SANDIA REPORT
SAND2010-0878
Unlimited Release
Printed February 2010

Foundational Development of an Advanced Nuclear Reactor Integrated Safety Code

Rodney C. Schmidt, Kenneth Belcourt, Kevin T. Clarno, Russell Hooper, Larry L. Humphries, Alfred L. Lorber, Richard J. Pryor, William F. Spotz

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia is a multiprogram laboratory operated by Sandia Corporation,
a Lockheed Martin Company, for the United States Department of Energy's
National Nuclear Security Administration under Contract DE-AC04-94AL85000.

Approved for public release; further dissemination unlimited.

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

NOTICE: This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government, nor any agency thereof, nor any of their employees, nor any of their contractors, subcontractors, or their employees, make any warranty, express or implied, or assume any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represent that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government, any agency thereof, or any of their contractors or subcontractors. The views and opinions expressed herein do not necessarily state or reflect those of the United States Government, any agency thereof, or any of their contractors.

Printed in the United States of America. This report has been reproduced directly from the best available copy.

Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865) 576-8401
Facsimile: (865) 576-5728
E-Mail: reports@adonis.osti.gov
Online ordering: http://www.osti.gov/bridge

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Rd.
Springfield, VA 22161

Telephone: (800) 553-6847
Facsimile: (703) 605-6900
E-Mail: orders@ntis.fedworld.gov
Online order: http://www.ntis.gov/help/ordermethods.asp?loc=7-4-0#online

SAND2010-0878
Unlimited Release
Printed February, 2010

Foundational Development of an Advanced Nuclear Reactor Integrated Safety Code

Rodney C. Schmidt, Russell Hooper, Larry L. Humphries, Alfred A. Lorber, and William F. Spotz
Sandia National Laboratories
P.O. Box 5800
Albuquerque, New Mexico 87185-0316

Kenneth Belcourt
Ktech Corporation
Albuquerque, New Mexico

Kevin T. Clarno
Oak Ridge National Laboratory
Oak Ridge, TN 37831-6172

Richard J. Pryor
Consultant
Albuquerque, New Mexico

Abstract

This report describes the activities and results of a Sandia LDRD project whose objective was to develop and demonstrate foundational aspects of a next-generation nuclear reactor safety code that leverages advanced computational technology. The project scope was directed towards the systems-level modeling and simulation of an advanced, sodium cooled fast reactor, but the approach developed has a more general applicability. The major accomplishments of the LDRD are centered around the following two activities.

- The development and testing of LIME, a Lightweight Integrating Multi-physics Environment for coupling codes that is designed to enable both “legacy” and “new” physics codes to be combined and strongly coupled using advanced nonlinear solution methods.
- The development and initial demonstration of BRISC, a prototype next-generation nuclear reactor integrated safety code. BRISC leverages LIME to tightly couple the physics models in several different codes (written in a variety of languages) into one integrated package for simulating accident scenarios in a liquid sodium cooled “burner” nuclear reactor.

Other activities and accomplishments of the LDRD include (a) further development, application and demonstration of the "non-linear elimination" strategy to enable physics codes that do not provide residuals to be incorporated into LIME, (b) significant extensions of the RIO CFD code capabilities, (c) complex 3D solid modeling and meshing of major fast reactor components and regions, and (d) an approach for multi-physics coupling across non-conformal mesh interfaces.

ACKNOWLEDGMENTS

We gratefully acknowledge the important contributions and support provided by the following individuals.

- Chris Moen for providing remarkably responsive technical support and code updates of the RIO CFD code in support of our project, including the incorporation of the Brinkmann-Forchheimer equations.
- Mike Borden, Michael Brewer, and Zachariah Pickett for their expert use of CUBIT and other tools to generate all of the different solid models and meshes (both simple and complex) for use in this project.
- Xiangmin Jiao (Assistant Professor at Stony Brook University) for providing the Surfdiver code, answering many questions about its use, and performing several important bug fixes that were needed for our problem space.

Special thanks are also expressed to Marianne Walck, John Kelly, Dana Powers and Randall Summers for their steady advocacy and management support of this work.

This work was supported by the Laboratory Directed Research and Development (LDRD) program at Sandia National Laboratories.

CONTENTS

FIGURES

TABLES

NOMENCLATURE

ABTR	Advanced Burner Test Reactor
BRISC	Burner Reactor Integrated Safety Code -- the proto-type code developed here.
C	The C programming language
C++	The “C++” programming language
CFD	Computational Fluid Mechanics
DOE	Department of Energy
DRACS	Direct Reactor Auxiliary Cooling System
F77	The Fortran 77 programming language
F90	The Fortran 90 programming language
FSI	Fluid Solid Interface
GNEP	Global Nuclear Energy Partnership
IHX	Intermediate Heat eXchanger
IPSC	Integrated Performance and Safety Code
JFNK	Jacobian Free Newton Krylov
LDRD	Laboratory Directed Research and Development
LIME	Lightweight Integrating Multi-physics Environment for coupling codes
LMR	Liquid Metal Reactor
LWR	Light Water Reactor
MELCOR	A systems level severe accident analysis code developed by the NRC
MWt	MegaWatt thermal
NPP	Nuclear Power Plant
NRC	Nuclear Regulatory Commission
ORNL	Oak Ridge National Laboratory
PIRT	Phenomena (or Process) Identification and Ranking Table
RIO	An unstructured finite volume code for solving the low Mach number Navier-Stokes equations with heat and mass transfer
SNL	Sandia National Laboratories
ULOF	Unprotected Loss-Of-Flow accident

1. INTRODUCTION AND BACKGROUND

This LDRD project concerns the development of an advanced proto-type “systems-level” integrated performance and safety code (IPSC) for nuclear reactors. Fundamental to this work was the concurrent development of a lightweight multi-physics coupling strategy (and associated software) that would enable the assembly of different codes and models into a new multi-physics simulation capability.

1.1 Motivation

Global environmental concerns, uncertainties in fossil fuel resources, and a growing worldwide demand for energy have led to a dramatic resurgence in efforts to develop, license, and build advanced nuclear power systems. New systems are expected to be safer, reduce the amount of nuclear waste, and address proliferation concerns while still remaining economically attractive. In this context a variety of different reactor types are being considered. For example, the Global Nuclear Energy Partnership (GNEP) [1] program proposed a nuclear fuel cycle based on an advanced “fast” reactor technology. These reactors would be optimized for incineration of long-lived actinides, resulting in more efficient long-term storage options compared to the current once-through fuel cycle. In addition, advanced high temperature gas cooled reactors (HTGC), small modular reactors, and a new generation of advanced light-water reactor designs are being investigated.

With new reactor types and designs being seriously considered, a great need exists to develop new and better tools that leverage advanced computational tools and techniques. This is because legacy reactor analysis codes currently available (e.g. MELCOR [2], SASSYS [3], CONTAIN [4], etc.) are based on decades-old serial computing technologies and associated modeling strategies. Major simplifications of the reactor and associate plant components were necessitated by limitations of the engineering and computational sciences of the time. By leveraging modern high performance computational technologies, vastly improved reactor analysis and safety analysis tools are now possible. In addition to providing a deeper understanding of the underlying physical phenomena and fundamental processes, this will greatly improve our ability to quantify reactor safety margins by significantly reducing uncertainty bounds, reduce the number of expensive large-scale systems tests that may be required to demonstrate safety, and minimize the risk of regulatory complications and unacceptably high overall program costs. Estimates suggest that the economic benefits derived by reducing power distribution uncertainties by only 1 to 2% are in the millions of dollars.

1.2 Scope, objectives and focus

The objective of this LDRD was to develop and demonstrate foundational aspects of a next-generation systems-level nuclear reactor safety code that leverages advanced computational technology. By "systems-level" we mean that all important system components and physical processes relevant to assessing the response of a nuclear reactor to normal, off-normal or hypothetical accident scenarios are treated. The LDRD work scope focuses on sodium-cooled

fast reactors, a reactor type that was of particular interest to the GNEP program at the beginning of this project. However, from a foundational standpoint the development activities have a much broader application space.

Several factors have guided the decisions made while pursuing the LDRD goals. First, although new computational technologies (e.g. advanced linear and non-linear solvers, parallel scaling tools and strategies, improved numerical methods, etc.) are vitally important, they are in fact only some of the puzzle pieces. Equally important to success are factors such as code portability, "ease of use" by potential users, and the ability to leverage current generation tools when appropriate.

To license a nuclear power plant, a range of computational analysis is required in order to address safety issues of the system during normal, off-normal, and severe-accident conditions. Appendix C gives details of an assessment activity conducted at the beginning of this project whose objective was to help define these requirements for a potential new “burner” reactor system. While assessing the physics and phenomena required to model various scenarios it became clear that addressing all physical processes associated with all potential accident scenarios (including severe accidents) was not possible within the LDRD project resources. The scope of the physics addressed in this work was therefore limited to those processes that occur in the “Phase-1” stage of an accident. For our purposes the Phase 1 and Phase 2 periods are defined as follows:

> Phase 1 - Initial transient: Begins at the accident initiating event and includes the time period when the state of the system moves from its normal operating state to a point where the structural geometry or material phases of system components begin to change.
>
> Phase 2 - Damage transient: Extends from the point when structural or material changes begin to occur until the end of the accident.

Despite the limitations of scope imposed on the LDRD work, it was clearly recognized that any improved approach must be based on more effectively addressing the following four fundamental major challenges:

(1) the multi-scale issue,
(2) the coupled multi-physics issue,
(3) the complex geometry issue, and
(4) the uncertainty quantification issue.

Of these four, only the first three are directly addressed in this LDRD project.

1.3 The Argonne advanced burner test reactor preconceptual design

Prior to the beginning of this LDRD, Argonne National Labs released a report describing a pre-conceptual design of a 250 MWt "Advanced Burner Test Reactor" (ABTR) that was prepared in support of the GNEP effort [5]. As described in this report, the ABTR is an advanced "pool-type fast reactor design, with the reactor core, primary pumps, intermediate heat exchangers, and direct reactor auxiliary cooling system heat exchangers all immersed in a pool of sodium coolant

within the reactor vessel." The reactor core consists of 24 assemblies in an inner enrichment zone and 30 assemblies in an outer zone. Reactivity control and neutronic shutdown are provided by seven primary and three secondary control rod assemblies. The level of detail provided in Reference [5], together with its advanced design features, made the Argonne ABTR design an ideal target reactor for this LDRD project to focus on.

By viewing several key figures from the ABTR design report, one can more easily grasp the large complex multi-scale, multi-component and multi-physics nature of the problem to be addressed by a systems-level IPSC. Figures 1.1 through 1.6 are reproduced from Reference [5] and provide a visual overview of various plant and reactor system components that range in scale from the size of the overall reactor plant site (of order 100 meters) to that of the fuel pin cladding (of order 10^{-4} m).

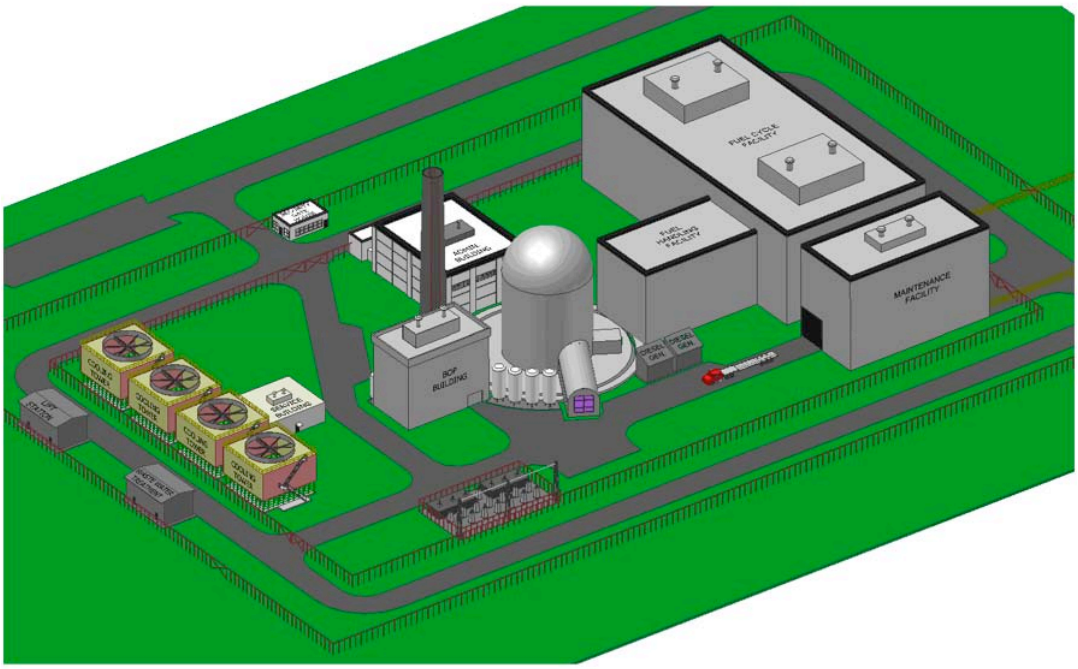


Figure 1.1 Overall site view of the Argonne advanced burner test reactor preconceptual design as shown in Reference [5].

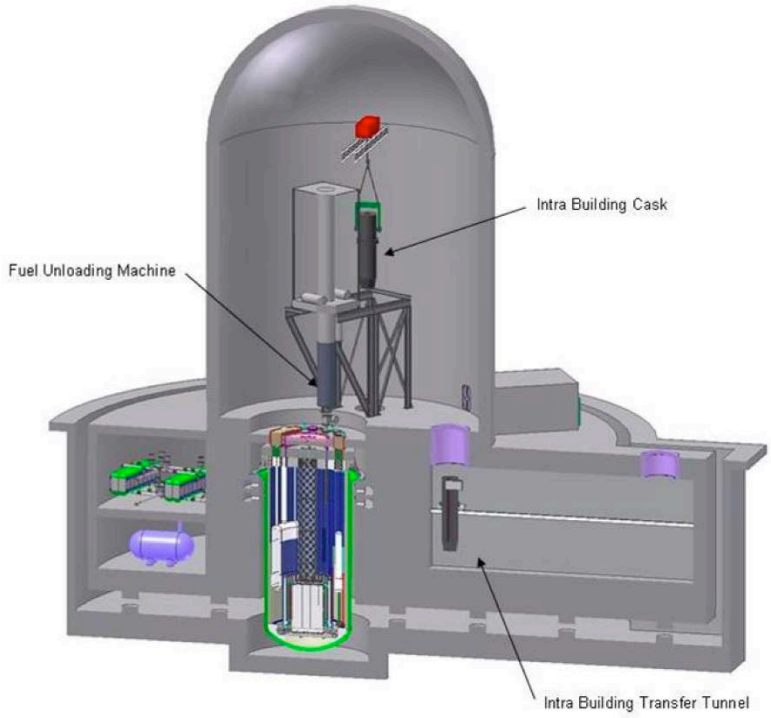


Figure 1.2 Schematic of the containment building and fuel handling system for the Argonne advanced burner test reactor preconceptual design as shown in Reference [5].

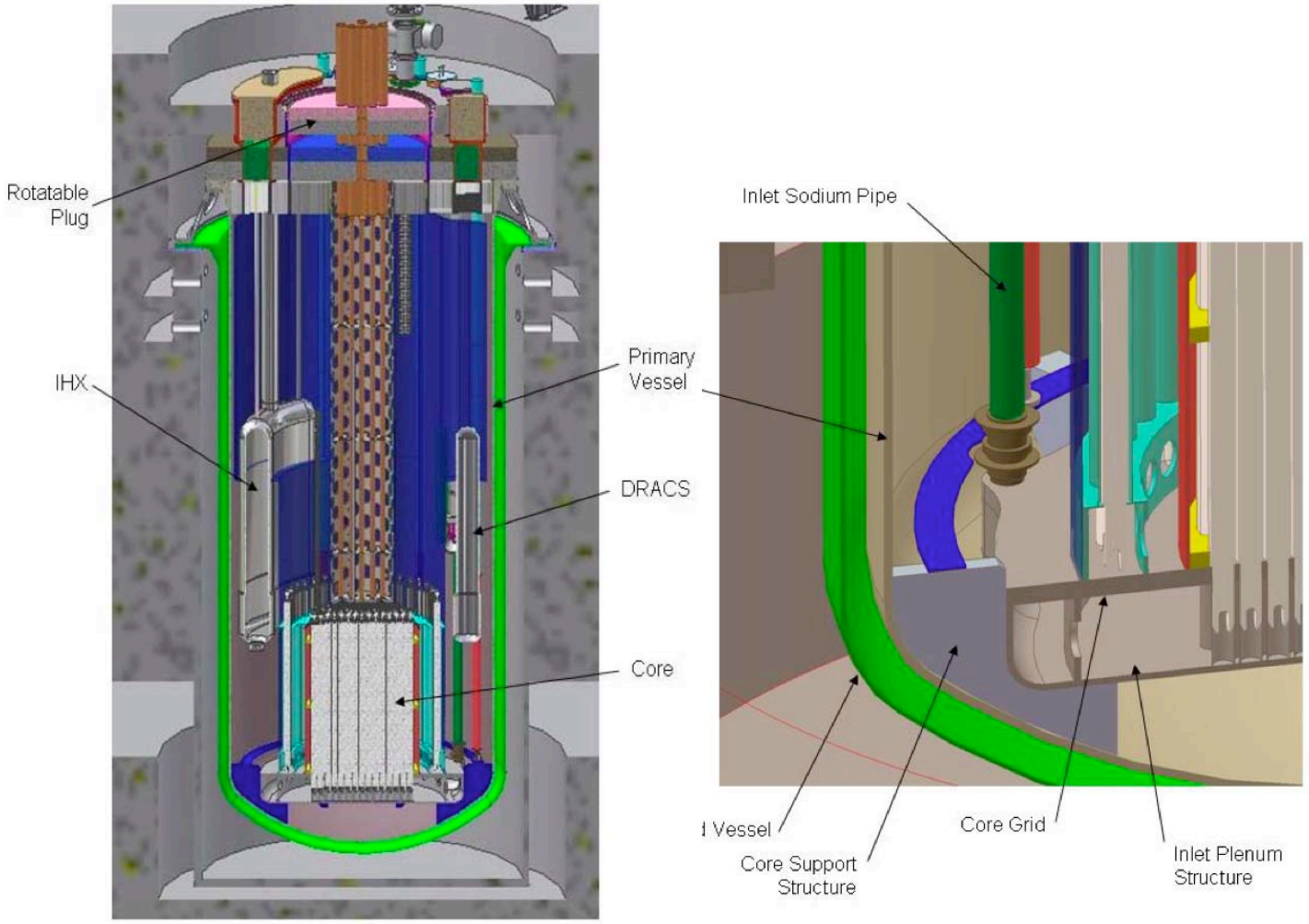


Figure 1.3 Reactor vessel, primary system, and lower support structures in the Argonne advanced burner test reactor preconceptual design as shown in Reference [5].

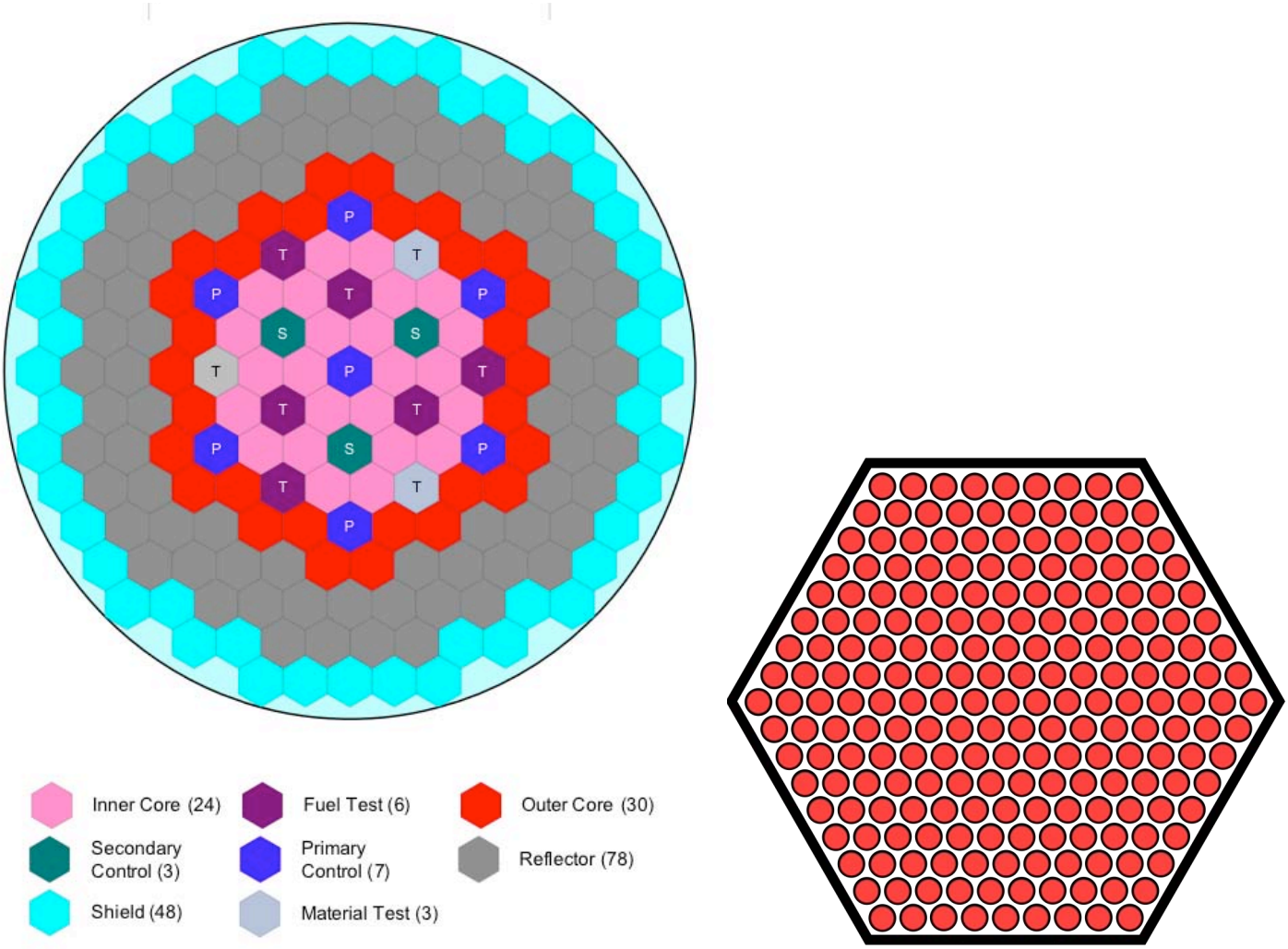


Figure 1.4 Cross sections of the reference core configuration and a 217 pin fuel canister in the Argonne advanced burner test reactor preconceptual design as shown in Reference [5].

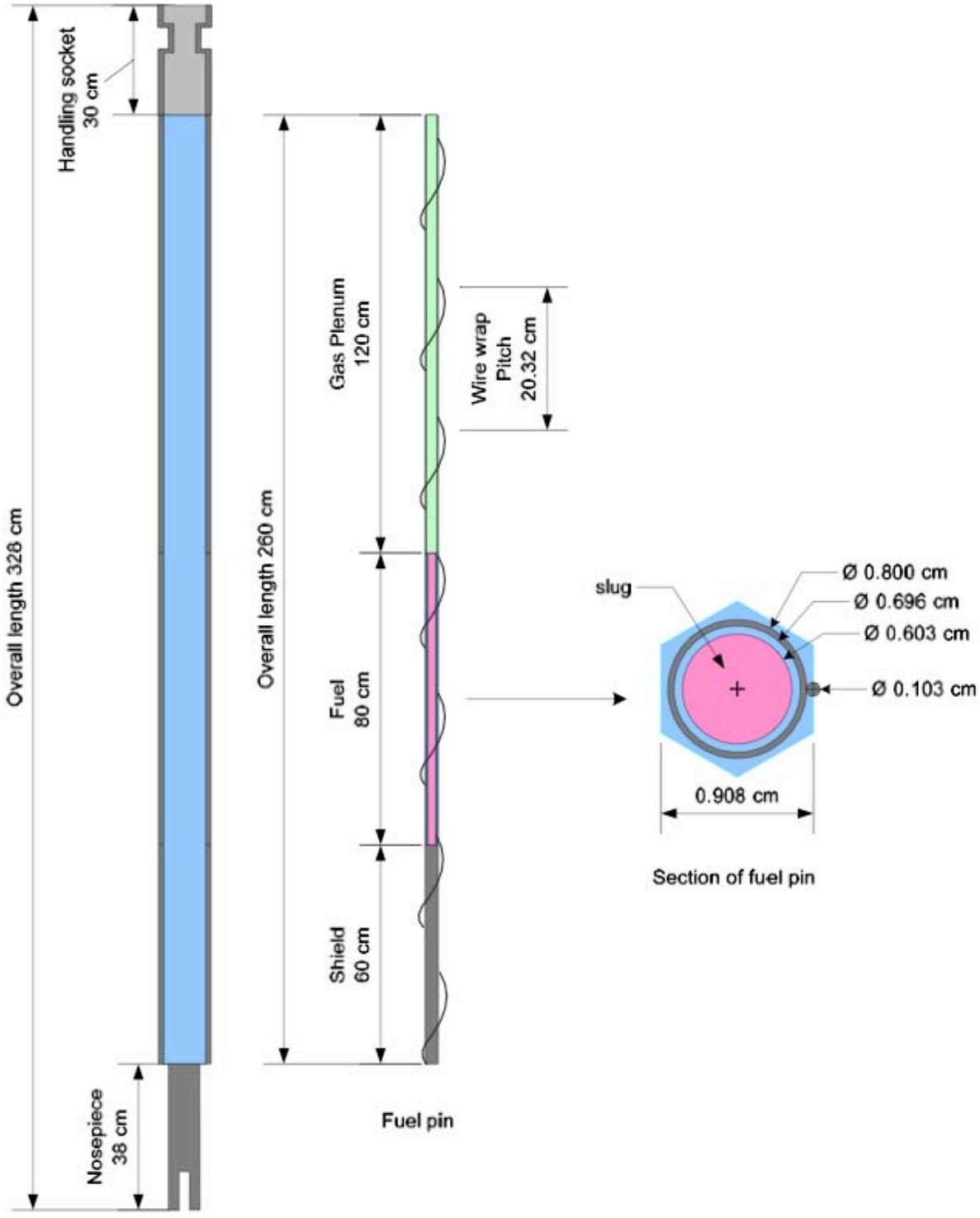
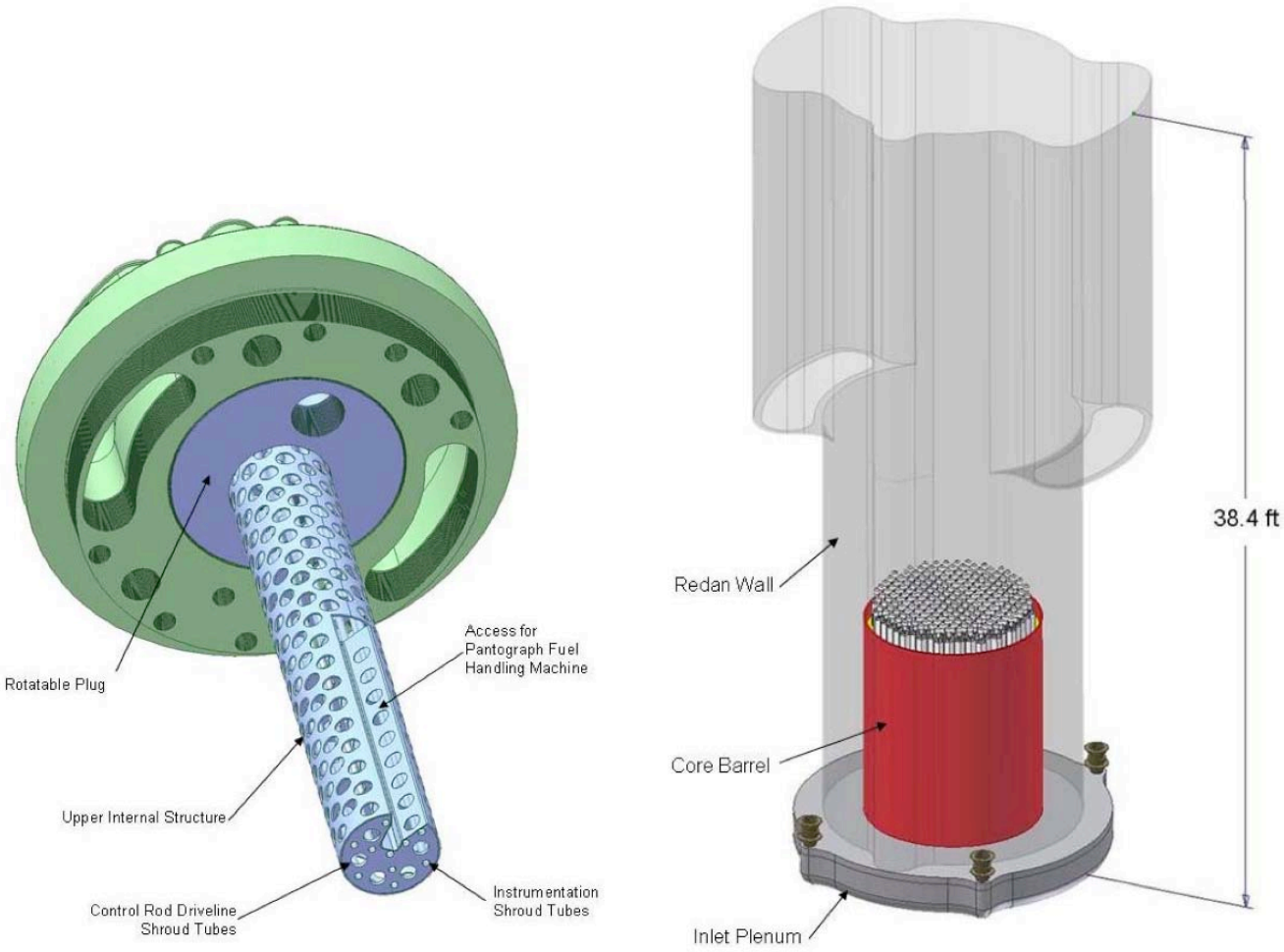


Figure 1.5 Fuel assembly schematic and fuel pin cross section in the Argonne advanced burner test reactor preconceptual design as shown in Reference [5].



a. Upper internal structure b. Redan structure

Figure 1.6 Schematic illustrations of the upper internals, redan, and core barrel structures in the Argonne advanced burner test reactor preconceptual design as shown in Reference [5].

2. DEVELOPMENT OF A "LIGHTWEIGHT" MULTI-PHYSICS COUPLING APPROACH

2.1 Central importance and desired attributes

The need to obtain accurate solutions of "coupled" "transient" "multi-scale" "multi-physics" problems is becoming ubiquitous in scientific and engineering venues, including the nuclear engineering community. As illustrated above in Section 1.3, an operating nuclear reactor system consists of many interacting components and involves many physical processes. For different purposes, various subsets and/or all of these system components together may need to be simulated to assess their performance or safety characteristics under normal and off-normal conditions. The physics of these processes are described by models and equation sets that evolve the important state variables (e.g. temperature, density, velocity, neutron flux, etc.) in time over the spatial domain of relevance. Because these processes interact with one another, the equations must be solved in such a manner that changes in any state variable are properly reflected in all equation sets that are affected by that variable. If the effect of changes in variable “X” on the evolution of variable “Y” is small, then the coupling is said to be weak. Conversely, if the effect is large, then the coupling is said to be strong. Another important aspect of coupling is whether the effect is a linear effect, or a nonlinear effect. The most challenging problems to model accurately are those where the coupling is both strong and nonlinear -- which is the case for many of the coupled processes in a nuclear reactor system.

In complex systems like a nuclear reactor, multi-physics coupling may occur within a shared spatial domain, across an interface, through action at a distance (e.g. thermal radiation) or by some combination of these. Consider first the shared interface case where in one region one set of physical processes are at play, but in a neighboring domain, a different or more limited set of processes may govern the behavior. These two regions are coupled through their shared interface where, for example, heat transfer may occur. It is common, and often useful, to do separate simulations for which the coupling is either neglected or approximated. Often, distinctly different computer codes are written to treat the particular set of physics associated with a given component or region. However, these codes may represent the shared interface quite differently and with a different level of resolution. If the coupling is strong, a complete system simulation can only be accurate if these different domains are properly coupled during the computation.

When the coupled physics is within a shared spatial domain, other types of modeling challenges can arise. One common issue is when the numerical discretization appropriate for one set of physics is different from that required for one or more of the other physical processes. For example, heat transfer and fluid flow within a reactor core may require one type of mesh, but the neutronics code that computes the local neutron flux (and associated heat generation within the fuel) may require a different mesh. Different codes are often used to support the modeling of these different physical processes. This situation then requires some means to transfer the state variable representation on one mesh to an equivalent representation on the other mesh, and vice-versa.

For many years the question of how best to solve strongly coupled nonlinear multiphysics problems in complex geometries has been an area of active research, and there are many facets to this problem. The approaches used in computer codes written 20-30 years ago in the nuclear reactor industry were limited both by the computational resources available and the maturity of the field of computational science. Many, if not most, computer codes currently in use by the nuclear industry and/or the NRC today originated during this time period. However, after the investment of many man-years of development, they reflect a legacy of use, qualification, testing, lessons learned, model improvement and refinement that cannot be quickly or easily duplicated in new computer codes. Some also have substantial user communities throughout the world. A consideration of this situation led our project team to pursue different strategies for addressing the multi-physics problem than other efforts pursued in recent years (e.g. [6] [7]).

Although new computational technologies (e.g. advanced linear and non-linear solvers, parallel scaling tools and strategies, improved numerical methods, etc.) are critical, other factors are also of great importance. For our needs an ideal multi-physics environment should, in our view,

- enable both new and existing applications to be combined with strong coupling (when needed) though non-linear solution methods (e.g. JFNK, fixed point),
- preserve and leverage any specialized algorithms and/or functionality an application may provide,
- minimize the requirements barrier for an application to participate,
- work within advanced solver frameworks (e.g. as extensions to the Trilinos/NOX nonlinear solver libraries),

and should not be limited to:

- codes written in one particular language,
- a particular numerical discretization approach (e.g. Finite Element), or
- physical models expressed as PDE's.

The approach described here reflects our efforts to achieve these objectives.

2.2 Description of LIME

Figure 2.1 illustrates key components of a lightweight multi-physics coupling environment developed in this LDRD. By lightweight we mean two things. First, the main software is relatively small in size and complexity, requires only a few standard libraries to build (all openly available) and can be easily ported to a wide range of computing platforms. Second, the constraints placed on the codes and models to be incorporated are modest in scope.

This approach builds on earlier work [8, 9, 10], and on the Trilinos/NOX software libraries [11, 12, 13, 14]. These provided important building blocks and components as a starting point for the work completed in this LDRD. We have named the approach **LIME**, an acronym for **Li**ghtweight **I**ntegrating **M**ulti-physics **E**nvironment for coupling physics codes.

At the highest level (level 3), LIME currently consists of a Multi-Physics driver, a Problem Manager, and the Trilinos/NOX non-linear solver library. (Note that an additional component for

sensitivity and uncertainty quantification purposes is envisioned as being added in the future.) At the middle level (level 2) are an arbitrary number of "Model Evaluators"; one for each physics code being coupled. Model Evaluators basically serve to transform (or wrap) a previously stand-alone physics code into a call-back subroutine that can communicate with the Problem Manager. Additional Model Evaluators used for transferring data between the physics codes also participate but are not shown. The Problem Manager in turn handles aggregation and segregation of data and functionality embodied in each Model Evaluator and presents itself as another Model Evaluator (a meta-Model Evaluator) to the Trilinos Solver Library and the Multi-Physics Driver. At the lowest level of the hierarchy are the physics codes themselves, which only communicate directly to their respective model evaluator. The next three sub-sections provide a more detailed discussion of these three levels in LIME.

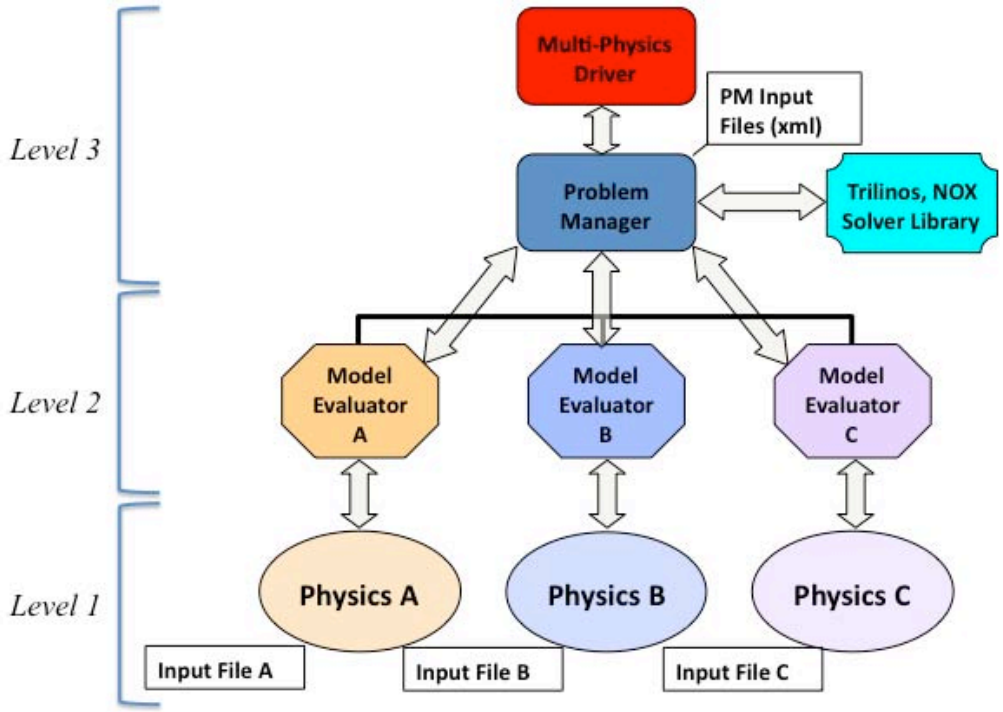


Figure 2.1 Illustration of key components of **LIME**, a **Li**ghtweight **I**ntegrating **M**ulti-physics **E**nvironment for coupling physics codes.

Note that LIME by itself is not a code for doing multi-physics simulations. Instead, LIME provides the key high-level software, a well defined approach (including example templates for Model Evaluators), and interface requirements for participating physics codes that enable the assembly of these codes into a new multi-physics simulation capability unique to the class of problems of interest for a particular need. The prototype BRISC code described in Chapter 5 (for systems level nuclear reactor safety calculations) is one example of a code built using LIME.

2.2.1 The Multi-Physics Driver (MP_Driver), Problem Manager, and Solver Library

2.2.1.1 The MP_Driver

The MP_Driver is the highest-level software component in LIME. At present it contains approximately 600 lines of C++ code. It performs the following three administrative functions:

1. Do all set-up tasks for the problem that is to be run, including creating a Problem Manager specific to the problem. The set-up phase includes the following specific tasks:
    - If running in parallel, create the mpi communicator based on the number of processors requested.
    - Read the XML input file containing problem and transfer configuration specifications
    - Create the multi-physics coupling Problem Manager object with solution strategy and MPI communicator
    - Initialize transfer object creation
    - Set up (create) and register the physics package objects for each Model Evaluator / Physics Code pair.
    - Set up data transfers between Model Evaluators
    - Call the Problem Manager to set up the output
2. Call the Problem Manager to solve the problem
3. Gracefully end the simulation

2.2.1.2 The Problem Manager

The Problem Manager is the central orchestrating software component that controls the simulation being performed. It is designed to encapsulate the various Model Evaluators being coupled, to drive various types of fixed-point coupling iterations and to present a single-problem view to the Trilinos solver framework when performing tighter coupling via JFNK (described later). For the LDRD we adopted a design for the Problem Manager as extensions to the NOX nonlinear solver package in Trilinos [14]. We started with a prototype from a previous LDRD [10] and expanded the API to incorporate more functionality embodied in physics codes as well as to lower the barrier to entry for physics codes to participate in coupling algorithms. We also made the Problem Manager configurable from input files in a manner consistent with how other solver packages in Trilinos are configured

The Problem Manager registers physics codes and associated data transfers wrapped as Model Evaluators (by requiring that they inherit an EpetraExt ModelEvaluator base class). It can then query each Model Evaluator to ascertain what input arguments it can receive and what output arguments it can provide given the input arguments. Based on the results of the queries, various coupling algorithms are possible, and the Problem Manager checks to make sure its configuration settings from an input file are compatible with the functionality available from the registered Model Evaluators. For example, the Model Evaluator for Physics Code A can indicate that it supports a state, x, and with this can compute and provide a residual $R_A(x)$. This would permit it to participate in various Newton-based coupling algorithms. Applications unable to provide residuals can only participate in a coupling algorithm via nonlinear elimination or fixed-point. If a physics code can provide additional callback support such as its own Jacobian, the

structure of its Jacobian or its own specialized solution method, these can be incorporated into the Jacobian or a preconditioner or both for the complete coupled problem.

At present the Problem_Manager supports time stepping of the various applications by requiring applications to supply a candidate time step size and then negotiating a most conservative value to be used by all. Extension to more elaborate operator splitting methods is possible but was beyond the resources of the LDRD. Hooks are also provided for pre- and post-time step functionality to enable such functionality as cycling time plane values, I/O, computing derived quantities, computing a predictor, check-pointing a solution, etc.

Because each application is wrapped as a Model Evaluator, the Problem Manager can apply a variety of coupling algorithms, ranging from fixed-point approaches to various implementations of the Jacobian-Free Newton-Krylov (JFNK) method [15]. In addition it is also possible to mix coupling algorithms so that some parts of the overall coupled system are treated using fixed-point while others are coupled more tightly using JFNK. This may be desirable in cases where the dependence among some applications is either weak or inherently one-way. (This mode is possible but has not been tested as of yet.) In all cases, transfers of data among the applications must be performed in a manner consistent with each application's requirements, e.g. temperature from A to B and heat flux from B to A. These transfers can range from very simple copies of data to very elaborate mesh interpolations of derived quantities corrected to preserve some physical conservation property across nonconformal surface discretizations (e.g. see Section 4.6 below). The Problem Manager accommodates data transfers of general complexity by simply treating them as additional Model Evaluators. Any specialized solution techniques that are needed to perform the transfer can be implemented by the user.

At present the LIME Problem Manager contains approximately 2200 lines of C++ code.

Key tasks of the Problem Manager can be outlined as follows.

1. Create the global state vector X representing the degrees of freedom for the single-problem view of the coupled problem and define mappings between X and the input arguments to each Model Evaluator representing either a physics code or a data transfer.

2. Configure and initialize the fixed-point and/or JFNK coupling solvers.

3. Perform time integration, including the following for each time step:
    - Perform time-step control: Negotiate/calculate based on all the physics.
    - Request a "predicted" solution state for time step n+1 from each physics code (if available) through its model evaluator.
    - Obtain a converged solution for time step n+1 using a non-linear solution strategy defined by the user input. Current options include simple fixed point iteration and JFNK using the Trilinos/NOX nonlinear solver library.
        - Request residuals from physics codes
        - Request physics-based preconditioning
        - Update state vector
    - Perform output (each physics code)

2.2.1.3 The Trilinos Solver Framework and NOX Nonlinear Solver Library

The Problem Manager of the previous section was designed to fit within and leverage the Trilinos Solver Framework [11, 12]. We achieved this by inheriting and expanding various APIs from the NOX nonlinear solver package [13, 14] and the EpetraExt Model Evaluator. The resulting design allows specialized functionality such as physics-based preconditioning, time step predictors, and internal subcycling to be incorporated as pieces of higher level coupling algorithms. Moreover, the Problem Manager can present itself as a single-problem callback interface to NOX so that all the nonlinear solver algorithms as well as associated interactions with linear solver and preconditioner packages in Trilinos can be immediately employed. Benefits to this include computing an approximate numerical Jacobian matrix efficiently using colored finite-differences, solving smaller representative coupled problems using a direct linear solver and doing cost-performance comparisons for different types of preconditioners used with iterative linear solvers. All of these proved useful to the BRISC project in achieving converged solutions to real-world multi-physics problems and in debugging.

2.2.2 Overview of Coupling Algorithms

This section provides a brief overview of coupling algorithms employed in the BRISC LDRD. A more detailed description of the general coupling algorithms can be found in [10].

2.2.2.1 Fixed Point Coupling

Fixed point coupling is the simplest approach to multi-physics coupling. In short, it seeks to find a solution X to a system of nonlinear equations such that X=F(X), hence the name. In practice within a multi-physics setting, fixed-point iterations are performed by sequentially solving each physics code, performing any data transfers needed for the physics code before its solve. The sequence is repeated until the solution states of all coupled code doesn't change and/or until other convergence measures are satisfied, e.g. residual norms from all codes are below a prescribed tolerance. Typically, fixed-point iterations converge linearly and can be more robust than stronger coupling algorithms (discussed next). Robustness here means that an initial guess for the nonlinear solution may converge where the same initial guess fails to converge for other coupling algorithms. In terms of implementation complexity, fixed-point coupling can be performed in a script-like manner where the "solve" method for each physics code is called along with intermediate data transfers and a test for convergence performed at the end of each fixed-point sequence. For this LDRD we consider the ability to participate in fixed-point iterations the lowest barrier to entry for an application and designed the Problem Manager to drive fixed-point iterations in ithe script-like manner as one option for coupling.

2.2.2.2 Jacobian-Free Newton Krylov (JFNK) Coupling

A key theme of the BRISC LDRD is leverage; we want to leverage both modeling capabilities as well as algorithm capabilities and allow for rapid turnaround of results when changing components of either. Accordingly, we designed the Problem Manager to take any functionality exposed by Model Evaluators and then use it to perform stronger types of coupling. We consider the strongest form of coupling to be an ideal represented by a monolithic physics application

employing an exact Newton method to solve its nonlinear system of equations. This would imply accurate and full accounting of sensitivities (derivatives) of all equations to all variables (ie a Jacobian matrix computed exactly) as well as exact linear solves with no numerical round-off when computing updates to the problem variables using Newton's method. This has long been recognized as prohibitively expensive if not impossible. This LDRD seeks to approach this ideal using approximations to Newton's method constructed from available functionality exposed by the various Model Evaluators comprising a particular coupled problem. We achieved this by designing the Problem Manager to implement approximations to Newton's method using at a minimum the same functionality required for fixed-point iterations. Our approach is value-added. If physics codes can expose more data or functionality through their respective Model Evaluators, the Problem Manager can leverage this to move the approximate Newton method closer to the ideal. The core of this approach is based on JFNK.

In short, JFNK is Newton's method implemented with approximations and with minimal restrictions. The key approximation is that the linear problem solved at each Newton (nonlinear) iteration can be performed without requiring the Jacbobian matrix to be formed; only its action on a vector is needed. This implies use of an iterative (Krylov) linear solver which in practice requires some form of preconditioning to obtain linear-problem convergence. Preconditioning is usually performed using a matrix that approximates the true Jacobian, hence the designation of the method as Jacobian-Free instead of Matrix-Free. To implement JFNK, the Problem Manager registers itself with NOX as the callback interface for residual fills and aggregates data from Model Evaluators (e.g. residuals) to send to NOX as well as segregates data (eg solution state) from NOX to send to respective Model Evaluators. Value–added enhancements come from the ability of the Problem Manager to aggregate additional data, e.g. a physics code's Jacobian matrix, or additional functionality, e.g. the action of a physics code's specialized preconditioner, into the JFNK algorithm.

One important feature of the Problem Manager developed as part of this LDRD is the ability to incorporate into the JFNK coupling algorithm a physics code that only provides a response (i.e. output) given appropriate input. The balance of plant component as well as the neutronics component are examples of this type of Model Evaluator. The Problem Manager incorporates these "response-only" Model Evaluators into the JFNK algorithm using nonlinear elimination. Every time NOX requests a computation of the residual vector for what it views as a single-problem nonlinear problem, the Problem Manager converts the state vector provided by NOX into the appropriate inputs (via data transfer Model Evaluators) to the response-only Model Evaluators, triggers their "solve" method and converts the responses into the forms needed by dependent physics codes before calling the residual computations for the Model Evaluators that can provide residuals. This approach captures the equation-to-variable sensitivities of all physics codes being coupled in a manner consistent with Newton's method. Making this work robustly for real problems required some significant effort which involved variable scaling which was also built into the Problem Manager.

2.2.3 Model Evaluators

A Model Evaluator must be written for each physics code being coupled. A Model Evaluator is a customized piece of C++ software that is based on the particular physics code it controls and the set of interacting physics codes whose respective state variables are needed by the model

equations being solved in the Physics Code. It inherits the Problem Manager base class and implements supported Problem Manager interfaces in terms of calls to the underlying physics package.

Each Model Evaluator has control of one defined piece of the overall state vector. It also must have access to all other state vector pieces that appear directly, or indirectly (e.g. through property variation affects) in its equation set or models.

Model Evaluators communicate up to the Problem Manager, between themselves, and down to a specific Physics Code. For very simple physics models, the model equations may be contained within the Model Evaluator itself, for example, as an included subroutine.

Appendix B provides additional details about the use and design of Model Evaluators in LIME.

2.2.4 Physics codes

Few restrictions are associated with the kinds of models or physics codes that can be coupled using LIME. As described in later section, separate codes written in Fortran, C and C++ were successfully coupled into a single multi-physics simulation code. However, in order to interface with LIME, some modifications are typically required. These include the following.

1. Console IO must be redirected (no pause statements or read/write to standard streams (cin, cout, cerr, clog)).
2. The Physics Code must be written and compiled as a library so the Multi Physics Driver can link to it (i.e. no “main”).
3. The Physics Code must be organized into several key parts that can be called independently. i.e.
    a. Initialization
    b. Solve
    c. Output
4. Routines to perform the following functions must be available (probably added).
    a. Register coupling capabilities
    b. Pass control variables
    c. Compute and pass data for coupling to other Physics Codes
    d. If the Physics Code supports MPI, then a function to coordinate MPI with the Multi Physics Driver must be written
5. Additional optional routines that could be used if available include functions to:
    a. Compute residuals
    b. Perform preconditioning (e.g., physics based preconditioning)

We have also learned that global data (e.g. F77 common blocks, F90 modules, C extern and static) must be treated with care to avoid problems with subtle errors for non-mangled names (link data symbol instead of identically named code symbol), and because global or static data can inhibit threading.

3. 3-D GEOMETRIC MODELING AND MESHING ISSUES

This section describes our approach to representing the complex geometry of the targeted advanced burner reactor preconceptual design introduced in Section 1.3. Although many other meshes were created for various test problems used during the code development process, these were relatively simple meshes and are not discussed here.

3.1 Discrete representation of the multi-scale multi-physics domain

The complex multi-scale nature of the overall reactor system geometry precludes representing all geometrical features of all system components with a fine numerical mesh. In our approach we address this issue by dividing the modeling problem into three different length-scale domains; a "subgrid" scale, a "resolved" meso-scale, and a "plant" scale. Figure 3.1 illustrates this strategy with reference to the target ABR design.

In BRISC, the goal is for all major geometric aspects of each major in-vessel reactor component to be resolved on a fully 3 dimensional unstructured mesh that captures features down to a scale of ~ 2-10 cm. This resolution is sufficient to capture many geometrical complexities, but is not fine enough for resolving features such as the temperature gradients within individual fuel pins and pin cladding, turbulent eddies in the liquid sodium flow field, or other small-scale geometric complexities associated with the pumps, heat exchangers, and upper internal structures. Section 3.2 describes the solid modeling and meshing issues associated with the 3D meshes generated for the target ABR design.

Components and physics whose features are too small for resolution on the 3D mesh must be treated in one of two different ways; (1) through a distinct sub-grid-scale model of a geometrical component (such as a fuel pin), or (2) as a closure model to the meso-scale resolved equations (e.g. a turbulent heat transfer or friction factor coefficient). Section 4.4 describes the simple pin model that was written and used for the BRISC prototype, and Section 4.1 describes the fluid flow and heat transfer modeling that accounts for the unresolved effects of turbulence.

The overall reactor system contains many important components outside of the reactor vessel that are fundamental to its operation, performance and safety. These include pipes, pumps, valves, heat exchangers, turbines, containment structures, and so forth. Several different computer codes currently exist which have been developed to treat these "balance-of-plant" components through models that do not resolve their 3D geometry. MELCOR [2] is one example of a code developed at Sandia National Labs that has been specifically developed to address the safety and performance of the overall reactor system under accident conditions. BRISC is designed to couple to a code such as MELCOR through the transfer of heat that occurs through the intermediate heat exchanger. In this strategy, all reactor system processes, physics, and components that exist outside the reactor vessel, including the thermal-hydraulics of the secondary cooling loop, are modeled by a code such as MELCOR. Section 4.5 provides additional details of this approach.

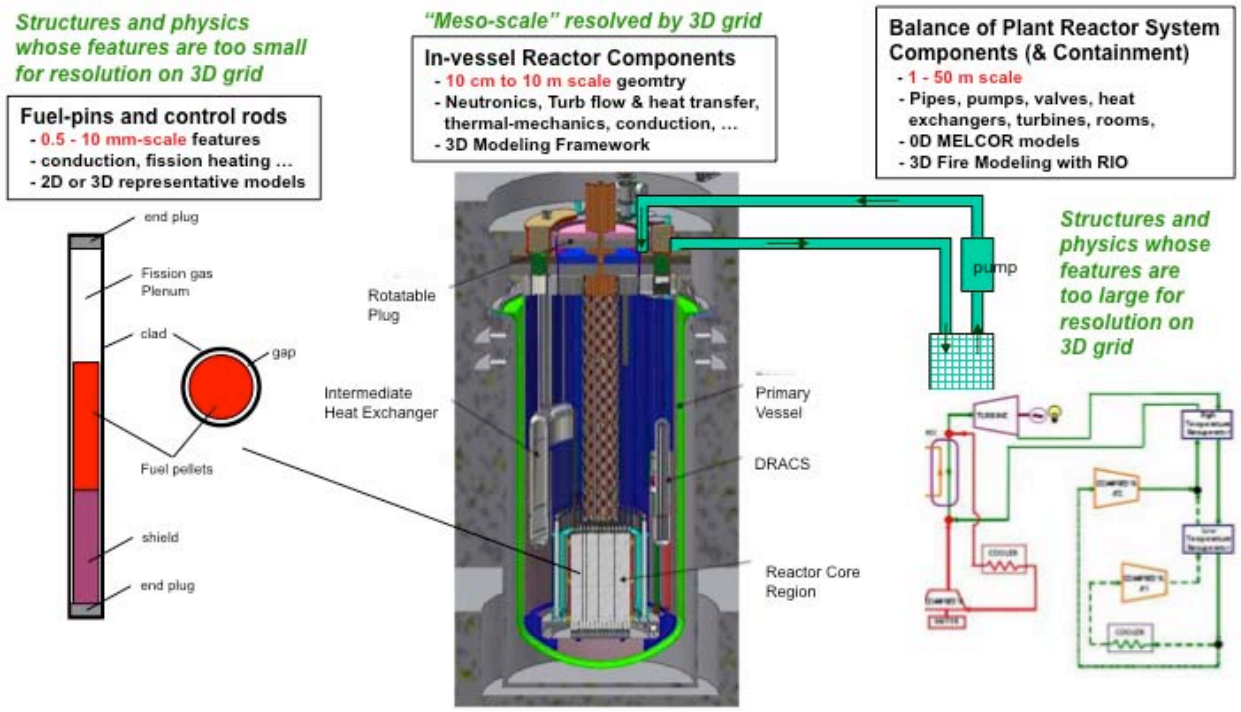


Figure 3.1 Conceptual Illustration of the BRISC 3-level multi-scale modeling strategy (portions of this figure are adapted from Ref. [5])

3.2 Fluid and solid region meshes

From a meso-scale modeling standpoint, we treat the reactor vessel as consisting of two distinctly different physical domains - the fluid domain and the solid domain.

The fluid domain consists of all the physical space where, within the resolution of the meso-scale mesh, fluid flow can occur. The fluid domain includes two-phase regions that contain solid structure not resolved on the meso-scale mesh (such as the fuel pins and control rods within the assemblies) as well as regions that are pure fluid regions. Two-phase (fluid-solid) regions are modeled as non-equilibrium anisotropic porous media through a special set of equations described in Section 4.1.

The solid domain consists of all of the physical space occupied by resolved solid structures. In our model, this includes the space occupied by the reactor vessel itself, the redan, the core barrel, lower plenum structure, and so forth. Since these components are solid, only their thermal-mechanical behavior is modeled.

The geometric union of the solid and fluid domains accounts for all of the space within the reactor vessel. These two regions share a geometrically complex interface surface that marks the boundary between the solid and fluid domains.

Before meshing the solid and fluid regions, solid-geometry models of these two regions were constructed using Pro/ENGINEER (a commercial 3D solid modeling and design tool, [16]) based on the description of the reactor vessel and its components provided in reference [5]. However, since this work was directed towards developing a proto-type code, we did not concern ourselves with generating a perfectly exact representation. For example, when geometric details needed by the solid modeler were not available from the report, we simply used our subjective judgment to guess a reasonable value.

Because of geometrical complexities within the reactor, there are advantages to modeling and meshing the fluid and the solid regions independently. By independent, we mean that the mesh elements on either side of the shared interface are generated independently from one another, and therefore do not conform to one another at the interface. We therefore say that the two meshes are "non-conformal". Note that conformal meshes could be (and were initially) generated. However, doing so led to a dramatic increase in the overall element count as well as to poor element shapes in certain regions. The liability of using non-conformal meshing of the fluid and solid region is that the heat transfer coupling across the fluid-solid interface requires special treatment, including the generation of a special fluid-solid interface mesh. The generation of this mesh is described next in Section 3.3. The modeling of the fluid-solid interface heat transfer is described in Section 4.6.

Figures 3.2 through 3.5 illustrate the level of geometric detail captured in the solid and fluid domain meshes. In each case the geometric model constructed using Pro/ENGINEER is shown first, followed by an all-hex mesh generated using CUBIT[17]. These figures are annotated to identify different in-vessel components and structures that are resolved by the modeling.

As seen in these figures, the mesh shown assumes 1/4 symmetry within the reactor vessel. Outside of the core this is very nearly true in the target ABR design. But within the core, the different types of assemblies are actually arranged in a 1/3 symmetry fashion. To reduce the size of the mesh in this proto-typing exercise, the core region assembly configuration was modified for our BRISC model to accommodate the 1/4 symmetry approximation.

Figure 3.6 shows a cross section of a single fuel-pin assembly, each of which contains 219 individual pins. The mesh we used resolved each assembly cross-section with three elements, as shown in part a. Thus three different subgrid pin models might be associated with each assembly. Part b illustrates that a more refined representation is also possible.

Figure 3.7 shows more clearly the relatively large number of different element blocks that are identified by the modeling within the fluids domain mesh. As mentioned above, the fluids domain consists of many two-phase (fluid-solid) regions that will be treated as a special type of porous media. By identifying these different element blocks, the different physical characteristics associated with these regions can be more easily modeled. For example, the different types of assemblies in the core region (fuel, control, reflector and shield assemblies) must be distinguished and their different vertical sub-regions resolved. A fuel assembly is also divided into five vertical sections to capture the differences in the nosepiece, shield, fuel, gas plenum, and handling socket parts of the assembly.

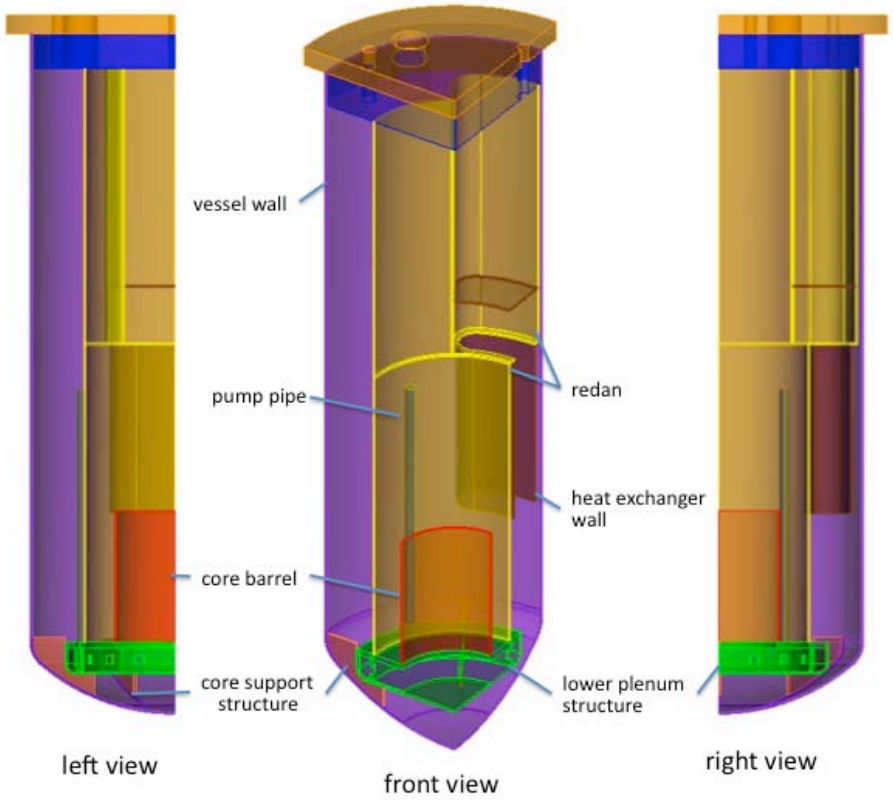


Figure 3.2 Illustration of the solid region structures constructed using Pro/ENGINEER.

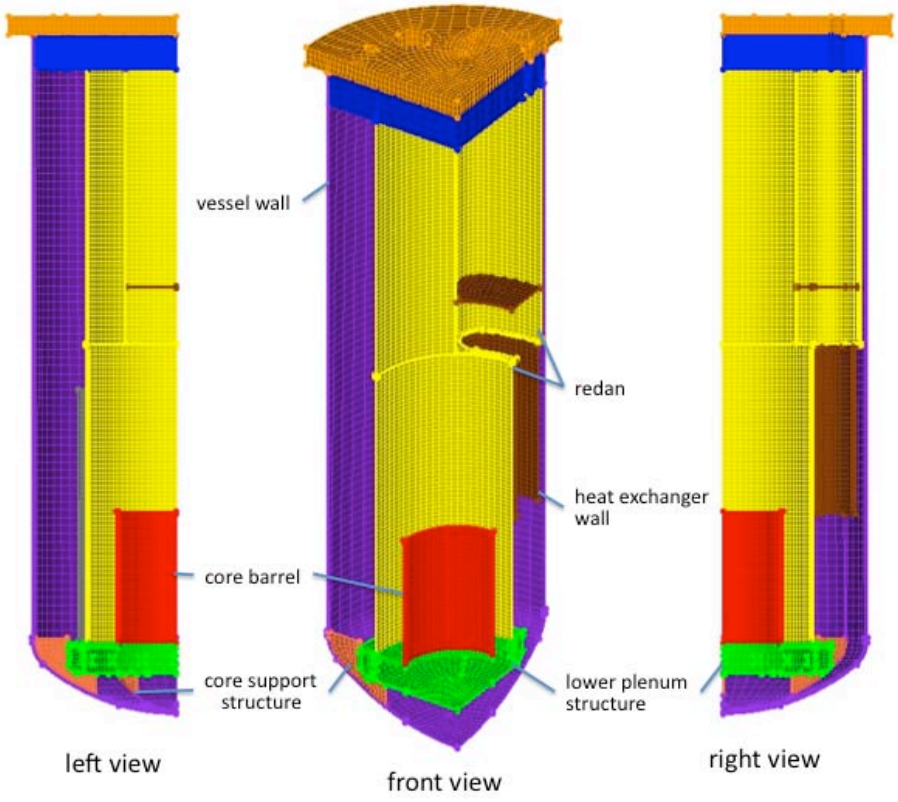


Figure 3.3 Illustration of a solid region mesh generated using CUBIT.

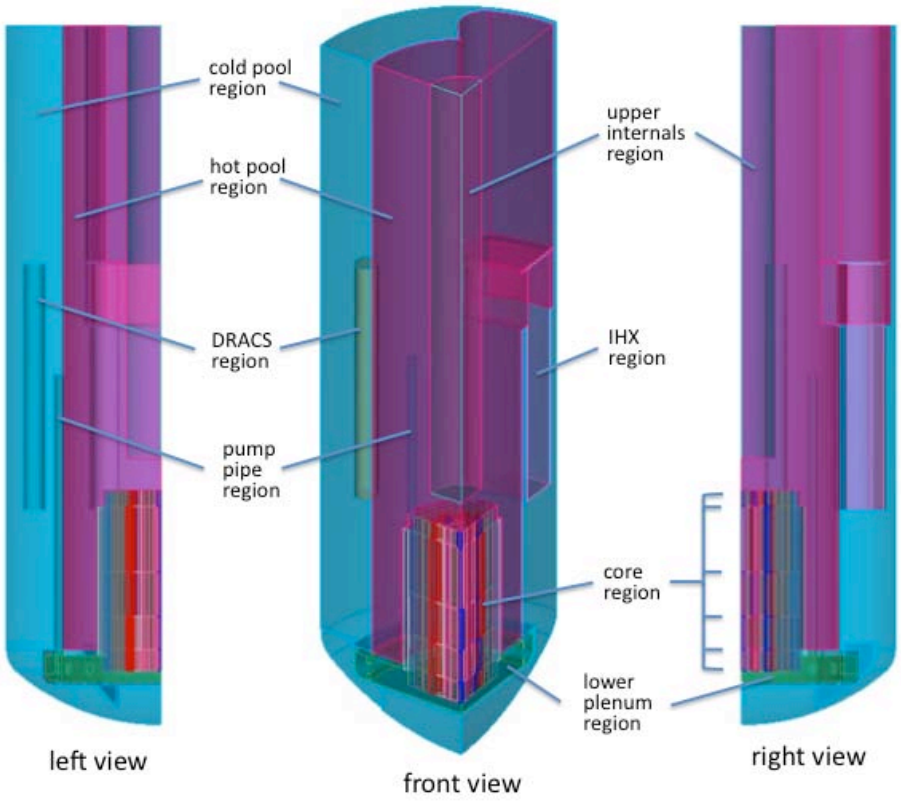


Figure 3.4 Illustration of the fluid region sub-regions constructed using Pro/ENGINEER.

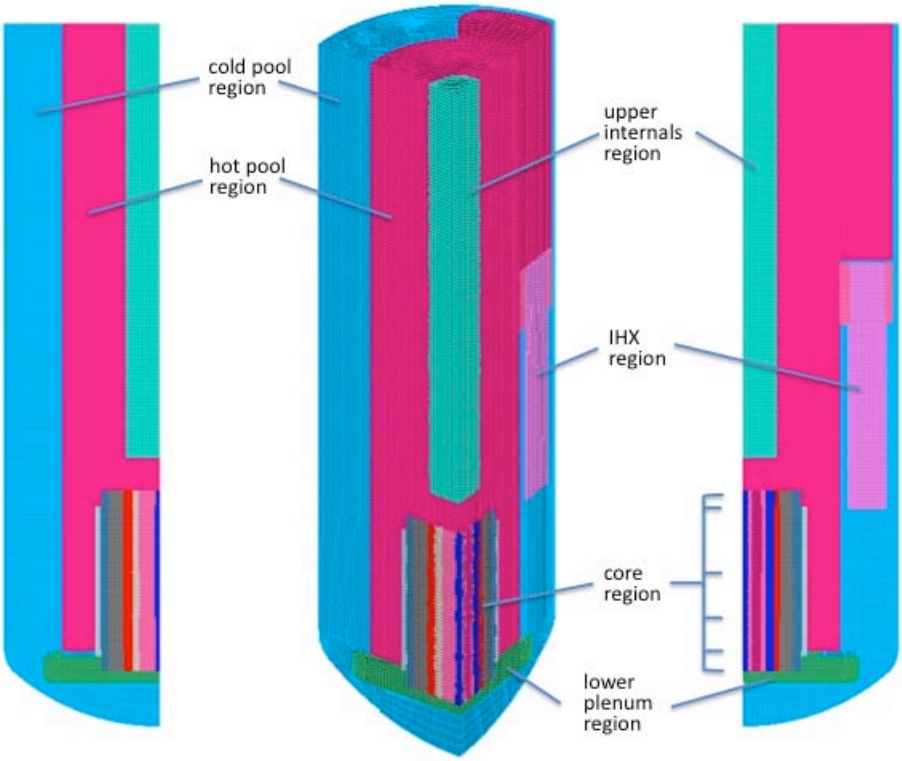


Figure 3.5 Illustration of a fluid region mesh generated using CUBIT.

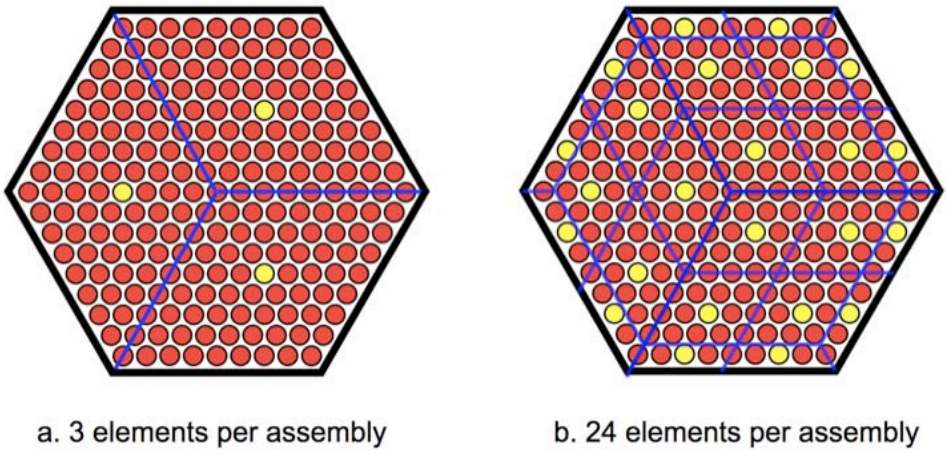


Figure 3.6 Coarse and fine mesh (not applied in our work) resolutions of the hexagon-shaped fuel pin assembly cross-section.

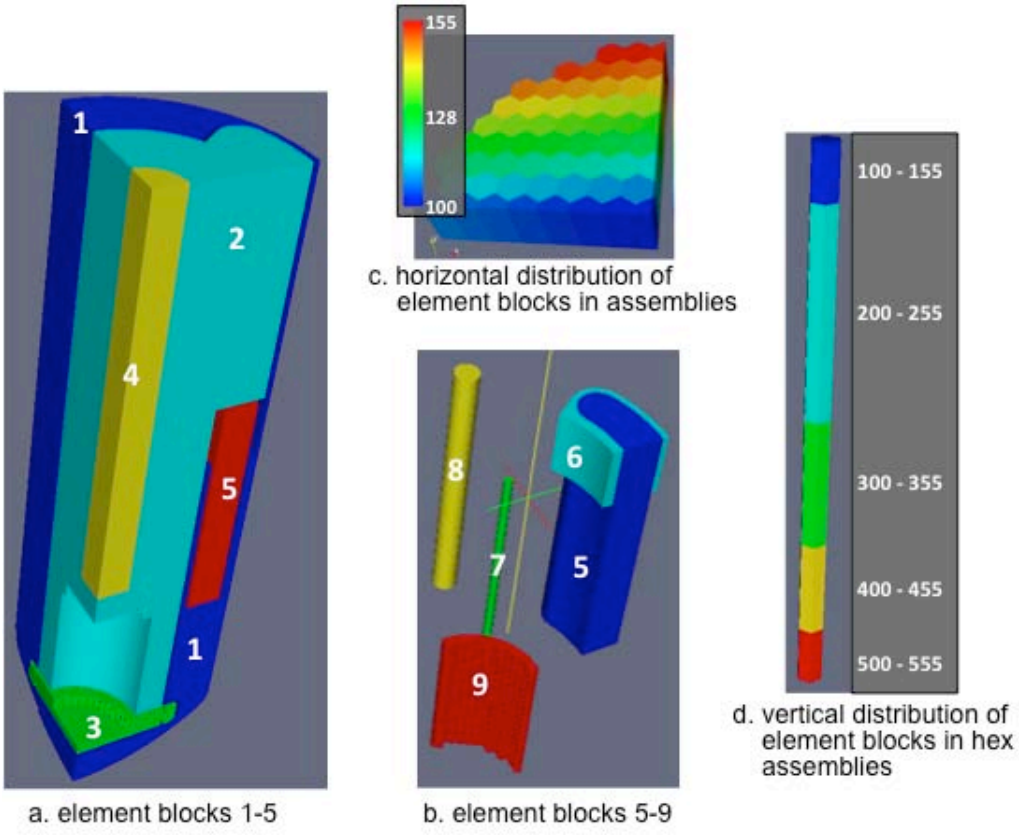


Figure 3.7 Illustration of element blocks defined within the fluid region mesh.

3.3 Creating a fluid-solid interface mesh

Because the fluid and solid meshes are non-conformal, a special interface mesh must be constructed in order to properly transfer heat across the fluid-solid interface. In this section we describe the creation of the fluid-solid interface (FSI) mesh that is required to do this, and will refer to this as either the FSI mesh or the "exchange "surface mesh. Section 4.6 describes the equations solved to perform the heat transfer.

The FSI surface mesh we use is based on the creation of what is called a "common refinement" in the literature. A common refinement of two surface meshes is a mesh composed of polygons that subdivide the polygons of both input meshes simultaneously. Each cell of a common refinement has two geometric realizations, which in general are different but must be close to each other for physically meaningful data transfer.

Figure 3.8 illustrates a simple problem where two physics domains share a common interface.

Figure 3.9 illustrates simple nonconformal volume meshes for these two regions, and shows how an exchange surface mesh is defined. Regions Φ and Ψ are artificially separated in Figure 3.9 so that the concept of an exchange surface mesh can more easily be understood. Based on the volume mesh of region Φ, the shared interface is represented with just two elements S_i^{Φ}, which in a physics code would typically be designated as a side set. In contrast, the volume mesh for region Ψ produces a three-element surface representation S_k^{Ψ}. The exchange surface mesh is formed from the union of these two representations. In Figure 3.9 this consists of the four elements designated S_j^E.

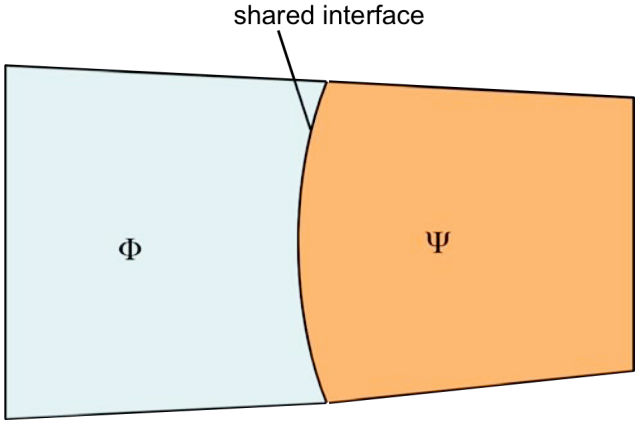


Figure 3.8. Simple domain showing physics regions Φ and Ψ that share an interface.

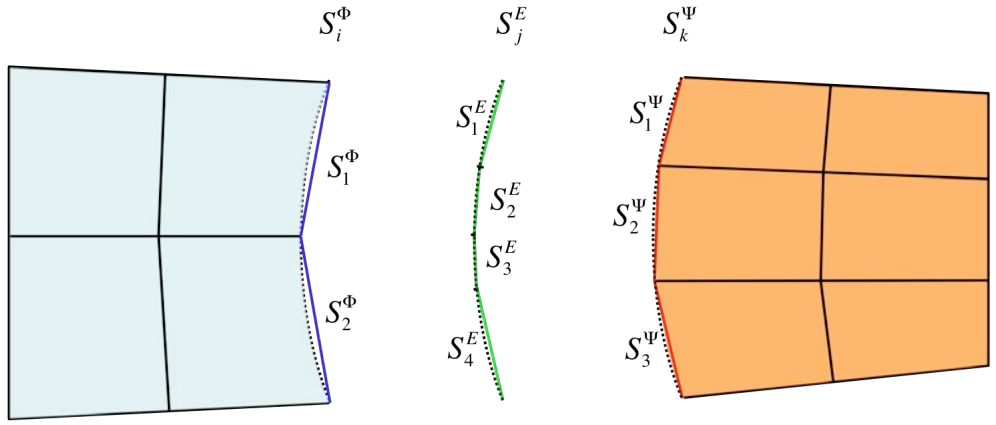


Figure 3.9 Non-conformal meshes for regions **Φ** and **Ψ** showing how an "exchange" surface is defined.

The central tool we use in creating the FSI mesh is a utility code called Surfdiver, developed by Xiangmin Jiao as part of the Center for Simulation of Advanced Rockets (CSAR) effort [18] at the University of Illinois at Urbana-Champaign. CSAR was one of five university-based centers of excellence founded in 1997 and funded by the U.S. Department of Energy's Advanced Simulation and Computing (ASC) program. Surfdiver was developed as a preprocessor of Rocface, a service utility for transferring data between surface meshes in a fashion that is numerically accurate and physically conservative. The purpose of Surfdiver is to generate a common refinement of two surface meshes using a sophisticated algorithm described in References [19] and [20].

Several steps are required in order to generate a FSI mesh file that is useable in a BRISC calculation, and this requires some additional utility operations. These steps are illustrated in Figure 3.10, and described here. The format of the surface mesh files used is Geomview's Object File Format (OFF), which is defined in References [21] and [22].

1. Identify the side-sets in both the fluids-domain exodus mesh and the solid-domain exodus mesh that define the fluid-solid interface.

2. Create an Object File Format file from the fluids-domain side sets (fluidSS.off), and save the Exodus [23] file mapping information as an annotation. This information maps the fluidSS.off file face id to the Exodus file face id and the Exodus file element that this face is a part of.

3. Create an Object File Format file from the solid-domain side sets (solidSS.off), and save the Exodus file mapping information as an annotation. This information maps the

solidSS.off file face id to the Exodus file face id and the Exodus file element that this face is a part of.

4. Run Surfdiver to create fluidSS_sdv.off and solidSS_sdv.off

5. Run the Merge_SDV_Code utility to

   (a) Combine the information in fluidSS_sdv.off and solidSS_sdv.off,
   (b) merge unneeded triangles into quads,
   (c) add the Exodus file mapping information contained in fluidSS.off and solidSS.off, and
   (d) save this information in a file called briscFSI.off

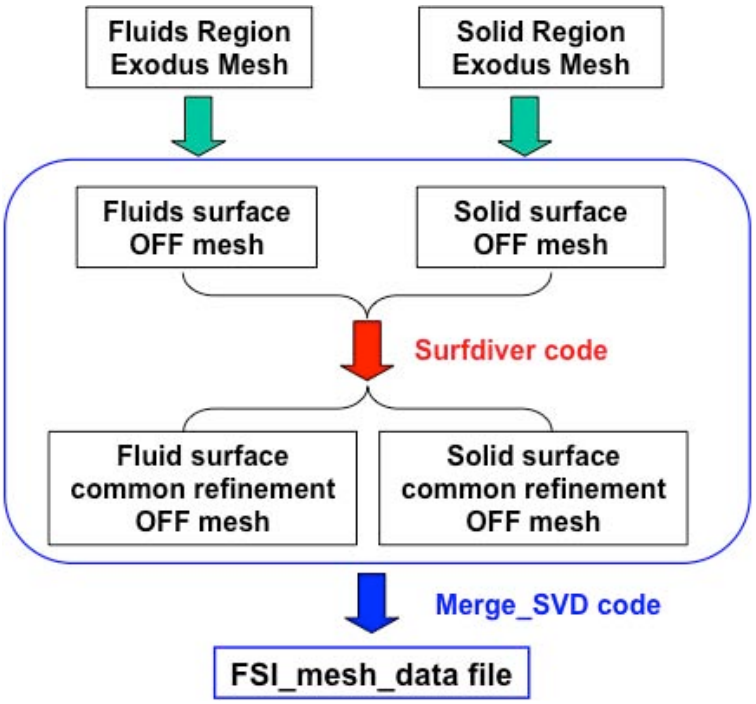


Figure 3.10 Steps to creating a fluid-solid interface mesh file (FSI_mesh_data) for use in BRISC

Steps 2 and 3 are accomplished using a small utility code, exodus_to_off (exodus_to_off.cpp).

Each exodus_to_off invocation requires the name of the Exodus mesh file and a list of side set ids defining the common interface surface in that mesh. The Exodus mesh is read and stored in memory so that the element face and the face vertex connectivity of the interface surface are readily available. For each side set id in the interface mesh, we generate (a) the unique list of elements connected to the faces, (b) the ordered set of vertex ids corresponding to each face, (c) the local face ordinal (index of the face using Exodus face ordering), and (d) the coordinates of each vertex on the face. With this information we write the OFF file.

The OFF file consists of two sections, the vertex coordinates and the interaction faces. The vertex coordinates in the first section are indexed using zero-based indices (Exodus uses ones-based indexing). The interaction face vertex connectivity is written to the second section where the vertex ids have been converted to zero-based indices. An end-of-line comment follows each interaction face description. End-of-line comments begin with a #, and follow the face vertex connectivity information. The comment section contains (a) the Exodus element id containing this face, (b) the Exodus local face ordinal, and (c) the side set id this face belongs to. Information in the end of line comment is used in the merge code to provide a bidirectional mapping between OFF and Exodus mesh entities.

The name of the OFF output file is the same as the input file replacing the Exodus suffix (*.g) with the OFF suffix (*.off).

Step 5 requires the use of a separate utility code called Merge_SDV (merge_sdv.cpp) that was written to post-process the Surfdiver output files into a form usable by BRISC.

The Merge_SDV utility expects two Surfdiver output filenames, representing the common refinement of the solid and fluid interface meshes. These OFF files are combined with the original OFF files input into Surfdiver. After MergeSDV has reduced the mesh size through merging unneeded triangles, a file called FSI_mesh_data is written. This fluid-solid-interface mesh file contains two lists of faces, the fluid faces and the solid faces, along with an index that maps from a fluid face to the index of the same face in the solid mesh. This mapping is essential for transferring quantities across this face. Other face data includes the face area, as well as the corresponding Exodus element, side set id, and face ordinal information. A number of consistency checks were added to the FSI model evaluator to ensure that the solid and fluid meshes were consistent and that they mapped back to the original Exodus mesh as used by the two instances of Rio.

The following is the listing of an example script file demonstrating the steps involved in producing OFF files used in BRISC.

```
# Fluid mesh side sets are 5..15
../multiphysics/c++_driver/exodus_to_off.x fluid.g 5..15 > fluid.out

# Solid mesh side sets are 5..14
../multiphysics/c++_driver/exodus_to_off.x solid.g 5..14 > solid.out

# Run Surfdiver (fluid mesh must precede solid mesh or we tickle bug in Surfdiver)
../Surfdiver/bin/surfdiver fluid.off solid.off

# Run the merge code
../multiphysics/c++_driver/merge_sdv.x fluid.off solid.off
```

Figure 3.11 shows an the fluid and solid domain surface elements on the lower plenum surface compared to the common refinement mesh generated by Surfdiver

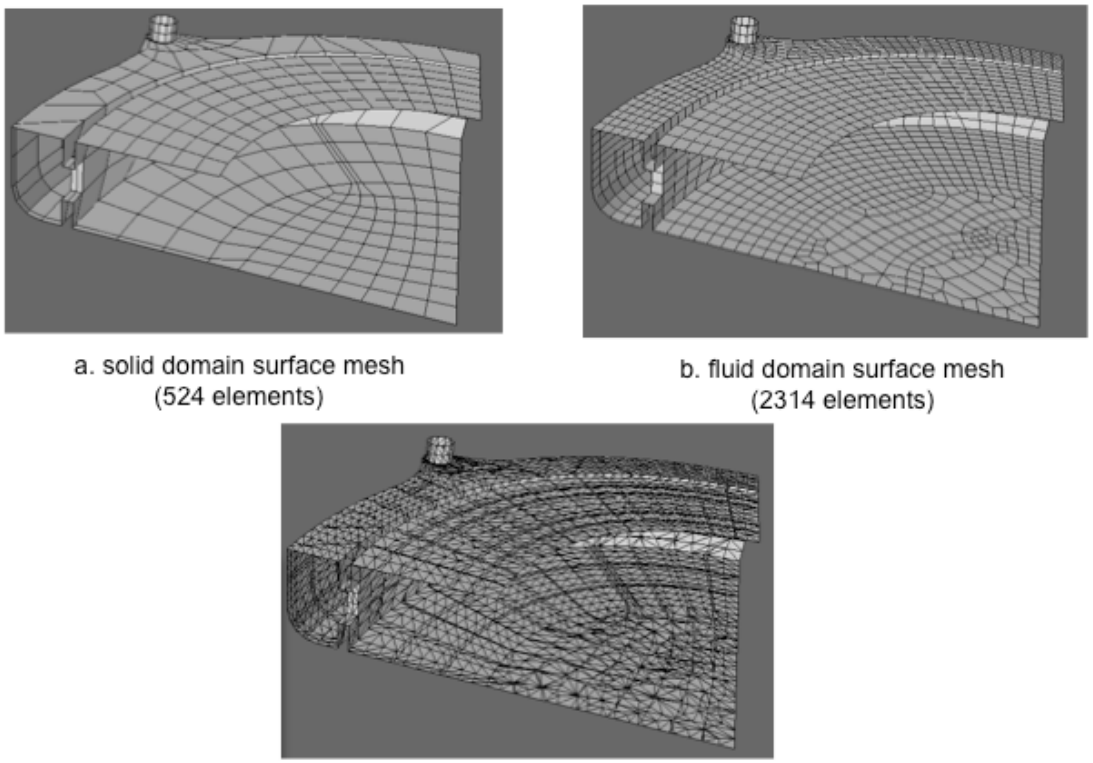


Figure 3.11 Fluid and solid domain surface elements on the lower plenum surface compared to the common refinement mesh generated by Surfdiver.

4. PHYSICS CODES AND MODELS

To computationally simulate the system response of a sodium-cooled fast reactor, a wide range of physical processes and system components must be modeled. An important step in developing any simulation code is to clearly identify those physical phenomena which are most important to the problems of interest to the user of the code. In our work, the target code application is to perform systems-level safety analysis, such as might be required by the Nuclear Regulatory Commission as part of a safety analysis report. Therefore, one of the first activities of this LDRD project was to gather a group of reactor safety experts for the purpose of defining and discussing the physics and physical processes important for the safety analysis of future advanced burner reactors (ABRs). An important result of this meeting was the creation of a preliminary set of tables, similar to what are sometimes called "Phenomena/Process Identification and Ranking Tables" (PIRT), in which the important physical processes and physics that were identified are listed, together with some assessment as to their level of importance and the adequacy of current understanding and modeling. Appendix C summarizes key results and comments from this meeting and documents the tables generated in these discussions

As part of the PIRT meeting, various alternative ways of categorizing the different phenomena were considered, with the following five categories finally deemed adequate for our purposes.

1 Neutronics
2 Fluid Flow and Heat Transport
3 Material Interactions & Chemistry
4 Material Properties and Equation of State
5 Structural Mechanics and Dynamics, Relocation

It was noted that these processes are not independent and that strong coupling may exist between phenomena listed in different groups.

As the tables were generated, the importance ranking was also separated into Phase 1 - "initial transient" and Phase 2 - "damage transient" periods. These two phases were defined as follows:

> Phase 1 - Initial transient: Begins at the accident initiating event and includes the time period when the state of the system moves from its normal operating state to a point where the structural geometry or material phases of system components begin to change.
>
> Phase 2 - Damage transient: Extends from the point when structural or material changes begin to occur until the end of the accident.

To limit the work scope suitably for an LDRD project, it was subsequently decided that the BRISC effort would only attempt to address the modeling issues needed for Phase 1 physics. In addition, one particular accident transient, typically referred to as an unprotected loss-of-flow (ULOF) accident, was chosen to further focus the LDRD development effort. In a ULOF accident, the plant experiences a complete loss of power (including emergency power supplies) to the reactor cooling system while the plant is at full power. The power generation plant immediately ceases operation and provides no heat rejection capacity. Finally, one assumes complete failure of the reactor safety system to scram the reactor and no operator actions to mitigate any aspect of the accident.

The subsections that comprise the remaining parts of this chapter describe the physics models and codes that were chosen to account for the important physical processes determined as described above. The codes chosen, and the models implemented were chosen as representative of approaches that might be taken. By representative we mean that they address the proper physics in a reasonable way that might be chosen if a production code was to be developed. In addition to model fidelity, factors such as code usability, flexibility, and source code availability also affected the choice of codes and models incorporated into the BRISC prototype code.

Subsection 4.1 through 4.6 describe the models and codes used in BRISC for

- in-vessel fluid flow and heat transfer,
- thermal conduction through solid structures,
- neutronics,
- a sub-grid scale fuel pin,
- ex-vessel and balance of plant components and systems, and
- heat-transfer across a fluid-solid interface with non-conformal meshes.

We note here that thermal expansion (or more generally thermal-mechanics) is an important physical process in fast reactor safety analysis that is not separately modeled in the BRISC prototype. Instead, the effects of thermal expansion are incorporated into the neutronics modeling indirectly. A discussion of thermal-mechanical effects and their strong coupling to the neutronics model is included Section 4.3.

4.1 In-vessel fluid flow and heat transfer

Compared to light water reactors, fluid flow and heat transfer modeling in a sodium-cooled fast reactor has an important simplification – under normal operating conditions and throughout the initial transient of any accident simulation liquid sodium can be treated as a single-phase fluid. However, compared to water liquid sodium has a much higher thermal conductivity (~100x), a slightly lower density and a much lower specific heat (~1/4). The Prandtl number for liquid sodium is thus of order several hundred times smaller than that of water.

As illustrated in Section 1, the in-vessel domain through which the sodium coolant flows is geometrically complex. It includes large open areas, convoluted flow paths through pumps, pipes, and orifices, as well as small restricted regions in the reactor core assemblies and in the primary and auxiliary heat exchangers. In general, the flow will be turbulent and both forced and free convection flow regimes must be modeled. Because the local flow Reynolds numbers can be very high, resolving the flow field sufficiently for direct numerical simulation (DNS) of the entire system would require computational capabilities that are literally millions of times beyond those expected to be available in the next ten to fifteen years. Therefore, the equation sets targeted for use in BRISC are not “first-principle” equations, but rely on empirical coefficients associated with various simplified models to represent the turbulent processes that are not resolved by the numerical mesh. This compromise is unavoidable given the physical length and time scales that otherwise must be resolved.

Many codes for modeling fluid flow and heat transfer were considered as candidates for use in BRISC. A Sandia-developed code called RIO [24] was chosen as the best option for our LDRD needs. In addition to the turbulent flow and heat transfer models already built into the code, key factors that made RIO a good choice for this LDRD included

(1) a modern code architecture leveraging advanced parallel solver libraries and unstructured meshing technologies,
(2) a code design that includes optional user-written subroutines for incorporating a wide range of customized models,
(3) access to and freedom to modify the source code if needed,
(4) excellent code documentation, and
(5) the ability to easily consult with the primary code developer at Sandia.

Section 4.1.1 provides an overview and additional details about the RIO code.

As part of this LDRD, the RIO code was extended in several ways and additional models incorporated into the source code. Section 4.1.2 describes a non-equilibrium anisotropic model for conjugate fluid/porous/solid domains that was used in RIO/BRISC to model the in-vessel flow and heat transfer in a sodium-cooled fast reactor.

4.1.1 RIO

The RIO code is an unstructured finite volume code for solving the low Mach number Navier-Stokes equations with heat and mass transfer. Extensive documentation of RIO is provided in reference [24]. In RIO, a large set of configurable math models are available ranging from laminar flow with detailed transport and chemistry to turbulent transport. Turbulence models are primarily based on the Reynolds averaging approach, with a simple LES model also available. To address problems with large variations in density, mass-weighted Favre-averaging is used to derive math models. The equations are discretized using a cell-centered approach and solved using a segregated, approximate projection method for parallel computing using domain-decomposition and message-passing.

The low Mach number equations are a subset of the compressible Navier-Stokes (and continuity and energy) equations, admitting large variations in fluid density while remaining acoustically incompressible. The low Mach number equations are used to avoid resolving fast-moving acoustic signals which have no bearing on the transport processes. The equations are derived from the compressible equations using a perturbation expansion in terms of the lower limit of the Mach number squared; hence the name. The asymptotic expansion leads to a splitting of pressure into a spatially constant thermodynamic pressure and a locally varying mechanical pressure. The mechanical pressure is decoupled from the thermodynamic state and cannot propagate acoustic waves. The thermodynamic pressure is used in the equation of state and to determine thermophysical properties. The thermodynamic pressure can vary in time and can be calculated using a global energy balance.

Two time integration schemes are currently available in RIO - first-order backward-Euler or second-order three-point backward differencing with implicit treatment for convected values and diffusion processes. All math models are cast in conservative integral form and mass is balanced across each control volume. The transport equations are advanced in time by solving the discrete, nonlinear equations for the given time step. In stand-alone RIO calculations, Picard looping may be employed over the nonlinear step to converge the equations as well as Picard looping over individual equations within the nonlinear solve. When incorporated into BRISC in this LDRD, additional non-linear solutions strategies where made available through the LIME multi-physics environment that links to Trilinos[11, 12].

The RIO code was originally developed as part of a study of alternative methods to the vertex-centered, control volume finite element method used in the SIERRA/FUEGO code [25, 26]. At present the RIO project lives on at Sandia as a platform for prototyping numerical methods and testing advanced math models, including multiphase flow using the multifluid approach, turbulent mixed convection using algebraic flux models, electrokinetic transport, and for coupling with a network solver [27, 28].

RIO is designed to take advantage of distributed, massively parallel computing via domain decomposition and message passing. Message passing uses the MPI-1 API [29]. The mesh domain is subdivided along element boundaries. The ghost element concept is used to construct fluxes at processor boundaries. The ghost elements are one layer deep, providing nearest-neighbor support.

The RIO code is linked with several third-party libraries for functionality such as domain-decomposition management, parallel communication, linear solvers, and thermophysical properties in addition to the ubiquitous BLAS [30], LINPACK [31], and LAPACK [32] math libraries.

A powerful characteristic of the RIO code is its use of “user-subroutines” that enable a great deal of flexibility in creating customized models and problem definitions. Through the user controlled input, RIO can call user-subroutines after initialization and both before and after time steps and non-linear iterations. User-subroutines may be written to calculate values for math model source terms, initial conditions, thermophysical properties, and boundary condition values, and are typically used to perform calculations that depend on geometric location. For example, Figure 3.7 (see Section 3) illustrates how several hundred different element blocks are defined in the 3D fluids region mesh created to model the ABTR. Problem specific customized user routines enable different element blocks to have completely different models for each of these element block regions.

4.1.2 Non-equilibrium Anisotropic Model for Porous Fluid-Solid Domains

As part of this LDRD, the RIO code was modified to treat non-equilibrium anisotropic flow through porous fluid-solid domains. The modeling approach is based on the Brinkman-Forchheimer form of the conservation equations. These equations are designed to model conjugate fluid flow and heat transfer in domains consisting of pure fluid, porous, and pure solid

regions and have seen application in a wide range of engineering applications. References [33] through [39] are a selected sample of papers in the literature that discuss the theory, development, and use of these model equations, including their application in nuclear reactor safety modeling.

This section begins by defining the different types of volume averaging that can be used, and then describes the equations and models implemented in this modeling approach.

4.1.2.1 Volume Averaging

In defining equations for porous media the type of volume averaging applied must be carefully defined. As described by Betchen et al. [33], three different types of volume averages can be defined in a porous region. These are defined in this subsection.

We begin by defining a control volume of total volume V potentially occupied by multiple constituents. The volume occupied by constituent s is denoted V_s, and a generic property ψ associated with constituent s, is denoted ψ_s.

Three distinctly different averages of ψ can be defined as follows.

$$\langle \psi_s \rangle = \frac{1}{V} \int_{V_s} \psi_s dV \qquad \text{extrinsic average} \tag{4.1.1}$$

$$\langle \psi_s \rangle^s = \frac{1}{V_s} \int_{V_s} \psi_s dV \qquad \text{intrinsic average} \tag{4.1.2}$$

$$\langle \psi \rangle = \overline{\psi} = \frac{1}{V} \int_{V} \psi \, dV \qquad \text{total average} \tag{4.1.3}$$

Now consider a porous medium domain where the porosity φ is defined as the volume fraction of the “open” or “porous” region within the domain is saturated with a constituent fluid (denoted here with subscript f) exists.

$$\varphi = \frac{V_f}{V} \tag{4.1.4}$$

Therefore, in a porous medium - which consists of fluid and solid material, the extrinsic and intrinsic averages defined above are related as follows

$$\langle \psi_f \rangle = \varphi \langle \psi_f \rangle^f \tag{4.1.5}$$

4.1.2.1 Equation set

In the equations that follow all three types of averaging described above are applied. For convenience, we drop the angle bracket notation used above and define the following rules to be applied in our notation.

* All fluid velocities and the fluid pressure are extrinsic averages. (The velocities can be equivalently defined as total averages because the solid velocities are identically zero.)
* All material properties are intrinsic averages where the subscripts “f” and “s” will denote the “fluid” and “solid” phases of the porous medium.
* The porosity is a total average.

In addition, although the equations are written in continuum fashion with differential operators, it is to be implicitly recognized that these equations are solved here in a discrete finite-volume formulation.

Under reactor conditions where sodium will not vaporize, the in-vessel fluid-filled regions below the sodium pool height can be treated as a composite porous-media/clear fluid region and the Brinkman-Forchheimer form of the conservation equations used (See [33] - [39]) These equations take the following form.

Fluid Flow Continuity Equation**:**

$$\frac{\partial \rho_f}{\partial t} + \frac{\partial}{\partial x_j}\left(\frac{\rho_f u_j}{\varphi}\right) + \dot{\rho}_{f,src} = 0 \tag{4.1.6}$$

Brinkmann Forchheimer form of the Fluid Flow Momentum Equation:

$$\frac{\partial \rho_f u_i}{\partial t} + \frac{\partial}{\partial x_j}\left(\frac{\rho_f u_j u_i}{\varphi}\right) = -\frac{\partial P}{\partial x_j} + \frac{\partial}{\partial x_j}(\tau_{ij}) - \left(\frac{\varphi \mu_f}{K} + \rho_f \frac{F\varphi}{K^{1/2}}|u|\right)u_j + \varphi \rho_f g_i + (\dot{\rho}_{f,src} u_{i,src}) \tag{4.1.7}$$

where

K is the permeability,

F is the Forchheimer parameter,

$$\tau_{ij} = \mu_{f,eff}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right),$$

and the effective fluid viscosity $\mu_{f,eff}$ may include turbulence effects.

Energy Equation:

Two cases are described here.

Case 1: Thermal non-equilibrium

Fluid Phase

$$\frac{\partial}{\partial t}\left(\rho_{\mathrm{f}} \mathrm{C}_{\mathrm{f}} T_{\mathrm{f}}\right)+\frac{\partial}{\partial x_j}\left(\frac{\rho_{\mathrm{f}} u_j \mathrm{C}_{\mathrm{f}} T_{\mathrm{f}}}{\varphi}\right)=\frac{\partial}{\partial x_j}\left(k_{f,eff} \frac{\partial T_{\mathrm{f}}}{\partial x_j}\right)+\dot{Q}_{\mathrm{f}}+\frac{A_{\mathrm{fs}}}{\varphi} h_{\mathrm{fs}}\left(T_{\mathrm{s}}-T_{\mathrm{f}}\right)+\dot{\rho}_{\mathrm{f,src}} \mathrm{C}_{\mathrm{f,src}} T_{\mathrm{src}} \tag{4.1.8}$$

Solid Phase

$$\frac{\partial}{\partial t}\left(\rho_{\mathrm{s}} \mathrm{C}_{\mathrm{s}} T_{\mathrm{s}}\right)=\frac{\partial}{\partial x_j}\left(k_{s,eff} \frac{\partial T_{\mathrm{s}}}{\partial x_j}\right)+\dot{Q}_{\mathrm{s}}-\frac{A_{\mathrm{fs}} h_{\mathrm{fs}}}{(1-\varphi)}\left(T_{\mathrm{s}}-T_{\mathrm{f}}\right) \tag{4.1.9}$$

Note that in Equation (4.1.9) only one average temperature for the solid is resolved within the control volume. This model can be enhanced by the incorporation of a sub-grid sub-component model (e.g. fuel rods, control rods, and canister walls) that enables small-scale temperature variations to be resolved. Section 4.4 describes a sub-grid scale pin model that can be applied in this manner, and Section 5.4 discusses the test problems where the model is actually used.

Case 2: Thermal equilibrium If local thermal equilibrium is assumed, then the two energy equations shown can be collapsed into one. For this case, we use the over-bar to denote a total average property that includes both the fluid and solid contribution.

$$\frac{\partial(\overline{\rho}\overline{C}\overline{T})}{\partial t}+\frac{\partial}{\partial x_j}(\rho_f u_j C_f \overline{T})=-\frac{\partial}{\partial x_j}\left(\overline{k}_{eff} \frac{\partial \overline{T}}{\partial x_j}\right)+\overline{\dot{Q}}+\varphi \dot{\rho}_{\mathrm{f,src}} \mathrm{C}_{\mathrm{f,src}} T_{\mathrm{src}} \tag{4.1.10}$$

Here, $\overline{k}_{eff}$ is the effective thermal conductivity of the porous medium that may be modeled to include the effects of thermal dispersion, turbulence, etc.

Required properties, parameters or closure models include:

μ_{eff}	effective viscosity (may include turbulence and other modeled effects) (kg/ m s)
$k_{f,eff}$	effective fluid thermal conductivity (includes turbulence, other modeled effects)
$k_{s,eff}$	effective solid thermal conductivity (may include special modeled effects) (W/m K)
F	the quadratic (Forchheimer) drag coefficient (non dimensional)
K	permeability of the solid matrix (m^2)
A_{fs}	specific interfacial area of the fluid-solid (1/m)
h_{fs}	fluid-solid heat transfer coefficient (W / m^2 K)
$\dot{Q}_f$	heat source in the fluid $(W/m^3)_f$
$\dot{Q}_s$	heat source in the solid $(W/m^3)_s$
$\dot{\rho}_{src}$	mass source per unit volume (kg/m^3 s) (e.g. in/out flow to/from a pipe)
$u_{i,src}$	velocity of component i associated with the mass source (m/s)
T_{src}	temperature of the mass source fluid (K)
C_{src}	specific heat of the mass source fluid (J/ kg K)

4.2 Thermal conduction through solid structures

Figures 3.2 and 3.3 illustrate large-scale solid region structures that are resolved in a BRISC representation of a nuclear reactor. Currently, conduction heat transfer is the only physical process modeled by BRISC in these structures, although a more general treatment of thermal mechanics (which would include thermal expansion effects) is a logical extension.

The equation governing transient thermal conduction through a solid structure is well known. It can be written in fairly general form as

$$\frac{\partial}{\partial t}\left(\rho_s C_s T\right) = \frac{\partial}{\partial x_j}\left(k_s \frac{\partial T}{\partial x_j}\right) + \dot{Q}_s \qquad (4.2.1)$$

where ρ_s is the density, C_s is the specific heat, T is the temperature, k_s is the thermal conductivity, and $\dot{Q}_s$ is a local heat source.

In BRISC, a separate instantiation of the RIO code (see section 4.1.1) is used to solve the heat conduction equation in the solid region. Through the use of RIO user subroutines, each of the material properties can be specified as functions of spatial location and local temperature. RIO solves the conduction problem on an unstructured discrete mesh defined in a standard EXODUS file format.

To solve Equation (4.2.1), initial and boundary conditions must also be specified. In RIO, a variety of boundary condition options can be applied, including specifying the heat flux or the surface temperature as functions of spatial location and time. This capability is used in doing BRISC nuclear reactor simulations so that heat transfer across the fluid-solid interface is consistent, conservative, and fully coupled. The fluid-solid interface (FSI) modeling required for this is described in Section 4.6.

4.3 Neutronics

In BRISC the required output from the neutronics calculation is the spatial distribution of heat sources within the reactor. In general, this requires knowledge of the local neutron density (or alternatively, the local neutron flux) – a potentially difficult problem for which many different methods, models and codes have been developed over the years. In this LDRD, only the relatively simple "Point-kinetics" type models (defined below) have been applied. However, a generalized interface to the multi-physics driver has been designed to accommodate higher fidelity models of greater complexity. As part of a staged development plan, progress was also made towards the incorporation of results from a 2D finite-difference diffusion model for cross-section changes to enhance the fidelity of the point kinetics solution in time. This was intended

to set the stage for incorporating even more accurate codes and models, such as the SCALE(TRITON) [40, 41] and NESTLE [42] codes being developed at ORNL.

This section is organized as follows. Subsection 4.3.1 briefly reviews basic concepts, terms and theory concerning reactor neutronics. Next, subsection 4.3.2 provides an introduction to some important attributes of liquid-metal reactor kinetics. Subsections 4.3.3 through 4.3.5 describe the modeling approaches and codes that were implemented, developed, or planned as part of the LDRD effort. Finally, subsection 4.3.6 briefly describes the proto-type of a generalized interface developed to couple the neutronics calculations to the multi-physics driver in BRISC.

4.3.1 Basic concepts and terms

When a balance exists between the number of neutrons produced in a reactor and those lost by absorption or leakage, the reactor is said to be "***critical***", and its power output is constant in time. The term "***multiplication factor***" (typically denoted by the symbol k) is used in this context to mean the ratio of the number of fission neutrons in one generation divided by the number of fission neutrons in the preceding generation. An important related term is the ***reactivity*** ρ, which is defined in terms of k as follows;

$$\rho = \frac{k-1}{k}. \tag{4.3.1}$$

By definition, a value of k equal to 1 ($\rho = 0$) corresponds to a reactor that is critical. If k is less than 1, the number of neutrons is decreasing in time and the reactor is said to be ***subcritical*** and have negative reactivity. If k is greater than 1, the number of neutrons in increasing in time and the reactor is said to be ***supercritical*** and have positive reactivity.

Although the steady-state neutron flux distribution in a critical reactor is an important problem, simulating the response of a reactor during accident scenarios also requires the ability to predict the transient distribution of the neutron flux for situations where k does not equal 1. The terms "***reactor kinetics***" and "***neutron dynamics***" refer to the study of the transient neutron distribution in a noncritical reactor. Because changes in temperature can have a major impact on k, a reactor kinetics problem is usually strongly coupled to the problem of calculating the coolant flow and heat transfer within the reactor.

Reactor kinetics problems are often classified into three time-scale groups – long, intermediate and short. In this context a long-time problem is of order days to weeks, an intermediate problem of order hours to days, and a short-time problem is anything less than order tens of minutes. Different methods are typically used for solving these different classes of problem. Safety calculations associated with reactor accidents are most commonly short to intermediate time transients.

Because the behavior of individual neutrons and nuclei cannot be predicted, the object of practical simulations is to compute the average behavior of a large population of neutrons. Specifically, we need to know the space, energy, and time distribution of neutrons in a complex heterogeneous media (i.e. the reactor). Formulating this problem usually begins with what is

called the ***Boltzmann transport equation*** – derived using the theory of statistical mechanics. The Boltzmann transport equation is basically a "balance" statement that accounts for additions to and subtractions from the neutron density in a given increment of space, energy, angular direction and time. Different types of neutron–matter interactions must all be accounted for, including processes such as scattering, radiative capture, absorption, and fission. In-depth discussions of this equation and its derivation can be found in many textbooks on nuclear physics and/or engineering, and various forms of the equation can be used depending on the importance of different terms to a particular problem of interest.

The extent to which neutrons interact with the nuclei of the surrounding matter is expressed in terms of quantities known as ***cross sections***. Two classes of cross sections are defined – microscopic cross sections (typically denoted by σ), and macroscopic cross sections (typically denoted by Σ).

σ_x = a characteristic area proportional to the probability that an event of type "x" will occur (cm^2)

Σ_x = probability per unit path length that an event of type "x" will occur
= σ_x (cm^2) times atom number density ($atoms/cm^3$) = events/cm

All neutron cross sections are functions of the energy of the incident neutrons and the nature of the interacting nucleus. To solve the Boltzmann transport equation the cross sections for each interaction of importance must be known over the range of energies important to the problem. Figure 4.3.1 illustrates how complicated this functional dependence can be by showing a plot of the U238 capture cross section. As can be seen (note the log scale), fully resolving the functional dependencies of cross sections with energy in numerical simulations adds a tremendous amount of additional complexity and cost to a potential calculation.

The Boltzmann transport equation cannot be solved analytically unless many simplifying assumptions are made. However, both stochastic (Monte-Carlo) and deterministic numerical techniques are now available that are capable of providing solutions of excellent accuracy. Unfortunately, the computational expense, even with today's modern computers, is often prohibitive for engineering types of calculations.

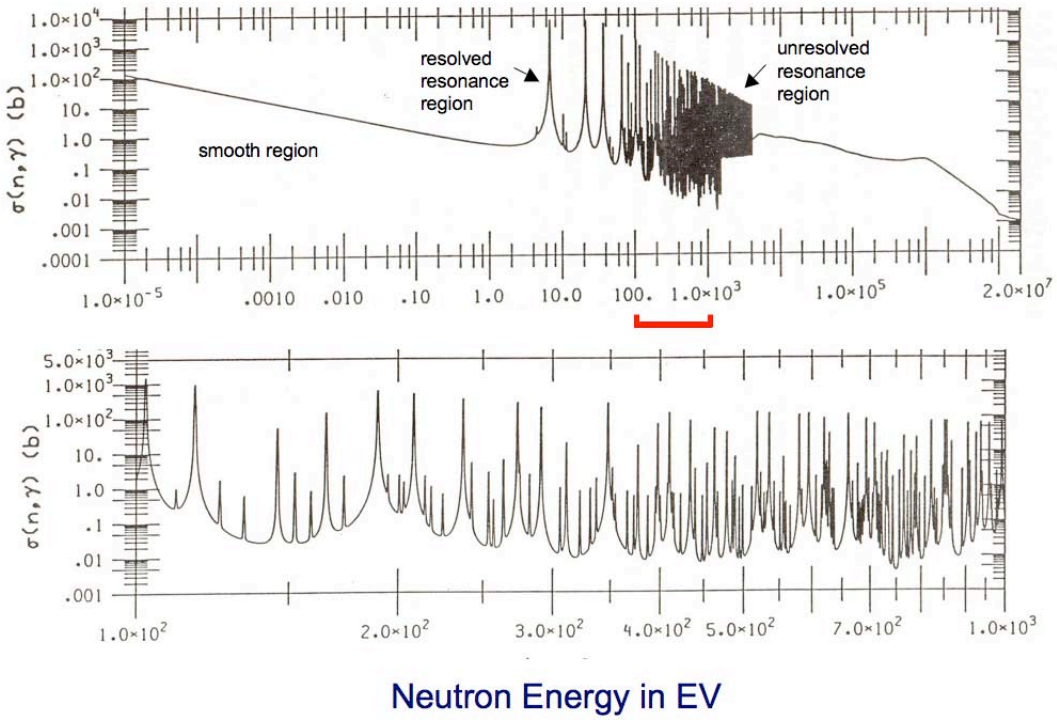


Figure 4.3.1 Plot of U238 microscopic capture cross-section as a function of energy

A useful approach that allows for a large reduction in complexity is to treat the average particle transport process with what is called the ***diffusion approximation***. This approach models the net current of neutrons as proportional to the gradient of the neutron flux. In one dimension, this relationship can be written as

$$J_x = -D\frac{\partial \phi}{\partial x} \tag{4.3.2}$$

where J_x is the net number of neutrons that pass per unit time through a unit area perpendicular to the x-direction, J_x has the same units as flux, i.e. neutrons/cm^2-sec, and D is the diffusion coefficient with units of cm. Note that in this approach, the diffusion coefficient becomes a very important property of the medium and can be a function of many different variables.

Although important limitations are associated with this approach, it is widely applied to provide reasonable estimates of reactor response. Work on a 2D diffusion model for application in BRISC is described in section 4.3.4.

Additional simplifications and assumptions lead to what is called the "point kinetics" modeling approach. A key aspect of a ***point kinetics model*** is the assumption that the spatial variation of the reactor flux does not change in time, only its overall magnitude. Section 4.3.3 describes a point kinetics model that we applied in the BRISC code.

4.3.2 Discussion of liquid metal reactor (LMR) neutron dynamics

An important characteristic of a properly designed sodium-cooled fast reactor with metal-fuel (TRU-Zr) is a passively safe neutronics / thermal-mechanical response to the heat-up of the reactor core during anticipated accident scenarios. This is one of the most important safety features associated with this type of reactor. References [43] to [50] describe analysis models used previously to quantify this response. Here we briefly review some of the important aspects of this problem.

At steady state, the neutron flux distribution across the reactor core is a function of the geometric configuration, material composition and the thermodynamic state of the reactor materials and components. Perturbations in the coolant, structural, and fuel materials and reactor geometry produce complex changes in the neutron spectrum, leakage, and capture-to-fission rate, which directly affect the power and power distribution within the reactor vessel. A positive (+) reactivity insertion produces an increase in the fission rate and power, while a negative (-) insertion decreases the power.

In general, any change to the system includes both positive and negative (often non-linear) responses, leading to a smaller net effect, which can be very problem dependent and difficult to discern with 'separate effects' studies, such as linear perturbation theory. However, there are general responses associated with most changes. In general, if an LMR loses coolant, there are fewer neutrons parasitically captured (+) in the coolant and the spectrum hardens, which causes less neutrons to scatter below the fission threshold and more fissions occur (+). However, there are also fewer neutrons that scatter back to the fuel region so that a higher fraction will leak out of the system (-). Similarly, if a material is heated, the resonances in the microscopic cross sections broaden (which is called the Doppler effect) and more neutrons are captured in the material (-). But if this material is fuel, then it also leads to more fissions occurring (+), making the net effect problem dependent. In fact, ^{239}Pu can have a positive Doppler coefficient. However, the primary mechanism for passive safety in a metal-fuelled LMR is associated with changes that increase the ratio of the surface area of the active core to fuel mass, which causes more neutrons to leak (-) out of the system.

Because of the high conductivity of the metal fuel and coolant, most reactivity responses in a LMR are considered prompt as compared to the response time in a Light-Water Reactor (LWR). However, because the fraction of total fission neutrons that are from delayed emission of fission products is much less in a fast reactor, the reactivity responses of some perturbations are more prompt than others.

In general, the reactivity responses to geometric and thermal changes can be classified as either local or global responses. The local responses include fuel temperature, and the direct effects associated with bundle pitch and coolant density. The global responses are due to changes in the number and distribution of neutrons leaking from the core associated with indirect effects, such as the axial and radial dimensional changes and changes to the coolant mass fraction in the active core. Accurate calculations of the effects of changes on reactivity are strongly dependent upon the forward and adjoint solutions at both the local and global levels. However, with adjoint

weighted reactivity coefficients, zero-dimensional reactor kinetics proves very accurate in a fast reactor. [51]

Additional details specific to different reactor components are discussed in subsections 4.3.2.1 thorugh 4.3.2.4. A brief discussion of uncertainty estimation for LMR neutron dynamics is provided in subsection 4.3.2.5.

4.3.2.1 Fuel Pins

As the fuel heats, the resonances broaden (the “Doppler” effect) and more neutrons are captured, leading to a direct negative reactivity response to an increase in fuel temperature. However, a primary shutdown mechanism for a metal-fueled fast reactor is axial expansion of the fuel, which has a volumetric expansion that is linear with temperature (density is linear with T^{-1}). In a properly designed metal-fueled LMR (aka with 75% "smear density"), there is enough room for radial expansion of the fuel, due to nominal thermal-expansion and irradiation-induced swelling, such that there will be no pellet-cladding mechanical interaction (PCMI) [52]. Therefore, the fuel is free to expand in all dimensions, leading to a radial expansion ($T^{1/3}$) of the fuel within the cladding, as well as an axial expansion ($T^{1/3}$). The radial expansion ejects the sodium in the fuel/clad gap out of the active core, while the axial expansion increases the surface area of the core, which increases the neutron leakage from the system and serves as the primary shutdown mechanism. [51]

4.3.2.2 Bowing of Fuel Assembly and Duct

The fuel pin assembly includes many wire-wrapped fuel pins in physical contact with a hexagonal duct. In early LMRs, these ducts were mechanically bound to "tie-downs" or core restraints at 3 axial locations (bottom, top of core, and top of duct) to prevent radial movement and minimize vibration. There is space between the ducts, with a by-pass flow of coolant, to allow for thermal expansion and irradiation-induced creep and swelling [51]. The axial expansion of the duct, in the early LMRs was restricted by the "tie-downs," which lead to a compression force that would cause the ducts to "bow." Improved core restraint systems (leaning post and free standing concepts) effectively eliminated the axial restriction. Irradiation-induced creep and swelling still occur in the active core, which leads to a "bowing" above the core, but this has a negligible effect on reactivity. In addition, as this issue is dominated by irradiation effects, it is minimized with low-swelling structural materials, such as HT9. [44, 53]

A power-gradient across an assembly could also lead to a torque stress on the assembly and duct and lead to a "bowing" of the assembly in high-burnup fuel assemblies. If a transient were to occur in an already-stressed assembly, the thermal gradient could increase the torque stresses and cause the assembly to "bow." However, low-swelling structural materials and intentional fuel shuffling patterns effectively eliminate this from consideration.

However, there is a substantial quantity of structural material in the active core, which contains materials with resonances. Therefore, the reactivity response to a perturbation in the cladding temperature has a small, but measurable, effect on the power during a transient. [51]

4.3.2.3 Sodium Loss Reactivity

The mass of sodium coolant in the active core could be changed by temperature changes (primary cause), geometric changes (radial expansion), or fission-gas injection from a fuel pin failure (highly localized). The net effect on reactivity of a loss of coolant is due to spectral hardening (large positive), increased leakage (large negative), reduced neutron capture (small positive), and a change in resonance self-shielding (small). However, the net effect is exceedingly space-dependent as a sodium loss from the center of the core tends to be positive (little change in leakage), whereas a loss from the periphery has a negative effect (large change in leakage).

In a metal-fuelled LMR, the primary source of neutron moderation (slowing down) is the sodium coolant. Because the "importance" of a neutron (the adjoint) increases with energy in a fast reactor, this spectral hardening increases the likelihood of a neutron causing a fission. In addition, the adjoint decreases with radius leading to a very positive effect for a loss of sodium near the center of the core. The loss of parasitic neutron capture in the coolant follows the same trend, but with a smaller magnitude. The change in self-shielding is small and highly-dependent upon the specific isotopic concentrations in the fuel. Proper computation of these reactivity coefficients must include a forward and adjoint solution at the assembly-level to determine these local effects.

The gradient of the neutron density and adjoint are small near the center of the core, but large at the periphery. Therefore, a change in sodium concentration near the center will have a much smaller effect on the system reactivity than a change at the periphery. Proper computation of the coefficients of reactivity must include a full-core calculation of the forward and adjoint solutions to compute these global effects. [51]

4.3.2.4 Core-Support Radial-Expansion Coefficient

The fuel assemblies are locked into a bottom core-support plate and allowed to loosely "float" radially with loose "tie-downs" at the top of the core and top of the ducts. If the core inlet coolant temperature increases, then this bottom support plate is heated and thermally expands. The radial effect of this thermal expansion separates the fuel bundles and effectively increases the surface area of the active core, which leads to two reactivity responses. The spreading of the fuel bundles (increasing the pitch) increases the coolant mass in the core, which was discussed above. In addition, the radius of the core has increased, which leads to a larger effective surface area with a constant fuel mass. This produces a negative reactivity response as there is an increase in neutrons leaking out of the system. [51]

4.3.2.5 Notes on Uncertainty Estimation for LMR Neutron Dynamics

LMR neutron dynamics have been studied for a variety of reactor configurations, including accelerator-driven systems [54-57]. Each provide estimates of the coefficients, but their spread demonstrates the uncertainty in their uncertainty estimates. Some of these results are reproduced in the Tables 4.3-1, 4.3-2, and 4.3-3. As is evident from these tables, there is a factor of two variation in the estimated uncertainty of the burnup swing related to nuclear data (plus another

50% on top of that from modeling), a factor of 3 in sodium void worth and a factor of 20 in the power peaking. This suggests that additional research in the literature to further confirm this trend would be of value.

Table 4.3-1 Uncertainty in LMR Neutronics from Nuclear Data (Reference [54])

Parameter	Reactivity Worth	Relative Uncertainty	Absolute Uncertainty
Burnup Swing (dk)	1.19	180%	2.14
Sodium Void Worth (dk)	1.17	97%	1.13
Doppler Coefficient	-0.32	46%	0.15

Table 4.3-2 Relative Uncertainty in LMR Neutronics (Reference [55])

Parameter	From Nuclear Data	From Modeling
Burnup Swing (dk)	70%	50%
Relative Power Peak	1%	3%
Reactivity Coefficients (component)	20%	20%
Power Distribution	1%	6%
Conversion Ratio	5%	2%
Control Rod Worth (total)	5%	4%
Multiplication Factor (dk/k)	1%	0.5%

Table 4.3-3 Relative Uncertainty in LMR Neutronics from Nuclear Data (Reference [56])

Parameter	Uncertainty
Sodium Void Worth (dk)	3.2
Burnup Swing (dk)	1.28
Relative Power Peak	20.5%
Relative Decay Heat	35%

Of note here is that these calculations have many inherent assumptions and approximations. It appears that most calculations were either performed with linear perturbation theory using a forward and adjoint solution or a difference of two forward solutions. When the responses are non-linear and/or there is substantial (computational and data) error in the solutions, these methods of computation are lacking. In addition, it is difficult to find evidence of verification (very common in the nuclear industry), and it is not clear that the software used has been

examined through peer-review. It does not appear that all contributors to the uncertainty are accounted for (data and computational) and a common assumption is that the primary uncertainty lies in the nuclear data, which is a very bold claim as to the accuracy of a coupled-physics calculation with thermo-mechanical and fluid-dynamic responses. Validation of coupled-physics solutions is very challenging, as the data tends to be global in nature or integrated over large times; therefore it is very prone to cancellation of errors giving the appearance of a much smaller uncertainty than actual for local effects. Finally, the significant safety-related parameters are these local effects (peak fuel/cladding temperature and linear heat generation rate), of which there is little empirical information for validation.

4.3.3 A BRISC point kinetics model

The point kinetics model is an approach commonly used for problems where the spatial variation of the reactor flux does not change in time, only its overall magnitude.

The point kinetics model used in BRISC is based on using specified data for point-kinetics delayed neutrons, six-precursor group point-kinetics delayed neutron data, and assumed reactivity coefficients data. These parameters serve as model input and can be updated or modified as desired. A within-pin power distribution is assumed and the BRISC interface is defined such that heat generation only occurs within fissionable materials.

In this model, the power distribution in the reactor core is given by the following relationship

$$P(\dot{x},t) = n(t)P(\dot{x},0) \tag{4.3.3}$$

where $P(\dot{x},t)$ is the power density at position $\dot{x}$ and time t, $n(t)$ is the point kinetics model output (a nondimensional scaling factor) at time t, and $P(\dot{x},0)$ is the specified initial power density as a function of spatial location (denoted by $\dot{x}$).

The specific equations solved in the point kinetics model can be expressed as follows.

$$\frac{dn(t)}{dt} = \left[\frac{\rho(t)-\beta}{\Lambda}\right]n(t) + \sum_{i=1}^{6}\lambda_i C_i(t) \tag{4.3.4}$$

$$\frac{dC_i(t)}{dt} = \left[\frac{\beta_i}{\Lambda}\right]n(t) - \lambda_i C_i(t) \tag{4.3.5}$$

where

$\rho(t)$ is the reactivity in dollars (100 cents),
β is the total delayed neutron fraction,
β_i is the delayed neutron fraction for delayed group i (note $\beta = \sum_{i=1}^{6}\beta_i$),
Λ is the prompt neutron lifetime,
λ_i is the decay constant for delayed neutron group i,

At time zero we specify the following initial conditions,

$$C_i(0) = \frac{\beta_i}{\lambda_i \Lambda} \tag{4.3.6}$$

$$n(0) = 1.0 \tag{4.3.7}$$

$$\frac{dn(0)}{dt} = \frac{dC_i(0)}{dt} = 0. \tag{4.3.8}$$

The reactivity $\rho(t)$ is evaluated from changes in the physical characteristics of the reactor core. The equation for $\rho(t)$ is of the following form:

$$\rho(t) = \sum_{j=0}^{j=n\_parm} \rho\_coef_j \left[property_j(t) - property_j(0) \right] \tag{4.3.9}$$

where $property_j(t)$ could be a temperature, a density, or some other characteristic. The value of a particular ρ_coef_j is assumed known from lattice and reactor physics calculations. Initial conditions for the reactor system are determined by applying the power distribution, $P(x,0)$, to the core and letting the system come to equilibrium. The values of each $property_j$ at time zero are then noted.

Note from Equation (4.3.4) how the reactor power is controlled by the value of the reactivity. When $\rho(t)$ is zero, the power is constant, and the reactor is said to be at ***delayed critica****l*. When $\rho(t)$ is less than zero, the reactor power is decaying and the reactor is ***sub-critical***. When $\rho(t)$ is greater than or equal to β, the reactor is ***prompt critical*** and the power is increasing exponentially. The reactivity is given in units of β. When $\rho(t)$ is equal to β, its value is one dollar.

To apply the point kinetics model in a calculation, values must be specified for the prompt neutron lifetime (Λ), delayed neutron fractions (β_i), and decay constants (λ_i). Also the reactivity coefficients (ρ_coef_j), and the properties required to compute the reactivity coefficients in Eq. (4.3.9) must be defined. Tables 4.3-4 to 4.3-6 provide illustrative values for each of these as extracted from Reference[5].

The point kinetics equations are often solved using the Runge-Kutta method. However, since this method is explicit in time, a stability condition will severely limit the time step size. The method implemented in BRISC is a Pade' method (defaulted to 4^{th} order) which is fully implicit in time (the reactivity coefficients are defined by the beginning of the neutronics time step). To handle very large time step sizes, an algorithm was implement to sub-cycle the point-kinetics calculation to reduce truncation error, with an assumed linear dependence of the reactivity coefficient dependent variables: various average temperatures). In addition, a standard first-order time discretization (explicit to implicit) has been implemented for tight-coupling of the point-kinetics with the thermal solution. However, because of the stiffness of the point-kinetics equations, the

resulting simulation would require a very small time step for accuracy, thus it is not recommended.

This point kinetics model was fully implemented, tested, and integrated with the overall code for parallel non-linear reactor transient problems.

Table 4.3-4 Prompt neutron lifetime estimates for the Argonne Advanced Burner Test Reactor Preconceptual Design (see reference [5]).

	Beginning of Cycle	End of Cycle
Prompt Neutron Lifetime, Λ (sec.)	4.15×10^{-7}	4.49×10^{-7}

Table 4.3-5 Estimates of the Point Kinetics Delayed Neutron data for the Argonne Advanced Burner Test Reactor Preconceptual Design (see reference [5]).

Group	Delayed Neutron Fraction, β_i		Decay Constant, λ_i (1/s)	
	Beginning of Cycle	End of Cycle	Beginning of Cycle	End of Cycle
1	6.7974×10^{-5}	6.8109×10^{-5}	0.01338	0.01338
2	5.6945×10^{-4}	5.7345×10^{-4}	0.03060	0.03060
3	3.8830×10^{-4}	3.9054×10^{-4}	0.1159	0.1159
4	8.9155×10^{-4}	8.9772×10^{-4}	0.3026	0.3026
5	4.8191×10^{-4}	4.8553×10^{-4}	0.8672	0.8673
6	1.6915×10^{-4}	1.7037×10^{-4}	2.918	2.919
Total	2.5683×10^{-3}	2.5857×10^{-3}		

Table 4.3-6 Estimates of six temperature-dependant reactivity coefficients for the Argonne Advanced Burner Test Reactor Preconceptual Design (see reference [5]).

Reactivity coefficients (cent / deg. C)	index	Startup Core		Recycled Core	
		BOEC	EOEC	BOEC	EOEC
Radial expansion	1	-0.55	-0.56	-0.58	-0.59
Sodium density	2	0.05	0.05	0.07	0.07
Structure density	3	0.04	0.04	0.04	0.04
Fuel density	4	-0.72	-0.73	-0.76	-0.77
Axial expansion	5	-0.08	-0.08	-0.09	-0.08
Fuel doppler	6	-0.07	-0.07	-0.07	-0.07

4.3.4 A 2-D diffusion model

Significant work was completed towards developing a higher fidelity point kinetics model. This is accomplished by using a 2-D diffusion model to compute the average flux across the reactor core and to compute the critical eigenvalue (k_{eff}) used in the point-kinetics model. The code written for this purpose is called Rascal.

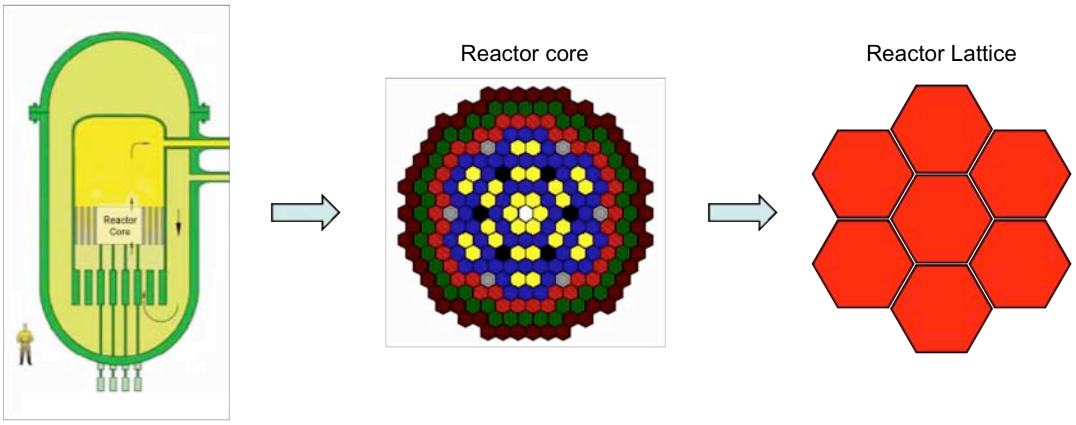


Figure 4.3.2 Rascal computes the neutron flux in reactor and lattice geometries

Rascal solves the following 2-D, steady state, multi-group diffusion equation for the average (group dependent) neutron flux ϕ in reactor and lattice geometries such as illustrated in Figure Figure 4.3.2.

$$-\nabla \cdot D_g(\vec{r})\ \nabla\phi_g(\vec{r}) + \Sigma_{rg}(\vec{r})\ \phi_g(\vec{r}) = \sum_{g'=1}^{G} \left[\frac{1}{k_{eff}} \chi_g \nu\ \Sigma_{fg}(\vec{r}) + \Sigma_{gg'}(\vec{r})\right] \phi_{g'}(\vec{r}) \qquad (4.3.10)$$

where

$\vec{r}$	2-D spatial position vector,
D_g	diffusion coefficient for energy group g,
Σ_{rg}	macroscopic total cross section for group g,
k_{eff}	critical eigenvalue of the system,
χ_g	fraction of fission neutrons (prompt and delayed) that appear in group g,
ν	total number of fission neutrons produced per fission,
Σ_{fg}	macroscopic fission cross section for group g,
$\Sigma_{gg'}$	macroscopic scattering cross section from group g' into group g,

and where the subscript “g” denotes the group number id in a multi-group analysis having “G” energy groups.

The output of Rascal is the average core neutron flux and the critical eigenvalue k_{eff}, where k_{eff} is directly related to the reactivity of the system as follows.

$$\rho_{rascal} = 1 - {}^{1}\!/\!_{k_{eff}} \tag{4.3.11}$$

However, since Rascal doesn't currently account for geometric changes, the net reactivity ρ_{net} is

$$\rho_{net} = \rho_{rascal} + \rho_{geom} \tag{4.3.12}$$

where ρ_{rascal} reflects changes in reactivity due to changes in space-dependent cross sections with a constant geometry (computed in 2D with Rascal) and ρ_{geom} accounts for reactivity changes due to changes in geometry (structural expansion due to temperature changes) computed from Table 4.3-6, index 1 and 5.

The average flux is used to compute the 2-D spatial shape of the power distribution using the fission cross section data provided at input. A cosine shape is assumed for the axial direction. This distribution is then normalized to the power computed by the point-kinetics model thus providing a 3-D, time dependant power solution. The power distribution is a heat source in the thermal analysis.

Rascal can handle both symmetry and vacuum boundary conditions.

Rascal computes the average neutron flux over a cell. In Rascal, a cell must be a simple polygon, such as a triangle, square, or hexagon. Each cell must have the same volume and cell sides must be the same length. Examples of four different cell geometries are given below in Figure 4.3.3. An example that uses a hex-shaped cell in an overall geometry defined for use in a BRISC full-scale reactor test problem is illustrated in Figure 4.3.4.

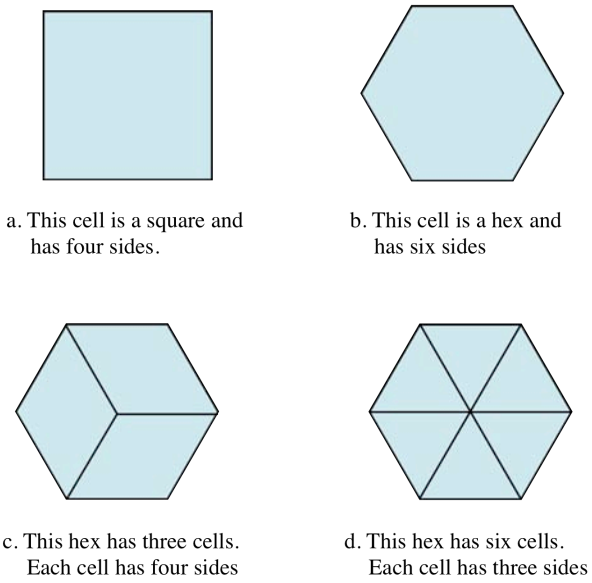


Figure 4.3.3 Example cell shapes usable in a Rascal calculation.

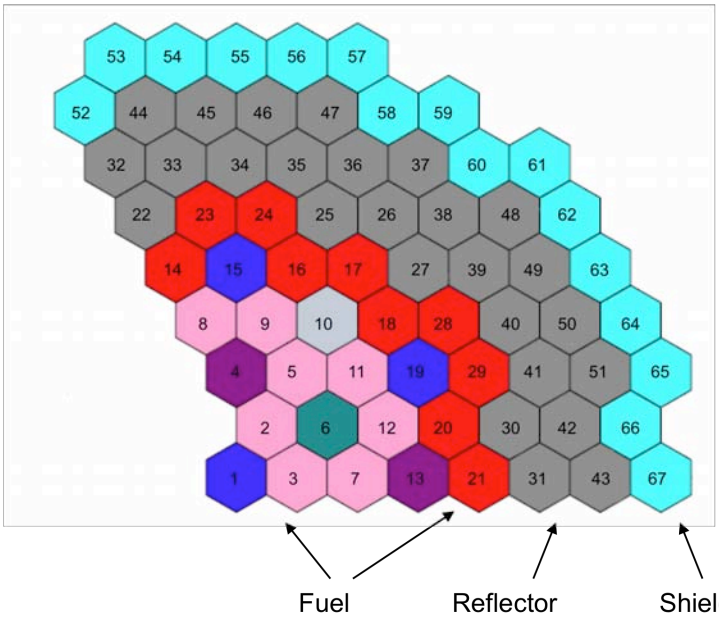


Figure 4.3.4 Illustrative Rascal geometry for an envisioned full-scale reactor problem. A single cell per hex is used and 120 degree symmetry is specified.

Rascal solves a geometrically static problem, so an increase in core height or in core radius that may occur during a transient cannot be properly modeled. Therefore, the overall reactivity is due to the change in reactivity due to changes in the macroscopic cross sections, as computed with Rascal, plus the changes estimated through point-kinetics parameters that account for radial and axial expansion. Initial efforts were developed to create the temperature-dependent cross section data from the SCALE(TRITON) code, but were never completed and implemented within Rascal.

4.3.5 Envisioned future use of SCALE(TRITON) and NESTLE

SCALE(TRITON) is a lattice physics code that generates time-temperature-state dependent macroscopic cross sections for use in a core-diffusion code, such as Rascal. SCALE(TRITON) has been closely developed and linked with both the NESTLE and PARCS[58] 3D core simulators. Much like Rascal, they take a temperature distribution, define the macroscopic cross section distribution, and solve the eigenvalue form of the diffusion equation. NESTLE has been integrated with the RELAP5-3D thermal-hydraulics code [59] and PARCS has been integrated with the TRACE thermal-hydraulics code[60], both for transient or steady state analyses. It is envisioned that the diffusion and kinetics equation in NESTLE (or PARCS) would be integrated into the BRISC code system and cross sections would be pre-processed by SCALE(TRITON). However, neither PARCS or NESTLE are presently capable of modeling geometric redistribution, such as axial and radial expansion. Therefore, future developments in a core simulator will be required for accurate fast reactor core transient analysis.

4.3.6 Generalized Neutronics Interface for Coupling to BRISC

This section describes characteristics of the proto-type of a generalized interface developed to couple the neutronics calculations to the multi-physics driver in BRISC.

The neutronics component requires material temperatures integrated over several specific locations, to determine the reactivity and/or cross section changes. The neutronics component provides either a zero- or three-dimensional power distribution. Therefore, a simple C interface was developed to either define a zero-dimensional (point kinetics) or two/three dimensional (point kinetics plus Rascal) problem. For the zero-dimensional problem, a vector of 4 double precision values for the temperature is provided and a single double precision power is returned. For the two/three dimensional problem, the same vector of 4 values is provided to define the spatially-averaged temperature of materials at various locations. However a vector of axially-averaged coolant temperatures is provided as well. This allowed Rascal to compute the 2D power distribution. Through the assumption of a cosine-shaped axial power distribution, the neutronics component returns a three-dimensional power distribution.

4.4 Sub-grid scale pin model

Figure 1.5 (see Section 1) illustrates the dimensions and key regions in a fuel pin envisioned for use in the Argonne advanced burner test reactor preconceptual design. Fuel pins and control rods are examples of important reactor components whose features are too small for resolution on the BRISC 3D meso-scale mesh. BRISC is designed so that one or more sub-grid scale models can be associated with any meso-scale control volume. However, only one sub-grid scale model per control volume is implemented in the current BRISC proto-type.

In BRISC, the sub-grid scale pin model is defined in a separate physics code where the mesh resolution is appropriate for the geometry and physics of interest in a fuel pin or control rod, but which represents the average behavior of the several-to-many pins/rods that may actually exist within the control volume. This section describes the relatively simple pin model used in the BRISC prototype code. It contains essential characteristics required for performing proto-type calculations that more elaborate models would have. In practice, this simple model should be replaced by a more accurate model.

4.4.1 Model theory and equations

The simple pin model used in BRISC makes the following approximations.

- Axial conduction is neglected and heat transfer is azimuthally symmetric.
- Material properties are constant.
- At any point in time the radial temperature profile in the fuel is approximated by a simple second order polynomial.
- The temperature difference across the cladding thickness is negligible.

These simplifications lead to the following 1-dimensional equation for temperature in the fuel region

$$T(r) = T_0 + \frac{(T_1 - T_0)}{R_1^2} r^2 = T_0 \left[1 - \left(\frac{r}{R_1} \right)^2 \right] + \left(\frac{r}{R_1} \right)^2 T_1 \tag{4.4.1}$$

where T_0 and T_1 are the fuel pellet centerline and surface temperatures respectively. Figure 4.4.1 illustrates such a profile for a given set of values for T_0, T_1 and cladding temperature T_c. This figure also shows the zero gradient boundary condition applied at the pellet centerline, r=0.

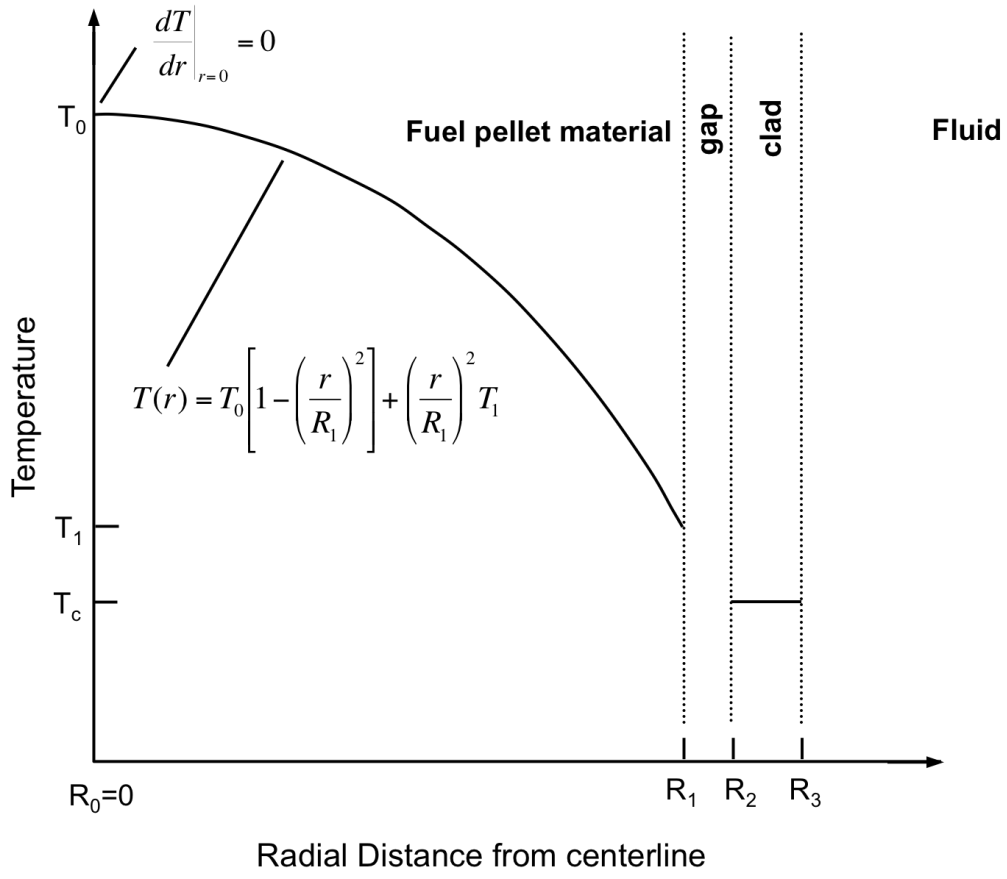


Figure 4.4.1 Illustration of the 1-D quadratic temperature variation assumed within a fuel pellet.

To evolve the temperature profile in time, equations expressing the conservation of energy within the fuel, across the gap, and within the clad can be written. If the gap thickness is assumed to be negligible, and heat transfer coefficients for the gap and for the fluid-clad interface can be specified, then the following set of three equations with three unknowns (T_0, T_1 and T_c) can be written.

heat flux continuity across gap

$$k_p \left.\frac{dT}{dr}\right|_{r=R_1} = \frac{2k_p}{R_1}(T_1 - T_0) = h_g(T_c - T_1) \tag{4.4.2}$$

pellet energy conservation

$$\frac{d\bar{T}_p}{dt} = \frac{1}{2}\frac{d(T_0 + T_1)}{dt} = \frac{Q}{[\rho Cp]_p} + \frac{h_g A_1}{[\rho CpV]_p}(T_c - T_1) \tag{4.4.3}$$

clad energy conservation

$$\frac{dT_c}{dt} = \frac{h_f A_3}{[\rho CpV]_c}(T_f - T_c) - \frac{h_g A_1}{[\rho CpV]_c}(T_c - T_1) \tag{4.4.4}$$

where

Cp:	Specific heat *J/(kg K)*
ρ:	density *kg/m^3*
k:	thermal conductivity *J/(s m K)*
Q:	volumetric heat source *J/(s m^3)*
h:	heat transfer coef. *J/(s m^2 K)*
R_1	radius of pellet *m*
R_2	inner radius of clad *m*
R_3	inner radius of clad *m*
V	volume *m^3*

with subscripts denoting the following

p	pellet
g	gap
c	clad
f	fluid.

In this model, the following quantities are specified constants that define the characteristics of the pin being represented at a given location.

R_1	radius of pellet (also equal to inner radius of the clad R_2)
R_3	outer radius of clad
h_g	heat transfer coefficient across the fuel-clad gap
h_f	heat transfer coefficient between the clad and the surrounding fluid
ρ_p	density of the fuel pellet material
Cp_p	specific heat of the fuel pellet material
k_p	thermal conductivity of the fuel pellet material
Q:	volumetric heat source in the fuel pellet
ρ_c	density of the clad material
Cp_c	specific heat of the clad material

Also, the surrounding fluid temperature T_f must be specified, which provides the coupling to the meso-scale fluid flow model.

Note that in this formulation the volumetric average temperature of the fuel pellet, $\overline{T}_p$, is

$$\overline{T}_p = \frac{2\pi}{V_p}\int_0^{R_1} T(r)rdr = \frac{2}{R_1^2}\int_0^{R_1}(T_0 r + \frac{(T_1 - T_0)}{R_1^2}r^3)dr = \frac{(T_0 + T_1)}{2}, \tag{4.4.5}$$

and the following geometric quantities can be computed as:

$V_p = \pi (R_1)^2$	Volume of fuel pellet
$A_1 = A_2 = 2\pi R_1$	Outer surface area pellet, inner surface area of clad
$A_3 = 2\pi R_3$	Outer surface area clad
$V_c = \pi [(R_3)^2 - (R_1)^2] = \pi(R_3-R_1)(R_3+R_1)$	Volume of the clad

If the equations are temporally discretized using a fully implicit first order approximation, then the following matrix equation can be written.

$$\begin{pmatrix} \frac{-2k_p}{R_1} & \left(\frac{2k_p}{R_1} + h_g\right) & -h_g \\ 1 & 1 + \frac{2\Delta t h_g A_1}{[\rho CpV]_p} & -\frac{2\Delta t h_g A_1}{[\rho CpV]_p} \\ 0 & -\frac{\Delta t h_g A_1}{[\rho CpV]_c} & \left[1 + \frac{\Delta t(h_f A_3 + h_g A_1)}{[\rho CpV]_c}\right] \end{pmatrix} \mathbf{X} \begin{pmatrix} T_0 \\ T_1 \\ T_c \end{pmatrix} = \mathbf{=} \begin{pmatrix} 0 \\ T_0^{n-1} + T_1^{n-1} + \frac{2\Delta t Q}{[\rho Cp]_p} \\ T_c^{n-1} + \frac{\Delta t h_f A_3}{[\rho CpV]_c} T_f \end{pmatrix} \tag{4.4.6}$$

In BRISC, this equation set is solved using direct guassian elimination.

Note that a separate pin model is associated with each meso-scale control volume that contains fuel pins. Energy conservation requires that the fluid-pin heat transfer be identical in both the sub-grid scale pin model and the meso-scale RIO code. This consistency is enforced in the overall solution algorithm that is controlled by the multi-physics driver (See Section 2).

4.5 Ex-vessel and balance of plant

When doing systems level modeling of a nuclear power plant, many important systems exist outside of the reactor pressure vessel. The successful recovery of a plant from off-normal transients and accident scenarios is often determined by the proper operation of these systems. Thus accurate modeling of these systems is equally important to the modeling of the in-vessel phenomena. However, in this LDRD resources were not devoted to developing improved ex-vessel or balance of plant models. Instead, our strategy was to develop a coupling capability that would enable any current or future code that treats these important phenomena to be incorporated into the overall system simulation. As a target code representing one of several options that currently exist we chose MELCOR [2]. MELCOR has a long history of development and support at Sandia as a second-generation plant risk assessment tool (with work funded primarily by the U.S. Nuclear Regulatory Commission).

4.5.1 MELCOR

MELCOR is described as a fully integrated, engineering-level computer code whose primary purpose is to model the progression of accidents in light water reactor nuclear power plants. Recently, it has been modified for liquid sodium properties, making it suitable for some LMR-type reactor simulations. The code itself is composed of an executive driver and a number of major modules, or packages, that together model the major systems of a reactor plant. The three general types of packages that are in MELCOR are

1. Basic physical phenomena, such as hydrodynamics, heat and mass transfer to structures, gas combustion, aerosol and vapor physics,
2. Reactor-specific phenomena, such as decay heat generation, core degradation, ex-vessel phenomena, sprays and ESFs,
3. Support functions, such as thermodynamics, equations of state, other material properties, data-handling utilities, and equation solvers.

Figure 3.1 (see Section 3) illustrates that the envisioned role of MELCOR in the overall BRISC simulation of a LMR NPP is primarily to model the operation and response of ex-vessel components, structures and other physical processes whose features are too large for resolution on the 3-D meso-scale grid. This potentially includes a flow and heat transfer through a host of pipes, pumps, valves, heat exchangers, turbines, and so forth, and which together make up the balance of plant. Fluid flow and heat transfer throughout the secondary sodium loop, and also in the primary loop pumps and pipes leading to the lower plenum will thus be modeled as a 1D network-type flow problem. For our target plant simulation (the Argonne ABR preconceptual design), the interaction point with the meso-scale models occurs in two places – the intermediate heat exchanger (IHX) and the direct reactor auxiliary cooling system (DRACS).

Details of the MELCOR computer code are not be presented here as excellent reference material is available elsewhere [2]. However, with respect to fluid flow and heat transfer, we note that MELCOR uses a control-volume/flow path formulation to solve linearized-implicit finite difference forms of the following conservation equations for mass, momentum, and energy.

Mass:

$$\frac{\partial \rho}{\partial t} + \nabla \bullet \left(\rho v\right) = \Gamma \tag{4.5.1}$$

Momentum:

$$\begin{aligned}\alpha_{j,\phi}\rho_{j,\phi}L_j\frac{\partial v_{j,\phi}}{\partial t} &= \alpha_{j,\phi}(P_i - P_k) + \alpha_{j,\phi}(\rho g \Delta z) + \alpha_{j,\phi}\Delta P_j \\ &\quad -\frac{1}{2}K^*_{j,\phi}\alpha_{j,\phi}\rho_{j,\phi}\left|v_{j,\phi}\right|v_{j,\phi} - \alpha_{j,\phi}\alpha_{j,-\phi}(v_{j,\phi} - v_{j,-\phi}) + \alpha_{j,\phi}\rho_{j,\phi}v_{j,\phi}(\Delta v)_{j,\phi} \\ &\quad +\alpha_{j,\phi}\rho_{j,\phi}v_{j,\phi}(\Delta v)_{j,\phi}\end{aligned} \tag{4.5.2}$$

Energy:

$$\frac{\partial E_{i,\phi}}{\partial t} = \sum_j \sigma_{ij}\alpha_{j,\phi}\left(\sum_m \rho^d_{j,m}h^d_{j,m}\right)v_{j,\phi}F_jA_j + \dot{H}_{i,\phi} \tag{4.5.3}$$

Convective boundary conditions to a volume can be modeled using heat transfer coefficients that are either calculated in MELCOR or specified by the user. This provides the key mechanism by which coupling to the meso-scale flow and heat transfer model will occur in BRISC.

4.5.2 Equations for coupling the Primary and Secondary Flow systems

In the BRISC representation of the LMR system, RIO models the fluid in the primary system and MELCOR models the fluid in the secondary system (as well as in any separate auxiliary systems). The codes are coupled through the heat transfer that occurs in the intermediate heat exchanger (IHX) and the direct reactor auxiliary cooling system (DRACS) heat exchanger.

In RIO, the heat exchanger region is represented as a 3D porous medium consisting of primary system fluid and a stationary solid phase. The solid phase represents the tubes through which the secondary system coolant is flowing. Heat transfer between the primary system fluid and solid phase is calculated by RIO. However, the solid phase is modeled as containing a heat source that is proportional to temperature difference between the RIO solid-phase and the associated MELCOR secondary system fluid temperature.

In MELCOR, the heat exchanger region is treated as part of a 1D flow network. The control volumes associated with the heat exchanger are assigned appropriate values for the surface to volume ratio (SVR) such that the pressure drop is high, and are set up to expect a heat source (see Equations (4.5.3) above.)

Figure 4.5.1 illustrates a simple heat exchanger where the MELCOR-RIO coupling is to be modeled. MELCOR treats the heat exchanger region with four MELCOR CVs with the secondary fluid flow path being upward. RIO treats the primary system as a 3D porous medium with a much larger number of control volumes. In this example there are 18 RIO control volumes associated with each MELCOR control volume.

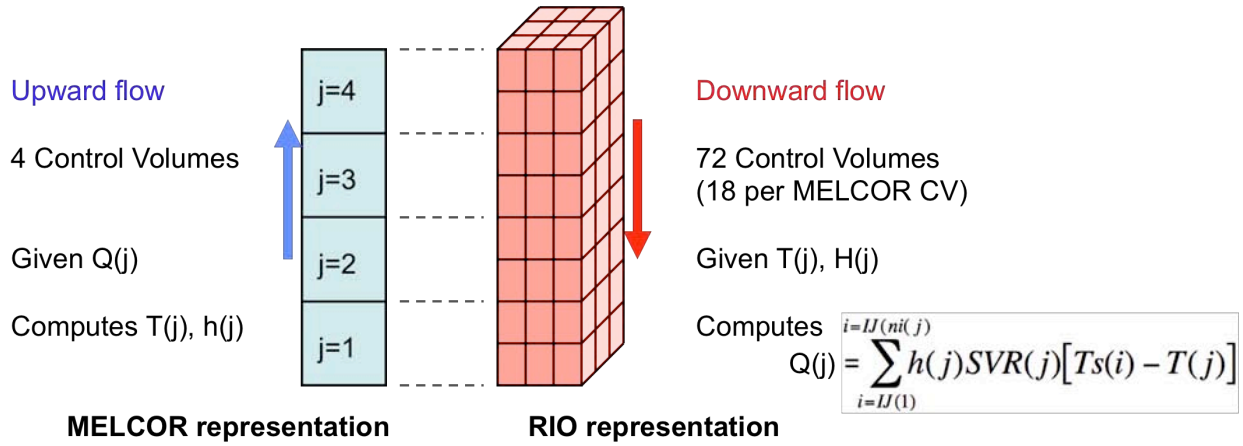


Figure 4.5.1 Illustration of an example RIO-MELCOR coupling through a heat exchanger.

At each RIO control volume "i", we have a solid phase temperature Ts(i). The heat sink/source $q_s(i)$ in this solid material corresponds to the heat transferred to/from the secondary side fluid, and is modeled according to the following:

$$q_s(i) = -h(j)SVR(j)\left[Ts(i) - T(j)\right] \tag{4.5.4}$$

where "j" is the index into the MELCOR heat exchanger discretization and $h(j)$ and SVR(j) are the secondary-side heat transfer coefficient and solid-surface-to-volume ratio respectively. Note that $h(j)$, SVR(j), and T(j) are values obtained from MELCOR.

The total heat transfer to the secondary side associated with MELCOR control volume j, Q(j), is a sum of the contributions from all RIO CVs that are contained in heat exchanger section "j".

$$Q(j) = \sum_{i=IJ(1)}^{i=IJ(ni(j))} h(j)SVR(j)\left[Ts(i) - T(j)\right] \tag{4.5.5}$$

where IJ() is the index list of all values of i (the RIO control volumes) corresponding to MELCOR control volume j, and ni(j) is the total number of RIO control volumes in MELCOR control volume j.

On the MELCOR side of the model, Q(j) is received from RIO as the local heat source into the secondary fluid. MELCOR requires this to solve its energy equation for the current fluid temperatures. Also, based on the local flow conditions a heat transfer coefficient h(j) is calculated.

The MELCOR and RIO solutions are only consistent and converged when the equations computing the heat transfer in both codes are identically equal.

4.6 Heat-transfer across a fluid-solid interface with non-conformal meshes

A nuclear reactor is a complex multiphysics system where different physical processes are important in different sub-domains of the overall system. When modeled, the different physical processes may require their own computational domains, each potentially discretized by a separate independent mesh. When these sub-domains are adjacent to each other, they may share a common boundary. Because the physical processes are coupled, an integrated simulation of the entire system involves data transfer at these common boundaries and interfaces. If the discrete meshes used to represent the different spatial domains are not conformal at the shared boundaries, then it becomes especially challenging to accurately transfer information across these boundaries.

From a meso-scale modeling standpoint, the BRISC approach treats the reactor vessel as consisting of two distinctly different physical domains - the fluid domain and the solid domain. An example of this is illustrated and discussed in Section 3.2. The fluid domain consists of all the physical space where, within the resolution of the meso-scale mesh, fluid flow can occur, and includes two-phase regions (that contain a solid phase representing structure not resolved on the meso-scale mesh) as well as regions that are pure fluid regions. The solid domain consists of the physical space occupied by "resolved" solid structures. In our model, this includes the space occupied by reactor vessel itself, the redan, the core barrel, lower plenum structure, and so forth. These solid components form a solid no-slip boundary to the reactor coolant and need only be modeled as heat conducting materials. Thus in BRISC, we need a method for transferring energy across a solid-fluid boundary when the fluids mesh and the solid mesh are not aligned (i.e. they are non-conformal), and which (a) is locally conservative, and (b) allows for a coupled solution that is consistent in time (i.e. not time-lagged).

In BRISC, two distinct instantiations of the RIO code are used to model the fluids domain and the solid domain physics. Because RIO is a finite volume code, the method for transferring energy across a solid-fluid boundary must be consistent with the RIO formulation. The method we use requires the construction and use of a special "exchange" surface mesh, which in recent literature [19, 20, 61, 62], is also called a "common refinement" surface mesh. Here we use the term "exchange surface" to denote that the heat transfer across the shared interface is "exchanged" through the use of this discrete surface.

Section 3.3 describes the steps used to create the exchange surface mesh in a form usable in BRISC. The central tool we use in creating the exchange surface mesh is a utility code called Surfdiver, developed by Xiangmin Jiao as part of the Center for Simulation of Advanced Rockets (CSAR) effort [18] at the University of Illinois at Urbana-Champaign. Surfdiver was developed as a preprocessor of Rocface, a service utility for transferring data between surface meshes in a fashion that is numerically accurate and physically conservative.

The purpose of this section is to describe the equations solved in BRISC to perform the heat transfer across this mesh.

4.6.1 Equations for heat transfer through the interface mesh element

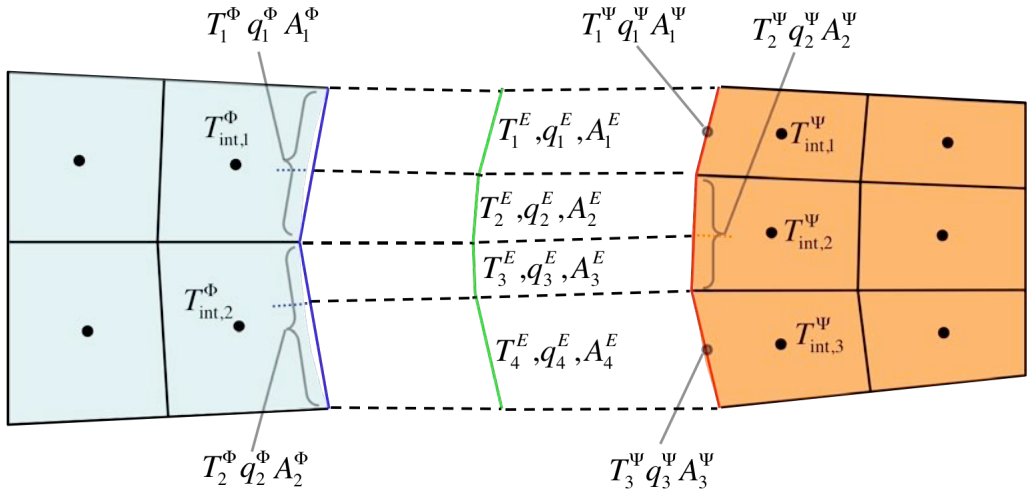


Figure 4.6.1 Illustration of how "exchange" surface elements (superscript "E") relate to the Φ and Ψ shared-interface side set elements.

Figure 4.6.1 illustrates a simple problem where two physics domains, Φ (e.g. fluids domain) and Ψ (e.g. solid domain), each with independently generated volume meshes, share a common interface. Regions Φ and Ψ are artificially separated in Figure 4.6.1 so that the exchange surface mesh (superscript "E") can be distinguished from the two surface meshes associated with either domain. In this example the solid domain mesh represents the surface with two surface elements, the fluid domain mesh represents it with three surface elements, and the exchange surface contains four elements.

Each of the three interface representations in Figure 4.6.1 consist of a finite number of surface elements, where each surface element has the following three attributes; (1) an area A (m^2), (2) a temperature T (K), and (3) a heat transfer rate q (J/s). Note that the heat flux (J/s-m^2) through each surface element is found by dividing the element heat transfer rate by the element area.

In RIO, the heat transfer q_i at any boundary surface element "i" satisfies

$$\frac{q_i}{A_i} = \Gamma_i \left(T_i - T_{\text{int},i} \right) \tag{4.6.1}$$

where T_i and A_i are the surface temperate and area respectively, $T_{int,i}$ is the element control volume temperature, and Γ_i (computed internal to RIO) is a function of the material thermal conductivity and element geometry. Knowing this, we define model equations for the heat

transfer across each individual exchange surface element in terms of the neighboring element control volume temperatures, consistent with the internal modeling of RIO.

$$q_i^E = -A_i^E \Gamma_i^E \left(T_{j(i)}^{\Psi} - T_{k(i)}^{\Phi} \right) \tag{4.6.2}$$

where

$$\Gamma_i^E = \frac{1}{\frac{1}{\Gamma_{j(i)}^{\Phi}} + \frac{1}{\Gamma_{k(i)}^{\Psi}}}, \tag{4.6.3}$$

and where j(i) denotes the surface element index in Φ that is adjacent to exchange surface i, and k(i) denotes the surface element index in Ψ that is adjacent to exchange surface i.

As a specific example, consider exchange surface 4 in Figure 4.6.1. On its left side

$$\frac{q_2^{\Phi}}{A_2^{\Phi}} = -\Gamma_2^{\Phi} \left(T_2^{\Phi} - T_{\text{int},2}^{\Phi} \right) \tag{4.6.4}$$

and on its right side

$$\frac{q_3^{\Psi}}{A_3^{\Psi}} = -\Gamma_3^{\Psi} \left(T_{\text{int},3}^{\Psi} - T_3^{\Psi} \right) \tag{4.6.5}$$

Then heat transfer through exchange surface 4 is modeled as

$$q_4^E = -A_4^E \Gamma_4^E \left(T_{\text{int},3}^{\Psi} - T_{\text{int},2}^{\Phi} \right) \tag{4.6.6}$$

where

$$\Gamma_1^E = \frac{1}{\frac{1}{\Gamma_2^{\Phi}} + \frac{1}{\Gamma_3^{\Psi}}}. \tag{4.6.7}$$

4.6.2 Consistent boundary conditions

In BRISC the fluid region code (RIO-fluid) is set up to require specified temperatures as boundary conditions at each of the FSI surface elements. Conversely, the solid region code (RIO-solid) is set up to require a specified heat flux (this choice is arbitrary as either option on either domain could be treated). These must both be consistent with the heat transfer rates through the exchange surface for the solutions to be fully coupled.

4.6.2.1 Specified heat flux

To compute the corresponding heat flux on side-set surfaces we simply apply energy conservation. Depending on which side-set we are computing this can be written as either

$$\frac{q_k^{\Psi}}{A_k^{\Psi}} = \frac{1}{A_k^{\Psi}} \sum_{i=IK(1)}^{i=IK(Nik)} q_i^E \tag{4.6.8}$$

or

$$\frac{q_j^{\Phi}}{A_j^{\Phi}} = \frac{1}{A_j^{\Phi}} \sum_{i=IJ(1)}^{i=IJ(Nij)} q_i^E \tag{4.6.9}$$

where Nij and Nik are the number of exchange surface elements that correspond to side-set elements j and k respectively, and IJ( ) and IK( ) are the index arrays containing the list of corresponding exchange surface elements.

The application of equations (4.6.8) and (4.6.9) are illustrated in Figure 4.6.2 for the simple example problem.

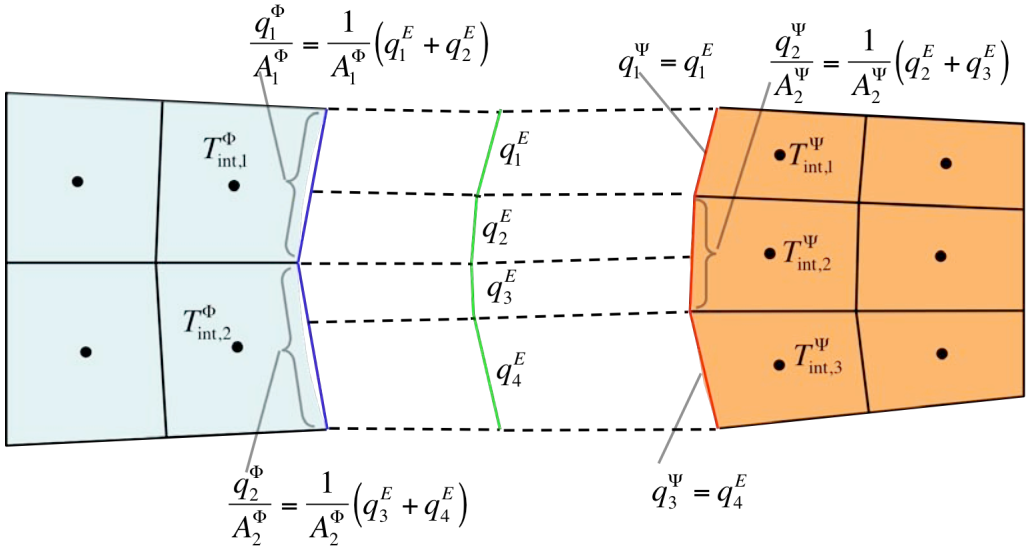


Figure 4.6.2 Illustration of how exchange surface element heat transfer rates are used to compute Φ and Ψ shared-interface side set element boundary condition heat fluxes.

4.6.2.1 Specified Temperature

To compute the corresponding temperature on side-set surfaces we must first compute exchange surface temperatures,and then use these to calculate area weighted side-set surface temperatures. Each exchange surface temperature T_i^E can be calculated either of two ways, corresponding to whether the solid or fluid domain is used as the reference.

$$T_i^E = T_{\text{int},j(i)}^{\Phi} - \frac{q_i^E}{A_i^E \Gamma_{j(i)}^{\Phi}} = T_{\text{int},k(i)}^{\Psi} + \frac{q_i^E}{A_i^E \Gamma_{k(i)}^{\Psi}} \tag{4.6.10}$$

where j(i) and k(i) are defined as before. Knowing these we compute the corresponding temperatures on side-set surfaces as area weighted averages. Depending on which side-set we are computing this can be written as either

$$T_k^{\Psi} = \frac{\sum_{i=IK(1)}^{i=IK(Nik)} A_i^E T_i^E}{\sum_{i=IK(1)}^{i=IK(Nik)} A_i^E} \tag{4.6.11}$$

or

$$T_j^{\Phi} = \frac{\sum_{i=IJ(1)}^{i=IJ(Nij)} A_i^E T_i^E}{\sum_{i=IJ(1)}^{i=IJ(Nij)} A_i^E} \tag{4.6.12}$$

The application of Equations (4.6.11) and (4.6.12) are illustrated for the simple example problem in Figure 4.6.3.

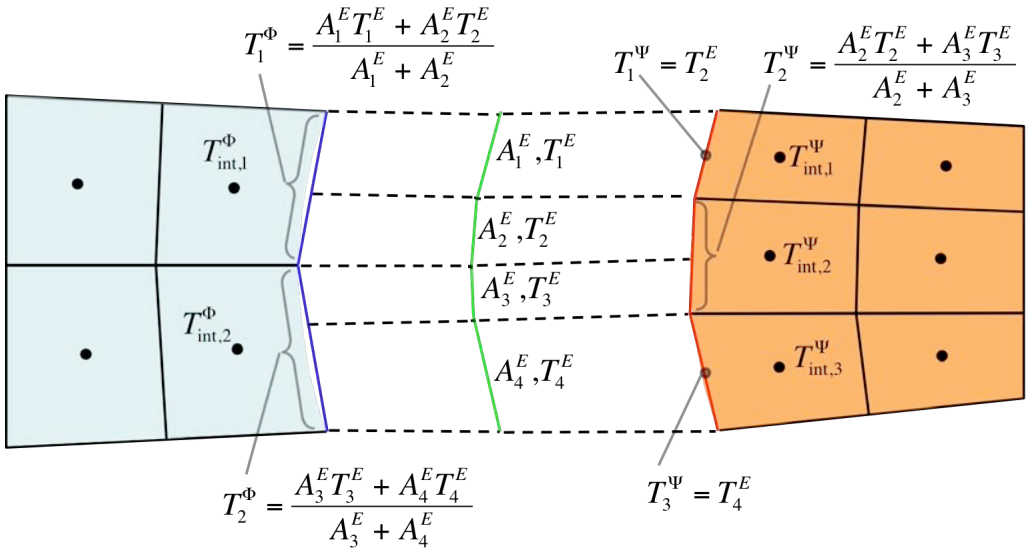


Figure 4.6.3 Illustration of how exchange surface element temperatures are used to compute Φ and Ψ shared-interface side set element boundary condition temperatures.

4.6.3 Note on computing residual equations

Many non-linear solution methods employ residual equations as a means of computing updates to the global solution vector. The equations in Section 4.6.2 can be used to define residual equations in several different ways. If using a JFNK approach, BRISC currently uses residuals that are computed on each volume element that contains a shared-interface surface element. These residuals are computed after the specified boundary conditions are updated using the most recent solutions to the equation sets.

The equations in Section 4.6.2 can also be applied directly in a simple Picard iteration scheme, which is also an option in BRISC to solve the non-linear coupling problem.

5. BRISC: A PROTO-TYPE CODE FOR AN ADVANCED IPSC

5.1 Code Structure

BRISC is based on the coupling of six different physics codes that have been brought together using the LIME multi-physics coupling strategy and environment described in Section 2. Figure 5.1 is a schematic diagram that shows each of these codes, the LIME components, input files and the interaction pathways that exist within BRISC. Output files created by the problem manager and potentially from each of the physics codes are not shown in this figure. Also, a code dubbed "MCLite" (for MELCOR Light) is shown in this figure because, except for some basic testing, this code was used instead of MELCOR in all of the coupled runs. MClite is an extremely simple code that was written to have essentially the same functional interface as MELCOR, but which computed the required values without any real physics.

Three different computer software languages are represented in these codes: C++ (Fuel Pin, FSI, MClite), C (RIO), and Fortran (Kinetics, MELCOR).

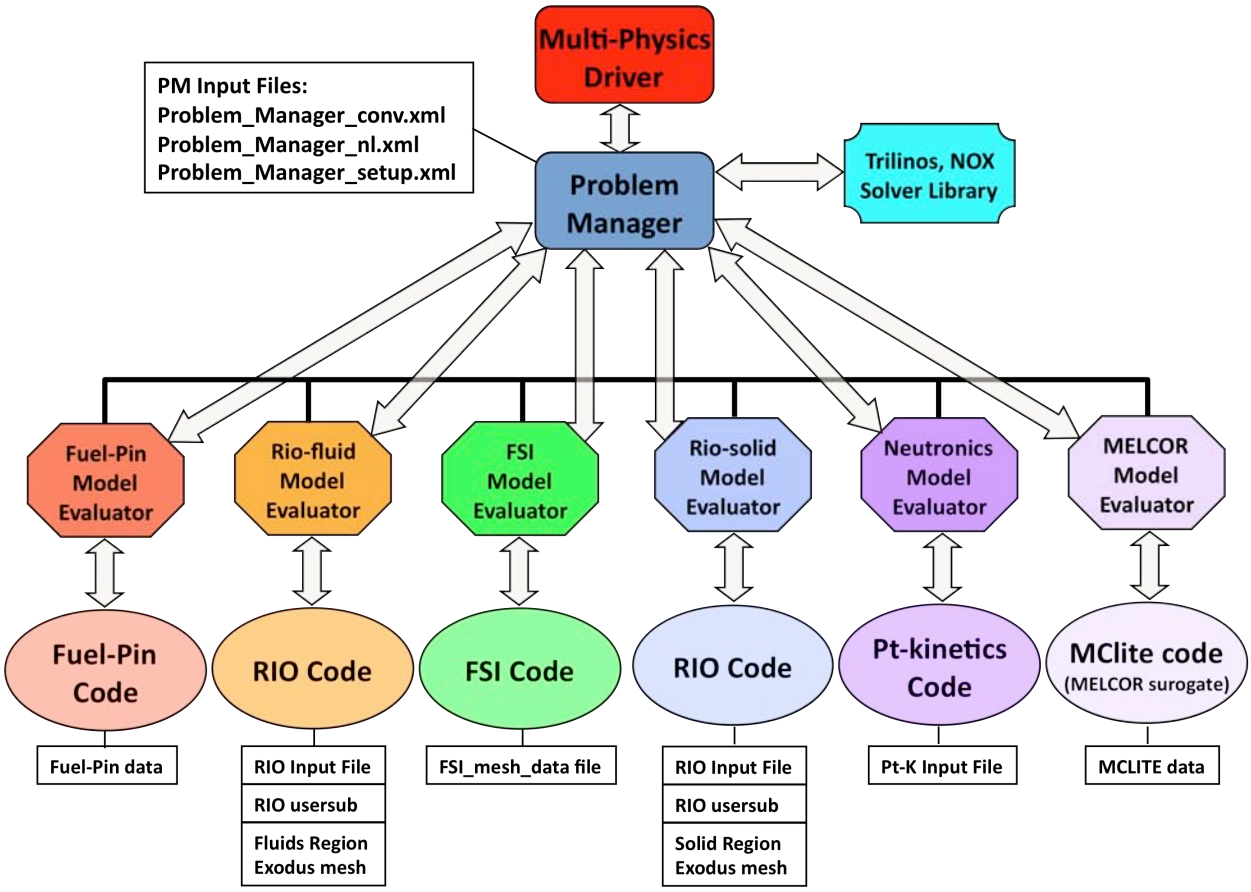


Figure 5.1 Schematic of software components making up the BRISC proto-type.

5.2 State variable types and data exchange requirements

Physics codes are typically designed to solve a set of equations that evolve values of certain "state variables" (e.g. temperature, density, velocity, neutron flux, etc.) in time over the spatial domain of relevance. However, a particular physics code may also be designed to provide output "answers" given some set of input data, with or without the concept of time. In any case, because these processes interact with one another, the equations must be solved in such a manner that changes in any state variable are properly reflected in all equation sets that are affected by that variable.

There are two types of state variables in BRISC: global state variables and local state variables.

Global state variables are owned and ultimately controlled by the problem manager. Each physics code designed to model the evolution of a global state variable must be able to compute a residual vector associated with that state variable. Each residual of these state variables contribute to the formation of a global residual vector, the magnitude of which determines whether the global solution is converged and therefore that the physics code solutions are fully coupled.

Local state variables are owned and controlled by an individual physics code, and are not known to the Problem Manager. Local state variables are typically associated with the use of the non-linear elimination strategy. In non-linear elimination, a collection of certain global state variables are provided to a physics code as input, and a different set of quantities are returned back as output. During any given time step, the output is entirely dependent on the input and is assumed to be an exact solution. Although local state variables are not known to the Problem Manager, they may be written by the physics code to an output file for plotting and visualization.

In the BRISC proto-type, non-linear elimination is used to apply the fuel pin, point kinetics and MELCOR (or MELCOR surrogate) codes. Therefore, each of these codes own local state variables that are not known by the Problem Manager. The FSI code is a special non-linear elimination case where its role is to compute boundary condition values for transfer between the RIO_fluid and RIO_solid codes. The RIO_fluid and RIO_solid codes are directly applied and the complete set of state variables in each code is owned by the Problem Manager.

Because the physics codes are coupled, various types of information, including state variable values, must be exchanged between physics codes. This is one of the central roles of the Model Evaluators associated with each physics code. Table 5-1 lists each code together with important characteristics of how it participates in BRISC and what its input and output variables are.

Table 5-1 Important characteristics of the physics codes coupled in BRISC

Physics Code	Participation Mode	Functionality	Input	Output
Fuel Pin	Nonlinear Elimination	• Predictor solution	• Fuel power • Fluid temperature • Pin-fluid heat transfer coefficient	• Clad temperature • Bulk average pin temperature
RIO-fluids	Full system Contributor	• Compute Residuals • Predictor solution • Precondition (JFNK) • Time-step control	• Global state variables: U, V, W, P, T, Ts • FSI temperature boundary condition vector • Pin-clad temperatures • Bulk avg pin temperature	• Residuals for U,V,W,P,T, and Ts equations • Predictor for U,V,W,P,T, and Ts • Preconditioner . . • Max. time step
RIO-solid	Full system Contributor	• Compute Residuals • Predictor solution • Precondition (JFNK) • Time-step control	• Global state variable Ts • FSI heat flux boundary condition vector	• Residuals for Ts equation • Predictor for Ts • Preconditioner . . • Max. time step
Pt. Kinetics	Nonlinear Elimination	• Predictor solution	• Avg. core inlet fluid T • Avg. core coolant T • Avg. core fuel T • Avg. core structure T	• Total reactor power
MELCOR or MClite	Nonlinear Elimination	• Predictor solution	• Volumetric heat flux	• Secondary system fluid temperature • Heat transfer coefficient
FSI code	Nonlinear Elimination	• Compute boundary conditions at FSI	• Fluid and solid region temperatures	• Temperature or heat flux at FSI

5.3 Setting-up and running a problem

To run a problem, the BRISC Multi-physics driver and Problem Manager must be configured through the following set of three XML input files;

Problem_Manager_setup.xml,

Problem_Manager_fp_conv.xml, and

Problem_Manager_nl.xml.

These files contain hierarchical specifications for configuring participating problems, associated data transfers, solver algorithms and convergence/stopping criteria. The following sections describe the various XML files and how to configure various problem and solver options. All XML files used for a given run are archived in a subdirectory, ”ParamLists”, located and created if need be within the local run directory. The files are stamped with the date and time of the run.

5.3.1 Description of input file Problem_Manager_setup.xml

The MPdriver is configured using an XML file named ”Problem_Manager_setup.xml” located in the local run directory. The file contains blocks for each physics code being coupled as well as

blocks for each associated data transfer. Top level settings for the coupling solve algorithm are also specified in this file. Figure 5.2 provides a representative example of a setup XML file.

```
<ParameterList>

  <ParameterList name="Physics">
    <!-- required input arguments -->
    <Parameter name="Name" type="string" value="Rio"/>
    <Parameter name="Instance" type="string" value="Fluid"/>
    <Parameter name="Input File" type="string" value="quads.i"/>
    <Parameter name="Input Mesh File" type="string" value="mph_quads.g"/>
    <!-- optional input arguments -->
    <Parameter name="Output Mesh File" type="string" value="quads.e"/>
  </ParameterList>

  <Parameter name="Use Predictor" type="bool" value="true"/>
  <Parameter name="Solver Strategy" type="string" value="JFNK"/>

</ParameterList>
```

Figure 5.2 Example Problem_Manager_setup.xml file.

The specifications in the ”Physics” block configure an instance of the RIO application for use as a fluids physics component and associate a RIO input file, input mesh file and exodus output file. This is the only physics involved in this multiphysics simulation. The remaining two lines at the top hierarchical level configure the MPdriver to use Newton-based coupling via Jacobian-Free Newton- Krylov and to precede the coupling solve with computation of a predicted solution for each physics application solved standalone (no coupling). For this particular single-physics scenario, the predictor step simply invokes RIO’s native solve algorithm followed by invocation of the JFNK algorithm to further drive convergence to a solution for each time step in the simulation.

A richer multiphysics and algorithmic scenario is represented in Figures 5.3 and 5.4, respectively. Figure 5.3 presents three physics instantiations comprising a Rio fluids component coupled to a Rio solids component via a non-conformal mesh-tying component. Two data transfers closing the dependencies follow the physics specification blocks. Figure 3 shows enhanced coupling solver options. Here, a SWITCHING coupling solver type is indicated along with a corresponding switching solver con- figuration block. Within the configuration block, stages for either fixed-point or JFNK coupling solves are defined. Configuration of fixed-point and JFNK coupling solvers currently are defined as they would be for single algorithm fixed-point or JFNK coupling. This means that fixed-point is configured using the file Problem_Manager_fp_conv.xml, and JFNK is configured using the two files, Problem_Manager_nl.xml and Problem_Manager_conv.xml, all of which are described later.

All specifications valid for Problem_Manager_setup.xml are summarized in Figure 5.5.

```
<ParameterList>

  <!-- define physics packages here -->

  <ParameterList name="Physics">
    <!-- required input arguments -->
    <Parameter name="Name" type="string" value="Rio"/>
    <Parameter name="Instance" type="string" value="Fluid"/>
    <Parameter name="Input File" type="string" value="MP_B3D.i"/>
    <Parameter name="Input Mesh File" type="string" value="rio_3d_fluid.g"/>

    <!-- optional input arguments -->
    <Parameter name="Output Mesh File" type="string" value="MP_B3D.e"/>
  </ParameterList>

  <ParameterList name="Physics">
    <!-- required input arguments -->
    <Parameter name="Name" type="string" value="Rio"/>
    <Parameter name="Instance" type="string" value="Solid"/>
    <Parameter name="Input File" type="string" value="MP_B3DS.i"/>
    <Parameter name="Input Mesh File" type="string" value="rio_3d_solid.g"/>

    <!-- optional input arguments -->
    <Parameter name="Output Mesh File" type="string" value="MP_B3DS.e"/>
  </ParameterList>

  <ParameterList name="Physics">
    <Parameter name="Name" type="string" value="Fsi"/>
    <Parameter name="Instance" type="string" value="1"/>
    <Parameter name="Input File" type="string" value="FSI_mesh_data"/>
    <Parameter name="Fluid" type="string" value="Rio_Fluid"/>
    <Parameter name="Solid" type="string" value="Rio_Solid"/>
  </ParameterList>

  <!-- define transfer operators here -->

  <ParameterList name="Transfer">
    <Parameter name="Name" type="string" value="Transfer_Rio_Fluid_To_Solid"/>
    <Parameter name="Instance" type="string" value="1"/>
    <Parameter name="Type" type="string" value="Post"/>
    <Parameter name="Source" type="string" value="Rio_Fluid"/>
    <Parameter name="Target" type="string" value="Rio_Solid"/>
    <Parameter name="Physics" type="string" value="Fsi_1"/>
  </ParameterList>

  <ParameterList name="Transfer">
    <Parameter name="Name" type="string" value="Transfer_Rio_Solid_To_Fluid"/>
    <Parameter name="Instance" type="string" value="2"/>
    <Parameter name="Type" type="string" value="Pre"/>
    <Parameter name="Source" type="string" value="Rio_Solid"/>
    <Parameter name="Target" type="string" value="Rio_Fluid"/>
    <Parameter name="Physics" type="string" value="Fsi_1"/>
  </ParameterList>
```

Figure 5.3 Example Problem_Manager_setup.xml file, Part 1 - Physics configuration.

```
  <!-- define solver settings here -->

  <Parameter name="Use Predictor" type="bool" value="true"/>
  <Parameter name="Solver Strategy" type="string" value="SWITCHING"/>

  <ParameterList name="Switching Solver">
    <Parameter name="Number of Stages" type="int" value="2"/>
    <ParameterList name="Stage 0">
      <Parameter name="Solver Type" type="string" value="Fixed-Point"/>
      <!-- settings for Fixed-Point currently come from the file,
           Problem_Manager_fp_conv.xml but could be added here. -->
    </ParameterList>
    <ParameterList name="Stage 1">
      <Parameter name="Solver Type" type="string" value="JFNK"/>
      <!-- settings for JFNK currently come from the files,
           Problem_Manager_nl.xml and Problem_Manager_conv.xml
           but certain reconfigurable options could be specified here. -->
    </ParameterList>
  </ParameterList>

</ParameterList>
```

Figure 5.4 Example Problem_Manager_setup.xml file, Part 2 - Solver configuration

5.3.2 Description of input file Problem Manager fp conv.xml

Coupling solves employing a fixed-point algorithm are configured from the file Problem_Manager_fp_conv.xml, which currently defines stopping and/or convergence criteria as shown in Figure 5.6. This file is also used when the Switching coupling solve is active and at least one of the solver stages uses fixed-point. Convergence is achieved of the L2-norm of the overall coupled problem residual vector is below the ”Absolute Tolerance” value, and iterations are limited to the number specified by ”Maximum Iterations”. It should be noted that each problem being coupled is solved sequentially and can use either the native application nonlinear solver or an instance of NOX for its stand-alone solve. In case of the latter, the NOX solver for each problem can be configured by two correspondingly named XML files. This is in the process of being reworked such that the XML files needed to configure a stand-alone NOX solver can be specified in the Physics configuration block for a given application.

5.3.3 JFNK Configuration

Newton-based coupling is configured using two XML files, Problem_Manager_nl.xml and Problem_Manager_conv.xml. Both of these files conform to hierarchical semantics directly relevant to the NOX nonlinear solver package in Trilinos. The file Problem_Manager_nl.xml can contain any of the parameter settings NOX recognizes. The most complete listing can be found on the NOX homepage.

The second XML file, Problem_Manager_conv.xml, conforms to the hierarchical structure required by the NOX StatusTest factory, with details provided again on the NOX doc page.

```
</ParameterList>

  <ParameterList name="Physics">
    <!-- Physics specs tailored to an application -->
  </ParameterList>

  <ParameterList name="Transfer">
    <!-- Transfer specs tailored to data flow between two applications -->
  </ParameterList>

  <Parameter name="Use Predictor" type="bool" value="true"/>
  <Parameter name="Solver Strategy" type="string"
             value="[ SWITCHING | FIXED-POINT | JFNK ]"/>

  <ParameterList name="Switching Solver">
    <Parameter name="Number of Stages" type="int" value="[ int ]"/>
    <ParameterList name="Stage 0">
      <Parameter name="Solver Type" type="string"
                 value="[ FIXED-POINT | JFNK ]"/>
    </ParameterList>
      <!-- additional stage specs -->
    </ParameterList>
  </ParameterList>

  <Parameter name="Jacobian Operator" type="string"
             value="[ Matrix-Free | Finite-Difference ]"/>
  <ParameterList name="Matrix-Free">
    <Parameter name="Constant Perturbation Value" type="double" value="[ double ]"/>
  </ParameterList>

  <Parameter name="Preconditioner Operator" type="string"
             value="[ None | Physics-Based | Finite-Difference ]"/>

</ParameterList>
```

Figure 5.5 Complete options for Problem_Manager_setup.xml file.

```
<ParameterList>
  <Parameter name="Maximum Iterations" type="int" value="[ int ]"/>
  <Parameter name="Absolute Tolerance" type="double" value="[ double ]"/>
</ParameterList>
```

Figure 5.6 Complete options for Problem_Manager_fp_conv.xml file.

5.4 Test and demonstration calculations

A large number of different small test problems were used during the development of BRISC to exercise various aspects of the individual codes and the multi-physics coupling. Many of these are included in a suite of regression test problems that were used to verify that any new changes did not adversely affect the solution of previously solved test problems. These small regression test problems are not reviewed or described here. Instead, this section describes the application of BRISC to a reactor accident transient problem called an unprotected loss-of-flow (ULOF) accident.

In an ULOF accident, the plant experiences a complete loss of power (including emergency power supplies) to the reactor cooling system while the plant is at full power. The power generation plant immediately ceases operation and provides no heat rejection capacity. Finally, one assumes complete failure of the reactor safety system to scram the reactor and no operator actions to mitigate any aspect of the accident.

Section 3 previously described the 3D geometric modeling and meshing issues associated with representing the complex geometry of the ABR. Within this geometric setting, there are many other aspects of the reactor system that must be modeled to perform a simulation. Figure 5.7 illustrates the different element blocks that are defined within the 3D fluid region mesh so that different regions can be uniquely modeled to account for different materials, different subgrid characteristics, and different physics couplings that are associated with different regions in the reactor.

Material properties, models and correlations applied to the test ULOF accident problem are reviewed in section 5.4.1.

Section 5.4.2 describes the application of BRISC to a 2D representation of the ABR test reactor undergoing an ULOF accident where the complete accident scenario was simulated during the calculation. Because a 2D simulation is of only modest computational expense, this could be accomplished on a single processor workstation. Section 5.4.3 describes the application of BRISC to the same problem but with a full 3D representation of the reactor. Because parallel issues were not fully resolved before the end of this project, a complete simulation of the 3D reactor problem was not performed and only a few seconds of simulation time were performed on a serial machine.

5.4.1 Properties, Models and Correlations

To solve a real reactor problem, material properties, component models (e.g. pump characteristics, etc.), and fluid flow and heat transfer correlations are required as user input to the various physics codes either through input files or through the user-written subroutines and functions. This section describes the representative material properties, models and correlations

that have been applied within BRISC for the ULOF accident problem to account for the different characteristics and different processes that are occurring in different regions in the reactor vessel.

Figures 5.7 and 5.8 are referenced in the descriptions provided to more clearly define the location where various models and correlations are to be applied.

It is important to note that our purpose in doing the test problems was to exercise the ability of BRISC to treat the many different regions in uniquely different ways. However, little attention was given to assure that the properties, component models, or correlations were the "best" available, only that they were reasonably representative. For example, all material properties were specified as constant values, independent of temperature. If a real reactor accident simulation were to be performed, all of the properties, component models, and correlations would need to be replaced with more sophisticated and accurate values or formulations.

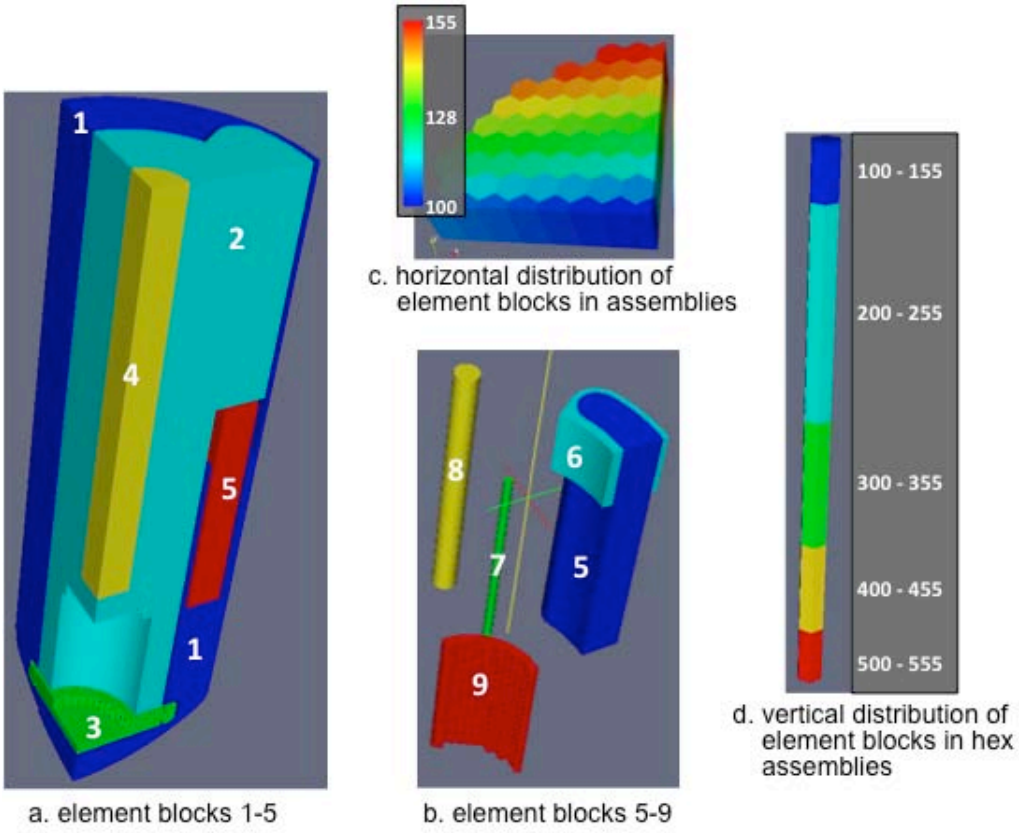
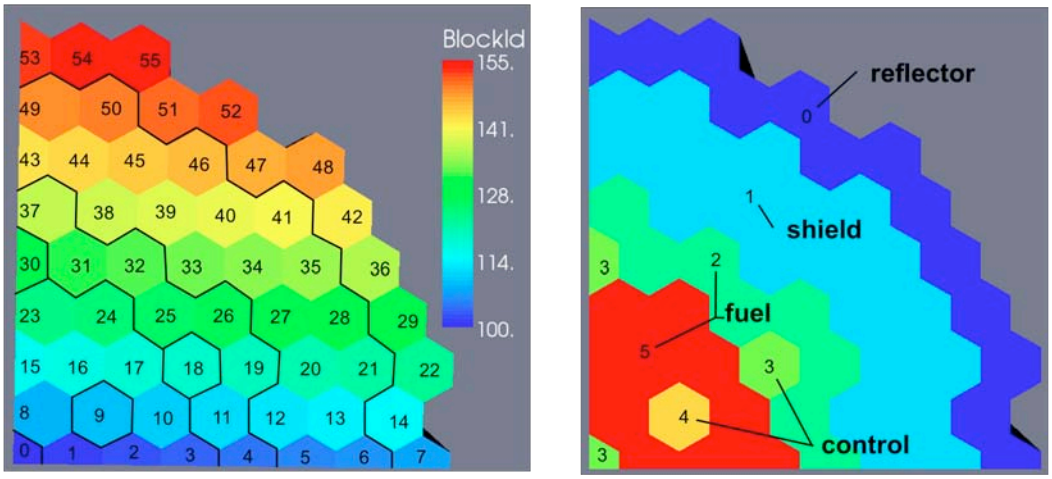


Figure 5.7 Illustration of element blocks defined within the 3D fluid region mesh.



a. Colored by element block ID b. Colored by assembly type

Figure 5.8 Cross section of the core region showing how element blocks are assigned to different assembly types in the 3D core region model

5.4.1.1 Pin-model parameters

The following list defines the key dimensions and properties used in the pin-model for the reactor test problems.

outer radius of fuel pellet	R_1 = .00348 m
inner radius of clad	R_2 = .00348 m
outer radius of clad	R_3 = .00400 m
density of fuel pellet	ρ_{pel} = 7875 kg/m^3 (note: this equals 10500 * 0.75)
specific heat of fuel	Cp_{pel} = 300 J/(kg K)
thermal conductivity of fuel	k_{pel} = 15 W/(m K)
density of clad	ρ_{cld} = 7800 kg/m^3
specific heat of clad	Cp_{cld} = 420 J/(kg K)
thermal conductivity of clad	k_{cld} = 30 W/(m K)
gap heat transfer coefficient	h_{gap} = 11.5e6 W/(m^2 K)

Using the above values, a volume average pin density "ρ_{pin}" and mass averaged specific heat "Cp_{pin}" are calculated for use when coupling to the RIO_fluids code. Also, in the pin plenum region, the mass averaged specific heat of the pin in the plenum region "Cp_{pin_plnm}" is adjusted to account for the empty space.

5.4.1.2 Liquid sodium material properties

The following constant property values for liquid sodium were used in the RIO-fluid code for the reactor test problems (see Table H-3 in Reference [63], values at 644 K).

fluid density	ρ_f	= 860 kg/m^3
fluid viscosity	μ_f	= 2.83e-4, (4.00e-4) kg/(s m)
fluid thermal conductivity	k_f	= 73 W/(m K)
fluid specific heat	Cp_f	= 1300 J/(kg K)
fluid thermal expansion coef.	B_f	= 0.0003

5.4.1.3 Solid-phase material properties

In two-phase (fluid-solid) regions, the density and specific heat for the solid phase were specified depending on the region. The following table shows the constant properties using in the RIO_fluid code for the solid phase properties in these different regions. Note where the average pin density (ρ_{pin}) and average pin specific heats (Cp_{pin} , Cp_{pin_plnm}) from the pin model (Section 5.4.1.1) are used.

Table 5-2 Solid phase properties used in the RIO_fluid code in the reactor test problems

Property	Fuel rod shield	Fuel rod fuel	Fuel rod gas plenum	All other regions
Density (kg/m^3)	7800	ρ_{pin}	7800	7800
Specific heat (J/(kg K))	420	Cp_{pin}	Cp_{pin_plnm}	420

5.4.1.4 Solid structure material properties

The following constant property values were used for all solid structures modeled in the RIO-solid code for the reactor test problems.

structure density	ρ_s	= 1000 kg/m^3
structure specific heat	Cp_s	= 500 J/(kg K)
structure thermal conductivity	k_s	= 10 W/(m K)

5.4.1.5 Model parameters and coefficients

Table 5-3 defines the model parameters and coefficients used in various spatial regions of the 3D reactor model. A similar collection of model parameters and values was used in the 2D reactor model.

Table 5-3 Model parameters and coefficients used for the 3D reactor case

Region Description	Element Block(s)	φ	D_h (m)	K	F	$\dot{M}_z$	h_{fs}
Cold Pool	1	1.0	-	1.e20 ($\sim\infty$)	0	0	-
Hot Pool	2, 6	1.0	-	1.e20 ($\sim\infty$)	0	0	-
IHX heat exchanger	5	0.4	.0021	Eq. (5.1) C_K=96	Eq. (5.2) C_F=.06325	0	1.e5
Upper Internal Structure	4	0.5	.20	Eq. (5.6)	Eq. (5.2) C_F=.00115	0	1.e5
Lower Plenum	3	0.9	-	1e-5	0	0	1.e5
DRACS heat exchanger	8	0.6	.0021	Eq. (5.1) C_K=32	0	0	1.e5
Pump and Pipe	7	0.5	-	7e-9	0	Eq. (5.4)	1.e5
Assembly handling socket	100-155	0.5	.0068	Eq. (5.1) C_K=96	0	0	1.e5
Assembly nosepiece	500-555	0.5	.0068	Eq. (5.1) C_K=96	0	0	1.e5
Region between core barrel and assemblies	9	0.4	-	1e-11	0	0	1.e5
Shield assembly region	Fig. 5.8	0.827	-	1e-11	0	0	1.e5
Reflector assembly	Fig. 5.8	0.157	.0034	Eq. (5.1) C_K=9600	0	0	1.e5
Control rod assembly	Fig. 5.8	0.367	-	1e-11	0	0	1.e5
Fuel Pin Shielded	400-455	0.321	.0034	Eq. (5.1) C_K = 96	Eq. (5.2) C_F=.06325	0	Eq. (5.3) C_h=.115
Fuel Pin Fuel	300-355	0.321	.0034	Eq. (5.1) C_K = 96	Eq. (5.2) C_F=.06325	0	Eq. (5.3) C_h=.115
Fuel Pin Gas Plenum	200-255	0.321	.0034	Eq. (5.1) C_K = 96	Eq. (5.2) C_F=.06325	0	Eq. (5.3) C_h=.115

Table 5-3 Nomenclature:

φ	porosity	F	Forchheimer coefficient
D_h	hydraulic diameter	$\dot{M}_z$	vertical direction momentum source (pump)
K	permeability	h_{fs}	fluid-solid heat transfer coefficient

5.4.1.6 Flow and Heat Transfer and Correlations

The fluid flow and heat transfer correlations for computing the permeability, the Forchheimer coefficient, and the heat transfer coefficients in different spatial locations are listed here.

Permeability:

$$K = \frac{1}{C_K}\varphi^2 D_h^2 \tag{5.1}$$

Forchheimer coefficient:

$$F = C_F(\mathrm{Re})^{0.2} \tag{5.2}$$

Heat transfer coefficients:

$$h_{fs} = \frac{k_f}{D_h} C_h(\mathrm{Re})^{0.8}\,\mathrm{Pr}^{0.4} \tag{5.3}$$

5.4.1.5 Component behavior models

This subsection describes the simple models used for the primary system pump, the loss of heat sink through the intermediate heat exchanger, and the spatially varying porous flow characteristics of the upper internals region of the 3D reactor test problem.

Primary system Pump Model:

In the ULOF accident sequence a complete loss of power is assumed. As a result, the primary system pumps coast down over a finite period of time. In this analysis we assume that these pumps coast down over a 450 second period, and apply the following linear equation to control a momentum source in the pipe-pump region to act as a simple pump model.

if (t < 600 seconds) then

$$\dot{M}_z(t) = \min\left[1.0\ ,\ 1.0 - \frac{(t-150)}{450}\right] * C_{pump} \tag{5.4}$$

else

$$\dot{M}_z = 0.$$

endif

where the value of C_{pump} is adjusted so as to get the desired initial flow rate during the initial 150 seconds.

Heat sink Model:

When power to the reactor system is lost, forced flow in the secondary system is also lost. This loss of heat sink is modeled in BRISC by zeroing out the local heat transfer to the heat exchanger over a 100 second period. Thus, the local heat transfer to the heat exchanger in each control volume "i" in the heat exchanger region "$q_s(i)$" (see Equation (4.5.3) in Section 4) is modified as follows.

if (t < 250 seconds) then

$$q(t) = q_s(i) * \min\left[1.0\ ,\ 1.0 - \frac{(t-150)}{100}\right] \tag{5.5}$$

else

$$q(t) = 0.$$

endif

Upper internal region permeability model:

Figure 1.6 in Section 1 illustrates the large perforated wall structure that forms a boundary of the upper internal structure region. The following simplistic model for the permeability K was applied in BRISC to create a spatially varying porous flow region in the 3D test problem. A similar model (but only operating in 2D space) was used in the 2D test problem. The model is described here as a code fragment from the user subroutine that calculates the local permeability.

```
} else if(rID == UI_ID) {     /* Upper Int. Struct region                */
   K = 2.e-7;                 /* general value                           */
   x = params[0];             /* x coordinate of the CV                  */
   y = params[1];             /* y coordinate of the CV                  */
   z = params[2];             /* z coordinate of the CV                  */
   r = sqrt(x*x + z*z);       /* radial distance from centerline         */
   rcage = 0.50;              /* UI radius is 0.605 m in this mesh       */
   ymin = -0.50;              /* UI extends down to -0.68 m in this mesh. */

   if(y < ymin) {             /* bottom wall with holes in upper internal */
      K = 1.e-9 ;
   }
   if(r > rcage) {            /* side wall with  holes in upper internal  */
                              /* just a simple dummy model here          */
      dhole = 0.2;
      j1 = (y - ymin)/dhole;
      j2 = fmod(j1+1,2);
      if(j2) K = 2.e-10;
   }
 }
```

(5.6)

5.4.2 2D reactor test problem

Figure 5.9 illustrates the computational domain of the 2D axisymmetric reactor problem compared to a cross-section of the Argonne ABR preconceptual design. This problem contains models for all of the important reactor components at length/time scales that are close to the proper scale and under the proper conditions, but represents them in a simplified two-dimensional context. Because of this, the computational expense for performing this simulation is much smaller than for a full 3D simulation.

The mesh shown in Figure 5.9 is a composite of the fluids mesh (gray, 10,104 elements) and the solids mesh (blue, 696 elements). For each of these computation domains, boundary conditions must be specified. For all shared surfaces, an isothermal no-slip boundary condition is specified for the RIO_fluids code and a specified heat flux boundary condition is specified for the RIO_solid code. As explained in Section 4.6, each step of the overall solution is fully coupled so that these boundary conditions are consistent with one another. In the RIO_solid code the outer boundary condition for the reactor vessel (top, bottom, and side) is specified as T = 350 C.

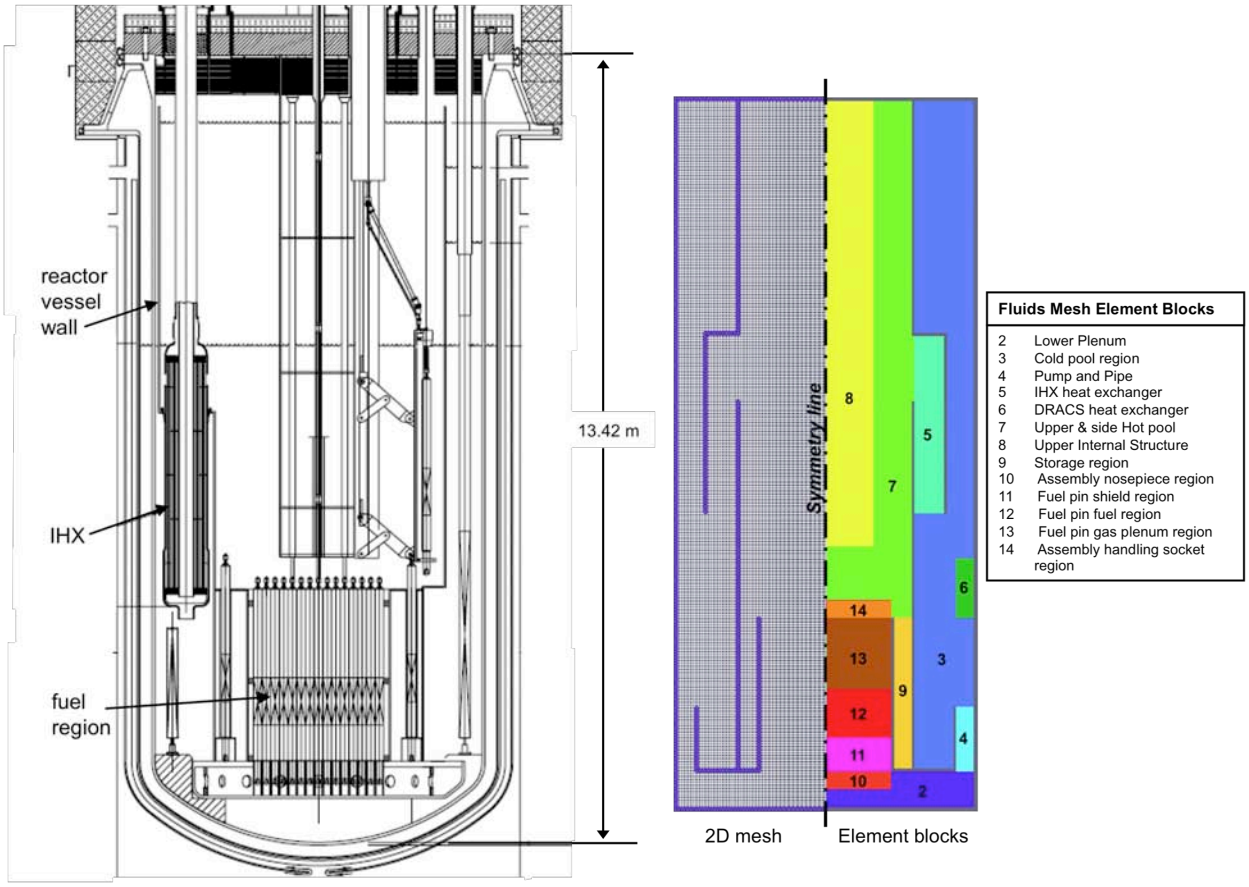


Figure 5.9 Simplified computational domain (10800 element unstructured mesh) of the 2D reactor problem compared to a cross-section of the Argonne ABR preconceptual design (see Figure II.2-1 in Ref. [5]). The blue mesh is the RIO_solid mesh, the gray is the RIO_fluid mesh.

To initialize the problem several conditions must be specified. For this test problem the initial temperatures of all reactor internal components and materials are initialized to 350 C. A reference reactor power of 1.e8 is set based on some guessed steady state values for the four input temperature required by the Pt. kinetics code. All velocities and pressures are initially zero. The overall simulation time for the problem is set to 1500 seconds.

Figure 5.10 plots several selected velocities in the reactor as a function of time. After the initial 2.5 minutes, the pump coast down period is clearly reflected in the pump, IHX, and central core region velocities. Because the code is being run in full transient mode large scale flow instabilities in the open hot pool region are resolved. These are primarily due to the large temperature gradients that are set up and the buoyancy driven natural convection mixing that is set up. The fluctuations in the lower hot pool velocity is a direct reflection of this.

Figures 5.11 and 5.12 plot selected temperatures and also the reactor power. These are usefully viewed together as they show the strong feedback that occurs as the reactor core temperatures rise. Seven different temperatures are plotted. Two of them are taken from the RIO_solids code output and show how these structures respond at two different locations to the heat transfer

occurring at their boundaries. Also shown are the differences between average and maximum temperatures in several key regions. Average temperatures are computed in the RIO_fluid code from solid and fluid phase control volume temperatures. The maximum fuel and maximum clad temperatures are computed in the pin_model. During the initial part of the transient the differences between maximum and average temperatures are significant. However, as the transient continues these differences become negligible in this simulation.

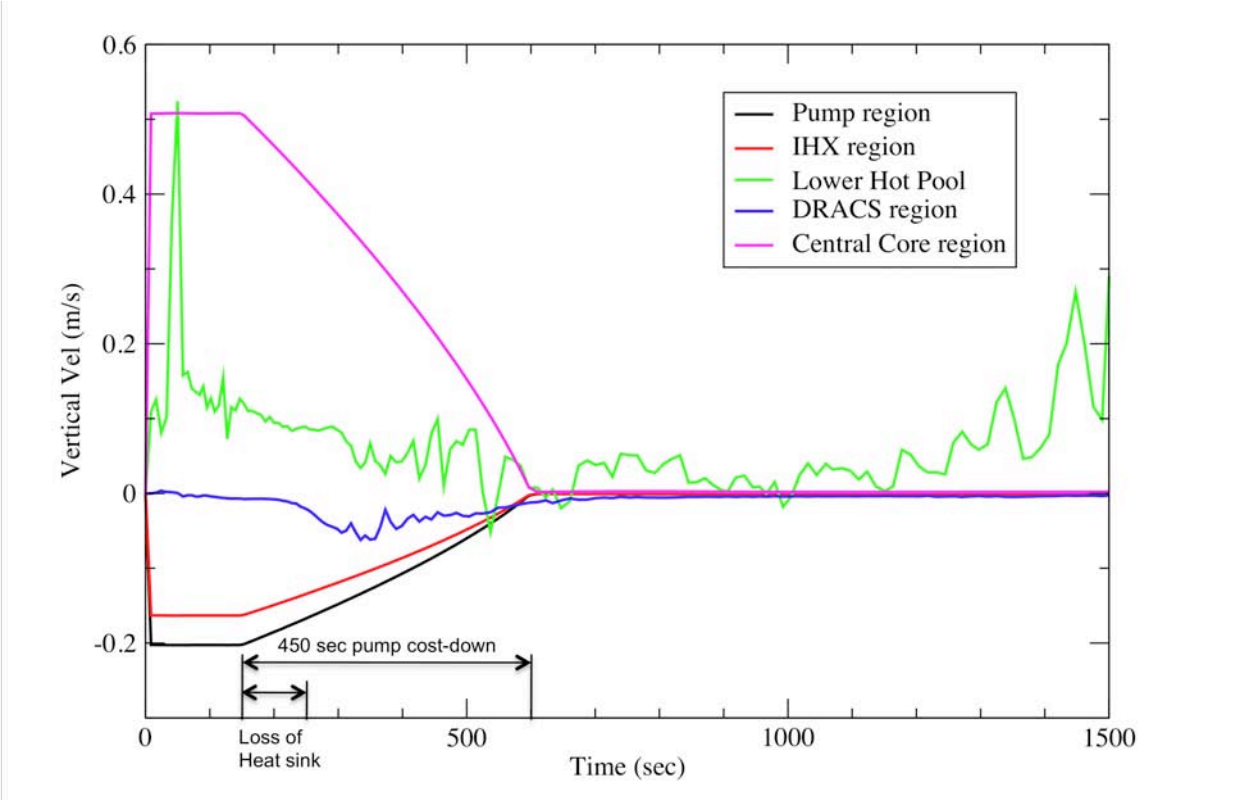


Figure 5.10 Selected vertical velocities as a function of time at various locations within the 2D reactor test problem domain.

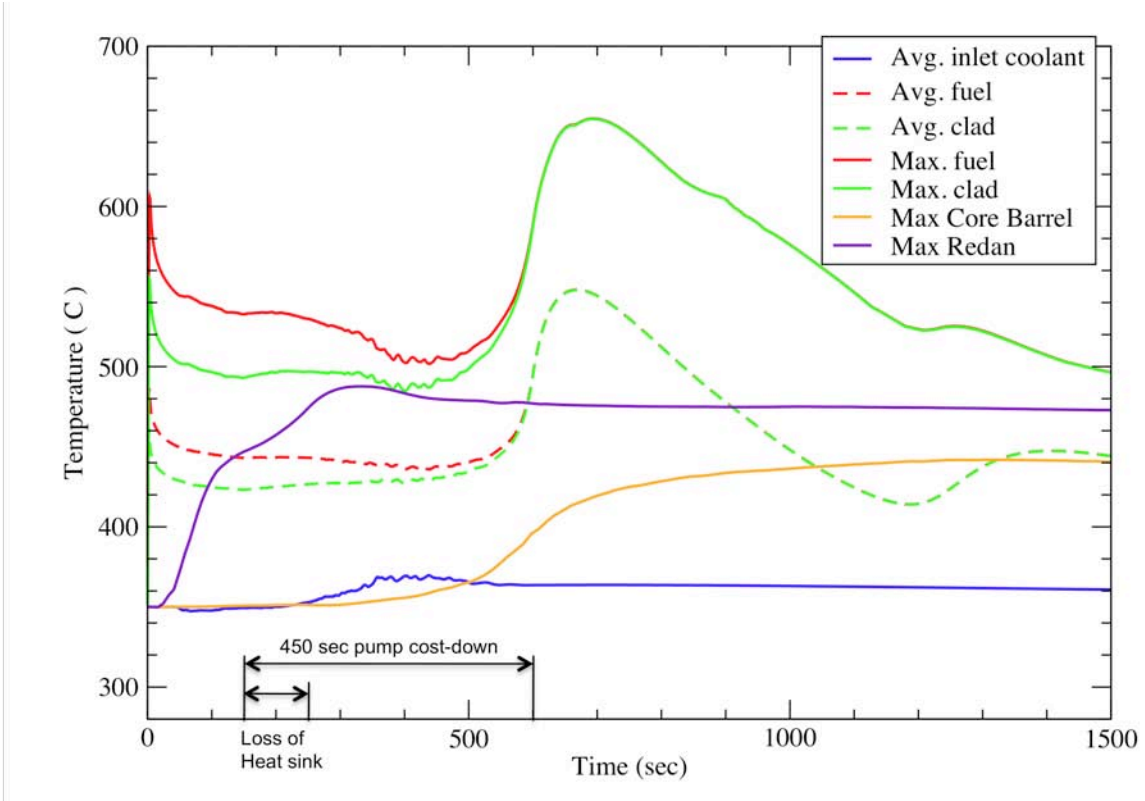


Figure 5.11 Selected temperatures as a function of time at various locations within the 2D reactor test problem domain.

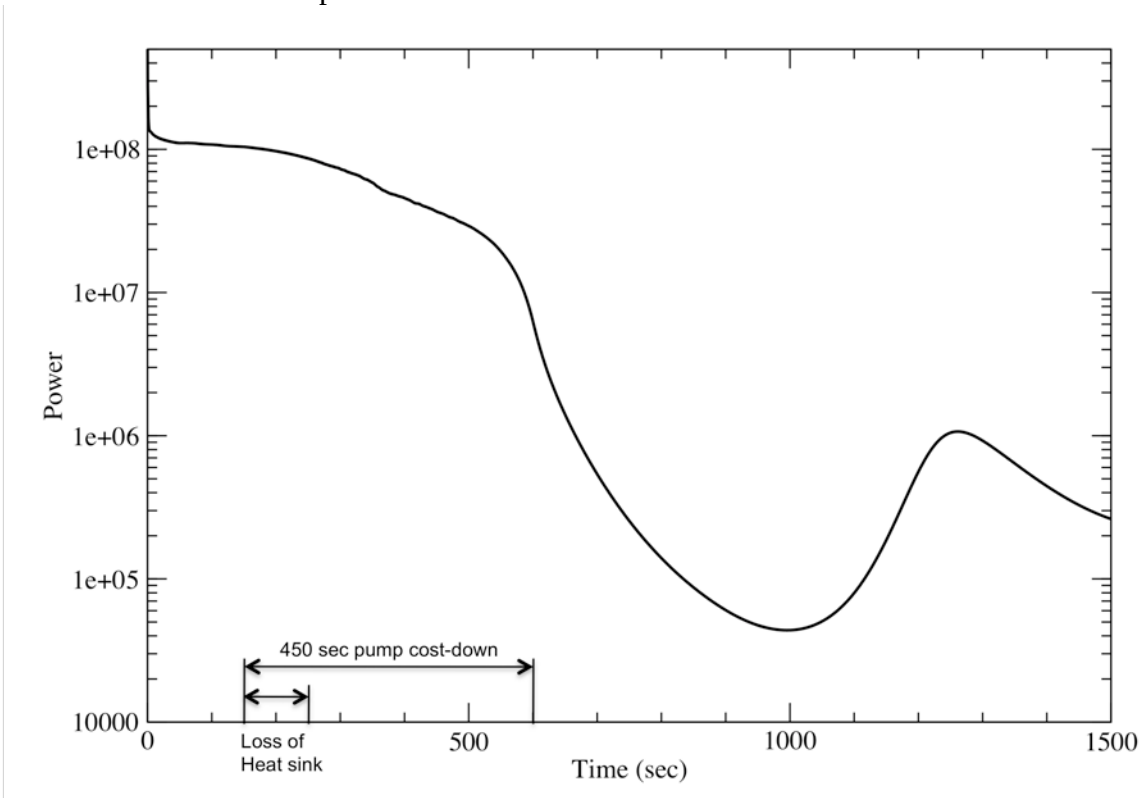


Figure 5.12 Reactor power as a function of time in the 2D reactor test problem

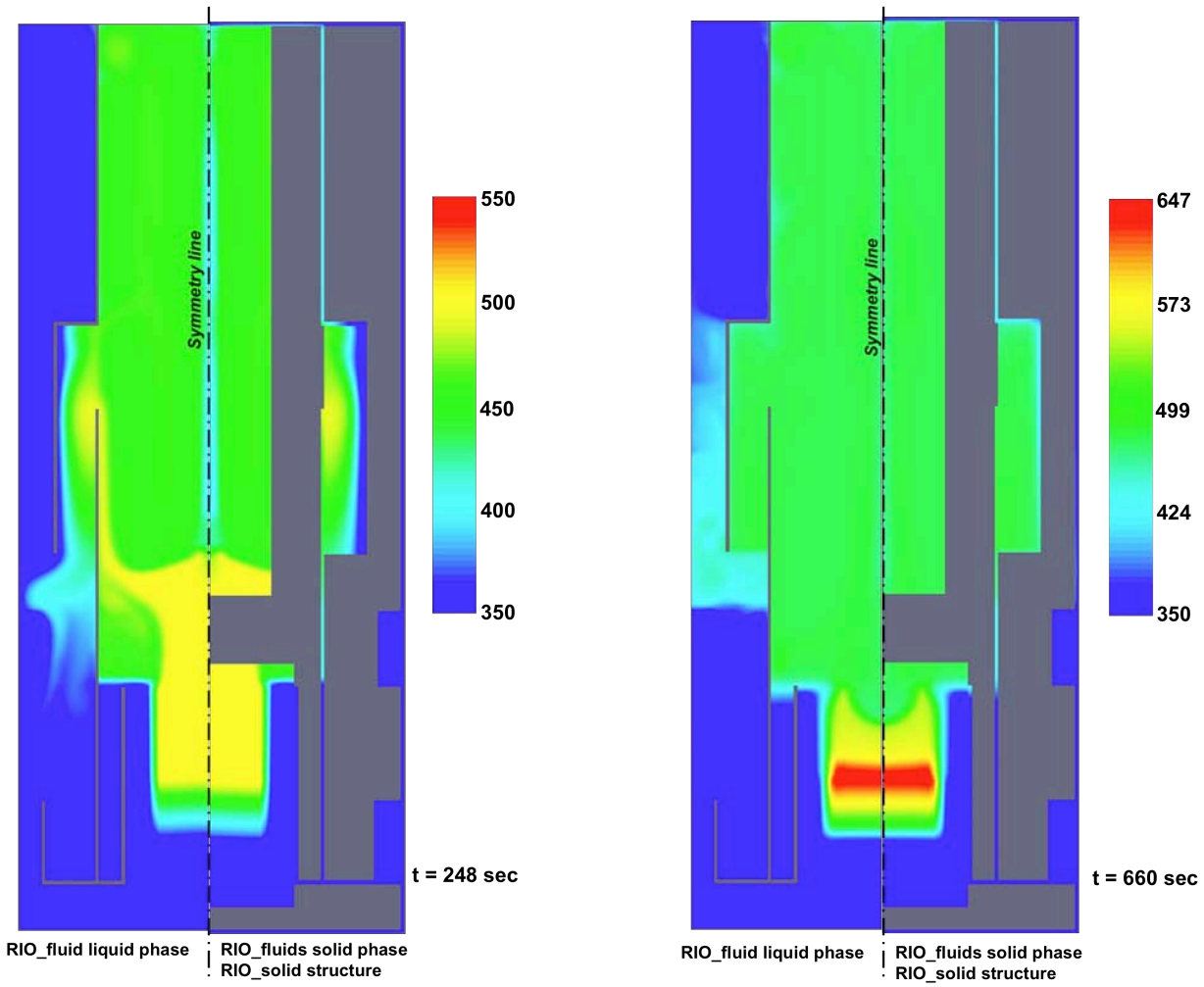


Figure 5.13 Temperature field at t = 248 and t=660 seconds. Fluid phase temperatures are on the left of the symmetry line. Solid phase temperatures and solid structure temperatures are on the right of the symmetry line. *Note that color scales are not the same for the two times.*

Visualization of the entire 2D domain at two different points in time is illustrated in Figure 5.13. Fluid phase temperatures are on the left and solid phase temperatures and the solid structure temperatures are on the right. Grey regions on the right correspond to regions where the porosity is 1.0 (i.e. no solid phase exists there). The first point in time corresponds to just after the heat sink is completely lost. The second point in time is when the peak transient temperatures are reached (see Figure. 5.11 above) shortly after the pump has completely coasted down. Because the heat transfer coefficients specified between the solid and fluid phases are large, the temperatures fields are very similar. However, the convective nature of the velocity field through the core is clearly reflected in the t=248 sec. temperature field before the pump coast down has completed.

Animations of the temperature field (such as shown in Fig. 5.13) as they evolve in time cannot be included in a report like this, but can be powerful tools that provide additional insight into the dynamic nature of the physical processes and interactions that are occurring.

Figure 5.14 is a plot of time step size and fixed point iterations as a function of time. Because the RIO_fluids code was run in fully transient mode with only its first-order time integration algorithm, a CFL-based time step restriction was imposed which restricted the time step to 75% of the CFL limit.

$$\Delta t_{\max} = 0.75 * \Delta t_{cfl} \tag{5.7}$$

Because of the CFL restriction, the time steps sizes taken during the simulation were very small, ranging from ~3e-2 sec to ~8e-2 sec. One advantage of the small time step size is the reduction in nonlinearity, thereby increasing the convergence rate. This is reflected in Figure 5.14 by the relatively small number of fixed point iterations required by the MP_driver to obtain convergence of the global solution. As can be seen, a typical step took from 2-4 iterations, with only occasional situations that took up to ~40 steps. Although a JFNK solution strategy was also possible for this problem, the additional cost of a single JFNK iteration was significantly more than a simple fixed point iteration when such small steps were being taken. As a result the JFNK approach was not used in this simulation.

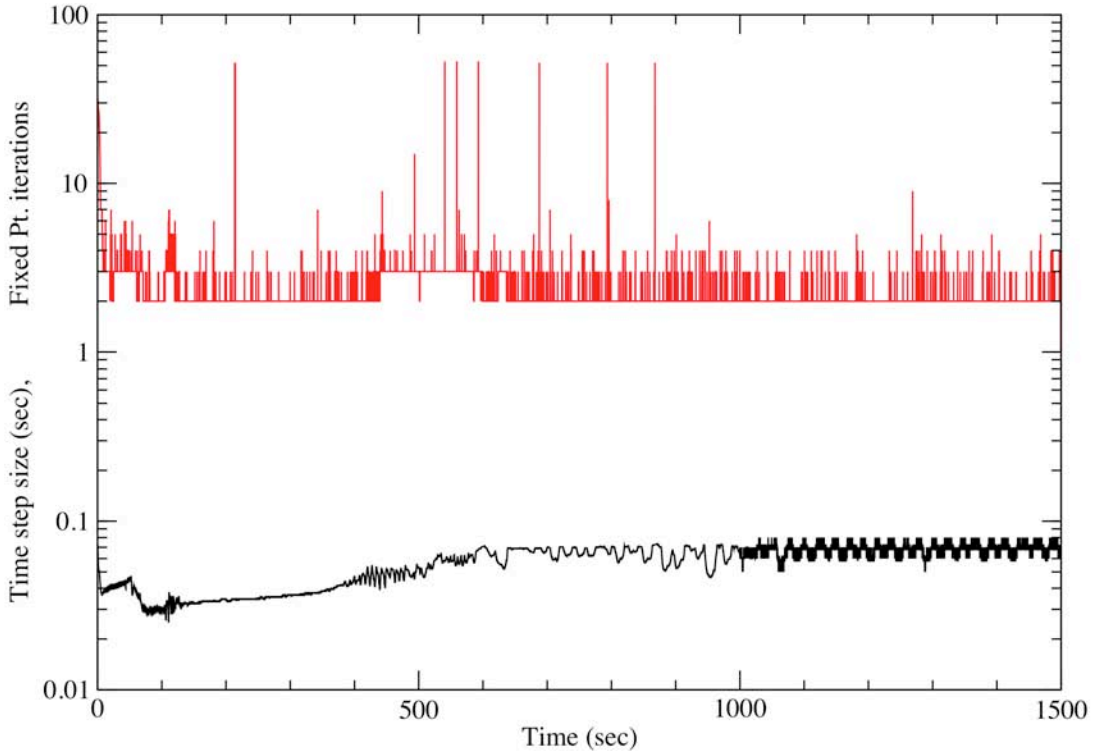


Figure 5.14 Time-step size and fixed point iterations as a function of simulation time for the 2D reactor test problem.

5.4.3 3D reactor test problem

A final objective of the LDRD project was to apply the BRISC code to a 3D representation of the Argonne ABR preconceptual design and perform a fully coupled parallel simulation of the target problem – an unprotected loss-of-flow (ULOF) accident scenario. This objective was not met for primarily one reason, current domain decomposition tools can only be applied separately (i.e. to the solid and fluid meshes independently) and do not address the complications arising from the FSI coupling across a shared interface. Although an approach for addressing this issue was defined, time and resources did not permit accomplishing this task. As a result, only serial calculations were performed in order to test that the simulation could proceed. But this simulation was too computationally expensive to do more than a few seconds of simulation time. In this section we briefly review the set-up of this 3D problem and present some initial results from the preliminary serial calculation.

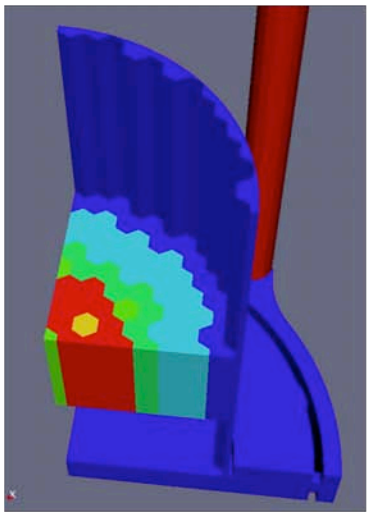
Figures 3.2 through 3.7 were discussed earlier in Section 3.2, and illustrate the fluid and solid region "meso-scale" meshes (and associated elements blocks) that were generated using CUBIT [17] for this problem. The unstructured fluids mesh contains 267,221 hexahedral elements and the solids mesh contains 21,942 hexahedral elements. Section 5.4.1 describes the representative material properties, models and correlations that were applied within BRISC to account for the different characteristics and different processes that are occurring in different regions represented on this mesh. An FSI interface mesh (see Section 3.3) was also generated to enable the coupling of these two physics regions. The FSI surface mesh had a total face count of 218,457 triangles. (Note that a merging process could have reduced this count number, but this was not performed.)

As with the 2D test problem, boundary conditions must be specified for both the fluid and solid region computation domains. For all shared surfaces, an isothermal boundary condition type is specified for the RIO_fluids code and a specified heat flux boundary condition type is specified for the RIO_solid code. As explained in Section 4.6, each step of the overall solution is fully coupled so that these boundary conditions are consistent with one another. In the RIO_solid code the outer thermal boundary conditions for the reactor vessel (top, bottom, and side) are T = 350 C. In the RIO_fluids code a no-slip velocity boundary condition is imposed on the fluid flow except on the inside of the pipe to the lower plenum (i.e. the pump-pipe region), which is modeled as a simple porous media with a momentum source. In both the RIO_fluids and RIO_solid codes the x and y symmetry planes are treated as symmetry type boundary conditions.

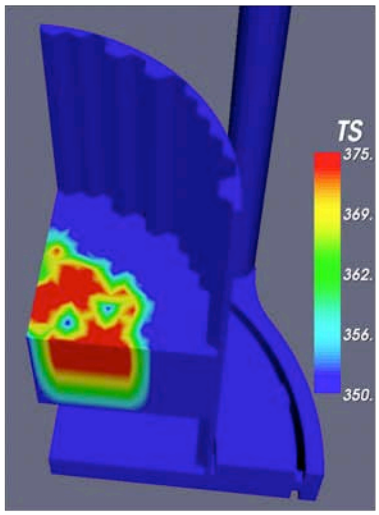
To initialize the problem several conditions must be specified. For this test problem the temperatures of all reactor internal components and materials are initialized to 350 C. A reference reactor power of 1.e8 is set based on some guessed steady state values for the four input temperature required by the Pt. kinetics code. All velocities and pressures are initially zero.

Visualizing the response of a complex 3D system is a challenging problem in and of itself. Figure 5.15 shows temperatures, velocities and pressures in selected reactor component regions as they respond during the initial few seconds of the transient initialization, and illustrate that the coupled system appears to be responding properly. For example, the 3D temperature response to the input power is evident in parts b. and c., where fuel and control rod assemblies are heating up differently. Part d shows the response of the vertical velocity W to the momentum source

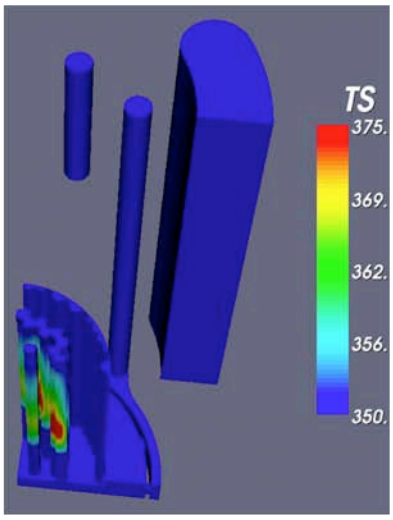
applied in the pump-pipe region. The magnitude of W is highest in the pipe (~ -3 m/s) where the coolant is pumped downward into the lower plenum. The coolant then passes through the inlet holes and upward into the assemblies. Flow velocities are greater in the fuel assemblies than in the control rod assembly, and almost zero in the bypass region next to the core barrel. A small downward velocity is also seen in the heat exchanger region. Part e. shows the corresponding pressure field, where the expected large pressure drop through the pump/pipe region is clearly visible. The pressure field is highest at the exit of the pump pipe, lowest at the entrance to the pipe, and varies between these two bounding extremes throughout the rest of the system.



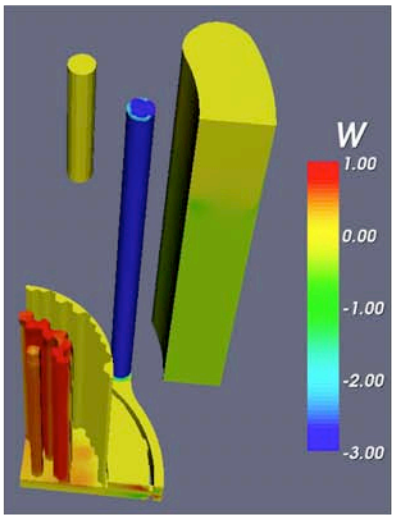
a. Core region element blocks grouped by assembly type.



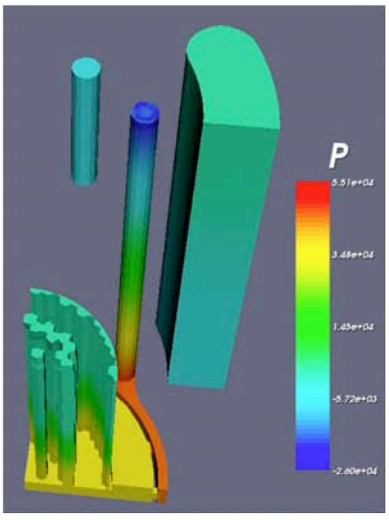
b. Temperatures in fueled regions responding to power input.



c. Axial variation of the temperature in selected fuel assemblies.



d. Vertical velocity field setting up as the coolant is pumped.



e. Pressure field setting up as the coolant is pumped.

Figure 5.15 A visualization of temperatures, velocities and pressures in selected reactor component regions as they respond during the initial few seconds of the transient initialization.

Because this test problem faithfully represents the major aspects of the real 3D geometry, both the simulation solution and the simulation requirements are very different than the 2D test problem. Even after only a few seconds of simulation time it is evident how the complexity of the geometry directly affects the variations in flow velocity and temperature fields that develop. For example, the flow velocities through an actual pipe are much higher than in a simple 2D axisymmetric channel, and the temperature variations in 3D (as evident in Figure 5.15 b and c) are clearly more complex. Also of note is that although the resolution scale was approximately the same, the numerical mesh element count went from approximately 11,000 elements in 2D to 290,000 in the 3D mesh (a 26X increase). With higher velocities, the CFL time step restrictions are much smaller, meaning that even if only large-eddy scale temporal fluctuations in velocity and temperature are being resolved, the computation cost is increased by another large factor. And finally, the cost of linear and nonlinear solutions generally does not scale linearly with the increase in unknowns, but increases more rapidly. These factors highlight both the benefits and the computation cost of performing simulations in 3D.

In summary, the initial preliminary results for the 3D reactor test problem have provided useful insights into the problem characteristics and costs, and appear to show that the different physics codes that are being coupled in BRISC are properly interacting. Additional insights and a more complete testing of the BRISC code will require addressing the remaining barriers to performing efficient simulations on high performance parallel machines.

6. CONCLUSION

6.1 Review and assessment

The main objective of this project was to develop and demonstrate foundational aspects of a next-generation systems-level nuclear reactor safety code. Thus one important aspect of the work was to leverage and apply new computational technologies such as advanced linear and non-linear solvers, improved numerical methods, modern meshing tools, parallel codes, and so forth. However, as we better understood the nature of the problem we were looking to solve, several other important challenges became equally important and also played a central role in guiding the decisions that were made during the development work. These included

- Creating a flexible and adaptable code framework that could incorporate and strongly couple multi-physics and multi-scale models and modeling constructs of differing fidelity and, potentially, from different sources (e.g. other National Laboratories or the nuclear industry).
- Creating a tool that was portable and sufficiently easy to apply that it could be used by organizations such as the NRC whose staff are not computer science experts.
- The need to demonstrate applicability to a real reactor safety problem on a full size nuclear power plant design.

The code development strategy was iterative in nature, and applied basic concepts from the "agile" code development world. After refining our understanding of LMR reactor safety physics and phenomena, a target reactor design (the Argonne ABTR) and target accident scenario (an unprotected loss of flow (ULOF) accident) were chosen to help focus the work. A three-cycle development plan was followed where each cycle involved the design, creation, and testing of an end-to-end simulation capability, and ended with a semi-formal review and evaluation.

The first iteration of BRISC was based on a 3-tiered multi-scale multi-physics modeling strategy, implemented and tested in a loosely coupled CFD-centric solution approach that used different codes for different physics. For thermal hydraulics we chose the RIO CFD code. RIO was modified to treat the Brinkmann Forchheimer porous media equations and also to enable loose (time-lagged) coupling to external models for the neutronics and balance of plant aspects of the problem. Initial efforts to model and mesh the 3D reactor domain were made and important insights gained concerning the geometric complexities that must be treated. Both 2D and 3D simulations of the target accident scenario were performed with the code we designated BRISC-alpha. Figure 6.1 illustrates the different components of the BRISC-alpha prototype and the central role that the thermal-hydraulics code played in loosely coupling these components during a time integration procedure.

A major design shift occurred after the first design cycle. While maintaining the 3-tiered multi-scale multi-physics modeling strategy for representing the reactor system, a very different

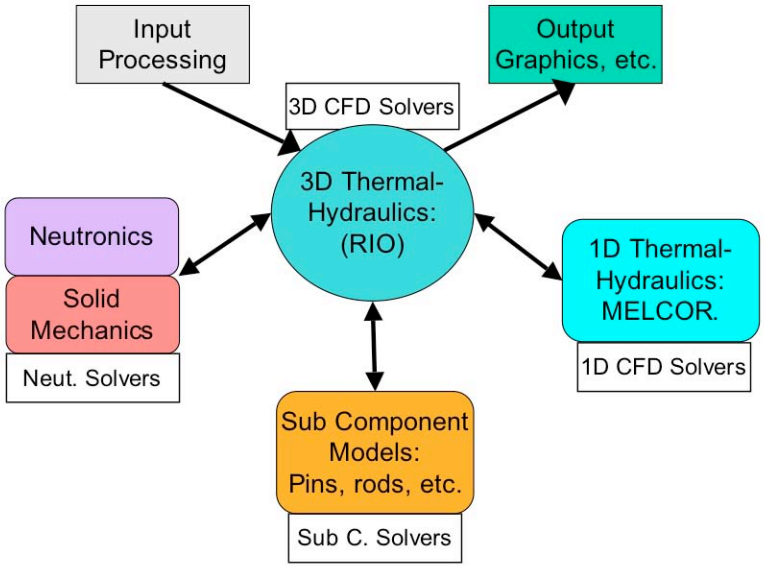


Figure 6.1 Illustration of the CFD-centric loose-coupling strategy used in BRISC-alpha

approach to coupling the different physics codes and models was envisioned for BRISC-beta. This approach was taken to enable strong coupling (e.g. JFNK approaches) of all of the physics codes, and was to be orchestrated by a Multi-Physics Driver/Solver Code. Figure 6.2 illustrates the initial concept of the overall code structure.

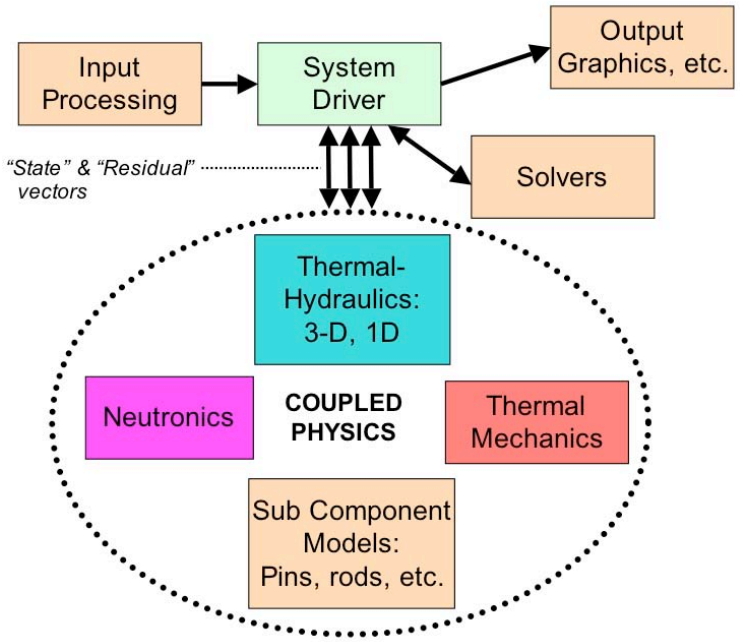


Figure 6.2 Illustration of the initially envisioned approach for how a multi-physics driver would be implemented in BRISC-beta to enable strong coupling.

Initially, an effort was made to develop the multi-physics driver software in Python. Python is a high-level, dynamic, object-oriented, interpreted programming language with characteristics that are amenable to this type of application. However, after a staffing change, the Multi-physics driver software was reformulated in C++ and the form of the high-level software components evolved as experience was gained during the testing of different strategies. As work progressed the Trilinos/NOX nonlinear solver library was leveraged so that all the nonlinear solver algorithms as well as associated interactions with linear solver and preconditioner packages in Trilinos could be employed. The end result of this continued development was the creation of what we now call LIME (for Lightweight Integrating Multi-physics Environment for coupling codes, see Section 2). The application of LIME to the creation of the final BRISC-proto code was illustrated earlier in Figure 5.1, and is reproduced here as Figure 6.3 for more convenient comparison with Figures 6.1 and 6.2.

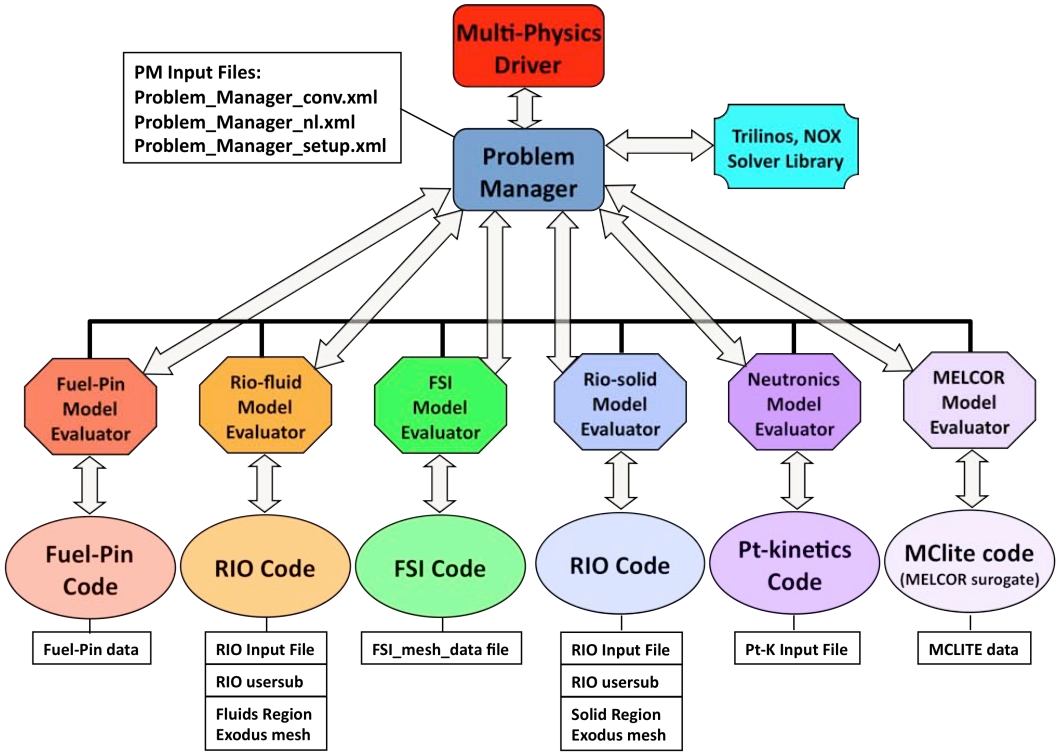


Figure 6.3 Illustration of software components making up the final BRISC proto-type code and coupled together using LIME.

As noted earlier, LIME by itself is not a code for doing multi-physics simulations. Instead, LIME provides the key high-level software, a well defined approach (now with example templates for Model Evaluators), and interface requirements for participating physics codes that enable the assembly of these codes into a new multi-physics simulation capability unique to the class of problems of interest for a particular need.

The BRISC prototype code for systems level nuclear reactor safety calculations is an example code built using LIME. In our estimation, the attributes of LIME, and its application to the

creation of the BRISC prototype code have successfully demonstrated the achievement of the primary objectives of this project, and are reflected in the following conclusions.

(1) The BRISC proto-type code clearly leverages advanced linear and non-linear solvers, improved numerical methods, modern meshing tools, and parallel codes.

(2) By building BRISC using LIME, we have demonstrated the ability of the LIME approach to

- enable both new and existing applications to be combined with strong coupling (when needed) though non-linear solution methods (e.g. JFNK, fixed point),
- preserve and leverage any specialized algorithms and/or functionality an application may provide,
- impose only a modest requirements barrier for an application to participate,
- work within advanced solver frameworks (e.g. as extensions to the Trilinos/NOX nonlinear solver libraries),

and also demonstrated that the LIME approach is not limited to:

- codes written in one particular language,
- a particular numerical discretization approach (e.g. Finite Element), or
- physical models expressed as PDE's.

(3) The main software components of LIME are relatively small in size and complexity and require only a few standard libraries to build (all openly available). LIME has been ported to several platforms during this project, and porting to a wide range of other computing platforms should not be a significant problem.

(4) The BRISC prototype code was successfully applied to the simulation of the full-scale target reactor design (the Argonne ABTR) and target accident scenario (an unprotected loss of flow accident).

Despite the above successes, not all of the desired objectives of the LDRD project were fully completed. Sections 6.2 and 6.3 highlight some of the lessons learned during the project and also some "unfinished business" that we might recommend as useful follow-on work.

6.2 Lessons Learned

We present here a few notes concerning lessons learned that were not discussed in the main body of the report, but which we feel are important to mention.

Note 1: One of the significant challenges to solving coupled multi-physics problems is that the state variables associated with different equation sets may have vastly different characteristic magnitudes. As a result, a well converged solution of one equation set might have a residual norm that is orders of magnitude different than the residual norm of one or more of the other equation sets when they are well converged. This can pose a serious problem when the convergence of the global solution is being based on the magnitude of a global residual norm. It also can dramatically degrade the ability of approximate Newton-based solution schemes, such as the JFNK method, to compute accurate updates during the solution process.

There are a number of strategies that can be employed to address this issue, but only two relatively simple ones are currently used in BRISC. The first of these is called variable scaling. In the variable scaling approach, a characteristic value of each state variable must be defined (which can be computed or user input). Before computing the residual for any equation associated with state variable "X", the equation is first non-dimensionalized based on the corresponding characteristic value for variable X. The scaled residuals computed in this manner are used in computing the scaled global residual norm. The second part is simply to require that the scaled residual norm computed for each of the individual equation sets associated with different physics, as well as the global residual norm, satisfy the convergence tolerance.

While developing and testing BRISC we learned that several of our test problems exhibited serious convergence problems without the application of these strategies. We believe that additional efforts to understand these types of issues and explore other methods to address them are clearly warranted in any future work.

Note 2: BRISC was exercised with two different nonlinear solution methods, JFNK and fixed point. In comparison, JFNK is significantly more expensive per update than fixed pt. But if good preconditioning is available and if accurate residuals can be computed, it provides a much more effective update vector, often yielding quadratic convergence rates. Thus there is trade off between computational cost and convergence rate when choosing between these two approaches. Initially, we generally expected that the JFNK method would prove the better approach to take. But experience with BRISC to date suggests that the simpler fixed point method can sometimes be significantly faster (in terms of computational cost) than JFNK, particularly when solving transient problems with small time steps. Thus both approaches were clearly valuable to have available.

Currently the Problem Manager can be directed to mix the different approaches according to some simple rules specified in the input. For the transient problems associated with the 2D and 3D target test problems, we found that using a small number of fixed point iterations at first, before resorting to JFNK, was a good strategy. We found that when small time steps were being taken, the fixed point algorithm along was sufficient to achieve the desired level of convergence in each step.

Note 3: During the development of BRISC, many different test problems were created and used to test various aspects of the code. Several of these were adopted for use in the regression suite of test problems used to assure that new modifications did not negatively affect the code. Exercising and checking the code with a range of test problems, and the faithful use of the regression test suite during the ongoing development process were invaluable to the success of the code development effort. We could not have progressed as far as we did without doing it.

6.3 Unfinished Business

As mentioned in Section 5.4.3, the problem of domain decomposition across the fluid-solid interface prevented a final objective (fully-coupled 3D simulation of a complete ULOF accident scenario) from being completed. This problem is that (to our knowledge) current domain decomposition tools can only be applied separately (i.e. to the solid and fluid meshes independently) and do not address the complications arising from the FSI coupling across a shared interface. Addressing this problem is the first issue that we would want to complete if we had additional time and resources to develop the BRISC prototype.

Parallel decomposition of meshes sharing one or more interface surfaces requires a tool capable of reading multiple mesh files and a list of the shared boundary side sets for each mesh, since the side set ids may not be the same in each mesh. With a description of the side sets to decompose, an internal mesh representation is constructed, and the boundary interfaces are partitioned so that each processor ends up with approximately the same number of boundary faces (ideally grouped so that the boundary faces are mostly adjacent to each other). The complexity is then to decide which elements attached to those boundary faces are put on which processor, and to write the fluid-solid interface mesh files consistent with this decomposition. Once a complete parallel decomposition has been formed, a mesh file is output containing the new decomposition along with field and other information contained in the input meshes. It can be convenient if the output meshes uses a different numbering scheme from the input meshes so additional output files may be helpful for mapping the mesh entities from the original mesh files to the new mesh decomposition.

Once the above issue was solved, we would suggest the following as important “next steps in developing the BRISC proto-type.

- Performing the 3D ULOF test problem over the entire 1500 sec. and using this as a basis for testing strategies for running efficiently in parallel.

- Incorporating and testing better Neutronics (2D and 3D) into BRISC. As noted in Section 5.4, only the relatively simple pt kinetics model was used in the test simulations. But Section 4.3 described the more generalized interface to the multi-physics driver that was designed to accommodate higher fidelity models of greater complexity. Our next steps would be to apply the 2D diffusion mode as explained in Section 4.3.4, and then move on the use of SCALE(TRITON) and NESTLE as explained in Section 4.3.5.

- Exploring the use of transient RANS turbulence models to remove the CFL time step restrictions. For many problems, resolving the large scale velocity fluctuations in the clear region of the reactor is not needed, and would allow for solutions that were significantly less expensive.

The usefulness of the basic approach taken in LIME to couple codes has, in our view, been demonstrated reasonably well in the work completed. However, there are many additional

aspects to applying LIME to create multi-physics applications that should be explored and developed. The following is a list of three potential candidates.

- Incorporating or linking tools to perform sensitivity studies and uncertainty quantification (e.g. DAKOTA). There are many challenging and important issues on how to best perform non-intrusive as well as intrusive methodologies into LIME.

- Rigorously exploring the issue of efficient parallelization in the LIME context. There are different ways that the Multi-physics driver could be structured to address running the different physics codes involved in the simulation when some or all of them are themselves being run in parallel, but each with different computing needs. Gaining a better understanding of these issues and providing for flexibility in adapting to the particular needs of a given simulation should be of great value.

- Additional work and testing of methods to improve robustness when state variables associated with different equation sets have vastly different characteristic magnitudes. This is follow-on to the discussion provided in note 1 of Section 6.2.

7. REFERENCES

1. http://www.gneppartnership.org/
2. R. O. Gauntt *et al.*, *MELCOR Computer Code Manuals*, NUREG/CR-6119, Vols. 1 and 2, Rev. 3, SAND2005-5713, Sandia National Laboratories, Albuquerque, New Mexico, September 2005.
3. J. E. Cahalan *et al.*, "Advanced LMR Safety Analysis Capabilities in the SASSYS-1 and SAS4A Computer Codes," *Proceedings of the International Topical Meeting on Advanced Reactors Safety*, American Nuclear Society, Pittsburgh, PA, April 17-21, 1994.
4. K. K. Murata *et al.*, *Code Manual for CONTAIN 2.0: A Computer Code for Nuclear Reactor Containment Analysis*, NUREG/CR-6533, SAND97-1735, Sandia National Laboratories, Albuquerque, New Mexico, 1997.
5. Y. I. Chang, P. J. Finck, C. Grandy *et al, Advanced Burner Test Reactor Preconceptual Design Report*, ANL-ABR-1 (ANL-AFCI-173), Argonne National Laboratory, Sept. 5, 2006.
6. H. Carter Edwards, *SIERRA Framework Version 3: Core Services Theory and Design*, SAND2002-3616, Sandia National Laboratories, Albuquerque, New Mexico, 2002.
7. D. Gaston, C. Newman, G. Hansen, and D. Lebrun-Grandié, "MOOSE: A parallel computational framework for coupled systems of nonlinear equations," *Nuclear Engineering and Design*, Vol. 239, pp 1768-1778, Oct. 2009.
8 R. W. Hooper, R. Schmidt, K. Belcourt, K. Clarno, "Development of a Lightweight Multi-Scale Multi-Physics Coupling Strategy for Nuclear Reactor Safety Simulations ," *2009 SIAM Conference on Computational Science and Engineering*, Mini Symposium 59, Miami, Florida, March 2-6, 2009.
9 Hooper,R., W. Spotz, A. Lorber, and R. Schmidt. "A Multi-Physics Coupling Approach for Integrated Nuclear Reactor Safety Calculations". *2008 American Nuclear Society Annual Meeting, Anaheim*, CA, June 2008.
10 R. Hooper, M. Hopkins, R. Pawlowski, B. Carnes, and H. Moffat. *Final report on LDRD project: Coupling strategies for multi-physics applications*, SAND2007-7146, Sandia National Laboratories, April 2007.
11 M. A. Heroux *et al.*, "An overview of the Trilinos project," *ACM Trans. Math. Softw.*, Vol. 31, No. 3, pp. 397-423, 2005.
12 http://trilinos.sandia.gov/
13 R. R. Pawlowski, J. N. Shadid, J. P. Simonis, *et al.* "Globalization techniques for Newton-Krylov methods and applications to the fully coupled solution of the Navier-Stokes equations," *SIAM REVIEW*, Vol. 48, Issue 4, pp. 700-721, 2006.
14 http://trilinos.sandia.gov/packages/nox/
15 D. A. Knoll and D. E. Keyes, "Jacobian-free Newton-Krylov methods: a survey of approaches and applications," *Journal of Computational Physics*, Vol. 193, pp. 357-397, 2004.
16. http://www.ptc.com/products/proengineer/
17 http://cubit.sandia.gov/
18. http://www.csar.illinois.edu/about/index.html

19. X. Jiao and M. T. Heath, "Overlaying surface meshes, Part I: Algorithms," I*nternational Journal of Computational Geometry & Applications*," Vol. 14, No. 6, pp 379-402, 2004.
20. X. Jiao and M. T. Heath, "Overlaying surface meshes, Part 2: Topology preservation and feature matching," *International Journal of Computational Geometry & Applications*, Vol. 14, No. 6, pp 379-402, 2004.
21. http://www.geomview.org/
22. http://www.msri.org/about/computing/docs/geomview/html/index.html
23. L. A. Schoof and V. R. Yarberry, *EXODUS II A Finite Element Data Model*, SAND92-2137, Sandia National Laboratories, 1994.

24 C. D. Moen, *RIO – Version 1.1 : Computational Fluid Dynamics and Heat Transfer*, Technical Report, Sandia National Laboratories, Livermore, CA, (in preparation)

25 *SIERRA/FUEGO 2.7 Theory Manual*, Technical report, Sandia National Laboratories, Albuquerque, NM, April 2008. (Unreleased Document).

26 S. P. Domino, C. D. Moen, S. P. Burns, and G. H. Evans, "SIERRA/FUEGO: A Multi-Mechanics Fire Environment Simulation Tool," *AIAA Paper 2003-0149*, 41st AIAA Aerospace Sciences Meeting, Reno, NV, January 2003.

27 W. S. Winters, *A New Approach to Modeling Fluid/Gas Flows in Networks*, Technical Report SAND2001–8422, Sandia National Laboratories, Livermore, CA, 2001.

28 W. S. Winters, C.E.Hickox, G.H.Evans, B.M.Rush, M.J.Martinez, D.K.Gartling, G.F. Homicz, and J. S. Grieco, *Detailed Modeling and Simulation of Contaminant Transport in Architectural Spaces*, Technical Report SAND2005–7465, Sandia National Laboratories, November 2005.

29 W. D. Gropp and E. Lusk. *User's Guide for mpich, a Portable Implementation of MPI*, Technical Report ANL-96/6, Mathematics and Computer Science Division, Argonne National Laboratory, 1996.

30 J. Dongarra, J. Du Croz, S. Hammarling, and R. J. Hanson. "An Extended Set of FORTRAN Basic Linear Algebra Subprograms". *ACM Transactions on Mathematical Software*, Vol. 14, pp. 1–17, March 1988.

31 J. J. Dongarra and G. W. Stewart. "LINPACK – A Package for Solving Linear Systems". in *Sources and Development of Mathematical Software*, W. R. Cowell, ed., pp. 20-48, Prentice-Hall, Upper Saddle River, NY, 1984.

32 E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, *LAPACK Users' Guide*, Society for Industrial and Applied Mathematics, Philadelphia, PA, third edition, 1999.

33 L. Betchen, A. G. Straatman, and B. E. Thompson, "A NonEquilibrium Finite-Volume Model for Conjugate Fluid/Porous/Solid Domains," *Numerical Heat Transfer, Part A*, Vol. 49, pp. 543-565, 2006.

34 P. Yu1, T. S. Lee, Y. Zeng and H. T. Low, "A numerical method for flows in porous and homogenous fluid domains coupled at the interface by stress jump," *Int. J. Numer. Meth. Fluids*, Vol. 53, pp. 1755–1775, 2007.

35 V.A.F. Costa , L. A. Oliveira, B. R. Baliga, and A. C. M. Sousa, "Simulation of coupled flows in adjacent porous and open domains using a control-volume finite-element method," *Numerical Heat Transfer A*, Vol 45, pp. 675–697, 2004.

36 B. Alazmi, K. Vafai, "Analysis of fluid flow and heat transfer interfacial conditions between a porous medium and a fluid layer," *International Journal of Heat and Mass Transfer*, Vol. 44, pp. 1735–1749, 2001.

37 C. T. Hsu and P. Cheng, "Thermal dispersion in a porous medium," *International Journal of Heat and Mass Transfer*, Vol. 33, pp. 1587–1597, 1990.

38 M. K. Moallemi and R. Viskanta, "Coolant Recirculation in a Pressurized Water Reactor Core under Loss-of-Coolant Accident Conditions," *Nucl. Science and Eng.*, Vol. 898, pp. 209-225, 1987.

39 R. C. Schmidt, "Analysis of Air Ingression Into the Core Region During Hypothetical PWR Shutdown Accidents", *12th Proceedings of Nuclear Thermal Hydraulics*, 1997 Winter Meeting, Albuquerque, NM, Nov. 16-20, pp. 95-106, 1997.

40 *SCALE: A Modular Code System for Performing Standardized Computer Analyses for Licensing Evaluations*, ORNL/TM-2005/39, Version 5.1, Vols. I-III, November 2006. Available from Radiation Safety Information Computational Center at Oak Ridge National Laboratory as CCC-732.

41 M. D. DeHart, *TRITON: A Two-Dimensional Transport and Depletion Module for Characterization of Spent Nuclear Fuel*, ORNL/TM-2005/39, Version 5, Vol. I, Book 3, Section T1, November 2006.

42 P. J. Turinsky, R. M. Al-Chalabi, P. Engrand, H. N. Sarsour, F. X. Faure and W. Guo, "Code Abstract - NESTLE: A Few-Group Neutron Diffusion Equation Solver Utilizing the Nodal Expansion Method for Eigenvalue, Adjoint, Fixed-Source Steady-State and Transient Problems," *Nuclear Science and Engineering*, **120**, 72, 1995.

43 R. A. Wigeland, "Effect of a detailed Radial Core Expansion Reactivity Feedback Model on ATWS Calculations Using SASSYS/SAS4A," *Trans. Am. Nucl. Soc.*, Vol. 53, p. 303, Nov.1986.

44 T. J. Moran , "Core Restraint contributions to Radial Expansion Reactivity," *Proc. 1986 ASME/ANS Nuclear Power Conf.*, Philadelphia, Penn., July 20-23, 1986, p 454, 1986.

45 R. A. Wigeland, "Comparison of the SASSYS/SAS4A Radial Core Expansion Reactivity Feedback Model and the Empirical Correlation for FFTF," *Trans. Am. Nucl. Soc.*, 55, p. 423, Nov. 1987.

46 T. J. Moran, "Core Restraint Design for Inherent Safety," *Proceedings Joint ASME/ANS Nuclear Power Conference*, Myrtle beach, South Carolina, Apr. 17-20, 1988.

47 R. Wigeland, T.J Moran, "Radial Core Expansion Reactivity Feedback in Advanced LMRs: Uncertainties and Their effects on inherent safety," *Proc. Conf. Safety of Next Generation Power Reactors*, Seattle, Washington, May 1-5, 1988, p. 879, American Nuclear Society, 1988.

48 D. J. Hill, R. A. Wigeland, "Validation of the SASSYS Core Radial Expansion Reactivity Feedback Model," *Trans. Am. Nucl. Soc.*, Vol. 56, p. 380, June 1988.

49 P. Royl *et. al.*, "Performance of Metal and Oxide Fuel Cores during Accidents in Large Liquid-Metal-Cooled Reactors," *Nuclear Technology*, Vol. 97. Feb 1992.

50 T. Yokoo, H. Ohta, "ULOF and UTOP Analysis of a Large Metal Fuel FBR Core using a Detailed Calculation System," *J. of Nuclear Science and Technology*, Vol. 38, No. 6, p. 444-452, June 2001.

51 A. E. Waltar and A. B. Reynolds, *Fast Breeder Reactors*, Pergamon Press, ISBN 0-08-025982-0, 1981.

52 G. L. Hofman and L. C. Walters, "Metallic Fast Reactor Fuels," in *Materials Science and Technology, A Comprehensive Treatment, Volume 10A*, ISBN3-527-26823-5, 1983.

53 F. A. Garner, "Irradiation Performance of Cladding and Structural Steels in Liquid Metal Reactors," in *Materials Science and Technology, A Comprehensive Treatment, Volume 10A*, ISBN3-527-26823-5, 1983.

54 H. Choi, and T. J. Downar, "Liquid-Metal Reactor for Burning Minor Actinides of Spent Light Water Reactor Fuel - II: Nuclear Data Uncertainty Analysis," *Nuclear Science and Engineering*, Vol. 133, No. 1, pp. 1-22, 1999.

55 W. S. Yang, "Summary of Neutronics Modeling Activities in FY 2007," *Presentation at the GNEP Annual Meeting*, Litchfield, AZ, October 4, 2007.

56 G. Aliberti, G. Palmiotti, M. Salvatores, and C. G. Stenberg, "Impact of Nuclear Data Uncertainties on Transmutation of Actinides in Accelerator-Driven Assemblies," *Nuclear Science and Engineering*, Vol. 146, pp. 1-12, 2004.

57 T. J. Downar, H. S. Khalil, "Uncertainty in the Burnup Reactivity Swing of Liquid-Metal Fast Reactors," *Nuclear Science and Engineering*, Vol. 109, 278, 1991.

58 H. G. Joo, D. Barber, G. Jiang, and T. Downar, *PARCS: a multidimensional two-group reactor kinetics code based on the nonlinear analytical nodal method*, School of Nuclear Engineering, Purdue University, July 2002

59 *RELAP5-3D Code Manuals*, Idaho National Engineering and Environment Laboratory, INEEL-EXT-98-00834, Revision 1b, 1999.

60 F. Odar, et al., *TRACE V4.0 User's Manual*, United States Nuclear Regulator Commission, 2005.

61 X. Jiao and M. T. Heath, "Common-refinement-based data transfer between non-matching meshes in multiphysics simulations," *Int. J. Numer. Meth. Engng*, Vol. 61, pp. 2402–2427, 2004.

62 Ankita Jain, *Efficient Parallel Algorithms for Overlaying Surface Meshes*, Masters Thesis, College of Computing, Georgia Institute of Technology, Aug. 2007.

63. F. Kreith and W. Z. Black, *Basic Heat Transfer*, Harper and Row, Publishers., Inc., New York, N.Y., ISBN 0-700-22518-8, 1980.

APPENDIX A: BUILDING, COMPILING, AND RUNNING BRISC

This appendix shows the steps needed to build and compile the various components of BRISC if they were installed at a generic location “your_computer/svn/BRISC/software,” where “your_computer” is an Apple Mac workstation with the INTEL compiler suite installed.

BRISC Multi-physics depends on Trilinos, MELCOR, RIO, Kinetics and Expat. Kinetics and MELCOR must be built anytime before building the c++ Multiphysics driver. It's important to build Trilinos before RIO, since RIO is now dependent on an existing Trilinos installation. Naturally, Multiphysics can't be built until the RIO library has been built.

A.1 Building Expat

Checkout Expat, the XML parser.

```
svn co http://your_computer/svn/BRISC/software/expat
```

To build expat:

```
cd expat
./configure CC=icc
make
```

A.2 Building Kinetics

Checkout revision 546 of kinetics (this revision is required or the multiphysics tests won't pass).

```
svn co -r 546 http://your_computer/svn/BRISC/software/kinetics_ornl_I/trunk kinetics
cd kinetics
```

To build the kinetics, nucdata and reactivity libraries, and run their tests:

```
./tools/make.darwin_intel
```

The build should be clean with no compile, link or test errors. The libraries are installed here.

```
./kinetics/obj/libkinetics.a
./nucdata/obj/libnucdata.a
./reactivity/obj/libreactivity.a
./rascal/rascal/librascal.a
```

A.3 Building MELCOR

To checkout MELCOR

```
svn co http://your_computer/svn/BRISC/software/Melcor Melcor
```

To build optimized MELCOR:

```
cd Melcor
./scripts/compile ./scripts/opt.cmd ./scripts/empty
```

To build debug MELCOR:

```
cd Melcor
./scripts/compile ./scripts/dbg.cmd ./scripts/empty
```

You should now have both the melgen and MELCOR executables. MELCOR is two programs, melgen and MELCOR. The melgen program does problem setup and writes the initial restart file. The MELCOR program marches the transient forward in time. Run this test to ensure things were built okay. If prompted to append or overwrite files, respond with 'O' to overwrite. MELCOR should run for 1000 cycles, writing a restart file at the last step, and should terminate cleanly (here's what the end of the calculation should look like).

```
./melgen test_Lnew.inp
./melcor test_Lnew.inp
....
 Restart written TIME= 9.50604E+02 CYCLE= 983
 CYCLE= 990 T= 9.576043E+02 DT(MAX)= 1.000000E+00 CPU= 3.552964E+01
 Restart written TIME= 9.67604E+02 CYCLE= 1000
 Listing written TIME= 9.67604E+02 CYCLE= 1000
```

Now that the executables are built, we must build the MELCOR library that we'll link into Multiphysics (libmelcor.dylib on the Mac). First you need to build the library, then verify the library was properly built. There should be no output from the build library step.

```
brisc:~/Melcor kbelco$ ./scripts/build_library scripts/dbg.cmd scripts/empty
brisc:~/Melcor kbelco$ file libmelcor.dylib
libmelcor.dylib: Mach-O 64-bit dynamically linked shared library x86_64
```

The Melcor library is now ready to be linked into Multiphysics.

A.4 Building Trilinos

First obtain a copy the Trilinos 9.0.1 [13] and place it into a directory on your machine.

```
cd /your_directory/trilinos-9.0.1
```

Make a directory to perform the build in (i.e. mkdir INTEL_BUILD), this will become the Trilinos install directory. cd into that directory. Run the following configuration step from the INTEL_BUILD directory

```
../configure \
  --prefix=${PWD} \
  --with-mpi-compilers=/usr/local/mpi/openmpi-1.3.3/intel-10.1/bin \
  --with-mpi=/usr/local/mpi/openmpi-1.3.3/intel-10.1 \
  --enable-aztecoo \
  --enable-aztecoo-teuchos \
  --enable-zoltan --enable-ml \
  --enable-amesos \
  --enable-epetraext \
  --enable-nox \
  --enable-nox-epetra \
  --enable-nox-tests \
  --disable-loca \
  --cache-file=config.cache \
  --enable-export-makefiles \
  RANLIB="ranlib -s" \
  --with-gnumake \
  CXX=icpc \
  CC=icc \
  F77=ifort \
  CXXFLAGS=-g \
  CFLAGS=-g \
  FFLAGS=-g

make
make install
```

Now we have to install some extra NOX utilities required by Multiphysics.

```
cd packages/nox/test/utils
make
make install
```

At this point Trilinos should be configured, built, and installed.

A.5 Building RIO

Checkout RIO from subversion:

```
svn co http://your_computer/svn/BRISC/software/Rio/rio/trunk rio
cd rio
```

Edit buildRio and change the --with-trilinos path to point to your Trilinos installation (e.g. the path in buildRio for Brisc points to my Trilinos installation).

```
./buildRio
```

Following the RIO configuration step, edit the RIO Makefile in src/Makefile, and add some new dependencies on Trilinos libraries (epetraext and ifpack) that Chris Moen's copy of RIO does not require. Replace LIBAZTEC as shown here.

```
-LIBAZTEC=-laztecoo -lepetra -lteuchos #-lepetraext
+LIBAZTEC=-laztecoo -lepetraext -lepetra -lifpack -lteuchos #-lepetraext
```

Next go back to the top level directory and build Rio.

```
make install
```

A valid RIO executable should be produced by the make install step. The next step is to build RIO as a library because Multiphysics links against the Rio library. We need to make a few changes to the RIO source and Makefile.

Edit src/parallel.cpp and comment out the MPI_Init() call (lines 117 and 118), since it's an error to call MPI_Init() twice, and we'll be doing the initialization in Multiphysics. Edit src/rio.cpp, change line 49 to #if 0. This will comment out the RIO main, a necessary step to link RIO into Multiphysics. Edit RIO's src/Makefile, add rio.cpp and usersub.cpp to the CXXSRC variable. Change to the RIO src directory and run make, the RIO library should be updated, but now the RIO executable will fail to link, since there's no main. Ignore the error if it's just the link error for the RIO executable. If the library was built, you are ready to proceed to building Multiphysics.

A.6 Building Multiphysics

Checkout multiphysics:

```
svn co http://your_computer/svn/BRISC/software/MultiPhysics/trunk multiphysics
cd multiphysics/c++_driver
```

Edit the Makefile and update the path to the Trilinos, Kinetics and Rio installation directories. The Makefile variables are TRILINOS_INSTALL_DIR, KINETICS_INSTALL_DIR, and RIO_INSTALL_DIR. Change these to point to your own copy of the Trilinos, Kinetics and Rio installations.

```
make clean
make
make testing
```

The testing target will automatically run several simple regression test problems.

APPENDIX B: CREATING MODEL EVALUATORS FOR COUPLED COMPONENT CODES

Model evaluators are the interface code between the problem manager that drives the simulation and the physics packages that implement the coupled physics capabilities. Model evaluators encapsulate, or hide, the underlying physics package behind the interface code and the problem manager only interacts with the physics packages through this interface. This encapsulation helps decouple the underlying physics package that implements specific capabilities from the more general capabilities that the problem manager knows about.

There are a number of capabilities that a physics package might support, such as computing a residual or predictor, time stepping (integration), variable scaling and data transfer operations between model evaluators, to name a few. Not all physics packages support or require all these capabilities, so the model evaluator interface is structured to impose as few constraints as possible on a physics package that wants to participate in a coupled physics simulation.

In creating the model evaluator interface, our goals were to make it (1) fairly lightweight, and (2) easy to implement just the capabilities a physics package supports. In the next sections we describe the model evaluator programming interface (API), the broad classification of programming interfaces into capabilities known to the problem manager, and how to create model evaluators for use in coupled physics simulations.

The model evaluator interface design is object oriented in nature, providing a base class with numerous virtual functions for each of the capabilities known to the problem manager. The model evaluator base class, called problem_manager_api, exposes interfaces that a model evaluator may wish to implement and that are typically only called by the problem manager itself. Normally each model evaluator implementing a physics capability will create a new class that inherits from the problem_manager_api base class, and implement the virtual interfaces corresponding to the code's capabilities. None of the interfaces are abstract, meaning that a model evaluator should implement only those capabilities that the underlying physics package supports. Broadly classified, the core capabilities are

- time stepping,
- ability to reinitialize model evaluators to "constructor" state,
- ability to register degrees of freedom for both input and output variables to compute sensitivities used in JFNK solver,
- residual and predictor support,
- ability to perform standalone solve,
- support for problem manager notifying physics code of converged solution and time step,
- variable scaling, and
- physics-based preconditioning.

There are two other base classes in addition to the model evaluator base class. The first is a data transfer base class called Data_Transfer_Operator (DTO) class. The second is an elimination module base class called Elimination_Module (EM) class. Both the DTO and the EM base

classes contain a single abstract function that you must implement when you derive from these classes.

When writing a model evaluator to wrap a physics package, it's important to consider what type of coupled problem is to be run. For example,

- Is the problem integrated in time or does it simply run some fixed number of steps and terminate?
- Does this model evaluator need to exchange data with other model evaluators and, if so, when during the solution process does this exchange need to occur?

It is important to recognize that some codes can advance the solution state separately from the call to compute the solution, permitting multiple calls to solve within a step (such as is done when solving the global problem using a JFNK method).

The physics associated with a given model evaluator can either be computed in a separate computer code (C, Fortran, etc…) or it can be coded directly into the model evaluator class itself. If the physics is a separate code, we recommend that you compile the code into an archive library and link it into BRISC. Creating the archive library may require code changes to the physics code such as

- exposing data and functions that the model evaluator needs to access,
- removing or redirecting all console input and output,
- having a coherent error reporting mechanism that doesn't call exit, abort, stop, pause, etc.

This last point is important in that physics packages that terminate without exiting through the problem manager driver code will likely hang a parallel computation.

It is possible for multi-language (e.g. C and Fortran) input and output to be used in the same program without problems, but care and attention is required here if, for example, you create multiple instances of a physics code through separate model evaluator classes, and each model evaluator instance reads or writes files. Another key concern is whether the physics package is parallel aware (MPI, Threads, OpenMP). If so, additional work may be required to, for example, pass MPI communicators from the problem manager driver code into the physics code. Of related concern is a package that runs threads directly or through OpenMP, as this can cause severe system overloading and lead to performance problems.

Some codes may expect access to the command line arguments and these may not be readily available to the physics packages. In some cases it may be preferable to pass required arguments directly into the physics package through a custom interface rather than working around potential Iargs / Nargs problems. As part of calling a separate physics code from a model evaluator you must declare as external the functions and data you wish to call. That is, because the model evaluator classes are C++, you must declare C and Fortran constructs you wish to access through an extern "C" {function and data declaration list } interface, typically declared at the top of the model evaluator source (not header) file.

Our experience with constructing model evaluators leads us to recommend that developers start with wrapping a single physics package inside a model evaluator. Our file naming convention during this project was physics_model_evaluator.[hc]pp, where "physics" is replaced with the physics package name. For example, our fluid mechanics code is called Rio so we named the

model evaluator rio_model_evaluator.[hc]pp. Once the model evaluator is successfully linked into BRISC, you can add your time stepping interfaces and hook them up to your physics code so that the ProblemManager can drive the model evaluator through the simulation in time. Our fluid mechanics code, Rio, is C++ so we didn't need to declare anything external as we simply constructed a Rio object, initialized that object in the Rio model evaluator constructor, and accessed Rio field data and functions directly off that object.[1]

In general we suggest starting with a simple problem with no other dependencies and a known good output result so that you can compare the result of a standalone run of the physics package outside of BRISC with the result produced by driving the physics package with BRISC. We found that computing numerically sensitive quantities and outputting these for comparison enabled us to ensure BRISC was replicating standalone behavior and helped us quickly identify changes that broke the code as soon as they occurred.

Once a single standalone code is running correctly, a second code can be added, again wrapped in an appropriate model evaluator. It is then useful to run both codes standalone and inside of BRISC, comparing the results of both runs. The second code is chosen so that once the two individual model evaluators are running correctly a data transfer from one code to the other can be added. Data transfers can be done automatically by the problem manager through a registration process, or directly between the explicitly cooperating codes without involving the driver. Model evaluators are free to choose whichever transfer type works best for their particular case. With code coupling it can be easiest to initially use a custom transfer capability to debug the transfer (i.e. to determine when the transfer must occur, what data is exchanged between the packages, whether the data transfer is bidirectional, etc…). BRISC supports data transfers with the DTO class and data elimination with the EM class. An example of an elimination transfer can be seen in the Rio Kinetics coupling. In this case Rio transfers temperatures to the Kinetics package where Kinetics takes the temperatures and eliminates them returning the reactor power back to Rio.

[1] Because we used Rio for both the fluid and solid mechanics problems, we created two separate instances of the rio_model_evaluator, one for the fluid problem and one for the solid problem, and initialized them accordingly. We mention this to show how customization of the model evaluators is possible and not to imply that all coupled problems will require this level of functionality.

APPENDIX C: RESULTS OF A PRELIMINARY SCOPING PIRT MEETING FOR ABR SAFETY ANALYSIS

On 9 Nov. 2006, a group of Sandia staff spent most of a day in a meeting focused on defining and discussing the physics and physical processes important for the safety analysis of future advanced burner reactors (ABRs). A list of participants is given at the end of this appendix. Its purpose was to provide initial guidance and perspective to the code and model development activities of the LDRD project described in the main body of this report. These results were very useful as a starting point, but are and were considered preliminary and subject to correction and later revision.

The discussions were organized around seeking answers to the following questions.

1. What are the key "Figures of Merit" for ABR Safety?
2. What are the important accident scenarios that must be evaluated for ABR safety?
3. What are the physical phenomena and processes that must be modeled in order to accurately simulate each of the important ABR accident scenarios?
4. How well are each of the important physical phenomena and processes understood, and how good are currently available models?

An important result of this meeting was the creation of a preliminary set of tables, similar to what are sometimes called "Phenomena/Process Identification and Ranking Tables" (PIRT), in which the important physical processes and physics that were identified are listed, together with some assessment as to their level of importance and the adequacy of current understanding and modeling.

This appendix summarizes key results and comments from this meeting and documents the tables generated in these discussions. It also contains some additional comments deemed important by various participants based on thoughts that occurred during or after the meeting. It is organized into sections that correspond to the different questions discussed.

C.1 Questions addressed

C.1.1 Question 1

What are the key Figures of Merit for ABR Safety?

The discussion revealed a range of perspectives. Clearly the key figures of merit depend on the setting and/or context in which the safety questions are being asked (regulatory, design, etc.) The following seven Figures-of-Merit were each deemed to be important in some context.

1 Radiological Consequence: to the Public, Workers, Environment
2 Source Term: The timing, magnitude, and composition of radionuclide release.
3 Containment Failure: The timing and type of failure mechanism.
4 Reactor Vessel Failure: The timing and type of failure mechanism.

5 Core Damage: The timing and extent of any physical and/or chemical damage to core-region components.
6 Margin-to-Damage: Some measure (e.g. temperature?) of how close the system came to a damage state.
7 Safety System Effectiveness: Some measure of how effective the safety systems operated.

C.1.2 Question 2

What are the important accident scenarios that must be evaluated for ABR safety?

The discussion of this question began by reviewing the potential causes leading to accident scenarios or damage events. Four were identified.

1 Mechanical and/or Electrical System Failures
2 Natural Disasters (e.g. earthquake, tornado, fire, . . .)
3 Man-made external events (e.g. airplane crash, ….)
4 Operator errors or Terrorist Acts (i.e. accidental actions or deliberate sabotage)

A range of specific accident scenarios that have been considered in previous work was reviewed, including those mentioned in Appendix C references [1-3]. There was some discussion about whether terrorist activities would require evaluating new scenarios that were not effectively covered by more traditional accident initiating events. Most (but perhaps not all) seemed to feel that the generic accident scenarios would cover all conditions that needed to be considered.

Among the various scenarios that were discussed, the following accident scenarios were considered to cover the full range of conditions that would need to be evaluated. It may be that this list could be further pruned and still remain comprehensive.

1 Unprotected loss-of-flow (ULOF): Complete loss of power (including emergency power supplies) to the reactor cooling system while the plant is at full power. The power generation plant immediately ceases operation and provides no heat rejection capacity. Failure of the reactor safety system to scram the reactor and no operator actions.
2 UTOP - Unprotected Transient Overpower: Assume that the worst case control rod withdrawal event occurs. Assume that all control rods remain full out (at the mechanical stops) for some specified period of time and then the reactor is scrammed. All externally powered cooling is lost at the time the control rods are withdrawn.
3 Station Blackout: Loss of all AC power for an extended period of time.
4 Flow Blockage: Complete blockage of flow to one or more fuel assemblies. Also, local power-cooling mismatch.
5 Large Sodium (Na) leaks: Sudden large leaks in the intermediate heat transport system piping.
6 Dynamic Structural loading: Like an Earthquake
7 Fire: Either external or internal (e.g. electrical fire) to the containment.

C.3 Questions 3 and 4

What are the physical phenomena and processes that must be modeled in order to accurately simulate each of the important ABR accident scenarios?

How well are each of the important physical phenomena and processes understood, and how good are currently available models?

This section reflects the results of the discussions for both of these questions because the comments and discussion were so closely connected, and also because the tables generated provide responses to both questions.

We began by discussing alternative ways of categorizing the different phenomena. It was agreed that the following five categories were adequate for the purposes of the meeting.

1 Neutronics
2 Fluid Flow and Heat Transport
3 Material Interactions & Chemistry
4 Material Properties and Equation of State
5 Structural Mechanics and Dynamics, Relocation

However, it was noted that these processes are not independent and that strong coupling can exist between phenomena listed in different groups.

The rest of the meeting was devoted to deciding what physical processes, phenomena, or material properties were sufficiently important to be listed in each of the five major categories listed above. This occurred through discussion, debate, explanations, short presentations, and so forth among the participating staff. The end product was a collection of five tables, listed below, that attempt to summarize the subjective opinions of the working group participants.

These tables were created in the following manner. As we discussed the physics and modeling issues in each category, an effort was made to assign an importance ranking to each process, phenomena, or material property that was deemed of some significance. This ranking was based on the subjective experience of the participating staff, and was chosen from a small list of choices: High (H), Medium (M), Low (L), Uncertain (U) or not applicable (na). The definitions for each of these importance ranking levels are listed below in Table C.1. However, as will be seen in the resulting tables, the importance ranking results achieved in this effort tended towards assigning many more Highs than Medium or Lows, reflecting the limitations of this effort and the fact that items were generally believed to either be quite important in some scenario (many were considered), or really not important at all.

Table C.1 Guidelines for Importance Ranking

Descriptor	Definition
High	First order importance to metric of interest. Model adequacy, code adequacy, and validation adequacy should be at the "High Level."
Medium	Secondary importance to metric of interest. Model adequacy, code adequacy, and validation adequacy should be at least the "Medium Level."
Low	Negligible importance to metric of interest. Not necessary to model this phenomena for this application.
Uncertain	Potentially important. Importance should be explored through sensitivity study of discovery experiment and the PIRT revised.

In these tables, which are listed below, the importance ranking was also separated into Phase 1 - "initial transient" and Phase 2 - "damage transient" periods. These were defined as follows:

> Phase 1 - Initial transient: Begins at the accident initiating event and includes the time period when the state of the system moves from its normal operating state to a point where the structural geometry or material phases of system components begin to change.
>
> Phase 2 - Damage transient: Extends from the point when structural or material changes begin to occur until the end of the accident.

Finally, an attempt was also made to assess the adequacy of currently available models for each of the physical processes, phenomena, or material properties. Table C.2 lists the definitions used for the High (H), Medium (M), Low (L), or "Strategy" levels that were the choices used to describe the model adequacy. A "?" was used if the participating staff felt they were unable to judge the adequacy of available models. In some cases, model adequacy was judged to vary, and a range of adequacy was therefore indicated (e.g. M->H). If participants did not agree on an adequacy level then the differing views were reflected with a slash (e.g. M/L).

Table C.2 Guidelines for Assessing Model Adequacy
(adapted from SAND2002-1740)

Descriptor	Definition
High	A mature physics-based model or correlation-based model is available that is believed to adequately represent the phenomenon over the full parameter space of the application.
Medium	Significant discovery activities have been completed. At least one candidate model form or correlation has emerged that is believed to nominally capture the phenomenon over some portion of the application parameter space.
Low	No significant discovery activities have occurred and the model form is still unknown or speculative.
Strategy	Inadequacies are addressed through explicitly stated strategy. This may include acceptance of the inadequacy, the parallel use of alternate plausible models, the use of stylized bounding models, or other documented strategies.

Table C.3 Neutronics

Physical process, phenomena, or material property	Importance Ranking Phase 1	Importance Ranking Phase 2	Model Adequacy
Lattice physics for macroscopic cross-sections and power distributions Affects reactivity feedbacks Neutron Energy distribution	H	H	H
Cross-section data for transuranics	H	H	?
3-D space time kinetics -> flux distribution	H	H	H
Coupling: thermal-mechanical-neutronic	H	H	M
Radioisotope inventory and depletion	H	H	H
Neutron Cross Section Libraries	H	H	M
Decay heat	M	H	H
Tracking geometrical changes	na	H	?

Table C.4 Fluid Flow and Heat Transfer

Physical process, phenomena, or material property	Importance Ranking Phase 1	Importance Ranking Phase 2	Model Adequacy
Single phase forced convective flow	H	M	H
Single phase natural convection	H	M	H
Multi-phase multi-component (compressible) forced convective flow	na	H	M
Multi-phase multi-component natural convection	na	H	M
Boiling heat transfer and mass transfer	M	H	H
Conduction heat transfer	H	H	H
Radiation heat Transfer	H	H	H
Fuel-coolant interactions	na	H	L
Relocation of degraded material (melt, solid)	na	H	M/L
Material properties	H	H	M -> H
Multi-field model constitutive equations (inter-field transport)	na	H	M
Flow/heat transfer in porous media (Debris bed dynamics)	na	?	M
Chemically reacting flow	na	?	M

Table C.5 Material Interactions and Chemistry

Physical process, phenomena, or material property	Importance Ranking Phase 1	Importance Ranking Phase 2	Model Adequacy
Intra-pin fission gas release/migration model	H	H	H
Fuel-cladding chemical interaction (laves phase)	na	H	M
Chemical activity of fission products in sodium	na	H	H
Vaporization of fission products out of sodium	na	H	H
Aerosol formation and growth	na	H	H
Aerosol material density and thermoconductivity	na	H	H
Aerosol shape factors (dynamic, collision)	na	H	M
Primary particle size distribution	na	H	H
Aerosol deposition modeling	na	H	H
Iodine chemistry model	na	H	M
Sodium concrete interactions	na	H	H
Sodium fires (sprays, and pool)	na	H	H
Hydrogen combustion	na	H	H
Corrosion	-		M
Radiation induced cracking	-		M
Absorption of “stuff” by the cladding			?
Source term mitigation systems (sprays, filters, leak paths)	na	H	H

Table C.6 Material Properties and Equations of State

Physical process, phenomena, or material property	Importance Ranking Phase 1	Importance Ranking Phase 2	Model Adequacy
Thermodynamics (i.e Eq. of state) of various gases, liquids, solids	H	H	H/M
Thermal physical properties at a broad range of temperatures and pressures	H	H	M->H
Conductivity, density, specific heat, phase change temperatures, thermal expansion, .mechanical ., burn-up, .	H	H	M->H
Especially: Fuel thermophysical properties	H	H	M
Phase diagrams for multi-component systems	L	U	M

Table C.7 Structural Mechanics, Dynamics, Relocation

Physical process, phenomena, or material property	Importance Ranking Phase 1	Importance Ranking Phase 2	Model Adequacy
Formation of rubble/debis bed	na	H	L
Thermal expansion of core components	H	H	H
Structural integrity of vessel and containment	na	H	H
Crack formation and propagation (cladding, other structural components)	L	H	M->H
Plastic straining of cladding (fission gas pressure, . .)	M	H	H
Structural response to prompt energetic events	na	H	H
Molten fuel/clad relocation and melting	na	H	M
Fuel swelling - longer time scale process (fuel foaming is an extreme example)	na	H	M
Fuel vaporization - short time scale event	na	H	H
Structural response to seismic-like events	H	?	H

C.2 Some additional high-level points-of-emphasis expressed or derived from the meeting and discussions.

Rich Pryor opened up the discussion on Neutronics by reviewing a set of three viewgraphs. His main point was to make clear that from a neutronics standpoint fast reactors are quite different from thermal spectrum reactors, and that the accurate modeling of the coupled neutronic-thermal-mechanical processes will be critical in performing safety analysis studies.

The question of whether or not an effort should be made to model Phase 2 physics and phenomena came up numerous times, and in a variety of contexts during the meeting. The arguments in favor seem to revolve around the idea that a "defense-in-depth" policy would require one to adequately model and simulate severe accident phenomenology in order to fully understand the consequence side of the risk equation. The arguments opposing such an effort point to (1) the view that any design with a risk significant chance of reaching this stage in an accident will never be built, and (2) the fact that the modeling problems in Phase 2 are extremely difficult and that the effort to improve model fidelity and perform the associated V&V (including experimental work) for such models may be cost prohibitive. No attempt was made in the meeting to build a consensus opinion on this issue.

Several members of the working group commented on the important role that PRA studies should play in the safety aspects of the licensing process. In their view, it will be PRA work that will (or should) be the basis for determining which of the various processes and phenomena that we have listed in our tables must be modeled in safety codes.

The large number of "Highs" in the importance ranking column in the PIRT tables was noted. It was suggested that more resolution might be obtained by soliciting input from key players, asking them to rank the highs against each other, and discouraging but not prohibiting too many ties. It was also suggested that it might be worthwhile to prioritize not only by the importance of phenomena at some final end point of development, but to think also about the evolving needs of the customer as the ABR program evolves.

For the purposes of supporting the NRC, it was suggested that it will be especially important to develop an early, independent, and credible capability to model accidents that pose very large challenges to the active and passive safety features of the plant but that *do not result in a large release*. In this persons opinion, for the ABR to succeed, it will be more important for the NRC to be able to map out the region of accident space that does not result in large releases than to be able to accurately calculate the large releases for the more severe accidents. Doing that mapping will allow the establishment of probabilities for the large releases, and whether or not the ABR is approved or not will depend more on those probability numbers than on the consequence numbers.

C.3 List of Meeting Participants

Name	Sandia Org.
Paul Pickard	6771
John Kelly	6770
Dana Powers	6770
Ken Bergeron	6770
Randy Gauntt	6762
Jeff LaChance	6762
Larry Humphries	6762
Mark Allen	6761
Veena Tikare	1814
Justine Johanna	1810
Steve Gianoulakis	1541
Rod Schmidt	1437
Rich Pryor	1433
Randy Summers	1431
Jim Tomkins	1420
Richard Coats	1383
Jim Dahl	1383

C.4 References

1. U.S. Nuclear Regulatory Commission, *Preapplication Safety Evaluation Report for the Power Reactor Innovative Small Module (PRISM) Liquid Metal Reactor, Final Report*, NUREG-1368, February 1994
2. U.S. Nuclear Regulatory Commission, *Preapplication Safety Evaluation Report for the Sodium Advanced Fast Reactor (SAFR) Liquid Metal Reactor*, NUREG-1369, December 1991.
3. Y. I. Chang, P. J. Finck, C. Grandy *et al*, *Advanced Burner Test Reactor Preconceptual Design Report*, ANL-ABR-1 (ANL-AFCI-173), Argonne National Laboratory, Sept. 5, 2006.

DISTRIBUTION

1	MS0123	D. Chavez, LDRD Office	1011 (electronic copy)
1	MS1318	R. Hooper	1414 (electronic copy)
1	MS1318	A. Salinger	1414 (electronic copy)
1	MS1318	R. Pawlowski	1414 (electronic copy)
1	MS1318	R. Hooper	1414 (electronic copy)
1	MS1416	S. Plimpton	1416 (electronic copy)
1	MS0321	J. Mitchiner	1430 (electronic copy)
1	MS0378	R. Summers	1431 (electronic copy)
1	MS0370	W. Spotz	1433 (electronic copy)
1	MS0316	R. Hoekstra	1437 (electronic copy)
1	MS0316	R. Schmidt	1437 (electronic copy)
1	MS0316	J. Shadid	1437 (electronic copy)
1	MS0836	A. Lorber	1541 (electronic copy)
1	MS0836	P. Notz	1541 (electronic copy)
1	MS0382	K. Belcourt	1543 (electronic copy)
1	MS0382	M. Glass	1543 (electronic copy)
1	MS0382	M. Borden	1543 (electronic copy)
1	MS0382	S. Owen	1543 (electronic copy)
1	MS0382	T. Blacker	1543 (electronic copy)
1	MS0374	Z. Pickett	2991 (electronic copy)
1	MS0748	R. Gauntt	6762 (electronic copy)
1	MS0748	L. Humphries	6762 (electronic copy)
1	MS1369	P. Mattie	6762 (electronic copy)
1	MS0701	M. Walck	6700 (electronic copy)
1	MS0736	J. Kelly	6770 (electronic copy)
1	MS0736	D. Powers	6770 (electronic copy)
1	MS0747	T. Bartel	6774 (electronic copy)
1	MS0747	V. Tikare	6774 (electronic copy)
1	MS0771	A. Orrell	6800 (electronic copy)
1	MS9001	C. Moen	8005 (electronic copy)
1	MS0899	Technical Library	9536 (electronic copy)


Sandia National Laboratories