

## RGG: Reactor Geometry (&mesh) Generator

Rajeev Jain and Tim Tautges  
Argonne National Laboratory  
9700 S Cass Avenue, Argonne, IL 60439  
Tel: 630-252-3176, Fax: 630-252-5986, Email: jain@mcs.anl.gov, tautges@mcs.anl.gov

**Abstract** – RGG takes advantage of information about repeated structures in both assembly and core lattices to simplify the creation of geometry and mesh, it is released as an open source software as a part of MeshKit mesh generation library. The methodology operates in three stages. First, assembly geometry models of various types are generated by a tool called AssyGen, next, the assembly model or models are meshed using MeshKit tools or the CUBIT mesh generation toolkit, optionally based on a journal file output by AssyGen. After one or more assembly model meshes have been constructed, they are arranged in a core model using a tool called CoreGen, it uses a copy/move/merge process to create the core model. In this paper, we present the current state of tools, new features and parallel-enabled CoreGen. CoreGen is ideally suited for parallelism, during creation of large reactor core models it was realized as a bottleneck in the process. For several problems speedups for CoreGen are super-linear, due to the problem fitting in available RAM at higher processor counts. Several RGG applications viz. VHTR models, a 1/4 PWR reactor core, and a Full Core model for MONJU are reported.

### I. INTRODUCTION

Creation of geometry and mesh are two very important steps in the simulation of reactor cores. Nuclear reactor cores are typically formed by arranging pins in a lattice of surrounding material. It is possible to describe a reactor as two-level hierarchy of lattices [1]. The first level of hierarchy corresponds to fuel or other assemblies consisting of cylindrical pins, while in the second level assemblies are arranged in a lattice to form the reactor core. Although the structure inherent in this two-level hierarchy could be used to automate parts of this generation process, experience shows that user interaction is often required. We describe a system for generating reactor core geometry and mesh models that balances lattice-guided automation and user interaction at key points in the process. This system can be formulated in a three-stage process. In the first stage, assembly geometry and meshing scripts are created, second stage creates the mesh for this assembly geometry, and finally the third stage creates the core model using the output from the first two stages. RGG contains two tools: AssyGen and CoreGen for modeling several types of nuclear reactor assembly and core models.

Literature overview and various other domain-specific tools for geometry and mesh generation are discussed in our previous paper [1]. Also, during the development of the tools and discussions with neutronics and thermo-

hydraulics groups new features like support for tetrahedral meshing, creation of axially varying assemblies, support for 2D core creation, extrusion and others were added to the toolset, these are reported in a journal publication earlier this year [2]. The toolset continues to grow robust with more features and ability to create large models with automated model creation process. In this paper, we focus on parallel version of the tool, new models and keywords for aiding post processing.

Various sections in this paper are organized as follows: Section II describes the current status of tools and provides a brief overview, Section III describes parallel version of the tool and new keywords to the input file language. Section IV describes core models created using these tools, along with performance data. Section V discusses our conclusions.

### II. ASSYGEN AND COREGEN

AssyGen and CoreGen are tools developed as a part of MeshKit [3] library developed and maintained at Argonne National Laboratory. These tools rely on geometry and mesh libraries developed as a part of Interoperable Tools for Advanced Petascale Simulations (ITAPS) project. The Common Geometry Module (CGM) [4] provides function for constructing, modifying and querying geometric models in solid model-based and other formats. Finite element mesh and mesh related data are stored in the

Mesh-Oriented database (MOAB) [5]. Along with functions to query, construct and modify, MOAB also provides efficient functions to handle mesh in parallel. Mesh generation is performed by using a combination of tools. The CUBIT mesh generation toolkit [6] provides algorithms for both tetrahedral and hexahedral mesh generation. MeshKit provides efficient algorithms for mesh copy/move/merge, extrude, and other algorithms. The AssyGen tool generates the assemblies and the CoreGen tool copy/move/merge(s) those assemblies to form a core.

AssyGen tool is capable of generating rectangular or hexagonal assemblies. Fig. 1. shows two such hexagonal assembly geometries being created: Assembly Geometry 1 and 2. It highlights the first two stages of the process. Input to AssyGen tool is a keyword-based text input file. Output is assembly geometry and mesh script.

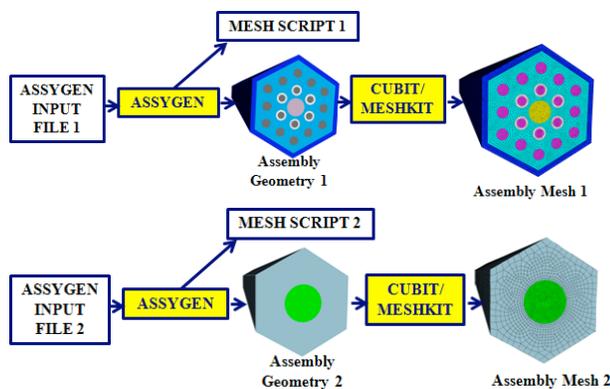


Fig. 1. First two stages of the geometry/mesh process, where AssyGen and CUBIT are executed for each assembly type.

Input file is based on a predefined set of keywords that are followed by values that describe the model. A complete list of keywords, options and values are described in the README file [7]. Geometry file created by AssyGen can be saved in formats supported by the geometry engine which was used to build CGM. ACIS [8] and OpenCascade [9] are currently supported by CGM. In the input file, we describe an assembly as a lattice of unit cells. Each unit cell has zero or more concentric cylindrical layers of material, cut from a background material defined for each unit cell or for the assembly as a whole. Unit cell shapes can also be imprinted on the background material, to more finely control the mesh in each unit cell. The assembly can be surrounded by one or more layers of duct wall material. Multiple pincell types can be defined, each with one or more concentric cylinders of material and a background material for the cell. Cylinders input for individual pin cells can be larger than the unit cell dimensions; these volumes will overlap neighboring pincell regions when present. In this case, a special keyword can be used to restrict these larger structures to

the unit cell, such that they do not overlap neighboring regions. Empty pin cells can be specified in the assembly lattice by specifying a predefined “XX” or NULL unit cell type, indicating that only background material (and structures from neighboring unit cells) overlaps the cell. Parameters can be specified multiple times with varying Z-dimensions and material properties to create assembly models with axially varying properties.

In the second stage of the process assembly geometry and the CUBIT mesh script, which is automatically generated by AssyGen are run to generate an assembly mesh. This assembly mesh has the materials and boundary conditions defined. Top, bottom and side surfaces of all materials are marked as boundaries, material names are suffixed with “\_top”, “\_bot” and “\_side” to name the boundary conditions respectively. Material names are defined in the text-based input file. AssyGen is capable of creating both surface and volume type geometry files and their corresponding mesh script files. It must be noted that for a given core configuration first two stages are repeated for each assembly that forms the core (see Fig. 2. for the core formed by assemblies created in Fig. 1). It is incumbent on the user to make sure that the meshes match between assemblies. The tools don’t explicitly enforce a constraint to match the nodes along the sides of the assemblies that sit next to each other. There are keywords in AssyGen to assign the intervals along the edges and in the z-direction. The same interval or bias factor must be used on all the assemblies to guarantee that neighboring assemblies are glued perfectly and the resulting core mesh is conformal. In case of a tetrahedral mesh, it is a bit difficult to enforce this constraint. Both translational and rotational symmetry of meshes on the side surface of all the assemblies are desired. This was achieved by meshing the sides first and then meshing the entire volume. Side surface are split and have the same mesh interval on the sides. One half of the split is meshed and then flipped onto the other half surface [2].

Meshing process is very brittle and often very sensitive to the input mesh size. This problem occurs due to several reasons, top surface is cut by a large number of cylindrical rods and is often hard to mesh using unstructured quadrilateral elements, also the ratio of largest to smallest dimension on the top surface of the model is very large, which entails having very small element size to mesh the top surface. Tight element budget and specific needs of the simulation scientists make it clear that finer control and freedom is required at during this stage of the reactor core generation process. At present we use CUBIT for the mesh generation process and specific keywords like ‘EdgeInterval’, ‘RadialMeshSize’ and ‘AxialMeshSize’ are available as variables in the script to better control the meshing operation. CUBIT is closed-source, efforts are being made to develop algorithms like Jaal [10] etc. in the MeshKit library to tackle such the mesh generation problems.

CoreGen tool reads all the assembly mesh files and a keyword-based text input file similar to AssyGen input file. This file describes the arrangement of assemblies in the core lattice and locations of models for each assembly type, along with those meshes; this information is used to generate the overall core mesh. Fig. 2. shows two hexagonal assembly mesh files, an interstices mesh and a CoreGen input file. These files are read by the CoreGen program to create a makefile and a full hexagonal core mesh file with 19 assemblies as per the specification in the CoreGen input file. Unlike the example in Fig. 2., which creates a core mesh, CoreGen can read in geometry files to create the resulting core geometry, in which case the overall process becomes a two stage process with no meshing. The interstices mesh is not copy/moved like the other assembly mesh files; it is a way to provide fixed pieces in the model. The makefile generated by CoreGen, automates the whole process, from various assembly to the creation of the core. Fig. 3. in Section IIIA demonstrates the generation of the same core mesh using parallel-enabled CoreGen.

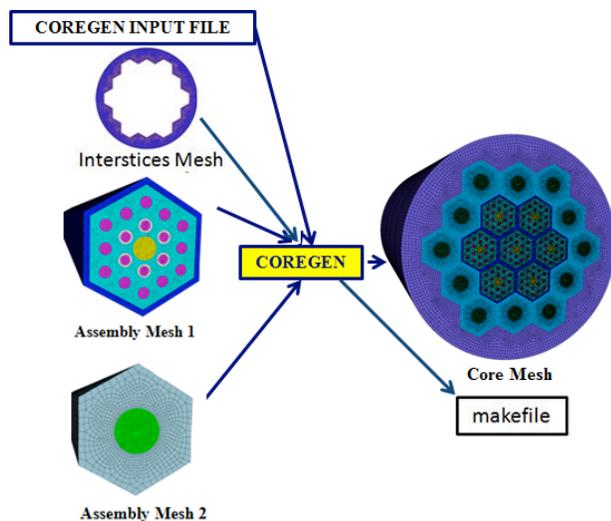


Fig. 2. Third stage of the geometry/mesh process, where CoreGen is executed.

CoreGen supports creation of  $1/6^{\text{th}}$ ,  $1/12^{\text{th}}$  and full core models for a hexagonal type core and full for a rectangular type core. Details on the specifications and conventions used for core definition can be found in [1]. Example IVA creates a  $1/6^{\text{th}}$  hexagonal core. Example IVC creates a full rectangular core. IVA also highlights another variation to this three stage process. A 2D core is formed in the first three stages, in the third stage, as per the user specification, the 2D core model is extruded to desired height and number of subdivisions to create a 3D core mesh. This 2D core generation plus extrusion process is faster than its 3D counterpart.

It must be noted that metadata or material and boundary conditions must propagate from individual assemblies to the core. This is achieved by defining abstractions specifying the handling of specific types of groupings according to those abstractions [2]. The groupings are of three different types: copy, expands and extrude, when an entity is copied and new entities are created it is assigned to a copy grouping, when the grouping needs to accommodate and expand, it is assigned to a expand grouping, for extrude grouping the entities of a group are replaced by the newly created entities of the corresponding higher dimension that were extruded.

Often sides or faces of overall core model are desired as boundary conditions. We use the 'NeumannSet' keyword in the input file for top, bottom and side faces; for side faces, equation of line in radial direction must be specified along with the keyword. This radial line sets the particular face or side of the core for which boundary condition is desired. Internally these boundary conditions on the core are assigned by obtaining the skin of the entire core and filtering the faces into relevant/desired groups specified in the input file.

### III. NEW SALIENT FEATURES

For several models the mesh size didn't fit in the available memory and we realized a need for parallel-enabled CoreGen, which is described in section IIIA. In section IIIB, we highlight the 'Info' keyword which helps in keeping track of pin and assembly number in both assembly and core mesh files.

#### IIIA. Parallel-Enabled CoreGen

During creation of core models with large number of assemblies formed with only a few types of specific assemblies, it was realized that large memory requirement was a bottleneck in the process. Recent development in the parallel capabilities for handling and manipulating meshes in MOAB created a perfect environment for parallel-enabled CoreGen. CoreGen itself is ideally suited for parallelism. The basic algorithm used for parallelizing can be summarized in five steps given below:

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, on each processor read assembly meshes for assemblies determined in step 1.
3. Perform assembly copy/move operations assigned to this processor.
4. Perform parallel merge.
5. Save output mesh.

For steps 4 and 5, CoreGen leverages the parallel merge and save algorithms developed in MOAB [5]. New algorithms were developed for shared vertex and metadata resolution among the processors [11].

The copy/move task distribution is deterministic; it is done on each processor based on the text-based CoreGen input file. In step 1, three different cases arise when distributing the assembly meshes among processors:

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

In case A, each processors loads more than one assembly mesh file and solely performs the copy/move operation associated with that assembly for the entire core. For case B, some mesh files are loaded in multiple processors, this mesh file selection is based on the frequency or the number of occurrence of that mesh file in the core. The file that appears most number of times in the core is assigned to multiple processors. This operation is deterministic and performed by all processors. For case C, some processors remain idle, copy/move task is divided per assembly, therefore, only  $nT$  processors can take part in this parallel algorithm.

Fig. 3. demonstrates a simple example to explain the parallel algorithm. Four processors P0 to P3 are used; individual assemblies are numbered from 1 to 19. The model uses two assembly mesh files 1 and 2 and one interstices mesh, making it three mesh files in total. Fig. 3A shows the processor, mesh file loaded and the copy/move task. Fig. 3B shows the CoreGen output mesh for a full hexagonal core with numbered assemblies.

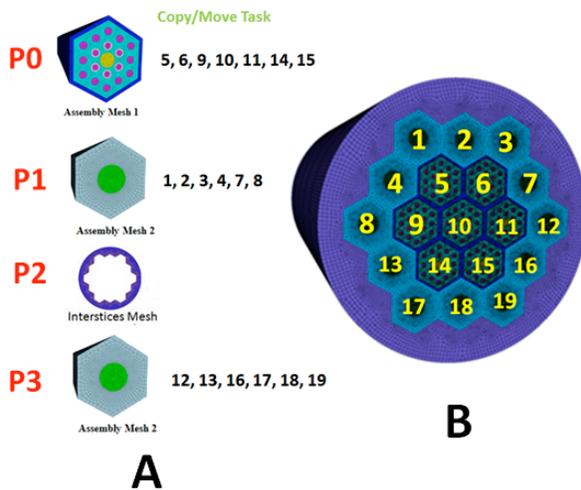


Fig. 3. Third stage of the geometry/mesh process, where CoreGen is executed.

Processor P0 loads assembly mesh 1, moves it to location 5 and then copies it to location 5, 6, 9, 10, 11, 14 and 15. Both processor P1 and P3 load assembly mesh 2; this mesh occurs 12 times in the core. The copy/move task is shared among P1 and P3 each handling 6 assemblies. P2 loads the interstices mesh and does not participate in the

copy/move process. Once the copy/move task is complete, CoreGen performs parallel merge, this algorithm does not delete mesh matching nodes, rather the parallel sharing information is modified to indicate that they are the same logical vertex [11]. Parallel merge mesh also significantly lowers the total wall clock time as shown in Example IV A. Finally, mesh is saved in parallel using MOAB's parallel HDF5-based writer. This writer can write a single output mesh file combining input from individual processors. Application codes may require having core mesh in separate small files for each processor and starting the simulation or they may require one mesh file from all the processors. CoreGen input file language defines a keyword "SaveParallel" which helps in specifying the option to save mesh from individual processor, one mesh file from all processors or both.

For several examples superlinear speedup was obtained, results for 1/6<sup>th</sup> VHTR core and MONJU reactor are discussed in Section IV A and IV C respectively.

### IIIB. Keywords and Options

Command line option "-m" and "-j" are added to CoreGen and AssyGen respectively. The "-m" option does not run the process; it only creates a makefile, which is often times desired for automatically generating the assemblies forming the core. The "-j" option in Assygen only creates a journal or mesh script file, without creating the geometry. Option "-t" was added to both the tools to printout detailed timing information, in case of CoreGen timing information is printed at the end of each step; for parallel version the maximum wall clock and CPU time required by a processor for each sub-process are given.

Fuel and other coolant pins are grouped by materials in the resulting assembly and core mesh. It is often desired to keep track of the pin/assembly number in the assembly/core model respectively. Post processing is one of the areas where it proves to be very useful to mark the pin and assembly numbers. For instance the radiation or temperature profiles for a particular pin in a particular assembly are measured experimentally and must be validated by simulations; experience has shown that it is tedious to recognize and locate that pin in hundreds of thousands of pins that form the reactor core. We achieve this by creating some information files as output of AssyGen and CoreGen program. These files can be loaded into the simulation software and populated along with the mesh to label the pins and assemblies.

"Info" keyword was introduced in both AssyGen and CoreGen to trigger the tools to generate extra files specifying pin/assembly number and their location. Both AssyGen and CoreGen generate a file with the base name suffixed with "\_info.csv", in case of AssyGen, this file has pincell number and location of the pin; for CoreGen the file contains assembly number, assembly index (mesh files 1 to 2 and number 1 to 19 in Fig. 3) and center of the

assembly. CoreGen also creates a file with base file name suffixed with “\_mesh\_info.csv” file. This contains the pincell number and the centroid of each element that belong to the specified pincell. Internally in AssyGen during geometry creation each pin is assigned a NAME tag [3] for material and an extra NAME tag for the pin number. We trick the system by creating each pin as a separate material in AssyGen stage. In CoreGen stage after retrieving the pin number from the assembly these extra materials are deleted.

This technique has been demonstrated with STAR-CCM+, where after loading the mesh file the info files with pin/assembly number and cell centroids are used to mark the pins using annotations defined in STAR-CCM+. This is achieved by means of a Java macro which reads the info file and populates the pin/assembly number based on the cell centroid to the existing simulation, the macro can be played from STAR-CCM+ GUI.

#### IV. EXAMPLES

In this section we present several examples. Example IV A is available in the MeshKit repository [7].

##### *IV.A. Sixth Core model, Constructed without /with Using Extrusion and Using Parallel-Enabled CoreGen*

In this example we construct the model shown in Fig. 4., one-sixth Very High Temperature Reactor (VHTR) core. We create this model using three different techniques and compare the performance of each technique. . The model consists of 11.8M hexahedral elements and 14M mesh vertices. There are a total of 58 assemblies in total and 12 different assemblies that form this core. This model is tailored to automatically run and create the core mesh from scratch without any user interaction. Appropriate values of axial and radial mesh size are specified in the input files. Makefile generated by CoreGen is used to run the problem.

First, this model was created by using the three-stage process described in this paper. The tools were run on a desktop Linux-based workstation with a clock speed of 2.5 GHz and 12 GB RAM. With the three-stage process without extrusion, it took 4 minutes to generate the geometries using AssyGen, 5 minutes to create hexahedral assembly meshes using CUBIT (version 12.2), and 15 minutes to generate the core model using CoreGen. Thus the total time using this method is 24 minutes.

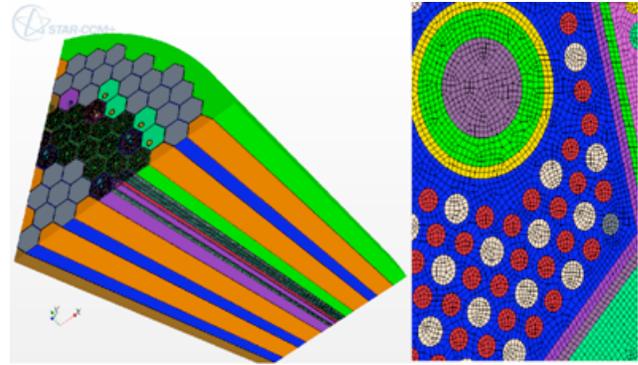


Fig. 4. One-sixth of a VHTR core model generated by using CoreGen (left); a closeup of assembly mesh in this model (right).

Second, the same model was constructed by using a four-stage process. First, two-dimensional assemblies were created using AssyGen; then, CUBIT was used to mesh these assemblies (with quadrilaterals); next, a two-dimensional core was generated by using CoreGen; in the fourth stage of this process, the entire 2D core mesh was extruded into the third dimension. Using this extrusion-based approach required 0.6 times the execution time of the three-stage process. The total execution time was reduced from 24 minutes to 15 minutes, or by almost 40%. This reduction was due primarily to the reduced number of vertices that needed to be copied, moved, and merged between assemblies during the CoreGen stage. For example, the number of vertices considered for merging was reduced from 162,690 in the 3D process to only 9,570 in the 2D CoreGen process.

The minimum and maximum shape metrics for the VHTR mesh are 0.00125 and 0.00618, respectively. These metrics are low (normally, shape metric values above 0.2 are deemed acceptable). However, the metric are due to the high aspect ratio of the assemblies (each assembly is 7.93 meters in length but only about 37 cm across) and the resulting mesh.

The performance of parallel CoreGen was measured using the without extrusion or three-stage process model described above. When using 56 processors for running CoreGen. The total execution time for creating this model is less than 10 mins (9 mins of serial execution time: AssyGen and Meshing + 0.33 mins of CoreGen time). CoreGen takes only 0.33 mins compared to 15 mins in the serial case.

CPU time in mins and maximum memory used for 1/6<sup>th</sup> VHTR core with 11.8M hexes.

procs	Copy/Move (mins)	Merge (mins)	Save (mins)	Total (mins)	Memory (GB)
1	10.3	3.8	0.4	14.7	3.18
8	5.8	7.5	0.35	13.8	2.13
16	0.2	6.7	0.18	7.2	1.33
32	0.03	0.9	0.09	1.06	0.41
56	0.004	0.2	0.06	0.33	0.2

Table I lists the CPU time and the maximum memory used for various steps of the CoreGen stage, when using different number of processors. It is observed that all the operations see super-linear speedups in some cases; memory usage data indicates that these steps are where the application goes from swapping to a state where the job fits in available memory. This indicates one important reason for parallelizing RGG, i.e. so the application can fit in memory without swapping. Next, mesh joining is observed to actually slow down going from one to eight processors; this is probably due to the communication overhead required in the parallel algorithm. However, at larger numbers of processors, 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 on increasing the number of processors.

#### VI.C. Full Core MONJU Reactor

Fig. 5 and 6 show a full core MONJU reactor which is made up of 8 different assembly types and consisting of 715 assemblies in total. AssyGen and meshing takes 5.5 minutes (serial process) and CoreGen on 712 processors on Fusion cluster at Argonne National Laboratory takes only 1.5 minutes to copy/move/merge and save 8 assemblies to 715 different locations in the core. The total wall clock time required to generate this 101M hexahedral element model is 7 mins.

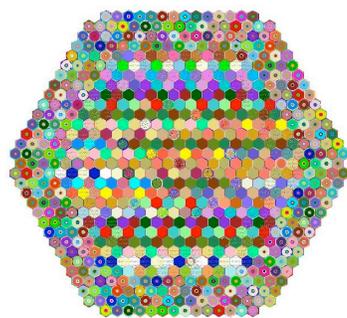


Fig. 5. Full core MONJU reactor (top view).

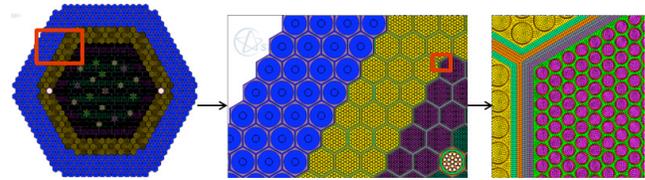


Fig. 6. Full core MONJU reactor; closeup area in red rectangular region is highlighted from left to right.

#### VI.D. 1/4<sup>th</sup> PWR Core Geometry

Fig. 7. shows the benchmark problem: “MOX Fuel Loaded Small PWR Core”, detailed description can be found on the website of Nuclear Reactor Analysis and Particle Transport Lab [12]. Individual assembly geometries are created using the AssyGen tool and then CoreGen tool is used to copy/move the assemblies and form the core geometry. It must be noted that trivial assemblies that do not contain any rods or the baffles are generated directly using CUBIT. When creating the core model “NeumannSet” keyword is used to create individual side faces of the core as boundary conditions. In Fig. 7. A, B and C are closeup of the area in red rectangular region highlighted on the core model. The model consists of approximately 11k volumes. On a Linux desktop: all the assembly geometry creation takes 8 mins and CoreGen takes 12 mins of wall clock time and uses 0.9GB of RAM.



Fig. 7. 1/4<sup>th</sup> PWR benchmark geometry with closeup views A, B and C showing details of the model.

## VII. CONCLUSION

The original three-stage approach for generating lattice-based models reported in earlier papers has proven to be very useful. Huge reduction in total clock time required to create geometry and mesh models for reactor cores are reported. CoreGen tool was modified to work in serial and parallel, parallel version of the tool allows the problem to fit in memory, superlinear speedups are observed due to the problem fitting in memory, thereby significantly reducing the total time required for generating large models. Parallel-enabled CoreGen has prompted new developments in parallel file save and mesh merge algorithm. AssyGen tool provides new keywords to aid creation of matching meshes between different assemblies forming the core. New options for reporting the timing, creation of only mesh script and makefile are added. Experience in using the tools has prompted the development of "Info" keyword which generates new files along with model files; these info files contain pin number, assembly number and their location. The info files can be useful in post processing and other such areas. New types of reactors are generated using the tools. The full core MONJU reactor example demonstrates the power of the parallel-enabled CoreGen tool. Geometry only models can be created, 1/4<sup>th</sup> PWR core geometry creation is presented.

## ACKNOWLEDGMENTS

We thank Jason Kraftcheck for his work on MOAB's parallel writer, Nathan Bertram for parallel merge mesh capabilities in MOAB, J. H. Thomas for helpful discussions and feedback during the development of the tools. We also thank the Fathom group at Argonne, who maintain the libraries required by this tool. This work was supported in part by the U.S. Dept. of Energy Office of Nuclear Energy Nuclear Energy Advanced Modeling & Simulation (NEAMS) Program; by the U.S. Dept. of Energy Office of Scientific Computing Research, Office of Science; and by the US Department of Energy's Scientific Discovery through Advanced Computing program, under Contract DE-AC02-06CH11357.

## NOMENCLATURE

nP: Number of processors.  
nA: Number of different assemblies forming the core.  
nT: Total number of assemblies forming the core.

## REFERENCES

1. T. J. TAUTGES AND R. JAIN, "Creating geometry and mesh models for nuclear reactor core Geometries using a lattice hierarchy-based approach," *Proc. 19th International Meshing Roundtable*, Springer, Berlin, 351 (2010).
2. T. J. TAUTGES AND R. JAIN, "Creating geometry and mesh models for nuclear reactor core Geometries using a lattice hierarchy-based approach", *Engineering with Computers*, (2011).
3. TAUTGES, T.J., KRAFTCHECK, J., PORTER, JIM, CACERES, ALVARO, GRINDEANU, IULIAN, KARPEEV, DMITRY, JAIN, RAJEEV, KIM, HONGJUN, CAI, SHENGYONG, JACKSON, STEVE, HU, JIANGTAO, SMITH, BRANDON, VERMA, CHAMAN, SLATTERY, STUART, WILSON, PAUL: MeshKit: A Open-Source Library for Mesh Generation. Proceedings, *SIAM Conference on Computational Science & Engineering*. SIAM, Reno, NV (2011).
4. T. J. TAUTGES, "CGM: A geometry interface for mesh generation, analysis and other applications," *Engineering with Computers*, 17, 486 (2005).
5. T. J. TAUTGES, R. MEYERS, K. MERKLEY, C. STIMPSON, AND C. ERNST MOAB: A mesh-oriented database, *SAND2004-1592*, Sandia National Laboratories, Albuquerque, NM (2004).
6. G. D. SJAARDEMA, T. J. TAUTGES, T. J. WILSON, S. J. OWEN, T. D. BLACKER, W. J. BOHNHOFF, T. L. EDWARDS, J. R. HIPPI, R. R. LOBER, AND S. A. *volume 1: Users manual*, Sandia National Laboratories, Albuquerque, NM (1994).
7. MeshKit website (2012), <http://trac.mcs.anl.gov/projects/fathom/browser/MeshKit>
8. Spatial website (2012) <http://www.spatial.com/>
9. Open CASCADE Technology website (2012), <http://www.opencascade.org>.
10. CHAMAN SINGH VERMA AND TIM TAUTGES, "Jaal: Engineering a high quality all-quadrilateral mesh generator", *Proc. 20th International Meshing Roundtable*, (2011).
11. TAUTGES, TIMOTHY J., KRAFTCHECK, JASON, BERTRAM, NATHAN, SACHDEVA, VIPIN, MAGERLEIN, JOHN, "Mesh Interface Resolution and Ghost Exchange in a Parallel Mesh Representation". Presented at the *26th IEEE International Parallel & Distributed processing Symposium*, Shanghai, China, May 21 (2012).
12. 1/4<sup>th</sup> PWR Benchmark Problem (2012), <http://nurapt.kaist.ac.kr/benchmark/>