuclear reactor core consists of uranium fuel rods, instrumentation rods, control rods, ducts, coolant, grid spacers, load pads, restraint ring systems, and other supporting structure. Although these core models are quasi-2.5-dimensional, modeling of interassembly gaps forming the whole core model from the component assemblies can get

Peer-review under responsibility of organizing committee of the 23rd International Meshing Roundtable (IMR23)
doi:10.1016/j.proeng.2014.10.396

cumbersome and difficult to manage by using scripts or traditional meshing packages. Creating a fully detailed reactor core model with a few million elements can take a few days. Moreover, constructing these models on typical desktop computers can require large runtimes, and for large models such efforts may not succeed at all because of memory constraints. Analysts require a parallel modeling tool that can generate large core models in a few hours and can simplify the overall process of generating reactor core models.

This paper is an extension to our previous paper [1] that describes a lattice-based approach to simplify and automate the creation of reactor assembly and core meshes. The primary contribution of this paper is a parallel version of the assembly and core generation processes and also a graphical user interface for our tools. During the creation of models for various large multiphysics simulations, we show that our approach significantly reduces the time taken to create core models and in several cases demonstrates superlinear speedups. The remainder of this paper is organized as follows. Section 2 gives the background, which includes descriptions of the MeshKit and Reactor Geometry (and mesh) Generator (RGG) tools AssyGen and CoreGen [1] developed in our previous paper. Section 3 presents the parallel algorithm. Section 4 highlights the open source RGG nuclear application developed by Kitware Inc. Section 5 describes example core models created by using these tools, along with performance data. Section 6 discusses the models and our plans for future work. Section 7 briefly presents our conclusions.

2. Background

This section summarizes three major components that are essential to the algorithms presented in Section 3 of this paper. Section 2.1 describes the MeshKit mesh generation library, and Sections 2.2 and 2.3 respectively describe the two major tools AssyGen and CoreGen that are a part of MeshKit.

2.1. MeshKit

MeshKit is an open-source mesh generation library under development at Argonne National Laboratory, providing efficient algorithms for mesh copy/move/merge, extrude, and other algorithms [2]. It is implemented in C++ and provides a traditional C++-based API for interactions with other codes. A Python interface is also provided for interactive access to the library. MeshKit relies on geometry and mesh libraries developed as part of the Interoperable Tools for Advanced Petascale Simulations (ITAPS) project. The Common Geometry Module (CGM) [3] provides functions for constructing, modifying, and querying geometric models in solid model-based and other formats. While CGM can evaluate geometry from several underlying geometry engines, this work relies mostly on ACIS [4], with an Open Cascade–based [5] version also supported. Finite-element mesh and mesh-related data are stored in the Mesh-Oriented datABase (MOAB) [6]. MOAB provides query, construction, and modification of finite-element meshes, plus polygons and polyhedra. Various options are available for writing and visualizing the final meshes produced by meshing algorithms. MOAB uses an HDF5-based file format, which can be visualized by using a ParaView plugin that is implemented by the MOAB library. The Visit visualization tool can also be configured and built with MOAB to provide a similar import capability.

MeshKit's design philosophy is twofold: it provides a collection of meshing algorithms for use in real meshing problems, along with other tools commonly needed to support mesh generation (coordination of BREP-based meshing process, mesh smoothing, etc.); and it serves as a platform in which to perform mesh generation algorithm research. MeshKit also supports the connection of external meshing algorithms to the rest of the library, enabling the use of proprietary or experimental algorithms. MeshKit uses a graph-based process for specifying the overall meshing approach, with graph nodes representing meshing and other operations, and graph edges as dependencies between those operations. Provision is made for operations to construct other operation nodes automatically, for the purpose of meeting input constraints by specific algorithms; in this way, users need only focus on the graph nodes for which specific input is needed. Executing the meshing process consists of traversing the graph twice; the "setup" phase starts at the leaf node and proceeds in reverse up the graph, with graph nodes create any upstream nodes on which they depend; then the "execute" phase traverses from the root and proceeds in the forward direction, executing nodes in graph-topological traversal order.

The graph-based approach supports the traditional BREP-driven meshing process, which usually proceeds by meshing BREP entities in increasing topological dimension, starting with vertices, then edges, and so on. However,

a graph-based process also enables meshing tasks not possible with a strict BREP-based approach. First, not every meshing process needs or has a geometric model representing the entire domain to be meshed. The best example is the Reactor Geometry (and mesh) Generator (RGG) tool, in which individual assembly types have geometric models but are then copy/moved into a lattice of assembly models forming a reactor core (Sections 2.2 and 2.3 describe the AssyGen and CoreGen tools in RGG, respectively). Second, a meshing procedure may not involve only a once-through meshing of each BREP entity; again, RGG is a good example, in which the first part of the process involves meshing BREP models but the last step involves copy/moving mesh subsets into a larger core lattice. Third, the procedure-driven approach to meshing represented by most CAD-based meshing tools fails to capture the parallelism and dependency structure that can be found in most meshing problems (including BREP-based ones); representing and exploiting this richer structure provide more flexibility while still being applicable to BREP-based problems.

2.2. AssyGen

AssyGen is the first step of the three-step core mesh creation process implemented in RGG. AssyGen reads an input file describing a reactor assembly lattice and generates an ACIS or OCC-based geometry file, along with a template script for generating a mesh for the assembly using the CUBIT meshing toolkit [7, 8]. The second step is meshing, where the user may choose to perform meshing using the CUBIT mesh script generated by AssyGen or using meshing algorithms in MeshKit. For the Core Mesh shown in Fig. 1, AssyGen was run twice to create Assembly Mesh 1 and Assembly Mesh 2. Several features are supported by AssyGen, such as axial numbering material and boundary conditions, sectioning, rotation, and information about location of pins. These have been described in our previous paper [1].

2.3. CoreGen

The CoreGen tool reads an input file describing the reactor core arrangement and generates the reactor core mesh or geometry from its component assemblies. CoreGen uses the CopyMesh, ExtrudeMesh, CopyGeom, and MergeMesh algorithms in MeshKit. Figure 1 shows the two assembly meshes and an interstice mesh file that form a 19-assembly reactor core. A makefile is generated by CoreGen to automate this process. Details about the overall core meshing parameter specification, boundary creation, creation of a 2D-core followed by extrusion, and extrude/expand/copy sets for metadata propagation to the final model are detailed in our previous paper on RGG [1].

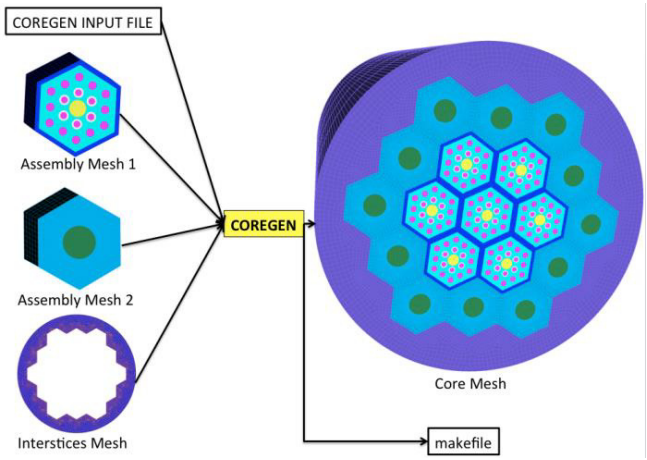

Fig. 1. A 19-assembly core model formed by using 3 mesh files and an input file describing the arrangement.

3. Parallelism Approach

Parallelizing this process not only allows faster turnaround times for large reactor core simulations but also improves the workflow, eliminating user error that sometimes occurs as a result of long generation times. Section 3.1 describes parallel AssyGen and meshing algorithms, while Section 3.2 details parallel CoreGen and its component algorithms.

3.1. Parallel AssyGen and Meshing Generation

This section describes two steps of the RGG process: geometry creation using AssyGen and meshing of the assembly created by AssyGen. A single reactor assembly may comprise several thousand geometric regions if one consider the materials and axially varying properties of the assembly. One possible approach for parallelizing AssyGen involves having each processor create a region based on the location of the pin it obtains from the master processor. The problem with this approach is that the subtraction of regions from the outermost duct is a serial process. Because of the unavailability of parallel subtraction of geometry from components, this feature hasn't been developed.

Instead, we are developing in MeshKit a parallel meshing algorithm that partitions geometry created by AssyGen and meshes these chunks in different processors to create the assembly mesh. Once all the assembly and core input files are created, the CoreGen program is invoked to generate a makefile (see Fig. 1) that automates the whole process from geometry creation using AssyGen input files to the final core mesh file creation. This makefile can be invoked in parallel by using the "-j" option of the "make" command to run the geometry creation and mesh generation of individual assemblies in different CPU cores. Thus, while the creation of individual assembly geometry and the meshing of individual assemblies are not parallel, the generation of all assemblies needed for a given core is parallelized, by using the basic parallel execution facilities of the standard make process. Since complicated reactor cores often use more then ten different assembly types, substantial speedup in this part of the RGG process can result.

3.2. Parallel CoreGen

Hundreds of assembly meshes along with interstices meshes such as grid spacers and restraint rings form a complete reactor core mesh. These models can be very large and involve to several billion mesh elements. Generation of such models in serial is a time-consuming process. The parallel CoreGen algorithm comprises seven steps:

1. On each processor: read CoreGen input file, parse, and determine assembly copies assigned to this processor based on a round-robin distribution.
2. Locally read assembly meshes for assemblies determined in step 1.
3. Perform assembly copy/move operations assigned to this processor.
4. Perform local merge of on-processor mesh.
5. Perform parallel merge of mesh between processors.
6. Parallelize the expand/extrude/copy set (metadata) handling.
7. Save output mesh.

Details of steps 1, 5, 6, and 7 are given in Section 3.2.1, 3.2.2, 3.2.3, and 3.2.4, respectively. A graphical depiction of the four-processor parallel CoreGen process is shown in Fig. 2. Each processor loads one assembly mesh file (Fig. 2A). The copy/move task distribution can be computed independently in all processors, since it is deterministic. After each processor finishes the copy/move task for its assigned assembly instances, the merging of nodes, handling of mesh metadata, and saving the output mesh happen in parallel. The final mesh model in Fig. 2B is the whole core mesh created after running CoreGen.

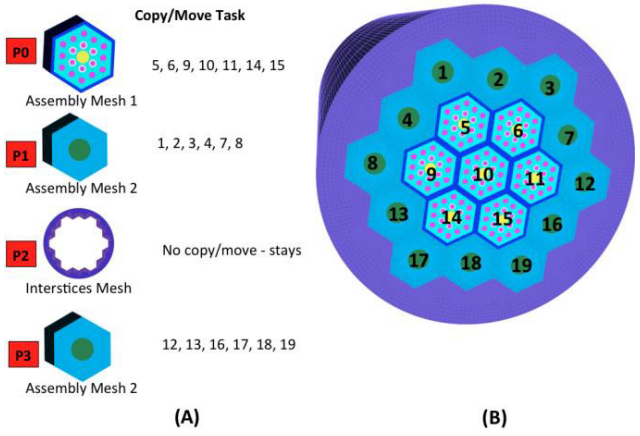

Fig. 2. Simple example demonstrating CoreGen input and output files. (A) Copy/move task distribution among processors and (B) final core-mesh with numbered assemblies.

3.2.1 Copy/Move Task Distribution

During a parallel CoreGen run there is no master processor, each processor analyzes the overall workload and decides the mesh file(s) that it must load. During this task distribution phase three different cases (A, B, and C) may occur, where nA is the total number of different assembly mesh files, np is the number of processors, and nT is the total number of assemblies in the reactor core.

A. $np < nA$
B. $nA < np < nT$
C. $np > nT$

In case A, each processor loads one or more assembly mesh files and solely performs the copy/move operation associated with that assembly for the entire core. For case B, some mesh files are loaded into multiple processors and copy/move task is distributed among them. Two schemes have been developed for selection of mesh files for copy/move task distribution: Scheme 1 is based on the frequency or the number of occurrence of that mesh file in the core, the assembly type that appears the most number of times in the core is assigned to multiple processors in a round robin fashion. Scheme 2 is based on recursive analysis of load on each processor after each mesh file load has been decided. Scheme 2 leads to better load balancing of meshes among processors and faster overall completion time when compared to scheme 1. Another schemes based on the size of the mesh file and a combination of scheme 1 and 2 is under development. For case C, there are more processors than available work units, so only nT processors can take part in this parallel algorithm.

Results for MONJU reactor detailed in Section 5.2 uses scheme 2 highlighted above for copy/move task distribution, round-robin distribution scheme leads to highly unbalanced load among processors causing the problem to not fit in the available CPU memory.

3.2.2 Parallel Merge Mesh

Executing RGG in serial requires the entire model to be represented on a single processor, which limits the overall size of the assembled mesh to what can be held in memory on a given machine. The merge mesh algorithm is based on matching mesh vertices based on geometric proximity. This algorithm is created by slightly modifying

the vertex-matching algorithm described in a previous paper [9]. First, instead of merging based on the global id and partitioning the global id space over processors, the geometric bounding box of all vertices is partitioned over processors, with each processor responsible for a distinct geometric region (plus a small epsilon layer whose thickness is twice the distance tolerance of the merge). The spatial extent covered by each processor can be computed deterministically based on the global bounding box, the number of processors, and the merge tolerance. Each processor retrieves vertices on the skin of the local mesh and performs a local merge on those vertices. Next, for the remaining skin vertices, each processor assembles a tuple list [9] that holds the (x, y, z) position of the vertex, and the destination processor is assigned according to the processor(s) responsible for that position in space. Note that in contrast to the serial merge, merging a vertex with another on another processor does not result in one of those vertices being deleted; rather, the parallel sharing information inherent in MOAB's parallel mesh representation is modified to indicate that they are the same logical vertex. The actual merging is done as part of the parallel write process described in Section 3.2.4.

3.2.3 Parallel Metadata Handling

In a MOAB representation of mesh, entity sets (arbitrary groupings of entities and other sets) are used to represent material and boundary condition groupings, with tags used to indicate the purpose of a given set (e.g., MATERIAL_SET, NEUMANN_SET) and tag values identifying distinct sets of that type. Each processor makes a copy of the sets that will receive new entities on a given processor resulting from the copy/move operations performed there. In the problem specification, sets are marked as being of three distinct types for the purposes of RGG:

- EXPAND: New entities copied from an original entity will be added directly to any expand sets containing the original entity.
- COPY: Any copy set containing the original entities will itself be copied and will receive the copies of entities according to original entity membership in the original copy set(s).
- EXTRUDE: When MeshKit is being used for the 2D to 3D extrusion, extrude sets containing entities of dimension *d* will be given the *(d+1)*-dimensional entities resulting from the extrusion of the original extrude set entities.

Expand sets are used to represent material in the core mesh. Copy sets can be used to group entities making up assembly instances (i.e., to distinguish the *n* copies of a given assembly type in the core). Expand sets are used for modeling Neumann boundary conditions in extruded or swept models.

3.2.4 Parallel Save Mesh

The parallel save option in MOAB writes a single file containing the entire mesh, with single copies of entities and entity sets that were formerly spread or shared between multiple processors. Parallel save algorithm can be divided into four steps:

1. Compute global mesh properties (number of entities/sets of each type, global numbering of file entities etc.).
2. Write header information (root processor only).
3. Write vertex coordinates, element connectivity for owned entities (concurrent).
4. Write vertex/element tags for owned entities (concurrent).
5. Write set membership (serial).

HDF5-based files resulting from this write process can be repartitioned using a MOAB-based partitioning tool and used as input to simulations able to read MOAB files.

4. RGG Nuclear GUI Application

Kitware Inc. has developed a GUI for creating and visualizing new and existing AssyGen and CoreGen input files. This tool is freely available and can be downloaded from the Kitware website [10]. Users can create a full-core geometry or mesh models from scratch using this GUI. RGG examples available with MeshKit library can also be visualized and run to create core meshes using this application. Figure 3 shows an example of this tool displaying the 19 assemblies from the model in Fig. 1. This GUI is also capable of displaying the final MOAB mesh files output by CoreGen.

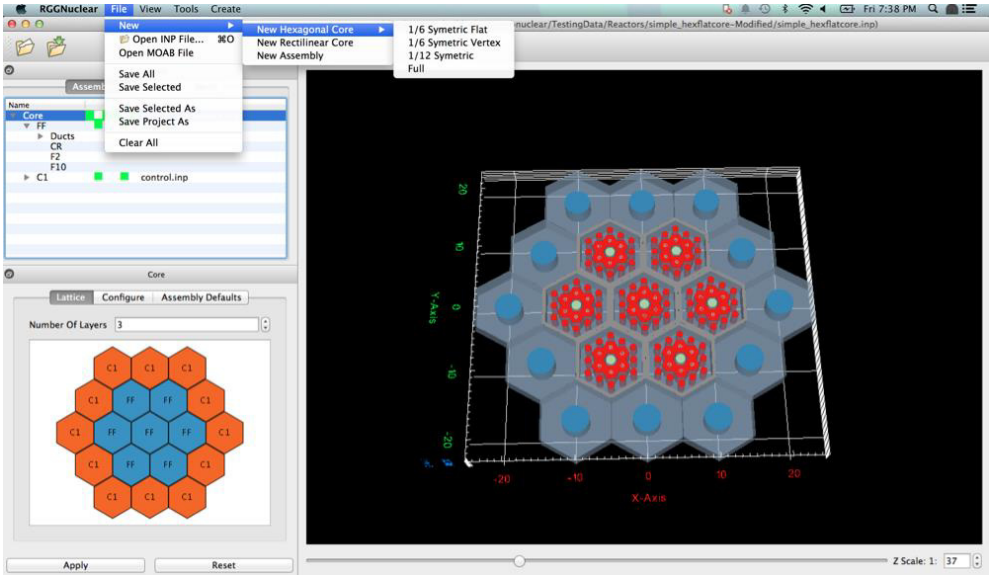

Fig. 3. RGG nuclear application launched on Mac OSX showing loading of the simple hex-flat-core example from the RGG test directory in MeshKit.

5. Results

5.1. Very High Temperature Reactor

The 1/6 very high temperature reactor (VHTR) core model shown in Fig. 4 consists of 19.6M hexahedral elements and 20.5M mesh vertices. The size of this mesh file on disk is 2.2 GB.

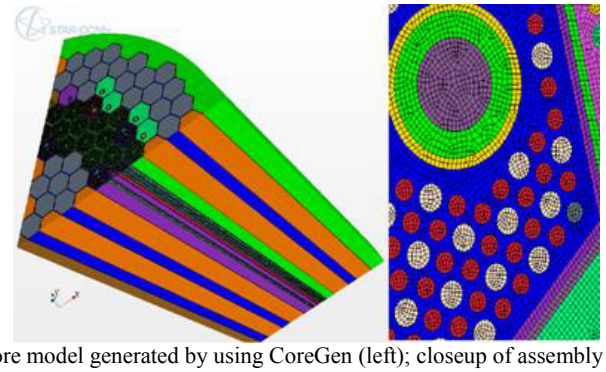

Fig. 4. One-sixth of a VHTR core model generated by using CoreGen (left); closeup of assembly mesh in this model (right).

AssyGen took 10.5 minutes to generate the geometry, journal files, and run CUBIT in order to mesh individual assemblies. The serial CoreGen program took 48 minutes to generate the core model. When using 56 processors for running CoreGen, this model can be generated in less than 3 minutes (2.1 minutes with 12 processors; 12 assemblies form this 58-assembly model) from scratch. The AssyGen+CUBIT are run on 1, 4, 8, and 12 processors by using the –j option of the makefile; the maximum number of processors is limited to 12 for this step.

Table 1 lists the AssyGen+CUBIT, copy/move, merge, parallel save, total time, and the maximum memory used for various steps of the CoreGen stage, when using different number of processors. Figure 5 shows these timing results and maximum memory used vs. number of processors. Superlinear speedups are observed in almost all cases, due to the job fitting in available memory. Mesh joining (or merge) is observed to be actually slow as the number of processors increases from one to four; this result is probably due to the communication overhead required in the parallel algorithm. At larger numbers of processors, however, the joining time is reduced far below the serial time. As expected, the total time, time taken to save, and maximum memory used by a processor decreases as the number of processors increases.

Table 1. CPU time in minutes and maximum memory in gigabytes used for 1/6 VHTR core

# Procs	AssyGen+CUBIT	Copy/Move	Merge	Save	Total	Memory
1	10.5	17.6	10.4	0.7	48.2	4.9
4	3.4	11.0	11.7	0.01	26.2	2.6
8	2.8	11.1	5.6	0.01	19.5	2.5
16	2.1	0.4	4.7	0.01	7.3	1.7
32	2.1	0.03	0.56	0.01	2.7	0.48
56	2.1	0.0005	0.31	0.005	2.43	0.33

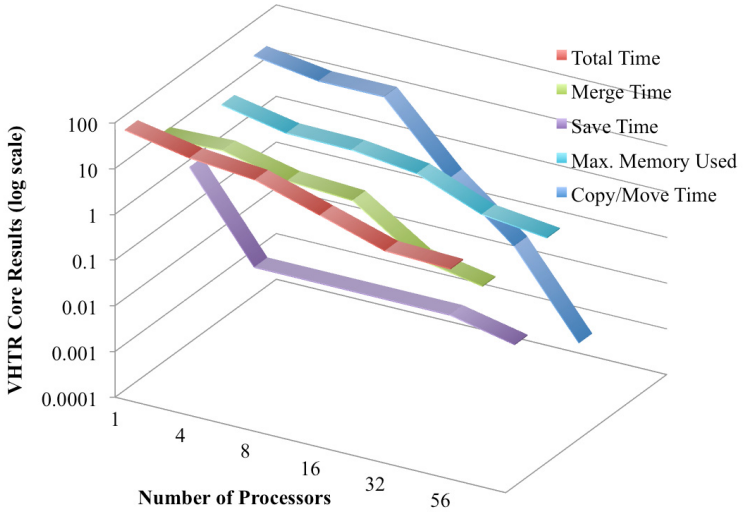

Fig. 5. Log scale results from Table 1 vs. number of processors for 1/6 VHTR reactor core.

5.2. Full-Core MONJU Reactor

Figure 6 shows a full-core MONJU reactor, which comprises 8 assembly types and consists of 715 assemblies in total. AssyGen and meshing take 5.5 minutes (8 processors using make –j8 option) to mesh the 8 assemblies. CoreGen on 712 processors takes only 90 seconds. The total wall-clock time required to generate this 101M hexahedral element model from scratch is 8 minutes. The maximum memory used by a processor is only 196 MB. This model cannot be run in serial because the problem does not fit in memory. The size of the mesh file on disk is 14 GB.

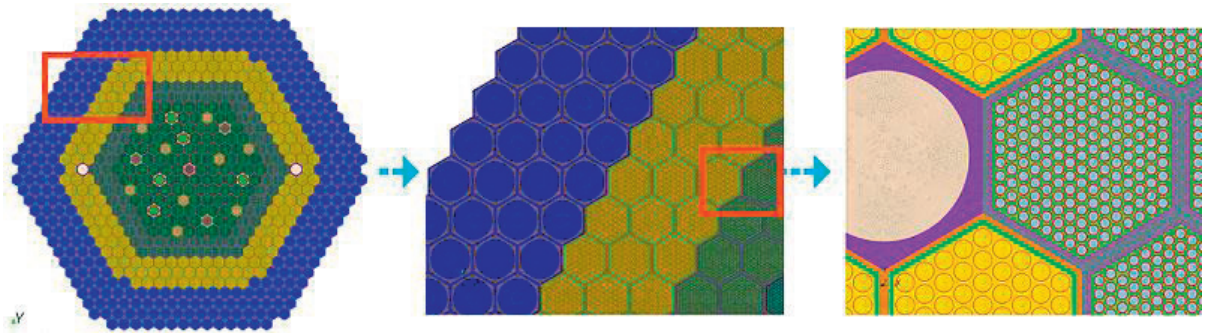

Fig. 6. Full-core MONU reactor; closeup area in red rectangular region is highlighted from left to right.

Table 2. CPU time in mins and maximum memory in GB used for MONJU core

#Procs	AssyGen+CUBIT	Copy/Move	Merge	Save	Total	Memory
16	5.5	35	163	3.8	207.64	4.2
64	5.5	0.03	61	9.01	76.7	1.04
128	5.5	0.03	293	15.5	315	0.532
256	5.5	0.03	291	24	321.5	0.244
320	5.5	0.002	0.45	16.5	22.8	0.233
512	5.5	0.002	0.49	3.85	9.99	0.233
712	5.5	0.002	0.18	2.3	7.89	0.196

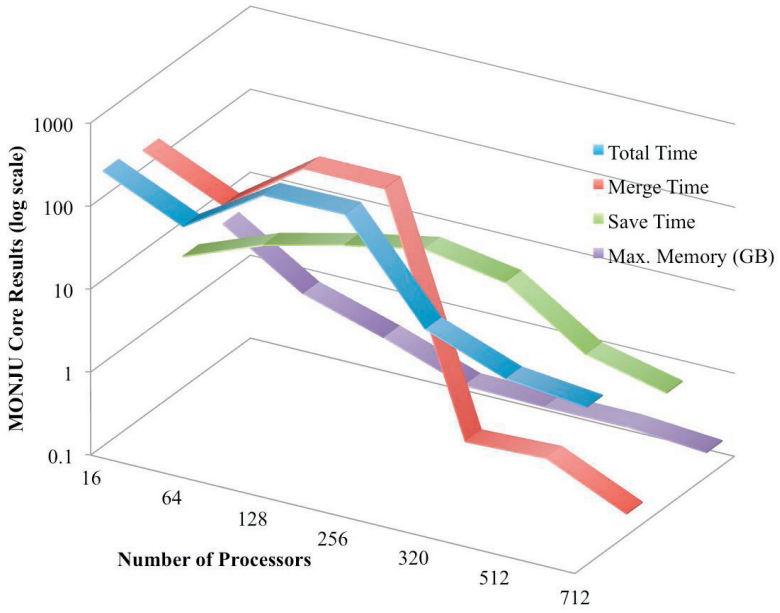

Fig. 7. Log scale results from Table 2 vs number of processors plot for MONJU reactor core.

Using a round-robin scheme to establish the copy/move work for this reactor causes the load on some processors to be larger than the available memory. Therefore, we use a recursive load-balanced scheme, wherein after deciding every assembly load, the resulting assembly copy/move work due to this load is analyzed and then the decision on the next assembly load is made. Table 2 lists the AssyGen+CUBIT, copy/move, merge, parallel save, total time, and maximum memory used with different numbers of processors.

Figure 7 shows the graph of log scale results from Table 2 vs the number of processors used. One can see the sudden jump in merge/save and total time for 128 and 256 processors. It has been found that a few processors performing local merge spend a majority of the time. Local merge creates an adaptive kd-tree and then finds and merges the nodes on the skin for all elements local to a particular processor. This jump in time for merging is possibly due to large number of searches for determining the nodes to be merged locally. We plan to fix this in future versions of the tool by using a different tree and/or fixing the existing kd-tree implementation.

Compared to serial the total time required to generate this model drops to 22mins for 320 processors and for 512 and 712 processors this time further drops to 10 mins and 8mins respectively. The overall maximum memory required by a processor decreases as the number of processors increase.

5.3. 1/4 Pressurized Water Reactor

Figure 8 shows the benchmark problem "MOX Fuel Loaded Small PWR Core"; a detailed description can be found on the website of the Nuclear Reactor Analysis and Particle Transport Lab [11]. Individual assembly geometries are created by using AssyGen; then CoreGen is used to copy/move the assemblies and form the core geometry. Figures 8A, 8B, and 8C are closeup areas highlighted in red on the core model. The model consists of approximately 11k volumes, 5.2M hexes, and 5.9M vertices. On a Linux desktop, the assembly geometry creation takes 8 minutes, and CoreGen takes 12 minutes of wall-clock time and uses 0.9 GB of RAM for creating the core geometry model from component assembly geometries (no mesh generation). The maximum number of processors that can be used by CoreGen for this problem is 25, which is the total number of assemblies in this core model. Using 25 processors for creating the core mesh model, CoreGen takes 45 seconds and 210 MB maximum memory to create the mesh; the majority of the time is used in merging the nodes between the assemblies. The serial version of CoreGen takes 1.9 minutes and 1.7 GB of RAM. The copy/move takes 42 seconds, and the serial merge takes 50 seconds.

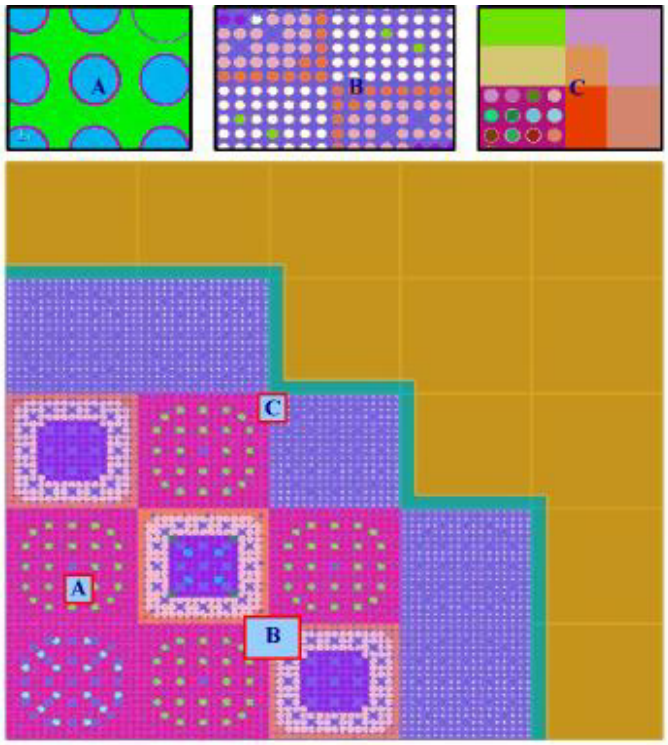

Fig. 8. Benchmark geometry of ¼ pressurized water reactor with closeup views A, B and C showing details of the model.

5.4. XX09 Assembly and Homegenized EBRII Core with XX09 Assembly

The instrumented XX09 assembly was used in shutdown heat removal tests (SHRTs) that demonstrated the passive safety features of the EBR-II Experimental Breeder Reactor [12]. A homogenized EBRII core mesh with a detailed XX09 assembly is shown in Fig. 9. The sectioned view of the core in the figure highlights the start and the end of the fuel rods and the conical/cylindrical coolant flow regions in the XX09 assembly. This core model is formed of 217 assemblies: 216 homogenized assemblies and 1 XX09 assembly. The model comprises 375k hex8 elements for the neutronics mesh, 440k hex8 elements for the structural mechanics mesh, and 440k hex27 elements for the thermohydraulics mesh. The geometry for the XX09 and other homogenized assemblies is generated by using AssyGen; the meshing is performed by CUBIT; and CoreGen creates the resulting EBRII core mesh. For the six assemblies that surround the XX09 assembly, meshing is performed with AssyGen by specifying the same interval on the edge shared with the XX09 assembly (see Fig. 9 marked "Outlet"). A coarse-edge interval is assigned on all other edges. On 128 processors CoreGen takes 45 seconds to create this core model from component assemblies; the maximum memory used by a processor is only 300 MB. Serially, it takes 16.8 minutes and 520 MB of memory. In the serial version, 16 minutes out of the total 16.8 minutes are taken by the mesh merge operation.

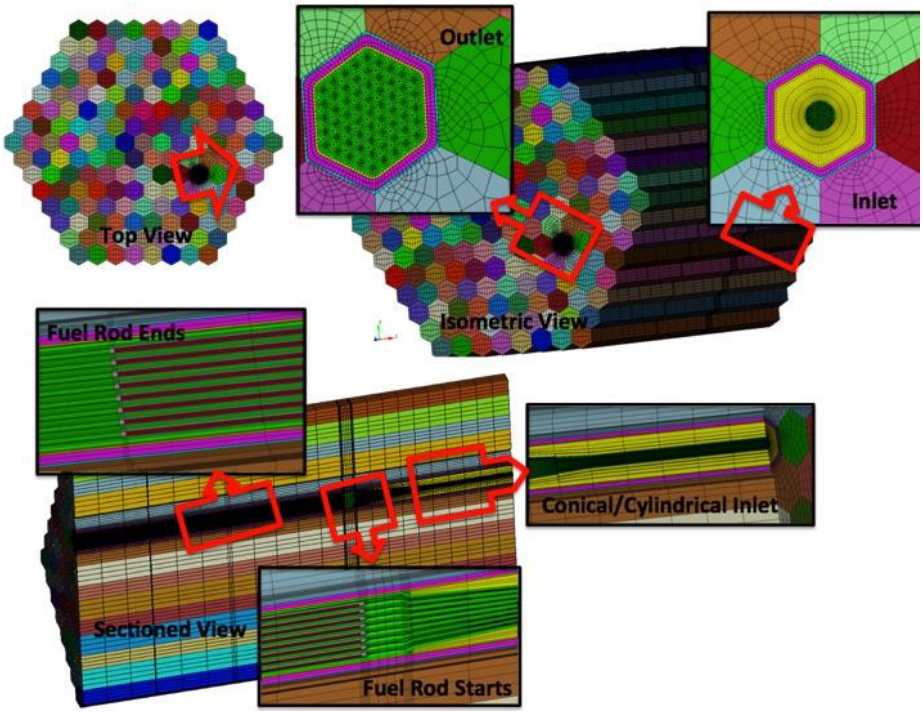

Fig. 9. Homogenized EBRII core with XX09 assembly. Inlet, outlet, isometric, and sectioned views are zoomed in to show the fuel and other instrumentation pins in the model.

5.5. ABTR Fuel Assembly and ABTR Full-Core Model with Restraint Rings and Load Pads

The geometry-only CoreGen feature was utilized to create the outer covering with restraint rings. All the assembly geometries created by AssyGen are first copy/moved to create a geometric core model. Another separate circular pin (ring) with the same axial divisions as all the other assemblies is created by using AssyGen. This pin is subtracted from the geometric core model to create the outer core model and obtain the outer covering geometry. The meshing, material, boundary specification, and clearance gap between the TLP (Top Load Pad) and ACLP (Above Core Load Pad) are modeled after this step. The clearance gap between TLP/ACLP restraint ring and load

pad is modeled by shrinking the inner surfaces by clearance gap dimensions. Once the outer covering mesh is generated, it is used as an 'interstices mesh' (see Fig. 1) in the CoreGen process. Fig. 10 shows the detailed configuration with I-J numbering and the number of occurrences of each of the assembly types. Since all assemblies have varying properties in the axial direction, a common axial configuration must be determined in order to have coincident nodes along the height of all eight assemblies that form this 199-assembly core model. This configuration leads to a conformal mesh that is fit for simulations. It must be noted that the actual model has fuel and other rods that are a part of each of the 8 assemblies listed in Fig. 10, they have not been modeled for this particular simulation.

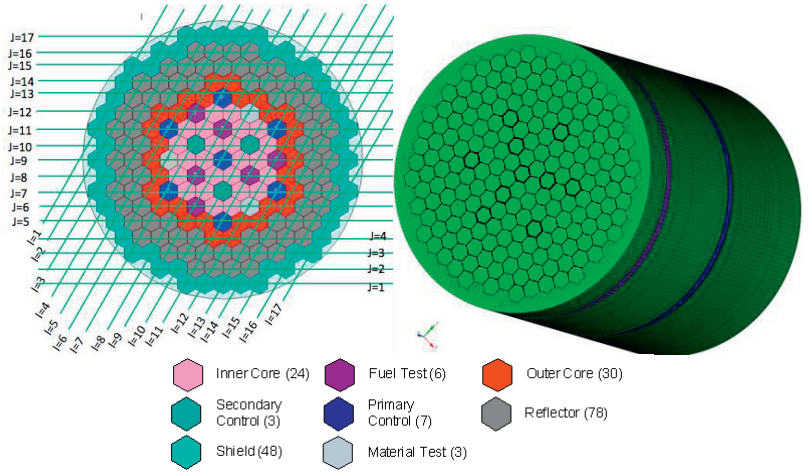

Fig. 10. ABTR core configuration with lines of constant logical I, J assembly regions.

CoreGen is run to create the final homogenized core model from assembly meshes and this outer covering mesh. Two models were created for this ABTR core model:

1. Coarse model: The final mesh shown in Fig. 10 consists of 800k hex elements. The structural mechanics and thermohydraulics mesh consists of 1,500 material blocks. The neutronics mesh consists of 7,200 material blocks; it requires more material blocks along the height of the fuel pins. CoreGen takes 5 seconds to assemble the core using 128 processors. The maximum memory used by a processor is 110 MB. The serial runtime for CoreGen is 45 seconds with 280 MB of memory used.
2. Fine model: Each assembly in this model consists of 5-10M hexahedral elements. The final mesh size of the core model created is 1.2B hexahedral elements, the size of this file on disk is 137GB. CoreGen on 200 processors (each processor loads one mesh file: 199 assembly + 1 interstices mesh file) takes 30 mins to create this file of which parallel save takes up 27 mins and parallel merge takes 1.7 mins. The maximum memory used by a processor is 1.96GB.

6. Discussion and Future Work

Simulation of complex systems such as nuclear reactors requires detailed models that properly capture the geometric shape and have correct specification of material and boundary conditions. Different physics such as neutron transport, fluid flow, thermal expansion and heat transfer must be studied to fully understand the performance and safety aspects of nuclear reactors. The parallel RGG tools enable the creation of such large and complicated reactor models for different physics simulations.

The size of finer ABTR mesh file on disk is 137GB and it consists of more than a 1 billion hexahedral elements, such meshes are impossible to construct using serial meshing processes on a standalone workstation. All the models shown in this paper have been used in coupled multiphysics simulations. The size of EBRII and PWR core models

have been intentionally kept coarse, internal details such as fuel and other instrumentation pins are ignored in ABTR and EBRII core for better understanding of the overall coupled simulation process. We plan to create fully detailed models and coupling results for these models. RGG is capable of creating meshes larger that those presented in this paper.

Work is in progress for developing geometry partitioning-based meshing algorithms to mesh the assemblies in parallel. Also, new schemes are being formulated for a better load-balancing during copy/move task distribution, this scheme will be a combination of existing schemes with more weightage given to meshes with larger element count. Parallel AssyGen development would be creation of individual pins/components in parallel, but we lose the speedup with serial subtraction of pins, this development is subject to development of parallel subtraction of geometries.

7. Conclusion

A powerful and robust parallel algorithm for generating very large reactor core meshes from their component assemblies has been developed. The algorithms are a part of an open source library called MeshKit, which provides meshing algorithms and infrastructure for meshing research and mesh generation tasks.

Acknowledgments

We thank the SIGMA (**S**calable **I**nterfaces for **G**eometry and **M**esh based **A**pplications) group at Argonne, who maintains the libraries required by this tool. The material based upon this work was supported in part by the U.S. Department of Energy Office of Nuclear Energy Nuclear Energy Advanced Modeling and Simulation (NEAMS) Program; by the U.S. Department of Energy Scientific Computing Research, Office of Science; and by the U.S. Department of Energy, Office of Science, Scientific Discovery through Advanced Computing program, under Contract DE-AC02-06CH11357.

References

[1] Tautges, T. J., and Jain, Rajeev. (2012). Creating geometry and mesh models for nuclear reactor core geometries using a lattice hierarchy-based approach. *Engineering with Computers*, *28*(4), 319–329.

[2] MeshKit: http://sigma.mcs.anl.gov/meshkit-library/

[3] Tautges, T. J. (2005) CGM: a geometry interface for mesh generation, analysis and other applications. *Engineering with Computers, 17*:486–490.

[4] Spatial website (2010) http://www.spatial.com/

[5] Open CASCADE technology website (2000–2010) http://www.opencascade.org.

[6] Tautges, T. J., Meyers, R., Merkley, K., Stimpson, C., and Ernst, C. (2004). MOAB: A mesh-oriented database, SAND2004-1592. Sandia National Laboratories, Albuquerque, NM.

[7] CMBNuclear download page: http://cmb.kitware.com/CMB/resources/software.html

[8] Detomi, D. (2002). A procedure for tetrahedral boundary layer mesh generation. *Engineering with Computers*, *18*(1), 66–79.

[9] Tautges, T. J., Kraftcheck, J. A., Bertram, N., Sachdeva, V., & Magerlein, J. (2012, May). Mesh interface resolution and ghost exchange in a parallel mesh representation. In *Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2012 IEEE 26th International* (pp. 1670-1679). IEEE.

[10] Sjaardema, G.D, Tautges, T. J, Wilson, T. J, Owen, S. J, Blacker, T. D, Bohnhoff, W. J, Edwards, T. L, Hipp, J. R, Lober, R. R, and Mitchell, S.A (1994). CUBIT mesh generation environment, users manual, vol 1. Sandia National Laboratories, Albuquerque, NM.

[11] 1/4 PWR Benchmark Problem (2012), http://nurapt.kaist.ac.kr/benchmark/

[12] EBR-II, http://en.wikipedia.org/wiki/Experimental_Breeder_Reactor_II