NUREG/IA-0541

International Agreement Report

Multi-scale Coupling of TRACE and SUBCHANFLOW based on the Exterior Communication Interface (ECI)

Prepared by:
Kanglong Zhang, Victor Hugo Sanchez-Espinoza

Institute for Neutron Physics and Reactor Technology (INR), Karlsruhe Institute of Technology (KIT)
Hermann-von-Helmholtz-Platz 1
Eggenstein-Leopoldshafen, Baden-Württemberg, 76344, Germany

Division of Systems Analysis
Office of Nuclear Regulatory Research
U.S. Nuclear Regulatory Commission
Washington, DC 20555-0001

Manuscript Completed: July 2022
Date Published: November 2023

Prepared as part of
The Agreement on Research Participation and Technical Exchange
Under the Thermal-Hydraulic Code Applications and Maintenance Program (CAMP)

Published by
U.S. Nuclear Regulatory Commission

AVAILABILITY OF REFERENCE MATERIALS IN NRC PUBLICATIONS

NRC Reference Material

As of November 1999, you may electronically access NUREG-series publications and other NRC records at NRC's Library at www.nrc.gov/reading-rm.html. Publicly released records include, to name a few, NUREG-series publications; *Federal Register* notices; applicant, licensee, and vendor documents and correspondence; NRC correspondence and internal memoranda; bulletins and information notices; inspection and investigative reports; licensee event reports; and Commission papers and their attachments.

NRC publications in the NUREG series, NRC regulations, and Title 10, "Energy," in the *Code of Federal Regulations* may also be purchased from one of these two sources.

1. **The Superintendent of Documents**
   U.S. Government Publishing Office
   Washington, DC 20402-0001
   Internet: https://bookstore.gpo.gov/
   Telephone: (202) 512-1800
   Fax: (202) 512-2104

2. **The National Technical Information Service**
   5301 Shawnee Road
   Alexandria, VA 22312-0002
   Internet: https://www.ntis.gov/
   1-800-553-6847 or, locally, (703) 605-6000

A single copy of each NRC draft report for comment is available free, to the extent of supply, upon written request as follows:

Address: **U.S. Nuclear Regulatory Commission**
Office of Administration
Digital Communications and Administrative Services Branch
Washington, DC 20555-0001
E-mail: Reproduction.Resource@nrc.gov
Facsimile: (301) 415-2289

Some publications in the NUREG series that are posted at NRC's Web site address www.nrc.gov/reading-rm/doc-collections/nuregs are updated periodically and may differ from the last printed version. Although references to material found on a Web site bear the date the material was accessed, the material available on the date cited may subsequently be removed from the site.

Non-NRC Reference Material

Documents available from public and special technical libraries include all open literature items, such as books, journal articles, transactions, *Federal Register* notices, Federal and State legislation, and congressional reports. Such documents as theses, dissertations, foreign reports and translations, and non-NRC conference proceedings may be purchased from their sponsoring organization.

Copies of industry codes and standards used in a substantive manner in the NRC regulatory process are maintained at—

The NRC Technical Library
Two White Flint North
11545 Rockville Pike
Rockville, MD 20852-2738

These standards are available in the library for reference use by the public. Codes and standards are usually copyrighted and may be purchased from the originating organization or, if they are American National Standards, from—

American National Standards Institute
11 West 42nd Street
New York, NY 10036-8002
Internet: www.ansi.org
(212) 642-4900

Legally binding regulatory requirements are stated only in laws; NRC regulations; licenses, including technical specifications; or orders, not in NUREG-series publications. The views expressed in contractor prepared publications in this series are not necessarily those of the NRC.

The NUREG series comprises (1) technical and adminis-trative reports and books prepared by the staff (NUREG–XXXX) or agency contractors (NUREG/CR–XXXX), (2) proceedings of conferences (NUREG/CP–XXXX), (3) reports resulting from international agreements (NUREG/IA–XXXX),(4) brochures (NUREG/BR–XXXX), and (5) compilations of legal decisions and orders of the Commission and Atomic and Safety Licensing Boards and of Directors' decisions under Section 2.206 of NRC's regulations (NUREG–0750),and (6) Knowledge Management prepared by NRC staff or agency contractors.

NUREG/IA-0541

International Agreement Report

Multi-scale Coupling of TRACE and SUBCHANFLOW based on the Exterior Communication Interface (ECI)

Prepared by:
Kanglong Zhang, Victor Hugo Sanchez-Espinoza

Institute for Neutron Physics and Reactor Technology (INR), Karlsruhe Institute of Technology (KIT)
Hermann-von-Helmholtz-Platz 1
Eggenstein-Leopoldshafen, Baden-Württemberg, 76344, Germany

Division of Systems Analysis
Office of Nuclear Regulatory Research
U.S. Nuclear Regulatory Commission
Washington, DC 20555-0001

Manuscript Completed: July 2022
Date Published: November 2023

Prepared as part of
The Agreement on Research Participation and Technical Exchange
Under the Thermal-Hydraulic Code Applications and Maintenance Program (CAMP)

Published by
U.S. Nuclear Regulatory Commission

ABSTRACT

This report describes the multi-scale coupling of the system code - TRACE and the sub-channel thermal-hydraulic code – SUBCHANFLOW (SCF) for a better description of the multi-dimensional thermal-hydraulic phenomena inside the Reactor Pressure Vessel (RPV) of a Pressurised Water Reactor (PWR).

In the recent versions of TRACE, the ECI was activated for multi-tasking and coupling of TRACE with different solvers e.g., CFD, sub-channel codes. To couple TRACE with the Karlsruhe Institute of Technology (KIT) in-house sub-channel code – SUBCHANLFOW (SCF), a specific ECI-module for SUBCHANFLOW was developed. In this report, the implemented spatial mapping of the involved thermal-hydraulic domains and the time synchronization of the involved solvers are described for both stationary and transient simulation. A domain decomposition approach and a weighted field-mapping method were adopted for this purpose.

Besides, an explicit operator splitting method is implemented for the data transfers during the time advancement of both codes either in stationary or transient simulations.

The prediction capability of the coupled code is demonstrated by the analysis of two academic coolant-mixing cases and a VVER-1000 coolant-mixing benchmark. The results obtained by TRACE standalone and by the coupled system TRACE/SCF were compared together and it shows that the coupling system of TRACE/SCF could better predict the coolant mixing along the core than TRACE-standalone.

FOREWORD

This assessment report deals with the development of a coupling interface for TRACE and SCF using the ECI-interface developed specifically for TRACE. In this report, the development of an ECI-interface for SCF is described and the implementation of the coupling of the code for stationary and transient simulations is discussed as well. The method for the spatial mapping of the computational domains as well as the synchronization of the time advancement of the two coupled codes is also discussed. The first results obtained with the coupled code TRACE/SCF are presented and discussed.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

EXECUTIVE SUMMARY

The scope of this report is to present the development of the multi-scale thermal hydraulic code TRACE/SCF based on the ECI interface. First of all, the original ECI module in TRACE was debugged. Second, a specific ECI was developed for SCF. Then, the two codes with their interfaces are coupled together. The validation with a VVER-1000 coolant mixing benchmark shows that the coupled code can predict a stronger coolant mixing in the core compared to TRACE standalone.

ABBREVIATIONS AND ACRONYMS

ABN	Approximate Block Newton
BWR	Boiling Water Reactor
CFD	Computational Fluid Dynamics
CHF	Critical Heat Flux
CMFD	Computational Multi-Fluid Dynamics
COW	Cluster of Workstations
CPR	Critical Power Ratio
DES	Detached Eddy Simulation
DNS	Direct Numerical Simulation
DSM	Distributed Shared Memory
ECI	Exterior Communication Interface
IT	Information Technology
JFNK	Jacobian-Free Newton-Krylov
LBLOCA	Large Break LOss of Coolant Accidents
LES	Large Eddy Simulation
LOCA	LOss-of-Coolant Accident
LOFW	LOss of Feed Water
LWR	Light Water Reactor
MD	Molecular Dynamics
MIMD	Multiple Instruction Multiple Data
MISD	Multiple Instruction Single Data
MPI	Message Passing Interface
MPP	Massively Parallel Processor
MSLB	Main Steam Line Break
NPP	Nuclear Power Plant
OSSI	Operation Splitter Semi-Implicit
OS	Operating Splitter
PTS	Pressurized Thermal Shock
PVM	Parallel Virtual Machine
PVP	Parallel Vector Processor
PWR	Pressurized Water Reactor
RHR	Residual Heat Removal

RPV	Reactorr Pressure Vessel
SBLOCA	Small Break LOss of Coolant Accidents
SCF	SUBCHANFLOW
SGTR	Steam Generator Tube Ruptures
SIMD	Single Instruction Multiple Data
SISD	Single Instruction Single Data
SMP	Symmetric Multiprocessor

1 INTRODUCTION

1.1 Motivation

Thermal-hydraulic simulation tools play an increasingly crucial role in the present-day Nuclear Power Plant (NPP) safety analysis and design. The thermal-hydraulic physical phenomena occurring in the NPP components e.g., the Nuclear Pressure Vessel (RPV) involve information on different spatial scales. Their characteristic lengths vary from meters down to nanometers. The NEPTUNE project classified those simulations to three main scales: system scale, component scale (also known as sub-channel scale or CFD in a porous medium) and average scale (also known as CFD in open medium) (Guelfi, et al. 2007) . The European NURESIM and NURISP projects follow its classification method but also refer the average scale to mesoscale and further include a microscale (Bestion, et al. 2012). Moreover, reference (Niceno, et al. 2010) extends the simulation to nanoscale which usually resorts to Molecular Dynamics (MD) modeling techniques. Some other works also use different names for simulation scales but refer to the same stuff (D'Auria, F.; Galassi, G.M. 2010). To describe the thermal-hydraulic processes undergoing at these spatial scales, various thermal-hydraulic simulation codes were developed e.g., 1D or 3D system thermal-hydraulic codes, porous-media codes (CFD-porous media, sub-channel codes, porous-media 3D codes), open-medium-CFD, Large Eddy Simulation (LES), Detached Eddy Simulation (DES) and Direct Numerical Simulation (DNS).

The system codes were designed to simulate almost all normal and accident scenarios of the whole plant e.g., Large Break LOss of Coolant Accidents (LBLOCA), Small Break LOss of Coolant Accidents (SBLOCA), Steam Generator Tube Ruptures (SGTR), LOss of Feed Water (LOFW), Main Steam Line Break (MSLB), loss of Residual Heat Removal (RHR) system (Petruzzi and D'Auria 2007). Traditional system codes such as RELAP (RELAP5-3D code manuals, Volumes I, II, IV, and V 1999), TRACE (US NRC 2010), ATHLET (Lerchl and Austregesilo 2006), CATHARE (Emonot, et al. 2011) has already been widely used for many years. Their simulations mostly lie on a system scale.

The simulations on the component scale are mostly carried out using the sub-channel analysis codes e.g., COBRA-TF (Thurgood, et al. 1983), VIPRE (Sung, Schueren and Meliksetian 1999), FLICA (Toumi, et al. 2000), SUBCHANFLOW (SCF) (Sánchez-Espinoza, Imke and Ivanov 2010) to estimate the various thermal-hydraulic safety parameters in the core e.g., Critical Heat Flux (CHF) ratio, Critical Power Ratio (CPR), fuel centerline temperature, fuel surface temperature, sub-channel maximum temperature and bulk coolant outlet temperature (Chelemer, Weisman and Tong 1972). A very detailed review of sub-channel analysis methods and codes was given in reference (Moorthi, Sharma and Velusamy 2018), where each aspect of sub-channel simulation was carefully inspected. Reference (Cheng and Rao 2015) presents another brief review of sub-channel codes and addresses some words for CANDU codes. Recently, to take profit from the keeps-growing computer power and accelerate the simulation efficiency, some sub-channel codes were parallelized e.g., SCF and CORBA-TF (Kucukboyaci and Sung 2015). Another trend is the development of dedicated 3D porous media two-phase flow codes with a Cartesian or unstructured grid to overcome the limitations of 1D system codes. Instances are CUPID (Jeong, et al. 2010), PORFLOW (Runchal and Sagar n.d.), TWOPORFLOW (Chavez, Imke and Sanchez-Espinoza 2018), PORFLO (Ilvonen, Hovi and Inkinen 2010), et al.

Moreover, to meet the rapidly increasing requirement of the multidimensional simulation within the Reactor Pressure Vessel (RPV) e.g., the downcomer, lower plenum and upper plenum of a Light Water Reactor (LWR), the CFD codes e.g., ANSYS CFX, STAR-CD, OpenFOAM, TrioCFD

are now playing important role in the thermal-hydraulic analysis of NPP, focusing on simulation of microscale. Reference (Mahaffy, et al. 2007) provides a professional guideline on how to use CFD in nuclear reactor safety analysis, as well as a broad overview of such applications including some special cases such as Pipe Wall Erosion, Natural Conversion, Thermal Cycling. Some more general applications e.g., Coolant Mixing, Stratified Flow, Hot Channel in the core, are discussed in Reference (H¨ohne, Krepper and U.Rohde 2009). Computational Multi-Fluid Dynamics (CMFD) is another most promising trend for nuclear reactor safety analysis. But due to the lack of flow schemes modeling as well as some other difficulties summarized by Reference (Bestion, D. 2014), this process is heavy, long and expensive. Reference (Yadigaroglu 2005) also emphasize those difficulties but narrows its eyesight to specific cases such as condensation of large bubbles in a pool of water. Nevertheless, it is still a promising trend thanks to the massive application of powerful and cheap computer clusters (Use and Development of Coupled Computer Codes for the Analysis of Accidents at Nuclear Power Plants 2003) and the constant effort on CMFD study.

To enhance the description and prediction of thermal-hydraulic phenomena in a more precise manner than what was done by isolated application of single codes, different thermal-hydraulic codes on different simulation scales are coupled together and such multi-scale codes and analysis have been initiated on different teams worldwide. The typical correlational research was carried out by the 6th and 7th European Framework Programs NURESIM and NURISP, in which several multi-scale couplings and simulations were developed to better investigate Pressurized Thermal Shock (PTS), Critical Heat Flux (CHF), et al. Reference (Calvin and Nowak 2010)also presents some cases with the aid of multi-scale tools such as Steam or Feed Lines Break. But it also highlights some of the multi-scale and multi-physics codes and teasing out the coupling methods. Other coupling aspects e.g., classifications of the coupling are also investigated in Reference (Use and Development of Coupled Computer Codes for the Analysis of Accidents at Nuclear Power Plants 2003) and (D'Auria, et al. 2004). Various coupling of system thermal-hydraulic codes sub-channel codes have already been performed using different methodologies, e.g., RELAP5/CORBA (Jeong, et al. 1997), CATHARE2/TRIO_U-MC (Anderhuber, et al. 2015), RELAP7 (Zhang, Zou, et al. 2014). Moreover, coupling of system codes and CFD-codes were also well investigated, e.g., RELAP5-3D/CFX (Aumiller, Tomlinson and Bauer 2001), ATHLET/FLUENT (Macek and Vyskoci 2013), CATHARE/TRIO_U (Bavière, et al. 2013).

At the Karlsruhe Institute of Technology (KIT), the multi-scale investigations are devoted to the coupling of system codes with sub-channel and CFD codes. The system thermal-hydraulic code TRACE has already been successfully coupled with the sub-channel code SUBCHANFLOW (SCF) (Zhang and Sanchez-Espinoza 2018), using a newly developed SOCKET-based Exterior Communication Interface (ECI) (Mahaffy, J.; Hansell, B. 2003). The validity and rationality have been verified with two specially designed coolant mixing cases. Besides, the coupled codes are validated against a VVER-1000 coolant mixing benchmark.

1.2 Scope of the Report

This report is subdivided into seven chapters. The first chapter talks more about the motivation of the multi-scale work. The second and third chapter describes the elemental modification to TRACE and SCF for the coupling purpose, respectively. The detailed coupling issues are presented in Chapter 4. The verification of the prediction capability of the coupled TRACE/SCF code is discussed in Chapter 5. The validation process is exhibited in Chapter 6. Finally, the main conclusions and outlook are summarized in Chapters 7 and 8. The technical details are exhibited in the appendix.

2 TRACE FOR THE MULTI-SCALE COUPLING SYSTEM

Overview of TRACE is the priority to be delivered in this section. What's more, the inherent coupling capability of TRACE which is known as the multi-task system and the corresponding highly implemented coupling tool which is called the Exterior Communications Interface (ECI) is also the key concept desired for an explanation. Since the multi-task employing ECI is one of the ways of implementing the coupling in this work.

2.1 Overview of TRACE

TRACE is the abbreviation of TRAC/RELAP Advanced Computational Engine which is formerly called TRAC-M. It is the latest in a series of advanced, best-estimate reactor systems codes developed by the NRC for analyzing transient and steady-state neutronic thermal-hydraulic behavior in light water reactors. It combines the capabilities of the NRC four main system codes: TRAC-P, TRAC-B, RELAP5, and RAMONA. TRACE has been designed to perform best-estimate analyses of loss-of-coolant accidents (LOCAs), operational transients, and other accident scenarios in pressurized light-water reactors (PWRs) and boiling light-water reactors (BWRs). Also, it can simulate the physical phenomena in some NPP simulation oriented experimental facilities.

TRACE takes a component-based approach to modeling a reactor system. Each physical piece of equipment in a flow loop can be represented as some type of component, and each component can be further nodalized into some number of physical volumes (also called cells) over which the fluid, conduction, and kinetics equations are averaged. TRACE component modules currently include BREAKs, FILLs, CHANs, CONTANs, EXTERIORs, FLPOWERs, HEATRs, HTSTRs, JETPs, POWERs, PIPEs, PLENUMs, PRIZERs, PUMPs, RADENCs, REPEAT-HTSTRs, SEPDs, TEEs, TURBs, VALVEs, and VESSELs with associated internals (downcomer, lower plenum, reactor core, and upper plenum). Among the various components, VESSEL is the special 3D item that can model the RPV, RWST and other components in which 3D phenomena take place. By taking advantage of VESSEL, TRACE can simulate a 3D (x, y, z) Cartesian- and/or (r, θ, z) cylindrical-geometry flow calculation.

The basic governing equation set of TRACE includes six equations that can simulate a full two-fluid hydrodynamic like the gas-liquid flow. Besides, two more equations are applied to describe the non-condensable gas field and to track dissolved solute. The eight equations give TRACE full non-homogeneous, non-equilibrium modeling capabilities. Beyond that, a flow-regime-dependent constitutive-equation package has been incorporated into the code to supply a reference for the flow topology dependence by steam-liquid interaction. Benefited from this package and some other techniques, TRACE gains a comprehensive heat transfer capability and can perform detailed heat-transfer analyses of the vessel and the loop components.

For the solution of the equation set, two methods are available in TRACE: the semi-implicit method and the SETS method. Compared to the explicit method, the semi-implicit method eliminates the sound speed from the Courant stability by implicit the pressure in momentum equations, leaving what is commonly referred to as the "material courant" limit. The material Courant limit is eliminated to some extent by the SETS method through pre-solving a so-called "stabilizer momentum equation" before the semi-implicit step and post-solving two so-called "stabilizer continuous and energy equations". The solution of stabilizer continuous equation is an attempt to implicit the density solution. However, it is not a complete implicit way, which means

even though the material Courant limit could be loosed sufficiently it is still not an unconditional stability method anyway.

Now, jump out of the mathematical and numerical species and turn to the code's logic. TRACE can run both SS and Transient calculations. The SS behaves quite similar to the Transient except that the SS has a special SS criterion check after every pre-determined steps and applies a tiny modification to the Transient equations' time term. A quite high-level workflow of TRACE is displayed in Figure 2-1. Usually, TRACE runs in a typical serial mode: one model, one executable and one thread. Nevertheless, TRACE could run in parallel too. Under this mode, the integrated model is divided into several sub-models and each of them is simulated by one TRACE executable. Each TRACE executable is labeled as a task thus form a so-called "multi-task" system. Within the system, all the tasks run at the same time (this can be understood as a parallel way) and they communicate with each other through a specially designed interface ECI. The naming of ECI comes from the fact that it utilizes the EXTERIOR component to assist the data transfer.

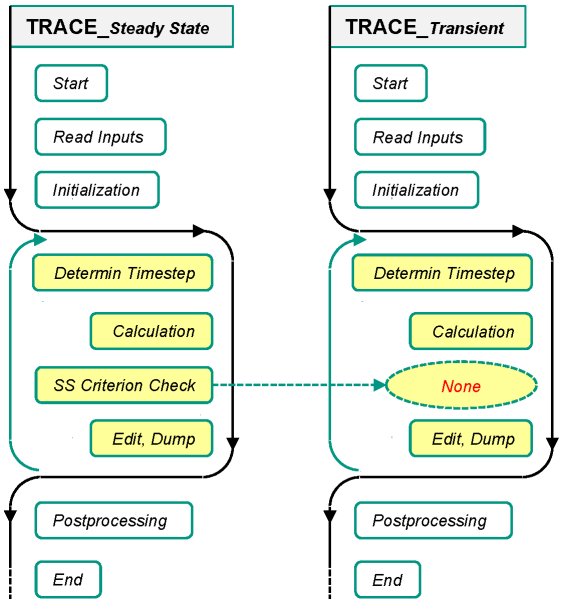


Figure 2-1 General Workflow of TRACE

2.2 The Exterior Communication Interface (ECI) and Multi-task System

The previous section said that the TRACE itself could be paralyzed forming a multi-task system that looks like Figure 2-2. Three characteristics of ECI could be derived from the figure. They are

1) ECI is a highly integrated module in TRACE since it is closely related to TRACE data structure (notice the intimate relationships between TRACE subroutines and ECI);
2) Each task should have its own ECI. What's more, each special simulation code should have its customized ECI (notice ECI is specified as the "ECI for the code");
3) This is a server-less system since each task could communicate directly with one another (notice the three bold double-sided arrows).

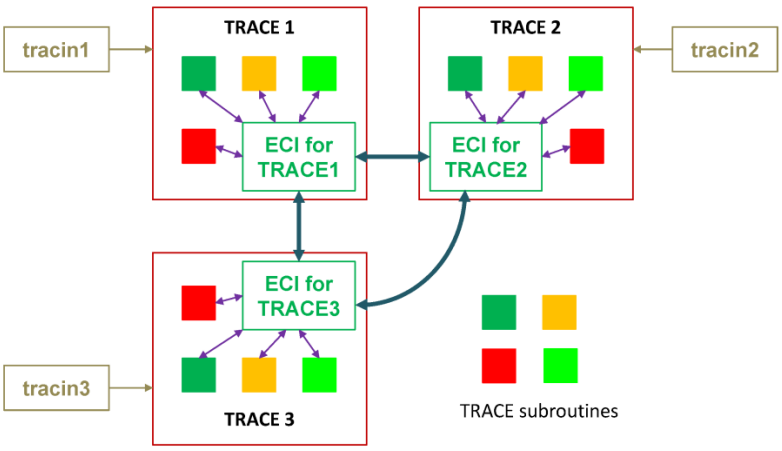


Figure 2-2 Multi-task System of TRACE

The originally published ECI for TRACE V51051 suffers several fatal bugs which usually lead to the data transfer failure. It was found that the ECI normally works properly for TRACE-TRACE coupling but always fails for TRACE and SCF coupling. This is because the pure TRACE coupling only involves the transfer of scalars while the latter one has to deal with the heavy transfer of arrays. The bugs are almost all derived from this concern, while also, there are several other minor bugs which not impact on the data transfer but throw out meaningless messages. All of the bugs are carefully inspected and finally fixed. The brief bug report is summarized in Appendix A.

Since ECI is closely relevant to the computer science and its elemental concept is the network communication unit – SOCKET, some basic pieces of knowledge about the computer architecture and the SOCKET are explained in Appendix C and Appendix D. The EXTERIOR component, as the cornerstone for pure TRACE multi-task coupling, is well inspected and described in Appendix E. Moreover, the details of the establishment of the multi-task system with ECI are illustrated in Appendix F.

2.3 Tips for a Successful Multi-task Execution

Up to now, the elements which are essential for a successful calculation is still not completed. Another two elements known as taskList and Driver are required. the taskList is a text file specifying all the tasks' information, including the task number, the executable directory and name, the program name and the input files' directory (Figure 2-3). It could also be accessed in Appendix F.

Task1	directory of TRACE/trace.exe
TRACE	directory of the tracin1
Task2	directory of TRACE/trace.exe
TRACE	directory of the tracin2
Task3	directory of TRACE/trace.exe
TRACE	directory of the tracin3

Figure 2-3 The taskList File Structure

The driver is a java program coordinating the tasks' running. The normal working procedures of the multi-task system could be illustrated as the way shown in Figure 2-4. Two steps are required to run the system:

1) Run the driver;
2) Run TRACE. This TRACE will first read the taskList. Then send the tasks information to the driver. According to the information, the driver will run the other two TRACEs. Then the three TRACEs will read their input files (tracin) and start the calculation with data communications all through the problem time.

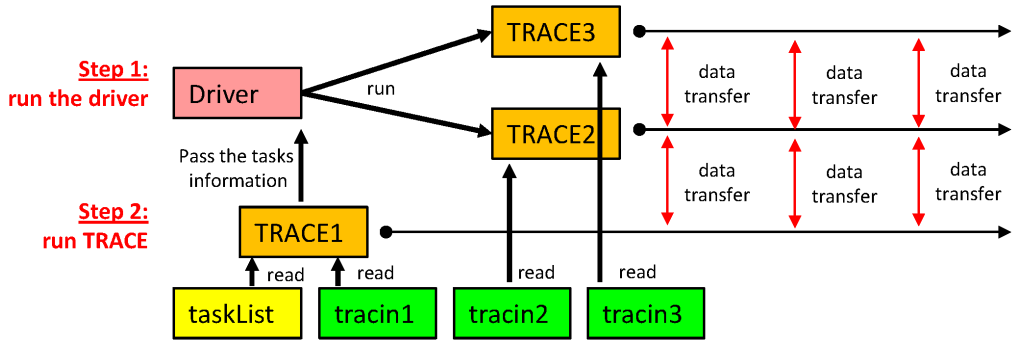


Figure 2-4 Running Procedure of the Multi-task System

The previous sections have introduced that the multi-task system is a server-less system. However, the first started TRACE in step 2 has a special function. That is, gathering and sending the calculation control flags from and to all other tasks thus all the tasks could keep tight synchronization along the problem time. It can further gather information from all other TRACEs to generate an exterior equation set and then solve it. This is quite a beautiful function which makes the coupling fully implicit. It is worth noting here that the generation of the multi-task system, along with its coordination, synchronization and inter-task communication, are all in the charge of ECI. In another word, the multi-task system is a concept, while ECI is the exact tool implementing the concept.

Each TRACE in the multi-task system has its input file – tracin. It is a little different from the normal tracin file. Several tricky points must be paid special attention otherwise there would rise quite confusing and unclear errors. From the practical perspective, the tracin files should be automatically exported from SNAP, no matter for the normal tracin and the multi-task tracins. The

problem here is that the exported tracin from SNAP for multi-task execution can not be directly used. Several parts should be modified. Appendix G explains sufficiently about this kind of "shortcoming" of SNAP. Moreover, there are two other tips for an efficient and successful multi-task calculation using ECI.

1) The master's working directory should contain the input file "tracin" and the task description file "taskList". All the other clients' working directories should contain the input file "tracin". These are exactly all the required files to perform a multi-task calculation. Other intermediate files such as "taskList.copy" or "parent.PID" does not influence the calculation because the code will automatically delete such kind of files before it creates the new intermediate files.
2) The driver routine should run first and it may have an arbitrary location on the host as long as it is launched from the master directory.

2.4 Adapt TRACE to the Multi-scale Coupling System

Because the numerical solvers of TRACE and SCF differ significantly, modifications to the ECI in TRACE and to TRACE source are necessary to couple TRACE and SCF together. Appendix B is exactly about such modifications and some newly added logics into the code's source. Additionally, there are two new input parameters for TRACE, they are defined in the "model flags" block (Figure 2-5):

1) *bound_l* – a parameter identifying the components which act as the data transfer interfaces with SCF from the normal TRACE components.
2) *j_skip* – the flag telling TRACE in which coupling mode is selected by the user.

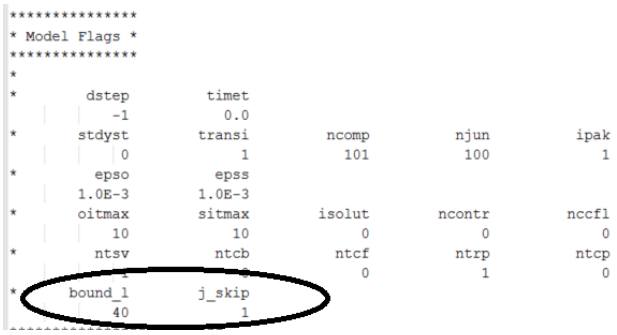


Figure 2-5 The New Input Parameters of TRACE for TRACE/SCF

Details of the usage of the two parameters are explained together with the coupling approaches in Chapter 4.

3 SUBCHANFLOW (SCF) FOR THE MULTI-SCALE COUPLING SYSTEM

3.1 Overview of SCF

SCF is a thermal-hydraulic sub-channel code developed for the simulation of fuel rod bundles and cores of light water and innovative reactor systems being developed at KIT/INR (Imke, Sanchez and Gomez 2010). It is based on the COBRA-family (Rowe 1973) (Wheeler, et al. 1976) (Basile, et al. 1999), while those old codes' obsolete programming style was updated and refined with a modular Fortran-95 style with dynamic memory allocation. Also, the British Units were sufficiently replaced by the SI Units. Some other programming efforts were made to simplify the SCF structure as much as possible. For example, no pointers or derived types were adopted in the code and all the global variables were packed in a single module, instead of using common syntax. The definition of input file was also simplified by making it possible to use external tables.

SCF can handle both rectangular and hexagonal fuel bundles and core geometries. It can achieve sub-channel fuel assembly simulation and "channel-wise" whole core simulations. Both SS and Transient calculations are available in SCF. Different from most simulation tools, the SS calculation in SCF is a real steady-state solution which removes the time term from the equation set, instead of time approaching method. Also, the boron concentration prediction is not possible for a SS calculation. One more key point to be claimed here is that the first step of Transient will always be a SS calculation. The general workflow of SCF is displayed in Figure 3-1.

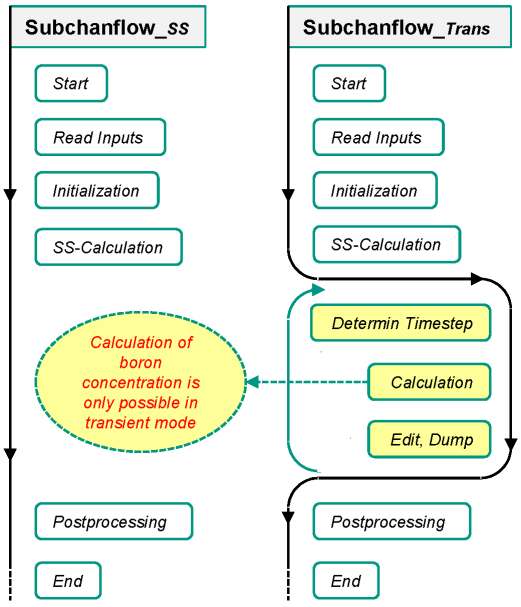


Figure 3-1 General Workflow of SCF

Moreover, SCF could utilize the OpenMP capability to make its calculation in parallel. This is a shared memory parallelization approach, in which the channels will be divided into several groups and each group will be calculated by one thread. This trick could significantly cut the computer elapsed time. What's more, SCF has already been successfully coupled to SALOME which is an

open-source software providing a generic platform for pre- and post-processing for numerical simulation.

SCF can simulate various coolant systems, including water (IAPWS 97), lead, lead-bismuth, sodium, helium, and air. It solves mass, momentum enthalpy conservation equations with some special techniques, like the flow regime dependent heat transfer models (boiling curve), the flow regime dependent turbulent cross-flow mixing models (equal mass or equal volume), “Critical Heat Flux” correlations, 2D (r-z) fuel pin heat conduction model, pressure drop models including spacers and wire-wraps, simplified correlation-based fuel pin behavior (cracking, swelling, fission gas release) and gap conductance model for the fuel-cladding gap. To simulate the lateral flow in the reactor core, a simplified lateral momentum equation (the convection term was removed) was added to complete the governing equation set. Additionally, the point kinetic model based on transient power calculation was also available in SCF, which enables SCF to coarsely handle the neutronics.

To solve the equation set presented above, several approaches were proposed. The pressure, mass, and enthalpy could be solved fully implicitly for pure upward flow. While for recirculation and natural convection dominated flow, a semi-implicit solution was adopted. Move a further step, the pressure matrix could be solved by direct, SOR or BiCGStab methods and the convection terms in momentum and energy equations are solved by a first-order upwind method. The boron concentration and the point kinetics are somewhat the items which are not tightly bound to the main solution dataflow. So they can be solved explicitly. As to the searching for critical heat flux conditions, an iterative procedure was applied.

SCF has been validated against a wide range of test cases or benchmarks. Appropriate experiments at different scales representing different reactor types are selected for post-test investigations with SCF. The extensive experimental data obtained at the NUPEC BFBT facility and the NUPEC PFBT was chosen to validate the SCF BWR and PWR module respectively. The data gained from the Karlsruhe compact sodium boiling loop (KNS) was chosen to validate the liquid metal module of SCF. The result of SCF showed good agreement with the experimental data. (Sanchez, Imke und Ivanov 2010)

3.2 ECI for SCF

The major advantage of ECI is the capability to parallelize the calculation of the coupling system randomly. It means that the coupled codes may be potentially extended to be run on powerful distributed computing systems. However, this natural merit also leads to a crucial challenge for the coupled system: synchronization. The precondition to implementing the coupling based on ECI is to assure that the “right data” is transferred at the “right place” and at the “right time”. This is a complex task, especially for the multi-task parallel system. Hence, the key challenges for the development of an ECI-module for SCF are the questions: how to determine which data to be transferred at which time and through which channel to implement tight and accurate data synchronization between TRACE and SCF.

The ECI-module adopts 17 build-in synchronization points for TRACE-TRACE coupling, which strictly reflects the solving procedures of the TRACE set of equations (in an implicit coupling approach). However, since the logic of the SCF solver engine differs from that of TRACE, only a subset of the synchronization points are used for the coupling of TRACE and SCF. To develop the ECI-module, the SCF-source code was re-organized as the primary step. Then, the ECI-

module was developed and implemented into SCF. In Figure 3-2, the difference of the data transfer between two TRACE-processes and between TRACE and SCF is exhibited, which helps readers to seize the ideas quickly.

Before the execution of the codes, the broad channels for tasks are established first. Then, each code reads the inputs and does the initialization. After the identification of the components and data belonging to other tasks, the channels between components and variables are established sequentially. Now, the system is ready for data transfer. Some global flags and geometry parameters are transferred at synchronization point 2 i.e., before the time-step iteration loop. When calculation enters into the time-step loop, at each time step, the time-step size is calculated first. Then, the time-step size and some other dynamic flags are transferred at synchronization points 3 and 4. Afterward, the real numerical solution is performed. Finally, the computational results of the current time step are transferred. According to this coupling approach between TRACE and SCF, it can be inferred that any other tasks at the synchronization point 1 and point 5 to 16 are blocked since they are only needed for a TRACE to TRACE Coupling.

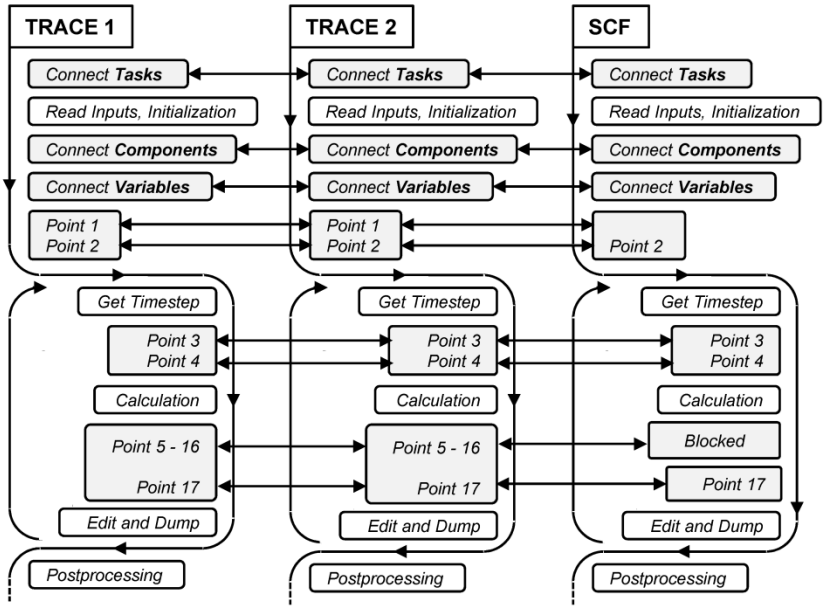


Figure 3-2 The Data Transfer Scheme Between TRACE and SCF in the Frame of the ECI-based Coupling

With the implementation of ECI in SCF, the multi-tasking capability is also feasible. In this coupling approach, SCF simulates the core and TRACE for the rest of the RPV using the 3D VESSEL-component. In case that a TRACE-problem is solved using the multi-tasking capability, more than one TRACE-executables could be present. In such a situation, SCF must be able to identify the target TRACE containing the VESSEL-component to establish the communication channel between TRACE and SCF while all other channels with TRACE that not contain the VESSEL-component are in silence mode. In general terms, it can be done also manually by specifying special parameters in the source code. However, this is not the most convenient option since it complicates the maintenance of the coupled code. Hence, an automatic mechanism that enables SCF to find the target TRACE among all the different tasks was developed and implemented. A scheme of this automatic approach including the different function is illustrated in Figure 3-3.

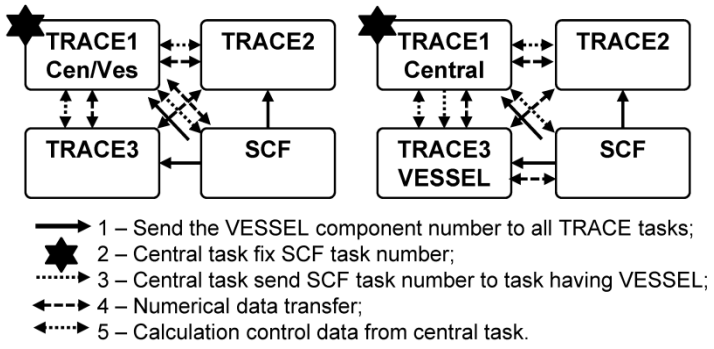


Figure 3-3 Automatic Mechanism of ECI in SCF to Identify the "Target TRACE" Using the Auto-location Function

1) First, SCF broadcasts the "wish number" of the VESSEL-component to all TRACE tasks.
2) Then, each TRACE checks whether they have the VESSEL-component.
3) If the "central TRACE-simulation" contains the target VESSEL, it writes an SCF's task number into memory and establishes the communication channel for the numerical data transfer.
4) If the "central TRACE-simulation" does not have the target VESSEL, it broadcasts the SCF's task number to all other TRACE-simulations. The TRACE-simulation with the VESSEL-component writes the SCF's task number into memory and establishes the communication channel with SCF for numerical data transfer.
5) Next, numerical data is exchanged through numerical channels.
6) At the same time, the variables controlling the calculation are broadcasted by the "central TRACE-simulation" to all other involved codes or tasks. They are also gathered from all other tasks to the central task.

3.3 Adapt SCF to the Multi-scale Coupling System

As section 3.2 describes, a customized ECI was developed for SCF. As a supplementary, several new logics are also developed and implemented into the SCF source. Details of the new ECI and new logics to SCF are summarized in Appendix B. As to TRACE in the coupling system, they use an exactly integrated TRACE model without any items missing. The condition is quite different for SCF. Now that TRACE has no special request for transferring data, this job has to be done by SCF. That is to say, SCF will clarify the data which must be extracted from and given to TRACE so that ECI will get the information on which data to be transferred to where. So, similar to TRACE, several new input parameters should be defined in the SCF input file to tell SCF some very basic information of the TRACE model at the very beginning of the execution. There are four new input parameters for SCF:

1) Target component number: *componum*. The working procedure of ECI could be divided into four steps: connect tasks; connect components; connect variables; transfer data. For step 2, ECI will build a list that stores the missing components' number and a list which stores the existing components' number of the local task. The connections of components from different tasks are exactly established by referencing the two lists. As mentioned

above, TRACE has no missing components but its existing components' numbers are extracted by its ECI anyway to form the corresponding list, which provides a point to be recognized by SCF. However, SCF has no such component properties as TRACE, which means its ECI will generate two empty lists. The existing component list could be empty since the TRACE missing component list is also empty. However, there must be at least one item in the SCF missing component list which could provide the other point of the connection. Thus a new parameter stating the missing component number should be implemented to the SCF data structure. Anyone from the TRACE existing component list could be set to this parameter. However, since there will always be a VESSEL in TRACE, the VESSEL number could be the most ideal option.

2) The number of the VESSEL core radial section which connects the newly added FILLs and BREAKs: *radial_n_f* and *radial_n_b*. People should make sure how many radial rings belongs to the core and how many rings should connect newly added FILLs and BREAKs. Most of the transferred data are arrays that usually have an allocable property and their dimensions must be consistent with each other. All the arrays from the two codes are allocated before the ECI process, which means data associated with the array dimension must be predefined by input. This operation is only for SCF since all the dimension data comes from TRACE VESSEL whose properties have already been fixed. Moreover, two values are closely associated with the array's dimension: the VESSEL radial section number and its angular section number, defined as radial_n and angular_n respectively.
3) The number of the VESSEL angular section: *angular_n*.

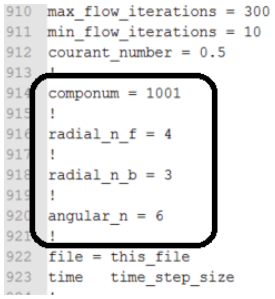


Figure 3-4 New Input Parameters in SCF Input File

All the four input parameters should be integer defined within the "calculation_control" section of the input file (Figure 3-4). Also, the code was given a new capability to generate an ECI log file for SCF named submsg which could print ECI working steps. While special attention should be paid to the file flag "imout" whose value should be changed from 7 to 15 to avoid unexpected file conflicts. Additionally, the other four suggestions should be followed for the benefit of successful running.

1) The input file's name should be input.txt rather than a .inp file since SCF under this running mode is started by a driver programmed in Java instead of a batch.
2) The stop time of SCF should be set to a value larger than that defined by TRACE. This is because TRACE is designed to take charge of the whole computing process and have the authority to run, terminate and stop the processes. It is not a suggestion but a rule which must be followed to avoid unnecessary failures.

3) The number of time steps should be set to a pretty large value so that SCF will not reach this termination criterion ahead of the true coupling system termination.
4) The name length of the variables which are transferred by ECI shouldn't exceed 8. Otherwise, ECI would ignore the excess part which may lead to errors.
5) The task name of SCF on the taskList file must be "SCF".

There are several variables whose value will be sent by TRACE forcibly. They are allStdy, longEd, shortEd, graphEd and rstDump. The variables are used to control the process running, result editing and dumping, etc. For TRACE-TRACE, their functions are fully used because of the complete synchronization scheme. For TRACE-SCF, data transfer of those variables is also performed however, calculation of SCF will not be influenced by them. Additionally, only four synchronization points are adopted for data transfer. Unlike TRACE, modification of the ECI module ExTransferM.f90 is not necessary. The only thing to do is just don't call the transfer related subroutines at these points. Finally, the last point calling for special attention is the time domain definition. The rules shown in Table 3-1 have to be followed otherwise SCF will terminate in error.

Table 3-1 The Definition of SCF Time Domain for Multi-scale Coupling

Steady State	set_boron_transport= off; stop_time> 0.0; 0<time_step< 1000; number_time_step> 0
Transient	stop_time> 0.0; 0<time_step< 1000; number_time_step> 0

4 DESCRIPTION OF THE ECI-BASED COUPLING OF TRACE AND SUBCHANFLOW (SCF)

Generally speaking, the coupling of TRACE and SCF involves two segments: spatial coupling and numerical coupling. Two sections were arranged to describe the two items. Before that, the overall guidance of running the codes is required to get the audience an intuitionistic “figure” of the codes.

4.1 Run the Codes

Besides the ECI based coupling codes, standalone TRACE and SCF should also be included as the base of coupling. The required elements and the starting procedure of the two standalone codes are quite similar: One executable, one input file (ignore the restarts, the input file for TRACE is tracin and input.txt is for SCF) and simply start the executable in the input file’s directory (Figure 4-1).

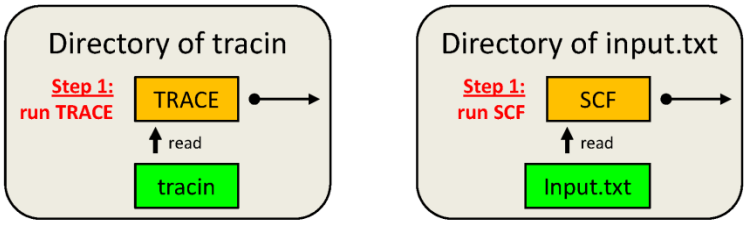


Figure 4-1 Run Standalone TRACE and SCF

The running required items for ECI-based parallel coupling of TRACE and SCF (TRACE/SCF-ECI) include two executables: TARCE and SCF, two input files: tracin and input.txt, a task information file: taskList (Figure 4-2) and a driver. This is a multi-task system which has already been sufficiently explained. Two steps should be done to start the processes (Figure 4-3).

```
Task1     directory of TRACE/trace.exe
TRACE     directory of the tracin
Task2     directory of TRACE/subchanflow.exe
SCF       directory of the input.txt
```

Figure 4-2 TaskList for TRACE/SCF-ECI

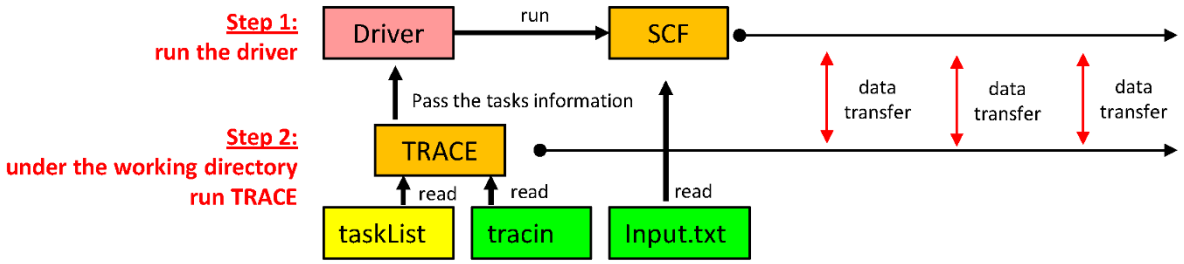
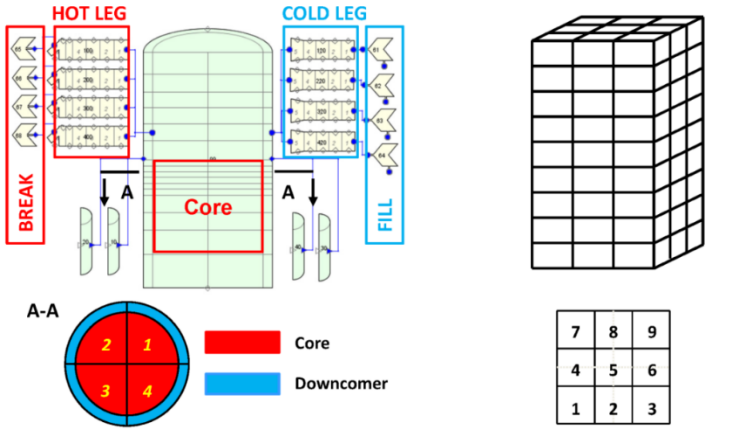


Figure 4-3 Run TRACE/SCF-ECI

Figure 4-3 is similar to Figure 2-4. However, two differences still exist. One is that Figure 2-4 illustrates a multi-task system that only includes TRACE while Figure 4-3 includes both TRACE and SCF. The other one is that the former includes three tasks while the latter includes only two. Actually, in a TRACE/SCF-ECI system, SCF could be flexibly coupled with more than one TRACE. However, the coupling strategy of TRACE and SCF is quite different from that of TRACE themselves, though they are in the same TRACE/SCF-ECI system. This feature was already illustrated in section 3.2.

4.2 Spatial Coupling

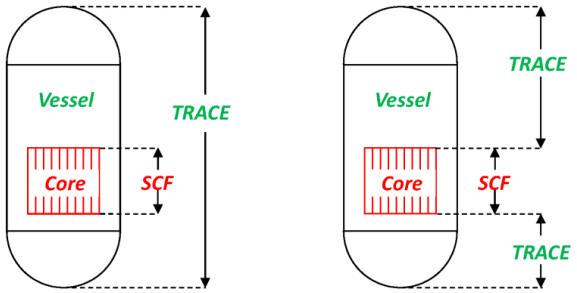
To illustrate the spatial coupling vividly, a TRACE model having four loops (Figure 4-4a) and an SCF model having nine channels (Figure 4-4b) are proposed. Both of them are originally standalone models which are also the base models for the full section. There are two elements involved in the spatial coupling: the computational domain coupling and the field mapping. Both of the two items will be illustrated with the base models.



a. TRACE stansalone model **b. SCF standalone model**

Figure 4-4 TRACE Standalone Model and SCF Standalone Model

The domain coupling could be further classified into overlapping and decomposition coupling. For the former strategy, TRACE will calculate the whole domain and SCF will calculate the core area in the vessel (Figure 4-5a). Under this mode, TRACE will send its data at the top and bottom of the core to SCF as boundary conditions and SCF return refined data to TRACE as an additional source to the core data structure in TRACE. For the latter strategy, only SCF calculates the core area and TRACE will calculate other domains except for the core (Figure 4-5b). Data transfer only performs at the core inlet and outlet. The data from TRACE are treated as boundary conditions to SCF and the data from SCF are also treated as boundary conditions to TRACE. This is a much convenient and efficient way since TRACE has exact boundary components as FILL and BREAK which can directly introduce the accurate influence from SCF to its domains. Consequently, the decomposition approach was selected for the domain coupling.



a. Overlapping b. Decomposition

Figure 4-5 Domain Coupling of TRACE and SCF

To implement the non-overlapping coupling, the TRACE originally standalone model should be properly modified (Figure 4-6). The changes take place in three positions.

1) The core area in TRACE is blocked. The code still holds the core data within its data structure in practice. However, since the axial flow areas in the core are set to zero, the data becomes meaningless.
2) New FILLs and PIPEs are added and connected to the core's upper boundary. The new FILLs receive data from SCF and the new PIPEs return data to SCF. Detailing of the mechanism will be described later on.
3) New BREAKs and PIPEs are added and connected to the core's lower boundary. The new BREAKs receive data from SCF and the new PIPEs return data to SCF.

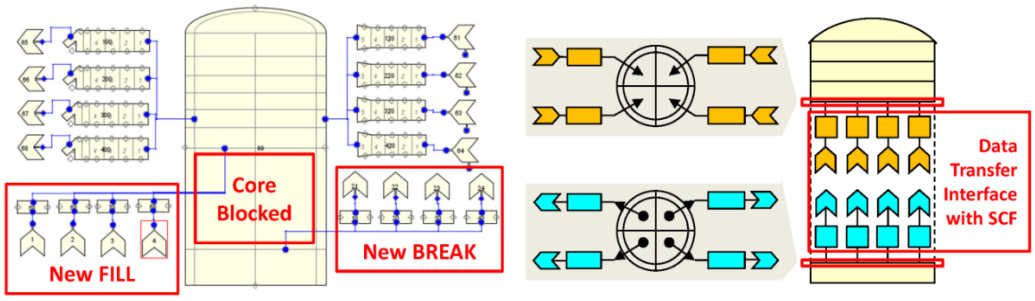


Figure 4-6 TRACE Model for Coupling

Compared with the TRACE model, there is nothing to change for the SCF model. Nevertheless, new subroutines and logics should be merged into the source of both the two codes to get and put appropriate data from and to their data structure. With all models and source code modifications done, the domain coupling was established (Figure 4-7). Normally, four sets of data: mass flow rate, temperature, solute concentration, and pressure are to be transferred. The basic working flow of the coupling system could be:

1) Pressures from the FILL-connected PIPEs in TRACE are extracted and send to the upper boundary of SCF based on the position relationships between TRACE radial/azimuthal sections and SCF channels. Mass flow rate, coolant temperature and solute concentration from the BREAK-connected PIPEs in TRACE are extracted and send to the lower boundary of SCF.
2) Mass flow rate, coolant temperature and solute concentration from the upper boundary of SCF are extracted and send to the FILLs in TRACE. Pressures from the lower boundary of SCF are extracted and send to BREAKs in TRACE.

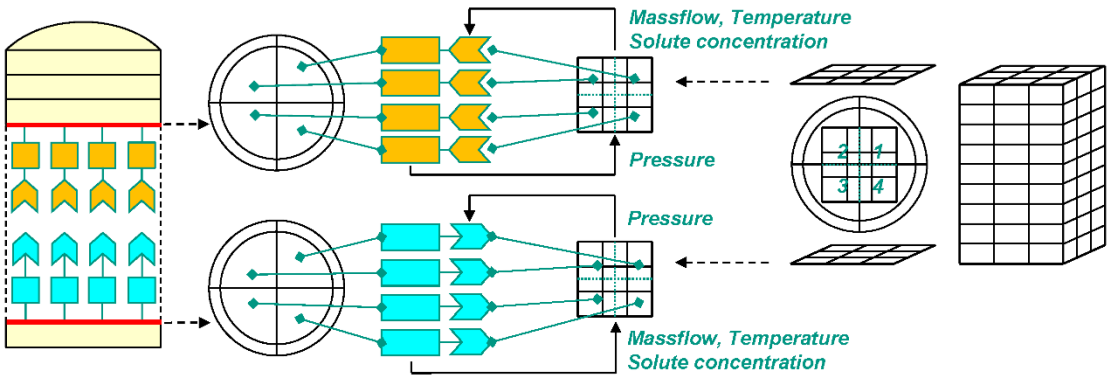


Figure 4-7 Domain Coupling of TRACE and SCF

According to the two steps, the two boundary components in TRACE: FILL and BREAK are only the place to put data while not getting data. This is because data in such components don't update along with the calculation. Thus additional data-updated PIPEs have to be used for getting data from TRACE. The other essential reason is that the boundary components can't just directly

connect the VESSEL component. Hence PIPEs have to be used as “intermediary”. Another doubt arising here could be: how to fix the FILLs and BREAKs for putting data since there could be other FILLs and BREAKs which are not the data transfer interfaces in the system (Figure 4-6, the four FILLs on the cold leg and the four BREAKs on the hot leg). One solution is writing the interfaces’ information directly to the source to specify their roles. However, it is quite an inefficient way because every time a new model was developed, the source code has to be updated and compiled once again to fit the model’s configuration. A much more efficient way was proposed to enable the code to recognize the interfaces automatically without the change and recompiling of the source. In this approach, a new input parameter for TRACE was designed: bound_l (call back to section 2.4), helping to implement this function (Figure 4-8).

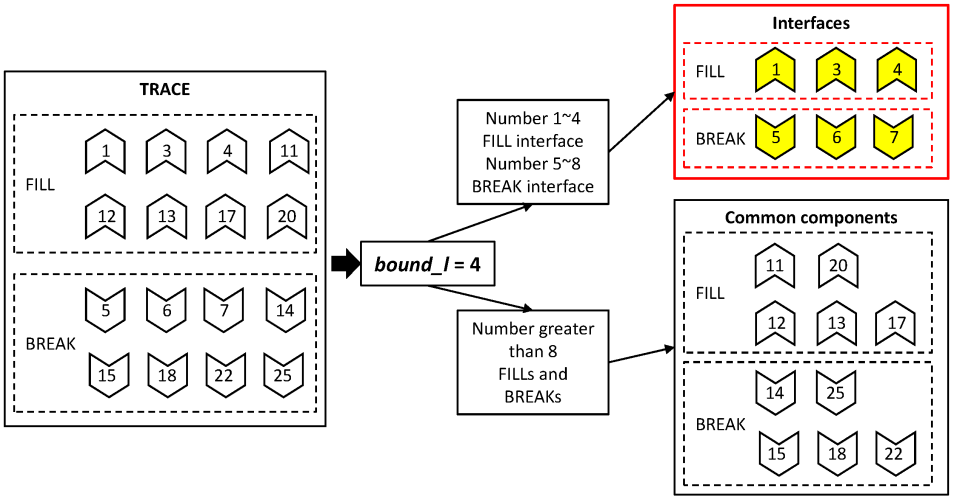


Figure 4-8 Strategy on TRACE Coupling Interface Recognition

Set ***bound_l*** to x. FILLs numbered from 1 to x and BREAKs numbered from x+1 to 2x will be recognized as data transfer interfaces. Other FILLs and BREAKs whose component numbers are greater than 2x are treated as common components. The PIPEs which connect the interfaces can also be recognized at the same time. In Figure 4-6, Figure 4-7 and Figure 4-8, the number of interface FILLs and BREAKs are the same. Nevertheless, they can be different and their arrangement can be very flexible (Figure 4-9). Special effort must be paid to coding the source, of cause.

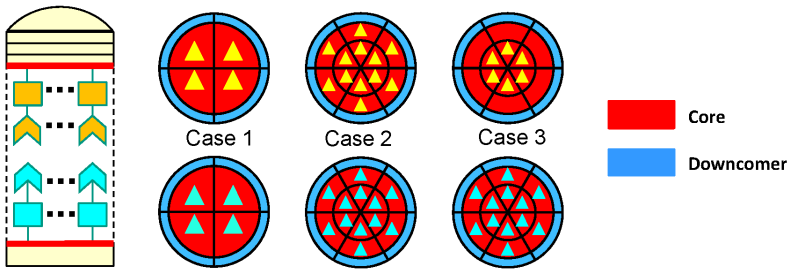


Figure 4-9 FILL and BREAK Interfaces to VESSEL

Normally, the number of interface FILLs and BREAKs keeps consistent with the radial and azimuthal nodalization in the TRACE model (Figure 4-9 – Case1 and Case2). Nevertheless, those interface components can also be flexibly connected to the VESSEL (Figure 4-9 – Case3). Three new SCF input parameters are related to this feature (call back to section 3.3):

1) ***angular_n***: the azimuthal sections of the TRACE model.
2) ***radial_n_f***: the radial rings of the core area of the TRACE model.
3) ***radial_n_b***: the radial rings of the core area of the TRACE model.

All of the three new SCF parameters are about the TRACE model geometry. The reason why SCF needs TRACE geometry information is that SCF needs such data to allocate arrays for data transfer. The three new inputs must be properly defined in the SCF input file within a TRACE/SCF-ECI system. SCF in the TRACE/SCF-IS system can directly get the data by passing parameters through functions. Corresponding to Figure 4-9, the values of these new inputs are listed in Table 4-1.

Table 4-1 New Input Parameters for SCF in TRACE/SCF-ECI System

	Case 1	Case 2	Case 3
angular_n	4	6	6
radial_n_f	1	2	2
radial_n_b	1	2	1

Now, the domain coupling of TRACE and SCF was sufficiently described. However, an essential problem hiding in Figure 4-7 is how to map the four cylindrical sections of TRACE to the nine cartesian channels of SCF. This is known as the field mapping and will be explained by the next section.

Field Mapping

Figure 4-10 presents the positional correspondence of the TRACE sections and SCF channels captured from Figure 4-7. The boundary data from the codes should be first translated before the transfer since several channels may contribute to one section and several sections may also contribute to one channel in turn, according to their overlapping area proportions. This is a process normally known as mapping. And to make the mapping as accurate as possible, the contribution

proportions from each TRACE section to each SCF channel and from each SCF channel to each TRACE section should be properly calculated.

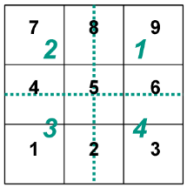


Figure 4-10 Positional Correspondence of the Four Sections and Nine Channels

The contributions are easy to calculate for this case since there are only nine channels and four sections and all of them have regular, uniform shapes and are centrosymmetric distributed. The proportions can be represented by two arrays (Figure 4-11).

SCF channels

TRACE sections	1	2	3	4	5	6	7	8	9
1	0	0	0	0	1/9	2/9	0	2/9	4/9
2	0	0	0	2/9	1/9	0	4/9	2/9	0
3	4/9	2/9	0	2/9	1/9	0	0	0	0
4	0	2/9	4/9	0	1/9	2/9	0	0	0

a. from SCF to TRACE

SCF channels

TRACE sections	1	2	3	4	5	6	7	8	9
1	0	0	0	0	1/4	1/2	0	1/2	1
2	0	0	0	1/2	1/4	0	1	1/2	0
3	1	1/2	0	1/2	1/4	0	0	0	0
4	0	1/2	1	0	1/4	1/2	0	0	0

b. from TRACE to SCF

Figure 4-11 Two Arrays for Mapping Data

Figure 4-11a is the array translating data from SCF to TRACE. Take notice that the sum of each row is always one. The items in the array represent contributions from a specified channel to a specified section. Figure 4-11b is the array translating data from TRACE to SCF. The sum of each column is always one. The items in this array represent contributions from a specified section to a specified channel. It is easy to get the arrays manually since there are not too many channels or sections and they are regularly arranged. However, when it comes to reality, hundreds of channels or sections might be enclosed in the calculation and they may have irregular areas, shapes, and arrangements. Take the model shown in Figure 4-12 for instance, SCF has 163 channels and TRACE has 16 core area sections. Then take the red-line marked channel for instance, its boundary data comes from four TRACE sections: 2, 3, 10 and 11, multiplied by their contribution proportions. The channel itself, in turn, contributes to the four TRACE sections as well. The positional relationship is quite complicated and the generation of the two mapping arrays is a tough issue.

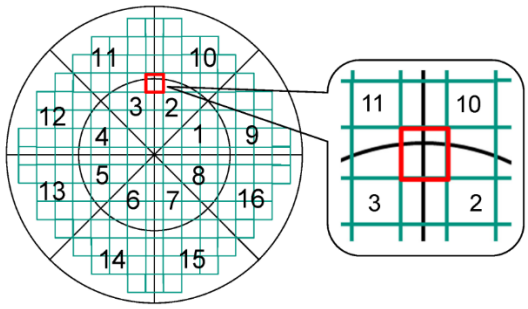


Figure 4-12 A More Complicated Model for Data Mapping

In this work, a subroutine named “mapping.f90” was developed to generate the two arrays for cases whose sections and channels have arbitrary nodalizations, areas, shapes, and locations, automatically. The subroutine can handle more complicated cases than that in Figure 4-12. It is a comprehensive tool covering all possible configurations. Since the overlapping area calculation of an arbitrary shape channel is meaningless (SCF only need the area of the channels and don’t care about their shapes), one key assumption assuming that all the channels are circles is proposed. The subroutine will first classify the channels into two basic classes according to their relative positions to the TRACE center. The channels which cover the center point belong to class one and are further classified into three sub-classes according to their relative positions to the first TRACE radial section. Details of this subroutine are in Appendix H.

4.3 Temporal Coupling

The temporal coupling is understood as the process to synchronize the data transfer between the codes during the stationary (iterations) and transient time advancement considering the different data flow of both TRACE and SCF. The temporal coupling organizes the data flow between the two codes consistently. There are mainly three methods to implement the temporal coupling (Zerkak, Kozlowski and Gajev 2015) for nuclear reactor analysis.

1) The Operating Splitter (OS) method, some also call it Operation Splitter Semi-Implicit (OSSI) (Zhang, Guo, et al., An Assessment of Coupling Algorithms in HTR Simulator TINTE 2018): It could be understood as an explicit coupling approach. Data transfer is carried out only once at the beginning or end of each time step. No check of the convergence of the codes’ result within a time step is one. Hence, the results may be inconsistent with each other.
2) Picard or fixed-point-iteration method (Hamilton, et al. 2016) (Zhang, Guo and Lu, et al., The Improvement of Coupling Method in TINTE by Fully Implicit Scheme 2018): It is kind of a semi-implicit coupling. Data transfer is performed many times until all the codes’ result convergence within one time step. Hence, it is more robust than the OS method. However, it is complicated to be implemented.
3) Jacobian-Free Newton-Krylov (JFNK) method (Knoll and Keyes 2004): It is kind of an implicit coupling method. The whole equation set is solved together. This is a very efficient and robust coupling approach but needs significant modification to codes source and is almost impossible for two isolated already-well developed codes e.g., TRACE and SCF.

It is worth noting here that there are lots of other techniques for temporal coupling e.g., staggered time grids (Aragonés, et al. 2004), Anderson acceleration (Hamilton, et al. 2016) (Walker and Ni 2011), Approximate Block Newton (ABN) methods (Yeckel, Lun and Derby 2009), NLEm (Zhang, Guo and Li, et al. 2018). Considering the stability of the numeric solution of TRACE and SCF, along with the simplicity requirement by the coupling, the OS approach was followed here. Hereafter, the implemented OS coupling scheme for the steady-state and transient coupling of TRACE and SCF will be presented.

Within the OS temporal coupling, both steady-state and transient calculations are available. From another perspective, TRACE/SCF-ECI is an inter-process parallel coupling system (SCF can utilize OpenMP to parallel its process, however, it is not multi-process parallel running), in which TRACE and SCF are two self-governed executables. TRACE and SCF communicate with each other via ECI which is the key project in this section. Based on the ECI explanation in Appendix F, the general workflow of ECI could be represented as Figure 4-13 shows, which includes four basic working steps: task connection, component connection, variable connection and the real data transfer which includes 17 synchronization points.

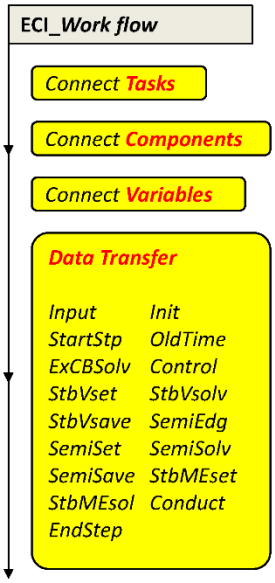
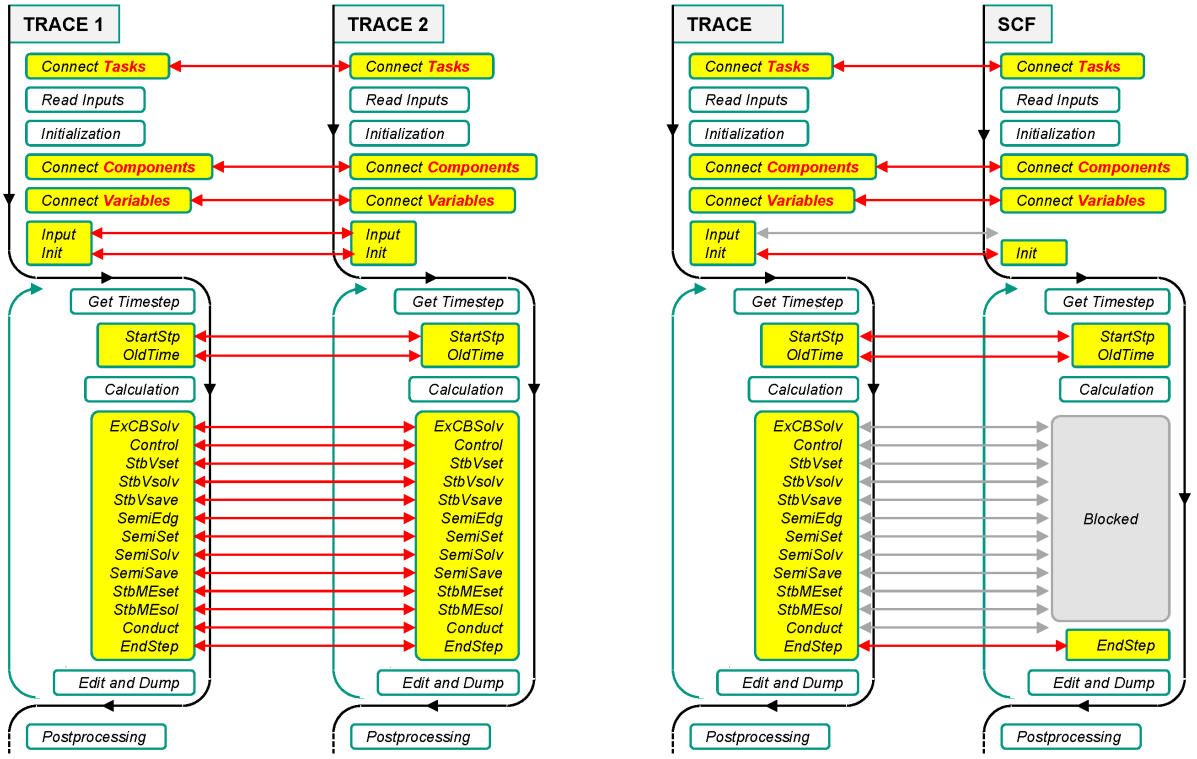


Figure 4-13 General Workflow of ECI

ECI is already available for pure multi-TRACE coupling. As stated in section 2.2, it keeps a close relationship with the TRACE data structure. The coupling of TRACE and TRACE via ECI is represented by Figure 4-14a associated with Figure 2-1. New ECI should be designed for SCF to implement the TRACE/SCF-ECI coupling system. Normally, the ECI for TRACE can be referenced as a template. However, dozens of its logics have to be modified to fit SCF's data structure. This is mainly due to ECI's three features which have been stated in sections 2.2 and 3.2. They are:

1) ECI is closely related to the code's data structure.
2) ECI is specific for each code.
3) The multi-task system established by ECI is a server-less system.

Take into account the fact that numeric of SCF and TRACE differ from each other, SCF is not able to follow all the 17 synchronization points for data transfer. In this work, four main synchronization points are applied for SCF ECI design: Init, StartStp, Oldtime and EndStep (Figure 4-14b). What's more, due to the inconsistent running procedures between the two codes (the first step of SCF will always be a SS calculation while TRACE always calls its transient subroutines), their synchronization becomes a tough task. Several new flags have to be designed for this purpose.



a. multi-TRACE

b. TRACE/SCF

Figure 4-14 ECI Workflow in Multi-TRACE and TRACE/SCF Coupling System

Steady-State of TRACE/SCF-ECI

Data transfer performs at two locations: the interfaces between TRACE's upper plenum and SCF's core outlet and the interfaces between TRACE's lower plenum and SCF's core inlet. Five major parameters are transferred. They are pressure, mass flow rate, coolant temperature, solute concentration, and time-step size. As discussed at the very beginning of section 3.2, the most essential difficulty of the coupling is how to keep the two codes' tightly synchronization (This is not the well-known normal synchronization focusing on time and data. It here actually refers to the coordination of the two codes workflows). To accomplish the goal, two new flags that coordinate the code synchronization were designed. They are:

1) ***tranflag***: it coordinates the data transfer at different synchronization points. Three integer values could be set to this flag.
   a. 0 – the initial value, means only the data transfer at the Init point is possible.

   b. 1 – TRACE run to SS for the first time. Data transfer at StartStp point is activated. But the data could only transfer from TRACE to SCF.
   c. 2 – Data now could transfer between TRACE and SCF at all synchronization points.
2) ***rflag*****:** the flag is specially designed for SS of TRACE/SCF-ECI. It is controlled by TRACE and decides whether the data could be transferred.
   a. 0 – initial value, all data transfer is not permitted.
   b. 1 – data transfer at all points could be performed.

Two TRACE built-in flags also dominate in the data transfer process in addition to the two new flags. They are:

1) ***iSteady*****:** indicates the system SS status. It is specially used for the SS calculation. Two integer values are available for this flag:
   a. 0 – the system has not reached the SS.
   b. 1 – the system converges and the calculation will terminate.
2) ***allStdy*****:** this is a special flag for the multi-task system. Two integer values are available for this flag:
   a. 0 – at least one task doesn't converge yet. The whole system still keeps running.
   b. 1 – all the tasks reach SS. All the tasks terminate. The calculation is over.

After the sufficient introduction of the calculation-control flags, it is time to illustrate the system working procedures (Figure 4-15).

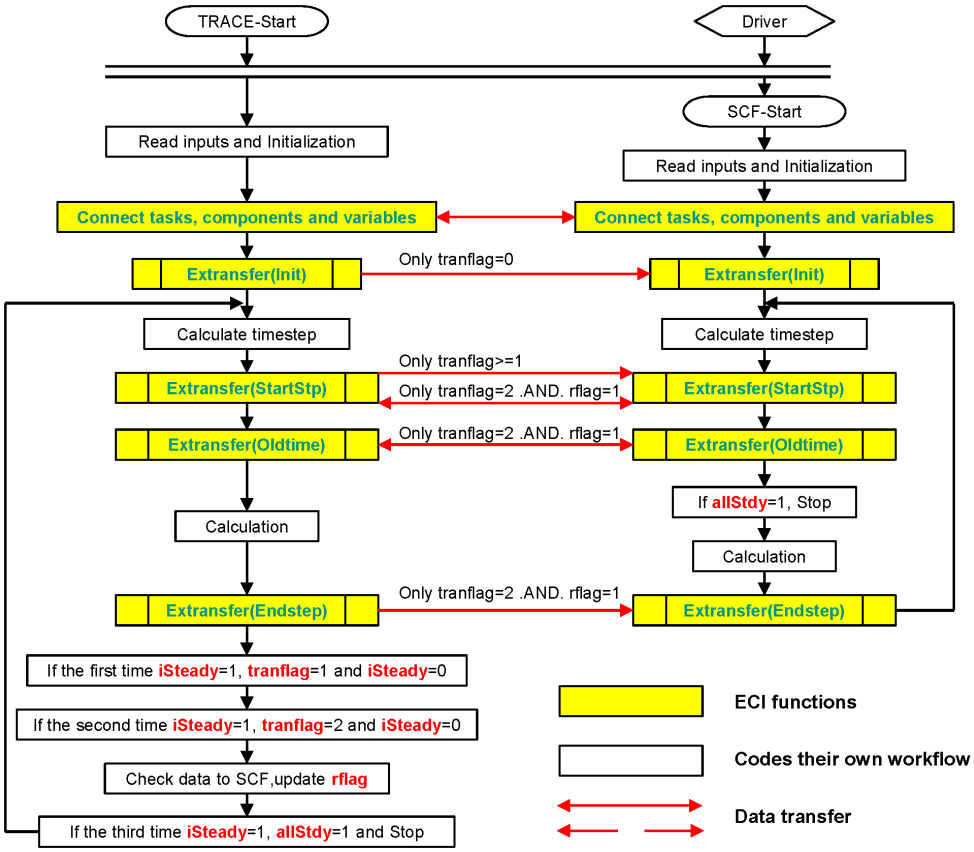


Figure 4-15 General Workflow of TRACE/SCF-ECI Steady-State

1) First, run the drier script from any location.
2) Run TRACE under the working directory. TRACE will read in the taskList and send SCF information to the driver which will then run SCF.
3) TRACE and SCF read their input files and do their initialization.
4) ECIs in TRACE and SCF will interact with each other and connect the two tasks, the components, and the variables sequentially.
5) Both the two codes reach the first data transfer synchronization point: Init. Now the transflag maintains its initial value – 0. So the Init data transfer could be performed. The data are all about the TRACE geometry. They are radial_n_f, radial_n_b and angular_n. The three variables and reasons for the transfer have already been well explained in section 3.3.
6) Both of the two codes go into the time-step iteration.
    a. SCF will set its time-step to quite big a value to make a SS step. Then it will stop at the StartStp point since the socket in ECI is in blocking mode and there is no data available from TRACE for the time being. It will keep waiting until there are data available from TRACE.
    b. TRACE will skip all the synchronization points since tranflag is 0. It will run a standalone steady-state iteration until the system converges. The iSteady then will become 1. As a consequence, the tranflag will turn to be 1. Next, iSteady will be reset to 0 (the SS flag passed from SCF is always 1. If iSteady is 1, then the allStdy

will become 1 and the whole calculation will terminate. To keep TRACE running, iSteady must be 0). The rflag will be set to 1.

7) The first interactive running of TRACE and SCF.
   - **a.** TRACE go back to the loop head, flows down and transfer the SS boundary data (mass flow rate, coolant temperature, solute concentration, and pressure, shown in Figure 4-7) to SCF at the StartStp point since tranflag is 1 now. For the current step, no data is transferred from SCF, which is just because SCF has not run a step yet. TRACE then will skip other points and run to the loop end. Since the last step has already reached steady-state, the result of the current step must be at SS status too. Thus iSteady becomes 1 for the second time. Consequently, tranflag will turn to 2 and iSteady will be reset to 0.
   - **b.** SCF gets the boundary data from TRACE at the StartStp point and proceeds. The steady-state data will be obtained once SCF finished the first step.
8) The second step of the interactive running of TRACE and SCF.
   - **a.** At the StartStp point, TRACE sends the boundary data from its last time-step to SCF and receives the boundary data from SCF. Then at the Oldtime point, TRACE sends its current time-step size to SCF and later on receives the global time-step size from SCF. With all the boundary conditions and time-step size fixed, TRACE will do one step calculation and sends some editing-related flags to SCF which includes: longEd, shortEd, graphEd and rstDump. Then TRACE will compare the current-to-SCF data with the previous-to-SCF data and check whether there is over-criterion change. If the answer is positive, rflag will remain 1, otherwise, it will be reset to 0.
   - **b.** At the StartStp point, SCF sends its steady-state step boundary data to TRACE and receives the corresponding data from TRACE. Then at the Oldtime point, SCF receives the current time-step size of TRACE and returns to TRACE just as it is. SCF will keep its time-step size the same as its SS step and zero its step counter thus the current step will be treated as a steady-state step again. With all the data fixed, SCF will do the current step calculation and then at the Endstep point, receive the editing-related flags from TRACE.
9) From the third step to the antepenultimate step.
   - **a.** If rflag equals 1, TRACE first sends and receives the boundary data to and from SCF at the StartStp point. Next, it will send the time-step size to SCF and receive the global time-step size from SCF at the Oldtime point. Then the current step calculation is done. The flags are sent to SCF at the Endstep points. At last, TRACE will compare the current-to-SCF data with the previous-to-SCF data and check whether there is an over-criterion change. If the answer is positive, rflag will remain 1, otherwise, it will be reset to 0. If rflag is 0, all the points are skipped and TRACE will run standalone. At the last step, TRACE reaches the SS for the third time and iSteady is set to 1. Sequentially the rflag now is forced to 0.
   - **b.** If rflag in TRACE equals 1, SCF will send its previous transient step boundary data to TRACE and receive the boundary data from TRACE at the StartStp point. Then at the Oldtime point, SCF receives the current time-step size of TRACE and returns to TRACE just as it is. SCF will keep its time-step size the same as its steady-state step and zero its step counter thus the current step will be treated as a SS step again. With all the data fixed, SCF will do the current step calculation and then at the Endstep point, receive the editing-related flags from TRACE. If rfalg in TRACE is 0, SCF will pause and wait for the data from TRAC

10) At the penultimate step.
    a. The operations of TRACE are similar to previous steps. The difference is, TRACE will change allStdy from 0 to 1 in the inner logics at the Oldtime point but nothing is transferred since rflag is 0 now.
    b. SCF is waiting now.
11) At the last time-step. All the synchronization points are forced to be available regardless of the value of rflag.
    a. TRACE will send data to SCF at all the points. Within the inner logic at the Oldtime point, it will set the terminate flag to true and then stops the calculation.
    b. SCF receives allStdy=1 from TRACE and terminate its running right after the transfer at the Oldtime point.
12) TRACE and SCF will do their post-processing.

Though the above steps 1-12 and Figure 4-15 almost illustrate the same general outline of TRACE/SCF-ECI steady-state, the 12 steps contain more details and don't agree with the figure. The reason is that SCF can work with more than one TRACE process and may have different data transfer with different TRACE. It could be a little bit complicated trying to cover all the details of a multi-task system within just one diagram. Nevertheless, we will do the work more easily. It could be also inferred that SCF always runs the steady-state steps and some SCF steps sometimes can be skipped by TRACE according to the boundary data check. They are exactly the tricks to accelerate the system convergence. The motivations and principles are to be discussed. But before that, points related to the coupling system steady-state calculation should be made clear.

1) TRACE applies a time-approaching way for the SS calculation, which means TRACE is calling its transient subroutines (a little bit different from the real transient) even for its steady-state mode. This process was well illustrated in Figure 2-1.
2) The steady-state mode in SCF only needs a single steady-state step, without iteration, which means the steady-state in TRACE/SCF-ECI can't adopt the steady-state in SCF since time-step iteration is desired.
3) The transient mode in SCF is adopted to implement the steady-state in TRACE/SCF-ECI.
4) The first step of SCF transient is always a steady-state step which is the same with its pure steady-state calculation (section 3.1). The time-step was set to a quite huge value aiming to eliminate the time terms in the equation set. The actual transient begins from the second step.

Normally, the interaction of the two transients can converge properly. However, SCF always takes much longer elapsed time than TRACE for a single time-step, which means SCF's performance is the bottleneck of the whole coupling system. In other words, it contains too many transient details which may lead to unexpected long elapsed time. To introduce the boundary perturbation from TRACE to the SCF domain as fast as possible, the transient calculation steps of SCF within the iteration are all replaced by the SCF SS steps. With this modification, the information from TRACE can deliver all through the SCF core area during just one single step and the data from SCF back to TRACE becomes accurate for the current step. Thus accelerate the convergence. Although the replacement of SCF transient steps by SCF SS steps makes the SS calculation of TRACE/SCF-ECI more efficient, one more improvement was implemented. The motivation comes from the consideration of a condition which is shown in Figure 4-16. Meanings of the curves are:

1) Curve 1: the convergence parameter curve of TRACE on a global scale.
2) Curve 2: the convergence parameter curve of TRACE at the interfaces. The curve stands for the convergence of the data from the TRACE upper plenum-connected and lower plenum-connected PIPEs to SCF.

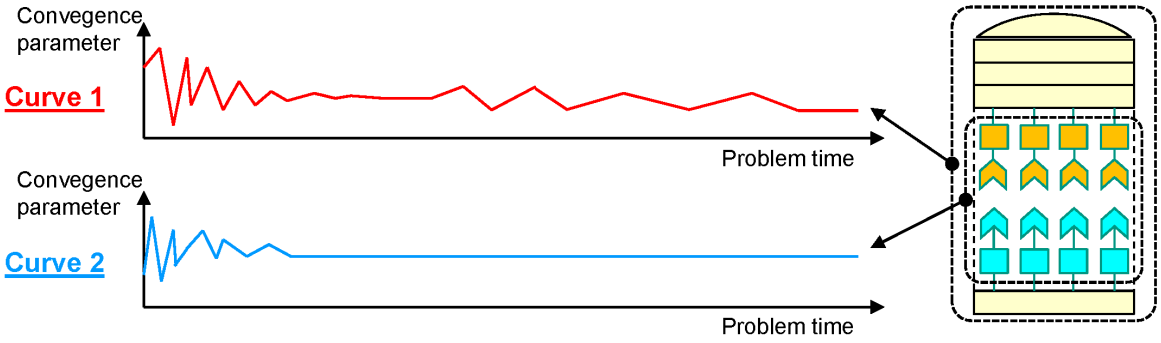


Figure 4-16 Possible TRACE Convergence Curves in Global Scale and Interface Scale

This phenomenon was observed during some test procedures. It is found that the data from TRACE interfaces converge soon sometimes though a pretty long period was consumed to get the whole system converged. This is reasonable since the TRACE model consists of various components and the flow complexity varies. When the data from TRACE to SCF becomes stable, the flow in the lower plenum and upper plenum could still be unsteady. The calculation will proceed, however, the data transfer and SCF steps will become insignificant repeats. What's worse, as stated above, SCF captures the majority elapsed time, those insignificant repeat SCF steps will eat needless computer resources. So the new improvement is adding new judging logics to source code to determine whether SCF is activated or not. A new flag was designed to assist the judgment: rflag (has already been presented in the above 12 steps). The newly implemented function could be illustrated by the following case shown in Figure 4-17, which could also help to understand the 12 steps above.

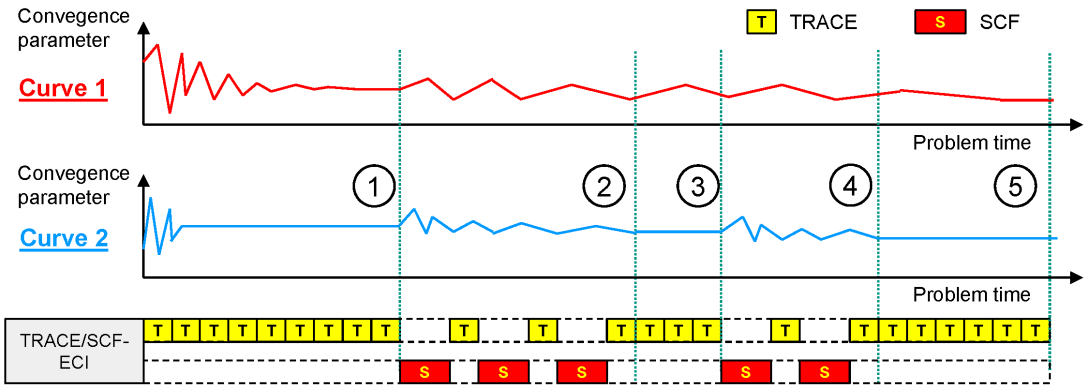


Figure 4-17 Curves for Illustrating the SCF Flexibly Activated Function in TRACE/SCF-IS SS

1) TRACE standalone first runs a SS iteration and reaches the first point in the figure. Then SCF will be activated.

2) TRACE and SCF run an interactive iteration and reach the second point where data from TRACE become stable. rflag is set to 0. SCF is inactivated.
3) TRACE standalone runs to the third point where data from TRACE become over-criterion again. rflag is set to 1. SCF is reactivated.
4) TRACE and SCF run an interactive iteration and reach the fourth point where data from TRACE become stable again. rflag is set to 0. SCF is inactivated.
5) TRACE standalone runs to the steady-state and terminate the calculation.

As stated previously, this trick is exactly an efficient approach to save elapsed time. The mechanism can be vividly illustrated in Figure 4-18. Five SCF are saved by new steady-state than old steady-state in this figure.

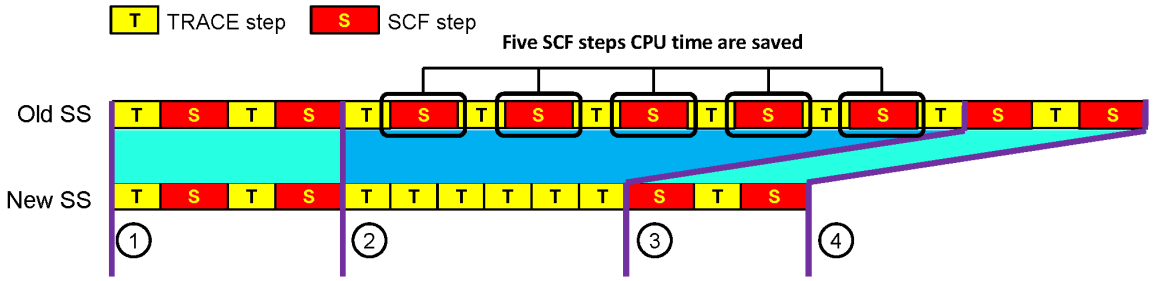


Figure 4-18 Mechanism of the SCF Activated/Inactivated Trick in TRACE/SCF-IS Steady-State

1) TRACE/SCF-ECI starts from the first point and runs an interactive iteration. At the second point, data from TRACE to SCF become stable. SCF is inactivated. This step totally the same for both old steady-state and new steady-state.
2) For new steady-state, TRACE standalone runs to the third point where boundary perturbation becomes over-criterion and SCF is reactivated. For old steady-state, TRACE and SCF run together to the third point though there is no significant feedback from SCF to TRACE. According to the figure, five SCF steps are saved by new steady-state.
3) From point 3 to point 4, old steady-state and new steady-state both run interactive iteration.

Transient of TRACE/SCF-ECI

Compared with the steady-state in TRACE/SCF-ECI, the SCF steps within transient calculation are exactly the real transient steps. Nevertheless, the first step of SCF in the iteration will always be the steady-state step. This feature allows SCF don't need to run a restart case even though the transient calculation should normally run restarts using steady-state results. It can be implied that this transient process is a step to step coupling since TRACE and SCF run one by one sequentially. The general running sequence of the transient now could be represented as Figure 4-19 presents.

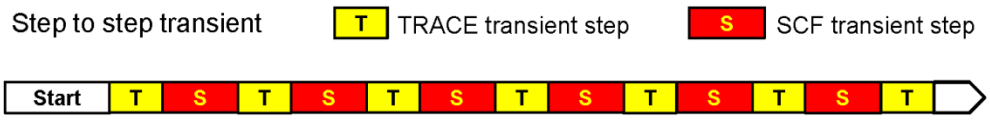


Figure 4-19 Transient of TRACE/SCF-IS in Step-to-Step Mode

The step to step coupling strategy for the transient calculation enables TRACE and SCF to keep tightly synchronization since the time-step size and the step number are always the same. Nevertheless, an SCF time-step skipped strategy was proposed to improve the system's efficiency. The motivation comes from some essential features of the two codes' numeric.

1) The calculation of SCF can be treated as unconditional stable since it applies a fully implicit method to solve the equations. In other words, the time-step of SCF could be a very big value.
2) Normally, the consideration of defining SCF time-step (especially for the upper limit of the time-step size which is defined in the input file) is that the biggest time-step size should be small enough to cover all the transient details.
3) The TRACE time-step size could be quite small compared with SCF's and the real transient.
4) For a single step, SCF takes a much longer time than TRACE, in most cases.

Points 1, 2 and 3 imply that there may be too many SCF steps during the step to step transients since the original time-step size of SCF might be replaced by the TRACE's due to the time-step determination mechanism, which would result in too many unnecessary SCF steps. Point 4 makes this matter even worse. The redundant SCF steps could consume heavy but unnecessary computer resources. Consequently, the SCF step skipped strategy was proposed. Its general process is displayed in Figure 4-20 by comparison with the step to step transient. The elapsed time could be significantly cut by this new trick. The illustration follows the figure.

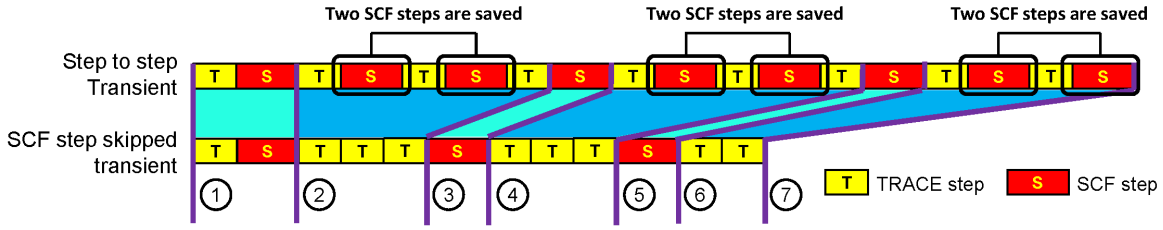


Figure 4-20 Mechanism of the SCF Step Skipped Trick in TRACE/SCF-ECI Transient

1) The two modes both start form point 1, run one TRACE and one SCF in sequence and reach point 2.
2) From point 2 to point 3, the step to step mode run TRACE and SCF one by one while the SCF step skipped mode will only run TRACE. Two SCF steps are skipped and the corresponding elapsed time is saved.
3) Both of the two codes will run an SCF step reaching point 4.

4) From point 4 to point 5, the step to step mode run TRACE and SCF one by one while the SCF step skipped mode will only run TRACE. Two SCF steps are skipped and the corresponding elapsed time is saved.
5) Both of the two codes will run an SCF step reaching point 6.
6) From point 6 to point 7, the step to step mode run TRACE and SCF one by one while the SCF step skipped mode will only run TRACE. Two SCF steps are skipped and the corresponding elapsed time is saved. The calculation terminates here.

As stated in the previous sections, the most essential difficulty of the steady-state coupling is how to keep the two codes' tightly synchronization. This is also the same issue to be handled by the transient of TRACE/SCF-ECI. To accomplish the goal, three new flags that coordinate the code workflow were designed. They are:

1) **tranflag:** despite the same name with tranflag in TRACE/SCF-IS, this transflag has nothing to do with the steady-state conditions. It coordinates the data transfer at different synchronization points. Three integer values could be set to this flag.
    a. 0 – the initial value. means only the data transfer at the Init point is possible.
    b. 1 –Data transfer at StartStp point is activated. But the data could only transfer from TRACE to SCF.
    c. 2 – Data now could transfer between TRACE and SCF at all synchronization points.
2) **aflag:** the flag is specially designed for transient of TRACE/SCF-ECI. It is controlled by TRACE and decides whether the data could be transferred.
    a. 0 – all data transfer is not permitted.
    b. 1 – initial value. Data transfer at all points could be performed.
3) **n_end:** this flag is controlled by TRACE but acts a special role in the SCF process controlling. It is the termination trigger specially used for the step to step calculation. Two integer values are possible.
    a. 0 – initial value. SCF now has not reached the end time of TRACE.
    b. 1 – SCF reaches the TRACE end time and terminates.

Two TRACE built-in flags also dominate in the data transfer process in addition to the three new flags. They are:

1) **allStdy:** this is the same flag as SS. However, it is used here not for the SS termination check but the transient. Especially, it is the termination trigger for SCF step skipped calculation while not for the step to step calculation. Two integer values are possible.
    a. 0 – initial value. SCF has not approached the end time of TRACE.
    b. 1- SCF is now quite close to the end time of TRACE and will terminate.
2) **normalEndFlag:** this is a TRACE build-in flag which acts as the termination trigger. It is the target for n_end and is effective for both the step to step calculation and the SCF step skipped calculation of TRACE. Two integer values are possible for this flag.
    a. 0 – initial value. TRACE has not reached its end time.
    b. 1 – TRACE reaches its end time and terminates.

During the calculation of TRACE/SCF-ECI transient, TRACE is in charge of all the flags. Additionally, the end time of TRACE must be the global end time for both codes, which indicates that the end time on SCF input should be defined to a quite huge value or larger than TRACE at least. The basic outline of this transient calculation is presented in Figure 4-21 and the illustration follows.

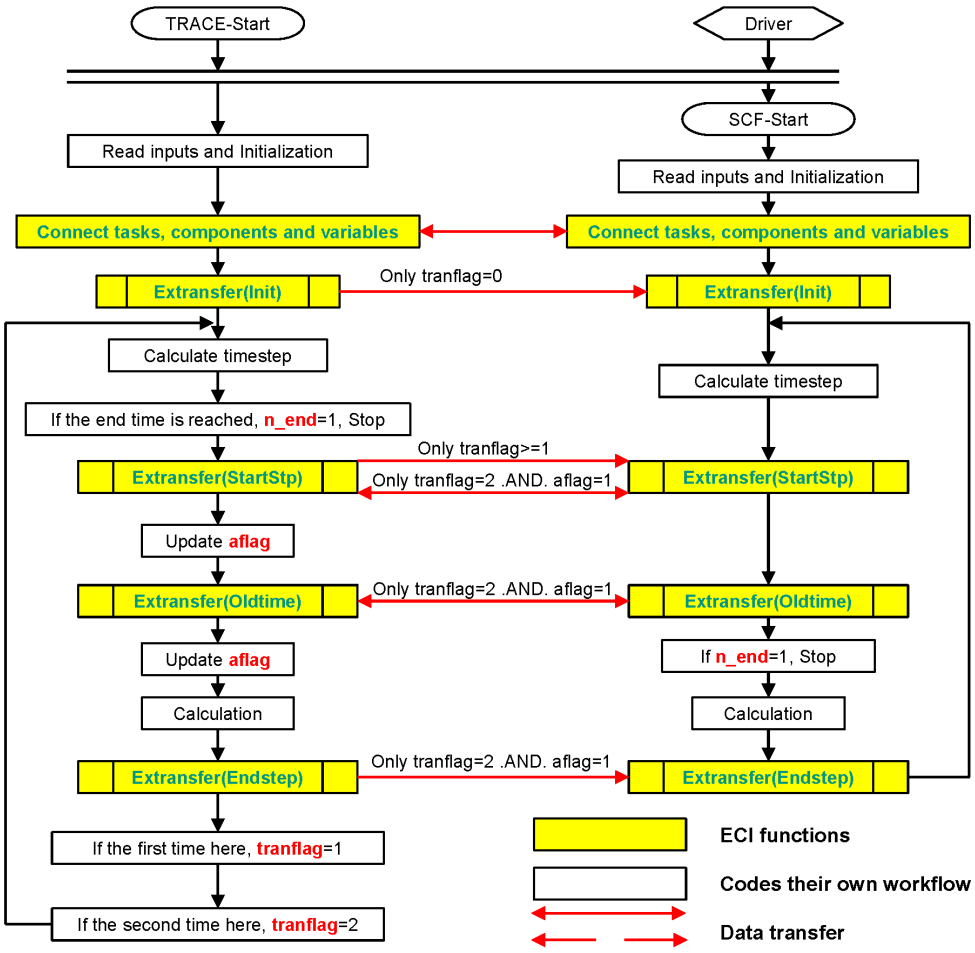


Figure 4-21 General Workflow of TRACE/SCF-ECI Transient

1) First, run the driver script from any location.
2) Run TRACE under the working directory. TRACE will read in the taskList and send SCF information to the driver which will then run SCF.
3) TRACE and SCF read their input files and do their initialization.
4) ECIs in TRACE and SCF will interact with each other and connect the two tasks, the components, and the variables sequentially.
5) Both the two codes reach the first data transfer synchronization point: Init. Now the transflag maintains its initial value – 0. So the Init data transfer could be performed. The data are all about the TRACE geometry. They are radial_n_f, radial_n_b and angular_n. The three variables and reasons for the transfer have already been well explained in the previous section.
6) Both of the two codes get into the time-step iteration.
    a. SCF will set its time-step to quite big a value to make a steady-state step. Then it will stop at the StartStp point since the socket in ECI is in blocking mode and there is no data available from TRACE for the time being. It will keep waiting until there are data available from TRACE.

b. TRACE will skip all the synchronization points since tranflag is 0. It will first try to run a transient step forward standalone. Then tranflag will turn to be 1. The value of aflag maintains 1 for the time being.

7) The first interactive running of TRACE and SCF.

a. TRACE go back to the loop head, flows down and transfer the SS boundary data (mass flow rate, coolant temperature, solute concentration, and pressure, shown in Figure 4-7) to SCF at the StartStp point since tranflag is 1 now. For the current step, no data is transferred from SCF, which is just because SCF has not run a step yet. TRACE then will skip other points and run to the loop end. Then tranflag will turn to 2. aflag maintains 1 for the time being.

b. SCF gets the boundary data from TRACE at the StartStp point and proceeds. The steady-state data will be obtained once SCF finished the first step (the SCF first step will always be a steady-state step).

8) The second step of the interactive running of TRACE and SCF.

a. Step to step calculation:

i. At the StartStp point, TRACE sends the boundary data from its last time-step to SCF and receives the boundary data from SCF. Then at the Oldtime point, TRACE first receives the current time-step size to SCF, compares with its own calculated current time-step size, determines the global time-step size and later on sends the global time-step size to SCF. With all the boundary conditions and time-step size fixed, TRACE will do one step calculation and sends some editing-related flags to SCF which includes: longEd, shortEd, graphEd and rstDump. aflag is always 1.

ii. At the StartStp point, SCF sends its SS step boundary data to TRACE and receives the corresponding data from TRACE. Then at the Oldtime point, SCF first sends its current calculated time-step size to TRACE and later on receives the current global time-step size from TRACE. With all the data fixed, SCF will do the current step calculation and then at the Endstep point, receive the editing-related flags from TRACE.

b. SCF step skipped calculation:

i. At the StartStp point, TRACE sends the boundary data form its last time-step to SCF and receives the boundary data from SCF. Then at the Oldtime point, TRACE first receives the global time-step size from SCF and stores this value to a special local variable. Then aflag will be set to 0 and the current TRACE time-step will be added to an accumulated local variable. TRACE will skip the send operation at Oldtime point. Then with all the boundary conditions and time-step size fixed, TRACE will do one step calculation forward. However, since aflag is 0 now, the following data transfers will be skipped.

ii. At the StartStp point, SCF sends its steady-state step boundary data to TRACE and receives the corresponding data from TRACE. Then at the Oldtime point, SCF first sends its current time-step size to TRACE and pause here and waits for the time-step size from TRACE.

9) From the third step to the antepenultimate step.

a. Step to step calculation:

i. At the StartStp point, TRACE sends the boundary data from its last time-step to SCF and receives the boundary data from SCF. Then at the Oldtime point, TRACE first receives the current time-step size to SCF, compares with its own calculated current time-step size, determines the global time-step size and later on sends the global time-step size to SCF. With all the boundary conditions and time-step size fixed, TRACE will do one step

calculation and sends some editing-related flags to SCF which includes: longEd, shortEd, graphEd and rstDump. aflag is always 1.

ii. At the StartStp point, SCF sends its SS step boundary data to TRACE and receives the corresponding data from TRACE. Then at the Oldtime point, SCF first sends its current calculated time-step size to TRACE and later on receives the current global time-step size from TRACE. With all the data fixed, SCF will do the current step calculation and then at the Endstep point, receive the editing-related flags from TRACE.

b. SCF step skipped calculation:

i. If aflag is 0, data transfer at the StartStp point will be skipped. If aflag is 1, data transfer at StartStp will be performed. Then at the Oldtime point, the receive operation will also be skipped. TRACE will add its current calculated time-step to the accumulated local variable and compare it with the previously received SCF time-step size. If the accumulated value is equal or larger than the SCF time-step size, then aflag will be set to 1, the accumulated value will be transferred to SCF as its current time-step size, then the accumulated will be reset to zero and the data transfer at Endstep point will be performed. Otherwise, aflag remains 0, the accumulated value keeps adding TRACE time-step size and all the following data transfer are blocked.

ii. If TRACE sends the data at Oldtime, SCF will receive the data as its current time-step size. Then SCF will run a transient step forward and receive the variables from TRACE at Endstep. Then, SCF will go back to the top of the next step and exchange boundary data with TRACE at StartStp. If there is still no data from TRACE at Oldtime, SCF will keep waiting.

10) At the penultimate step. This is not exactly the real penultimate step but is the step after which SCF will end its calculation.

a. Step to step calculation: are the same as the previous steps.

b. SCF step skipped calculation:

i. The flag aflag must be 1 for this step. The TRACE remaining problem time is less than the last SCF time-step size. So SCF will not run a further step. TRACE will set allStdy from 0 to 1 and pass to SCF. Then it will finish this step.

ii. SCF keeps waiting for receiving data from TRACE at the Oldtime point until it gets allStdy from TRACE. Since allStdy is 1, SCF will immediately jump out of the iteration, do the post-processing and terminate.

11) The remaining time-steps.

a. Step to step calculation: are the same as the previous steps.

b. SCF step skipped calculation: since SCF is already over, TRACE will run standalone for these steps.

12) TRACE and SCF do their post-processing for step to step calculation and for SCF step skipped calculation, only TRACE will do the post-processing since SCF has already terminated the calculation.

Just like the steady-state calculation, the transient illustration above actually contains more details and doesn't agree with the figure. The reason is that SCF sometimes could work with more than one TRACE and may have different data transfer with different TRACE. Now, one may have the idea that the step skipped mode is more efficient than the step to step mode. However, two questions may arise here. One is how to determine the skipped steps number. The other one is how to keep the synchronization between TRACE and SCF. The questions could be well explained by the instance presented in Figure 4-22. A new flag was designed to assist the step

skipped function: aflag. The function of this flag is similar to rflag in steady-state. It also has two values. The operations which set its value are to be explained following the figure.

1) aflag=1: SCF is activated and Subchan_run will be called.
2) aflag=0: SCF is inactivated and Subchan_run will be skipped.

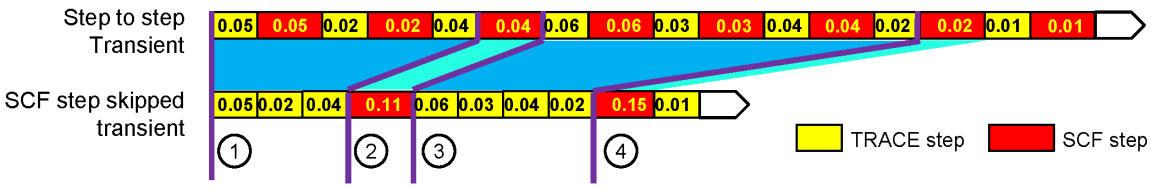


Figure 4-22 Synchronization between Step-to-Step Mode and SCF Step Skipped Mode of TRACE/SCF-ECI

1) At point 1, the calculated time-step of TRACE is 0.05s and the SCF's is 0.1s.
   a. Step to step: Subchan_time will select 0.05s as the global time-step. TRACE and SCF both run a 0.05s forward transient step.
   b. Step skipped: Sunchan_time adds 0.05s to its local time-accumulator whose initial value is 0 and finds that the time-accumulator is smaller than 0.1s. Then it will return 0.05s to TRACE as its time-step and set aflag to 0.
2) From point 1 to point 2.
   a. Step to step: TRACE and SCF run one by one with the same time-step.
   b. Step skipped: Subchan_time adds the passed-in time-step to the time-accumulator and finds it is still smaller than 0.1s. Then it returns the current TRACE time-step as it was passed in back to TRACE, and keeps aflag 0.
3) At point 2.
   a. Step to step: normal.
   b. Step skipped: Subchan_time adds the passed-in time-step to the time-accumulator and finds 0.11s is greater than 0.1s. Then it returns the TRACE passed-in 0.04s to TRACE, turns aflag to 1 and set 0.11s as the SCF's current time-step. At last, zero the time-accumulator.
4) From point 2 to point 3.
   a. Step to step: normal.
   b. Step skipped: since aflag is 1, Subchan_run will be called by TRACE with a time-step of 0.11s.
5) At point 3, the new calculated time-step of TRACE is 0.06s and the SCF's is 0.15s.
   a. Step to step: Subchan_time will select 0.06s as the global time-step. TRACE and SCF both run a 0.06s forward transient step.
   b. Step skipped: Sunchan_time adds 0.06s to its local time-accumulator whose value is temporarily 0 now and finds that the time-accumulator is smaller than 0.15s. Then it will return 0.06s to TRACE as its time-step and reset aflag to 0.
6) From point 3 to point 4.
   a. Step to step: TRACE and SCF run one by one with the same time-step.
   b. Step skipped: Subchan_time adds the passed-in time-step to the time-accumulator and finds it is still smaller than 0.15s. Then it returns the current TRACE time-step as it was passed in back to TRACE, and keeps aflag 0.
7) At point 4.

a. Step to step: normal.
b. Step skipped: Subchan_time adds the passed-in time-step to the time-accumulator and finds 0.15s is equal to the pre-calculated 0.15s. Then it returns the TRACE passed-in 0.02s to TRACE, turns aflag to 1 and set 0.15s as the SCF's current time-step. At last, zero the time-accumulator.

8) From point 4.
a. Step to step: normal.
b. Step skipped: since aflag is 1, Subchan_run will be called by TRACE with a time-step of 0.15s.

From Figure 4-22 and the associated illustrations, it can be concluded that the skipped SCF steps number is not predetermined and depends on the two codes' real-time calculated time-steps. This is a quite flexible strategy. What's more, the step to step coupling might be desired sometimes. So, not only the SCF step skipped mode, but also the step to step mode are available in this work. A newly designed input parameter ***j_skip*** should be defined in tracin to figure out which mode is desired (call back to section 2.4).

1) j_skip=1, the SCF step skipped mode is applied.
2) j_skip=0, the step to step coupling mode is applied.

Now, let's explore the SCF step skipped function at a deeper level. What has already been well explained in this section is almost all about the time synchronization method which aims to assure TRACE and SCF keep the same problem time. However, this is not the full image of synchronization which should also include so-called data synchronization. Figure 4-23, which is the left half of Figure 4-22 is presented for a better illustration of this concept.

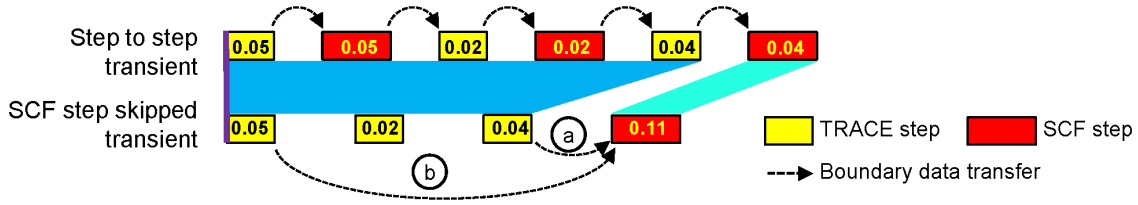


Figure 4-23 Data Synchronization of SCF Step Skipped Function for TRACE/SCF-ECI Transient

For the step to step calculation, boundary data is passed between TRACE and SCF one by one and the logic is clear. For instance, the first SCF step uses boundary data from the first TARCE step, the second TRACE step use data from its adjacent previous SCF step, the second SCF step use data from its adjacent previous TRACE step, following steps are the same. When it comes to the SCF step skipped calculation, conditions are a little bit complicated. There are two possible ways to transfer the boundary data:

1) a: SCF gets the boundary data from its adjacent previous TRACE step. In Figure 4-23, it means the 0.11s SCF step use data from the 0.04s TRACE step.
2) b: SCF gets the boundary data from the first TRACE step in the TRACE steps block. In Figure 4-23, it means the 0.11s SCF step use data from the 0.05s TRACE step.

Option b is the right way while option a is in practice the typical wrong way for boundary data transfer. The 0.11s SCF step should get the TRACE data just like the 0.05s SCF step in the step to step calculation, which implies getting data at the right time. So, the data at problem time 0.05s should be used not only for the step to step 0.05s SCF step but also for the step skipped 0.11s SCF step. As to option a the TRACE data from 0.04s step is 0.06s delayed than the right current data for SCF, which makes the system out of synchronization.

5 VERIFICATION OF THE COUPLED CODE

For the testing of the coupled system TRACE/SCF-ECI, the problem shown in Figure 4-4 was analyzed. TRACE represents an RPV of a PWR with four inlet and outlet boundary conditions for the flow and temperature of the primary circuit, where the core is represented by four sectors (3D volumes). SCF represents the core by nine square meshes, Figure 5-1.

Two coolant mixing cases were defined to be simulated with the coupled code to demonstrate that the improved description of the coolant mixing when the core is simulated by SCF instead of TRACE.

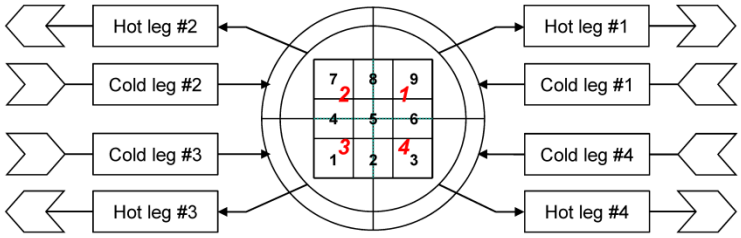


Figure 5-1 Four Loops and Nine Channels of TRACE/SCF

5.1 Coolant Mixing Problem

The problem to be solved consists of a transient where the coolant temperature of the cold leg 1 is increased from 400 K to 500 K within 20 seconds and it remains at that temperature for 20 seconds while the coolant temperature of the other cold legs is kept constant at 400 K, see Table 5-1. The solute boron concentration is 0.01 expressed in a mass ratio (the solute mass to water mass ratio, around 9900 ppm).

Table 5-1 Boundary Conditions for the Coolant Mixing Problem

	Loop #1 Inlet			*Loop #2 #3 #4 Inlet*
Time (t)	0s ~ 20s	20s ~ 30s	30s ~ 50s	0s ~ 50s
Temperature	400K	400 + (t-20)*10K	500K	400K
Velocity	10m/s			10m/s
Solute mass ratio	0.01			0.01

The transient lasts for 50s. The coolant inlet velocity is fixed to 10m/s and the solute ratio was fixed to 0.01 for all of the four loops. Under such conditions, a coolant mixing will take place in the downcomer and the resulting mixing pattern will be propagated through the core. Two simulations, one with TRACE and the other one with TRACE/SCF-ECI, are performed to analyze this test problem.

The coolant temperatures of the hot legs predicted by TRACE stand-alone and by TRACE/SCF-ECI are shown for the duration of the transient in Figure 5-2 and Figure 5-3. There it can be observed, that TRACE predicts a lower temperature increase of in the loops 2 and 4 of around 9 to 10 K than the one predicted by TRACE/SCF which amounts around 18 K. As a result of it, the

heat up of hot leg #1 predicted by TRACE is also higher than the one predicted by the coupled code (74 K > 50 K). The main reason for it is the fact that TRACE/SCF predicts a stronger coolant mixing along the core between the hot meshes and the neighbor meshes linked to sector 2 and 4 than TRACE does. One of the reasons is that the lateral flow model in SCF is better than that of TRACE. The other reason is that the SCF-model is more detailed (12 mesh boundaries for cross-flow) compared to the very coarse four sector model of TRACE. Temperature increases at loop 2 and 4 are more or less the same because they are symmetrically located concerning loop 1. Moreover, the temperature at loop 3 was not that much affected because it has no direct mass exchange interfaces with loop 1.

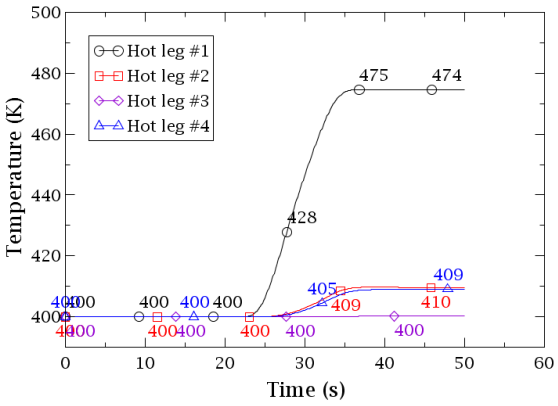


Figure 5-2 Temperature at Four Hot Legs by TRACE Standalone

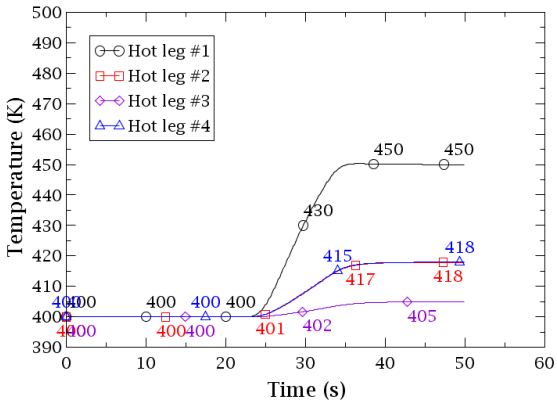


Figure 5-3 Temperature at Four Hot Legs by TRACE/SCF-ECI

5.2 Boron Concentration Problem

The second test problem consists of the variation of the boron concentration of the loop-1 as is indicated in Table 5-2. Otherwise, the problem is the same as the one described under Subchapter 6.1. The coolant solute mass ratio of loop-1 is changed linearly from 0.01 to 0.05 and this value is maintained until the end of the transient.

Table 5-2 Boundary Conditions for the Concentration Problem

	Loop #1 Inlet			*Loop #2 #3 #4 Inlet*
Time (t)	0s ~ 20s	20s ~ 30s	30s ~ 50s	0s ~ 50s
Temperature	400K			400K
Velocity	10m/s			10m/s
Solute mass ratio	0.01	0.01 + (t-20)*0.04	0.05	0.01

Again, this problem was solved with both TRACE and TRACE/SCF. In Figure 5-4, the boron concentration predicted by TRACE standalone at the hot legs is plotted. In Figure 5-5, the solute mass ratio at the four hot legs predicted by the coupled code TRACE/SCF is shown. A review of the results shows that TRACE predicts a higher value of the solute mass ratio, Figure 5-4, of leg-1 compared to the value predicted by the coupled code TRACE/SCF for the same hot leg, Figure

5-5. Due to the stronger mixing within the core predicted by SCF, an increase of the solute mass ration in the hot legs 2 and 3 are predicted only by TRACE/SCF.

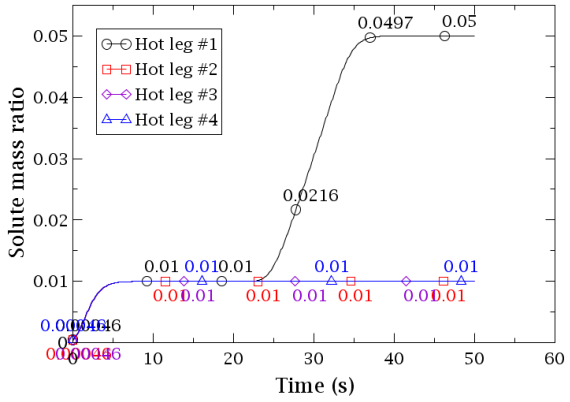


Figure 5-4 Solute Mass Ratio at Four hot Legs by TRACE Standalone

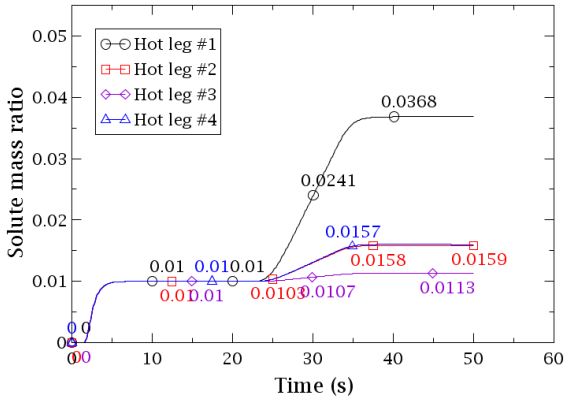


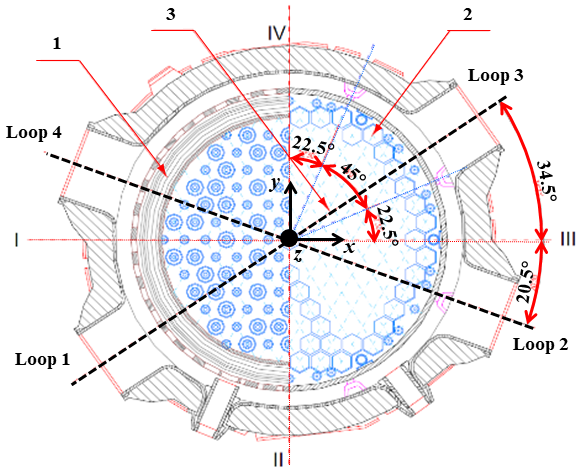
Figure 5-5 Solute Mass Ratio at Four Hot Legs by TRACE/SCF-ECI

6 VALIDATION OF THE COUPLED CODE

The validation of the new TRACE/SCF capability is done by the analysis of the VVER-1000 coolant mixing benchmark problem. Selected parameters predicted by the coupled code are compared with TRACE standalone and measured data at the Kozloduy nuclear power plant.

6.1 Description of the VVER-1000 Coolant Mixing Experiment

The coolant mixing experiment was performed during the commissioning phase of Kozloduy NPP to study the mixing of loop flows in the reactor vessel of VVER-1000 V320 (Kolev, et al. 2010). In the test, the steam isolation valve was closed and the steam generator was isolated resulting in the heat-up of primary coolant in the sector of the downcomer linked to the affected steam generator (first loop). Due to the coolant mixing in the downcomer, the temperature of the other primary loop is partly also affected. The mixing pattern established in the downcomer propagates to the core where also coolant mixing takes place. VVER-1000 is a four-loop pressurized water reactor with hexagonal core geometry and horizontal steam generators. The core is open-type and contains 163 hexagonal fuel assemblies. The location of the main inlet and outlet nozzles of the reactor vessel is non-uniform in the azimuthal direction and asymmetric concerning the core symmetry axes. The cross-section sketch of the reactor vessel is described in Figure 6-1.



1 – Block of Shielding Tubes (BST) chimney
2 – Fuel assemblies
3 – Lower spacer grid
LEFT – Cut at hot legs
RIGHT – Cut at cold legs

Figure 6-1 Vessel Cross-Sectional Sketch of VVER-1000 Including the Vessel Inlet and Outlet

The main initial operating parameters before the test are summarized in Table 6-1.

The transient test consists of the following events:

1) Closure of the steam isolation valve and isolation of the steam generator from feed water of loop 1;

2) Coolant temperature at the cold leg of loop 1 increases by about 14 degrees, Figure 6-2;
3) Coolant mixing occurs first in the downcomer;
4) Coolant mixing takes place in the lower plenum, core and upper plenum;

Coolant temperature at the other three loops all increase by different degrees, see Figure 6-2 and Figure 6-3.

Table 6-1 Main Initial Operating Parameters Before Test

Parameter	Initial State	Accuracy
Thermal power, MW	281	± 60
Pressure above the core, MPa	15.593	± 0.3
Pressure drop over RPV, MPa	0.418	± 0.043
Coolant temperature at core inlet #1, K	541.75	± 1.5
Coolant temperature at core inlet #2, K	541.85	± 1.5
Coolant temperature at core inlet #3, K	541.75	± 1.5
Coolant temperature at core inlet #4, K	541.75	± 1.5
Coolant temperature at core outlet #1, K	545	± 2.0
Coolant temperature at core outlet #2, K	545	± 2.0
Coolant temperature at core outlet #3, K	544.9	± 2.0
Coolant temperature at core outlet #4, K	545	± 2.0
Mass flow rate of loop #1, kg/s	4737	± 110
Mass flow rate of loop #2, kg/s	4718	± 110
Mass flow rate of loop #3, kg/s	4682	± 110
Mass flow rate of loop #4, kg/s	4834	± 110

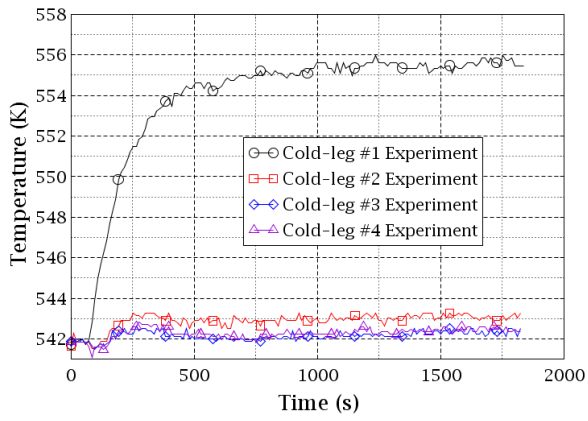


Figure 6-2 Measured Evolution of the Coolant Temperature at Cold Legs During the Transient of the VVER-1000 Coolant Mixing Benchmark

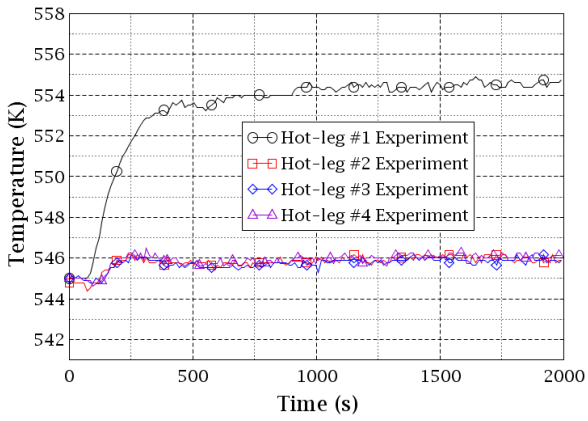


Figure 6-3 Measured Evolution of the Coolant Temperature at Hot Legs During the Transient of the VVER-1000 Coolant Mixing Benchmark

The test lasts for 1800 seconds and the final core power increased up to 286 MW. Due to the isolation of the steam generator 1, the heat transfer from the primary to the secondary side

decreases leading to the increase of the coolant temperature on the cold-leg-1 and finally on the hot-leg-1. The coolant temperature of the other three loops also increases due to the coolant mixing in the downcomer and lower plenum. Reverse heat transfer occurs in the isolated steam generator #1. At about 300s, the temperature of cold-leg-1 exceeds that of hot-leg-1, see Figure 6-2 and Figure 6-3. The difference stabilizes to 0.6~0.8 degrees in about 20 min. Cold-leg-2 has a slightly higher coolant temperature than cold-leg-3 and 4 during the transient, while the outflows on the four hot-legs almost have the same temperature, indicating the coolant well mix.

6.2 Description of the Thermal-Hydraulic Models of the RPV and Core

TRACE/SCF employs a domain decomposition approach in which SCF simulates the reactor core while TRACE simulates the rest of the vessel. For the simulation of TRACE/SCF, modifications e.g., blockage of the core and addition of FILL and BREAK components at the core inlet and outlet planes which represent the interface planes of the coupled code should be performed to the original TRACE model, Figure 6-4. The SCF model has 163 hexagonal cells demonstrating the fuel assemblies and a hexagonal core geometry, Figure 6-5. No further modification to the original model is required.

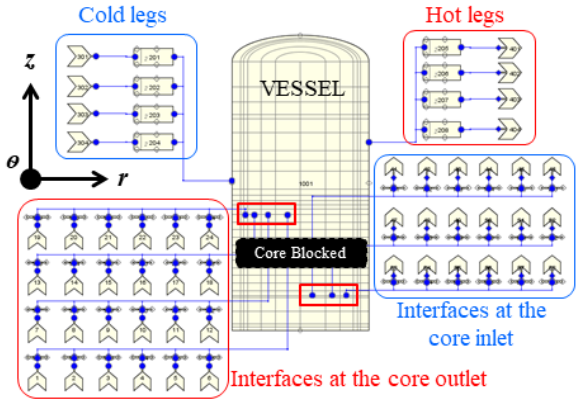


Figure 6-4 TRACE Model for the VVER-1000 Coolant Mixing Benchmark

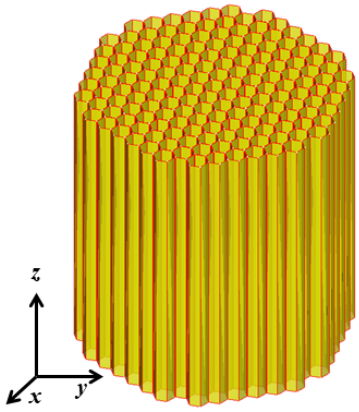


Figure 6-5 SCF Model for the VVER-1000 Coolant Mixing Benchmark

To map the mesh data of both codes consistently, an overlapping-area-weighted algorithm and a toolkit have been used (see Appendix H). The data mapping is illustrated in Figure 6-6, where TRACE extracts the pressure at its core outlet and transfer it to SCF after a data mapping process. Meanwhile, it extracts the mass flow rate and coolant temperature at the core inlet and transfers to SCF. The data from SCF to TRACE occurs oppositely.

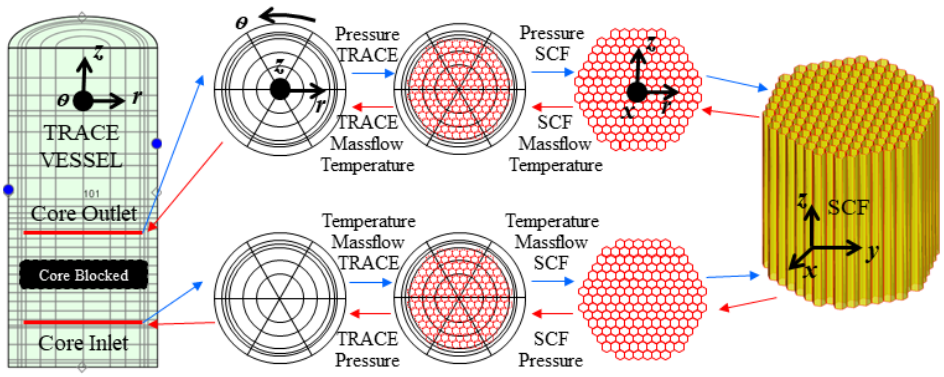
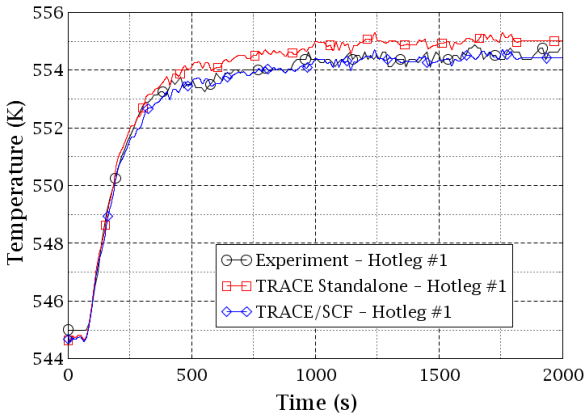


Figure 6-6 Data Mapping Between TRACE and SCF for the VVER-1000 Coolant Mixing Benchmark

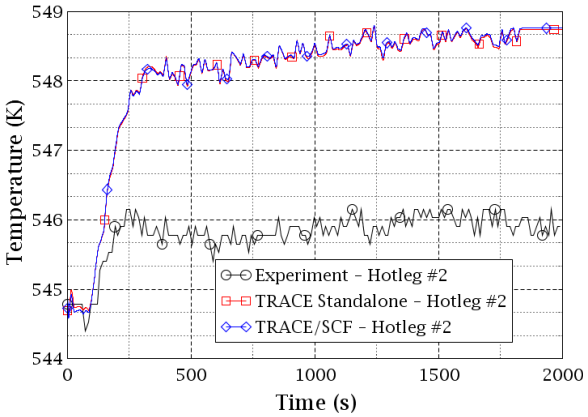
6.3 Discussion of Selected Results

The coolant mixing test is analyzed with both TRACE standalone and the coupled TRACE/SCF. The codes predict the coolant mixing in the downcomer, the lower plenum, and the propagation of this mixing pattern through the core as well as the mixing in the upper plenum. The time-dependent boundary conditions e.g., coolant temperature at the cold legs and outlet pressure at the hot legs are taken directly from the benchmark specifications.

The coolant temperature at the four hot legs predicted by TRACE and TRACE/SCF is compared to the measured data in Figure 6-7. The temperature rise of hot-leg-1 can be observed in Figure 6-7a, where TRACE standalone tends to over-predict the heat-up while the coolant temperature calculated by TRACE/SCF is very close to the plant data. In Figure 6-7b and Figure 6-7c, the predictions of TRACE and TRACE/SCF are all not satisfactory. They over-predict the heat-up at hot-leg-2 while the coolant temperatures at hot-leg-3 are under-predicted. In Figure 6-7d, TRACE standalone under-predicts the heat-up on hot-leg-4 while TRACE/SCF gives a closer result to the measured data.



a. hot leg 1



b. hot leg 2

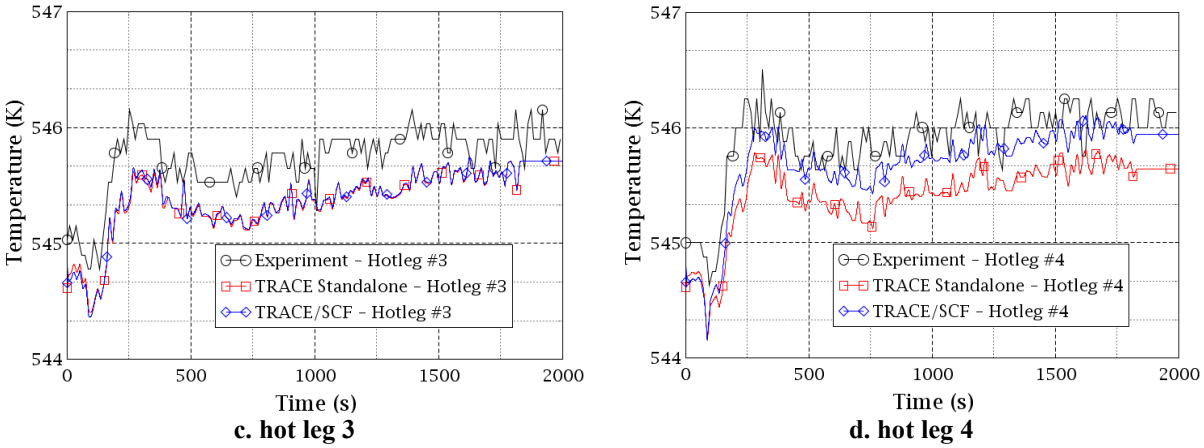


Figure 6-7 Temporal Evolution of the Measured and Computed (by TRACE Standalone and TRACE/SCF) Coolant Temperature at the Hot Legs of the VVER-1000 Coolant Mixing Benchmark, Which Show TRACE/SCF Could Produce a Closer Result to the Measured Data than that by TRACE Standalone

Before digging any further into the simulated data, it is worth noting that the cold and hot legs in the RPV model of TRACE are located at sector #4, #6, #1, and #3, respectively, see Figure 6-8. Also, the loops are not arranged symmetrically and there is an anti-clockwise angle shift concerning the core symmetry axes or the loops in the TRACE model, also see Figure 6-8. TRACE could hardly represent this kind of irregular geometry unless the RPV model is split into a larger number of sectors in the azimuthal direction. But this will lead to the poor numerical performance of any code with coarse 3D thermal-hydraulic cells. Here, CFD-codes are more suitable to treat such kind of non-symmetrical problems than system codes.

It is important to pay attention to the numbering of the sectors in the TRACE vessel model and loops configuration. The two numbering sets are in more or less an opposite way which is not reader-friendly. However, the numbering of the sectors strictly follows the TRACE modeling rule for the vessel and the numbering of the loops strictly follows the real VVER-1000 geometric configuration. Moreover, there are no loops in the TRACE vessel model while there are only sectors. The sectors are the basic computation unit for the TRACE vessel and the mass and heat transfer inspection in the core.

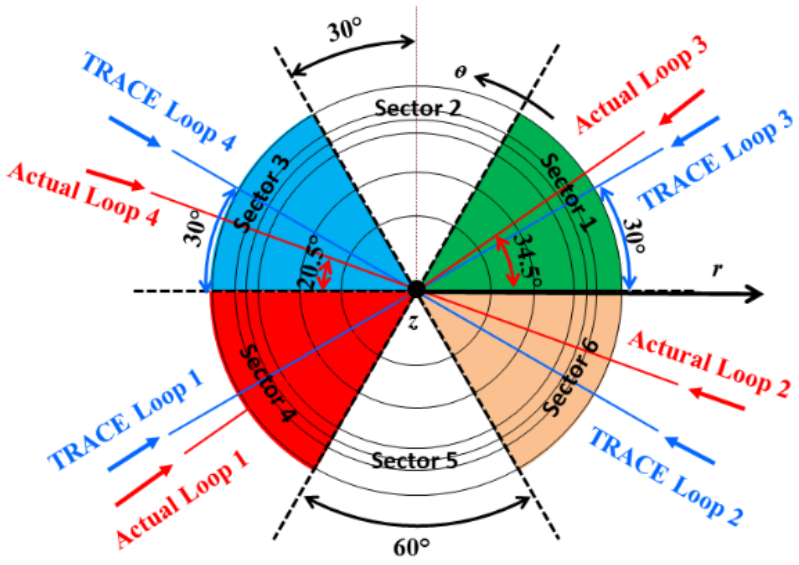


Figure 6-8 Geometric Mismatch of Modeled TRACE RPV and Real Existing Set-up

Thanks to the simulation of the core with a sub-channel code in the coupled system TRACE/SCF, the predicted coolant temperatures at hot-leg-1 and 4 tends to be improved compared to TRACE standalone but not for the hot-leg-2 and 3. Figure 6-9a presents the coolant temperature distribution at the core inlet by TRACE standalone. The data are derived from the six azimuthal sectors at the second ring of the TRACE core inlet plane (see Figure 6-8 for reference). The coolant temperature at sector 1 (correspond to loop 3), 2 and 3 (correspond to loop 4) are almost the same lowest value while sector 6 (correspond to loop 2) keeps around 2 degrees higher temperature than the first three sectors during the transient. Combined with the high-temperature curve at sector 5, which is located between loop 1 and loop 2, it can be concluded that the most significant coolant mixing takes place between sector 4 and sector 6 through sector 5. This phenomenon is the most probable reason for the over-predicted coolant temperature at hot leg-2 by TRACE, see Figure 6-7b. Figure 6-9b presents the coolant temperature distribution at the core outlet by TRACE standalone. Compared with Figure 6-9a, it can be observed that the temperature rises at the six sectors are almost the same, by about 3 degrees, indicating that there is little mixing predicted by TRACE when the coolant flows up through the core.

The coolant temperature distribution at the core inlet by TRACE/SCF is identical with that by TRACE standalone, see Figure 6-9c. Whereas the data at the core outlet show difference when comparing Figure 6-9d with Figure 6-9b. There is no data change at sector 1 (loop 1), 2, 5 and 6 (loop 2) between the two figures, meaning there is no remarkable coolant mixing enhancement simulated by SCF than TRACE at these four sectors. Nevertheless, a slight temperature increase and decrease are observed in sector 4 (loop 1) and sector 3 (loop 4) respectively, indicating the coolant mixing is strengthened between the two sectors. This effect is believed to be the reason why TRACE/SCF predicts better results at hot-leg-1 and 4 than TRACE standalone, see Figure 6-7a, and Figure 6-7d.

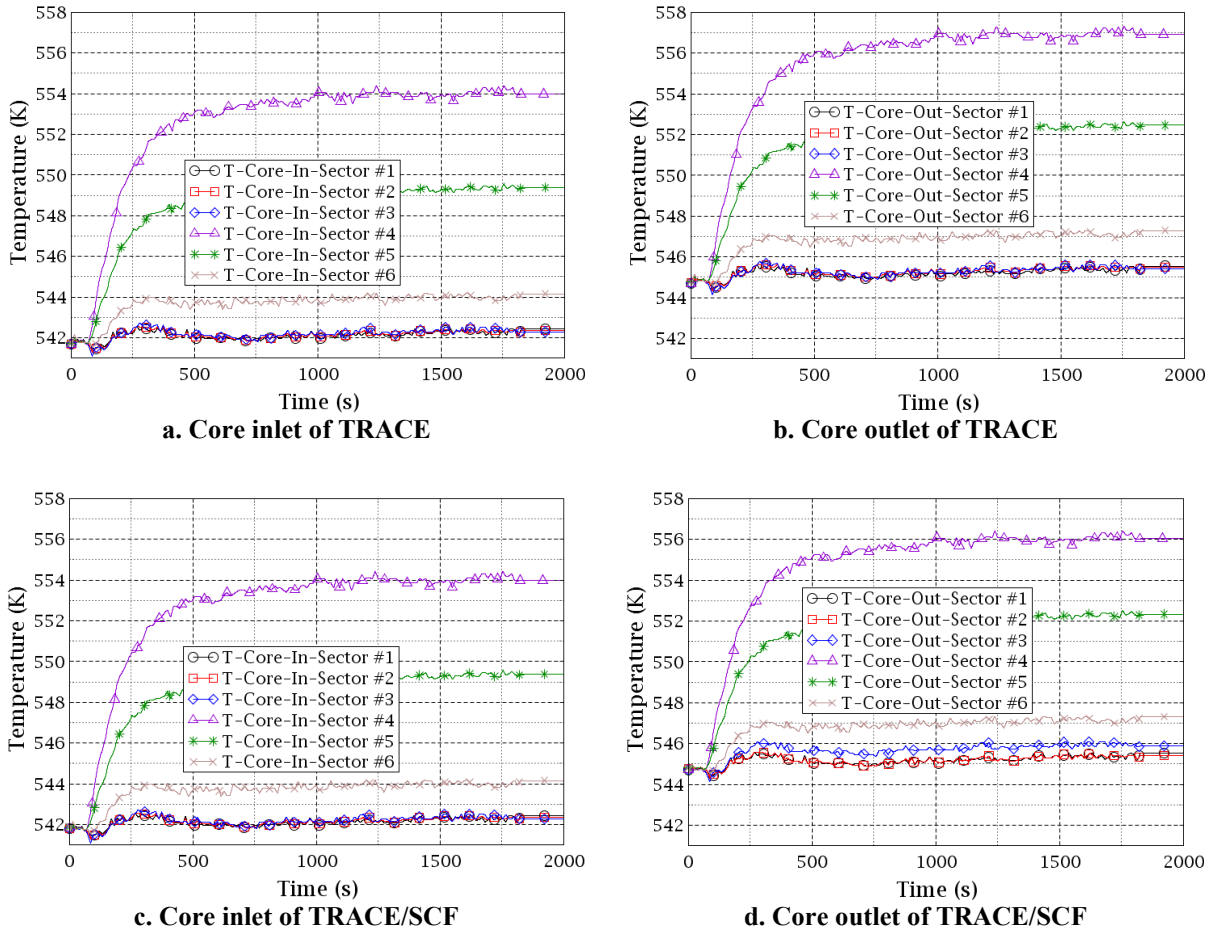


Figure 6-9 Temporal Evolution of the Computed (by TRACE and TRACE/SCF) Coolant Temperature at the Core Inlet and Outlet (on TRACE mesh) of the VVER-1000 Coolant Mixing Benchmark, Which Shows TRACE/SCF Could Predict Stronger Coolant Mixing Between Sector 4 and 3 Than That by TRACE

Figure 6-10 shows the azimuthal mass flowrate curves at the downcomer inlet, where the absolute mass flowrate at the interfaces between sector 1 and 2 (face 1&2), face 2&3, face 4&5, face 5&6 are as large as 800 kg/s while that at face 3&4 and face 6&1 are quite small. The positive direction of the flow at each face is right the same with the normal vector on the anti-clockwise direction (refer to the coordination shown in Figure 6-8). A positive value means an anti-clockwise flow while a negative value means a clockwise flow. The data indicate that the coolant in sector 2 comes mainly from sectors 1 and 3 and the coolant in sector 5 comes mainly from sectors 4 and 6. The curves also illustrate that the coolant mixing effect between sector 4 (first loop) and sector 3 (fourth loop), between sector 6 (loop 2) and sector (loop 3) is relatively weak.

Figure 6-11 plots the mass flowrate through the six azimuthal interfaces along the downcomer in the reactor pressure vessel. The mass flowrate from sector 4 (loop 1) and sector 6 (loop 2) to sector 5 decrease from 800 kg/s at the downcomer inlet to 0 kg/s at the RPV height of around 3 m. After that, the flow reverses from sector 5 to sector 4 and sector 6. The conditions are the same for sectors 1, 2 and 3. Note the mass flow rate through face 3&4 and face 6&1 keeps close

to 0 kg/s along the downcomer. Figure 6-11 tells the fact that there are two somehow isolated mixing regions within the downcomer where sector 1 to 3 form a mixing body while sector 4 to 6 form the other one. It also shed light on the remarkable coolant temperature rise at hot-leg-2 compared that at hot-leg-3 and 4.

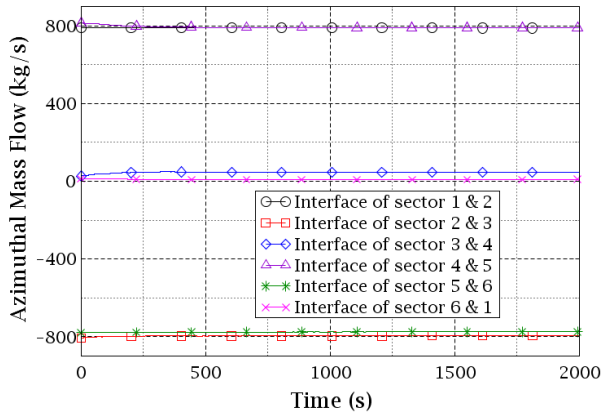


Figure 6-10 The Computed Azimuthal Mass Flowrate Over the Six Azimuthal Faces at the Downcomer Inlet for the VVER-1000 Coolant Mixing Benchmark

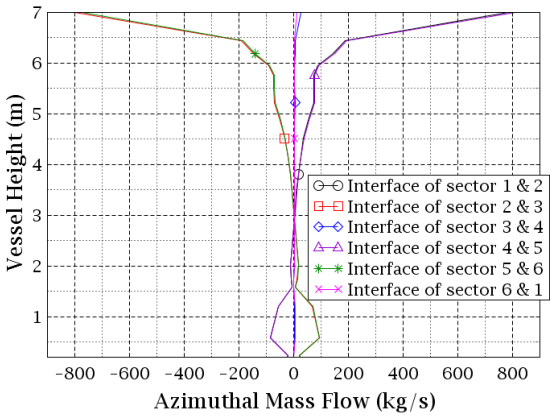


Figure 6-11 The Computed Azimuthal Mass Flowrate Over the Six Azimuthal Faces Along the Downcomer for the VVER-1000 Coolant Mixing Benchmark

Figure 6-12 and Figure 6-13 present the coolant temperature distribution at the core outlet assembly-wise predicted TRACE/SCF i.e., by SCF and measured in the experiment respectively. There, one can observe that the location of the four loops is shifted anti-clockwise compared to their positions in the TRACE model. A significant temperature change can be seen between sector 3 and sector 4 (refer to the model configuration in Figure 6-8) in Figure 6-12 confirming a relatively weak coolant mixing between the two sectors. Moreover, the rapid temperature change could be extended to include the entire horizontal line above which the coolant is cooler (sector 1, 2, and 3) while below which the coolant is hotter (sector 4, 5, and 6). In the experimental data shown in Figure 6-13, a more smooth temperature transition area is seen between sectors 3 and 4. Furthermore, the measured temperature is more symmetrically distributed than the ones of the simulations.

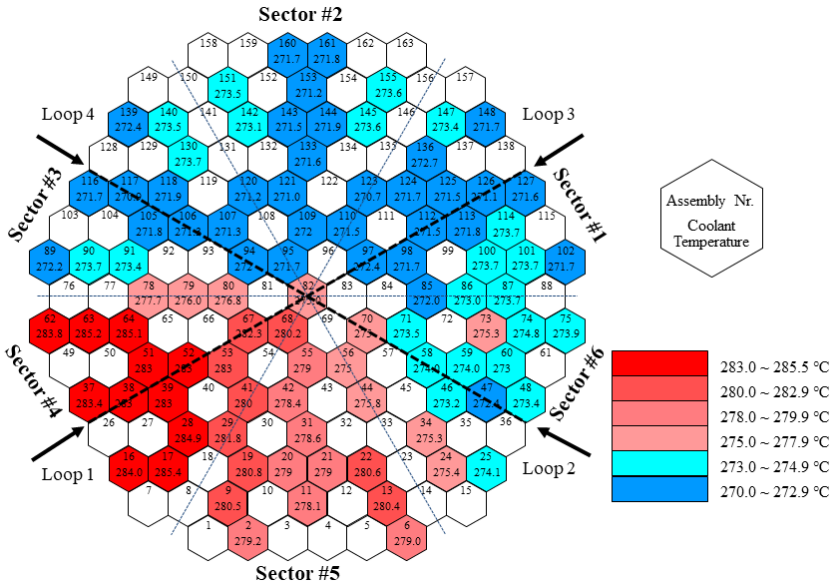


Figure 6-12 Coolant Temperature Distribution at the Core Outlet by TRACE/SCF for the VVER-1000 Coolant Mixing Benchmark Showing the Hottest Coolant in Sector 4

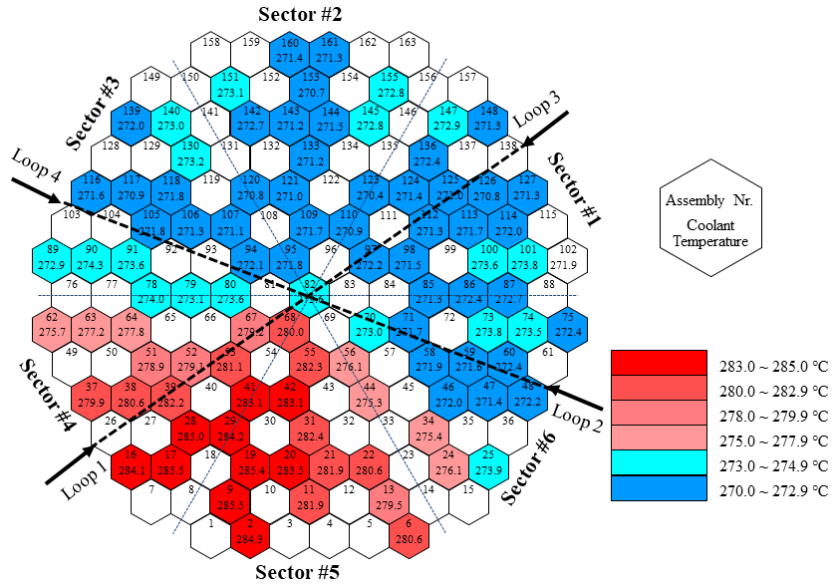


Figure 6-13 Measured Coolant Temperature Distribution at the Core Outlet for the VVER-1000 Coolant Mixing Benchmark Showing the Hottest Coolant between Sector 4 and Sector 5

It could be concluded from this section that SCF in the coupled codes improves the thermal-hydraulic prediction in the core region, while better simulations of the thermal-hydraulic behavior e.g., the coolant mixing in the reactor pressure vessel of this VVER-1000 RPV can only be expected if the non-symmetrical location is correctly described by a model.

CONCLUSION

This report presents the coupling of TRACE using the newly developed ECI-module with SUBCHANFLOW (SCF) to improve the prediction of the thermal-hydraulic behavior in reactors cores. The peculiarities of ECI and the multi-task capabilities of TRACE were explained for a better understanding of the coupling implementation. A new ECI-module was developed for SCF which can automatically establish a link to the TRACE-components in a multi-task system. A domain decomposition approach was adopted and a new toolkit for the radial field mapping was developed to ensure a consistent data transfer between the computational domains. Key-features of the newly-developed toolkit is the unique techniques for flexible-assignment and auto-recognition of interface-components for data transfer in TRACE. The Operator Splitting (OS) coupling approach (temporal coupling) was implemented for data synchronization during the time advancement. Two academic test cases and one benchmark case were defined to demonstrate the improved description of the core thermal-hydraulics thanks to the use of a sub-channel code instead of a system code with 3D coarse meshes. The obtained results are very promising and indicate that the codes were properly coupled to each other.

OUTLOOK

The next steps are the selection of more experimental data performed under realistic conditions to validate the prediction capability of the new-coupled code and in this connection to further optimize the coupling regarding accuracy, usability, and robustness. Moreover, other coupling approaches for system codes with sub-channel and CFD-codes such as the ICoCo-method developed during the NURESAFE project (Chanaron, et al. 2015) will be explored.

REFERENCES

Anderhuber, M., A. Gerschenfeld, N. Alpy, J. Perez, and JM. Seiler. 2015. "Simulation of GR19 Sodium Boiling Experiments with CATHARE 2 System Code and TRIO_U MC Subchannel Code." *NURETH-16, 16th International Topical Meeting on Nuclear Reactor Thermal Hydraulics.* Chicago, U.S.

Aragonés, J. M., C. Ahnert, O. Cabellos, and N. García-Herranz. 2004. "Methods and Results for the MSLB NEA Benchmark Using SIMTRAN and RELAP-5." *Nuclear Technology*, 146: 29-40.

Aumiller, D.L., E.T. Tomlinson, and R.C. Bauer. 2001. "A coupled RELAP5-3D:CFD methodology with a proof-of-principle calculation." *Nuclear Engineering and Design* 205 (1-2): 83–90.

Basile, D., R. Chierici, M. Beghi, E. Salina, and E. Brega. 1999. "COBRA-EN, an Updated Version of the COBRA-3C/MIT Code for Thermal-Hydraulic Transient Analysis of Light Water Reactor Fuel Assemblies and Cores, Report 1010/1."

Bavière, R., N. Tauveron, F. Perdu, and E. Garré. 2013. "System-CFD Coupled Simulation of the PHENIX Reactor Natural Circulation Test." *NURETH-15 The 15th International Topical Meeting on Nuclear Reactor Thermal - Hydraulics.* Pisa, Italy.

Bestion, D. 2014. "The difficult challenge of a two-phase CFD modelling for all flow regimes." *Nuclear Engineering and Design* 279: 116–125.

Bestion, D., D. Lucas, H. Anglart, B. Niceno, and L Vyskocil. 2012. "Multi-Scale Thermalhydraulic Analyses Performed in NURESIM and NURISP Projects." *20th International Conference on Nuclear Engineering and the ASME 2012 Power Conference.* Anaheim, California, U.S.

Calvin, C., and D. Nowak. 2010. *Handbook of Nuclear Engineering: 12 High-Performance Computing in Nuclear Engineering.* Springer.

Chanaron, B., C. Ahnert, N. Crouzet, V. Sanchez, N. Kolev, O. Marchand, S. Kliem, and A. Papukchiev. 2015. "Advanced multi-physics simulation for reactor safety in the framework of the NURESAFE project." *Annals of Nuclear Energy*, 166-177.

Chavez, V. J., U. Imke, and V.H. Sanchez-Espinoza. 2018. "TWOPORFLOW: A two-phase flow porous media code, main features and validation with BWR-relevant bundle experiments." *Nuclear Engineering and Design* 338: 181-188.

Chelemer, H., J. Weisman, and L.S. Tong. 1972. "Sub-channel thermal analysis of rod bundle cores." *Nuclear Engineering and Design* 21 (1): 35–45.

Cheng, Z., and Y.F. Rao. 2015. "Strategies for Developing Subchannel Capability in an Advanced System Thermal hydraulic Code: A Literature Review." *AECL Nuclear Review* 4 (1): 23-41.

D'Auria, F., A. Bousbia Salah, G.M. Galassi, J. Vedovi, F. Reventós, A. Cuadra, J.L. Gago, et al. 2004. *Neutronics/Thermal-hydraulics Coupling in LWR Technology, State-of-the-art*

Report. Nuclear Energy Agency, Organisation for Economic Co-Operation and Development.

D'Auria, F.; Galassi, G.M. 2010. "Scaling in nuclear reactor system thermal-hydraulics." *Nuclear Engineering and Design* 240 (10): 3267–3293.

Emonot, P., A. Souyri, J.L. Gandrille, and F. Barre. 2011. "CATHARE-3: a new system code for thermal-hydraulics in the context of the NEPTUNE project." *Nuclear Engineering and Design* 241 (11): 4476-4481.

Guelfi, A., D. Bestion, M. Boucker, P. Boudier, P. Fillion, M. Grandotto, J. Hérard, E. Hervieu, and P. Péturaud. 2007. "NEPTUNE: A New Software Platform for Advanced Nuclear Thermal Hydraulics." *Nuclear Science and Engineering* 156 (3): 281-324.

H¨ohne, T., E. Krepper, and U.Rohde. 2009. "Application of CFD Codes in Nuclear Reactor Safety Analysis." *Science and Technology of Nuclear Installations* 2010 (3).

Hamilton, S., M. Berrill, K. Clarno, R. Pawlowski, A. Toth, C. T. Kelley, T. Evans, and B. Philip. 2016. "An assessment of coupling algorithms for nuclear reactor core physics simulations",." *Journal of Computational Physics*, 331: 241–257.

Ilvonen, M., V. Hovi, and P. Inkinen. 2010. *PORFLO development, applications and plans in 2008-2009.* Finnland: VTT.

Imke, U, V Sanchez, and R Gomez. 2010. "SUBCHANFLOW: Anew emprical Knowledge based Subchannel Code." Berlin: German Annual Meeting on Nuclear Technology.

Jeong, J.J., H.Y. Yoon, I.K. Park, and H.K. Cho. 2010. "The CUPID Code Development and Assessment Strategy." *Nuclear Engineering and Technology* 42 (6): 636–655.

Jeong, J.-J., S. K. Sim, C. H. Ban, and C. E. Park. 1997. "Assessment of the COBRA/RELAP5 Code using the LOFT L2-3 Large-Break Loss-of-Coolant Experiment." *Annual of Nuclear Energy* 24 (14): 1171-1182.

Knoll, D. A., and D. E. Keyes. 2004. "Jacobian-free Newton–Krylov methods: a survey of approaches and applications." *Journal of Computational Physics*, 193: 357–397.

Kolev, N., S. Aniel, E. Royer, U. Bieder, D. Popov, and T. Topalov. 2010. "VVER-1000 Coolant Transient Benchmark Phase 2 (V1000CT-2): Final Specifications of the VVER-1000 Vessel Mixing Problem. Report." Paris: OECD Nuclear Energy Agency, NEA/NSC/DOC(2010)10, ISBN: 978-92-64-99152-1.

Kucukboyaci, V., and Y. Sung. 2015. "COBRA-TF Parallelization and Application to PWR Reactor Core Subchannel DNB Analysis." *ANS MC2015 – Joint International Conference on Mathematics and Computation(M&C), Supercomputing Nuclear Applications (SNA) and the Monte Carlo (MC) Method.* Nashville.

Lerchl, G., and H. Austregesilo. 2006. *ATHLET Mod 2.1 Cycle A, User's Manual.* Gesellschaft für Anlagen- und Reaktorsicherheit (GRS) mbH.

Macek, J., and L. Vyskoci. 2013. "Simulation of SDA Opening Test at VVER-1000 NPP by

Coupling System of ATHLET, DYN3D and FLUENT Codes." *NURETH-15 The 15th International Topical Meeting on Nuclear Reactor Thermal - Hydraulics.* Pisa, Italy.

Mahaffy, J., B. Chung, F. Dubois, F. Ducros, E. Graffard, M. Heitsch, M. Henriksson, et al. 2007. *Best Practice Guidelines for the use of CFD in Nuclear Reactor Safety Applications.* Nuclear Energy Agency, Committee on Safety of Nuclear Installations.

Mahaffy, J.; Hansell, B. 2003. *The Exterior Communications Interface and Its Application to the USNRC Consolidated Safety Code TRACE.* Penn State University Applied Research Laboratory.

Moorthi, A., A. K. Sharma, and K. Velusamy. 2018. "A review of sub-channel thermal hydraulic codes for nuclear reactor core and future directions." *Nuclear Engineering and Design* 332: 329–344.

Niceno, B., Y. Sato, A. Badillo, and M. Andreani. 2010. "Multi-scale Modeling and Analysis of Convective Boiling Towards the Prediction of CHF in Rod Bundles." *Nuclear Engineering and Technology* 42 (6): 620-635.

Petruzzi, A., and F. D'Auria. 2007. "Thermal-Hydraulic System Codes in Nuclear Reactor Safety and Qualification Procedures." *Science and Technology of Nuclear Installations* 2008 (3).

1999. *RELAP5-3D code manuals, Volumes I, II, IV, and V.* Idaho National Engineering and Environmental Laboratory.

Rowe, D.S. 1973. *COBRA IIIC: A Digital Computer Program for Steady-State and Transient Thermal Analysis of Rod Bundle Nuclear Fuel Elements, BNWL-1695.* Richland,Washington(USA): Pacific Northwest Laboratory.

Runchal, A.K., and B. Sagar. n.d. *PORFLOW: A multifluid multiphase model for simulating flow, heat transfer, and mass transport in fractured porous media.* Washington, DC, U.S.: Nuclear Regulatory Commission.

Sanchez, V., U. Imke, and A. Ivanov. 2010. "SUBCHANFLOW: A Thermal-Hydraulic Sub-Channel Program to Analyse Fuel Rod Bundles and Reactor Cores." Cancun, Mexico: 17th Pacific Basin Nuclear Conference.

Sánchez-Espinoza, V.H., U. Imke, and A. Ivanov. 2010. "SUBCHANFLOW: A Thermal-Hydraulic Sub-Channel Program to Analyse Fuel Rod Bundles and Reactor Cores." *17th Pacific Basin Nuclear Conference.* Cancún, Q.R., México.

Sung, Y.X., P. Schueren, and A. Meliksetian. 1999. *VIPRE-01 Modeling and Qualification for Pressurized Water Reactor Non-LOCA Thermal-Hydraulic Safety Analysis.* Westinghouse Electric Company.

Thurgood, M.J., J.M. Kelly, T.E. Guidotti, R.J. Kohrt, and K.R. Crowell. 1983. *COBRA/TRAC—A Thermal-Hydraulics Code for Transient Analysis of Nuclear Reactor Vessels and Primary Coolant Systems.* Washington, DC, U.S.: Pacific Northwest Laboratory.

Toumi, I., A. Bergeron, D. Gallo, E. Royer, and D. Caruge. 2000. "FLICA-4: A Three-Dimensional Two-Phase Flow Computer Code with Advanced Numerical Methods for Nuclear

Applications." *Nuclear Engineering and Design* 200 (1-2): 139–155.

US NRC. 2010. *TRACE V5.0 Theory Manual - Field Equations, Solution Methods, and Physical Models.* U.S. Nuclear Regulatory Commission.

2003. *Use and Development of Coupled Computer Codes for the Analysis of Accidents at Nuclear Power Plants.* Vienna, Austria: Proceedings of a Technical Meeting Held in Vienna.

Walker, H. F., and P. Ni. 2011. "Anderson acceleration for fixed-point iterations." *SIAM Journal on Numerical Analysis*, 49: 1715–1735.

Wheeler, C.L., C.W. Stewart, R.J. Cena, D.S. Rowe, and A.M. Sutey. 1976. *COBRA-IV-I: An Interim Version of COBRA for Thermal Hydraulic Analysis of Rod Bundle Nuclear Fuel Elements and Cores, BNWL- 1962.* Richland, Washington(USA): Pacific Northwest Laboratory.

Yadigaroglu, G. 2005. "Computational Fluid Dynamics for nuclear applications: from CFD to multi-scale CMFD." *Nuclear Engineering and Design* 235 (2-4): 153–164.

Yeckel, A., L. Lun, and J. J. Derby. 2009. "An approximate block Newton method for coupled iterations of nonlinear solvers: Theory and conjugate heat transfer applications." *Journal of Computational Physics*, 228: 8566-8588.

Zerkak, O., T. Kozlowski, and I. Gajev. 2015. "Review of multi-physics temporal coupling methods for analysis of nuclear reactors." *Annals of Nuclear Energy*, 84: 225-233.

Zhang, H., J. Guo, F. Li, Y. L. Xu, and T.. J. Downar. 2018. "Efficient simultaneous solution of multi-physics multi-scale nonlinear coupled system in HTR reactor based on nonlinear elimination method." *Annals of Nuclear Energy* 301-310.

Zhang, H., J. Guo, J. Lu, F. Li, Y. Xu, and T. J. Downar. 2018. "An Assessment of Coupling Algorithms in HTR Simulator TINTE." *Nuclear Science and Engineering*, 190: 287–309.

Zhang, H., J. Guo, J. Lu, F. Li, Y. Xu, and T. J. Downar. 2018. "The Improvement of Coupling Method in TINTE by Fully Implicit Scheme." *Nuclear Science and Engineering* 156-175.

Zhang, H., L. Zou, H. Zhao, and R. Martineau. 2014. *Developing Fully Coupled Subchannel Model in RELAP-7.* Idaho National Laboratory Originating Organization.

Zhang, K.L., and V.H. Sanchez-Espinoza. 2018. "Multi-Scale Coupling of the TRACE SUBCHANFLOW based on ECI." Karlsruhe, Germany: Submitted to Progress in Nuclear Energy.

APPENDIX A ECI BUG REPORT

Submitter

Name: Kanglong Zhang

Email: kanglong.zhang@kit.edu

Institute for Neutron Physics and Reactor Technology (INR)

Karlsruhe Institute of Technology (KIT), Germany

Date Seen

12/2016 – 02/2017

Environment

Operating System and version: Windows 7 Enterprise

Software title and version: TRACE V5.1051 and Subchanflow 3.3

Hardware information: Processor – Intel(R) Core(TM) i3 CPU 550 @3.2GHz 3.20 GHz; Installed memory (RAM) – 8.0 GB (7.87 usable); System type – 64 bit

Operating System and version: Debian GNU/Linux 8

Software title and version: TRACE V5.1051 and Subchanflow 3.3

Hardware information: Processor – 48 Intel(R) Xeon(R) CPU E5-2697 v2 @ 2.70GHz; Installed memory (RAM) – 378 GB; System type – 64 bit

Bug Description

Bug 1: Line 2547 – 2549 of subroutine “ReadRequests” in ExTransferM.f90. The three lines will assign the export tables and task tables within an array “sync”. The old counter is “iTbl” which didn’t take the array’s dimension into account.

Bug 2: Line 3180 – 3192 of subroutine “SchedTransmits” in ExTransferM.f90. This belongs to a block that handles the array “putdatalist”. However, some of the array’s names were miswritten to “getdatalist”.

Bug 3: Line 3291 of subroutine “SchedTransmits” in ExTransferM.f90. The pointer “toReal” of the export table in “sync” should be changed to “fromReal”.

Bug 4: Line 409 - 417 of subroutine “AcceptTransmits” in ExTransferM.f90. This is the same bug with bug 1. To fix it, an additional new counter should be adopted.

Bug 5: Line 375 of subroutine “AcceptTransmits” in ExTransferM.f90. This is probably a lapse. The right counter should be “ii” rather than “i”. This bug could lead to meaningless errors.

Bug 6: Line 87 - 89 in ExTransferM.f90. Satellite tasks can’t check their Steady State. This bug especially only impacts the TRACE to TRACE Steady State coupling. This is the definition of the variables which are under the charge of the server and to be transferred to clients to control their calculation. However, there is one variable missing which is named “chkSdy” and used to enable the SS check for clients. This effect can be noticed only for TRACE-TRACE coupling since there

is no need for SCF to check its SS. The SS running by TRACE-TRACE with this ECI bug may lead to unexpected termination.

Bug 7: Line 207 in the main program trac.f90. Generate wrong warning message for TRACE and Subchanflow Steady State coupling. It is not a bug since all the calculation has finished when the code reaches this point. However, it may record unnecessary warnings to the output and msg files and remove the summary. The warning is for SS, which means the user-defined end time is reached before the final convergence. The original statement of this control logic is "IF (stopSoon .AND. normalEndFlag == 0) THEN". For a normal transient run, when the end time was reached, "stopSoon" will turn to be TRUE and "normalEndFlag" turn to be 1. For a normal SS run, "stopSoon" will always stay FALSE and "normalEndFlag" will be forced to be 0 at the end of the calculation. If the end time is reached without convergence during a SS calculation, "stopSoon" will turn to be TRUE and "normalEndFlag" will change to be 1. However, "normalEndFlag" will be reversed to 0 by the code. For the SS run of coupling codes, "stopSoon" is not triggered by the end time but the flag "allStdy" which indicates the SS of the whole system. Once the system reaches SS, "stopSoon" will turn to be TRUE and "normalEndFlag" will be reversed to 0 by the code. Thus active the unexpected warning record. The solution is add another flag to the logic – "IF (transi == 1 .AND. stopSoon .AND. normalEndFlag == 0) THEN". This may effectively filter out the warning message.

Those bugs are mostly encountered when trying to couple TRACE and SCF. Bug 1-5 will cause array transfer errors between TRACE and Subchanflow and will terminate the running. Only bug 6 was found to have a significant impact on TRACE-TRACE coupling. Since the former testing cases which include the four-pipes four-tasks, the three-pipes three-tasks, and the two-pipes two-tasks could successfully run with those bugs unsolved, additional comments should be made to explain why this could happen.

Bug 1: The old codes didn't take arrays into account. While all the testing models didn't even have an array to be transferred.

Bug 2: The key purpose of these lines is to extract the variable type from the putdatalist. But the related variable types in getdatalist and putdatalist of these testing models are the same. It is more like a coincidence rather than specially designed. Moreover, this block deals with arrays while there are no arrays to be transferred for these cases.

Bug 3: This line was not even executed at all by these cases.

Bug 4: These lines were not even executed at all by these cases.

Bug 5: This line was not even executed at all by these cases.

Bug 6: This is a bug not easy to locate since it only affects the calculation occasionally. Whether this bug could be activated depends on the model's initial condition. In most cases, codes could run properly and give the right results. However, for some special initial conditions, this bug could lead to rather confusing results.

Bug 7: This bug has no impact on TRACE-TRACE coupling.

Severity

Trivial, Minor, Major, or Catastrophic

Bug 1-5: Severe bugs for array transfer which will terminate the running.

Bug 6: TRACE could run with this bug existing, while may give out wrong results.

Bug 7: Codes could run with this bug existing and give the right results, while may generate wrong warning messages only for Steady-State coupling of TRACE and Subchanflow.

Steps to Reproduce

Bug 1-5: Run a TRACE and SCF coupling case.

Bug 6: Run a TRACE and TRACE coupling, simulate a Steady State case. Check the SS-check flag “chkSdy” in the satellite TRACEs during the simulation.

Bug 7: Run a Bug 1-5 cleared TRACE and Subchanflow coupling, simulate a Steady State case. Check the final trcout or trcmsg.

Actual Behavior

Bug 1-5: A fatal error message saying message sending or receiving errors occur.

Bug 6: The simulation could finish properly while the result could be strange and hard to understand.

Bug 7: Several messages will be printed on trcout and trcmsg saying that an unexpected termination is reached before the end time.

Expected Behavior

Bug 1-5: No such fatal error messages should appear and the calculation should continue.

Bug 6: The results should be reasonable.

Bug 7: The end of trcout and trcmsg should be a summary of the calculation which may include the CPU time and other convergence information.

Reparation of the Bugs

Bug 1: All the variables “iTbl” codes “sync(iSync)%exportTbl(iTbl)” should be replaced by “iTbl_r:iTbl” and the new “iTbl_r” should be defined at the beginning of the subroutine. Additionally, the line “iTbl_r = iTbl” should be added in front of the “select case” statement.

Bug 2: The “getdatalist” should be rewritten to “putdatalist” within the block.

Bug 3: The pointer “toReal” of the export table in “sync” should be changed to “fromReal”.

Bug 4: This is the same bug with bug 1. To fix it, an additional new counter should be adopted.

Bug 5: This is probably a lapse. The right counter should be “ii” rather than “i”. This bug could lead to meaningless errors.

Bug 6: The Steady State check flag “chkSdy” should be added into the “cpVarPut” list.

Bug 7: Add another flag to the logic “IF (transi == 1 .AND. stopSoon .AND. normalEndFlag == 0) THEN”.

APPENDIX B CHANGES AND NEW SUBROUTINES TO TRACE AND SUBCHANFLOW SOURCE

To implement the TRACE/SCF-ECI system, dozens of modifications and fresh subroutines have to be accomplished to the code's source, for both TRACE and SCF. They are summarized in the following two tables. The bold texts are the real files to be modified or added and the normal texts are the subroutines within the real files.

Important!! – Use the MELD tool or other tools for codes comparison, there the newly-developed and modified subroutines compared to the source codes that could be easily found.

Table B-1 Modifications and New Subroutines to TRACE in TRACE/SCF-ECI

Modification to ECI	Modification to TRACE	New to TRACE
ExTransferM.f90	CountingCompM.f90	InterDataM.f90
cpVarGet	DriverCount	InterFuncM.f90
cpVarPut	CountBreak	alloallinter
AcceptTransmits	CountFill	setfi1toT
ExtTranserialI	ReadTprRstNums	setbi1toT
ExtTranGet	LoadGenHSTab	locatejuncell
ExtTranSend	FillM.f90	getfromTforC
GetCPvarPtr	fillx	getTforC_i
ReadRequests	GlobalDatM.f90	rearrange
SchedTransmits	InfoOut.f90	
SetTasks	SetFlowRate	
Statuscheck	Tprvessel.f90	
SpecExTransM.f90	ReTprVessel	
GetPointer	TracInputM.f90	
	mainCntl	
	UtilM.f90	
	Mean1DFlowVars	
	VessInputM.f90	
	RVssl	
	init.f90	
	trac.f90	
	trans.f90	

Table B-2 Modifications and New Subroutines to SCF in TRACE/SCF-ECI

ECI to SCF	Modification to ECI	Modification to SCF	New to SCF
CfacesM.f90	ExTransferM.f90	mod_var_global.f90	mapping.f90
CIpcFunc.c	cpVarPut	in_calculation_control.f90	ECI_subM.f90
CIpcInclude.h	AcceptTransmits	prop.f90	ECI_processing_1
CmdArgWrapperM.f90	ExtTransferI	shut_down.f90	ECI_processing_2
CmdLineArgsM.f90	ExtTranserialI	subchanflow.F90	SetMissingVar
cTracTypes.h	ExtTransolveI		Do_mapping
error.f90	ExtTranGet		Map_TR_to_sub
GenUtilM.f90	ExtTranSend		Map_sub_to_TR
ErrorInterfaceM.f90	GetCPvarPtr		
ExtEqnArrayM.f90	ReadRequests		
ExTransferM.f90	Resolvecomponents		
FluidVarListsM.f90	SchedTransmits		
HSvarListsM.f90	SetExTransfers		
IntrTypem.f90	TaskDataM.f90		
IoM.f90	SpecExTransM.f90		
StopCode.f90	ClearExtEqns		
SyncPoints.f90	FinishSolve		
TableTransfersM.f90	GetPointer		
xdr.h	LookForComp		
TaskDataM.f90	MultiVarReq		
tracstop.f90	MycopmProp		
SpecExTransM.f90	SerialCalc		
	SerialSolve		
	SetExtJunInfo		

APPENDIX C ELEMENTARY KNOWLEDGE OF COMPUTER ARCHITECTURE

ECI is originally designed to disperse a single serial TRACE process to form a multi-task system. In other words, ECI is the tool for TRACE parallelization. Nevertheless, it was also developed for the coupling of TRACE and other simulation codes, in a parallel way. In this work, ECI is exactly the tool coupling TRACE and SCF. Because ECI involves some Information Technology (IT) related elements, basic knowledge of the computer architecture classification should be clarified before the detailing of ECI, which could be quite helpful for understanding ECI more efficiently and conveniently. Based on architecture, computers can be classified to four types (Figure C-1): Single Instruction Single Data (SISD), Single Instruction Multiple Data (SIMD), Multiple Instruction Single Data (MISD) and Multiple Instruction Multiple Data (MIMD), by the taxonomy of whether they are operating a single set or multiple sets of instructions, and whether those instructions are handling a single set or multiple sets of data.

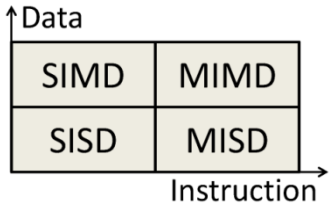


Figure C-1 Computer Architecture Classification

The SISD machine is not parallel since it just deals with one instruction at a time. While the MISD machine does not have too many instances in real life since it is unusual to process one data with more than one instruction. However, a typical MISD example is the computer used in space shuttle which is designated to guarantee the shuttle's stability and safety. As to SIMD and MIMD, they are both machines process multi-data at a time. The instruction number to process data at a time is the main difference between these two machines and the difference can be demonstrated by the following two examples.

Example 1*: For a SIMD machine, suppose that there is a one dimension array A=[1, 2, 3, 4] and all of the array's four elements plus one getting the new A=[2, 3, 4, 5]. As above, the single plus one operation was applied to all of the four elements in A. That is the fundamental working principle for a SIMD machine.*

Example 2*: Suppose that there is a function a+b+c-d+f×g to be solved. This task can be decomposed into three sub-tasks as (a+b), (c-d) and (f×g) by a MIMD machine, which will call three relevant independent operations including addition, subtraction, and multiplication to solve the three sub-tasks at a time. That is the fundamental working principle for a MIMD machine.*

In contrast, MIMD is a more popular parallel computer architecture than SIMD and also a more common computer architecture in life. Two basic elements for this kind of machine may be the computing unit and the memory. Based on how the memory is used by the computing unit, the MIMD machine can be conceptually classified into three types: shared memory, distributed memory and distributed shared memory. The three types of schematic diagrams are presented in Figure C-2.

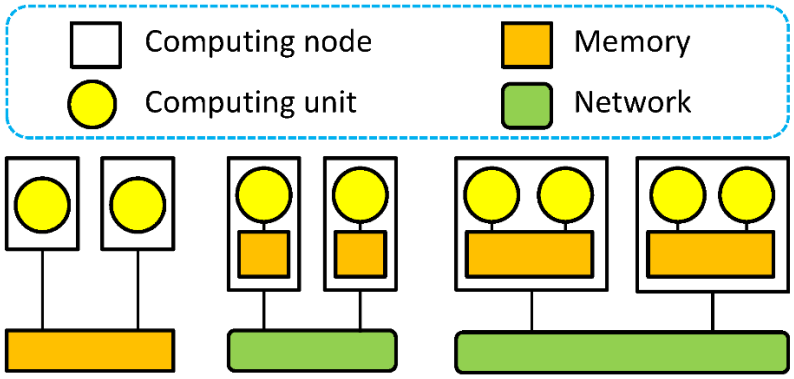


Figure C-2 MIMD Machine Classification

The shared memory machine's computing units work together through accessing a public memory, in other words, the computing units share the memory. Each computing unit can be considered as a computing node just as the Figure C-2 illustrates. The major advantage of this architecture is speed because the data stored in the memory can be processed by all the computing nodes, which means the data exchange can perform rapidly among computing nodes. Nevertheless, this kind of MIMD machine has a good performance only when there are not too many nodes participating in the parallel calculation. This limitation mainly comes from the access violation when too many nodes try to access public memory. Due to the access violation, simply increasing the computing nodes' amount will not get the machine a corresponding growing calculating capability, which even leads in the opposite direction. Generally speaking, the computing efficiency of a single computing node may get worse with the increasing number of nodes. This is also the major disadvantage of the shared memory machine which limits its high performance.

However, this defect can be sufficiently removed by utilizing a distributed memory architecture whose computing units have their isolated memory as Figure C-2 shows. The computing nodes process data in their local memory and they exchange data with other nodes using the network through which the nodes are connected. However, the elimination of access violation of the computing nodes doesn't mean the machine can obtain higher performance by simply increasing the amount of the computing nodes. This is mainly due to the reason that the nodes' network-communication consumption overweighs the access violation at preventing high computing performance. It can be summarized as the more nodes there are, the more communications there will be, the more complex will the system become and the less efficiency will the single node has.

One effective solution to decrease computing nodes to minimize the communications while at the same time maintain the total computing units amount to keep the computing capability was brought forward by combining the advantages of shared memory and the distributed memory – the distributed shared memory, whose architecture is illustrated in Figure C-2. Compared to distributed memory, the distributed shared memory's computing nodes have more than one computing unit, which significantly enhances the computing capability of the single nodes. Within each node, two or more computing units share a public local memory. This kind of shared memory on the local scale while distributed memory in global scale architecture is a better approach to get

a higher performance for parallel computing through decreasing the communications while at the same time introducing no significant access violation among computing units. To get a general understanding of the efficiency of a single computing unit for the three machines, the line sketch Figure C-3 may help.

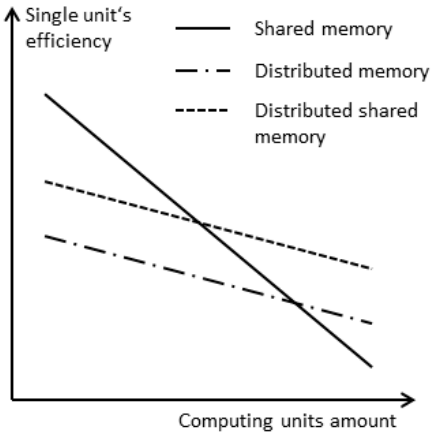


Figure C-3 Variation Trend of Computing Units

Curves in Figure C-3 may not be identical with real conditions, but the trend of the curves may help to reveal the relationships of the three types of MIMD machine architectures. The efficiency of a single computing unit(note the distinction between the computing unit and computing node) in a shared memory machine monotonically decreases as the total unit number increases. This is because that the computing unit can't access the memory properly all the time when it has such a request since other units may be occupying the memory at that time and the unit must wait until the memory becomes free. It can be inferred that the more units there are, the more complex the calling strategy will be and the longer time the unit has to wait for. As to the other two types, communication consumption can be well handled by optimized programming though it can be hardly eliminated. Another point is that since the communication of a distributed shared memory system is less than that in a distributed memory when they share the same units' number, the former unit's efficiency is higher than the latter's. Based on the three concepts, A more detailed classification can be derived for the MIMD machine, which is listed in Table C-1.

Table C-1 Typical MIMD Machines

Abbreviation	Full name	Architecture
PVP	Parallel Vector Processor	Shared Memory
SMP	Symmetric Multiprocessor	
MPP	Massively Parallel Processor	Distributed Memory
DSM	Distributed Shared Memory	Distributed Shared Memory
COW	Cluster of Workstations	

As ECI focus on communications with socket(this will be explained in the following subsection) which is appropriately applied to distributed memory machines (socket may also run on a shared.

memory computer, but that is not its original goal), the PVP and SMP are not covered by the report. The other three types of machine use either distributed memory and distributed shared memory. The two architectures' difference has already been illustrated above. However, there are still three essential differences of these three machines, one is that MPP and DSM use the customized network while COW use internet or other general networks, the second one is that COW use commercial components rather than the specially selected or designed ones which are usually used by MPP and DSM, the last one is that the nodes in COW have their isolated operating systems just like PC while nodes in MPP and DSM have not. The three features give COW better extensibility and maintainability which makes it the most promising supercomputer architecture nowadays.

APPENDIX D ELEMENTARY KNOWLEDGE OF SOCKET

On the software side, the methods to realize inter-process communication for parallel computing mainly include three approaches: Parallel Virtual Machine(PVM), Message Passing Interface(MPI) and Socket. PVM is a software tool for the parallel networking of computers. It is designed to allow a network of heterogeneous Unix and/or Windows machines to be used as a single distributed parallel processor. Thus large computational problems can be solved more cost-effectively by using the aggregate power and memory of many computers. PVM was a step towards modern trends in distributed processing and grid computing but has, since the mid-1990s, largely been supplanted by the much more successful MPI standard for message passing on parallel machines. MPI is a language-independent communications protocol used for programming parallel computers and it is a standardized portable message-passing system designed by a group of researchers from academia and industry to function on a wide variety of parallel computers. The standard defines the syntax and semantics of a core of library routines useful to a wide range of users writing portable message-passing programs in different computer programming languages such as Fortran, C, C++, and Java. MPI uses sockets as the underlying model. A socket is one of the most fundamental technologies of computer networking. Sockets allow applications to communicate using standard mechanisms built into network hardware and operating systems. Since the socket is used by the current version of ECI for inter-process communication, it will be described in detail.

A socket represents a single connection between exactly two pieces of software. More than two pieces of software can communicate in client/server or distributed systems (for example, many web browsers can simultaneously communicate with a single webserver) but multiple sockets are required to do this. Socket-based software usually runs on two separate computers on the network, but sockets can also be used to communicate locally (inter-process) on a single computer. A typical communication system using sockets may be composed of a server and many clients. All of them should have sockets act as points to be connected. A socket is characterized by the following factors: Local socket address - local IP address and port number, Protocol - a transport protocol (e.g.,, TCP, UDP, raw IP). There are several functions to be called to build and connect the sockets and finally establish the connections between servers and clients. They are socket(), bind(), listen(), connect(), accept() and close(). Their functions invocation strategies within server and client have some distinction which is illustrated in Figure D-1 and the detailed description of these functions follows.

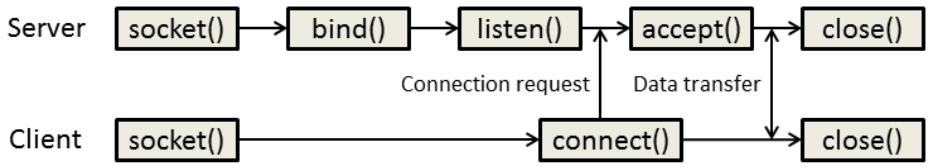


Figure D-1 Invocation Mode of Server and Client

Socket(int *domain*, int *type*, int *protocol*);

This function creates a socket description number for a process. And the socket number is specifically used by the process waiting to establish connections with other processes´ sockets. The function has three parameters.

Domain – protocol domains or protocol family, familiar ones are AF_NET, AF_INET6, AF_LOCAL(or AF_UNIX on Unix) and AF_ROUTE.

Type – several types of Internet socket are available: Datagram sockets, also known as connectionless sockets, which use User Datagram Protocol (UDP); Stream sockets, also known as connection-oriented sockets, which use Transmission Control Protocol (TCP) or Stream Control Transmission Protocol (SCTP); Raw sockets (or Raw IP sockets), typically available in routers and other network equipment. Here the transport layer is bypassed, and the packet headers are made accessible to the application. The stream socket can transfer messages between processes more stably and it is selected to implement in ECI.

Protocol – familiar ones are IPPROTO_TCP, IPPROTO_UDP, IPPROTO_SCTP, and IPPROTO_TIPC. The protocol should be carefully selected to reflect the socket's type, just as described in the "type" subsection.

Bind(int *sockfd*, const struct sockaddr **addr*, *socklen_t*, *addrlen*);

This function binds the host's IP and port to the socket number and it is only used by the server. This is because the server should have a widely known IP and port so that all the clients can conveniently access it whenever there is such a request. The function has four parameters.

Sockfd – the socket description number.

**Addr – a sockaddr structure pointer that contains the information of the server's IP and port number. This indicates the position where the socket points.*

Socklen_t – the address length of the socket.

Addrlen – the IP address length.

Listen(int *sockfd*, int *backlog*);

This function is also only called by the server. It sets the socket of the server to the listening state which means this socket is ready for the connection to a client socket and it will stay active all the time to check whether there is any connecting request from the client socket until it is closed. It has two parameters.

Sockfd – the socket description number.

Backlog – the maximum number of a connection request that could be allowed in a waiting queue.

Connect(int *sockfd*, const struct sockaddr **addr*, *socklen_t*, *addrlen*);

This function is called only by the client and it generates a linking request from the client to the server. The socket number in this function is the client's while the IP and port refer to the server. The connect function substantially sheds light on the relationship between server and client through determining the two ends for a connection. It has four parameters.

Sockfd – the socket description number of the client.

**Addr – a sockaddr structure pointer that contains the information of the target server's IP and port number. This indicates the position where the socket points.*

Socklen_t – the address length of the socket.

Addrlen – the IP address length.

Accept(int *sockfd*, struct sockaddr **addr*, *socklen_t*, **addrlen*);

This function accepts the connecting request from the client to the server. The connection will be successfully established once the function was successfully executed. It will return a new socket number for the server to communicate with the client socket while the original server socket will be released to keep listening to other connecting requests from other clients. It has four parameters.

Sockfd – the socket description number of the server.

**Addr – a sockaddr structure pointer that contains the information IP and port number. This variable may not be predetermined.*

Socklen_t – the address length of the socket.

Addrlen – the IP address length.

With the detailed knowledge of the socket related functions, the process in Figure APP D-1 can be more sufficiently understood. It can be summarized as six steps to make a network connection using a socket. Here, assuming the system only contains one server and one client.

(1) Create a socket description number for the server and the client. Suppose the number is x and y, respectively.
(2) Bind the server's IP and port to its socket number x.
(3) Set the server socket x to the listening state, waiting for the connection request submitted by the client.
(4) The client sends a connection request to the server thorough its socket y.
(5) The server listening socket x receives the connection request sent by the client and accepts this request. A new socket number z is created immediately to displace the old socket x to establish the connection with client socket y. While socket x is released and keeps on listening.
(6) Transfer data and close the sockets.

Some additional points should be made clear for step 5 particularly. Within this step, a new socket z was created to connect to the client socket y and it is classified as the connected socket. The listening socket x is released to keep listening to new connection requests to the server. There are two kinds of sockets existing in the server: the listening socket x and the connected socket z. Usually, there is only one listening socket during a server's lifecycle. It keeps listening and will not be closed. While, there may be many connected sockets existing since each client must have a corresponding socket in the server, the connected socket in the server would be closed once the client finished its work. Up to now, the procedure of establishing a connection between a server and a client has been clearly illustrated, while the actual data communications between those

processes have not been involved. The data transfer are driven by several paired functions: read()/write(), recv()/send(), readv()/writev(), recvmsg()/sendmsg() and recvfrom()/sendto(). Each pair of these functions can drive the data communication process and ECI chooses recv()/send().

Send(SOCKET *s*, const char FAR* *buf*, int *len*, int *flags*);

This function sends the specified data through the local socket within the process. The socket in the process on the other end of the connection will then receive the data.it has four parameters.

S: a socket description number, it is the local socket in the running process.

Buf: a buffer that contains data waiting to be sent.

Len: the buffer length.

Flags: execution method.

Recv(SOCKET *s*, char **buf*, int *len*, int *flags*);

This function receives the specified data from the locally connected socket within the process. The data come from the socket of the process on the other end of the connection.it has four parameters.

S: a socket description number, it is the local socket in the running process.

Buf: a buffer waiting to be filled with received data.

Len: the buffer length.

Flags: execution method.

There are also two modes for the socket I/O operation which is blocking and non/blocking. In blocking mode, the send and receive functions will not return until all the data has been correctly processed. While in the non-blocking mode, the I/O functions don't have to wait for the complement of the data transfer process, they can immediately return with the information of the successfully transferred data size or an error. The two modes can be set by calling a function named ioctlsocket.

Ioctlsocket(SOCKET *s*, long *cmd*, u_long **argp*);

S - A descriptor identifying a socket.

Cmd - A command to perform on the socket s. It can be FIONBIO, zero means forbidding the non-block mode while non-zero values mean allowing the non-blocking mode; FIONREAD, the data size limit to read by the socket; SIOCATMARK, make sure whether all the data has been correctly processed.

Argp - A pointer to a parameter for cmd.

APPENDIX E THE EXTERIOR COMPONENT IN TRACE

E.1.1 Basics of the EXTERIOR in TRACE

A general task may be composed of several sub-tasks or processes and TRACE usually acts as the central task while other codes or other TRACE processes run as clients. Since TRACE plays the most important role in the computation, it is essential to make clear its features for multi-task computation. Assuming that the central processes and all the clients are TRACE, which means each process has an incomplete sub-model decomposed from an integrated model. A correctly built integrated model should run properly without any doubt. Take the TRACE model shown in Figure E-1 for instance.

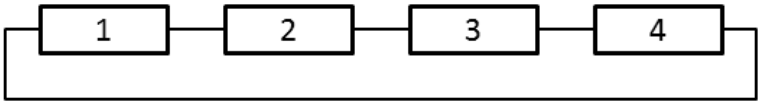


Figure E-1 A TRACE Integrated Model

This model has four components which are all PIPEs and each of them is linked end to end to its adjacent components indicating this is a fluid circuit model. Within the model, every component has a unique number to distinguish from others and also have clearly defined connections to other components. This is a simple but very representative model for a typical single-task TRACE process that can run on its own. The integrated model may need some modifications to be transferred to a dispersed one since a multi-task calculation is desired in this report. The first step is splitting the integrated task into several sub-tasks(the task number is not unique) as Figure E-2 shows.

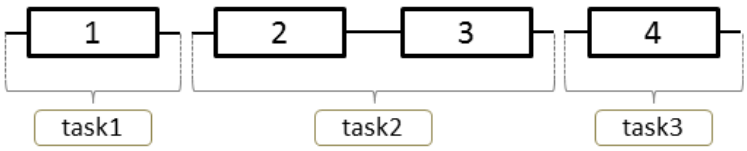


Figure E-2 Split TRACE Model

Task 1 contains one pipe while task 2 and task 3 contain two and one pipes respectively. They will run as three dependent processes and communicate with each other through ECI during the runtime theoretically. But the most basic problem to be solved is how can the tasks recognize the others and by which way the components from one task know which components from other tasks should they connect. Based on solving that problem, the selection of the junction is also a problem to be solved, since one component may have more than one junctions. The key feature used by TRACE to solve those problems is the introduction of the EXTERIOR component. This kind of component is added to the model in Figure E-3 and is displayed in Figure E-3.

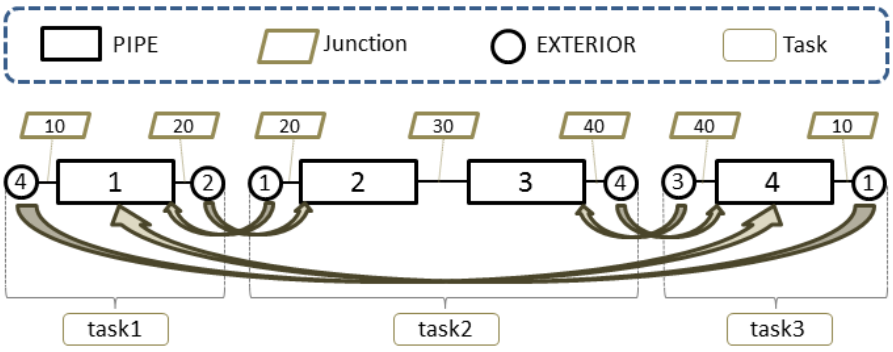


Figure E-3 A Typical TRACE Model for Multi-task Calculation

The figure conveys two pieces of significant information about the connections corresponding to the former two challenges to be solved for the multi-task calculation. The first one is the discriminations of the components to be connected between different tasks. This work was done by specifying the number of EXTERIORs and PIPEs. As the picture shows, every real component (for this model, here refers to the PIPEs) has a unique number under a global scale of all tasks, which means no duplicated numbers are allowed for the component numbering no matter what they occur in one task or several different tasks. With this in mind, one can notice the relationship of the EXTERIORs numbering scheme with the real PIPEs': in the integrated model, PIPE 1's left hand and right hand are connected to PIPE 4 and 2 respectively, while within task 1, the EXTERIOR component which connects the left hand of PIPE 1 has a number same with PIPE 4 in task 3, it is the similar condition for EXTERIOR 2 on the right hand of task 1. It is through the specified numbering of EXTERIORs that the real components know which components in other tasks they should connect. Also, since the real components' numbers are non-redundant, a specified component number just indicates a specified task. So far the components pairs have been determined between different tasks, while the connections have not been set correctly yet. Focus on task 1 and task 4, this problem arises because the two PIPEs both has two junctions as Figure E-3 shows. The left hand of PIPE 1 knows it should connect PIPE 4, but it doesn't know which junction should it connects since PIPE 4 has two junctions. To solve this problem, the junction number of the components must be carefully specified. It can be observed from Figure E-3 that the junction on the left hand of PIPE 1 is 10 and junction on the right hand of PIPE 4 shares the same junction number as 10. This indicates that the left hand of the PIPE 1 should connect PIPE 4's right hand.

Up to this point, the basics of the connection for multi-task processes have been illuminated, it is time to go further. The components can, for the most part, be nodalized into some number of physical volumes, also called cells between which the connections are exactly established rather than the components. Those cells can be classified into internal cells and junction cells depending on their positions in a component. Further on, the junction cells can be classified as exterior junction cells and normal junction cells based upon whether they locate on an exterior component or not. The junctions can also be classified into exterior junctions and normal junctions according to the same principle. Those kinds of cells and junctions within one task are depicted in Figure E-4. What calls special attention is the nodalization of the exterior component because it seems the exterior component has only one cell in one component. That is indeed not an accurate description since the exteriors refer to the actual components in other adjacent tasks, the junction cells in exteriors stand for the junction cells in the related tasks' real component. This point is illustrated by a simple model and will be explained later on. Another point that calls special

attention is that the cell's numbering is different from the components which are on the global scale while the cells are on the local scale. To instantiate this difference, pick up the two normal components in Figure E-4 for instance. The PIPE on the left has two cells 1 and 2 while PIPE on the right has three cells 1, 2 and 3. Compared with the identical global number for components, the cell number is only available within one component. That is to say, the numbering of cells within a component is independent of other components. In reality, only the cells at the end of the components or more generally speaking, the cells which have junctions known as junction-cells, have a meaningful cell number in the TRACE source code. Nevertheless, the number of internal cells is introduced for a better understanding of the data structure.

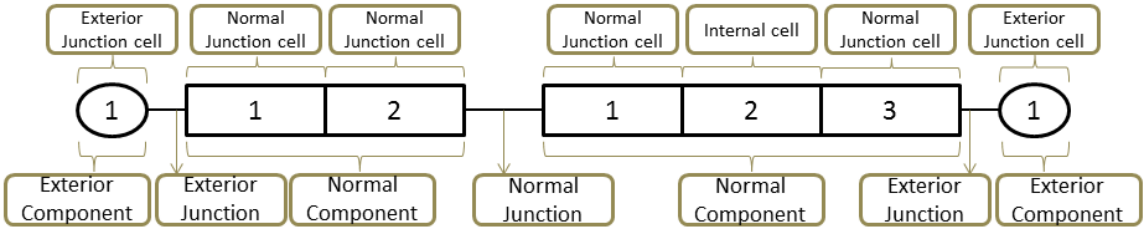


Figure E-4 Typical Classifications of Cells and Junctions for TRACE Multi-task Models

It can be inferred from Figure E-4 that the junctions and junction cells dominant in the tasks' interaction schemes. Two types are intimately involved with the junction cells and the exterior junctions within the TRACE source code, *junctioncellsT,* and *extjunInfoT*. The former type is used to store the general junction cells' information and the latter is specially designed for the exterior junction. Their corresponding arrays are *juncells(:)* and *extjun(:)*. They both contain many variables while some of these variables which are closely involved with the relationships between components and junctions were selected to be presented.

juncells(:) – type junctioncellsT

compNum: the component number assigned in the input deck for the component containing this junction cell.

cellNum: cell number within the component for this junction cell. Only the cells have junctions or the junction-cells, have a meaningful cell number in the TRACE source code.

junNum: junction number assigned in the input deck for this junction.

extjun(:) – type extjunInfoT

compNum: the component number assigned in the input deck for the component containing this junction cell.

junNum: cell number within the component for this junction cell. Only the cells have junctions or the junction-cells, have a meaningful cell number in the TRACE source code.

itaskother: the number of a task which the exterior component connects.

ijunC/E: the index number of a center/edge variable for an exterior junction, it is a local variable, only available within one task.

ijunotherC/E: the number of ijunC/E of the adjacent exterior junctions in the adjacent tasks. It indicates the required variable positions in the adjacent tasks.

ivarC/E: the index number of a center/edge variable for an exterior junction, it is a global variable, this number is unique within all the tasks.

ivarOtherC/E: the number of ivarC/E of the adjacent exterior junctions in the adjacent tasks. It indicates the required variable positions in the adjacent tasks.

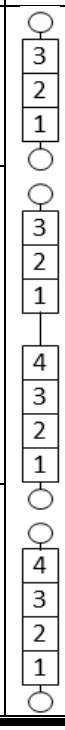
Table E-1gives the number list of those variables for a simplified model. It may help in understanding the numbering strategy of exterior components and junctions in TRACE.

The component number and junction number in Figure E-4 can be set to other values by users, but it is a recommendation using well-organized sets of a number to describe the models to make it more standardized and accessible to others. The remaining variables which are not presented in the table may have some relationship with the TRACE data structure and most of them are specified by the code itself. The following paragraph will focus on those code defined variables.

The first one is cellNum which has already been briefly explained before. This variable is the index of the junction cells' position within a component. Take the components of task 1 for example, the pipe has three cells among which there are 2 junction cells whose cellNum is 1, 3 indicating the two ends of the pipe. The three cells are automatically numbered by the code from one to three in ascending order, and the two ends' cellNum will be 1 and 3 which stands for the cells order. For other types of components such as the TEEs which have 3 junctions and other high dimension components, the conditions may be a little complicated but the value of cellNum is still the cell order number and this concept will be explained in the next section. As to the exteriors of task 1, focus on component 40, since it refers to the pipe 40 in task 3, the cellNum of the junction cells of exterior 40 in task 1 is just the cellNum of the junction cell 1 of pipe 40 in task 3. It is the same case for other junction cells.

Unlike the type junctioncellsT that stores all junction cells information, the type extJunInfoT is only assigned for the junctions of exterior components just as Table E-1 shows. The variable itaskOther is the index of the task connects to the exterior junction and the ijunOther is the index of the exterior junction indicating the order of the junction in a special array. The two variables ijunC and ivarC are used to represent both the cell-centered variables and the cell edge variables. While the ijunC is a local variable indicating the exterior junction position in a related array and ivarC marks the global position of the exterior junction in a global array. The ivarOtherC supplies the position of the target junctions for the local one. Since ijunC is just locally valid, an ijunOtherC is meaningless for the connections between tasks. It should be kept in mind that there are still other variables existing in the two types, but the majorities involved in the establishment of the connections have been explained.

Table E-1 Numbering of Components and Junctions for a Simple TRACE Model

Task No.		Comp No.	junctioncellsT			extJunInfoT						
			comp Num	cell Num	jun Num	comp Num	jun Num	itask Other	ijun Other	ijunC	ivarC	ivar Other
1		40	40	1	100	40	100	3	2	1	1	6
		10	10	3	100							
			10	1	200							
		20	20	3	200	20	200	2	1	2	2	3
2		10	10	1	200	10	200	1	2	1	3	2
		20	20	3	200							
			20	1	300							
		30	30	4	300							
			30	1	400							
		40	40	4	400	40	400	3	1	2	4	5
3		30	30		400	30	400	2	2	1	5	4
		40	40	4	400							
			40	1	100							
		10	10	3	100	10	100	1	1	2	6	1

E.1.2 Low Level of EXTERIOR

It is the EXTERIOR that helps TRACE achieving the multi-task calculations goal. The exterior components can be adapted to deferent components and junctions through the proper setup of its variables. There are not too many exterior variables in the source code and by specifying the variables' values within a small chuck in the input file the key features of the exterior components are scheduled and defined. Those exterior variables include type, num, userid, njuns, comptype, ndim, nx, ny, nz, junnum, junix, juniy, juniz and junface. The following part gives the definition for these variables.

Type – the component type. It is always the exterior.

Num – the component number. It should be the same with this exterior's target component in the adjacent task.

Userid – the identification defined by the user. It uses its default value 0 for most cases.

Njuns – the exterior junctions that the target component in the adjacent task has. It depends on the component type and dimensions.

Comptype – the target component type. It would be fluid, heat structure, power, and special data processing with corresponding numbers of 1, 2, 3 and 4.

Ndim – the target component dimensions. It could be 1~3.

Nx, ny, nz – the target component cells amount along with the directions of the coordinate. When ndim=1, the only nx is needed, when ndim=2, the additional ny is needed and when ndim=3, nz is also needed. The case nx=2, ny=0, nz=0 means the target component has one dimension and 2 cells on x coordinate; The case nx=2, ny=3, nz=0 means the target component has two dimensions and 2 cells on x coordinate, 3 cells on y coordinate; The case nx=2, ny=3, nz=4 means the target component has three dimensions and 2 cells on x coordinate, 3 cells on y coordinate, 4 cells on z coordinate.

Junnum – the junction number. It could be a scalar or vector that just depends on the exterior junction's amount that the target component has.

Junix, juniy, juniz – the indexes indicating the exterior junction position in the target component. When the component has only one dimension, the only junix should present whose value is just the cell number. When the component has more than one dimension, the three variables will be used as combinations to locate the exterior junction on the multi-dimensional component.

Junface - the target side of the hydraulic connection. It could be 6 values with rage from -3 to 3 (no 0) representing the negative and positive directions of x, y, and axial faces.

The variables are defined in the input file and mainly stored in two arrays – exterTab and exterAr within the TRACE data structure. The scalars-comptype, njuns, jdim, nx, ny and nz are stored in exterTab while the arrays-junnum, junix, juniy, juniz and junface are stored in exterAr. The two arrays are type exterTabT and exterArrayT respectively and they also have several other variables within their structure except for the mentioned ones. Another important two derived variables that may help to understand the exterior junctions in the two arrays are ncells and icell which give the total cell amounts and the exterior junction positions for the target components. Several input deck samples for exteriors are put forward to illustrate those obscure concepts. Figure E-5 presents a simple multi-task model that only contains one-dimensional PIPEs components with only one exterior junction.

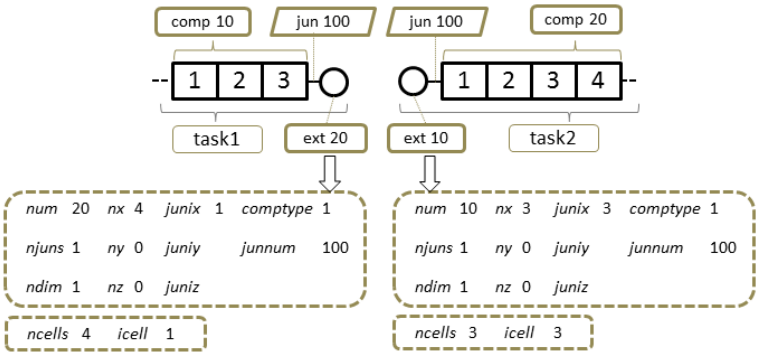


Figure E-5 A TRACE Multi-task Model and its Selected Variables in Input Decks for EXTERIOR

The variables in the upper box are part of the variables defined in the input deck and they are selected to be presented in the figure because they are more characteristic variables than others defining a unique exterior component. The variables in the lower box are the two derived ones

stored in exterTab and exterAr which may help a lot in making the connections between exterior components clear.

Several of the variables in the figure have obvious meanings, such as the num and junnum. They have already been sufficiently explained in the last section. The paired exteriors have the rightly opposite number-num compared with the real pipe components in the model by which way the two real components in two separate tasks find the target components they should connect. The paired exterior junction number-junnum have the same value so that the two components get to know which end of the exterior component they should use for the connection. Because the model is one dimensional and each pipe component has only one exterior junction, the njuns and ndim are both set to 1. The comptype has a value of 1 since both of the two pipes are hydraulic components. As to the dimensional variable group nx, ny and nz, only nx is defined since the model is one dimensional. The nx of exterior 20 is set to 4 to reflect the cell number of component 20 in task 2. The nx's value of exterior 10 is 3 meaning the cell number of component 10 is 3. As to the exterior junction location variable group junix, juniy and juniz, the only junix is defined to locate the junction cell for exterior connection. The junix of exterior 20 is 1 means component 20 in task 2 uses its no.1 cell for the connection and the junix of exterior 10 is 3 indicates component 10 in task 10 uses its no.3 cell for the connection. The derived variables ncells and icell have the same value with nx and junix respectively because for a one-dimensional model, the variables ncells and nx, icell and junix have the same definitions. But for a high dimensional model, their difference may show up. Before exploring the more complicated features for exteriors with a high dimensional model, a one dimensional model with a TEE component are demonstrated to illustrate the multi exterior junction style. This is displayed in Figure E-6.

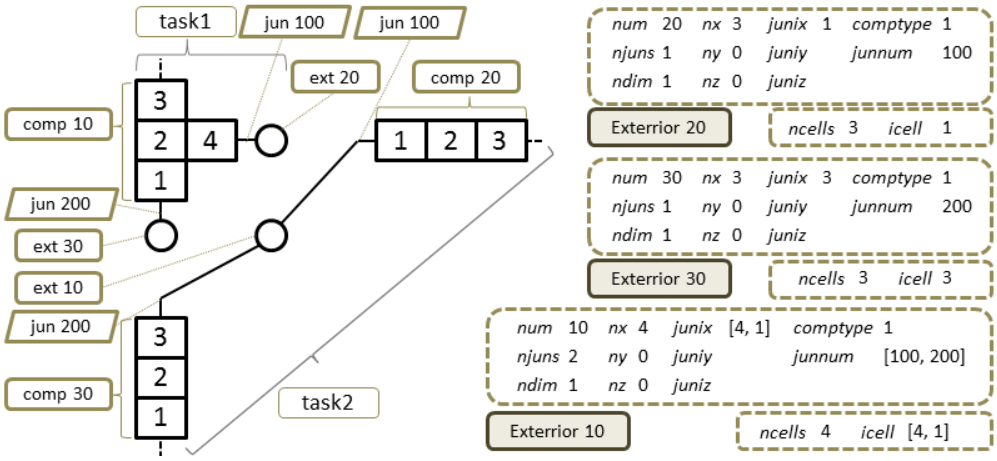


Figure E-6 A TRACE Multi-task Model with Multi Exterior Junctions and Its Input Decks for EXTERIOR

The model is a little more complicated than that in Figure E-5 because the TEE component 10 in task 1 has two exterior junctions. The variables' definitions are the same as argued above while the difference arises in the junix and junnum assignments which are arrays rather than scalars. This change is required because of the need to define two exterior junctions within one exterior component. As the figure shows, component 10 has two exterior junctions connect to cell 4 and 1 numbered as 100 and 200. Assuming the data can be processed as two groups (10, 4, 100) and (100, 1, 200), the first group describes the exterior junction 100 connects cell 4 in component 10 while the other one describes the exterior junction 200 connects cell 1 also in

component 10. In this way, the component which has more than one exterior junction can be properly defined in the input deck and only one exterior icon is required to represent the multi exterior junction component just as the figure shows. The condition may become relatively simple when task 2 can be divided into two separate tasks so that two independent exterior components can be assigned to components 20 and 30 respectively. Then the junix and junnum turn to scalar even if they are originally predefined as arrays in the source code. The two derived tasks decomposed from the previous task 2 both with their inputs for exteriors are shown in Figure E-7.

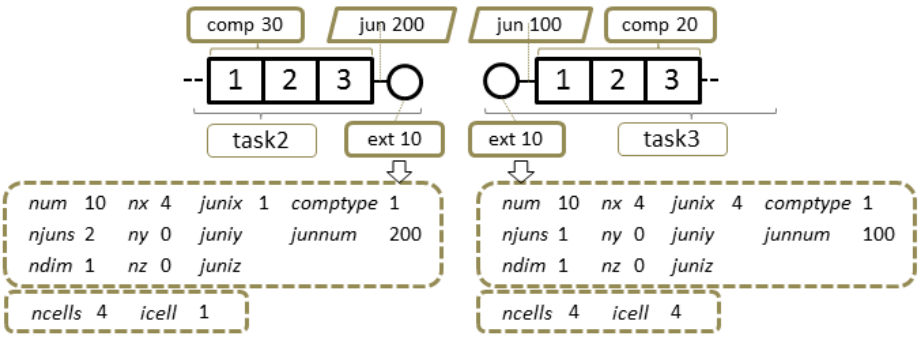


Figure E-7 Two Tasks Sharing the Same Exterior Component

It can be inferred from the figure that the inputs of the two tasks are simply a division of the exterior 10 input in Figure E-7. The junix, junnum and the derived icell of the two tasks are exactly extracted from the corresponding three arrays in original task 2 while other variables just preserve their old values. The point that should be noted is that the task's division is just performed here for the explanation of a multi exterior junction component within a multi-task model from various viewpoints. It may seemingly simplify the exterior input while an input just like exterior 10 shown in Figure E-7 is not that complicated at all. It is the characteristics of the model itself that decide the task division strategy for a multi-task calculation rather than the seeming simplification of the exterior inputs. With the knowledge of the feature of a component having multi exterior junctions, more complicated high dimensional cases are to be illustrated. Such kind of a model is given in Figure E-8.

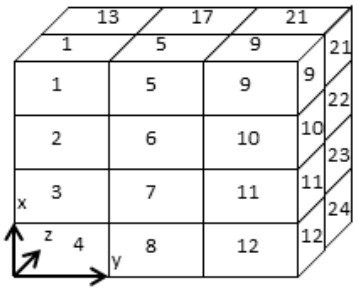


Figure E-8 A Three-Dimensional Component Model

Only the three-dimensional component which connects the exteriors in the task is presented in the figure while others are omitted because the simplified diagram may help to focus on the

explanation of the multi exterior junctions' configuration in a high dimensional component. The component number is 10 and it has four cells along x coordinate, three cells along y coordinate while two cells along z coordinate. The total cell number is 24 while each cell is numbered by an order from top to bottom, left to right and front to back with a range from 1 to 24. The upper part which includes cells 1, 5, 9, 13, 17, 21 are connected to the exterior components, which means there are six potential exterior junctions within the component. Three exterior input demos are presented to illuminate the features of the exterior junctions associated with these three-dimensional components.

(1)

```
num 10   nx 4   junix 1   comptype 1
njuns 1  ny 3   juniy 2   junnum   100
ndim 3   nz 2   juniz 2
ncells 24  icell  17
```

*This is a three dimensional (**ndim**=3) component numbered as 10 which has one exterior junction (**njuns**=1) numbered as 100 (**junnum**=100). It has 4, 3, 2 units along its x, y and z coordinates respectively (**nx**=4, **ny**=3, **nz**=2). And the junction location index is (**junix**=1, **juniy**=2, **juniz**=2) which means the cell number should be **junix**+**nx***(**juniy**-1)+**nx***ny***(**juniz**-1) = 1+4*(2-1)+4*3*(2-1) = 17 (**icell**=17). The component is a hydraulic part (**comptype**=1) and has 24 cells in total(**ncells**=24).*

(2)

```
num 10   nx 4   junix  [1, 1, 1]
njuns 3  ny 3   juniy  [1, 2, 3]
ndim 3   nz 2   juniz  [2, 1, 2]
comptype 1      junnum [100, 200, 300]
ncells 24  icell  [13, 5, 21]
```

*This is a three dimensional (**ndim**=3) component numbered as 10 which has three exterior junction (**njuns**=3) numbered as 100, 200 and 300 (**junnum**=100, 200, 300). It has 4, 3, 2 units along its x, y and z coordinates respectively (**nx**=4, **ny**=3, **nz**=2). And the junction location index are (**junix**, **juniy**, **juniz**) = (1, 1, 2), (1, 2, 1) and (1, 3, 2) which means the cells number should be **junix**+**nx***(**juniy**-1)+**nx***ny***(**juniz**-1) = 1+4*(1-1)+4*3*(2-1) = 13, 1+4*(2-1)+4*3*(1-1) = 5, 1+4*(3-1)+4*3*(2-1) = 21 (**icell**=17, 5, 21). The component is a hydraulic part (**comptype**=1) and has 24 cells in total (**ncells**=24).*

(3)

```
num 10   nx 4   junix  [1, 1, 1, 1, 1, 1]
njuns 6  ny 3   juniy  [1, 2, 3, 3, 2, 1]
ndim 3   nz 2   juniz  [2, 1, 2, 1, 2, 1]
comptype 1      junnum [100, 200, 300
                        400, 500, 600]
ncells 24  icell  [13, 5, 21, 9, 17, 1]
```

*This is a three dimensional (**ndim**=3) component numbered as 10 which has three exterior junction (**njuns**=6) numbered as 100, 200, 300, 400, 500 and 600 (**junnum**=100, 200, 300, 400, 500, 600). It has 4, 3, 2 units along its x, y and z coordinates respectively (**nx**=4, **ny**=3, **nz**=2). And the junction location index are (**junix**, **juniy**, **juniz**) = (1, 1, 2), (1, 2, 1), (1, 3, 2), (1, 3, 1), (1, 2, 2) and (1, 1, 1) which means the cells number should be **junix**+**nx***(**juniy**-1)+**nx***ny***(**juniz**-1) = 1+4*(1-1)+4*3*(2-1) = 13, 1+4*(2-1)+4*3*(1-1) = 5, 1+4*(3-1)+4*3*(2-1) = 21, 1+4*(3-1)+4*3*(1-1) = 9, 1+4*(2-1)+4*3*(2-1) = 17, 1+4*(1-1)+4*3*(1-1) = 1 (**icell**=13, 5, 21, 9, 17, 1). The component is a hydraulic part (**comptype**=1) and has 24 cells in total (**ncells**=24).*

It is to remind here that the exterior components defined by the inputs and the target three-dimensional components shown in Figure E-8 locate in different but adjacent tasks. Another point to note is that if each of the exterior junctions connects to an isolated task, the exterior inputs for the task may share a similar writing style like (1) without any arrays in its inputs data. Up to this point, The features of TRACE that contribute the multi-task calculation were discussed and the essentials required by the establishment of connections between tasks were supplied in this section.

E.1.3 The EXTERIOR functions in TRACE

The previous two sections have illuminated the relationships between tasks for a multi-task calculation because the tasks were originally belonging to an integrated model and must be re-contacted to each other to form an “integrated model” in another style since the tasks may communicate with each other when they were running. Though it is known that the tasks need communication, the root cause for the necessity of the communication is still unclear so far. This section tries to outline this reason.

For a numerical model, one of the most important preconditions to achieve a successful simulation is the proper setting of its boundary conditions. Taking an integrated TRACE model for instance (the upper one in Figure E-9), it is composed of three components: a pipe, a fill, and a break, among which the pipe is divided into seven cells. The fill and break are the flow boundary conditions for this model whose values keep constant during runtime. Then split the model to two sub-models (the lower two in Figure E-9). Compared with the integrated model in a single task that is responsible for the whole solution, the task in a multi-task system is only responsible for the solution for part of the integrated model. The problems arise because only one side of the two sub-models has a fixed value while the other sides remain unknown. Once their undetermined boundaries are decided, it becomes capable to solve the sub-models properly. That is exactly what the two sub-models communicate for.

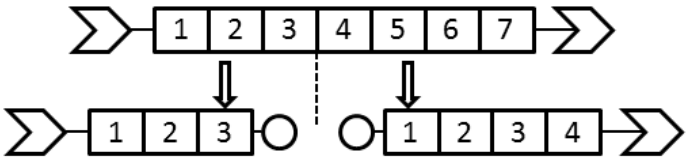


Figure E-9 Two TRACE Sub-models and Their Original Integrated Model

The tasks in a multi-task calculation interact with their adjacent tasks receiving required data to set their boundary conditions. Some form of implicit coupling scheme is used to avoid potential numerical instabilities. Then an exterior equation set specially designed for the solution of the sub-models boundary conditions is derived. All the required information of the tasks to set up the exterior equations are collected by the process taking control of the equation solution. This process could be TRACE as well as other codes such as a process that provides detailed CFD modeling of a specific region within the full system. The multi-task model in Figure E-9 is labeled and represented in Figure E-10.

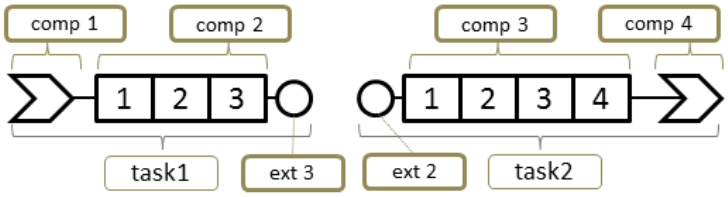


Figure E-10 A Multi-task TRACE Model

Within the model, the cells of component 2 are indexed as 2.1, 2.2 and 2.3. The number before the decimal point indicates the component number while the number behind the decimal point indicates the cell number. The indexes for component 3 cells share a similar pattern. At this stage, focusing on the exterior equations of centered variables and assuming their partial differential equations follow the style: $\frac{\partial \varphi}{\partial t} + \nabla \cdot (a \cdot \varphi) = b$. The centered variables' discrete equations both with their deduction are displayed.

$$\frac{\partial \varphi}{\partial t} + \nabla \cdot (a \cdot \varphi) = b$$

$$\Downarrow$$

$$\frac{\varphi_i^n - \varphi_i^o}{\Delta t} + \frac{a_{i,i+1} \cdot \varphi_{i+1}^n - a_{i-1,i} \cdot \varphi_{i-1}^n}{2 \cdot \Delta x} = b_i$$

$$\Downarrow$$

$$\varphi_i^n = \varphi_i^o + n_i - (m_{i,i+1} \cdot \varphi_{i+1}^n - m_{i-1,i} \cdot \varphi_{i-1}^n)$$

$$\Downarrow$$

$$\varphi_{2.1}^n = \varphi_{2.1}^o + n_{2.1} - (m_{2.1,2.2} \cdot \varphi_{2.2}^n - m_{2.1,fill} \cdot \varphi_{fill})$$

$$\varphi_{2.2}^n = \varphi_{2.2}^o + n_{2.2} - (m_{2.2,2.3} \cdot \varphi_{2.3}^n - m_{2.2,2.1} \cdot \varphi_{2.1}^n)$$

$$\varphi_{2.3}^n = \varphi_{2.3}^o + n_{2.3} - (m_{2.3,3.1} \cdot \varphi_{3.1}^n - m_{2.3,2.2} \cdot \varphi_{2.2}^n)$$

$$\varphi_{3.1}^n = \varphi_{3.1}^o + n_{3.1} - (m_{3.1,3.2} \cdot \varphi_{3.2}^n - m_{3.1,2.3} \cdot \varphi_{2.3}^n)$$

$$\varphi_{3.2}^n = \varphi_{3.2}^o + n_{3.2} - (m_{3.2,3.3} \cdot \varphi_{3.3}^n - m_{3.2,3.1} \cdot \varphi_{3.1}^n)$$

$$\varphi_{3.3}^n = \varphi_{3.3}^o + n_{3.3} - (m_{3.3,3.4} \cdot \varphi_{3.4}^n - m_{3.3,3.2} \cdot \varphi_{3.2}^n)$$

$$\varphi_{3.4}^n = \varphi_{3.4}^o + n_{3.4} - (m_{3.4,break} \cdot \varphi_{break} - m_{3.4,3.3} \cdot \varphi_{3.3}^n)$$

The seven equations are a set for the upper integrated model in Figure E-10 and they will be solved implicitly to get the φ s for new time step. The superscript *n* indicates this is a new time step variable while the superscript *o* indicates an old one. The subscripts of the variables represent the components and cells, which have already explained in the last paragraph. The variables $\varphi_{fill}$and $\varphi_{break}$are the boundary conditions predefined for the fill and break component and keep constant during runtime. The coefficients m and the source items n may be functions of the variables and their calculations use the old variable values so that the equations can keep their linearity. This is a closed equation set and it can be properly solved. However, when the equation set is divided into two separate parts, the integrated matrix no longer exists and the equation set for a separate task becomes non-closed. This feature is shown in the following

formula part. Keep in mind that some variables for a certain cell in a certain component are rewritten to exterior variables within a sub-model. The equations derivation performs below.

$$\begin{aligned}
&\varphi_{2.1}^{n}=\varphi_{2.1}^{o}+n_{2.1}-(m_{2.1,2.2}\cdot\varphi_{2.2}^{n}-m_{2.1,fill}\cdot\varphi_{fill}) && \varphi_{2.1}^{n}=\varphi_{2.1}^{o}+n_{2.1}-(m_{2.1,2.2}\cdot\varphi_{2.2}^{n}-m_{2.1,fill}\cdot\varphi_{fill})\\
&\varphi_{2.2}^{n}=\varphi_{2.2}^{o}+n_{2.2}-(m_{2.2,2.3}\cdot\varphi_{2.3}^{n}-m_{2.2,2.1}\cdot\varphi_{2.1}^{n}) \Rightarrow && \varphi_{2.2}^{n}=\varphi_{2.2}^{o}+n_{2.2}-(m_{2.2,ext2}\cdot\varphi_{ext2}^{n}-m_{2.2,2.1}\cdot\varphi_{2.1}^{n})\\
&\varphi_{2.3}^{n}=\varphi_{2.3}^{o}+n_{2.3}-(m_{2.3,3.1}\cdot\varphi_{3.1}^{n}-m_{2.3,2.2}\cdot\varphi_{2.2}^{n}) && \varphi_{ext2}^{n}=\varphi_{ext2}^{o}+n_{ext2}-(m_{ext2,ext3}\cdot\varphi_{ext3}^{n}-m_{ext2,2.2}\cdot\varphi_{2.2}^{n})
\end{aligned}$$

$$\begin{aligned}
&\varphi_{3.1}^{n}=\varphi_{3.1}^{o}+n_{3.1}-(m_{3.1,3.2}\cdot\varphi_{3.2}^{n}-m_{3.1,2.3}\cdot\varphi_{2.3}^{n}) && \varphi_{ext3}^{n}=\varphi_{ext3}^{o}+n_{ext3}-(m_{ext3,3.2}\cdot\varphi_{3.2}^{n}-m_{ext3,2.3}\cdot\varphi_{ext2}^{n})\\
&\varphi_{3.2}^{n}=\varphi_{3.2}^{o}+n_{3.2}-(m_{3.2,3.3}\cdot\varphi_{3.3}^{n}-m_{3.2,3.1}\cdot\varphi_{3.1}^{n}) \Rightarrow && \varphi_{3.2}^{n}=\varphi_{3.2}^{o}+n_{3.2}-(m_{3.2,3.3}\cdot\varphi_{3.3}^{n}-m_{3.2,ext3}\cdot\varphi_{ext3}^{n})\\
&\varphi_{3.3}^{n}=\varphi_{3.3}^{o}+n_{3.3}-(m_{3.3,3.4}\cdot\varphi_{3.4}^{n}-m_{3.3,3.2}\cdot\varphi_{3.2}^{n}) && \varphi_{3.3}^{n}=\varphi_{3.3}^{o}+n_{3.3}-(m_{3.3,3.4}\cdot\varphi_{3.4}^{n}-m_{3.3,3.2}\cdot\varphi_{3.2}^{n})\\
&\varphi_{3.4}^{n}=\varphi_{3.4}^{o}+n_{3.4}-(m_{3.4,break}\cdot\varphi_{break}-m_{3.4,3.3}\cdot\varphi_{3.3}^{n}) && \varphi_{3.4}^{n}=\varphi_{3.4}^{o}+n_{3.4}-(m_{3.4,break}\cdot\varphi_{break}-m_{3.4,3.3}\cdot\varphi_{3.3}^{n})
\end{aligned}$$

The index 2.3 was replaced by ext2 while 3.1 was replaced by ext3. Eliminate the variables indexed with 2.1 and 2.2 for the upper equation set and the following equation can be derived. This is the exterior equation for task 1.

$$(1+\frac{m_{ext2,2.2}\cdot m_{2.2,ext2}}{1-m_{2.1,2.2}\cdot m_{2.2,2.1}})\cdot\varphi_{ext2}^{n}+m_{ext2,ext3}\cdot\varphi_{ext3}^{n}=\varphi_{ext2}^{o}+n_{ext2}+m_{ext2,2.2}\cdot\frac{\varphi_{2.2}^{o}+n_{2.2}+m_{2.1,2.2}\cdot\varphi_{2.1}^{o}-m_{2.1,2.2}\cdot n_{2.1}-m_{2.1,2.2}\cdot m_{2.1,fill}\cdot\varphi}{1-m_{2.1,2.2}\cdot m_{2.2,2.1}}$$

The exterior equation of task 1 can be written in a style like: $k_{task1,ext2}\cdot\varphi_{ext2}^{n}+k_{task1,ext3}\cdot\varphi_{ext3}^{n}=Q_{task1}$. While the task 2's exterior equation can also be written in a style similar to task 1. The two derived equations finally comprise the exterior equation set of the multi-task process.

$$k_{task1,ext2}\cdot\varphi_{ext2}^{n}+k_{task1,ext3}\cdot\varphi_{ext3}^{n}=Q_{task1}$$

$$k_{task2,ext2}\cdot\varphi_{ext2}^{n}+k_{task2,ext3}\cdot\varphi_{ext3}^{n}=Q_{task2}$$

This is a closed equation set. All the unknowns are on the left side while the known on the right side. The new values of $\varphi_{e}{}^{n}{}_{xt2}$ and $\varphi_{e}{}^{n}{}_{xt3}$ which are also the values of $\varphi_{2.3}^{n}$ and $\varphi_{3.1}^{n}$ are implicitly calculated. The two new values will act as the current boundaries for task 1 and task 2, thus the two tasks' equation sets are closed and can be solved to get the new values for the internal cells. Then the coefficients may be updated using the new values. From the discussion, we get the knowledge that each exterior junction has a unique exterior equation. Considering a loop is a more common model in a PWR plant. The illustration of a loop model's exterior equation set within a multi-task process may be more essential. The model is shown in Figure E-11.

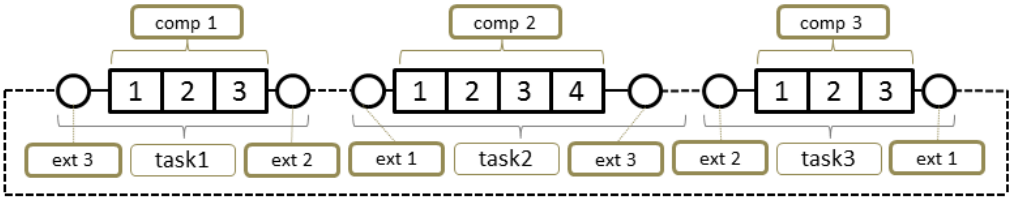


Figure E-11 A Multi-task Loop Model

The model has three pipe components that connect each other end to end and the dashed line stands for the connections between the tasks. Unlike the model demonstrated earlier, each component in this model has two exterior junctions, which means the exterior equation set of this model comprises of six equations. First, get all the cells' discrete equations.

$$\varphi_{1.1}^{n}=\varphi_{1.1}^{o}+n_{1.1}-(m_{1.1,1.2}\cdot\varphi_{1.2}^{n}-m_{1.1,3.3}\cdot\varphi_{3.3}^{n})$$

$$\varphi_{1.2}^{n}=\varphi_{1.2}^{o}+n_{1.2}-(m_{1.2,1.3}\cdot\varphi_{1.3}^{n}-m_{1.2,1.1}\cdot\varphi_{1.1}^{n})$$

$$\varphi_{1.3}^{n}=\varphi_{1.3}^{o}+n_{1.3}-(m_{1.3,2.1}\cdot\varphi_{2.1}^{n}-m_{1.3,1.2}\cdot\varphi_{1.2}^{n})$$

$$\varphi_{2.1}^{n}=\varphi_{2.1}^{o}+n_{2.1}-(m_{2.1,2.2}\cdot\varphi_{2.2}^{n}-m_{2.1,1.3}\cdot\varphi_{1.3}^{n})$$

$$\varphi_{2.2}^{n}=\varphi_{2.2}^{o}+n_{2.2}-(m_{2.2,2.3}\cdot\varphi_{2.3}^{n}-m_{2.2,2.1}\cdot\varphi_{2.1}^{n})$$

$$\varphi_{2.3}^{n}=\varphi_{2.3}^{o}+n_{2.3}-(m_{2.3,2.4}\cdot\varphi_{2.4}^{n}-m_{2.3,2.2}\cdot\varphi_{2.2}^{n})$$

$$\varphi_{2.4}^{n}=\varphi_{2.4}^{o}+n_{2.4}-(m_{2.4,3.1}\cdot\varphi_{3.1}^{n}-m_{2.4,2.3}\cdot\varphi_{2.3}^{n})$$

$$\varphi_{3.1}^{n}=\varphi_{3.1}^{o}+n_{3.1}-(m_{3.1,3.2}\cdot\varphi_{3.2}^{n}-m_{3.1,2.4}\cdot\varphi_{2.4}^{n})$$

$$\varphi_{3.2}^{n}=\varphi_{3.2}^{o}+n_{3.2}-(m_{3.2,3.3}\cdot\varphi_{3.3}^{n}-m_{3.2,3.1}\cdot\varphi_{3.1}^{n})$$

$$\varphi_{3.3}^{n}=\varphi_{3.3}^{o}+n_{3.3}-(m_{3.3,1.1}\cdot\varphi_{1.1}^{n}-m_{3.3,3.2}\cdot\varphi_{3.2}^{n})$$

Replace the index 1.1, 1.3, 2.1, 2.4, 3.1, 3.3 to ext1.1, ext1.3, ext2.1, ext2.4, ext3.1 and ext3.3 and rewrite the equations.

$$\varphi^n_{ext1.1}=\varphi^o_{ext1.1}+n_{ext1.1}-(m_{ext1.1,1.2}\cdot\varphi^n_{1.2}-m_{ext1.1,ext3.3}\cdot\varphi^n_{ext3.3})$$
$$\varphi^n_{1.2}=\varphi^o_{1.2}+n_{1.2}-(m_{1.2,ext1.3}\cdot\varphi^n_{ext1.3}-m_{1.2,ext1.1}\cdot\varphi^n_{ext1.1})$$
$$\varphi^n_{ext1.3}=\varphi^o_{ext1.3}+n_{ext1.3}-(m_{ext1.3,ext2.1}\cdot\varphi^n_{ext2.1}-m_{ext1.3,1.2}\cdot\varphi^n_{1.2})$$

$$\varphi^n_{ext2.1}=\varphi^o_{ext2.1}+n_{ext2.1}-(m_{ext2.1,2.2}\cdot\varphi^n_{2.2}-m_{ext2.1,ext1.3}\cdot\varphi^n_{ext1.3})$$
$$\varphi^n_{2.2}=\varphi^o_{2.2}+n_{2.2}-(m_{2.2,2.3}\cdot\varphi^n_{2.3}-m_{2.2,ext2.1}\cdot\varphi^n_{ext2.1})$$
$$\varphi^n_{2.3}=\varphi^o_{2.3}+n_{2.3}-(m_{2.3,ext2.4}\cdot\varphi^n_{ext2.4}-m_{2.3,2.2}\cdot\varphi^n_{2.2})$$
$$\varphi^n_{ext2.4}=\varphi^o_{ext2.4}+n_{ext2.4}-(m_{ext2.4,ext3.1}\cdot\varphi^n_{ext3.1}-m_{ext2.4,2.3}\cdot\varphi^n_{2.3})$$

$$\varphi^n_{ext3.1}=\varphi^o_{ext3.1}+n_{ext3.1}-(m_{ext3.1,3.2}\cdot\varphi^n_{3.2}-m_{ext3.1,ext2.4}\cdot\varphi^n_{ext2.4})$$
$$\varphi^n_{3.2}=\varphi^o_{3.2}+n_{3.2}-(m_{3.2,ext3.3}\cdot\varphi^n_{ext3.3}-m_{3.2,ext3.1}\cdot\varphi^n_{ext3.1})$$
$$\varphi^n_{ext3.3}=\varphi^o_{ext3.3}+n_{ext3.3}-(m_{ext3.3,ext1.1}\cdot\varphi^n_{ext1.1}-m_{ext3.3,3.2}\cdot\varphi^n_{3.2})$$

These are equation sets of the three tasks while they are all non-closed. To get the exterior equations, eliminate the internal cell variables. Take task 1 for instance.

$$(1-m_{ext1.1,1.2}\cdot m_{1.2,ext1.1})\varphi^n_{ext1.1}+m_{ext1.1,1.2}\cdot m_{1.2,ext1.3}\cdot\varphi^n_{ext1.3}-m_{ext1.1,ext3.3}\cdot\varphi^n_{ext3.3}=\varphi^o_{ext1.1}+n_{ext1.1}-m_{ext1.1,1.2}\cdot\varphi^o_{1.2}+m_{ext1.1,1.2}\cdot n_{1.2}$$

$$m_{ext1.3,1.2}\cdot m_{1.2,ext1.1}\cdot\varphi^n_{ext1.1}+(1-m_{ext1.3,1.2}\cdot m_{1.2,ext1.3})\varphi^n_{ext1.3}+m_{ext1.3,ext2.1}\cdot\varphi^n_{ext2.1}=\varphi^o_{ext1.3}+n_{ext1.3}+m_{ext1.3,1.2}\cdot\varphi^o_{1.2}-m_{ext1.3,1.2}\cdot n_{1.2}$$

The two exterior equations can be rewritten in a more simple style like the previous model. The equations of the other two tasks can also be processed similarly. Finally, the exterior equation set of this model is obtained.

$$k_{1,ext3.3}\cdot\varphi^n_{ext3.3}+k_{1,ext1.1}\cdot\varphi^n_{ext1.1}+k_{1,ext1.3}\cdot\varphi^n_{ext1.3}=Q_{1,task1}$$
$$k_{2,ext1.1}\cdot\varphi^n_{ext1.1}+k_{2,ext1.3}\cdot\varphi^n_{ext1.3}+k_{1,ext2.1}\cdot\varphi^n_{ext2.1}=Q_{2,task1}$$
$$k_{3,ext1.3}\cdot\varphi^n_{ext1.3}+k_{2,ext2.1}\cdot\varphi^n_{ext2.1}+k_{1,ext2.4}\cdot\varphi^n_{ext2.4}=Q_{1,task2}$$
$$k_{3,ext2.1}\cdot\varphi^n_{ext2.1}+k_{2,ext2.4}\cdot\varphi^n_{ext2.4}+k_{1,ext3.1}\cdot\varphi^n_{ext3.1}=Q_{2,task2}$$
$$k_{3,ext2.4}\cdot\varphi^n_{ext2.4}+k_{2,ext3.1}\cdot\varphi^n_{ext3.1}+k_{2,ext3.3}\cdot\varphi^n_{ext3.3}=Q_{1,task3}$$
$$k_{3,ext3.1}\cdot\varphi^n_{ext3.1}+k_{3,ext3.3}\cdot\varphi^n_{ext3.3}+k_{3,ext1.1}\cdot\varphi^n_{ext1.1}=Q_{2,task3}$$

This is the closed exterior equation set for the loop model. The implicitly solved exterior variables will act as the latest flow boundaries of the sub-models. Then the equation sets of the sub-models can be solved.

APPENDIX F WORKING MECHANISM OF ECI IN TRACE

There is two key information derived from Appendix E:

(1) Connections between tasks need to be established since the tasks must communicate with each other to generate the exterior equation set for the whole process and finally obtain their required boundary conditions through the solution of the equation set. It is the essential motivation for the inter-task connection and communication.

(2) The EXTERIOR components are used by TRACE as the nodes waiting to be connected and by applying a specialized junction assignment to the EXTERIOR component the tasks get the potential ability to determine their correspondences.

With all the backgrounds illuminated, it is time to get down to illustrate the real process to build the inter-task connections. This work is under the charge of ECI whose workflow can be primarily divided into five steps (Figure F-1):

(1) Launching all the processes within a multi-task calculation.
(2) Building the connections of tasks and their communication channels.
(3) Building the connections of components and junctions which belongs to different tasks.
(4) Building the connections of variables that belongs to different tasks.
*(5) Performing the data transfer (*Figure F-2*).*

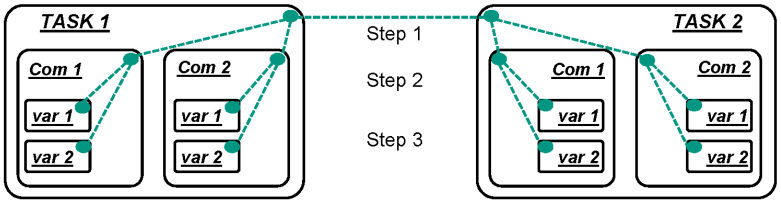


Figure F-1 ECI Steps to Establish the Connections

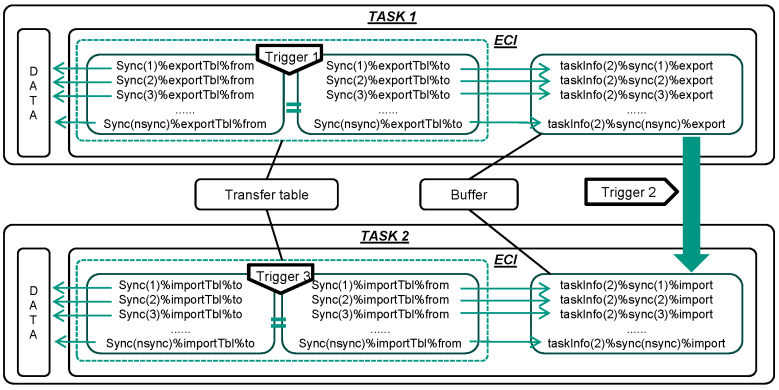


Figure F-2 Data Transfer Between Tasks

The first step is not the function of ECI since it is under the charge of a driver routine which is not an ECI module. However, considering the close relationship between the “driver” and ECI, their

functions were put together for the benefit of an easier way to illuminate the ECI workflow. The illumination shall begin with an introduction of a variable that plays the most important role throughout the process. It is a "TaskInfoT" type, named ***taskInfo(ntasks)***, the ***ntasks*** is the task amount within the multi-task calculation. This variable must be explicitly explained since it stores almost all the information on the inter-task connections. The following part lists its built-in variables.

taskName – the task' name. It is defined in a file naming "taskList" which is edited by the user.

hostname – the host machine's name. It is defined in "taskList" and specifies the machine on which the progress runs.

workingDir – the working directory which stores the process's working directory information.

ProgName – the progress's name which can be TRACE or other codes along with which the progress directory must also be supplied to specify the progress's position on the host.

processID – the process' identification which is unique for each process.

nmissing – the missing component amount of the other tasks. The value comes from other processes.

missing(:) – the array storing the number of the missing components of another task. These values come from other processes.

nFound – the amount of the components which are missing from another task while found in this task.

CompIndex(:) – the local indexes of the component which are missing from another task while found in this task.

nExtComp – the amount of the component which is missing from this task while found in another task. The values come from other processes.

nextJun - the amount of the junction which is defined as an exterior junction in this task while found in another task. The values come from other processes.

extComp(:) – the information of the component which is missing from this task while found in another task. The information comes from other processes.

Type "extCompInfoT":

CompNum – the component number.

finalSolve – the flag claiming control of the final matrix solution.

modeltype – the type of model used by the component.

meshtype – the computational mesh type which can be fixed or variable.

method – the numerical method associated with this process.

extJun(:) – the information of the junction which is defined as an exterior junction in this task while found in another task. The values come from clients and are stored by the central process.

Type "extJunInfoT":

junNum – the junction number.

momEval – the request for evaluation of the junction momentum equation. It can be 0, 1 and 2.

vOutsign – according to the velocity sign convention along the direction perpendicular to the junction face. This is the sign of the local velocity that gives flow out of this task's spatial domain through the junction face.

compNum – the exterior component number that owns this exterior junction.

method – the numerical method used by this process at this flow junction.

iDoEdge – the flag determines which task is responsible for the solution of the edge variables.

junIndex – this is an optional data structure specific index that may be returned for extraction of other junction information in other process-specific operations.

ivarC – the unique index for variables at the cell center adjacent to the junction on a global scale.

ivarE – the unique index for variables at the cell edge directly across the volume from the external junction on a global scale.

ivarotherC – the unique index for variables at the cell center adjacent to the junction in the adjacent task on a global scale.

ivarotherE – the unique index for variables at the cell edge directly across the volume from the external junction in the adjacent task on a global scale.

iJunC – the unique index for variables at the cell center adjacent to the junction on the local scale.

iJunE – the unique index for variables at the cell edge directly across the volume from the external junction on the local scale.

ijunother – the unique index for variables at the cell center adjacent to the junction in the adjacent task.

itaskother – the unique index for the adjacent task which also connects this junction.

constC – the flag indicating whether this cell-centered variable is a constant or not.

constE – the flag indicating whether this cell edge variable is a constant or not.

Sync(nSync) – the array which stores the transferred data at each synchronization point.

Type "transferBuffersT":

nImports(4) – the size of data that needs to be imported. nImports(1) is the size of importR, nImports(2) is the size of importI, and nImports(3) is the size of importC. nImport(4) is the synchronization point index.

importI(:) – the integers need to be imported.

importR(:) – the reals need to be imported.

importC(:) – the characters need to be imported.

nImpIot – the total size of integers, reals, and characters which need to be imported.

nExports(4) – the size of data that needs to be exported. nExports(1) is the size of exportR, nExports(2) is the size of exportI, and nExports(3) is the size of exportC. nExport(4) is the synchronization point index.

exportI(:) – the integers need to be exported.

exportR(:) – the reals need to be exported.

exportC(:) – the characters need to be exported.

nImpIot – the total size of integers, reals, and characters which need to be exported.

Some of the variables listed above have already been illustrated in Figure F-2 which may help the understanding of their meaning. All the variables were summarized and organized together in Figure F-3 to supply a more quick view for ***taskInfo*** and they are defined step by step along with the workflow of ECI. The variables definition process will be presented in the following sections.

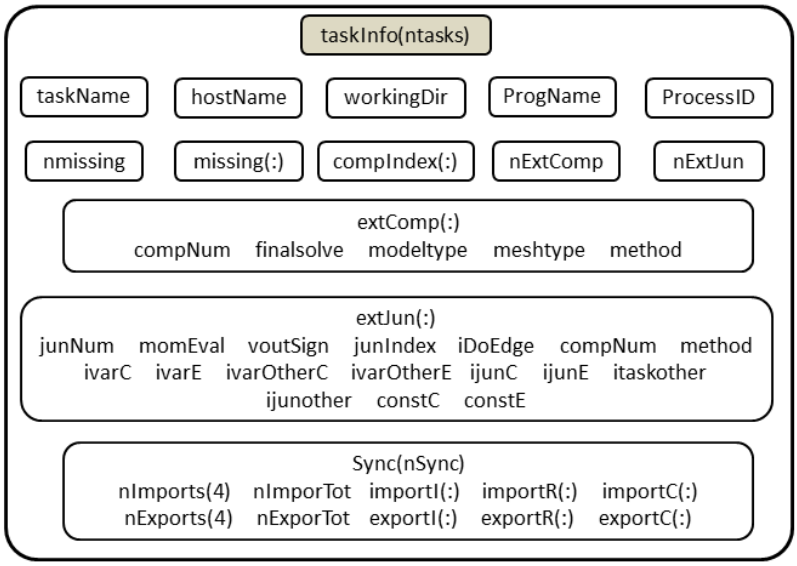


Figure F-3 Summary of the ECI Relevant Variables

1. Establishment of Communication Channels Between Tasks

At the very beginning of the calculation, the process must get some general sufficient information about the tasks which are involved in the whole system. These kinds of information are supplied by the most important file named "***taskList***" which contains all the tasks information for multi-task system simulation and should be implemented by the user (Figure F-4). This file consists of pairs of task descriptor lines and any number of blank or comment (start with # or !) lines. The first line in a task descriptor pair contains an arbitrary name for the task and a path to the executable. The second line in the task descriptor pair contains the name of the host on which

the program will be executed and the working directory for the task. A sample ***taskList*** file is given below.

```
#
trace1      /home/hy5099/V5890/bin/trace
inrsim06.inr.kit.edu      /home/hy5099/four_shiyan/t4a
#
trace2      /home/hy5099/V5890/bin/trace
inrsim06.inr.kit.edu      /home/hy5099/four_shiyan/t4b
#
trace3      /home/hy5099/V5890/bin/trace
inrsim06.inr.kit.edu      /home/hy5099/four_shiyan/t4c
#
trace4      /home/hy5099/V5890/bin/trace
inrsim06.inr.kit.edu      /home/hy5099/four_shiyan/t4d
```

Figure F-4 A Typical taskList File

The sample describes a four task system. The “trace1(2, 3 or 4)” is the task name which specifies a unique task within a multi-task system. The “/home/hy5099/V5890/bin/trace” indicates the execution’ name is “trace” and its route is “/home/hy5099/V5890/bin”. The “inrsim06.inr.kit.edu” is the name of the host on which the execution runs and the remaining descriptor “/home/hy5099/four_shiyan/t4a(b, c or d)” determine the task’s working directory. With the information given by the taskList, the process gets the knowledge of which execution on which host is responsible for which task locating on which directory. And it is possible to start the process now.

The driver and the subroutine “settasks” are responsible for the work summarized as step 1 and 2 of the ECI workflow. And Appendix D which explained the socket concepts in detail could be read as a supplement of this section since ECI uses socket to build the communication channels. Execution of the two subroutines involves the procedures given below:

(1) Driver: Launch the central process and build a connection with it using socket.
(2) Central task: Read the taskList file according to which define every task’s taskInfo(:)% – taskName, ProgName, hostName, and workingDir. Get the central process’s ID and assign it to the taskInfo(1)%processID. Send taskList to the driver.
(3) Driver: Receive the taskList from the central task. Read the clients' information from the taskList and launch them while at the same time get their ID. Send the clients’ ID to the central task.
(4) Central task: Receive the clients’ ID from the driver and save to the taskInfo(2:ntasks)%processID.
(5) All the tasks: Build the connections and establish the communication channels so that each task is capable to communicate with any other tasks.
(6) Central task: Send the processID to all the clients.
(7) Clients: Receive all the processes’ ID from the central task.

The key point is step 5 because it is a substantial operation to establish the inter-task communication channels. According to Appendix D, the functions used by servers and clients are quite different and the sever plays a leading role in the communication strategy. Under such a

strategy, if two clients want to communicate with each other, they must turn to the server. This the most common way that happens on the network. However, ECI applies a more efficient way to realize the communication: non-server system. This means there are no specialized servers or clients and each process within the multi-task system can act as a server or client so that each two of the processes can communicate directly. To achieve the goal, each task should contain the listening sockets both with the connecting sockets. A sample which has four tasks is given to explain the principles of the server-less system. Numbering the four tasks as ta, tb, tc and td. And each of them will create its listening socket array and connecting socket array.

(1) ta: Create the listening socket [x, b, c, d], the x means there is no definition for the socket for a specified place in the array. To supply a clear unique address of ta for the other three tasks tb, tc and td, the port number is calculated by the task according to a special formula within the code, the result is [x, 1, 2, 3]. There is no connecting socket for the first task ta.

(2) tb: Create the listening socket array [x, x, c, d] whose corresponding port number array is [x, x, 6, 7]. Create the connecting socket array [a, x, x, x] whose corresponding port array is [1, x, x, x].

(3) tc: Create the listening socket array [x, x, x, d] whose corresponding port number array is [x, x, x, 11]. Create the connecting socket array [a, b, x, x] whose corresponding port array is [2, 6, x, x].

(4) td: There is no listening socket. Create the connecting socket array [a, b, c, x] whose corresponding port array is [3, 7, 11, x].

All the tasks' IP and port numbers are stored in the structure "SockAddr" which will be used by connecting (). With all the address information prepared, the relationships of the tasks can be inferred from the figure given below. The listening socket and the connecting socket have a one-to-one relationship (Figure F-5).

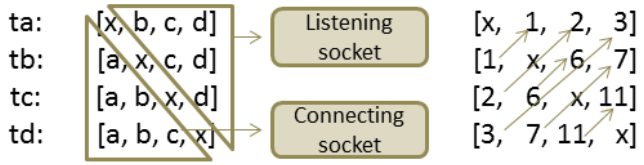


Figure F-5 A Briefly Demonstrated Mechanism of ECI Inter-task Connections

(1) ta: the three sockets b, c and d are bind to the port 1, 2 and 3, respectively. They are set to listening status and wait for the connection requests from other tasks.

*(2) tb: the socket a is used as the connecting socket and sends the connect request by using the command Connect(int sockfd, const struct sockaddr *addr, socklen_t addrlen) within which the *addr contains the target port number 1. Then, the tb's a is connected to ta's b. At the same time, tb also binds the port number 6 and 7 to its sockets c and d and set them to listening.*

*(3) tc: the sockets a and b are used as the connecting socket and send the connect request by using the command Connect(int sockfd, const struct sockaddr *addr, socklen_t addrlen) within which the *addr contains the target port number 2 and 6. Then, the tb's a and b are connected to ta's c and tb's c, respectively. At the same time, tc also binds the port number 11 to its sockets d and set it to listening.*

*(4) td: the sockets a, b and c are used as the connecting socket and send the connect request by using the command Connect(int sockfd, const struct sockaddr *addr, socklen_t addrlen) within which the *addr contains the target port number 3, 7 and 11. Then, the td's a, b and c are connected to ta's d, tb's d and tc' d, respectively.*

Now, unique communication channels are built between each two of the tasks. And the tasks are now capable to send and receive whatever they want. The relationships of the tasks could be represented by the diagram below (Figure F-6).

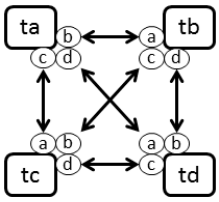


Figure F-6 The Connections of SOCKET within a Multi-task System

It is to note that several of the taskInfo variables were defined in this section and the updated taskInfo is given below. The five variables in the yellow boxes were defined (Figure F-7).

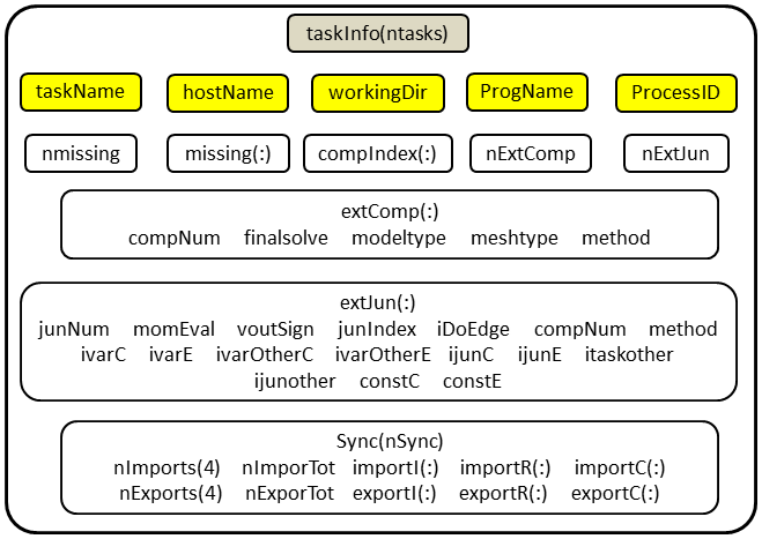


Figure F-7 The Connected Variables During the Task Recognition Phase

2. Establishment of Connections for Exterior Components and Junctions

Still, assuming all the tasks within a multi-task system are TRACE and based on what has been already illuminated in Appendix E, it is known that the exterior component is the basic element that TRACE exactly used as the identified points for the inter-task connections. With this in hand, the tasks will go to the others and look for their needed real components and confirm the junctions through the channels which were just established in the first part of this Appendix. In case of all the elements could be successfully located, the connections of the components and junctions

would be established. This function is in the charge of a subroutine named Resolvecomponents whose workflow could be summarized as follows. The four task model in the last section is still being used and the following illumination will take ta for instance.

(1) ta: Checking the model and finding all the missing components in ta, writing their component number to an array MissingComp(:). After this step, ta know the components which it should get from tb, tc, and td. – subroutine missingList
(2) ta: Sending the ta's MissingComp(:) to tb, tc, and td. – subroutine SendMissingComps
(3) ta: Receiving tb, tc, and td' MissingComp(:), writing to ta's taskInfo(2, 3, 4)%missing(:). Writing the missing components' amount to ta's taskInfo(2, 3, 4)%nmissing. After this step, ta know all the missing components of tb, tc, and td. – subroutine ReadMissingComps
(4) ta: Searching for the components within ta according to ta's taskInfo(2, 3, 4)%missing(:). Finding the component missing from tb, tc, and td while existing in ta. Writing the found components amount to taskInfo(2, 3, 4)%nFound and writing their indexes in ta to taskInfo(2, 3, 4)%CompIndex(1: taskInfo(2, 3, 4)%nFound). After this step, ta know the amount of the components it owns while need by tb, tc, and td, also knows the components' position index within its structure. – LookForComp
(5) ta: Sending the found components' number, model type, mesh type, method, and control type to tb, tc, and td respectively. – subroutine SendComInfo
(6) ta: Receiving the information of the components which are missing from ta while existing in tb, tc, and td. Writing the information to taskInfo(2, 3, 4)%nExtComp and taskInfo(2, 3, 4)%extComp(:)%compNum, finalsolve, modeltype, meshtype, method. After this step, ta know the distribution of its missing components in tb, tc, and td both with the components information. - subroutine ReadCompInfo

Up to now, all the ta's missing components have been found in tb, tc, and td, but their connections to the ta's already existing components remain unknown because one single component may have more than one junctions. This notion has already been illuminated in Appendix E. The remaining steps focus on the definition of the exterior junctions between tasks to complete the inter-task connections. What calls for special attention is the steps 1~6 which work on the components make no distinction between central task and client, or in other words, all the tasks share an equal state. While the following steps which work on the junctions are the central task dominant process and the central task will collect all the other tasks exterior junction information, build the junction connections, send the resolved junctions information back to the clients. Normally, the central task occupies the first place on the taskList which means ta will be the central task for this four task model.

(7) tb, tc, td: Checking themselves and gathering their exterior junction information (actually comes from the cells adjacent to the exterior junctions). – subroutine SendJunToCP
(8) ta: Receiving the exterior junction information from tb, tc, and td. Then checking itself and gathering this kind of information within its structure. Writing those information to taskInfo(1, 2, 3, 4)%extJun(:)%junNum, compNum, momEval, voutSign, junIndex, ivarC, ivarE, method. – subroutine ReadJunsFromSP
(9) ta: This step can be subdivided to several substeps. – subroutine SetExtFlowInfo
    a. Looping each task. For each exterior junction within the task, finding the same junction number in another task and setting the corresponding task's number and the junction's index to the looping task's taskInfo(looping task id)%extJun(:)%itaskOther, ijunOther.
    b. Comparing the two sides of the exterior junction and determining which side will be responsible for the momentum equation solution. Setting the taskInfo(looping

task id)%extJun(:)%iDoEdge to 1 or 0 based on whether this task is responsible for the solution.

c. *For the cell edge variables, setting the taskInfo(looping task id)%extJun(:)%constE to 1 or 0 based on the original ivarE. Assigning an index for the junction and writing to taskInfo(looping task id)%extJun(:)%ijunE. Resetting the taskInfo(looping task id)%extJun(:)%ivarE. The variables assigning principles can be reviewed in section 3.1.*

d. *For the cell-centered variables, the process goes similar to the edge ones. And the taskInfo(looping task id)%extJun(:)%constC, ijunC, ivarC are determined.*

e. *Assigning the taskInfo(looping task id)%extJun(:)%ivarOtherE, ivarOtherC with the ivarE and IvarC of the cell on the other side of the exterior junction.*

(10) ta: Sending all the taskInfo(1:4)%extJun(:) to all the clients. – subroutine SendJunInfo

(11) tb, tc, td: Receiving the taskInfo(1:4)%extJun(:) and writing to their data structure. – subroutine ReadJunInfo

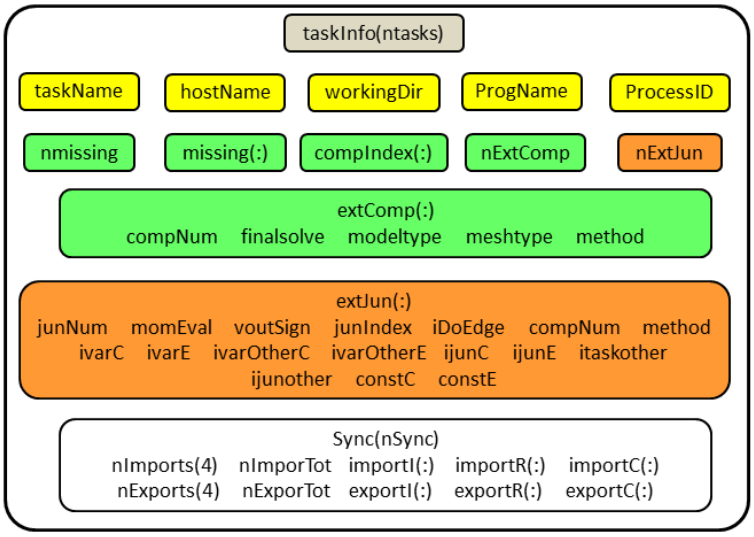


Figure F-8 The Connected Variables During the Component Recognition Phase

Several variables of taskInfo were defined in this section and the updated taskInfo is given above (Figure F-8). The variables in the green boxes are defined by step 1~6 while the ones in the brown boxes were defined by the remaining steps. Up to now, the inter-task connections of components and junctions have been established. The data flow now gets to know the source cell from which it should start, the communication channel through which it should pass and the destination cell to which it should go. Nevertheless, the sources and destinations used above are only the formal ones, while the real ones are the physical variables associated with the cells. The connections of the variables will be completed in the next section.

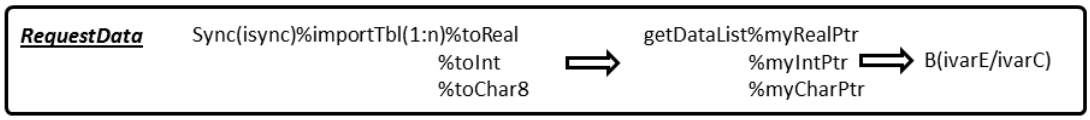
3. Establishment of Connections for Variables

This process is under the charge of a subroutine named SetExTransfers. And two arrays named getDatalist(:) and putDatalist(:) are used to store the information of the task's missing variables that need to be received from other tasks. The information within the two arrays is gathered by

calling to a subroutine named is Missing which assigns special pointers to a position in the TRACE data structure. According to the data's type at this position, the pointers are classified into three types: real, integer and character. Their names within getDatalist(:) and putDatalist(:) are myRealPtr, myIntPtr and myCharPtr. To successfully locate the variables, other required information is also enclosed in the arrays such as the variables index, the number of the junction and component they belonging to, etc.

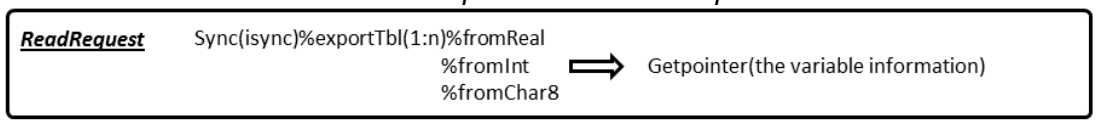
Another significant array to note is the sync(nsync) which will act as a transmission table for the data exchange process. It contains two transfer-related arrays named importTbl(:) and exportTbl(:). The two arrays' structure is both the "transferTableT" type and contains variables such as toReal, toInt, toChar8, fromReal, fromInt, and fromChar8, etc. The relationships of the variables are explained in four steps and illustrated by several simple figures.

(1) (a) Getting the position of the missing variable within the TRACE data structure. Assigning the pointer getDatalist(:)%myRealPtr or %myIntPtr or %myCharPtr to this position according to the data type it should be. Additionally, assigning another pointer Sync(isync)%importTbl(1:n)%toReal or %toInt or %toChar8 to getDatalist(:)%myRealPtr or %myIntPtr or %myCharPtr. This operation is shown in the following figure. The leftmost bold RequestData is the name of the subroutine which is responsible for this step. While the rightmost B is the right side of a matrix within the TRACE data structure. The ivarE and ivarC indicate a determined position in B.

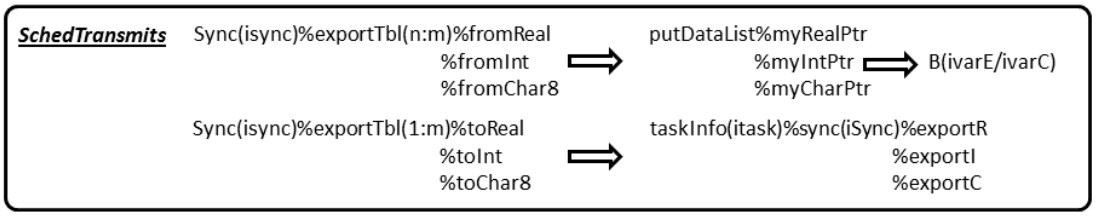


- *(b) Gathering the variables information which was used for their locating in other tasks and saving the information in getDatalist(:). Then sending the information to the designated tasks.*

(2) Receiving the locating information of the missing variables form others. According to the information, finding the corresponding variables existing in this task while need by other tasks. Assigning a pointer to the variables found in this task by calling a subroutine named Getpointer(). Additionally, assigning another pointer Sync(isync)%exportTbl(1:n)%fromReal or %fromInt or %fromChar8 to the existing one. This operation is shown in the following figure. The leftmost bold ReadRequest is the name of the subroutine which is responsible for this step.

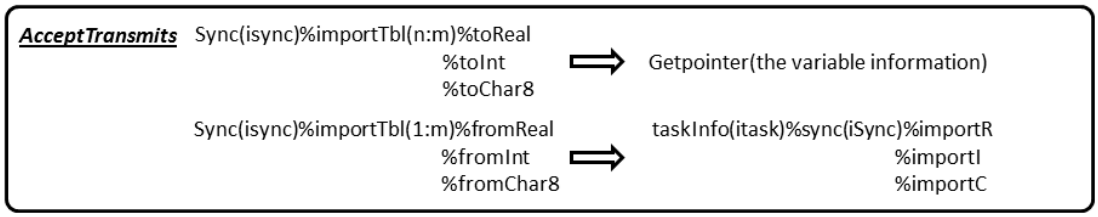


(3) (a) Adding some extra variables to the variable list waiting to be sent to other tasks. These kinds of variables information are saved in putDatalist(:). Their corresponding pointers within putDatalist(:) are putDatalist(:)%myRealPtr or %myIntPtr or %myCharPtr. Assigning other extra pointers Sync(isync)%exportTbl(n:m)%fromReal or %fromInt or %fromChar8 to the existing ones. Next, assigning the pointers Sync(isync)%exportTbl(1:m)%toReal or %toInt or %toChar8 to the taskInfo(itask)%sync(iSync)%exportR or %exportI or %exportC. Then, the connection of the TRACE-built transfer table Sync(isync)%exportTbl and the ECI-built transfer table taskInfo(itask)%sync(iSync) is established. This operation is shown in the following figure. The leftmost bold SchedTransmits is the name of the subroutine which is responsible for this step. This step completes transfer tables started by subroutine ReadRequests.



- *(b) Gathering the variables information which was used for their locating in other tasks and saving the information in putDatalist(:). Then sending the information to the designated tasks.*

(4) Receiving the locating information of the extra missing variables from other tasks. According to the information, finding the corresponding variables existing in this task. Assigning a pointer to the variables found in this task by calling a subroutine named Getpointer(). Additionally, assigning another pointer Sync(isync)%importTbl(1:n)%toReal or %toInt or %toChar8 to the existing one. Next, assigning the pointers Sync(isync)%importTbl(1:m)%fromReal or %fromInt or %fromChar8 to the taskInfo(itask)%sync(iSync)%importR or %importI or %importC. Then, the connection of the TRACE-built transfer table Sync(isync)%importTbl and the ECI-built transfer table taskInfo(itask)%sync(iSync) is established. This operation is shown in the following figure. The leftmost bold ReadRequest is the name of the subroutine which is responsible for this step.



After this step, the taskInfo(ntask) was completely defined (Figure F-9).

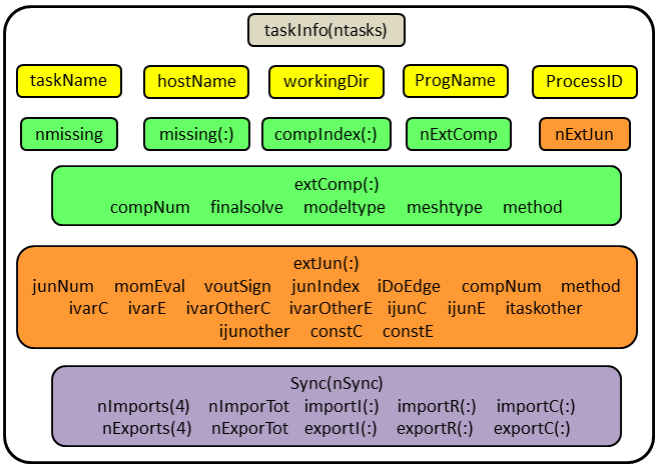


Figure F-9 The Connected Variables During the Variable Recognition Phase

The variables in the light purple box are defined in this step. The definition of the first two variables in each row is not explicitly explained during the illumination. They are the counters of the transferred data. The four elements in nImports(4) stand for the number of real, integer, character

to be imported and their synchronization index respectively. The nImporTot is the total number of imported data. The nExports(4) and nExporTot store similar information with the former two but they are designed for data to be exported. All of the four arrays are defined along with the definition of the other six arrays of taskInfo(:)%sync(nSync). With the complement of taskInfo(:), the connections of the variables are established and illustrated in Figure F-10 (a four tasks model was used here, just like the last two sections of this chapter).

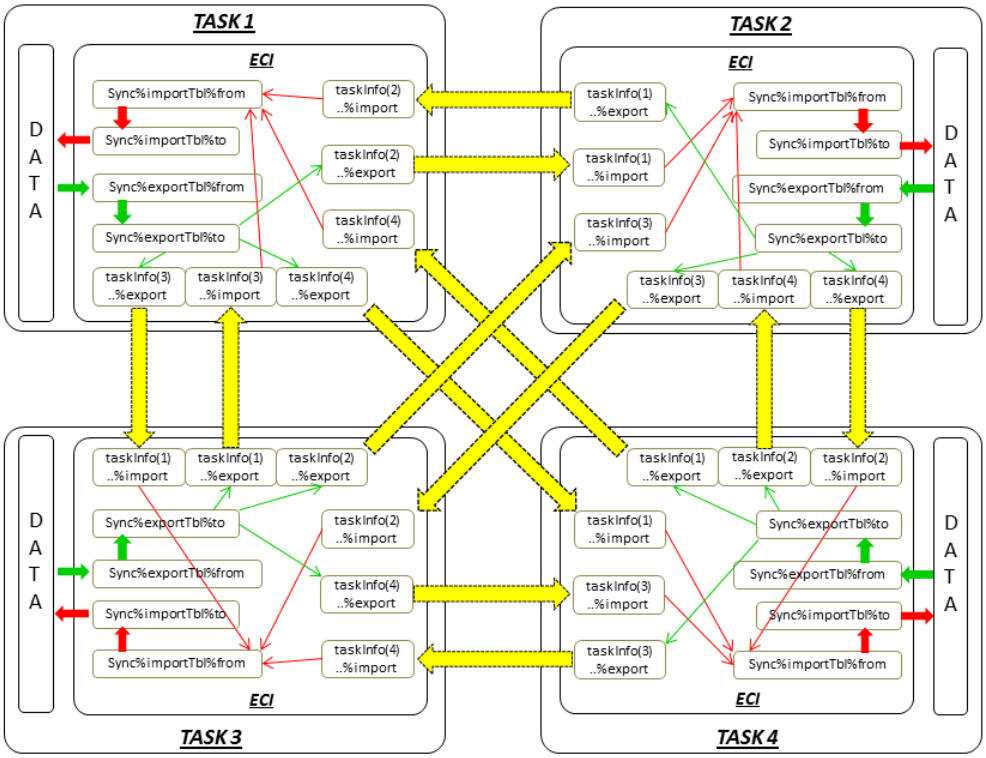


Figure F-10 The Connections of the Variables Between Tasks

The difference of Sync(nSync) and taskInfo(:)%sync(nSync) which can be inferred from Figure F-10 is that Sync(nSync) stores the pointers of all the variables to be exchanged with other tasks, while taskInfo(:)%sync(nSync) stores the pointers of the selected variables to be exchanged with a designated task. Taking task 1 for instance, the taskInfo(2, 3, 4)%sync(nSync)%import store the variables from task2, 3, 4 respectively. Those variables from three different arrays are then put together to an integrated array Sync(nSync)%importTbl%from. The Sync(nSync)%exportTbl%to stores the pointers of all the variables to be sent to other tasks. Then, the variables are classified and rewritten to taskInfo(2, 3, 4)%sync(nSync)%export according to the number of the task to which they should be sent. Actually, Sync(nSync)%importTbl%from and Sync(nSync)%exportTbl%to can be treated as the unions of taskInfo(2, 3, 4)%sync(nSync)% import and taskInfo(2, 3, 4)%sync(nSync)%export respectively.

There are a total of 12 channels for the four tasks system, which may be a little complicated for a clear explanation. Picking task 1 and task 2 from the system and just focusing the data transfer from task 1 to task 2. This single process is illustrated in Figure F-11 and some triggers

required by the process are also illuminated. The arrows between data and sync%export%from within task 1 and the arrows between taskInfo%sync%import and sync%import%to in Figure APP F-10 have opposite directions compared with Figure F-11. This is because Figure F-10 illustrates the actual data flow between tasks while Figure F-11 illustrates the pointer definition within the codes. The data exchange shown in Figure F-11 can't be realized at this stage, because only the variable pairs of data/ sync%export%from, sync%export%to/taskInfo%sync% export, taskInfo%sync%import/ sync%import%from and sync%import%to/data were connected through pointers after the four steps of subroutine SetExTransfers. While the two sets within the transfer table have not been connected. There are three "triggers" to be activated whenever a data transfer is required during the runtime. The first one sets the syns%export%from equal to syns%expor%to, completes the connections within the export transfer table. The second one sends the values of the actual variables from the export buffer of task1 to the import buffer of task2. The last one sets the syns%import%from equal to syns%impor%to, completes the connections within the import transfer table. The accrual data transfer occurs once the three operations were performed.

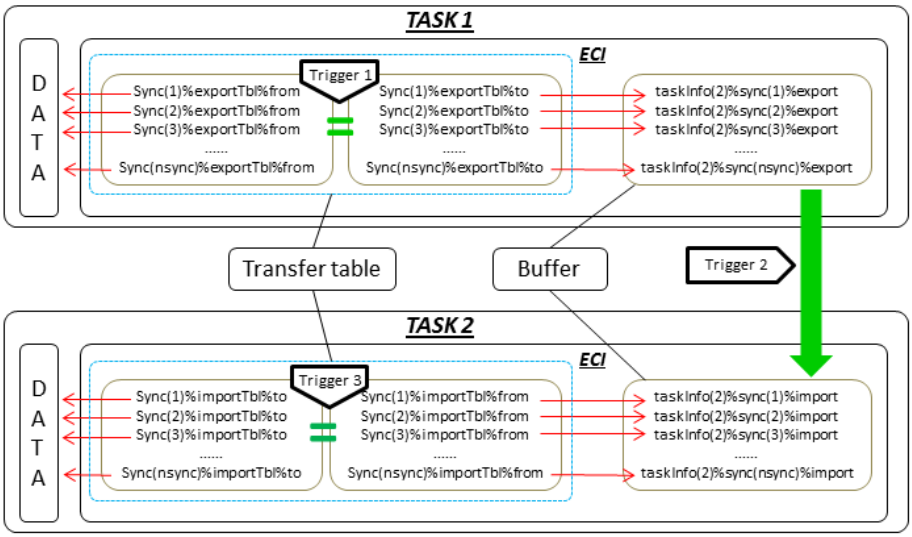


Figure F-11 Data Transfer from Task 1 to Task 2

The three "triggers" explained in the last paragraph are exactly the main functions of a subroutine named ExtTransfer(other two similar subroutines are ExTranSerial and ExTranSolve, which will make some modification to the data to be transferred) which will trigger the real data transfer process through the prepared channels. Now, we get the overall knowledge if ECI that the main body of ECI works on the build of the communication channels and when the channels are finally successfully established, the tasks may call an ECI supplied subroutine ExtTransfer to exchange data with other tasks whenever there is such a request.

APPENDIX G FROM SNAP EXPORTED TRACIN TO ECI ACCEPTABLE TRACIN

From the practical experience, it is better to use SNAP to generate the TRACE input file – tracin. This is also common sense to use SNAP to export the multi-task tracins for ECI usage. However, the directly exported tracin from SNAP can not be properly applied to the multi-task simulations where more than one TRACE was involved. This is because the SNAP exported tracin with the EXTERIOR component contains some contents which TRACE can not recognize. Presents a case that involves three tasks. The meaning of the symbols is exactly the same as Appendix E.

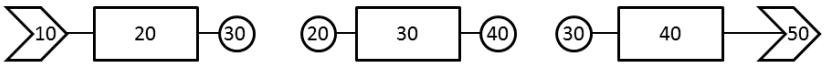


Figure G-1 The Three-pipe Model Demonstrating the Modifications from the Original SNAP Exported Tracin to ECI Accepted Tracin

To achieve a successful multi-task simulation, the following items have to be followed.

1) Based on the original tracin which is exported by SNAP, the “ncomp” should include the EXTERIORs.
2) Based on the original tracin which is exported by SNAP, the “component-number data” – “component input order (IORDER)” list should include also the EXTERIORs.
3) The most important: the central task should contain all EXTERIORs in the whole system. But pay attention, the “njuns” of the non-existing EXTERIORs in the central task should be 0. And the “junnum”, “junix” and “junface” should be empty.
4) Keep in mind: This is especially for TRACE V51051. The special version for coupling with SCF should be not used for pure TRACE.

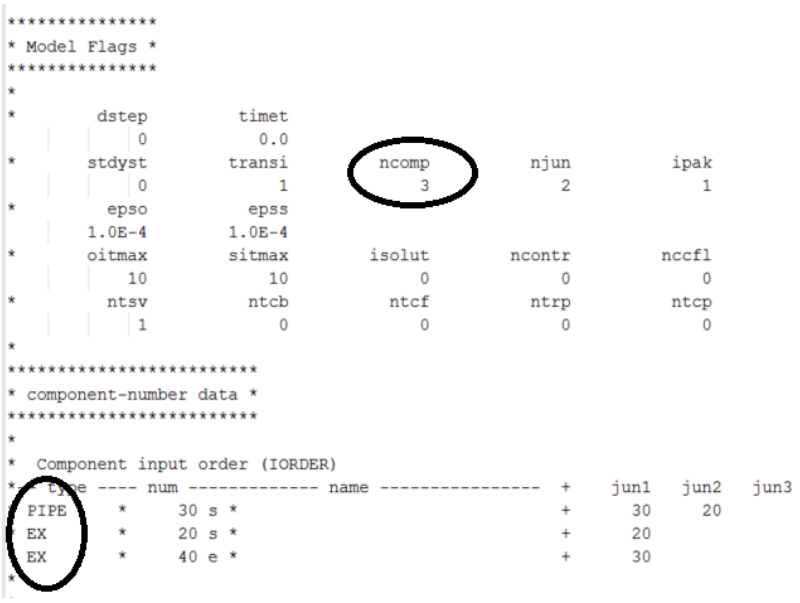
All of the tracin of the three tasks are displayed in the following figures.

```
***************
* Model Flags *
***************
*
*        dstep          timet
*            0            0.0
*       stdyst         transi         ncomp          njun          ipak
*            0              1              4             2             1
*         epso          epss
*       1.0E-4        1.0E-4
*       oitmax        sitmax        isolut        ncontr         nccfl
*           10            10             0             0             0
*         ntsv          ntcb          ntcf          ntrp          ntcp
*            1             0             0             0             0
*
************************
* component-number data *
************************
*
*  Component input order (IORDER)
*   type ---- num ------------ name ----------------  +    jun1   jun2   jun3
*  BREAK     *     10 s *                             +      10
*  PIPE      *     20 s *                             +      20     10
*  EX        *     30 s *                             +      20
*  EX        *     40 e *                             +      30
*
*
************************************************
* Starting Signal Variable Section of Model      *
************************************************
*
```

Figure G-2 The General Definition of the Central Task Tracin

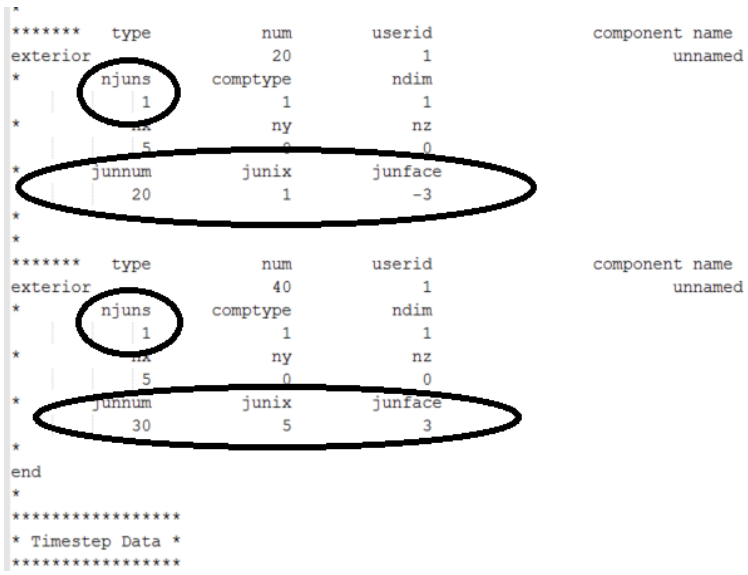
```
*
*******   type           num         userid                component name
exterior                  30              1                       unnamed
*         njuns       comptype          ndim
              1              1             1
*            nx             ny            nz
              5              0             0
*        junnum           junix        junface
             20              5             3
*
*******   type           num         userid                component name
exterior                  40              1                       unnamed
*         njuns       comptype          ndim
              0              1             1
*            nx             ny            nz
              5              0             0
*
end
*
*****************
* Timestep Data *
*****************
```

Figure G-3 The EXTERIOR Definition of the Central Task Tracin



```
***************
* Model Flags *
***************
*
*        dstep         timet
*            0           0.0
*       stdyst        transi        ncomp         njun         ipak
*            0             1            3            2            1
*         epso         epss
*       1.0E-4       1.0E-4
*       oitmax        sitmax       isolut       ncontr       nccfl
*           10            10            0            0            0
*         ntsv          ntcb         ntcf         ntrp         ntcp
*            1             0            0            0            0
*
*************************
* component-number data *
*************************
*
*  Component input order (IORDER)
*  type ---- num ------------ name ----------------  +    jun1   jun2   jun3
  PIPE     *     30 s *                              +      30     20
  EX       *     20 s *                              +      20
  EX       *     40 e *                              +      30
*
```

Figure G-4 The General Definition of Task 2



```
*
*******   type          num        userid              component name
exterior                20             1                       unnamed
*         njuns       comptype        ndim
              1             1             1
*                         ny            nz
              5             0             0
*        junnum         junix        junface
             20             1            -3
*
*
*******   type          num        userid              component name
exterior                40             1                       unnamed
*         njuns       comptype        ndim
              1             1             1
*                         ny            nz
              5             0             0
*        junnum         junix        junface
             30             5             3
*
end
*
*****************
* Timestep Data *
*****************
```

Figure G-5 The EXTERIOR Definition of Task 2

```
***************
* Model Flags *
***************
*
*        dstep         timet
               0           0.0
*       stdyst        transi        ncomp         njun         ipak
               0             1            3            2            1
*         epso         epss
            1.0E-4       1.0E-4
*       oitmax        sitmax       isolut       ncontr        nccfl
              10            10            0            0            0
*         ntsv         ntcb         ntcf         ntrp         ntcp
               1            0            0            0            0
*
*************************
* component-number data *
*************************
*
*  Component input order (IORDER)
*-- type ---- num ------------ name ----------------  +    jun1   jun2   jun3
  PIPE     *     40 s *                                +      40     30
  FILL     *     50 s *                                +      40
  EX       *     30 e *                                +      30
*
```

Figure G-6 The General Definition of Task 3

```
*
*******   type          num        userid                component name
exterior                30             1                        unnamed
*         njuns       comptype         ndim
               1             1            1
*            nx            ny           nz
               5             0            0
*       junnum          junix        junface
              30             1           -3
*
end
*
*****************
* Timestep Data *
*****************
```

Figure G-7 The EXTERIOR Definition of Task 3

APPENDIX H THE PRINCIPLES OF THE OVERLAPPED AREA-WEIGHTED MESH MANIPULATION SUBROUTINE MAPPING.F90

In this work of coupling TRACE and SCF, a user subroutine for field and mesh mapping was developed to generate the two arrays for cases whose sections and channels might have arbitrary nodal, area, shape and location, automatically. The subroutine can handle complicated cases by covering all possible configurations. Since the calculation of the overlapping area for a channel with arbitrary shape is meaningless (SCF only need the area of the channels and don't care about their shapes), one key assumption assuming that all the channels are circles is proposed. The subroutine will first classify the channels into two basic classes according to their relative positions to the TRACE center (Figure H-1-1 and Figure H-1-2). The channels which cover the center point belong to class one and are further classified into three sub-classes according to their relative positions to the first TRACE radial section (Figure H-2-1.1, 1.2, 1.3).

1) Sub-class 1.1: the channel fully covers the first TRACE ring;
2) Sub-class 1.2: the channel intersects the first TRACE ring;
3) Sub-class 1.3: the channel locates fully in the first TRACE ring.

The channels which separate from the center point belong to class two and are further classified into three sub-classes according to the relative positions of their centers and the x-axis.

1) Sub-class 2.1: the channel center locates on the x-axis;
2) Sub-class 2.2: the channel center locates up upon x-axis;
3) Sub-class 2.3: the channel center locates under the x-axis.

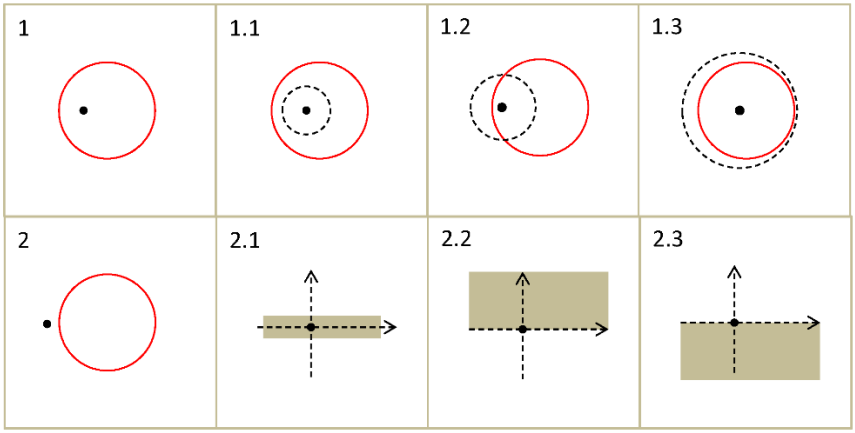


Figure H-1 Two Basic Classes of the Position of the SCF Channels

The sub-classes 1.1 to 1.3 can be further classified as Figure H-2 presents. As before, the red circle represents the SCF channel. The black dash circles represent TRACE radial sections. The new blue circles represent some possible indeterminate TRACE radial sections.

1) Sub-class 1.1.1: the channel is intersected by more than one ring;
2) Sub-class 1.1.2: the channel is not intersected by any rings;
3) Sub-class 1.2.1: the channel is intersected by more than one ring other than the first ring.

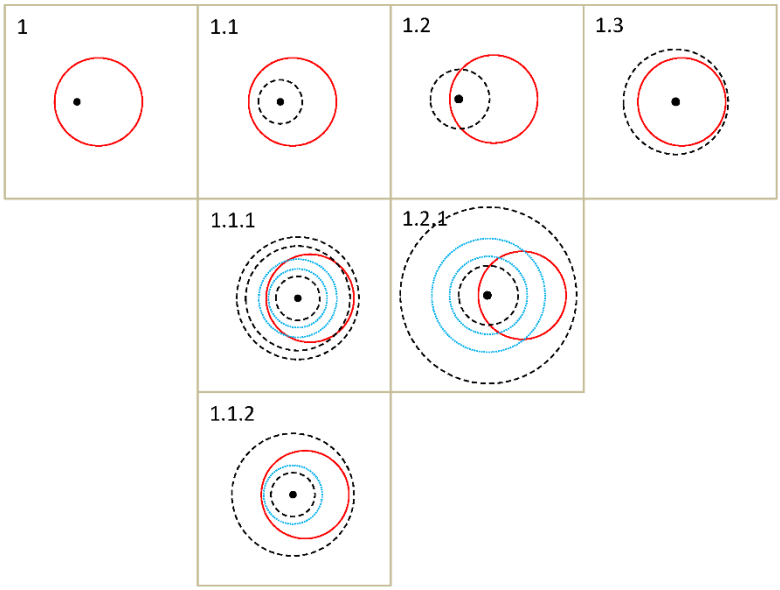


Figure H-2 Specific Classification of Class One

Sub-classes 2.1 to 2.3 can be further classified as Figure H-3 presents.

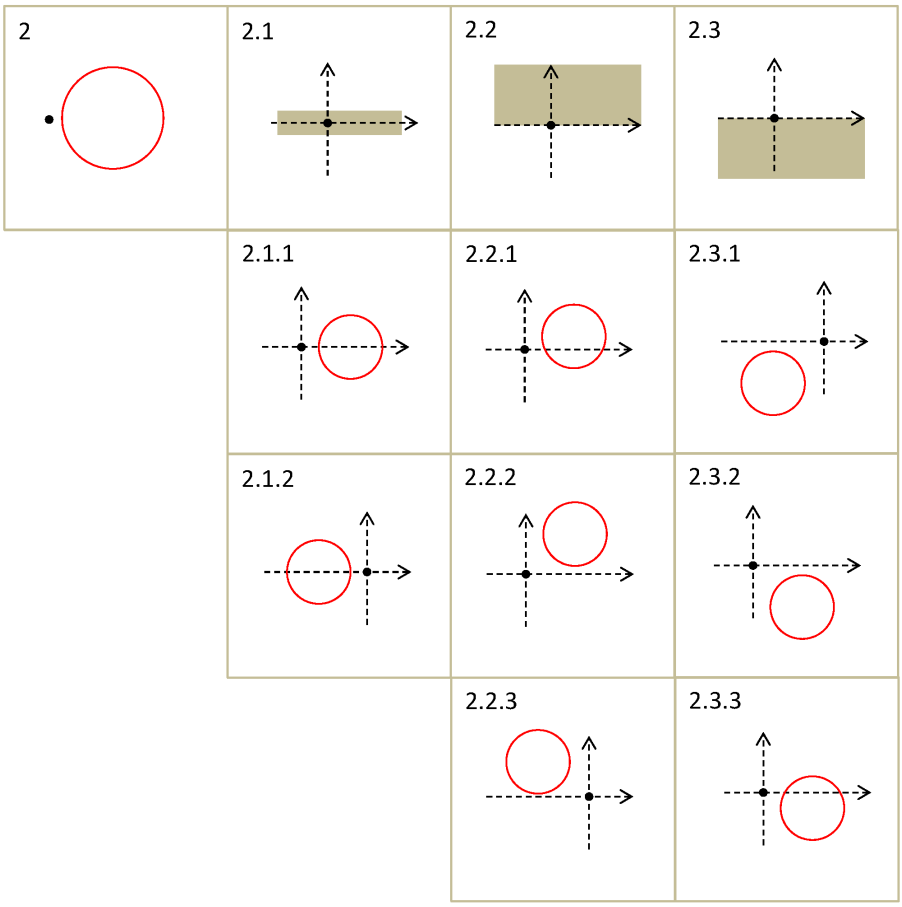


Figure H-3 Specific Classification of Class Two

1) Sub-class 2.1.1: the channel center locates on the positive x-axis;
2) Sub-class 2.1.2: the channel center locates on the negative x-axis;
3) Sub-class 2.2.1: the channel intersects the positive x-axis;
4) Sub-class 2.2.2: the channel locates at the first quadrant and doesn't intersect the positive x-axis;
5) Sub-class 2.2.3: the channel center locates at the second quadrant;
6) Sub-class 2.3.1: the channel center locates at the third quadrant;
7) Sub-class 2.3.2: the channel locates at the fourth quadrant and doesn't intersect the positive x-axis;
8) Sub-class 2.3.3: the channel intersects the positive x-axis.

Gather sub-classes 2.1.1 to 2.3.3 together and label it as class 2.123. Now, consider the azimuthal sectors. The total 2.123 can be clarified into four types: 2.123/1, 2.123/2, 2.123/3 and 2.123/4 (Figure H-4). Then consider the radial sectors, each of the four types can be further classified into three types (Figure H-4). The green arrows represent the determinate TRACE azimuthal sectors. The dash green arrows represent the indeterminate TRACE azimuthal sectors. A full explanation of the cases is summarized after the figure. Compared with class two, the classifications of class one (Figure H-2) don't include the azimuthal sectors. This is because the channel must intersect all of the azimuthal sectors since it covers the center point.

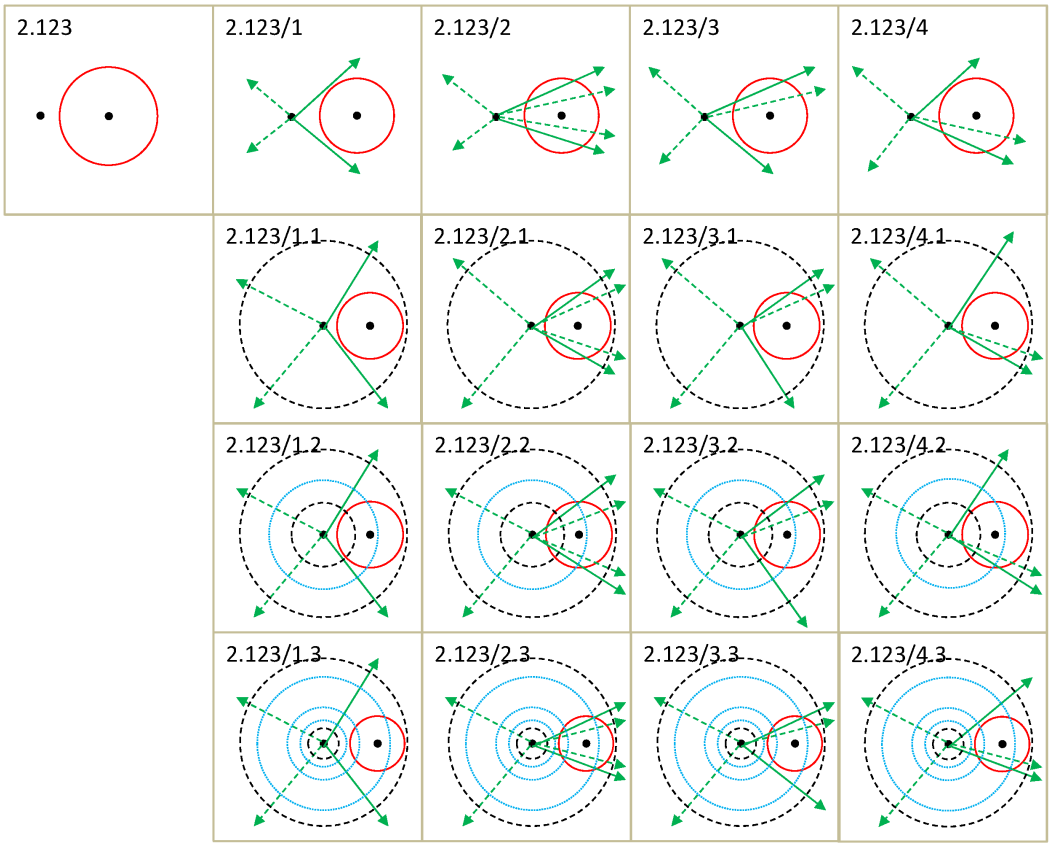


Figure H-4 Full Classifications of Class Two

1) 2.123/1: the channel is not intersected by any azimuthal sectors;
2) 2.123/2: both the upper and lower parts of the channel are intersected by the sectors;
3) 2.123/3: only the upper part of the channel is intersected by the sectors;
4) 2.123/4: only the lower part of the channel is intersected by the sectors.

The three further classifications of all the four types of 2.123/1. 2.123/2, 2.123/3 and 2.123/4 are more or less the same. Thus their explanation can be put together.

1) 2.123/1.1, 2.1, 3.1, 4.1: the channels locate fully in the first ring;
2) 2.123/1.2, 2.2, 3.2, 4.2: the channel intersects with more than one rings including the first ring;
3) 2.123/1.3, 2.3, 3.3, 4.3: the channel separates from the first ring and may intersect or don't intersect rings.

Based on the recognized channel type, the subroutine will then calculate the overlapping areas between SCF channels and TRACE sections and finally generate two contribution arrays storing the mapping proportions. To fully test the subroutine, cases should be designed to cover all possibilities. Figure H-5 lists the test cases for class one channels. The instruction follows the figure.

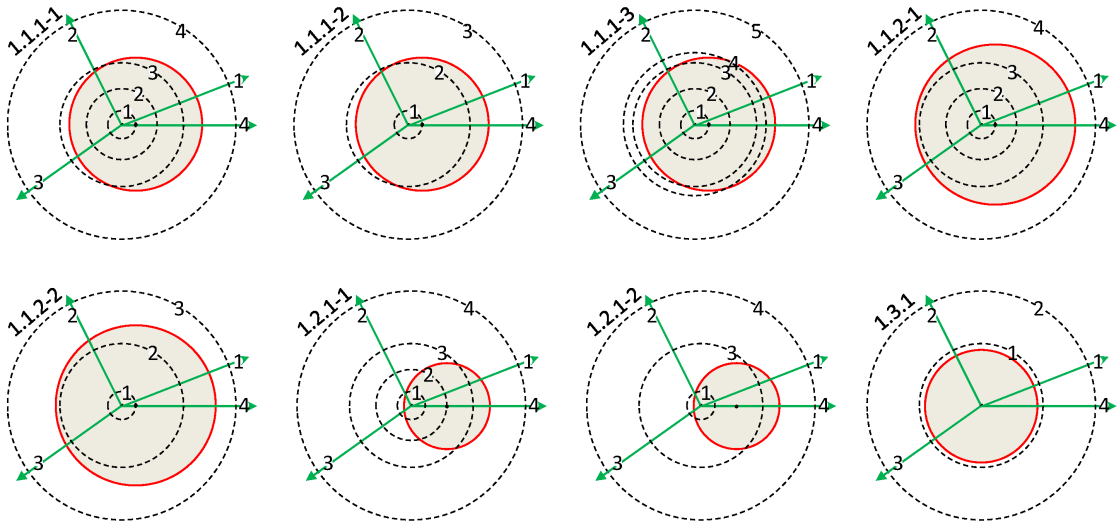


Figure H-5 Test Cases for Channels Belonging to Class One

The introduction of test cases should be combined with the classifications in Figure H-2

1) 1.1: The channel area covers the first vessel radial section.
- **a.** 1.1.1: The channel boundary intersects one of the vessel radial circles, either the innermost one of the outermost one.
  - **i.** 1.1.1-1: More than one vessel radial rings are covered by the channel.
  - **ii.** 1.1.1-2: Only one vessel radial ring is covered by the channel and only one ring intersects the channel.
  - **iii.** 1.1.1-3: More than one vessel rings are covered by the channel and intersect the channel.
- **b.** 1.1.2: There is no intersection between the channel and vessel radial circles.
  - **i.** 1.1.2-1: More than three-vessel rings are covered by the channel.
  - **ii.** 1.1.2-2: Only two-vessel rings are covered by the channel.

2) 1.2: The channel area intersects the first vessel radial ring.
- **a.** 1.2.1: The channel area intersects the first vessel radial ring.
  - **i.** 1.2.1-1: More than two-vessel radial rings intersect the channel.
  - **ii.** 1.2.1-2: Only two-vessel rings intersect the channel.

3) 1.3: The vessel's first radial circle covers the channel area.
- **a.** 1.3.1: There is only one condition.

Test cases for class two include more items (Figure H-6). According to the position of the channels' center, they can be divided into eight types. Since each test case has two channels which are central symmetrical to each other. Four types of cases are enough to cover all the possibilities. Since the 360-degree line is always a boundary, cases that have centers in quadrant 1 and 4 must be divided into two sub-cases respectively according to whether they intersect the axis x. If we take the azimuthal sectors into consideration, most of the eight cases above will be re-divided into nine types. Since the radial sections processing module is almost isolated from the module which processes the azimuthal sections (this is the fact for almost all the channel types except 2.123/1.1, 2.123/1.2 and 2.123/1.3), it is not necessary to test all the nine conditions. That is to say, an additional model to test the azimuthal section processing module will not be needed if the previous model has already tested such kind functions.

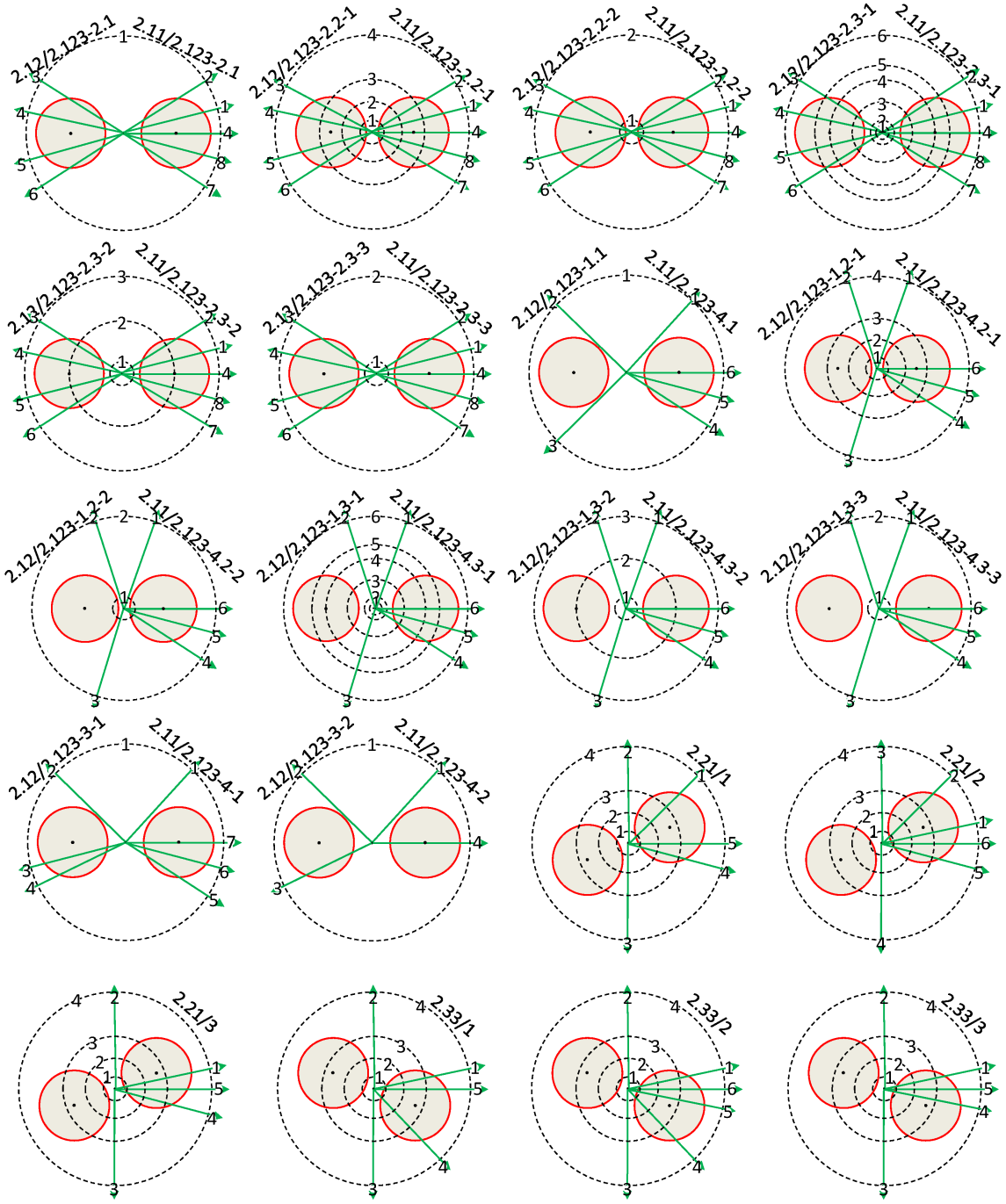


Figure H-6 Test Cases for Channels Belonging to Class Two

1) 2.1.1 – 2.1.2: These cases include two channels - channels 2.1.1 and 2.1.2.

a. 2.123 – 2 – Domi: These cases focus on condition 2.123/2 of channel 2.1.1 and channel 2.2.2. The majority functions of the radial and azimuthal processing module can be sufficiently examined.

i. 2.1: Corresponds to 2.123/2.1, the channel locates fully in the first vessel circle and is intersected by azimuthal boundaries both on its upper and lower parts.
ii. 2.2: Corresponds to 2.123/2.2, the channel intersects the first vessel circle and is intersected by azimuthal boundaries both on its upper and lower parts.
    - 1: The channel intersects more than one vessel circles.
    - 2: The channel intersects only the first circle.
iii. 2.3: Corresponds to 2.123/2.3, the channel is separated from the first vessel circle and is intersected by azimuthal boundaries both on its upper and lower parts.
    - 1: More than one vessel inner circles are separated from the channel and more than one circle intersects the channel.
    - 2: Only the first vessel inner circle is separated from the channel and only the second circle intersects the channel.
    - 3: Only the first vessel inner circle is separated from the channel and no vessel circles intersect the channel.

b. 2.123 – 4_1_3: With the majority, functions were tested by the models in 2.123-2-Domi, several remaining functions should be tested using channel conditions 2,123/1, 2,123/3 and 2,123/4.

2) 2.2.1 – 2.3.1: These cases include two channels - channels 2.2.1 and 2.3.1. There are three conditions to be tested as a supplement to the testing models above. Channel 2.2.1 is intersected by azimuthal boundaries on the lower part while channel 2.3.1 is not intersected by any azimuthal boundaries.
- **a.** 1: There is no first quadrant azimuthal boundaries in the lower part of the channel area.
- **b.** 2: There are first quadrant azimuthal boundaries in the lower part of the channel area.
- **c.** 3: There are only two fourth quadrant azimuthal boundaries in the lower part of the channel area.

3) 2.2.2 – 2.3.1: The two types of the channel have already been tested above.

4) 2.3.3 – 2.2.3: These cases include two channels - channels 2.3.3 and 2.2.3. There are three conditions to be tested as a supplement to the testing models above. Channel 2.3.3 is intersected by azimuthal boundaries on the upper part while channel 2.3.1 is not intersected by any azimuthal boundaries.
- **a.** 1: There is no fourth quadrant azimuthal boundaries in the upper part of the channel area.
- **b.** 2: There are fourth quadrant azimuthal boundaries in the upper part of the channel area.
- **c.** 3: There is only one first quadrant azimuthal boundaries in the upper part of the channel area.

A hiding trap in the testing process is that the uppermost and the lowermost parts of the channel may belong to the same TRACE section, which must be tested by some specially designed cases. Four additional cases were applied to test the special conditions which have a special relationship with 360 degrees azimuthal boundary (Figure H-7).

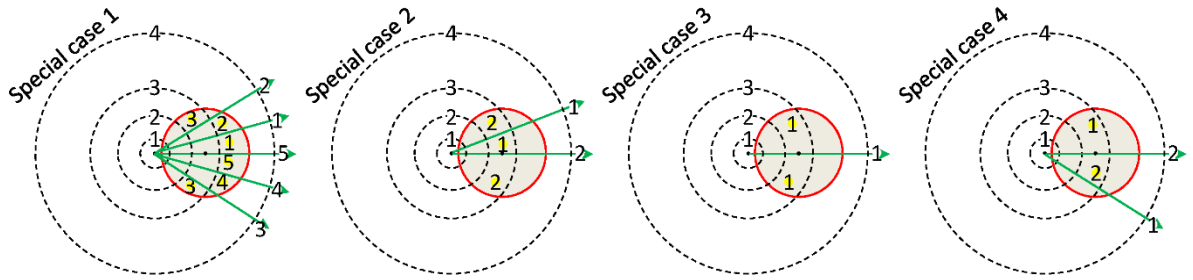


Figure H-7 Testing Cases for Special Conditions

1) 1: Both the lower and upper parts of the channel area are intersected by more than one vessel azimuthal boundaries.
2) 2: Both the lower and upper parts of the channel area are only intersected by one vessel azimuthal boundary.
3) 3: The vessel core only has one section and the channel is intersected by the boundary.
4) 4: The lower part of the channel is crossed by two azimuthal boundaries while the upper part has no such boundaries crossed.

The subroutine was sufficiently tested by the testing cases (Figure H-5, Figure H-6, and Figure H-7) and was proofed good robustness. The mapping between sections of the TRACE model and channels of SCF model for real Nuclear Power Plant (NPP) may not make full use of all the subroutine's potential capabilities, which is determined by the normally regular uniform sections and channels. Nevertheless, if some unusual and strange configurations are to be implemented, the subroutine is still available. It is a tool processing a quite flexible geometry coupling between TRACE and SCF anyway and may also be applied to other codes' coupling.

NRC FORM 335
(12-2010)
NRCMD 3.7

U.S. NUCLEAR REGULATORY COMMISSION

BIBLIOGRAPHIC DATA SHEET

(See instructions on the reverse)

1. REPORT NUMBER
(Assigned by NRC, Add Vol., Supp., Rev., and Addendum Numbers, if any.)

NUREG/IA-0541

2. TITLE AND SUBTITLE

Multi-scale Coupling of TRACE and SUBCHANFLOW with the Exterior Communication Interface (ECI)

3. DATE REPORT PUBLISHED

MONTH	YEAR
November	**2023**

4. FIN OR GRANT NUMBER

5. AUTHOR(S)

Kanglong Zhang; Victor Hugo Sanchez-Espinoza

6. TYPE OF REPORT

Technical

7. PERIOD COVERED (Inclusive Dates)

8. PERFORMING ORGANIZATION - NAME AND ADDRESS (If NRC, provide Division, Office or Region, U. S. Nuclear Regulatory Commission, and mailing address; if contractor, provide name and mailing address.)

Institute for Neutron Physics and Reactor Technology (INR), Karlsruhe Institute of Technology (KIT)
Hermann-von-Helmholtz-Platz 1
Eggenstein-Leopoldshafen, Baden-Württemberg, 76344, Germany

9. SPONSORING ORGANIZATION - NAME AND ADDRESS (If NRC, type "Same as above", if contractor, provide NRC Division, Office or Region, U. S. Nuclear Regulatory Commission, and mailing address.)

Division of Systems Analysis, Office of Nuclear Regulatory Research

U.S. Nuclear Regulatory Commission, Washington DC 20555-0001

10. SUPPLEMENTARY NOTES

K.Tien, NRC Project Manager

11. ABSTRACT (200 words or less)

This report describes the multi-scale coupling of the system code - TRACE and the sub-channel thermal-hydraulic code – SUBCHANFLOW (SCF) for a better description of the multi-dimensional thermal-hydraulic phenomena inside the Reactor Pressure Vessel (RPV) of a Pressurised Water Reactor (PWR).

In the recent versions of TRACE, the ECI was activated for multi-tasking and coupling of TRACE with different solvers e.g., CFD, sub-channel codes. To couple TRACE with the Karlsruhe Institute of Technology (KIT) in-house sub-channel code – SUBCHANLFOW (SCF), a specific ECI-module for SUBCHANFLOW was developed. In this report, the implemented spatial mapping of the involved thermal-hydraulic domains and the time synchronization of the involved solvers are described for both stationary and transient simulation. A domain decomposition approach and a weighted field-mapping method were adopted for this purpose.

Besides, an explicit operator splitting method is implemented for the data transfers during the time advancement of both codes either in stationary or transient simulations.

The prediction capability of the coupled code is demonstrated by the analysis of two academic coolant-mixing cases and a VVER-1000 coolant-mixing benchmark. The results obtained by TRACE standalone and by the coupled system TRACE/SCF were compared together and it shows that the coupling system of TRACE/SCF could better predict the coolant mixing along the core than TRACE-standalone.

12. KEY WORDS/DESCRIPTORS (List words or phrases that will assist researchers in locating the report.)

Multi-scale Coupling, TRACE, Subchanfow;.ECI

13. AVAILABILITY STATEMENT

unlimited

14. SECURITY CLASSIFICATION

(This Page)

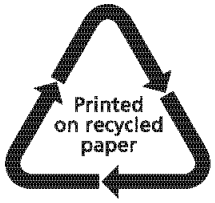
unclassified

(This Report)

unclassified

15. NUMBER OF PAGES

16. PRICE



Federal Recycling Program

UNITED STATES
NUCLEAR REGULATORY COMMISSION
WASHINGTON, DC 20555-0001

OFFICIAL BUSINESS

NUREG/IA-0541

Multi-scale Coupling of TRACE and SUBCHANFLOW with the Exterior Communication Interface (ECI)

November 2023