UNIVERSIDADE FEDERAL DE MINAS GERAIS
ESCOLA DE ENGENHARIA
DEPARTAMENTO DE ENGENHARIA NUCLEAR
PÓS-GRADUAÇÃO EM CIÊNCIAS E TÉCNICAS NUCLEARES

Vitor Vasconcelos Araújo Silva

Acoplamento neutrônico e termo-hidráulico usando os códigos milonga e OpenFOAM: uma abordagem com *software* livre

Belo Horizonte, MG

Dezembro de 2016

Vitor Vasconcelos Araújo Silva

Acoplamento neutrônico e termo-hidráulico usando os códigos milonga e OpenFOAM: uma abordagem com *software* livre

Tese apresentada ao Programa de Pós-Graduação em Ciências e Técnicas Nucleares do Departamento de Engenharia Nuclear da Universidade Federal de Minas Gerais, como requisito parcial à obtenção do título de Doutor em Ciências e Técnicas Nucleares.

Área de concentração: Engenharia Nuclear e da Energia.

Orientador: Cláubia Pereira Bezerra Lima

Coorientador: André Augusto Campagnole dos Santos

Belo Horizonte, MG

Dezembro de 2016

Silva, Vitor Vasconcelos Araújo.
S586a Acoplamento neutrônico e termo-hidráulico usando os códigos milonga e OpenFOAM [manuscrito] : uma abordagem com software livre / Vitor Vasconcelos Araújo Silva. – 2016.
109 f., enc.: il.

Orientadora: Cláubia Pereira Bezerra Lima.
Coorientador: André Augusto Campagnole dos Santos.

Tese (doutorado) Universidade Federal de Minas Gerais, Escola de Engenharia.

Inclui anexos.

Bibliografia: f. 95-102.

1. Engenharia nuclear - Teses. 2. Reatores nucleares - Teses. 3. Método dos volumes finitos - Teses. 4. Software livre - Teses. I. Lima, Cláubia Pereira Bezerra.II. Santos, André Augusto Campagnole dos. III. Universidade Federal de Minas Gerais. Escola de Engenharia. IV. Título.

CDU: 621.039(043)

UNIVERSIDADE FEDERAL DE MINAS GERAIS

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIAS E TÉCNICAS NUCLEARES

PCTN

FOLHA DE APROVAÇÃO

Acoplamento neutrônico e termo-hidráulico usando os códigos milonga e OpenFOAM: uma abordagem com software livre

VÍTOR VASCONCELOS ARAÚJO SILVA

Tese submetida à Banca Examinadora designada pelo Colegiado do Programa de Pós-Graduação em CIÊNCIAS E TÉCNICAS NUCLEARES, como requisito parcial para obtenção do grau de Doutor em CIÊNCIAS E TÉCNICAS NUCLEARES, área de concentração ENGENHARIA NUCLEAR E DA ENERGIA.

Aprovada em 19 de dezembro de 2016, pela banca constituída pelos membros:

Profa. Cláubia Pereira Bezerra Lima - Orientadora
Departamento de Engenharia Nuclear - UFMG

Profa. Antonella Lombardi Costa
Departamento de Engenharia Nuclear - UFMG

Dra. Patrícia Amélia de Lima Reis
Departamento de Engenharia Nuclear - UFMG

Dr. Hugo Cesar Rezende
CDTN/CNEN

Dr. Marcelo Antônio Veloso
CDTN/CNEN

Belo Horizonte, 19 de dezembro de 2016.

À Dani, por ser e estar.

Agradecimentos

É este o momento em que as injustiças são inevitavelmente cometidas.

Aos vários que fizeram parte deste trabalho e que não citarei nominalmente, por impossibilidade ou por falha da memória, minhas desculpas e meu obrigado.

Agradeço à minha orientadora Cláubia, que ouso chamar de amiga, pela dedicação extrema, pelas oportunidades proporcionados, pela confiança em mim depositada e pela paixão pelo que faz. Saiba que este exemplo seguirá comigo pela minha carreira.

Ao meu coorientador André, colega que se tornou um grande amigo, por me incentivar nos momentos em que tudo parecia perdido (e não foram poucos!), pela paciência em ensinar, pelas tantas vezes em que abriu mão do seu tempo pessoal para buscar soluções comigo. É uma honra trabalhar com você, é um prazer ser seu amigo.

Ao amigo improvável Germán, cujo trabalho abnegado permitiu que eu chegasse ao fim desta tese. *Muchas gracias.*

Aos professores, funcionários e colegas do DEN: sem vocês eu não estaria escrevendo estas linhas. Muito obrigado.

Aos colegas do SETRE e do CDTN, em especial Daniel, Marcelo, Carlinhos, Seu Edison, Hugo, Chico, Luiz Cláudio Meira Belo, Marcos e Leo, obrigado pelo incentivo, apoio e pelos valiosos ensinamentos.

Aos alunos do laboratório, obrigado pela saudável convivência e por me ajudarem a envelhecer sem perder a juventude.

Aos colegas da Universidade Politécnica de Valência, em especial Rafa, Gumer, Mónica e Álvaro, obrigado pela acolhida que me fez me sentir em casa em Valência. Foi um enorme prazer trabalhar com vocês.

Agradeço à CAPES pela bolsa de doutorado sanduíche.

Agradeço à minha família e amigos, pela compreensão das minhas ausências em tantas ocasiões.

Agradeço à Leia e à feijoada, pois às vezes comer é muito mais do que se alimentar.

Agradeço à Dani, meu amor, por me dar a mão nos infinitos momentos de dúvidas, tristezas, frustrações, e impaciência. Seu sorriso me faz querer seguir em frente.

"A work is never complete except by some accident such as
weariness, satisfaction, the need to deliver, or death:
for, in relation to who or what is making it,
it can only be one stage in a series of inner transformations.
(Paul Valéry)

Resumo

Nesta tese descreve-se o desenvolvimento de um sistema de cálculos neutrônicos e termo-hidráulicos acoplados que utiliza malhas finas não-estruturadas como domínio de solução e totalmente baseado em *software* livre. As contribuições propostas seguem em dois diferentes eixos: uma delas é o foco na utilização da forma de desenvolvimento de sistemas de computação abertos, um conceito amplamente utilizado em diversas áreas do conhecimento mas raramente explorado no campo de Engenharia Nuclear; a segunda é o uso de memória compartilhada do sistema operacional como uma área de armazenamento de dados de confiável e rápido acesso para acoplar o sistema de dinâmica dos fluidos computacional (CFD) *OpenFOAM* e o gratuito e flexível código de análise de reatores *milonga*. Este conceito foi aplicado na simulação do comportamento de um modelo simplificado de um combustível do reator TRIGA Mark 1 IPR-R1 em estado estacionário. As seções de choque macroscópicas utilizadas pelo modelo, um conjunto para dois grupos de energia, foram geradas utilizando o código WIMSD-5B. Os resultados mostram que esta inovadora forma de acoplamento neutrônico e termo-hidráulico leva a resultados consistentes, encorajando o futuro desenvolvimento deste sistema e seu uso na simulação de sistemas nucleares complexos.

Palavras-chave: neutrônica. termo-hidráulica. acoplamento. *OpenFOAM*. *milonga*. volumes finitos. *free software*.

Abstract

The development of a fine mesh coupled neutronics/thermal-hydraulics framework mainly using open source software is presented. The contributions proposed go in two different directions: one, is the focus on the open software development, a concept widely spread in many fields of knowledge but rarely explored in the nuclear engineering field; the second, is the use of operating system shared memory as a fast and reliable storage area to couple the computational fluid dynamics (CFD) software *OpenFOAM* to the free and flexible reactor core analysis code *milonga*. This concept was applied to simulate the behavior of the TRIGA Mark 1 IPR-R1 reactor fuel pin in steady-state mode. The macroscopic cross-sections for the model, a set of two-group cross-sections data, were generated using WIMSD-5B code. The results show that this innovative coupled system gives consistent results, encouraging system further development and its use for complex nuclear systems.

Keywords: neutronics. thermal-hydraulics. coupling. OpenFOAM. milonga. finite volumes. free software.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

CDTN	Centro de Desenvolvimento da Tecnologia Nuclear
CFD	*Computational Fluid Dynamics*
CNEN	Comissão Nacional de Energia Nuclear
FVM	*Finite Volumes Method*
GNU	*GNU's Not Unix*
GPL	*GNU Public License*
ISIRYM	*Instituto de Seguridad Industrial, Radiofísica y Medioambiental*
LWR	*Light Water Reactor*
MIT	*Massachusetts Institute of Technology*
MOC	*Method of Characteristics*
MPI	*Message Passing Interface*
PARCS	*Purdue Advanced Reactor Core Simulation*
PDE	*Partial Differential Equation*
PVM	*Parallel Virtual Machine*
PWR	*Pressurized Water Reactor*
RANS	*Reynolds Averaged Navier-Stokes*
TRIGA	*Training, Research, Isotopes, General Atomic*
UPV	*Universidad Politècnica de Valencia*

Sumário

1 Introdução

1.1 Contextualização

"Over a 20-year period, the (nuclear) industry will move from the application of conventional methods that rely on experimental correlations to using CFD (...)". (BAGLIETTO, 2012, p. 655).

A frase acima é um exemplo de ganho de importância da técnica de dinâmica dos fluidos computacional (CFD) na indústria nuclear nos últimos anos e da perspectiva da sua aplicação nos próximos. Uma das aplicações em que o uso de CFD tem se estendido é no cálculo termo-hidráulico de elementos combustíveis nucleares. Em boa parte dos trabalhos envolvendo CFD aplicado a problemas termo-hidráulicos, usualmente é definida uma potência inicial, fixa ou variável, à qual é submetido o sistema (USTINENKO et al., 2008). Apesar de extremamente útil, essa forma de prescrever a potência é uma forma de simplificação. Para obter uma potência o mais realista possível, é necessário ter em conta um fenômeno particular de um sistema nuclear: a cinética dos nêutrons, ou neutrônica.

O aumento do poder computacional, que tem permitido a ampliação do uso de CFD na indústria nuclear, chamou a atenção para um aspecto, até então pouco explorado dos cálculos termo-hidráulicos: a possibilidade de realizar cálculos termo-hidráulicos de forma acoplada com os cálculos neutrônicos (estes últimos, fundamentais para uma variedade de aspectos do funcionamento de reatores e combustíveis nucleares). Os cálculos neutrônicos e termo-hidráulicos acoplados, são também chamados cálculos multi-física ou, simplesmente, multi-física (LEPPÄNEN; VIITANEN, 2012).

Há diversas formas de se acoplar sistemas de cálculo neutrônico e termo-hidráulico. Entretanto, é necessário que tais sistemas ofereçam formas de escrever seus resultados e, principalmente, de ler resultados de outros sistemas. Caso contrário, resta somente a possibilidade de modificar tais sistemas de modo que sejam capazes de alguma forma de comunicação com o exterior. Nestes casos, a disponibilidade do código-fonte é condição básica para o desenvolvimento do sistema acoplado. Simplificadamente, o acoplamento dos dois códigos nada mais é do que fazer com que dois códigos sejam capazes de se comunicar e trocar dados para realizar seus cálculos.

1.2 Objetivo

O objetivo deste trabalho de tese é o desenvolvimento de um sistema acoplado de cálculos termo-hidráulicos e neutrônicos aberto e gratuito baseado em CFD e volumes

finitos utilizando o mesmo domínio de solução. Este sistema é construído utilizando-se de dois códigos independentes, também abertos e gratuitos, para os cálculos termo-hidráulicos e neutrônicos.

Apesar de desenvolvidos separadamente - inclusive em linguagens de programação diferentes - ambos os códigos usados fazem uso do Método de Volumes Finitos (FVM em inglês) (EYMARD et al., 2003) na solução do seu conjunto particular de equações. Cabe ressaltar que, neste primeiro momento, apenas cálculos em estado estacionário são previstos.

Entenda-se como abertos (STALLMAN, 2002), simplificadamente, programas, códigos ou sistemas que oferecem pleno acesso ao seu código-fonte. Essa informação poderia passar despercebida, mas há alguns pontos a considerar antes de se seguir. As licenças que permitem a distribuição de código aberto impõem, geralmente, que o código desenvolvido a partir de outro licenciado desta forma deva permanecer aberto. Apesar do grande debate sobre a segurança dos códigos abertos e suas vantagens e desvantagens (ANDROUTSELLIS-THEOTOKIS et al., 2010), é fato que desde o crescimento no uso do sistema operacional Linux (BRITANNICA., 2014), a disseminação de programas distribuídos de forma aberta é crescente. Não apenas para tarefas básicas, mas predominantemente em serviços de rede, internet, cálculo numérico dentre inúmeras outras aplicações de uso intensivo. Sistemas, códigos ou *frameworks* baseados em códigos abertos são extensivamente usados em aplicações de missão crítica (NORRIS, 2004).

Dado que a utilização de programas de código aberto ainda é relativamente tímida no domínio nuclear - e a expressão relativamente talvez seja inapropriada, pois começam a surgir projetos de instituições renomadas utilizando código aberto (ROMANO; FORGET, 2013; BOYD et al., 2014; HUFF et al., 2016) - esta tese tem também por objetivo iniciar a discussão sobre as possibilidades de sua aplicação neste domínio. Sendo os dois códigos usados para o acoplamento no âmbito desta tese disponibilizados de forma aberta, temos, portanto, um sistema também aberto. Sistema este, facilmente auditável por qualquer interessado, seja ele desenvolvedor, engenheiro, pesquisador ou regulador. Se essa forma de desenvolvimento de sistemas de *software* é adequada para as especificidades e restrições dos sistemas nucleares, não é possível dizer.

1.3 Justificativa

Pode-se dizer que há duas grandes justificativas para a realização deste trabalho. A primeira, de ordem principalmente técnica, é o uso de um código do tipo CFD acoplado para cálculo multi-física. O CFD permite o cálculo detalhado do comportamento do escoamento em um reator, subcanal ou ao redor de um elemento. Tal nível de detalhes permite a investigação de fenômenos físicos não modeláveis de outras formas. Além disso,

o acoplamento entre os fenômenos neutrônico e termo-hidráulico resolve a dependência entre ambos os fenômenos de forma fiel ao que realmente ocorre na prática. Obviamente, há um custo para a obtenção deste nível de detalhamento: a demanda computacional. Felizmente, os atuais processadores e suas características de multiprocessamento permitem a utilização de métodos outrora considerados uma ousadia. Isto posto, é possível afirmar que cálculos multi-física ou acoplados já são uma realidade e não se pode abrir mão de conhecer os novos aspectos técnicos, numéricos e teóricos envolvidos nesta nova forma de investigar a física de reatores.

A segunda justificativa, que pode também ser considerada uma motivação, vai além dos aspectos puramente técnicos. O chamado *software livre* consiste em códigos desenvolvidos dentro de uma filosofia de liberdade e de uso comunitário. Essa ideia, que remonta ao início dos anos 80, advoga que um programa ou *software* deve fornecer o código-fonte aos usuários. Os usuários podem modificá-lo, melhorá-lo e até mesmo vendê-lo, desde que o novo programa seja fornecido também com código aberto. Essa filosofia de desenvolvimento comunitário de *software* chocou o mercado de micro-computação quando surgiu. Cerca de 35 anos depois, não só é uma realidade como grande parte do *software* em uso para certos tipos de aplicações hoje é baseado em *software livre* (ANDROUTSELLIS-THEOTOKIS et al., 2010). Não seria de se esperar outra coisa que esta filosofia de desenvolvimento comunitário de *software* alcançasse a engenharia nuclear. As discussões sobre quão efetiva será a adoção deste modelo para uma área de missão crítica, como boa parte das aplicações em Física de Reatores, é, por si só, assunto para uma tese exclusiva sobre o tema. No presente trabalho, a motivação está na possibilidade de junção de forças entre grupos com diferentes *backgrounds*, estruturas de pesquisa e investimentos com um objetivo comum: o desenvolvimento de sistemas úteis para todo o grupo, revisado por pares - e com isso uma mais ampla rede de detecção de erros de implementação - e possíveis de serem modificados e alterados de acordo com as necessidades.

Com o acoplamento entre dois códigos abertos que se apresenta nesta tese espera-se trazer à luz a discussão sobre o uso de *software livre* na indústria e pesquisa nucleares entre especialistas no Brasil. Afinal, a filosofia do uso de *software livre* já começou nos centros de ponta espalhados pelo planeta (ROMANO; FORGET, 2013; BOYD et al., 2014; HUFF et al., 2016).

2 Revisão Bibliográfica

Este capítulo apresenta um histórico dos principais trabalhos envolvendo o uso de CFD em aplicações nucleares. Além disso, são também apresentados os trabalhos mais relevantes envolvendo acoplamento neutrônico e termo-hidráulico utilizando diferentes técnicas, tanto para o cálculo neutrônico quanto termo-hidráulico.

O crescimento na capacidade de processamento computacional fez com que os pesados cálculos de *CFD* passassem a ser atraentes para a Engenharia Nuclear. Mais do que isso, o uso desses códigos passou a ser razão de preocupação no sentido de garantir a validade dos seus resultados. O relatório da Comissão Regulatória Nuclear (NRC) dos Estados Unidos de 2010 (NOURBAKHSH, 2010, p.69), ao sugerir as melhores práticas e métodos para a atividade de regulação, é bastante claro ao apontar a necessidade de tirar proveito da capacidade oferecida por códigos de *CFD*, fornecendo inclusive sugestões de parceria com a indústria e universidades objetivando desenvolver simulações multidimensionais devidamente acompanhadas de validação e verificação. O conteúdo desse relatório é, por si só, prova de que a utilização de códigos *CFD* é, em definitivo, parte do processo de pesquisa e desenvolvimento de reatores nucleares.

Um exemplo desse uso é a utilização de *CFD* - nesse caso um código proprietário: *ANSYS-CFX* - para modelar o fenômeno de ebulição sub-resfriada para o desenvolvimento de combustíveis nucleares (KREPPER et al., 2007). Devido à capacidade dos códigos *CFD* de simular detalhes do escoamento de acordo com a granularidade com que se divide o domínio de simulação, nesse trabalho esta técnica é utilizada para avaliar o fenômeno de fluxo de calor crítico em um elemento combustível. As informações fornecidas pelo código *CFD* em relação a fenômenos como *cross flow* entre regiões adjacentes e concentração de bolhas (no caso em que se simule duas fases) permitem identificar *hot spots*, ou seja, regiões de especial interesse. Essas simulações permitem avaliar o comportamento de diferentes projetos de grades espaçadoras de elementos combustíveis (NAVARRO; SANTOS, 2011), por exemplo.

Um trabalho em conjunto entre pesquisadores do ISIRYM, UFMG e CDTN foi conduzido com o objetivo de investigar o comportamento do reator TRIGA IPR-R1 por um código do tipo *CFD* (MARTÍNEZ et al., 2012). Nesse caso, não houve tentativa de acoplamento, mas apenas a simulação simplificada do reator utilizando-se o *ANSYS-CFX*. Um fluxo de calor foi fornecido para as paredes do combustível obedecendo à uma distribuição axial característica do combustível. Na análise quantitativa dos resultados, os autores informam que os resultados da simulação numérica apresentam boa concordância com dados experimentais coletados durante a operação do reator.

O uso das técnicas de *CFD* não ficou restrito aos *softwares* ditos proprietários. Hrvoje Jasak discorreu sobre as razões pelas quais um único *software* monolítico não era adequado para a solução dos vários problemas modeláveis em *CFD*, tais como requisitos específicos para um certo problema que poderiam envolver propriedades experimentais, equações adicionais ao problema, ou acoplamento com pacotes externos. O autor ressalta, ainda, que para diferentes problemas, diferentes formas de discretização e soluções numéricas são necessárias. Além das suas críticas às dificuldades de uso e não-flexibilidade dos códigos fechados, propôs uma solução: uma biblioteca orientada a objetos para simulações numéricas escrita em linguagem C++ e, mais do que tudo, licenciada como *software* livre (JASAK et al., 2007) chamada *OpenFOAM*. Nesse trabalho o autor também desfaz as polêmicas sobre o desempenho de códigos escritos em C++ e demonstra como as equações são modeladas dentro do *OpenFOAM*. Um exemplo é a equação da energia cinética turbulenta no modelo RANS[1]:

$$\frac{\delta k}{\delta t} + \nabla.(\mathbf{u}k) - \nabla.[(\nu + \nu_t)\nabla k] = \nu_t \left[\frac{1}{2}(\nabla \mathbf{u} + \nabla \mathbf{u}^T)\right]^2 - \frac{\epsilon_0}{k_0}k \tag{2.1}$$

que é representada como código no *OpenFOAM* do seguinte modo:

```
1 solve
2 (
3    fvm::ddt(k)
4  + fvm::div(phi, k)
5  - fvm::laplacian(nu() + nut, k)
6 == nut*magSqr(symm(fvc::grad(U)))
7  - fvm::Sp(epsilon/k, k)
8 );
```

Este exemplo ajuda a ilustrar a relativa simplicidade - de acordo com o autor, mesmo alguém com conhecimento limitado em programação e sem referência à programação orientada a objetos em C++ é capaz de entender por alto o trecho apresentado - da modelagem de equações diferenciais no *OpenFOAM*.

Vale ressaltar que, devido ao custo computacional dos cálculos *CFD*, seu uso na indústria nuclear é, se comparado a outros métodos de simulação termo-hidráulica, bastante recente. Um artigo que veio de forma definitiva atestar a importância dos cálculos *CFD* na indústria nuclear foi apresentado por Emilio Baglietto (BAGLIETTO, 2012). Nesse artigo, o autor afirma que o uso de *CFD*, só ou combinado com códigos neutrônicos e ferramentas de análise, aumentou a capacidade e disponibilidade dos reatores nucleares atualmente em operação. Ele afirma, ainda, que tal tecnologia de análise está ajudando no desenvolvimento dos reatores de quarta geração, especialmente nesse caso, em que dados experimentais e operacionais estão menos disponíveis. Vendedores e operadores de reatores utilizarão muito mais modelagem, simulação e computação para determinar como reatores e seus componentes irão se portar quando em operação. O autor é categórico

[1] Os símbolos são apresentados com o único objetivo de descrição em relação à notação utilizado pelo *OpenFOAM*. O significado dos símbolos presentes na equação 2.1 é dado em (JASAK et al., 2007)

ao afirmar que “em um período de 20 anos, a indústria passará dos métodos convencionais baseados em correlações experimentais ao uso de *CFD* para pequenos componentes, grandes componentes e análise de uma planta completa”.

Entretanto, para obter resultados os mais realistas possível, é necessário simular tanto os fenômenos termo-hidráulicos quanto neutrônicos. Isso se deve ao fato de que os principais fatores que influenciam na distribuição do fluxo de nêutrons no núcleo de um reator nuclear são as condições do moderador e refrigerante. No caso de um reator do tipo PWR, embora grafite também seja usado, a água exerce ambos os papéis. Pequenas variações na temperatura, densidade e composição da água podem mudar consideravelmente o fator de multiplicação de nêutrons (k_{eff}). Daí a importância e a necessidade de acoplar os cálculos termo-hidráulicos - as variações no escoamento - com as variações no fluxo neutrônico.

Um dos primeiros trabalhos diretamente focados no acoplamento neutrônico e termo-hidráulico foi apresentado por (BARBER; DOWNAR, 1998). Esse trabalho é essencialmente um relatório técnico, no qual o foco está na construção de uma interface genérica de acoplamento do código PARCS com códigos de termo-hidráulica. Apesar de não gerar resultados de simulações, esse documento evidencia a importância e o interesse no procedimento de acoplamento, além de fornecer preciosas informações acerca da implementação de estruturas de dados, manipulação de entrada e saída, troca de mensagens, uso de biblioteca de troca de mensagens PVM (GEIST et al., 1994), dentre outros pontos cruciais da implementação de software. Cabe ressaltar que esta interface, chamada *General Interface* é ainda usada pelo código PARCS nas suas versões mais recentes.

O acoplamento das simulações de diferentes fenômenos físicos tem sido também chamado de **multi-física** (LEPPÄNEN; VIITANEN, 2012). Uma definição prática para multi-física ou física-acoplada é dada por Paul Lethbridge (LETHBRIDGE, 2005), definindo que a multi-física é, em essência, a análise de fenômenos físicos distintos de forma combinada.

No texto desta tese, por conveniência, a palavra acoplamento será usada tanto em referência ao processo de análise dos fenômenos físicos conjuntamente quanto em referência à implementação do software para essa tarefa. Nos casos em que possa haver ambiguidade, a definição será explicada de modo a não deixar margens a dúvidas.

Vale ressaltar, em referência à implementação do acoplamento em software, que uma aplicação multi-física pode levar entre quatro e seis anos para ser efetivamente útil, podendo chegar a uma vida útil de várias décadas (GRAHAM et al., 2004). Isto posto, é possível concluir que o esforço em construir o acoplamento é compensado pela possibilidade de uso do código durante vários anos.

Ainda sobre o crescimento da importância dos cálculos neutrônicos e termo-hidráulicos

acoplados, foi iniciado em 2012 no Centro de Pesquisas Técnicas VTT (Finlândia) o projeto *Numerical Multi-Physics (NUMPS)* (LEPPÄNEN et al., 2015). Dentre seus objetivos, estão o desenvolvimento do código de física de reatores *Serpent*, que utiliza o método de Monte Carlo e do código PORFLO, do tipo *CFD*. Esse desenvolvimento oferece oportunidades para a educação das novas gerações de especialistas, não apenas com compreensão de teoria e métodos, mas também com entendimento ao nível de código-fonte das ferramentas de cálculo. Com um viés diferente, o consórcio para a simulação avançada de reatores a água leve (CASL), criado em 2010, tem como objetivo prover capacidade de modelagem e simulação para apoiar e acelerar a melhora na competitividade econômica e redução do volume de combustível gasto por unidade de energia. Alguns de seus resultados são sistemas multi-física de alto desempenho com o propósito de atacar problemas antigos da indústria nuclear, tais como predição de comportamento de desgaste de revestimento de combustíveis e estudo da química da corrosão em várias escalas (TURINSKY; KOTHE, 2016).

Talvez o mais importante trabalho especificamente dedicado ao acoplamento neutrônico e termo-hidráulico seja o de Kostadin Ivanov (IVANOV; AVRAMOVA, 2007). Nele são analisados e classificados diferentes tipos de acoplamento, seus componentes e suas aplicações. Os diversos parâmetros do acoplamento são esquematicamente divididos em:

1. **Forma de acoplamento**: externo, quando o código neutrônico é combinado com parte do código termo-hidráulico, geralmente na forma de condições de contorno, e então acoplado ao sistema termo-hidráulico completo. O acoplamento é definido como interno quando a neutrônica é integrada ao modelo de transferência de calor do sistema termo-hidráulico. Deve-se dizer que não há, na literatura, uma concordância completa sobre a nomenclatura para a forma de acoplamento.

2. **Abordagens de acoplamento**: integração em série ou processamento em paralelo. Na integração em série o algoritmo para cálculo neutrônico é integrado como uma rotina ao sistema termo-hidráulico e então executado sequencialmente após os cálculos termo-hidráulicos. Na abordagem de processamento em paralelo, geralmente a troca de dados é intermediada por um sistema de troca de mensagens como PVM (GEIST et al., 1994) ou MPI (QUINN, 2004). Nesse caso, os módulos de neutrônica e termo-hidráulica têm capacidade separada de execução e resposta, o que permite uma execução mais eficiente do ponto de vista computacional.

3. **Sobreposição espacial de malhas**: pode ser fixo, quando um canal termo-hidráulico representa um canal neutrônico, ou flexível, quando são usados ou especificados esquemas de mapeamento. Um esquema simples de mapeamento foi proposto pelo autor desta tese (SILVA et al., 2015) apenas para malhas estruturadas. Esquemas

mais avançados de mapeamento e interpolação de malhas (BEAUDOIN; JASAK, 2008) são implementados em alguns dos modernos códigos de *CFD*. Há, na literatura, algumas soluções elegantes para o mapeamento entre malhas em problemas de acoplamento neutrônico e termo-hidráulico (JARETEG et al., 2015; RICHARD et al., 2015; SCHMIDT et al., 2015). O uso de esquemas de mapeamento de malhas trazem implicações, tais como arredondamento e truncamentos no pareamento entre células, além de custo computacional relativo a percorrer as malhas ou armazenamento de estruturas de mapeamento em memória.

4. **Algoritmos de controle de *time-step***: os transientes neutrônicos são geralmente muito mais rápidos que os transientes termo-hidráulicos. A utilização de um único *time-step* longo pode levar à não detecção de transientes rápidos e no caso oposto ao desperdício de recursos computacionais ao se simular eventos indistinguíveis repetidamente.

5. **Acoplamento numérico**: o autor se refere nesta classificação ao tempo de troca de informações entre os modelos neutrônico e termo-hidráulico. Pode ser explícito, semi-explícito e implícito, dependendo da forma como cada esquema realiza o cálculo das variáveis do *time-step* atual baseado em variáveis do *time-step* atual ou anterior.

6. **Esquemas de convergência do acoplamento**: definidos de acordo com a forma com que a simulação é considerada finalizada. Se há apenas uma estimativa para a convergência de ambos os códigos, o esquema é considerado fracamento acoplado, por exemplo.

Em relação ao último item, poucos autores se dedicam a uma análise sistemática da forma como se dá a convergência nos cálculos acoplados. Usualmente são apenas apresentados os esquemas de convergência e o foco das análises se concentra nos resultados dos cálculos acoplados. Zerkak (ZERKAK et al., 2015), por sua vez, faz uma revisão dos métodos de acoplamento com foco em como melhorar a convergência das atuais técnicas de acoplamento. O autor faz uma análise pormenorizada da técnica de *operator splitting* e várias de suas melhorias, além de analisar vários outros esquemas numéricos de solução de *PDE's*. Sua análise visa uma avaliação de vantagens e desvantagens dos métodos numéricos em relação à eficiência computacional, convergência, esforço de implementação e modularidade, sempre com vistas à análise de reatores nucleares.

O autor ainda comenta sobre a importância da geração de seções de choque adequadas aos transientes esperados na simulação acoplada e sua interdependência. Na presente tese foram geradas seções de choque para a obtenção dos coeficientes para a solução da equação de difusão para dois grupos de nêutrons. Uma breve explicação de como se-

ções de choque para poucos grupos podem ser geradas[2] pode ser encontrada no artigo de Friedman (FRIEDMAN, 2013). Cabe ressaltar que um grande número de trabalhos encontrados na literatura tem utilizado repetidamente o código *Serpent* (LEPPÄNEN, 2013) para a geração de seções de choque, tanto para cálculos estocásticos quanto determinísticos (JARETEG et al., 2014).

Dorval (DORVAL; LEPPÄNEN, 2015) investiga a geração de seções de choque para poucos grupos a partir de bibliotecas ACE utilizando o código *Serpent* para o cálculo neutrônico de reatores rápidos refrigerados a sódio. Além dos coeficientes de difusão padrão gerados pelo *Serpent*, é proposto um novo método para cálculo de coeficientes de difusão direcionais.

Confirmando o atual interesse em cálculos *CFD* no domínio nuclear, um trabalho de simulação envolvendo neutrônica e termo-hidráulica com uso de *CFD* propõe a simulação de reatores avançados refrigerados a gás (HOSSAIN, 2011). O modelo utilizado para a simulação neutrônica foi o de cinética pontual devido à sua simplicidade.

No modelo de cinética pontual, resolve-se o sistema linear para a reatividade considerando que não há variações nas propriedades neutrônicas e que a reatividade é mantida constante.

A variação no fluxo é dada pelo balanço entre nêutrons produzidos e perdidos, considerando seis classes de precursores para nêutrons atrasados. O desenvolvimento matemático leva a um sistema de sete equações diferenciais ordinárias, conhecidas como equações de cinética pontual. A implementação da solução desse sistema é, nesse caso, feita dentro do código termo-hidráulico utilizado (*TH3D*).

Em outro trabalho que se utilizou de *CFD* (YAN et al., 2011), foi simulado o comportamento do escoamento em um feixe de elementos combustíveis e grades espaçadoras utilizando-se de *CFD* acoplado à neutrônica. A neutrônica simulada é baseada no *method of characteristics (MOC)*, que evita a necessidade de geração de constantes para poucos grupos *a priori*. A abordagem usada no acoplamento é do tipo externa, com o código neutrônico DeCART e o código termo-hidráulico STAR-CCM+ executando sequencialmente e escrevendo e lendo arquivos em disco em um diretório comum. São dois critérios para finalizar a simulação: o código DeCART se baseia na convergência da fonte de fissão, enquanto o código STAR-CCM+ no resíduo da energia. No que toca à discretização espacial (sobreposição espacial de malhas), é usado um mapeamento entre as malhas utilizadas nos dois códigos. Apesar de diferenças nas malhas, as fronteiras materiais, ou seja, geometria, fronteira e posição de diferentes materiais são equivalentes em ambos os modelos, simplificando o esquema de mapeamento. Suas conclusões reiteram a importância da técnica de

[2] A geração de seções de choque mencionada se refere à utilização de dados preprocessados, geralmente em formato ACE MCNP. A geração de seções de choque para múltiplas aplicações a partir de bibliotecas de dados nucleares é um processo complexo e não é escopo desta tese.

CFD na indústria nuclear. Nas palavras do autor, esse acoplamento "permite um melhor entendimento da margem de DNB (*Departure from Nucleate Boiling*)". O autor finaliza apontando o interesse em incrementar a simulação agregando outros aspectos físicos, como um modelo de corrosão, um modelo de geração do particulado, um modelo de interação revestimento/pastilha e outros.

Corroborando com a ideia da importância dos cálculos acoplados, Faghihi faz uma análise dos diversos tipos de códigos usados para cálculos termo-hidráulicos no que se refere à forma como os sistemas são modelados (códigos de sistemas, códigos de sub-canais e códigos do tipo *CFD*) e outra análise semelhante para os códigos neutrônicos (FAGHIHI et al., 2011). Entretanto, seu foco está além da descrição de conceitos de acoplamento neutrônico e termo-hidráulico. O autor propõe o uso de redes neuronais treinadas com dados de outras simulações para a solução do fluxo neutrônico em reatores do tipo LWR.

Um dos raros trabalhos de acoplamento neutrônico e termo-hidráulico que pode ser classificado como acoplamento interno, de acordo com a nomenclatura proposta por Ivanov (IVANOV; AVRAMOVA, 2007), resolve todas as equações de forma totalmente implícita. Nesta abordagem (POPE; MOUSSEAU, 2009), é desenvolvida toda a metodologia matemática e sua implementação correspondente e modelado um reator do tipo placa simplificado. De acordo com os autores, trata-se de um projeto piloto, fundamentalmente conceitual, mas afirmam que os resultados mostram que esta abordagem é promissora.

Talvez o mais completo trabalho relativo ao acoplamento neutrônico e termo-hidráulico com uso do *CFD OpenFOAM* seja a tese de Klas Jareteg (JARETEG, 2012). Nela, o acoplamento é feito usando o mesmo software para a simulação termo-hidráulica e neutrônica. A metodologia utilizada pode ser brevemente resumida em alguns passos principais: 1) criação de uma malha adequada, 2) discretização das equações descrevendo o problema, 3) definição das condições de contorno, 4) geração dos parâmetros neutrônicos (por exemplo, seções de choque macroscópicas) e 5) solução do problema neutrônico e termo-hidráulico de forma acoplada. O autor usa um dos modelos de solução presentes no software de *CFD OpenFOAM* (OPENFOAM..., 2013) e altera esse modelo que simula um escoamento turbulento de um fluido compressível com transferência de energia por radiação térmica (`buoyantSimpleRadiationSolver`) adicionando uma implementação da equação de difusão multi-grupos ao modelo. Isto é feito utilizando a capacidade geral de discretização e solução de equações bem como os algoritmos para solução numérica já presentes no *OpenFOAM*. Dentre suas conclusões, estão que o acoplamento iterativo utilizado é funcional e estável, a solução para a neutrônica necessitou de menos iterações para convergir do que que a solução pressão-velocidade e que a termo-hidráulica precisa de malhas mais refinadas do que a solução neutrônica. A qualidade desse trabalho e muitas das soluções adotadas servem como referências às implementações e simulações a serem

feitas nesta tese.

O referido trabalho de tese deu origem a um artigo (JARETEG et al., 2014) no qual o acoplamento neutrônico e termo-hidráulico é classificado como interno e é utilizada uma malha refinada. O objetivo desse artigo é provar que tal acoplamento é factível e gera resultados com um nível de detalhes sem precedentes. Além disso, os autores apontam que cálculos com malhas menos refinadas geram discrepâncias de até 0,5% na potência por vareta e várias dezenas de *pcm* no fator de multiplicação. Os autores justificam, ainda, que apesar de a neutrônica não necessitar de uma malha tão detalhada quanto necessita a termo-hidráulica, seu uso se justifica pela não necessidade de interpolações ou mapeamentos entres os elementos da malha. Além disso, como trata-se de um acoplamento interno, não há necessidade de transferência de dados, poupando-se tempo de processamento. Nesse trabalho o problema é tratado como de regime permanente, bem como o problema escopo desta tese.

O *OpenFOAM* também foi a ferramenta escolhida por pesquisadores do Paul Sherrer Institut para reduzir os esforços de desenvolvimento ao prover rotinas de discretização e capacidade de execução em paralelo de soluções de equações diferenciais. A partir de *solvers* disponíveis no *OpenFOAM* foi implementado um *solver* multi-física batizado *General Nuclear Foam - GeN-Foam* (FIORINA et al., 2015). Esse *solver* possui a capacidade de resolver escoamentos compressíveis ou incompressíveis baseados no modelo de turbulência $\kappa - \epsilon$, sendo também capaz de resolver problemas em malhas menos refinadas utilizando o modelo de meios porosos. Tem, ainda, implementado um *sub-solver* de termomecânica e outro *sub-solver* para a solução da equação de difusão para multi-grupos. Este último, provê ainda a capacidade de resolver problemas com malhas móveis e tem implementados métodos de leitura de seções de choque geradas pelo código Serpent (LEPPÄNEN, 2013). Os autores classificam o acoplamento implementado como semi-implícito e como são usadas malhas diferentes para os modelos termo-hidráulico, termomecânicos e de difusão de nêutrons, é feito um mapeamento consistente entre as três malhas através do uso de um algoritmo de peso, célula e volume implementado pelo próprio *OpenFOAM* (OPENFOAM..., 2015b). A aplicação inicial desse *solver* foi na simulação do reator rápido a sódio europeu (ESFR). Mesmo com as simplificações no modelo, as respostas numéricas para as simulações de dois transientes testados ficaram próximas das obtidas pelas simulações de referências feitas com o código TRACE.

Uma das razões do uso do acoplamento, cujo objetivo é obter dados mais precisos e realistas, é na análise de segurança de reatores. Em projetos de reatores inovadores já são feitos cálculos de multi-física para análise de acidentes. No trabalho de Lázaro (LAZARO et al., 2013), um reator de IV geração refrigerado a sódio é simulado de forma acoplada. Códigos já usados na análise de reatores resfriados a água leve são adaptados para o uso em reatores de nêutrons rápidos resfriados a sódio. Um ponto-chave, segundo o autor, é

ser capaz de simular os fenômenos presentes na operação de reatores em três dimensões. Isso se justifica, ainda nas palavras do autor, devido a possíveis componentes assimétricos em transientes hipotéticos e que a modelagem unidimensional com cinética pontual não é capaz de reproduzir. Assim como em vários trabalhos sobre o tema de acoplamento, o código usado para a geração de seções de choque homogeneizadas foi o *Serpent* (LEPPÄNEN, 2013). Nesse trabalho, o código *Serpent* foi ainda usado para cálculo neutrônico do núcleo e validado pelo código PARCS (DOWNAR et al., 2006). Sua conclusão nesse trabalho é de que as adaptações feitas nos códigos levaram a resultados consistentes, mas que ainda há trabalho a ser feito para a simulação tridimensional completa desse tipo de reator.

Uma das mais utilizadas ferramentas para estudo de segurança de reatores nucleares é o código de sistemas termo-hidráulicos RELAP5 (RELAP5/MOD3. . . , 2003). Sua utilização para simulação de cenários de transitórios graves e acidentes é ampla, podendo ser considerado a ferramenta *de facto* para análise de acidentes termo-hidráulicos em centrais nucleares. Como consequência de seu alcance em número de usuários, não é de se espantar que esta ferramenta tenha também sido empregada em cálculos acoplados com neutrônica. Um exemplo que merece citação é o trabalho de Reis (REIS et al., 2015), em que o RELAP5 foi utilizado de forma acoplada com o código PARCS para simulação do reator TRIGA IPR-R1, do Centro de Desenvolvimento de Tecnologia Nuclear, reator também utilizado como referência nesta tese. Há, entretanto, um trabalho que utiliza o RELAP5 para cálculos acoplados que deve obrigatoriamente ser mencionado nesta tese. Maciel (MACIEL, 2011) utilizou o código RELAP5 acoplado com com um código local baseado no código DYNETZ para gerar uma biblioteca genérica de acoplamento. Apesar da limitação a uma única plataforma, esta biblioteca é o único trabalho encontrado na literatura - dentro do máximo conhecimento do autor desta tese - que utiliza memória compartilhada, base do desenvolvimento do acoplamento desta tese, para troca de dados entre códigos acoplados. Para isso, foi necessário alterar sensivelmente o código fonte do RELAP5 de modo a permitir que variáveis internas pudessem ser utilizadas de forma acoplada.

O pioneirismo do grupo da universidade de Chalmers, em Gotemburgo - Suécia, encabeçado por Christophe Demazière, além dos já citados trabalhos (JARETEG, 2012; JARETEG et al., 2014), também se refletiu em outras formas de acoplamento. Desta vez, sem o uso do *OpenFOAM*, foi implementada uma ferramenta para estimar flutuações no fluxo neutrônico, temperatura do combustível, densidade do moderador e velocidade do escoamento em um reator do tipo PWR (LARSSON; DEMAZIÈRE, 2012). Este tipo de ferramenta visa captar as flutuações em parâmetros essenciais do funcionamento do reator e, então, encontrar eventuais anomalias. A validação foi feita com dados de um reator PWR comercial e simulações feitas com os códigos RELAP5 e PARCS acoplados bem com o *CFD* comercial *Fluent*. Essas simulações foram feitas para um modelo tridimensional sem

mapeamento, ou seja, ambos os códigos utilizaram malhas idênticas, num acoplamento um para um entre neutrônica e termo-hidráulica.

Ainda na mesma linha da utilização do *OpenFOAM* como plataforma única para neutrônica e termo-hidráulica, Klaus Jareteg estendeu seu trabalho à simulação de elementos combustíveis de reatores do tipo PWR (JARETEG et al., 2015). Neste trabalho, é proposto um mapeamento entre os valores calculados de seções de choque - mais uma vez utilizando-se do *software Serpent* - gerados para uma região geométrica para células da malha. Isso se dá pelo uso de um *script* em linguagem *Python* que faz o mapeamento uma única vez na inicialização do problema e a partir daí utiliza os dados armazenados. Este sistema acoplado utiliza-se de forma elegante da capacidade fornecida pelo próprio *OpenFOAM* para decomposição de malhas e execução em paralelo. Com isso, o sistema pode ser escalado de acordo com o poder computacional disponível de forma absolutamente transparente. Pode-se dizer que este sistema é a evolução quase natural do trabalho apresentado pelo autor na sua tese de doutorado (JARETEG, 2012).

A importância dada aos cálculos de neutrônica e termo-hidráulica acoplados é ratificada no trabalho de Jaakko Leppänen sobre o código de Monte Carlo *Serpent* (LEPPÄNEN; VIITANEN, 2012). Até pouco tempo, a exigência computacional dos códigos de Monte Carlo impossibilitavam seu uso em cálculos acoplados e iterativos. O aumento quase exponencial na capacidade computacional jogou por terra tais restrições. Com os códigos de Monte Carlo executando mais rapidamente - em especial o código *Serpent-2* - o autor optou por oferecer uma interface acoplável por padrão neste *software*. No trabalho em questão, é implementada uma interface multi-física para acoplamento com um código termo-hidráulico externo. Atualmente, o *Serpent-2* conta com uma interface para utilização de malhas não-estruturadas no formato *OpenFOAM*.

A variedade de formas de acoplamento parece não parar de crescer. Miriam Vazquez (VAZQUEZ et al., 2012) propõe um sistema acoplado utilizando também o método de Monte Carlo, neste caso com o *software* MCNPX, juntamente com o código de subcanais COBRA-IV. Nessa primeira abordagem, os autores fazem algumas simplificações, como média no subcanal para a densidade do moderador, por exemplo. O intercâmbio de dados é feito na forma de arquivos texto, um método rudimentar mas funcional. Seções de choque são usadas de três formas diferentes, permitindo estimar alargamento *Doppler*. Este trabalho tem foco na simulação de um reator rápido a sódio, o que reforça o argumento do uso de cálculos acoplados no projeto de reatores inovadores.

Um trabalho que merece a atenção trata do acoplamento de neutrônica e termo-hidráulica com o objetivo de calcular transientes rápidos num reator conceitual que é uma variação de um reator do tipo CANDU (HUMMEL; NOVOG, 2016). Apesar de não trazer inovação na forma como se dá o acoplamento - uso de arquivos texto para leitura e escrita de dados intercambiados com cálculos separados de cada *software* - este trabalho

amplia o uso de sistemas de cálculos em estado estacionário para investigar situações de transientes rápidos no sistema conceitual.

O grupo de métodos multi-física do *Idaho National Laboratory* desenvolveu o sistema *MOOSE: Multiphysics Object Oriented Simulation Environment*. Este sistema objetiva a solução de problemas fortemente acoplados ao mesmo tempo em que fornece suporte para o desenvolvimento de, segundo o autor, "engenharia de aplicações de análise" (GASTON et al., 2009). O foco do seu desenvolvimento vai além do desempenho:

- Desenvolvido objetivando execução em paralelo **massiva** e escalabilidade, além de facilidade de uso das aplicações;
- Utiliza-se de modernos princípios de desenvolvimento de *software*, de modo a permitir manutenção e extensões de forma econômica;
- As aplicações devem obedecer à rigorosas especificações de verificação e validação, implicando no mesmo tipo de rigor para o próprio sistema.

Estas características tornam o *MOOSE* a plataforma de computação multi-física de base do *Idaho National Laboratory*, oferecendo suporte ao desenvolvimento de projetos e códigos de análise paralelos e multidimensionais.

Em referência à plataforma *MOOSE*, não tardaram a aparecer aplicações para este *framework*. Uma aplicação para a solução da equação de transporte de nêutrons em elementos finitos tridimensional, chamada $RATTLES_N AKE$, foi implementada. Para a queima de combustível nuclear, nos mesmos moldes e também utilizando elementos finitos em malha tridimensional, foi criada a aplicação *BISON*. Esta aplicação resolve implicitamente as equações termo-mecânicas acopladas, oferecendo modelos para inchaço e densificação do combustível, dentre outros. Sendo ambas as aplicações desenvolvidas sobre a plataforma *MOOSE*, o acoplamento entre ambas foi direto (GLEICHER et al., 2014). Os resultados dos cálculos acoplados para uma vareta combustível foram validados por uma simulação com o código Monte Carlo *Serpent*. Estes cálculos foram repetidos com malhas de diferentes granularidades e apontaram boa concordância com os resultados do *Serpent*.

Apesar de não ser um sistema *CFD*, mas de elementos finitos, a razão de se apresentar tanto a plataforma *MOOSE* quanto um exemplo de suas aplicações acopladas é atestar que a solução de problemas do domínio nuclear de forma acoplada não é hoje uma possibilidade, mas efetivamente uma realidade.

Vindo a reforçar esta ideia, o trabalho de Rodney Schmidt (SCHMIDT et al., 2015) apresenta o acoplamento de forma ainda mais ambiciosa. Neste trabalho, patrocinado pelo Departamento de Energia dos Estados Unidos (DOE), é proposto um ambiente virtual

para aplicações em reatores. O usuário é capaz de definir seu fluxo de trabalho obedecendo a regras simples e, a partir disto, realizar simulações do início ao fim usando uma variedade de aplicações distintas que se comunicam fazendo o cálculo multi-física. Algumas das aplicações são de neutrônica, termo-hidráulica, avaliação de desempenho de combustíveis e química dos reatores. Este é, até a presente data, um dos trabalhos de acoplamento com um objetivo mais amplo em termos de variedade de aplicações a serem usadas.

Recentemente foi apresentada uma metodologia de mapeamento orientada a objetos chamada SMITHERS (RICHARD et al., 2015). Consiste num sistema acoplado que faz mapeamento *on-the-fly* da distribuição de potência nos materiais num sistema de transportes de nêutrons baseado em MCNP para um outro código responsável pelos cálculos termo-hidráulicos. É uma metodologia complexa, que envolve cálculo de potência, queima, e utiliza um mapeamento robusto entre a representação nodal da maioria dos programas de cálculo termo-hidráulico e a geometria do código MCNP. Também faz uso de *scripts* em linguagem *Python* para tratamento do mapeamento além de estruturas de dados próprias dessa linguagem de programação para conseguir inserções e remoções em memórias a custo $O(1)$[3], o que significa que inserções e remoções em memória tem custo computacional constante. Os primeiros testes do uso do SMITHERS foram feitos para reatores do tipo BWR e PWR.

Alguns dos trabalhos apresentados na literatura, apresentam o que se poderia dizer, não sem certa pretensão, uma segunda "geração" de sistemas acoplados. Em especial os trabalhos envolvendo a plataforma *MOOSE* (e as aplicações feitas para serem usadas em seu *framework*) e, em uma escala um pouco menor mais ainda acima de acoplamentos específicos, a metodologia SMITHERS. Entretanto, há especificamente três trabalhos que se destacam por razões outras que merecem menção não apenas pela forma como estão relacionados com o trabalho desenvolvido nesta tese, mas pela potencial indicação de tendências no desenvolvimento de sistemas acoplados.

Desta lista mencionada, o primeiro trabalho consiste em um sistema neutrônico e termo-hidráulico acoplado (BENNETT et al., 2016). Neste acoplamento, são usados um código de Monte Carlo (*MCNP6*) para os cálculos neutrônicos e um código de sub-canais (*CTF*). Apesar de não apresentarem inovações grandes em relações a acoplamentos anteriores, há dois fatores para os quais deve ser chamada a atenção. O *MCNP6* é capaz de gerar seções de choque *on-the-fly*, uma característica desenvolvida com objetivo de oferecer suporte ao acoplamento termo-hidráulico. Há alguns anos, o uso de códigos de Monte Carlo para acoplamento era impensável. Entretanto, nove anos depois da publicação do principal trabalho de referência em acoplamento neutrônico e termo-hidráulico (IVANOV; AVRAMOVA, 2007), não só temos diversos exemplos do uso de códigos de

[3] Sobre a notação $O(n)$, amplamente utilizada em Ciência da Computação, uma referência é o livro Concrete Mathematics (GRAHAM et al., 1994, Seção 9.2)

Monte Carlo acoplados, como dois dos principais deles (*MCNP* e *Serpent*) já oferecem, por definição de projeto, formas de serem acoplados. A segunda razão que vale mencionar neste trabalho pode passar despercebida: como já mencionado, são nove anos entre dois trabalhos dos mesmos autores no tema de acoplamento. Isso serve como exemplo do que é hoje a execução de cálculos acoplados e a insistência em investigar formas de fazê-la cada vez mais robusta. A multi-física é hoje uma realidade.

Outro trabalho que merece uma atenção especial é o de Bryan Herman (HERMAN et al., 2015). Neste caso, novamente é usado um código de Monte Carlo para o acoplamento com a termo-hidráulica. A razão de apresentar este trabalho não reside na tentativa de demonstrar a importância dos cálculos multi física - espera-se que, neste ponto, o leitor já esteja convencido. O trabalho consiste principalmente em formas da aceleração da convergência das fontes de fissão, de modo a acelerar a obtenção do resultado final enquanto diminui-se o gasto de memória. Apesar da inovação na forma de acelerar os cálculos, a razão para uma menção especial deste trabalho não está no conteúdo técnico, mas no *software* de Monte Carlo em si. Neste trabalho é usado o código *OpenMC* (ROMANO; FORGET, 2013). Desenvolvido no *Massachusetts Institute of Technology* (MIT), é um código aberto oferecido com uma licença específica do próprio MIT. Esta licença garante a qualquer pessoa obter o código-fonte desse *software* e sua documentação associada sem custo. É também garantido o direito de usá-lo sem restrição. E é este o ponto em que se deseja dar ênfase: diferentemente da maioria dos *softwares* usados e disponíveis na indústria nuclear (leia-se indústria também englobando a academia) este é completamente aberto e livre. Um sinal de que a mudança de paradigmas na forma como se produz e utiliza *software* também chegou à área nuclear.

Feita a apresentação anterior, o terceiro trabalho que merece ênfase, já não pode ser considerado especificamente um sistema acoplado. O sistema *Cyclus* (HUFF et al., 2016) é certamente muito mais do que isso. Este sistema é um *framework* completo de simulação do ciclo do combustível nuclear. Seu desenvolvimento é, desde o projeto inicial, feito em forma de *software* livre. Seu objetivo é ser um sistema aberto, flexível, robusto e de uso geral, ao contrário das soluções existentes baseadas em sistemas fechados definidos para conjuntos de reatores específicos. No *paper* citado, os autores explicam os conceitos do projeto e desenvolvimento do *Cyclus*, com foco no uso de ferramentas abertas e desenvolvimento colaborativo e comunitário. Nesse sentido, um sistema como o *Cyclus* é, no mínimo, uma inspiração para o desenvolvimento de outros sistemas ou códigos com as mesmas características: de desenvolvimento aberto, comunitário e colaborativo. É este um dos objetivos desta tese: trazer as técnicas de desenvolvimento aberto, mas também oferecer o resultado obtido - um sistema acoplado de cálculos de reatores - a todos e quaisquer interessados em aprender, melhorar e contribuir com seu desenvolvimento.

Concluindo, nesse capítulo tentou-se dar uma visão geral do que é *software* livre

e do seu uso em situações de aplicações críticas e na indústria nuclear. Principalmente, chamou-se atenção para a importância que vêm ganhando as técnicas *CFD* na indústria nuclear e, sobretudo, seu recente uso na termo-hidráulica acoplada aos cálculos neutrônicos. A lista de aplicações citadas é certamente não-exaustiva e há razões para acreditar que o uso de *CFD* continuará a crescer, limitado apenas pela criatividade das suas aplicações. E não há razão para acreditar que essa última encontrará barreiras facilmente.

3 Metodologia

O trabalho de tese em questão trata do desenvolvimento e teste de um sistema de *software*. Desse modo, a metodologia utilizada no desenvolvimento deste sistema e, portanto, na solução do problema proposto, é construída com base nos seguintes aspectos:

- Adequação das ferramentas utilizadas para alcançar o objetivo de um sistema de cálculos acoplados;
- As restrições impostas pelas ferramentas escolhidas devido à sua estrutura intrínseca.

Desse modo, é desejável, do ponto de vista de clareza, descrever a metodologia utilizada no trabalho apresentando as características das ferramentas utilizadas, suas limitações e seus possíveis impactos no resultado final, de modo a então descrever a solução acoplada. Por sua vez, na descrição da solução acoplada, são apresentadas as modificações implementadas nas ferramentas e as formas de utilização de dois programas de computador independentes de forma conjunta.

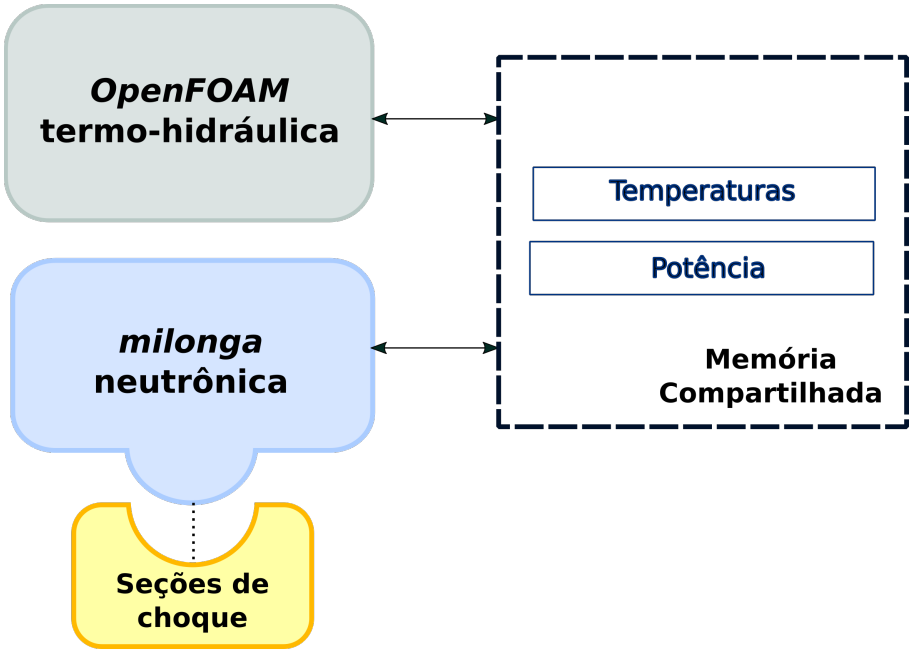
3.1 Visão Geral

O sistema de *software* desenvolvido tem algumas peculiaridades relativas à sua implementação. Isso se deve às particularidades do problema que se pretende resolver e ao fato não usual de envolver duas peças de *software* independentes para resolver um problema complexo, já apresentado como um problema multi-física.

Como apresentado no capítulo 2, acoplamento, no contexto dessa tese, é a execução de cálculos termo-hidráulicos utilizando a potência obtida pelo código de neutrônica que, por sua vez, utiliza as temperaturas dos materiais calculadas pela termo-hidráulica. Assim é possível adaptar, por meio de interpolações, os valores de seções de choque previamente calculados, de modo a calcular o fluxo de nêutrons e potência no combustível.

As interpolações referidas ocorrem a partir das temperaturas em cada volume de controle do cálculo por volumes finitos, sendo utilizadas tabelas previamente geradas de coeficientes utilizados na equação de difusão de nêutrons. Valores de seções de choque e coeficientes de difusão para dois grupos de nêutrons estão tabelados para quatro diferentes temperaturas e os valores destes coeficientes para cada célula são obtidos por interpolações lineares.

Figura 1 – Metodologia: o sistema acoplado.



A variação de densidades dos materiais modelados é ignorada neste trabalho. Entretanto, é necessário salientar que tal variação nas densidades tem implicação direta nos valores de seções de choque macroscópicas e que tal simplificação conceitual pode levar a diferenças importantes nos resultados dos cálculos.

As características não usuais desse sistema são listadas abaixo:

- Dois *softwares* independentes interligados;
- Uso de memória compartilhada para comunicação entre dois diferentes códigos;
- Natureza aberta das licenças de utilização de ambos os programas (item fundamental para o acesso, modificação e utilização do código-fonte);
- Uso da mesma malha por ambos os códigos.

Nas próximas seções serão apresentadas em seus detalhes cada uma destas características que permitiram o desenvolvimento de um sistema multi-física.

3.2 Conceitos

O sistema acoplado, ou multi-física, apresentado nesta tese possui algumas características particulares já citadas. Nesta seção, serão brevemente apresentados os conceitos sobre os quais foi possível desenhar e construir um sistema multi-física inovador com tais características.

3.2.1 Multitarefa

Multitarefa é a capacidade dos sistemas operacionais de executar distintas tarefas ao mesmo tempo. Talvez possa parecer corriqueira tal observação nos dias de hoje. Entretanto, há pouco mais de trinta anos, a maioria dos sistemas operacionais usados em computadores pessoais não ofereciam esta possibilidade. Em alguns casos, o que se chamou de multitarefa preemptiva, o usuário era capaz de carregar distintas tarefas em memória. Contudo, apenas uma era executada por vez.

Nos sistemas computacionais atuais, em especial no sistema Linux, utilizado para o desenvolvimento do sistema multi-física desta tese, é possível executar diversas tarefas ao mesmo tempo. O sistema operacional Linux implementa também um sistema padrão de execução que permite a comunicação entre programas distintos. Diferente de um sistema de *threads* (WALLI, 1995), em que todos os programas lançados enxergam a mesma porção de memória, no sistema Linux cada programa tem acesso exclusivo à memória alocada para si. Tudo isso acontece sob gerenciamento do sistema operacional.

A metodologia utilizada neste acoplamento prevê a execução separada dos programas de termo-hidráulica e neutrônica. Os detalhes e os algoritmos desta comunicação serão apresentados oportunamente (Seção 3.4). Por agora, é suficiente saber que os programas são lançados separadamente, cada qual utilizando a memória alocada para si exclusivamente, sem que outros programas possam acessá-la. Dessa forma, fez-se necessário desenvolver um sistema ou forma de comunicação entre os códigos de neutrônica e termo-hidráulica.

3.2.2 Memória compartilhada

Memória compartilhada (ou *Shared-memory*) é a memória de computadores que pode ser acessada simultaneamente por múltiplos programas em execução em diferentes espaços de usuários com o objetivo de oferecer comunicação entre eles, evitando cópias redundantes (ROBBINS; ROBBINS, 2003). É, ainda, uma forma eficiente de troca de dados entre programas ou processos. A interface POSIX (*Portable Operating System Interface*) (WALLI, 1995) é um padrão especificado pela *IEEE Computer Society* para garantir a compatibilidade entre diferentes sistemas operacionais na implementação das formas de utilização da memória compartilhada. Para isso, é definida uma API (*Application programming interface*) que explicita as funções e métodos de utilização da memória compartilhada. O padrão e a API disponibilizada garantem a compatibilidade no uso dos recursos entre diferentes programas ou *threads* (ATLIDAKIS et al., 2016).

Com mais de um programa acessando uma mesma área de memória, é necessário garantir a atomicidade das operações por cada programa. Por atomicidade, entenda-se que uma operação iniciada por um programa ou *thread* deve ser completamente encer-

rada antes que outro programa ou *thread* acesse esta memória. À possibilidade de acesso concorrente ao mesmo recurso de memória, dá-se o nome de condição de corrida.

Há diferentes técnicas para evitar condições de corrida. Dentre elas, uma das mais simples e largamente empregada é o uso de semáforos, usada nesta tese. Um programa ou *thread*, antes de acessar a memória, verifica se esta está livre para ser acessada. Se sim, altera o valor do semáforo enquanto opera na memória, voltando a alterá-lo para "livre" ao terminar. Caso outro programa ou *thread* necessite acessar a mesma porção de memória, encontrará o semáforo em condição negativa e aguardará um determinado tempo (definido pelo sistema operacional ou pelo próprio usuário) até tentar novamente. Com isso, fica garantida a não corrupção dos dados.

Deve-se dizer que há implicações negativas no uso de semáforos para controle de acesso à memória, como por exemplo, queda de desempenho dos sistemas, em especial quando há um número grande de programas acessando a memória compartilhada. Entretanto, a análise da possível degradação de desempenho causada por controle de concorrência entre programas não é escopo desta tese.

Mesmo com as considerações acima apresentadas, o uso de arquivos externos para acoplamento (HUMMEL; NOVOG, 2016) tem tempos de acesso ordens de grandeza acima dos tempos de acesso através de memória compartilhada (THELER et al., 2013).

3.2.3 Software livre

Nesta subseção, a análise se restringe às vantagens do acesso ao código-fonte em sistemas abertos.

Ambos os códigos usados no sistema acoplado, possuem a capacidade de serem acoplados sem alterações em seus códigos-fonte. O *milonga*, em especial, já foi desenvolvido com a possibilidade de acoplamento em mente. Portanto, oferece mais de uma forma de comunicação de dados, como por exemplo: arquivos texto, arquivos binários e, inclusive, de forma nativa, uso de memória compartilhada.

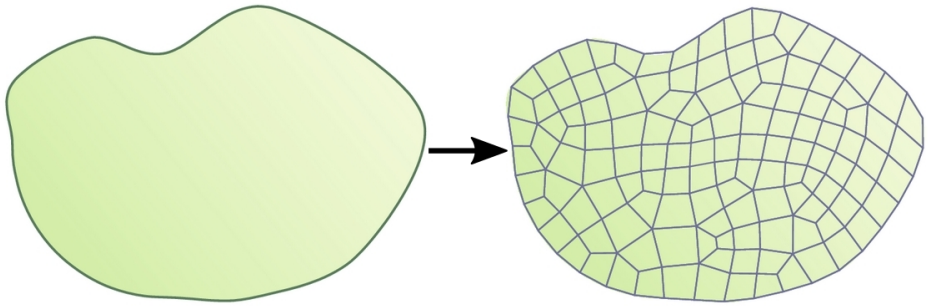
O *OpenFOAM*, entretanto, em um cenário de código fechado, só ofereceria uma forma de acoplamento. De acordo com sua implementação, um termo-fonte só pode ser lido a cada início de simulação. Seria, portanto necessário fazer com que o *milonga* lesse os dados de temperatura da saída do *OpenFOAM*, escrevesse uma distribuição de potências em formato de entrada para o *OpenFOAM* e que este fosse novamente executado do zero. Todo esse processo, deveria ser controlado por um *script* de execução.

Apesar de soar rudimentar, esta é a forma de acoplamento mais comumente empregada. Mesmo com severo comprometimento de desempenho se comparada ao compartilhamento de memória, técnica usada nesta tese, muitas vezes essa é a única forma de acoplamento quando o código-fonte não está disponível (IVANOV; AVRAMOVA, 2007).

3.2.4 Discretização do domínio

A formulação utilizada para a solução das equações diferenciais que modelam o problema multi-física desta tese é a de volumes finitos. Uma característica básica desta técnica é a utilização de pequenos volumes de controle de modo que as equações diferencias possam ser resolvidas como equações algébricas. A forma de obtenção destes pequenos volumes a partir de um domínio contínuo, ilustrado na Figura 2, é o que se chama **discretização de domínio**.

Figura 2 – Domínio contínuo (à esquerda) e discretizado (à direita).



Fonte: (THELER, 2013)

A discretização do domínio significa, na prática, gerar uma malha que represente o domínio contínuo original mantendo a conexão entre os pequenos volumes gerados. Se o domínio for tridimensional, será, geralmente, discretizado por uma malha volumétrica. Um dos diferenciais do acoplamento apresentado está exatamente na utilização da mesma malha para os cálculos neutrônicos, realizados pelo *milonga* e termo-hidráulicos, realizados pelo *solver OpenFOAM*. Com ambos os programas utilizando a mesma malha, a relação é de um pra um. Uma célula na neutrônica é representada exatamente pela mesma célula na termo-hidráulica.

Como mencionado anteriormente, a utilização da mesma malha para neutrônica e termo-hidráulica é uma característica particular do acoplamento proposto. São raros os acoplamentos com essa característica encontrados na literatura (JARETEG et al., 2014) e suas vantagens são, principalmente:

- Solução multi-física no mesmo nível de detalhes[1];
- Evita-se o mapeamento entre malhas que, no caso de malhas não-estruturadas, além de ser um problema de geometria computacional não-trivial , eventuais soluções podem acrescentar erros entre células (KRAEVOY; SHEFFER, 2004).

[1] Neste trabalho, se considera "nível de detalhes" o mesmo grau de discretização para ambos os problemas. O conceito pode ser estendido se for considerado o erro relativo dos cálculos em cada problema. Não é este o caso.

3.2.5 Limitações

O sistema multi-física construído tem limitações, principalmente devido à limitação intrínseca de um ou de ambos os programas usados no acoplamento. As restrições, como foram assim chamadas no início do capítulo 3, são apresentadas em forma de lista:

- **Sistema Operacional**: Ambos *OpenFOAM* e *milonga*, apesar de livremente disponíveis e, portanto, passíveis de utilização em diferentes sistemas operacionais são fornecidos especificamente para sistemas Linux. Alguns esforços recentes mostram eventuais aplicações do primeiro em sistemas Windows[2]. Já para o *milonga*, não há qualquer previsão de uso em outros sistemas operacionais;

- **Execução em Paralelo**: O *OpenFOAM* oferece de forma nativa a possibilidade de execução em paralelo. O *solver* `thesisCoupledFoam` foi, inclusive, implementado de modo a funcionar em paralelo. Entretanto, dada a limitação do *milonga*, até sua versão 0.4.65, em executar em paralelo, o sistema de comunicação do acoplamento está limitado à execução sequencial.

- **Seções de choque macroscópicas**: a geração de seções de choque deve ser feita por algum código externo. Os principais códigos empregados para geração de seções de choque são restritos [3] ou, como no caso do *Serpent* aberto mas restrito. Entretanto, neste campo também há uma recente atividade no que diz respeito a *software* livre. O conjunto de ferramentas *PyNE*, oferecido sob licença BSD (aberto e gratuito), tem como uma de suas propostas, a geração de seções de choque para aplicações de ciências nucleares e Engenharia Nuclear (SLAYBAUGH, 2014). A implementação desta funcionalidade ainda é incipiente. Outra iniciativa recente na geração de seções de choque a partir de dados nucleares, esta encabeçada pelo *Los Alamos National Laboratory*, é o lançamento do código NJOY (MCFARLANE et al., 2016) sob licença livre (BSD-3, 1999), com o objetivo de estender o desenvolvimento de novas funcionalidades à toda comunidade nuclear. O website desta nova versão é https://njoy.github.io/about/index.html.

- **Memória**: O *milonga* oferece, além da formulação de solução pela equação de difusão, algumas variações do método de ordenadas discretas. Entretanto, o consumo de memória e tempo de execução nestas formulações foi proibitivo no *hardware* disponível no momento de desenvolvimento desta tese.

[2] Em julho de 2016, a *OpenCFD*, detentora da marca *OpenFOAM*, lançou uma versão desta ferramenta para o sistema Windows. Esta versão utiliza uma tecnologia de distribuição similar a de máquinas virtuais. É não-nativa, portanto, não é, tecnicamente, uma distribuição para Windows.

[3] Por restrito, entenda-se que são fornecidos com código-fonte mas não são obtidos livremente nem podem ser assim distribuídos.

Apresentadas as limitações, cabe ressaltar que tais limitações, apesar de implicarem no uso, execução e até na qualidade dos resultados dos cálculos, não são definitivas. A quase totalidade das limitações e restrições apresentadas na lista acima podem ser resolvidas com moderado investimento de tempo. Especificamente, tempo empregado em implementação de novas funcionalidades como, por exemplo, a execução em paralelo do código de neutrônica ou a compilação das ferramentas para outro sistema operacional.

Isto posto, é possível perceber que no mundo do *software* limitações podem ser temporárias, de modo que um conceito bem estabelecido, mas com aplicação limitada, pode passar antes do que se imagina, a uma ferramenta de utilização ampla.

3.3 Ferramentas

Na seção anterior, foram brevemente descritos os conceitos sobre os quais foi construído o sistema acoplado desenvolvido nesta tese. Este sistema utiliza dois diferentes programas de computador. Cada um deles realiza um conjunto de cálculos separadamente e compartilha os dados necessários aos cálculos do outro.

Apesar de desenvolvidos separadamente e com objetivos diferentes, algumas características comuns - além do fato de serem ambos *software* livre - permitiram seu uso acoplado. Ambos possuem a capacidade de lidar com um mesmo formato aberto de arquivos de descrição de malhas, por exemplo.

As próximas subseções exploram as características destes programas que permitiram desenvolver a metodologia neste acoplamento. São também apresentados os principais desenvolvimentos técnicos relacionados a metodologia desenvolvida.

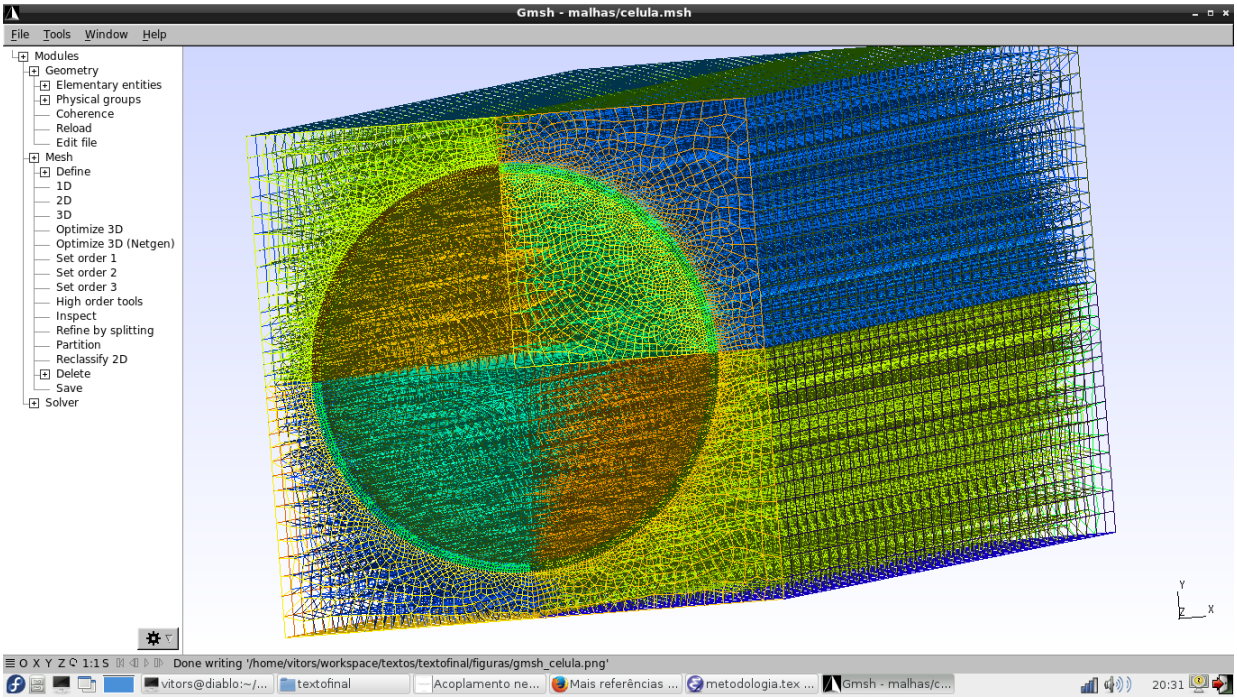
3.3.1 Geração de malhas: **Gmsh**

O software **Gmsh** (GEUZAINE; REMACLE, 2009) é um sistema gerador de malhas tridimensionais para elementos finitos com ferramentas de pré e pós processamento. Foi desenvolvido com o objetivo de oferecer uma ferramenta de geração de malhas rápida, leve e amigável para o usuário oferecendo, ao mesmo tempo, capacidade de entrada de dados paramétrica e visualização por interface gráfica. O Gmsh permite a criação de geometrias e malhas a partir de uma interface gráfica interativa, apresentada na Figura 3 com a malha gerada do modelo utilizado nesta tese. Outra opção para criação de geometrias, esta mais robusta que a interface gráfica, já que permite automatização através do uso de variáveis e estruturas simples de programação, é a leitura de arquivos textos em formato ASCII usando sua própria linguagem interna de *script*.

Assim como as outras ferramentas utilizadas no desenvolvimento do sistema acoplado desta tese, o Gmsh é distribuído nos termos da licença GPL (FREE SOFTWARE

FOUNDATION, 2007). Isso equivale a dizer que qualquer pessoa é livre para utilizar, modificar e distribuir o Gmsh, desde que não o utilize em sistemas fechados e/ou comerciais.

Figura 3 – Interface gráfica do Gmsh.



3.3.2 Termo-hidráulica: **OpenFOAM**

O *OpenFOAM* é um pacote para simulação numérica de Mecânica do Contínuo desenvolvido com base em orientação a objetos em linguagem $C++$. Do ponto de vista de Engenharia de Software, sua modularidade e flexibilidade são vantagens em relação a outros códigos monolíticos (JASAK et al., 2007). Sua implementação em componentes para manipulação de malhas, suporte à solução de sistemas lineares, operadores de discretização e modelos físicos em forma de bibliotecas o tornam um sistema CFD completo, aberto e gratuito. É utilizado em uma grande variedade de aplicações, desde a solução de escoamentos complexos envolvendo reações químicas, turbulência e transferência de calor, até acústica, mecânica dos sólidos e electromagnetismo.

OpenFOAM implementa um manipulador de malhas poliédrico, no qual células são descritas a partir de um conjunto de faces fechando um volume. As faces, por sua vez, são formadas por uma lista de pontos em suas coordenadas cartesianas e armazenados como vetores. Esta implementação é independente da discretização usada (JASAK, 2009). Entretanto, as principais classes que herdam as funcionalidades das malhas implementam o método de volumes finitos. Sendo assim, pode-se dizer que o *OpenFOAM* é um sistema de CFD baseado em volumes finitos (MARIĆ et al., 2014).

Além dos componentes dedicados a operações sobre o domínio, o *OpenFOAM* oferece um conjunto variado de programas independentes chamados *solvers*. Estes *solvers* são dedicados à solução de problemas físicos específicos. Para isto, utilizam outros elementos disponíveis no conjunto do *OpenFOAM*. O problema físico a ser resolvido nesta tese, especificamente do ponto de vista termo-hidráulico, é o da solução de troca de calor conjugada entre sólidos e fluidos distintos. Um dos *solvers* que o *OpenFOAM* possui com esse propósito é o `chtMultiRegionSimpleFoam` (OPENFOAM..., 2015b).

Este *solver* resolve problemas em estado estacionário (não são considerados termos variando no tempo) e para fluidos compressíveis. A modularidade do *OpenFOAM* fica clara na forma como o *solver* interage com os outros módulos: para regiões sólidas, são utilizados módulos de propriedades termofísicas específicos para sólidos. Por sua vez, para fluidos são oferecidos outros módulos com implementações relativas a propriedades de fluidos. Outro exemplo é a utilização de turbulência: caso o usuário estabeleça que seu escoamento é laminar, o módulo para escoamentos laminares é utilizado. Caso o escoamento seja turbulento, o usuário pode escolher entre implementações de diferentes modelos de turbulência.

O *solver* utilizado permite escolher entre energia interna e entalpia para a solução da equação da energia, de acordo com o módulo termodinâmico escolhido. Na implementação do problema acoplado, foi usada entalpia. O modelo de turbulência utilizado foi o $\kappa - \epsilon$ (LAUNDER; SPALDING, 1974).

O *OpenFOAM* é capaz de importar malhas no formato aberto **gmsh** (GEUZAINE; REMACLE, 2009). Este formato é o mesmo utilizado pelo *milonga* para leitura de malhas não-estruturadas.

3.3.3 Neutrônica: **milonga**

O sistema a ser resolvido consiste em uma representação em estado estacionário de um elemento combustível do tipo TRIGA. Este sistema pode ser resolvido utilizando-se a equação de difusão para solução do fluxo neutrônico e, a partir deste, calculada a potência nominal. Um *software* capaz de resolver este conjunto de equações diferenciais na mesma malha, ou seja, para o mesmo domínio utilizado pelo *OpenFOAM*, seria uma escolha ideal.

O *milonga* é um *software* aberto e livre para cálculo de física de reatores liberado sob licença to tipo GPL (FREE SOFTWARE FOUNDATION, 2007). Ele é construído utilizando-se de bibliotecas conhecidas como a *GNU Scientific Library* (GALASSI et al., 2009), a biblioteca PETSc (BALAY et al., 2016), que implementa ampla variedade de algoritmos para análise numérica, em especial para solução de sistemas lineares. O *milonga* utiliza também a biblioteca para solução de problemas da autovalores e autovetores SLEPc (HERNANDEZ et al., 2005). A reutilização de bibliotecas consagradas traz robustez para

o *milonga*, ao mesmo tempo seguindo os princípios do desenvolvimento de *software* livre.

O *milonga* resolve a equação de transporte de nêutrons multi-grupos em estado estacionário, utilizando a aproximação por difusão ou o método de ordenadas discretas. Oferece dois esquemas de discretização, elementos e volumes finitos. A capacidade de utilizar volumes finitos em malhas não-estruturadas permite sua utilização de forma acoplada com o *OpenFOAM*. O formato de leitura de malhas não-estruturadas utilizado pelo *milonga* é o **gmsh**, formato este que o *OpenFOAM* é capaz de importar, como mencionado anteriormente.

Dentre as distintas formas de resolver a equação de transporte oferecidas pelo *milonga*, optou-se por utilizar a aproximação por difusão. Apesar de resultados inacurados e das limitações na sua utilização em algumas circunstâncias (TRAHAN, 2014), a aproximação por difusão foi o modelo escolhido devido à sua execução mais rápida e menor consumo de memória do que os outros modelos disponíveis. A aproximação por difusão é apresentada na equação 3.1 já na sua forma discretizada para G grupos:

$$\begin{aligned} 0 = \nabla.\Big[D_g(\mathbf{x})\nabla\phi_g(\mathbf{x})\Big] - \Sigma_{tg}(\mathbf{x}).\phi(\mathbf{x}) \\ + \sum_{g'\neq g}^{G} \Sigma_{sg'\to g}(\mathbf{x}).\phi_{g'}(\mathbf{x}) + \chi(g) \sum_{g'\neq g}^{G} \frac{\nu\Sigma_{fg'}(\mathbf{x})}{k_{eff}}.\phi_{g'}(\mathbf{x}) \end{aligned} \tag{3.1}$$

onde,

D_g	Coeficiente de difusão para grupo g
$\mathbf{x}$	Vetor posição
ϕ_g	Fluxo para o grupo g
Σ_{tg}	Seção de choque macroscópica total para o grupo g
$\Sigma_{sg'\to g}$	Seção de choque de espalhamento do grupo g' para o grupo g
$\phi_{g'}$	Fluxo para o grupo g'
χ	Fração dos nêutrons de fissão oriundos do grupo g
ν	Número médio de nêutrons de fissão
$\Sigma_{fg'}$	Seção de choque macroscópica de fissão para o grupo g'
k_{eff}	Fator de multiplicação efetivo

Nessa formulação, espera-se que as seções de choque macroscópicas e os coeficientes de difusão sejam previamente computados por códigos de célula para serem usados na formulação por difusão. A dependência das seções de choque da temperatura pode ser considerada linear uma vez que os coeficientes são mantidos constantes durante cada passo do cálculo iterativo utilizado pelo *milonga*. Isso significa que, como a cada cálculo executado pelo *milonga* as seções de choque são efetivamente constantes, é fisicamente realístico interpolar as seções de choque linearmente para o conjunto de temperaturas disponível a cada iteração.

Na equação 3.1 assume-se que todos os coeficientes de difusão utilizados são contínuos no espaço. Na formulação de volumes finitos, o fluxo escalar é definido em cada célula. Quando células vizinhas são formadas pelo mesmo material, é possível estimar o módulo do gradiente do fluxo utilizando o teorema do valor médio (THELER, 2013).

Uma explicação elegante com o formalismo matemático utilizado na implementação do método de volumes finitos no *milonga* pode ser encontrada no trabalho de Theler (THELER, 2016, Seção 3.5.2).

De forma padrão, o *milonga* funciona lendo um arquivo de entrada que modela o problema a ser resolvido. Um dos casos mais simples em que se resolve o fluxo de nêutrons num sistema crítico, com seções de choque constantes, é apresentado na figura 4. Pode-se observar a definição da formulação, esquema de solução e número de grupos na linha 7. Os coeficientes da equação de difusão para um material qualquer são apresentadas nas linhas 10-16. Condições de contorno baseadas na malha estão definidas nas linhas 19-21 e, na linha 24, está o comando utilizado para iniciar os cálculos.

Além dos comandos já apresentados, o *milonga* possui um vasto conjunto de comandos e primitivas que vão desde pré-processamento da malha até primitivas de saída para visualização gráfica da solução.

Figura 4 – Exemplo de arquivo de entrada básico do milonga.

```
1  # milonga input example for two groups and two materials
2
3  # Read mesh file
4  MESH NAME fuelmesh FILE_PATH fuel.msh
5
6  # Define the problem formulation and characteristics
7  MILONGA_PROBLEM FORMULATION diffusion SCHEME volumes DIMENSIONS 3 GROUPS 2
8
9  # Define material characteristics for material 'mat1' for two groups
10 MATERIAL mat1 {
11   D1          1.110700e+00     SigmaT1      6.976520e-01       nuSigmaF1    3.668170e-03
12   SigmaS1.1   6.659240e-01     SigmaS1.2    2.721750e-02
13
14   D2          3.003750e-01     SigmaT2      1.999550e+00       nuSigmaF2    7.997020e-02
15   SigmaS2.1   1.962140e-04     SigmaS2.2    1.945410e+00
16 }
17
18 # Define boundary conditions accordingly to the mesh
19 PHYSICAL_ENTITY NAME top        BC vacuum
20 PHYSICAL_ENTITY NAME bottom     BC vacuum
21 PHYSICAL_ENTITY NAME wall       BC mirror
22
23 # Call the solver
24 MILONGA_STEP
25
```

Uma funcionalidade fundamental do *milonga* para o acoplamento neutrônico e termo-hidráulico é a possibilidade de definir as seções de choque como funções definidas geometricamente em x, y e z ou como combinação de ambos os métodos. Para o acoplamento, em especial, a definição de funções geometricamente permite estabelecer valores para os coeficientes de acordo com a posição da célula na malha. A aplicabilidade é ainda mais extensa, já que uma vez que é possível utilizar expressões pela posição na malha, é também possível definir concentração de veneno ou fração de vazio por elemento da

malha.

Além disso, o *milonga* pode ler dados em tempo de execução de arquivos de entrada, arquivos binários e, em especial no contexto desta tese, de memória compartilhada. Esta é uma funcionalidade deste *software* que o torna preparado para uso de forma acoplada.

3.4 Acoplamento

Sabe-se, neste ponto, que os dois programas, *milonga* e *OpenFOAM* funcionam independentemente e utilizam uma parte da memória do computador de forma compartilhada. Nesta seção, são apresentados os detalhes algorítmicos desta comunicação entre neutrônica e termo-hidráulica.

A Figura 1 apresenta uma representação gráfica do funcionamento do sistema acoplado desenvolvido.

Como funcionam separadamente, é necessário estabelecer, além dos dados a serem compartilhados como ambos os códigos irão se comportar. A forma de execução utilizada neste trabalho - e aqui cabe um comentário: há diversas formas de definir as bases de execução dos dois programas - é simples. Consiste em executar o *milonga*, que aguarda a inicialização do *OpenFOAM*. Os programas podem ser lançados em diferentes janelas ou na mesma janela em *background*, já que o funcionamento da memória compartilhada é independente, inclusive, do usuário que lançou o programa.

As próximas seções apresentam o algoritmo de acoplamento para cada código separadamente.

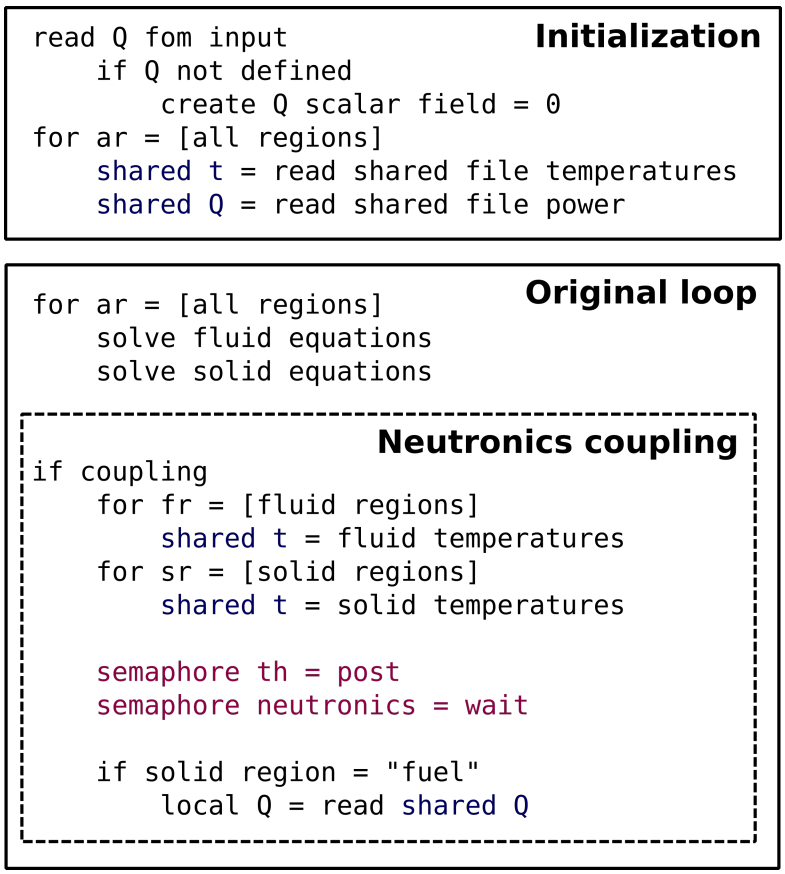
3.4.1 Algoritmo do sistema termo-hidráulico

O algoritmo de acoplamento do *solver* `thesisCoupledFoam` está implementado via código-fonte. Apesar de ser possível utilizar de referências ao código-fonte para apresentar o algoritmo, como feito para a implementação do termo-fonte na seção 3.4.3, esta forma é um pouco árida. Sendo assim, os algoritmos de acoplamento, para ambos os códigos, serão descritos por meio de pseudocódigo.

A Figura 5 apresenta o algoritmo de acoplamento para o *OpenFOAM*. As expressões em azul escuro mostram as instruções relativas ao uso de memória compartilhada, enquanto em vermelho estão as instruções relativas ao controle de acesso aos dados (semáforos). É possível notar, em relação à Figura 5, três principais divisões:

- **Inicialização:** Na inicialização do *OpenFOAM*, é lido o valor inicial da potência para a simulação caso o arquivo de definição de potência esteja disponível. Caso

Figura 5 – Algoritmo termo-hidráulica.



contrário, a potência é inicializada com zero. Os espaços em memória compartilhada são verificados. Caso não estejam corretamente alocados, o sistema assume que o sistema executará de forma não acoplada. Cabe lembrar, que o *milonga* é iniciado antes do *OpenFOAM* e, conforme será descrito no algoritmo da neutrônica, é ele o encarregado da alocação da memória compartilhada.

- **Laço original:** Este é o laço do *solver* original na sua execução do algoritmo SIMPLE. Este trecho não foi alterado em relação ao original e nesta etapa são resolvidas as respectivas equações para as regiões fluidas e sólidas. Ao fim deste laço, os valores de temperatura de cada célula de cada região estão calculados e disponíveis.

- **Acoplamento com a neutrônica:** Caso seja o momento de troca de dados (a implementação atual executa uma chamada acoplada para cada 100 iterações da termo-hidráulica [4].), para as regiões fluidas são copiados os valores de temperaturas para a memória compartilhada nas posições relativas às células das regiões fluidas. Em seguida, o mesmo é feito para as regiões sólidas. Com os valores calculados disponíveis na memória compartilhada, o *OpenFOAM* avisa (post) no semáforo relativo à termo-hidráulica (th) que a memória compartilhada está liberada para uso.

[4] O valor fixo é uma forma simples de controle de iterações. O valor 100 utilizado nesta tese é baseado em outra implementação similar na literatura (JARETEG et al., 2014).

No próximo passo, o *OpenFOAM* lê o semáforo da neutrônica que, neste instante, indica que o *OpenFOAM* aguarde. Quando receber o aviso (post) de liberação, o *OpenFOAM* continua e, apenas para a região nomeada como combustível (fuel), lê os dados de potência disponíveis na memória compartilhada para sua estrutura de dados local, um campo escalar volumétrico de potências.

Ao fim da execução da etapa relativa ao acoplamento, o controle de execução volta ao laço original. A inicialização é feita uma única vez para todos os passos de simulação.

3.4.2 Algoritmo do sistema neutrônico

No caso da neutrônica, não há necessidade de alterações no código-fonte do *milonga*. Absolutamente todo o controle do algoritmo de acoplamento é feito no arquivo de entrada. Na versão do *milonga* utilizada (0.4.65) as opções de utilização iterativa ainda são limitadas. Estruturas de repetição utilizando condicionais não estão disponíveis. Com isso, optou-se por definir um laço com número de execuções fixo, cujo valor é de mil chamadas da termo-hidráulica, suficiente para o caso de prova de conceito. A partir dessa restrição, foi desenvolvido o algoritmo de execução. A figura 6 mostra o algoritmo e suas etapas. As expressões em azul escuro mostram as instruções relativas ao uso de memória compartilhada, enquanto em vermelho estão as instruções relativas ao controle de acesso aos dados (semáforos). Cabe ressaltar que, como mencionado anteriormente, o *milonga* é iniciado **antes** do *OpenFOAM*.

- **Inicialização:** É definido o número de iterações a serem executadas - obrigatoriamente maior do que o número de vezes que o *OpenFOAM* fará chamadas ao *milonga*. Caso seja o primeiro passo, o *milonga* lê a malha, estabelece a formulação matemática, o esquema de discretização e o número de grupos de energia, exatamente como apresentado na figura 4. A seguir, através das primitivas de utilização de memória compartilhada presentes no *milonga*, são criadas áreas em memória compartilhada para armazenar temperaturas e potência. Os tamanhos destas áreas são obtidos do tamanho da malha carregada anteriormente. A inicialização, como é de se esperar, é feita uma única vez durante toda a simulação acoplada.

As instruções referentes ao acoplamento em si, ou seja, acesso à memória compartilhada e semáforos (caixas hachuradas na figura 6) estão localizadas dentro do laço de execução e serão descritas na explicação deste.

- **Laço de execução:**
  - **Acoplamento termo-hidráulico:** A execução dos cálculos neutrônicos se inicia com a leitura do semáforo da termo-hidráulica, o que faz com que o

Figura 6 – Algoritmo neutrônica.

```
steps = 1000                                    Initialization

if [first step]
    read mesh file
    problem formulation = diffusion
    problem scheme = volumes
    problem groups = 2

    shared t = create shared file temperatures
    shared Q = create shared file power

else                                            Execution loop
          ------------------------------------- Thermal-hydraulics coupling
    semaphore th = wait
    local t = read shared t
          -------------------------------------
    include cross-section data functions

    for am = [all materials]
        for ag = [all groups]
            assign = diffusion coeff[t]
            assign = sigma abs[t]
            assign = sigma scatter ag[t]

     assign boundary conditions
     solve diffusion equation
          ------------------------------------- Thermal-hydraulics coupling
     shared Q = read local Q
     semaphore neutronics = post
          -------------------------------------
```

milonga pare, aguardando os resultados dos cálculos termo-hidráulicos. Quando recebe a sinalização de que é possível prosseguir, o *milonga* armazena em sua estrutura interna os dados de temperatura provenientes da termo-hidráulica disponíveis na memória compartilhada.

– Cálculo neutrônico: São lidas as funções que armazenam seções de choque e todos os coeficientes previamente calculados para quatro diferentes temperaturas. A utilização de quatro temperaturas é uma referência para o teste conceitual. Para cálculos mais elaborados, a tabela de temperaturas dever ser detalhada e, neste caso, não pode ser negligenciado o impacto do tamanho de tal tabela no consumo de memória. Assim, para todos os materiais e para todos os grupos são atribuídos os coeficientes de difusão, seções de choque de absorção e seções de choque de espalhamento para cada célula, em função da temperatura da célula. Em seguida, são atribuídas as condições de contorno do problema neutrônico e iniciados os cálculos relativos à solução do problema.

– **Acoplamento termo-hidráulico:** Antes do fim do laço, com o problema neutrônico resolvido, fluxos e potência calculados, os valores de potência são escritos na memória compartilhada e então o *milonga* avisa (post) no semáforo relativo à neutrônica (neut) que a memória compartilhada está disponível para uso.

Ao fim da simulação termo-hidráulica, o *milonga* permanece parado. Para finalizá-lo deve ser enviado um sinal de cancelamento de execução via teclado. Todo o procedimento de acoplamento da neutrônica é feito por meio de um arquivo de entrada, sem necessidade de alteração do código-fonte e compilação, sempre interpretado em tempo de execução.

3.4.3 Implementação: visão geral do código-fonte.

Nesta subseção, são descritas as alterações promovidas no código-fonte do *solver* original do *OpenFOAM*. Este novo *solver* adaptado para o acoplamento foi chamado `thesisCoupledFoam`.

A principal alteração feita no *solver* originalmente implementado foi a adição de um termo-fonte. Esta alteração permite a geração de calor no sólido, útil tanto no problema acoplado quanto numa eventual simulação isolada em que se deseje geração de calor[5]. Isso foi feito acrescentando-se uma estrutura do tipo campo escalar volumétrico, estrutura de dados utilizada pelo *OpenFOAM*, para armazenar os valores de potência a serem usados. Sendo assim, na sua inicialização, o *solver* modificado busca, além dos arquivos padrão para as grandezas calculadas, um arquivo chamado `Q` que deve conter o campo volumétrico de potências em W/m^3. Caso este arquivo não seja encontrado, é inicializado com valores nulos para o termo-fonte.

O trecho de código em que a leitura do campo volumétrico para a potência é feita é apresentado na listagem 3.1.

Listagem 3.1 – Leitura/criação do campo volumétrico de potências.

```
1  PtrList<volScalarField> qVol(solidRegions.size());
2
3  // Read Q if it exists. If not, create a null (zero) volScalarField
4  IOobject Qfile("Q", runTime.timeName(), solidRegions[i],
5                 IOobject::READ_IF_PRESENT, IOobject::AUTO_WRITE);
6
7  // Must check it before creating the field
8  if(Qfile.headerOk())
9  {
```

[5] A partir da versão 2.2, o *OpenFOAM* introduziu em alguns dos seus *solvers*, incluindo o `chtMultiRegionSimpleFoam`, a possibilidade de definição de termo-fonte via `fvOptions`. Isso permite alteração da física do problema em tempo de execução. Infelizmente, a forma de implementação desta funcionalidade não atendia a uma utilização do *solver* de forma acoplada.

```
10      // Create a qvol field from dictionary
11      qVol.set(i, new volScalarField (Qfile, solidRegions[i]));
12    }
13    else
14    {
15      // If file is not there, create a new IOobject
16      // setting the dimensions of the field
17      qVol.set(i,
18      new volScalarField(IOobject
19                              ("Q", runTime.path(), solidRegions[i], IOobject::NO_READ,
20                                  IOobject::AUTO_WRITE), solidRegions[i], dimensionedScalar
21                              ("2", dimensionSet(1, -1, -3, 0, 0), scalar(0.0))
22                              )
23                          );
24    }
```

Na linha 1, é criada uma lista de ponteiros para campos escalares volumétricos. Essa estrutura retém as referências para os campos volumétricos de potencias de todas as regiões sólidas. Na presente implementação, apenas a região com nome **"fuel"** pode ter um valor não nulo para o campo de potências. Na linha 4 cria-se um objeto de entrada e saída de acordo com o arquivo **"Q"**. Se o objeto criado tiver o cabeçalho correto (linha 8) é criada uma entrada na lista de referências com os dados do arquivo (linha 11). Caso contrário, se um arquivo malformado for encontrado ou não for encontrado o arquivo, é criado um campo volumétrico escalar com valores nulos nas dimensões esperadas (linhas 17 a 23). A partir deste ponto, o campo escalar volumétrico **"Q"** está disponível para ser usado.

A criação do campo escalar volumétrico de potências é feita na inicialização dos campos dos materiais sólidos, anterior ao algoritmo de solução. O algoritmo de solução usado é o SIMPLE *(Semi-Implicit Method for Pressure-Linked Equations)*. Este algoritmo funciona estimando um valor inicial para o cálculo da pressão e então fazendo correções no valor estimado (VERSTEEG; MALALASEKERA, 2007). Esse procedimento é repetido até que se chegue a um valor aceitável dentro das condições de parada. Lembrando que este procedimento é feito para a solução do escoamento. O *solver* resolve separadamente cada região de acordo com seu tipo, fluida ou sólida, tudo isso dentro da iteração do algoritmo SIMPLE.

Um fragmento do laço de iterações do algoritmo SIMPLE é apresentado na listagem 3.2.

Listagem 3.2 – Fragmento do laço do algoritmo SIMPLE.

```
1   while (runTime.loop())
2   {
3     nIterations++;
4
5     forAll(fluidRegions, i)
6     {
7       // Fluid regions loop: removed for the sake of clarity
```

```
8     }
9
10    forAll(solidRegions, i)
11    {
12      #include "setRegionSolidFields.H"
13      #include "readSolidMultiRegionSIMPLEControls.H"
14      #include "solveSolid.H"
15
16      runTime.write();
17    }
```

O controle da simulação é feito pelo objeto `runTime`, responsável por avaliar resíduos e outras condições de parada. As regiões definidas como fluidas são resolvidas num laço (linha 5) e então são resolvidas as equações para as regiões sólidas (linha 10). O *OpenFOAM* utiliza-se de uma diretiva de inclusão (linhas 12 a 14) para adicionar ao código-fonte o conteúdo de outros arquivos definidos pelos nomes. Isso é feito com o objetivo de deixar o código-fonte mais legível como também de evitar reescrita de código, já que muitos dos arquivos incluídos são comuns a outros *solvers*. Um desses arquivos (linha 14) é o que contém as equações a serem resolvidas para a região sólida. É nele que o campo escalar volumétrico **"Q"** é utilizado. Um fragmento do arquivo de solução de sólidos é apresentado na listagem 3.3.

Listagem 3.3 – Criação do termo-fonte na equação da energia dos sólidos.

```
1   {
2     for (int nonOrth=0; nonOrth<=nNonOrthCorr; nonOrth++)
3     {
4       fvScalarMatrix hEqn
5       (
6       - fvm::laplacian(betav*alpha, h, "laplacian(alpha,h)")
7
8       // Source-term added to the equation
9       - Q
10      );
11
12      hEqn.relax();
13      hEqn.solve();
14    }
15  }
```

Dentro de um laço interno para correção de não-ortogonalidade, é definida a matriz de solução referente à entalpia (linha 4). Aos argumentos do objeto matriz sendo criado é adicionado o termo-fonte **"Q"** (linha 9). Estas são as principais modificações no código-fonte do *solver* para acrescentar o termo-fonte à equação do calor no sólido.

A segunda principal alteração no *solver* original para implementação do acoplamento está na adição das estruturas de dados para comunicação por memória compartilhada. Para isso, foi criado um arquivo fonte chamado `createCoupledFiedls.H` que foi posteriormente incluído no sistema completo.

Na listagem 3.4 são criadas as estruturas POSIX utilizadas para manipulação de memória compartilhada. Nas linhas 2 e 3 são declaradas as estruturas que serão utilizadas para acesso aos dados. Nas linhas 5 e 6 são declaradas as estruturas que serão utilizadas no mapeamento entre os dados em memória compartilhada e as estruturas de acesso aos dados. Apesar de contra intuitiva, deve-se lembrar que esta é a forma definida como padrão de utilização de memória compartilhada. Nas linhas 8 e 9 são declaradas as estruturas que representam os arquivos em memória compartilhada. Há, portanto, três estruturas para cada arquivo em memória compartilhada, sendo um arquivo para temperaturas e outro para potências. Nas linhas 12-13 são declarados os já mencionados semáforos, responsáveis pelo controle de acesso aos dados compartilhados. Na linha 19, é feito um teste se a execução é feita pelo programa principal - como também mencionado, o algoritmo implementando no *OpenFOAM* foi feito com vistas à execução em paralelo. Em sistemas paralelos, o mesmo código é executado por diferentes processos ou threads e operações de entrada e saída, como, por exemplo, leitura e escrita de arquivos, deve ser feita unicamente por um dos processos lançados. Sempre com vistas a manter a consistência. Em geral, este tipo de operação é feita pelo programa principal. É este o caso na implementação do sistema desta tese, em que no programa principal são executados comandos de leitura dos arquivos já presentes em memória compartilhada. Arquivos estes, criados pelo *milonga* na sua inicialização. Caso sejam encontrados os arquivos, o *OpenFOAM* segue em modo acoplado. Caso contrário, executará sem acoplamento.

Listagem 3.4 – Fragmento do código-fonte da criação das estruturas de memória compartilhada.

```
1  // Three C standard arrays are created to shared data with milonga
2  double *shmTarray = NULL;
3  double *shmQarray = NULL;
4
5  void *shmT;
6  void *shmQ;
7
8  int shmTfile = 0;
9  int shmQfile = 0;
10
11 // Posix C semaphores
12 sem_t *calcOf;
13 sem_t *calcMil;
14
15 // Posix structures initialization
16 calcOf = sem_open("calcOf", O_CREAT, 0666);
17 calcMil = sem_open("calcMil", O_CREAT, 0666);
18
19 if(Pstream::master())
20 {
21 // Semaphores and shared memory files are tested at this point.
22 // If milonga is not running, OpenFOAM runs uncoupled.
23     shmTfile = shm_open("temperaturas", O_RDWR, 0666);
24     shmQfile = shm_open("potencias", O_RDWR, 0666);
```

```
25
26      if(shmTfile == -1 || shmQfile == -1) coupling = false; // Error finding shared memory
27  }
```

Já em referência à listagem 3.5, estão apresentadas as estruturas de verificação de consistência. Em outras palavras, além declarar os dados, é necessário garantir que ao tentar utilizar os semáforos e o conteúdo em memória compartilhada, estes estejam disponíveis e em estado consistente. Nas linhas 4 e 5, dentro de um teste de execução sequencial, os arquivos em memória compartilhada são mapeados para as estruturas correspondentes, que guardarão referências para os dados em memória compartilhada (armazenados, por sua vez, em forma de arquivos). Na linha 8, são verificados os mapeamentos e, caso haja algum problema, o programa é interrompido. Caso contrário, segue normalmente a execução. Finalmente, nas linhas 11 e 12 é feito o que se convencionou chamar "coerção" em Ciência da Computação. A coerção é a mudança do tipo definido de uma variável para outro. Na presente implementação, essa coerção é feita para garantir que os dados utilizados para o acesso à memória compartilhada, cuja API é implementada em linguagem C, possa ser usada sem riscos pelo *OpenFOAM*, implementado em linguagem C++.

Listagem 3.5 – Fragmento do código-fonte da verificação das estruturas de memória compartilhada.

```
1  if(Pstream::master())
2  {
3  // After reading the shared memory files, they must be mapped to data
4      shmT = mmap(NULL, totalNumberOfCells*sizeof(double), PROT_WRITE, MAP_SHARED, shmTfile, 0);
5      shmQ = mmap(NULL, totalNumberOfCells*sizeof(double), PROT_WRITE, MAP_SHARED, shmQfile, 0);
6
7  // Check if all files were properly mapped
8  if((shmT == MAP_FAILED || shmQ == MAP_FAILED) && (coupling)) exit(errno); // Error mapping
       shared memory
9
10 // Make a C++ cast
11     shmTarray = reinterpret_cast<double*>(shmT);
12     shmQarray = reinterpret_cast<double*>(shmQ);
13 }
```

Espera-se, ao descrever algumas das implementações feitas em fragmentos do código-fonte do *OpenFOAM*, apresentar ao leitor o formato interno do *OpenFOAM* e detalhes de como se deu, na prática, a parte da metodologia de acoplamento desenvolvida nesta tese em relação às alterações no código-fonte do *OpenFOAM*. Em uma tese em que o desenvolvimento de *software* é a principal contribuição científica, julgou-se pertinente apresentar detalhes de como se deu parte deste desenvolvimento. Alguns exemplos são, por vezes, uma forma útil de fazer a conexão entre a metodologia e a prática.

Ao leitor interessado em aprofundar-se nos detalhes técnicos da implementação, fica o convite a examinar livremente código-fonte do *solver* `thesisCoupleFoam`, disponível no repositório `https://github.com/vitorvas/thesisChtMultiRegionFoam`.

4 Aplicação

De modo a avaliar o sistema de *software* desenvolvido com base na metodologia apresentada, foi desenvolvido um modelo **simplificado baseado** em um sistema físico existente, no caso, o reator TRIGA IPR-R1 do CDTN. A elaboração deste modelo levou em consideração os aspectos físicos de um único elemento combustível - envolvido por certa quantidade de água - do reator TRIGA em condições de utilização usuais de trabalho.

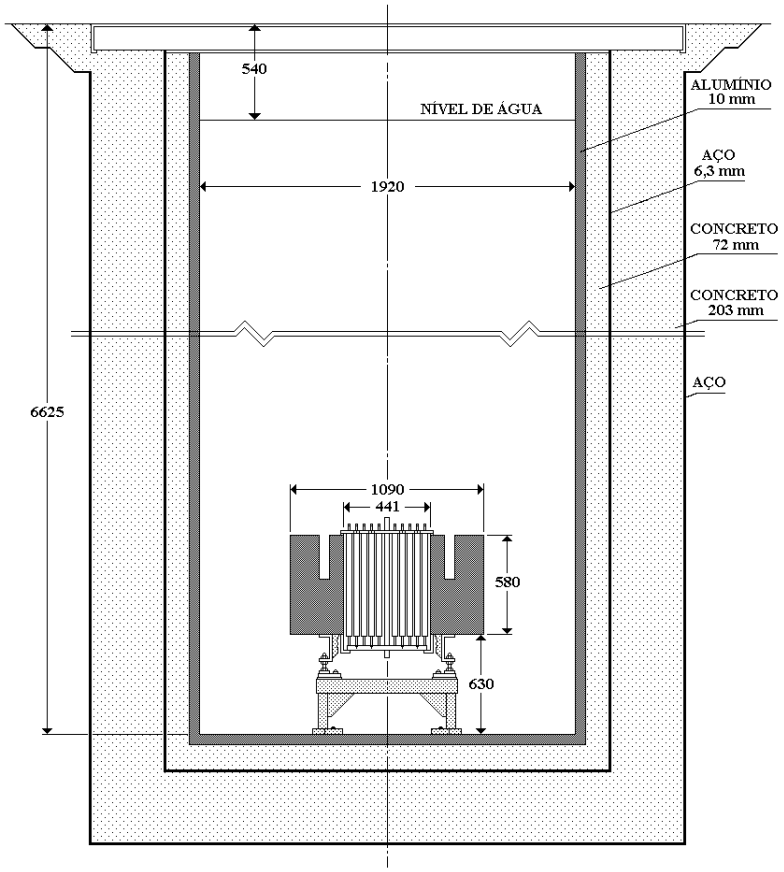
4.1 Reator TRIGA IPR-R1

Os reatores de pesquisa do tipo TRIGA®(*Training, Research and Isotopes*), desenvolvidos pela *General Atomics* (GA), são os mais utilizados no mundo (66 instalações em 24 países). Seu projeto, que data dos anos 50, objetivou simplicidade de operação e manutenção. Devido ao uso inovador de combustíveis de hidreto de zircônio e urânio, o reator TRIGA tem como característica um amplo coeficiente negativo de reatividade. Isso faz com que os reatores do tipo TRIGA Mark I sejam intrinsecamente seguros, tornando-os excelentes opções para uso em atividades de treinamento e pesquisa.

O reator TRIGA IPR-R1 é do tipo piscina aberta, ou seja, fica localizado no fundo de um tanque revestido de alumínio, coberto por água e visível da superfície, como apresentado na Figura 7. A refrigeração dos combustíveis presentes no núcleo do reator se dá por convecção natural. Devido à variação espacial no fluxo neutrônico no núcleo, alguns combustíveis são mais aquecidos que outros, levando a variações no aquecimento também na água nos volumes compreendidos entre os combustíveis. Estes “espaços” entre combustíveis por onde escoa água são chamados de subcanais.

A temperatura média nos subcanais foi calculada por Veloso (VELOSO, 2005, Capítulo 8) com vistas ao licenciamento do reator TRIGA IPR-R1 para operação a maior potência, junto ao órgão fiscalizador. Sua metodologia classificou cada um destes subcanais, realizando cálculos levando em consideração as propriedades físicas de cada um destes separadamente. Uma detalhada explicação sobre métodos de cálculos por subcanais com aplicação no reator TRIGA IPR-R1 pode ser encontrada na tese de Veloso (VELOSO, 2004). Os resultados obtidos neste trabalho, bem como as propriedades dos subcanais e do núcleo do reator completo, são utilizados como referência no desenvolvimento do modelo simplificado criado para a realização dos testes na presente tese. A divisão original em subcanais pode ser vista na Figura 8. Cabe ressaltar que na modelagem simplificada foram utilizadas as características do escoamento do subcanal número 6, mas a geometria foi modelada como um único pino cercado por água.

Figura 7 – Corte vertical do poço do reator TRIGA IPR-R1.



Fonte: (VELOSO, 2005)

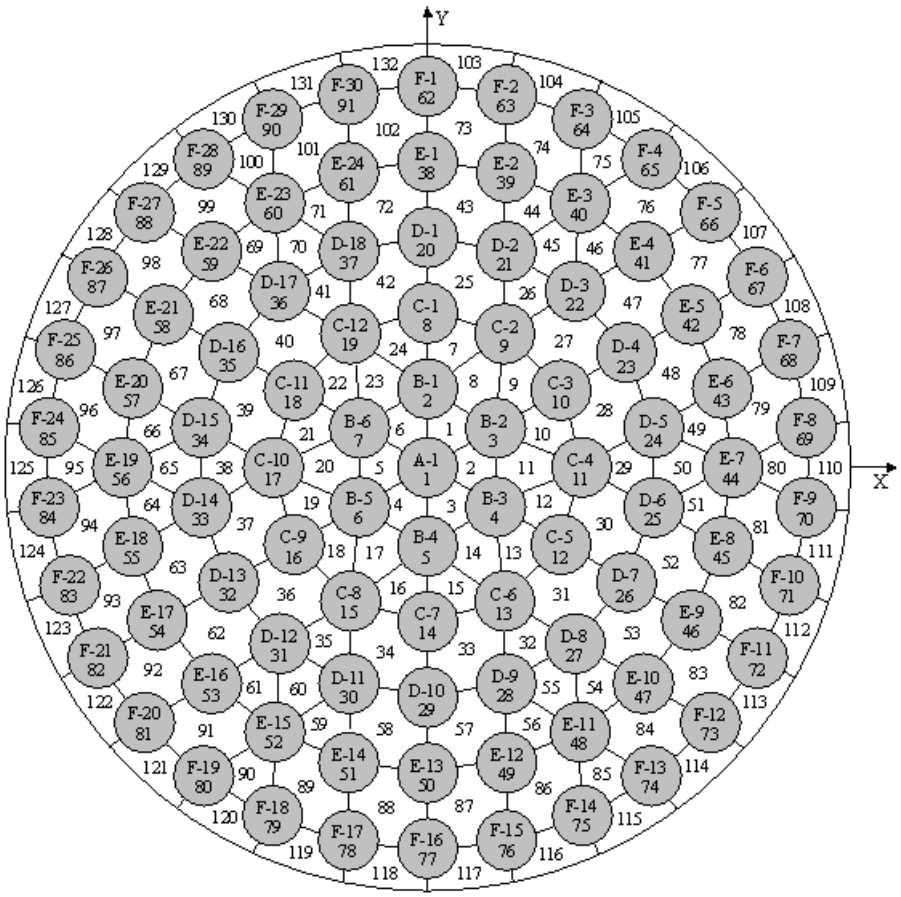
4.2 Modelo

Nesta seção é descrito o modelo, tanto nas suas características físicas quanto numéricas, utilizado nos testes do sistema acoplado. O desenvolvimento deste modelo se deu de modo a balancear as características termo-hidráulicas e neutrônicas desejáveis para um sistema de prova de conceito de cálculos acoplados e as limitações técnicas impostas, tanto em relação à capacidade computacional disponível quanto às limitações intrínsecas dos sistemas de *software* utilizados.

4.2.1 Fatores limitantes

O modelo de avaliação do acoplamento foi concebido de acordo com as restrições impostas pelos sistemas utilizados na implementação do acoplamento. O *OpenFOAM*, por se tratar de um sistema mais maduro e com ampla rede de usuários, não impôs restrições ao problema em termos de tamanho de malha a ser utilizada. Entretanto, a versão do *OpenFOAM* utilizada, apresenta um erro (*bug*, no jargão da Ciência da Computação) na implementação da resistência de contato, o que inviabilizou a simulação do *gap* exis-

Figura 8 – Divisão do núcleo do reator TRIGA para cálculos de subcanais (VELOSO, 2005).



Fonte: (VELOSO, 2005)

tente no combustível. Essa foi a primeira simplificação imposta ao modelo em relação ao combustível do reator TRIGA IPR-R1.

No tocante ao *milonga*, as restrições foram maiores. Foram duas as grandes limitações impostas pelo *milonga* na definição de um modelo mais complexo e computacionalmente exigente. A primeira é relativa à execução de forma sequencial. Como o código *milonga* não é ainda capaz de executar de forma paralela, o tamanho da malha a ser utilizada fica restrito à memória disponível. O tamanho da malha também implica diretamente no tempo de execução. Além disso, o *milonga* ainda não é capaz de lidar com elementos prismáticos de forma confiável, limitando a geração da malha por extrusão à utilização de elementos hexaédricos. Como no caso de um combustível do tipo TRIGA o revestimento tem espessura ordens de grandeza menor do que o raio do combustível, a não utilização de elementos prismáticos impactou na geração de uma malha mais refinada.

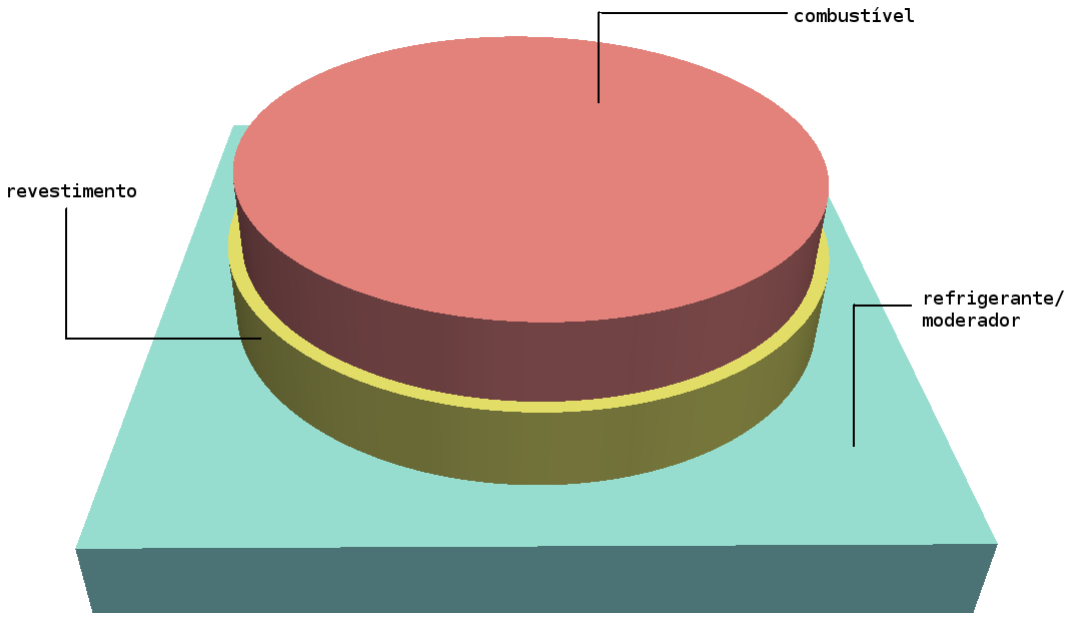
A segunda restrição imposta pelo *milonga*, também em razão de um *bug*, está na função de acesso por coordenadas espaciais, fundamental para os cálculos acoplados com variação de seções de choque. Este *bug* limita o uso de regiões com o mesmo nome a

regiões conectadas por elementos de malha. Por exemplo, numa suposta malha para todo o núcleo do reator TRIGA IPR-R1, são várias as regiões a serem definidas como combustível, estando tais regiões separadas por elementos de malhas de outras regiões. Neste caso, a função de acesso à coordenadas espaciais do *milonga* interrompe sua execução ao percorrer completamente a primeira região encontrada com determinado nome, ignorando outras regiões com o mesmo nome. Com isso, não é possível simular modelos de subcanais nem modelos do núcleo completo utilizando funções de coordenadas espaciais para cálculo de seções de choque. Como o acoplamento, como previsto, necessita da atualização em todas as regiões definidas como combustível, este erro limitou o modelo a ser utilizado a um modelo com uma única região de combustível[1].

4.2.2 Modelo: características físicas e numéricas

O modelo escolhido, de acordo com as limitações impostas já descritas, representa um elemento combustível (baseado no elemento B6) do reator TRIGA IPR-R1 8. Os combustíveis dos reatores do tipo TRIGA possuem uma grande sensibilidade na sua reatividade com a temperatura devido à já citada presença do hidreto de zircônio em sua constituição. Essa característica intrínseca permite que pequenas variações nas suas temperaturas levem a variações perceptíveis na neutrônica, o que torna este modelo interessante para avaliação do acoplamento.

Figura 9 – Modelo: regiões e materiais.



[1] Durante a execução desta tese, alguns *bugs* encontrados no *milonga* foram corrigidos e reportados ao seu autor. Uma das correções acabou com um vazamento em memória que impedia que o *milonga* fosse executado mais do que algumas vezes em modo iterativo.

Figura 10 – Vista isométrica do modelo completo.

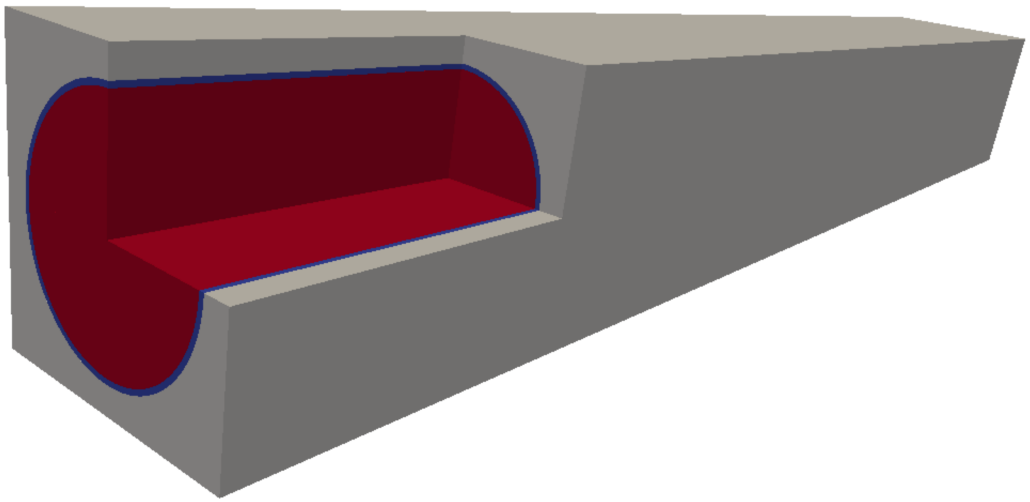
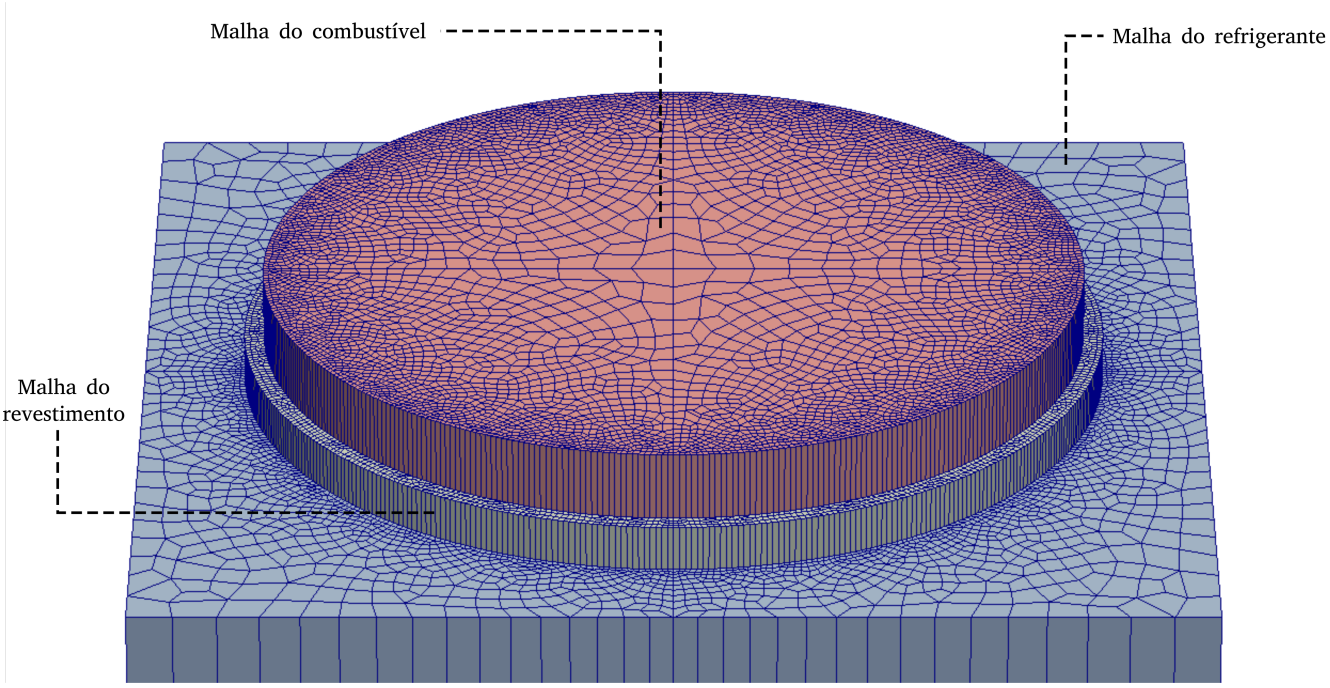


Figura 11 – Discretização do modelo: regiões e materiais.



O modelo consiste num combustível com revestimento de alumínio sem o *gap*, de modo que o diâmetro do combustível é mantido como no combustível de referência e o diâmetro do *gap* é estendido para ocupar este espaço. Além disso, o modelo considera apenas o comprimento ativo do elemento combustível. O volume de água ao redor do combustível foi modelado considerando a razão entre moderador e material físsil no núcleo completo do reator (razão Vm/Vf). Nenhuma outra estrutura sólida, como discos de samário ou elementos estruturais, é representada no modelo, sendo este idêntico axialmente em qualquer altura, como observável nas Figuras 9 e 10.

Tabela 1 – Medidas do modelo.

Estrutura	[cm]
Raio do combustível	1,78
Raio externo do revestimento	1,865
Arestas da região do refrigerante	4,57
Altura do modelo do combustível	35,0

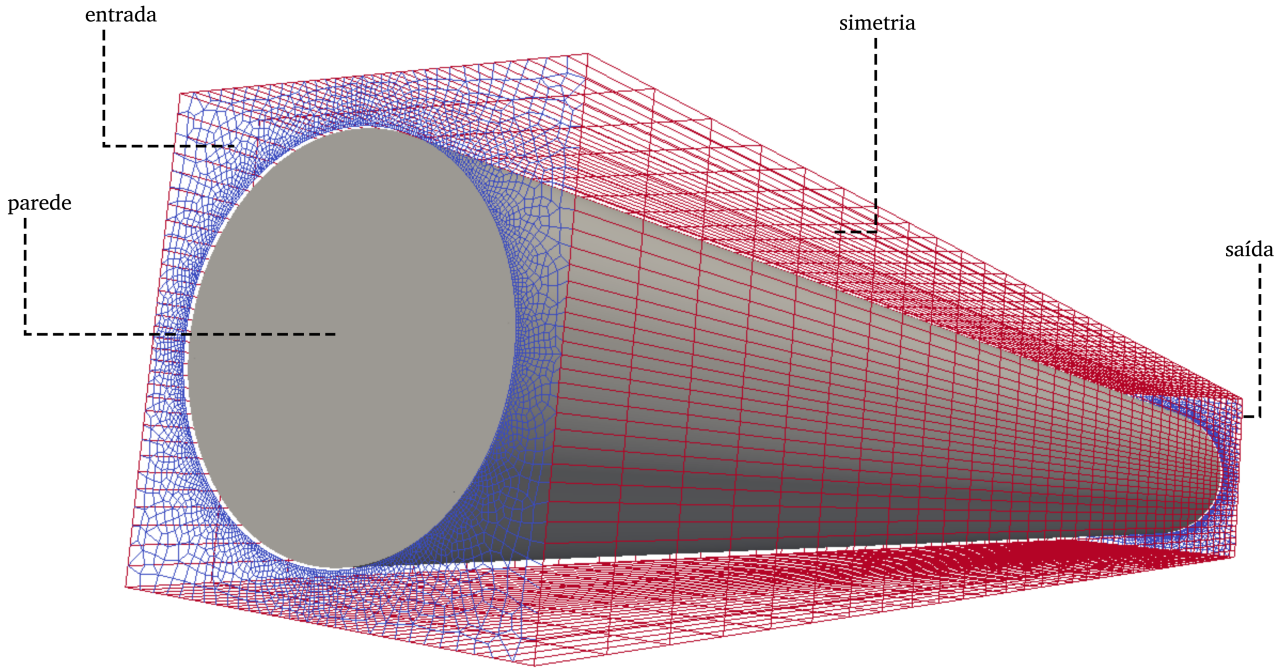
A malha gerada para o modelo foi construída por extrusão. Isso significa que os elementos de malha foram gerados para uma superfície e a dimensão espacial foi dada por um valor de comprimento e um número de camadas a serem geradas. Foram geradas 35 camadas (cada uma com 1 cm), sendo o tamanho relativo mínimo para elemento de malha de 0,5 cm e máximo de 4.0 cm. O tamanho relativo mínimo é usado próximo a arestas em interfaces de regiões para captura de camada limite e detalhes, enquanto o tamanho relativo máximo é utilizado em regiões de esperada baixa variação em propriedades e grandezas calculadas. Esses valores controlam o quanto o elemento próximo ao ponto com tal tamanho relativo crescerá durante a geração da malha. A razão de crescimento entre o refinamento do revestimento e as áreas restantes foi estabelecida em 1,2. Com isso, foi possível conseguir elementos suficientemente pequenos para realizar os cálculos na região do revestimento e ainda obter uma malha com tamanho tratável pelo *milonga*. A malha gerada tem 346.675 elementos.

As dimensões da malha do modelo geram implicações nos resultados dos cálculos, em especial a pequena granularidade axial (35 camadas). Esta granularidade pode ser observada na Figura 11, na qual é possível notar o refinamento entre as diversas regiões e, em contraste, o tamanho dos elementos longitudinalmente. Entretanto, mesmo com tais limitações impostas pelo *milonga* na sua capacidade de lidar com malhas de muitos elementos, é possível utilizar a malha gerada para verificar as diferenças nos resultados dos cálculos entre um sistema não acoplado e outro acoplado.

4.2.3 Condições de contorno e parâmetros numéricos: termo-hidráulica

São quatro condições de contorno utilizadas na solução da simulação termo-hidráulica nas superfícies que limitam o domínio. As extremidades do combustível e do revestimento são definidas como adiabáticas. Nas extremidades da região do refrigerante (água), são definidos entrada (*inlet*) e saída (*outlet*) para o escoamento. As laterais do modelo são definidas como simétricas. Na figura 12 são mostradas as condições de contorno utilizadas e as regiões correspondentes. As superfícies definidas como simétricas (*symmetry*) estão em vermelho enquanto em azul estão a entrada e saída de fluido. A região sólida representa o combustível e revestimento, ambos com condições de contorno adiabáticas nas extremidades.

Figura 12 – Modelo: condições de contorno da simulação termo-hidráulica e regiões correspondentes.



As superfícies entre regiões têm suas condições de contorno definidas automaticamente num dos passos da etapa de pré-processamento. Por definição, o *OpenFOAM* importa a malha num único conjunto de elementos, armazenando as definições de regiões contidas na malha. Um dos utilitários disponibilizados pelo *OpenFOAM* separa as regiões, gerando condições de contorno em cada uma das regiões que representam as interfaces entre regiões. Outro utilitário é utilizado em seguida para definir valores para os parâmetros das respectivas condições de contorno, de acordo com as definições previamente feitas pelo usuário. Como resultado do preprocessamento entre regiões, são obtidas condições de contorno que implementam a troca conjugada de calor entre as distintas regiões. Na tabela 2 são apresentadas as condições de contorno utilizadas nas superfícies entre regiões para a temperatura. É possível perceber que todas as superfícies, tanto para campos sólidos e fluidos, utilizam a mesma condição de contorno: `compressible::turbulentCoupledBaffleMixed`.

Tabela 2 – Condições de contorno entre superfícies internas (preprocessamento)

Região	Superfície	Condição de contorno	Campo
Combustível	*fuel_to_cladding*	compressible:: turbulentCoupledBaffleMixed	T
Revestimento	*cladding_to_fuel*		
	cladding_to_coolant		
Refrigerante	*coolant_to_cladding*		

As condições de contorno devem ser estabelecidas para todas as grandezas sendo

calculadas. Algumas destas grandezas, como a potência, por exemplo, são automaticamente calculadas nas superfícies a partir dos campos internos. Entretanto, isto não desobriga o usuário de explicitamente defini-las como *calculated*.

Esta condição de contorno utiliza as propriedades termofísicas dos respectivos materiais, que são previamente definidas na modelagem do problema, para o cálculo do fluxo de calor entre as regiões. As propriedades termofísicas dos materiais utilizados por este modelo são dadas na tabela 3.

Tabela 3 – Propriedades termofísicas dos materiais em função da temperatura (T)

Material	Densidade $[kg/m^3]$	Calor específico $[kJ/kg.K]$	Condutividade $[W/m.K]$	Viscosidade $[\mu Pa.s]$
Combustível $UZrH$	6280,0	$0,294+$ $6{,}196 \text{ x } 10^{-4}T-$ $2{,}748 \text{ x } 10^{-9}T^2+$ $1{,}354 \text{ x } 10^{-11}T^3$	22,872- $4{,}3131 \text{ x } 10^{-2}T+$ $1{,}124 \text{ x } 10^{-4}T^2-$ $1{,}0039 \text{ x } 10^{-11}T^3$	-
Revestimento Alumínio 110	2705,0	$0,892+$ $4{,}44361 \text{ x } 10^{-4}T+$ $3{,}632 \text{ x } 10^{-8}T^2$	223,7+ $4.756 \text{ x } 10^{-2}T+$ $1{,}0215 \text{ x } 10^{-5}T^2-$ $1{,}8887 \text{ x } 10^{-11}T^3$	-
Refrigerante Água (20°C)	995,0	4,18	0,62	797

As principais condições de contorno e iniciais do problema apresentado estão definidas na tabela 4. Estas condições descrevem as características iniciais do problema a ser resolvido. Alguns comentários sobre a tabela 4: a velocidade na entrada do sistema se baseia na velocidade média do escoamento na entrada do núcleo do reator (VELOSO, 2005). A região do revestimento, por ser sólida, herda a implementação do termo-fonte feita no *solver*. Ou seja, como trata-se de uma região sólida, a região equivalente ao revestimento também espera e necessita da definição de um campo volumétrico de potências. Entretanto, este é mantido nulo já que não há geração de energia no revestimento. O campo interno inicial de potências é **não-uniforme** e definido como variável. O campo não-uniforme permite estabelecer uma condição inicial de campo heterogêneo, tal como uma distribuição volumétrica de potência. O valor para este campo é definido como **variável**, já que esta distribuição de potências varia para cada diferente simulação.

Vale ressaltar que o que se convencionou chamar no texto de potência, tal qual implementada no *solver*, é dada em W/m^3, ou seja, é uma densidade de potências volumétrica.

O escoamento do fluido no modelo é turbulento, baseado nas condições usuais de operação do reator TRIGA IPR-R1 (VELOSO, 2005). A turbulência foi modelada utilizando-se o modelo $\kappa - \epsilon$ (LAUNDER; SPALDING, 1974), que assume que a viscosidade turbulenta está relacionada à energia cinética (κ) e dissipação (ϵ).

Tabela 4 – Condições de contorno iniciais para a termo-hidráulica

Região	Campo	Fronteira	Tipo	Valor
coolant	T	inlet	fixedValue	300 $[K]$
		outlet	zeroGradient	-
	U	inlet	fixedValue	0.1 $[m/s]$
		outlet	zeroGradient	-
	p	inlet	zeroGradient	-
		outlet	fixedValue	0 $[kg/m.s^2 = Pa]$
cladding	T	wall	zeroGradient	-
	Q	wall	calculated	-
		internalField	uniform	0 $[W/m^3]$
fuel	T	wall	zeroGradient	-
	Q	wall	calculated	-
		internalField	non-uniform	(variável) $[W/m^3]$

Uma vez definidas as características físicas do modelo e as condições iniciais e de contorno, o problema está fechado e é possível iniciar os cálculos propriamente ditos.

Entretanto, para isso, são necessárias, além das configurações relativas ao problema físico, as definições relativas aos esquemas de discretização a serem utilizados e aos algoritmos numéricos a serem utilizados. Entenda-se por esquemas de discretização, os algoritmos utilizados na transformação dos operadores matemáticos sobre grandezas contínuas em equações algébricas operando sobre grandezas discretas.

São apresentados na tabela 5 os esquemas de discretização utilizados nos cálculos acoplados e não acoplados. O manual do programador do *OpenFOAM* (OPENFOAM..., 2015a, Seção 2.4) descreve com detalhes alguns dos métodos de discretização de equações disponíveis na ferramenta.

Tabela 5 – Esquemas de discretização utilizados nos cálculos de volumes finitos

	Termo da equação				
Região	$\nabla\phi$	$\nabla.\phi$	$\nabla^2.\phi$	interpolação	normal à face da célula
Combustível	Gauss linear	-	Gauss linear não-corrigido	linear	não-corrigido
Revestimento	Gauss linear		Gauss linear não-corrigido	linear	não-corrigido
Refrigerante	Mínimos quadrados	*Bounded Gauss upwind*	Gauss linear limitado 1.0	linear	limitada 1.0

Os esquemas utilizados na solução do problema apresentado foram escolhidos na tentativa de manter um compromisso entre o tempo de execução e qualidade da solução (em termos de erros inerentes aos métodos de discretização). Por se tratar de um problema conceitual, como extensamente mencionado, a ideia por detrás da escolha dos métodos de discretização foi de interferir o mínimo possível com os valores padrão do *solver* utilizado

e, ao mesmo tempo, garantir que o termo-fonte adicionado não impactasse negativamente o processo de obtenção da solução numérica.

4.2.4 Condições de contorno e parâmetros numéricos: neutrônica

O *milonga* convenientemente trata internamente as condições de contorno entre as diferentes regiões, ou materiais, do ponto de vista dos cálculos neutrônicos, poupando o usuário de certa quantidade de trabalho extra e evitando introdução de erros de modelagem. Desse modo, é suficiente definir as condições de contorno para as superfícies externas do domínio.

Tabela 6 – Condições de contorno da neutrônica

Fronteira	Tipo	Região correspondente
inlet *outlet*	*Vacuum*	Refrigerante
extremes	*Vacuum*	Revestimento e combustível
walls	*Mirror*	Refrigerante

As condições de contorno que podem ser usadas são de um fluxo imposto (Dirichlet), uma corrente imposta (Neumann) ou uma combinação linear destas (Robin). A aplicação de fluxo nulo na fronteira é feita definindo-se a condição de contorno nessa fronteira como *vacuum* dentro do *milonga*. Já a condição de corrente deve ter derivada zero na direção da normal à fronteira e é aplicada utilizando-se a palavra-chave *mirror*. Caso se deseje utilizar uma condição mista, deve ser utilizada a palavra-chave *robin* seguida de um argumento de ponto flutuante entre zero e um.

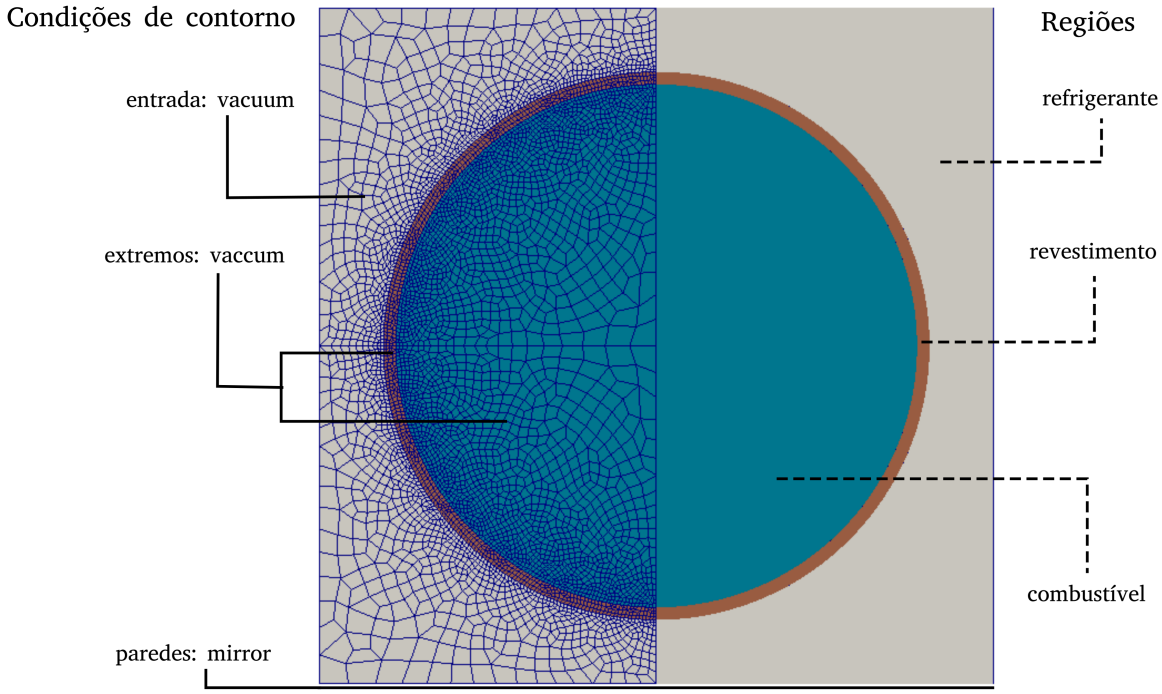
Uma explicação objetiva sobre quais são e como estão implementadas as condições de contorno no *milonga* está disponível no trabalho de Theler (THELER, 2013).

As condições de contorno utilizadas no modelo utilizado estão listadas na Tabela 6. Na figura 13 podem ser vistas na metade à esquerda a malha e as condições de contorno utilizadas enquanto na metade direita as regiões correspondentes a cada material.

4.3 Geração de seções de choque

A fim de verificar o impacto das variações de temperatura obtidas nos cálculos termo-hidráulicos no fluxo neutrônico e, portanto, na potência volumétrica, é necessário utilizar seções de choque fisicamente compatíveis com o problema a ser resolvido. A correta obtenção de seções de choque compatíveis com o problema a ser resolvido é fundamental para a solução do sistema físico proposto. As seções de choque são utilizadas no cálculo neutrônico pela aproximação por difusão como coeficientes dos distintos termos da equação diferencial que resolve o fluxo de nêutrons.

Figura 13 – Vista superior do modelo de combustível apresentando as distintas regiões e condições de contorno sobre a malha bidimensional.



O *milonga* utiliza mnemônicos específicos para cada coeficiente, de modo que estes possam ser representados, tanto como valores constantes, como por funções de temperatura, abordagem utilizada na modelagem nesta tese. Na tabela 7 estão apresentados os mnemônicos utilizados pelo *milonga* no modelo utilizado, bem como a divisão de grupos em energias[2].

Tabela 7 – Coeficientes da Equação de Difusão.

Coeficientes Mnemônicos		
Grupo 1: $>$ 0,625 MeV	Coeficiente de Difusão	D_1
	Seção de choque de absorção	Σ_{A1}
	Seção de choque de espalhamento	$\Sigma_{S1.2}$
	Nêutrons por fissão * Seção de choque de fissão	$\nu\Sigma_{F1}$
Grupo 2: $<$ 0,625 MeV	Coeficiente de difusão	D_2
	Seção de choque de absorção	Σ_{A2}
	Nêutrons por fissão * Seção de choque de fissão	$\nu\Sigma_{F2}$

Para a geração das seções de choque, um modelo com propriedades equivalentes ao modelo tridimensional a ser simulado foi definido. Esta equivalência foi garantida mantendo-se a proporção entre material físsil e moderador em ambos os modelos. Com base nessa proporção, todos os materiais foram definidos como anéis concêntricos.

[2] O valor limiar de 0,625 MeV para separação de nêutrons entre os grupos térmico e rápido é típico (SHER; FIARMAN, 1976)

Uma vez definida a geometria, foram definidas quatro diferentes temperaturas cobrindo a faixa de operação do reator TRIGA IPR-R1. A menor temperatura sendo a temperatura ambiente até a temperatura próxima à máxima do combustível do TRIGA IPR-R1 em operação a 250 kW (VELOSO, 2005), num grupo de quatro, apresentado na tabela 8.

Tabela 8 – Temperaturas de referência para os materiais.

Temperaturas dos materiais				
	T_1[K]	T_2[K]	T_3[K]	T_4[K]
Combustível	300	400	500	600
Revestimento	300	396	403	410
Refrigerante	300	308,5	317	341

A metodologia utilizada para a geração de seções de choque (REIS et al., 2015) foi desenvolvida pelo grupo de física de reatores do DEN/UFMG. Esta metodologia utiliza o código WIMSD-5B (HALSALL; TAUBMAN, 1986) para gerar seções de choque dependentes da temperatura para materiais específicos e para variado número de grupos de energia. No presente trabalho, foram utilizados dois grupos de energia separados em 0,625 MeV como apresentado na tabela 7. A composição dos materiais utilizados, em especial o combustível, que contem mais isótopos, se baseou na composição isotópica do reator TRIGA IPR-R1 no seu início de vida (BOL). Estas composições, massa e código de material utilizados na geração de seções de choque pelo WIMSD-5B estão na Tabela 9.

Tabela 9 – Composição dos materiais para geração de seções de choque com o WIMSD-5B (HALSALL; TAUBMAN, 1986; REIS et al., 2015).

	Material	Código	Densidade atômica
Combustível	H (no hidreto de zircônio)	5001	$3{,}7525\times10^{-02}$
	Zr	91	$3{,}7727\times10^{-02}$
	U^{235}	2235	$2{,}5744\times10^{-04}$
	U^{238}	8238	$1{,}0167\times10^{-03}$
Revestimento	Al	27	$6{,}0261\times10^{-02}$
Refrigerante	H	3001	$6{,}6653\times10^{-02}$
	O	6016	$3{,}3327\times10^{-02}$

As temperaturas utilizadas na geração das seções de choque e as seções de choque resultantes são apresentadas nas tabelas [3] 10, 11 e 12 respectivamente para o combustível, revestimento e moderador.

Devido a uma limitação do código WIMSD-5B na geração de seções de choque de espalhamento para cada material separadamente, as seções de choque de espalhamento

[3] Os valores tabelados estão fora das grandezas típicas por terem sido multiplicados por cm ($10^{-2}m$) em razão da malha utilizada para os cálculos ter sido definida nesta unidade.

Tabela 10 – Temperaturas para o combustível.

Combustível				
Parâmetro	$300K$	$400K$	$500K$	$600K$
D_1	0,0108467	0,0108430	0,0108398	0,0108396
Σ_{A1}	0,771195	0,784938	0,797017	0,807916
$\Sigma_{S1.2}$	3,3991	3,4329	3,4264	3,4206
$\nu\Sigma_{F1}$	0,500640	0,500722	0,500791	0,500853
D_2	0,0031356	0,0031729	0, 0032211	0,0032789
Σ_{A2}	11,1031	10,7948	10,3798	9,88378
$\nu\Sigma_{F2}$	20,3145	19,7339	18,9512	18,0157

Tabela 11 – Temperaturas para o revestimento

Revestimento				
Parâmetro	$300K$	$396K$	$403K$	$410K$
D_1	0,0299762	0,0299738	0,0299717	0,0299698
Σ_{A1}	0,0289086	0,0289016	0,0288954	0,0288899
$\Sigma_{S1.2}$	3,3991	3,4329	3,4264	3,4206
D_2	0,0376009	0,0376759	0,0377742	0,0379175
Σ_{A2}	0,879749	0,865116	0,845895	0,817942

Tabela 12 – Temperaturas para o refrigerante

Moderador				
Parâmetro	$300K$	$308,5K$	$317K$	$341K$
D_1	0,0126618	0,0126631	0,0126642	0,0126652
Σ_{A1}	0,0291631	0,0291502	0,0291388	0,0291285
$\Sigma_{S1.2}$	3,3991	3,4329	3,4264	3,4206
D_2	0,0021242	0,0021317	0021428	0021649
Σ_{A2}	1,43474	1,41539	1,39081	1,34991

foram geradas para uma homogeneização dos materiais em uma única célula, sendo esta a razão dos valores serem os mesmos para todos os materiais.

As seções de choque obtidas foram então escritas no formato *milonga* como funções de uma variável dependente da temperatura. Como o milonga tem informação das células em relação à malha, a temperatura de cada célula é argumento da função de uma variável dependente da temperatura. Se a temperatura não está exatamente tabulada, o *milonga* provê interpolação linear por padrão.

Esta característica do *milonga* permite o cálculo neutrônico independente por elemento de volume, sendo possível obter variações detalhadas de acordo com a granularidade da malha utilizada pelo modelo.

5 Resultados (ou Prova do Conceito)

A utilização da expressão "prova de conceito"[1] no título deste capítulo não é arbitrária. Muito pelo contrário. "prova de conceito" significa a execução (ou implementação) de certa ideia ou método de forma a demonstrar que tal ideia ou método funcionam. Melhor dizendo, é a demonstração de um princípio com o objetivo de verificar seu potencial prático.

Sendo assim, de forma a testar a implementação desenvolvida, foram feitos dois conjuntos separados de simulações, seguindo as modelagens físicas e numéricas apresentadas no Capítulo 4 desta tese. De modo a capturar os efeitos da variação da temperatura dos distintos materiais no fluxo neutrônico e, consequentemente, na potência gerada, foram feitas simulações para três diferentes potências nominais. Os valores de potência utilizados equivalem ao elemento com maior fator de potência (VELOSO, 2005) do núcleo do reator TRIGA Mark I IPR-R1 (elemento B6), considerando todo o núcleo do reator em operação à potência de 50 kW, 100 kW e 200 kW. Estas simulações foram feitas para dois casos distintos: um sistema não-acoplado e para o sistema acoplado desenvolvido, totalizando seis simulações. Na tabela 13 estão classificados os conjuntos de simulações realizados.

Tabela 13 – Casos e potências simuladas

Caso	Potências (equivalente núcleo)		
Não-acoplado	1,98 kW (50 kW)	3,97 kW (100 kW)	7,93 kW (200 kW)
Acoplado	1,98 kW (50 kW)	3,97 kW (100 kW)	7,93 kW (200 kW)

Cabe lembrar que para esta prova de conceito, foi utilizada apenas uma única malha para todos os casos.

5.1 Caso não-acoplado

O conjunto de simulações realizadas para este caso foi feito de modo a ser referência para o outro conjunto de simulações. Apesar de chamado "não-acoplado", pode ser considerado um caso "inicialmente acoplado". Esta afirmação ficará mais clara observando a lista de etapas para a geração da condição inicial de potência para o sistema:

[1] Do inglês "proof of concept". Definição do dicionário *Oxford:[mass noun] Evidence, typically deriving from an experiment or pilot project, which demonstrates that a design concept, business proposal, etc. is feasible:'In academia, he says, a narrowly focused solution is acceptable as a proof of concept.'*

1. Execução do *OpenFOAM* com temperaturas iniciais homogêneas (temperatura ambiente) e distribuição nominal de potência homogênea;
2. Pós-processamento destas simulações para obtenção da temperatura média em cada material;
3. Obtenção das seções de choque para as temperaturas médias a partir da função de interpolação de temperaturas de referência (Tabela 8 do Capítulo 4);
4. Execução do *milonga* com as seções de choque correspondentes a cada temperatura média;
5. Pós-processamento destas simulações para adaptação das distribuições de potência obtidas pelo *milonga* no formato para condição inicial do problema no *OpenFOAM*;

As temperaturas médias utilizadas e os fatores de multiplicação obtidos na geração da distribuição inicial de potências, de acordo com as etapas listadas anteriormente, são apresentados na tabela 14.

Tabela 14 – Temperaturas de referência para obtenção de seções de choque para o cálculo não-acoplado.

	Temperaturas [K]		
Potência [kW]	Refrigerante	Revestimento	Combustível
1,98	303,56	327,20	339,80
3,97	307,15	354,54	379,77
7,93	310,09	375,42	422,94

Os valores de fator de multiplicação (k_{eff}) nos três cálculos neutrônicos *standalone* são de $1,15370$ para condições à potência de 1,98 kW, $1,15264$ para condições à potência de 3,97 kW e $1,14704$ para condições à potência de 7,93 kW.

O caso não-acoplado nada mais é do que uma simulação termo-hidráulica utilizando como distribuição de potência inicial o resultado obtido pelos cálculos neutrônicos com seções de choque constantes. Que por sua vez, foram obtidas para temperaturas médias nos materiais calculadas pela termo-hidráulica com distribuição de potência homogênea.

As distribuições de potências obtidas para os volumes de combustível podem ser vistas na Figura 14.

As distribuições de potência axial obedeceram, para os três casos, a perfis cossenoidais, como pode ser visto na Figura 15. Estes são os perfis esperados para os combustíveis modelados (VELOSO, 2005).

Os perfis de temperatura obtidos axialmente, Figura 16, apresentam, novamente para os três casos, uma diferença entre as extremidades do combustível. Essa diferença

Figura 14 – Distribuição de potência para os três casos simulados.

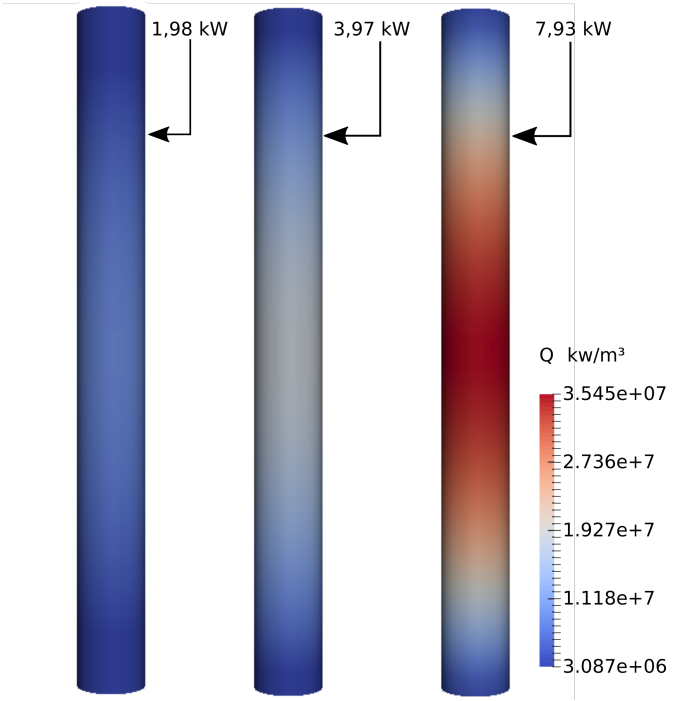
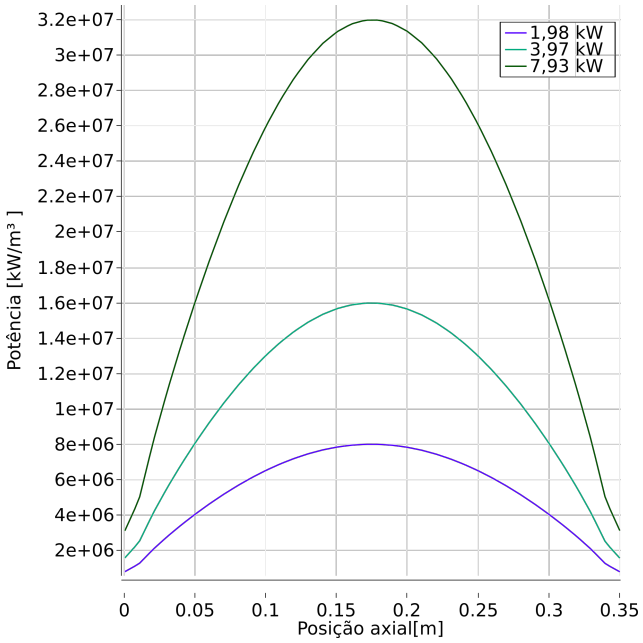


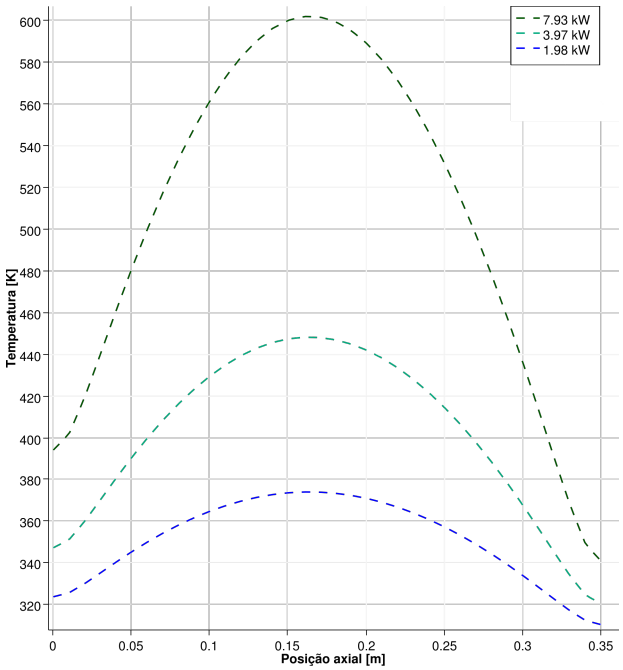
Figura 15 – Perfil de potência axial para os três casos simulados.



é provocada pela escoamento do refrigerante (água) no modelo. A água entra no sistema à temperatura constante de $300K$ e, no seu trajeto, absorve energia do sistema. Essa absorção é maior na entrada (*inlet*), levando a um maior resfriamento do sistema neste ponto do escoamento. Esse efeito ocorre em todos os casos, apenas com diferença nas temperaturas de cada material.

Radialmente, os perfis de temperatura, seguindo o mesmo padrão para os três casos simulados, possuem descontinuidades, como pode ser visto na Figura 17 (entre 0 m e 0,005

Figura 16 – Perfil de temperaturas axiais para os três casos simulados.



m e 0,04 m e 0,045 m). Estes degraus indicam mudança no gradiente de temperaturas na interface entre diferentes materiais, devido às diferentes condutividades térmicas de cada um.

Os resultados das simulações apresentadas para os três casos não-acoplados formam o conjunto de resultados de referência. Eles servem de base de comparação para as simulações acopladas, com objetivo de verificar se os resultados acoplados se diferenciam destes e em que medida ocorrem estas diferenças.

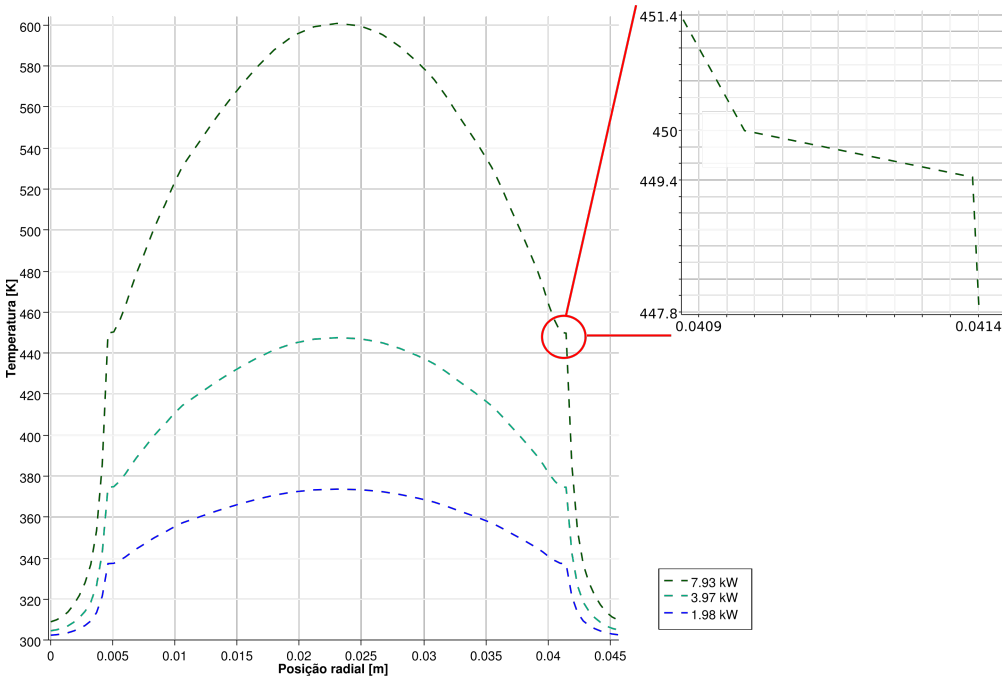
5.2 Caso acoplado

As simulações acopladas foram realizadas exatamente nas mesmas condições das suas homólogas não-acopladas. Os resultados destas são apresentados em comparação com os resultados de referência para visualização das diferenças entre elas.

Diferentemente dos cálculos não-acoplados, na simulação acoplada ocorre a variação do fluxo neutrônico. Essa variação, originada da variação de temperatura no sistema e da realimentação das seções de choque, ocorre nos três casos acoplados, cada um simulado numa potência.

Para as simulações a 1,98 kW, as diferenças no fluxo são as mais sutis. Com potência mais baixa, há menor diferença nas temperaturas entre o caso acoplado e o não-acoplado. Como as seções de choque variam de acordo com a temperatura, é menor a diferença entre seções de choque. Por consequência, menor a diferença no fluxo neutrônico. Outro fator que leva à similaridade entre os fluxos é a baixa granularidade da malha na

Figura 17 – Perfil de temperatura radial para os três casos simulados. No detalhe o perfil de temperatura nas interfaces entre os três materiais.



direção axial (apenas 35 camadas de elementos, como apresentado na Tabela 1).

Na Figura 18 são apresentados os fluxos axiais acoplados e não acoplados nas simulações à potência mais baixa, de 1,98 kW. A diferença entre os fluxos é melhor percebida no corte radial, apresentado na Figura 19. Além de uma maior variação radial devido à moderação de nêutrons nas paredes, e consequente aumento do fluxo térmico, a malha é mais refinada radialmente. Com isso, é possível captar mais detalhes do fluxo neutrônico.

As mesmas observações feitas para os fluxos relativos na menor potência são válidas para as simulações feitas à potência de 3,97 kW. Novamente, as diferenças entre os fluxos relativos são melhor observáveis no corte radial do que no corte axial. Entretanto, as diferenças entre os fluxos começam a aumentar de acordo com o aumento da potência. Os fluxos relativos axiais acoplados e não-acoplados para potência de 3,97 kW podem ser observados na Figura 20, enquanto os fluxos relativos radiais para a mesma potência podem ser observados na Figura 21.

As diferenças entre os fluxos relativos em ambas as simulações passam a ser melhor observadas nas simulações de maior potência, a 7,93 kW. Novamente, a variação nas temperaturas impacta nos cálculos acoplados, uma vez que as seções de choque são periodicamente realimentadas. É possível observar na Figura 22, que apresenta o perfil axial, uma sutil sobreposição entre as curvas, demonstrando um achatamento na curva referente aos cálculos acoplados. Isso se explica pelas propriedades do combustível modelado, que diminui sua capacidade de absorção de nêutrons térmicos com o aumento da tempera-

Figura 18 – Fluxos relativos axiais entre simulação acoplada e não acoplada para potência de 1,98 kW.

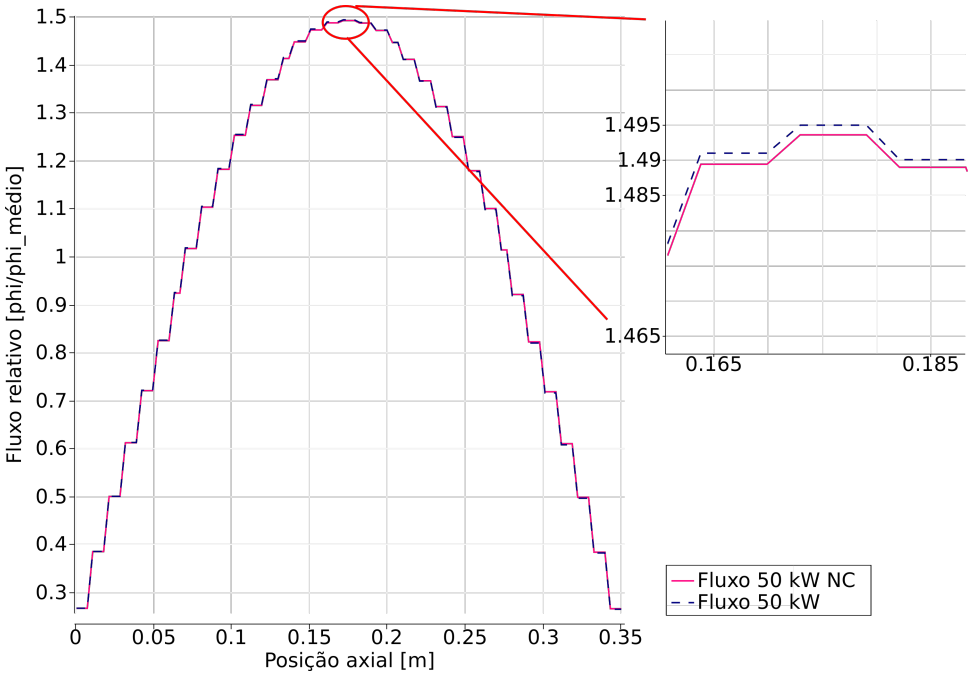
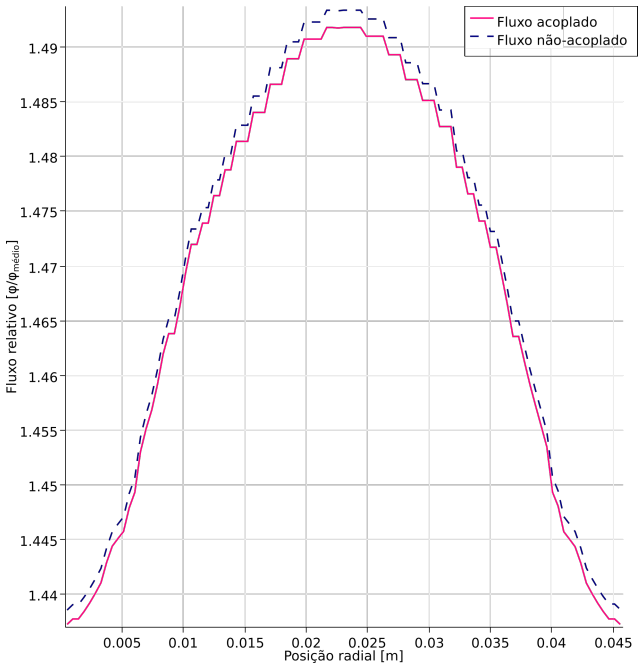


Figura 19 – Fluxos relativos radiais entre simulação acoplada e não acoplada para potência de 1,98 kW.



tura. Ao mesmo tempo, nas extremidades da curva, as temperaturas são menores que no caso não-acoplado, levando a um relativo aumento no fluxo. O fluxo acoplado reflete tais mudanças no seu perfil.

As diferenças entre os fluxos relativos radiais, para a potência de 7,93 kW, são facilmente visíveis na Figura 23.

Figura 20 – Fluxos relativos axiais entre simulação acoplada e não acoplada para potência de 3,97 kW.

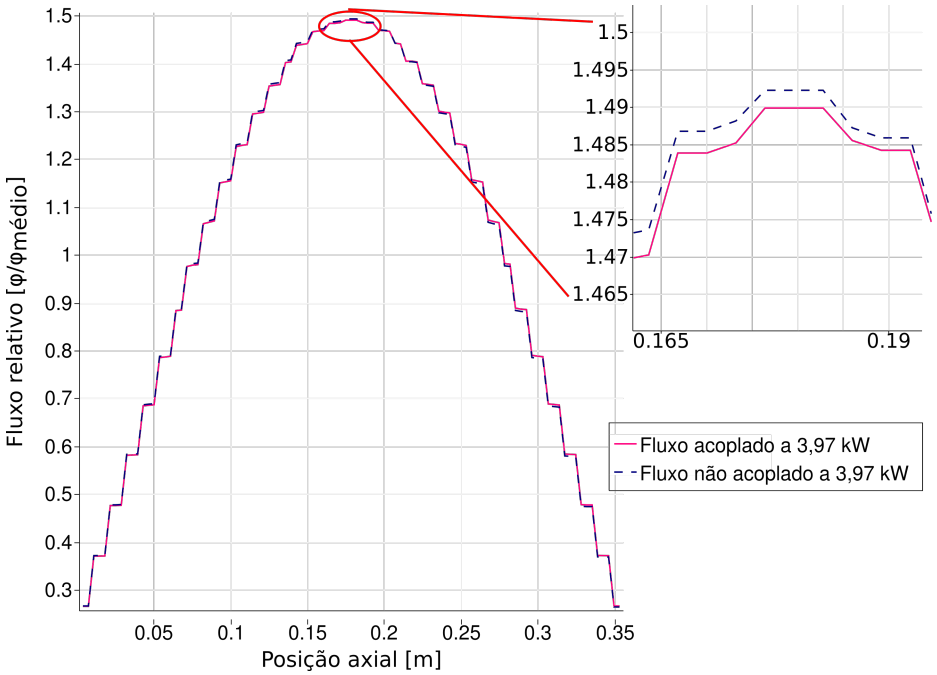
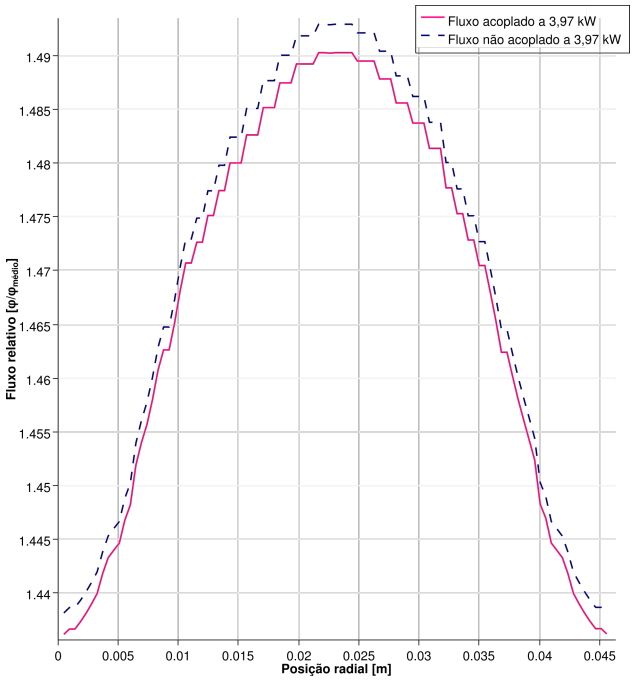


Figura 21 – Fluxos relativos radiais entre simulação acoplada e não acoplada para potência de 3,97 kW.



Os perfis de potência para os cálculos acoplados e não-acoplados seguem, como é de se esperar, o padrão de comportamento dos perfis de fluxo. A solução da equação de difusão é o fluxo neutrônico e, a partir dele, é calculada a potência volumétrica baseada em um valor de referência para a potência.

Figura 22 – Fluxos relativos axiais entre simulação acoplada e não acoplada para potência de 7,93 kW.

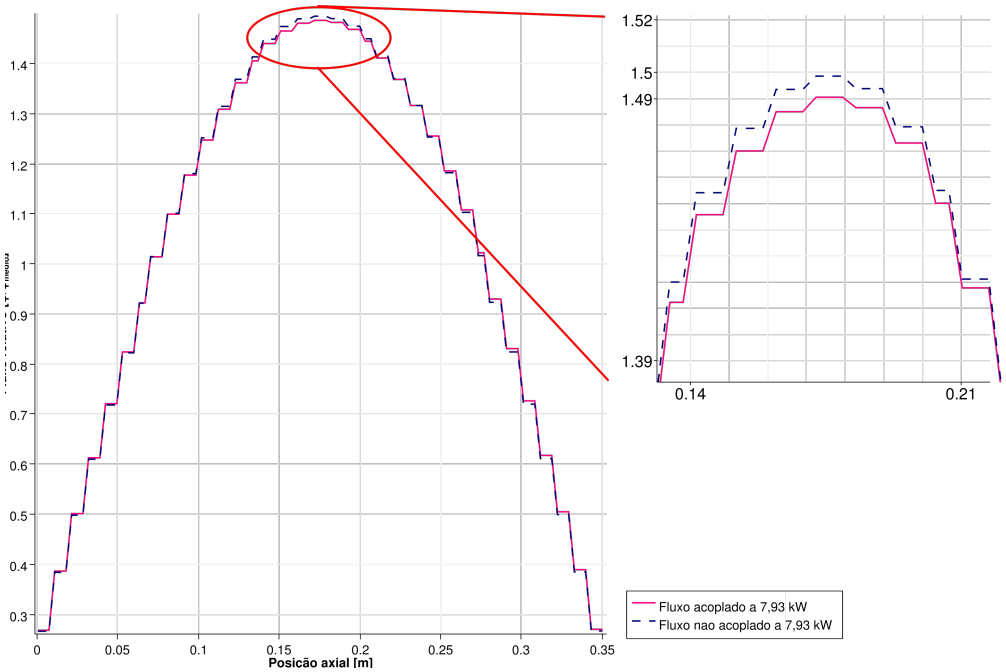
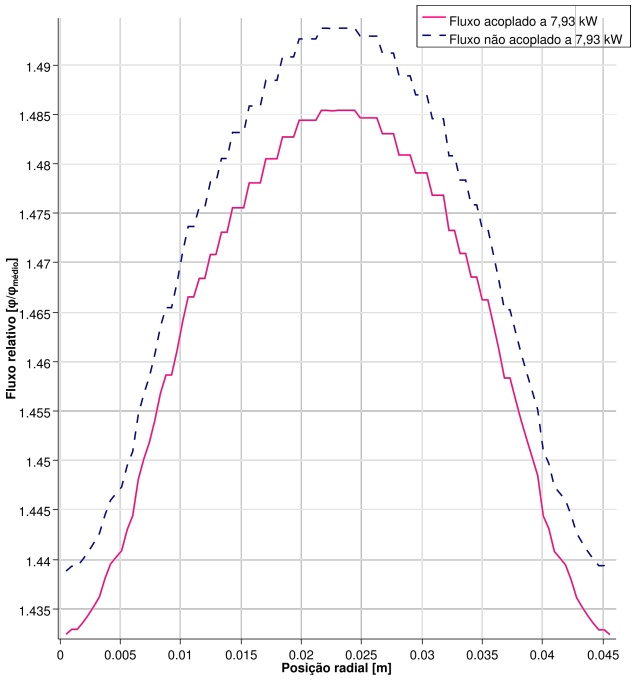


Figura 23 – Fluxos relativos radiais entre simulação acoplada e não acoplada para potência de 7,93 kW.



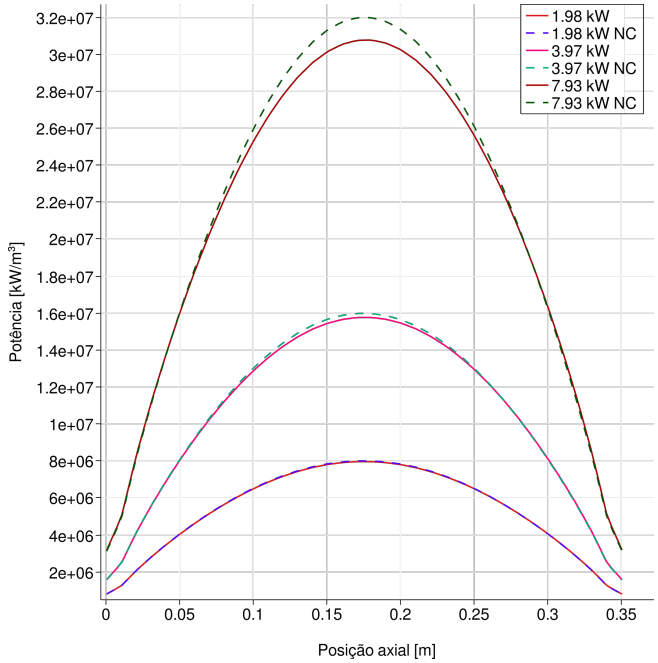
É a potência[2] , que entra como termo-fonte na equação na energia resolvida pela termo-hidráulica, que leva ao aquecimento dos materiais que, por sua vez, impactam no cálculos de seções de choque. As diferenças entre os perfis de potência obtidos nos

[2] A palavra potência é indistintamente utilizada neste capítulo como potência propriamente dita, em W ou como sinônimo de densidade de potência, dada em W/m^3, de acordo com o contexto em que é usada.

dois conjuntos de simulações, acopladas e não-acopladas, são apresentados na Figura 24. Observando esta Figura, é possível perceber que as diferenças entre os casos acoplados e não acoplados aumenta de acordo com o aumento da potência.

Além da diferença em valores absolutos - sutil, para potências mais baixas - entre as curvas obtidas pelos cálculos acoplados e não acoplados, é possível observar uma mudança no formato das curvas. O perfil de potência para o cálculo acoplado tem um pico abaixo do pico da simulação não acoplada, enquanto nas extremidades da curva, os perfis estão sobrepostos[3] (Figura 24).

Figura 24 – Curvas de potência axiais para os casos acoplados e não acoplados.



As curvas de temperaturas para as mesmas simulações são apresentadas nas Figuras 25 e 26, respectivamente para as distribuições radial e axial.

As diferenças entre temperaturas para simulações acopladas e não acopladas são esperadas, uma vez que há variações na distribuição de potências. As curvas apresentadas nas Figuras 24, 25 e 26 são as mesmas apresentadas para os casos não acoplados acompanhadas das curvas do caso acoplado. Os fenômenos físicos e suas explicações para o formato das curvas são os mesmos, já que as simulações são iguais a menos das potências e do fato de serem ou não acopladas.

O principal resultado obtido nesta tese consiste precisamente nas diferenças, pequenas em casos de baixa potência e maiores a potências mais elevadas, entre as curvas acopladas e não acopladas. Em suma, com retroalimentação devida ao acoplamento, a

[3] Na posição axial esta diferença não é visível sem aproximação. A diferença em potência é da ordem de dezenas de kW/m^3 para medidas da ordem de $10^7 kW/m^3$. Para a maior potência simulada, o perfil acoplado está acima nas extremidades.

relação entre os fenômenos físicos termo-hidráulicos e neutrônicos é representada e, como apresentado, gera resultados diferentes dos cálculos realizados separadamente.

Figura 25 – Curvas de temperatura axiais para os casos acoplados e não acoplados (NC).

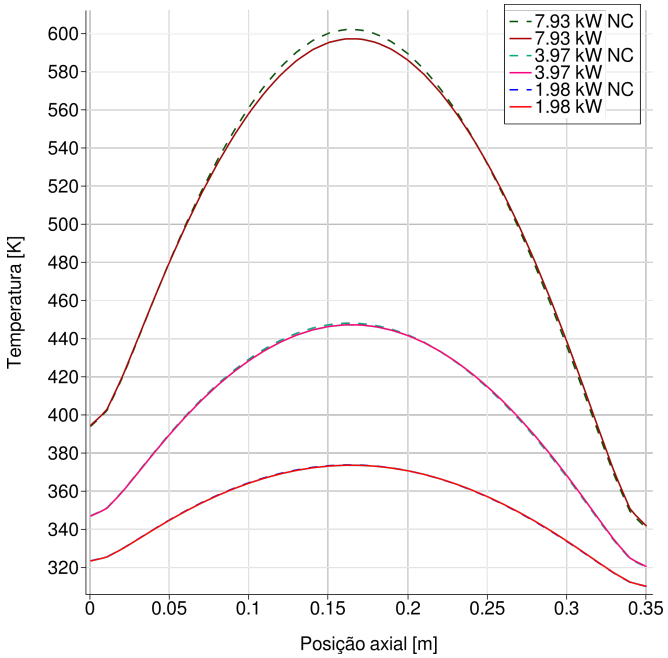
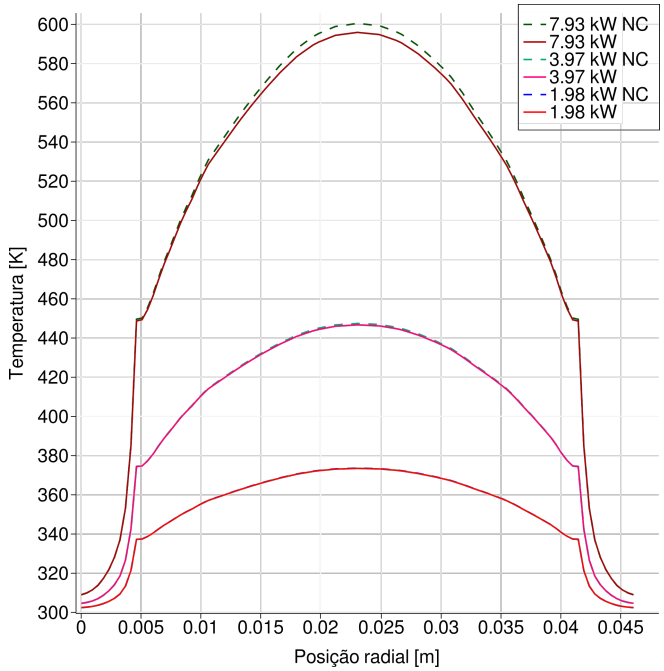


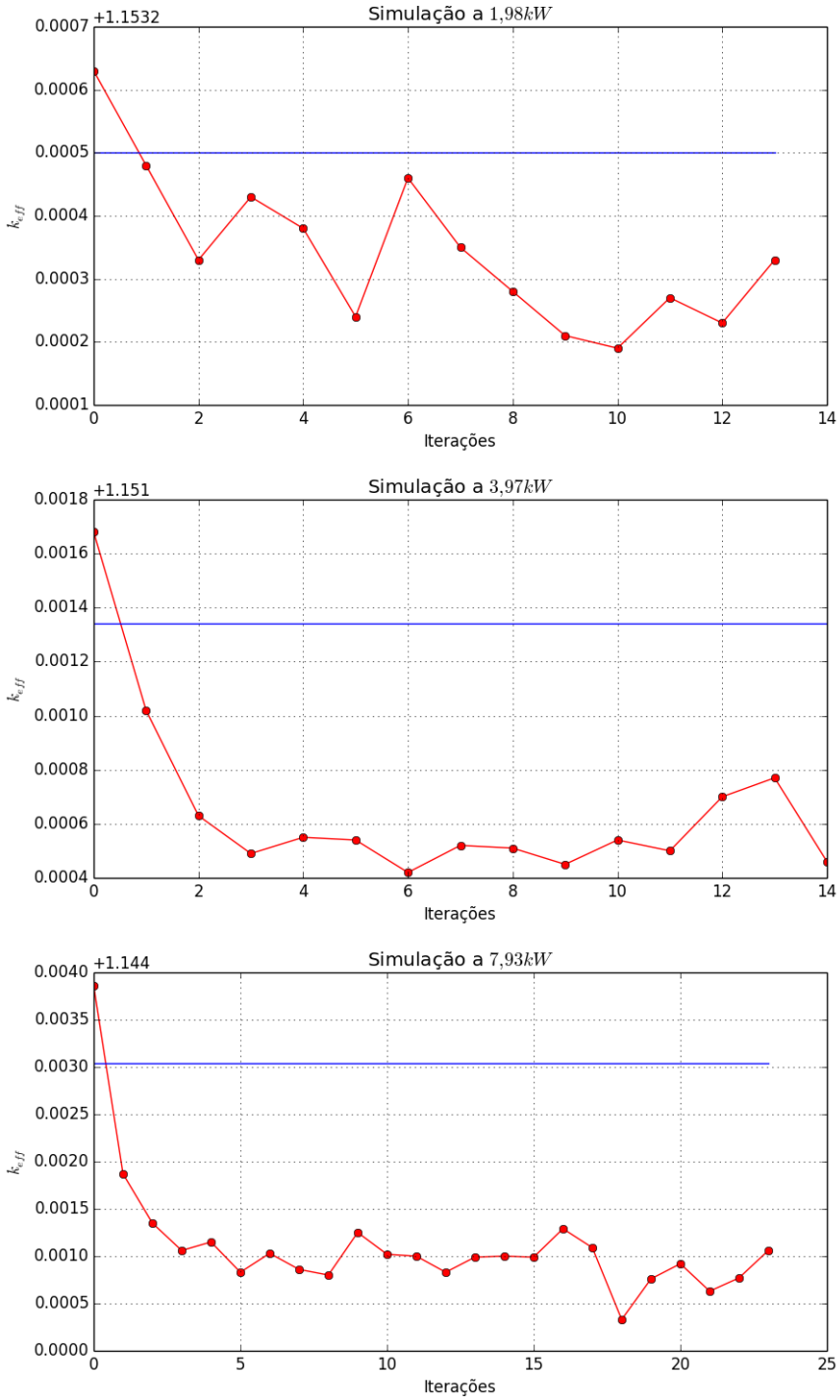
Figura 26 – Curvas de temperatura radiais para os casos acoplados e não acoplados (NC).



Nos cálculos acoplados, a cada execução do *milonga* são utilizadas novas seções de choque ajustadas a partir de tabelas de seções de choque previamente geradas para quatro diferentes temperaturas. As seções de choque utilizadas nos cálculos são obtidas da interpolação das temperaturas tabeladas em função das temperaturas de cada célula.

Em cada uma destas execuções, além do fluxo neutrônico, são calculados os fatores de multiplicação (k_{eff}) efetivos daquele "estado" do sistema.

Figura 27 – Variação dos fatores de multiplicação efetivo (k_{eff}) nas três simulações acopladas.



À medida em que as iterações com o *milonga* ocorrem, o fator de multiplicação calculado para o sistema varia, oscilando. Esta oscilação ocorre devido à iteração entre o sistema neutrônico e termo-hidráulico. É possível perceber que a partir de uma potência, as temperaturas aumentam na interação seguinte, que leva, por sua vez, a uma diminuição

Tabela 15 – Fatores de multiplicação obtidos para geometria similar com código PARCS.

Temperaturas de referência	Fator de multiplicação (k_{eff})
T1	1,320175
T2	1,314753
T3	1,308179
T4	1,302641

na reatividade (propriedade deste tipo de elemento combustível), que leva a uma diminuição na potência, que implica em variação da temperatura, que afeta a reatividade e assim por diante. O fim da simulação se dá por número fixo de iterações. O número de iterações é definido pela simulação termo-hidráulica, sendo 1500 passos para as simulações a 1,98 kW e 3,97 kW e 2400 passos para a simulação a 7,93 kW. A diferença é simplesmente devida ao número de iterações para convergência. Como é fixada em 100 a razão de iterações neutrônicas e termo-hidráulicas, em 1500 iterações termo-hidráulicas, são feitos 15 cálculos acoplados para os casos de menos potência e 24 para o de potência mais alta.

Na Figura 27 estão apresentados três gráficos, um para cada simulação acoplada. As linhas constantes (em azul) indicam o valor do fator de multiplicação calculado na execução separada do *milonga* com o conjunto inicial de seções de choque utilizadas no cálculo acoplado. As curvas em vermelho apresentam os fatores de multiplicação obtidos a cada iteração entre termo-hidráulica e neutrônica (as linhas entre pontos são apenas para indicação da diferença entre iterações seguidas, já que os resultados são discretos).

É possível ver que, em todos os casos, o fator de multiplicação cai imediatamente após a segunda iteração com a neutrônica. Na primeira iteração entre a termo-hidráulica e a neutrônica, as temperaturas do sistema vêm subindo devido à distribuição de potências previamente calculada. A partir desta primeira iteração, a realimentação do campo de potências já impacta na distribuição de temperaturas, o que fica claro na segunda iteração. A partir deste ponto, o sistema começa a apresentar pequenas oscilações ao redor de um fator de multiplicação de convergência.

Um conjunto similar de seções de choque, isto é, gerado às mesmas condições de temperatura da tabela de seções de choque (ver tabela 8) mas para um *cluster* de elementos combustíveis ao invés de um único combustível utilizando o código WIMS-5B foi utilizado no código PARCS para efeito de referência. Os valores para o fator de multiplicação obtidos neste caso, e apresentados na tabela 15, divergem dos obtidos pelos cálculos acoplados em até 13%, sendo menores para os cálculos acoplados.

Tais diferenças podem ser explicadas pela diferença nas seções de choque de espalhamento, distintas em ambas as simulações mas, especialmente, pelas diferenças em temperatura. Entretanto, esses fatores não são suficientes a magnitude das diferenças nos fatores de multiplicação. Sendo assim, fazem-se necessárias mais investigações na forma

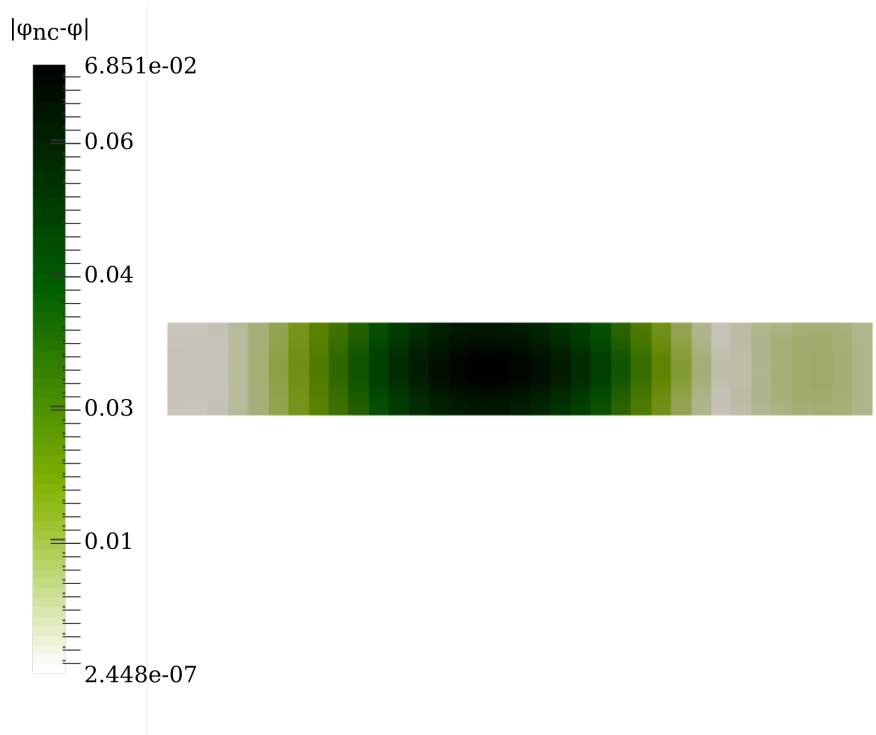
com que o sistema acoplado, em especial o código *milonga* realiza seus cálculos, em especial na forma como são tratadas condições de contorno.

As figuras 28 a 33 apresentam de forma visual, a partir da geometria utilizada, as diferenças entre fluxos de nêutrons e potências obtidas para os cálculos acoplados e não acoplados.

As diferenças entre fluxo são apresentadas para toda a geometria, incluindo materiais nos quais não ocorrem fissões: revestimento e refrigerante. Nestes casos, os valores apresentados são as diferenças entre células equivalentes nas simulações acopladas e não acopladas, sendo os dados obtidos pelas simulações feitas com o *milonga*. É possível notar, observando as figuras 28, 29 e 30 que não há interpolação entre células adjacentes.

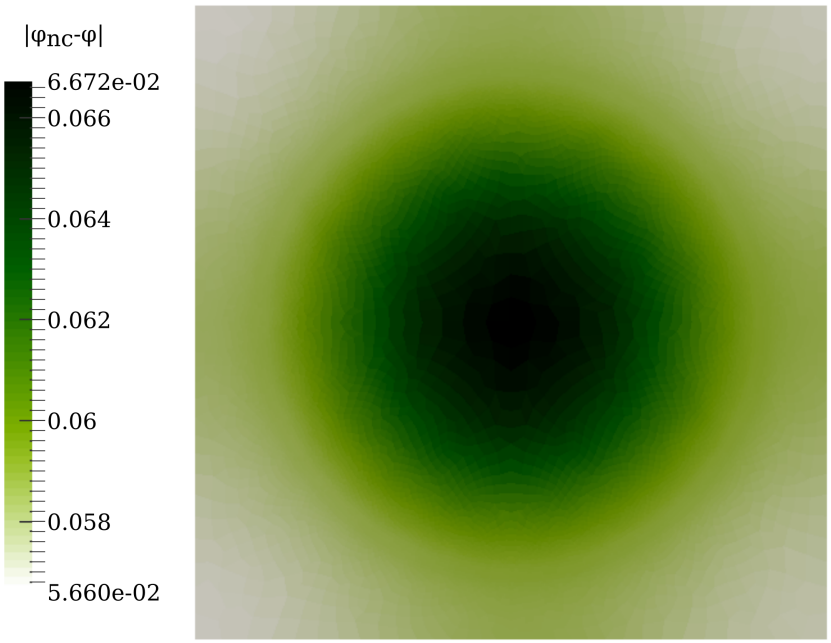
As diferenças entre potências, apresentadas nas figuras 31, 32 e 33, estão, para fins de exibição, apresentadas de forma interpolada entre células equivalentes. Estes valores foram obtidos via pós-processamento dos dados utilizados pelo *OpenFOAM*. O único material apresentado nestas figuras é o combustível, que possui, em sua constituição, material físsil. Como não ocorrem fissões nos outros materiais simulados, a potência gerada nestes é zero, de modo que estes materiais são omitidos na representação de potência.

Figura 28 – Diferença de fluxo entre o cálculo acoplado e não acoplado (7,93 kW): corte axial



Os resultados aqui apresentados não são suficientes para nenhuma afirmação sobre a efetiva capacidade de cálculos deste sistema acoplado para sistemas mais complexos do que o modelo utilizado como, por exemplo, uma seção de um núcleo completo de um reator do tipo TRIGA ou um elemento combustível reduzido de um reator do tipo PWR.

Figura 29 – Diferença de fluxo entre o cálculo acoplado e não acoplado (7,93 kW): corte radial



Porém, estes mesmos resultados provam que conceitualmente o acoplamento neutrônico termo-hidráulico proposto e desenvolvido nesta tese é fisicamente e computacionalmente consistente, além de funcional.

Figura 30 – Diferença de fluxo entre o cálculo acoplado e não acoplado (7,93 kW): vista isométrica

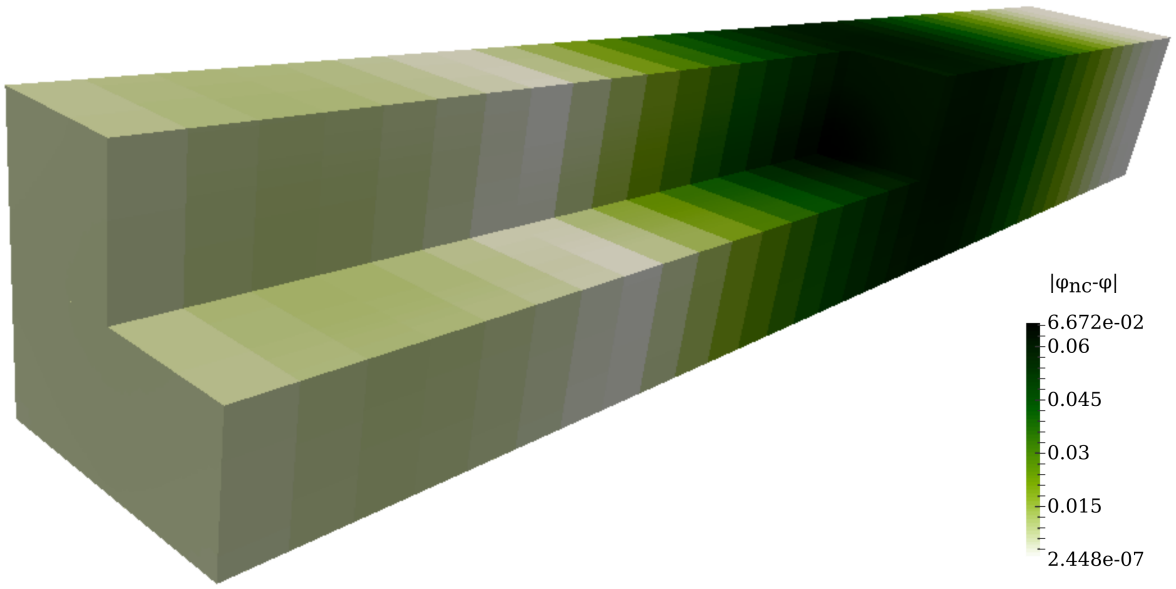


Figura 31 – Diferença de potência entre o cálculo acoplado e não acoplado (7,93 kW): corte axial

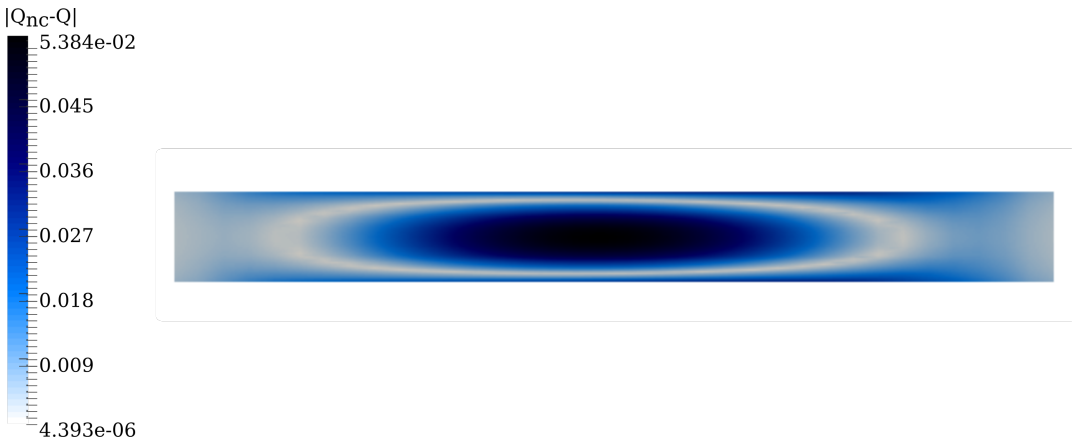


Figura 32 – Diferença de potência entre o cálculo acoplado e não acoplado (7,93 kW): corte radial

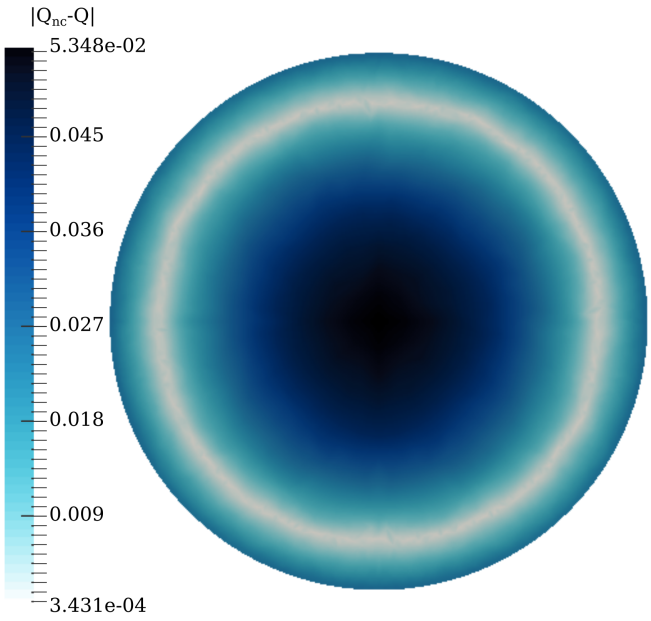
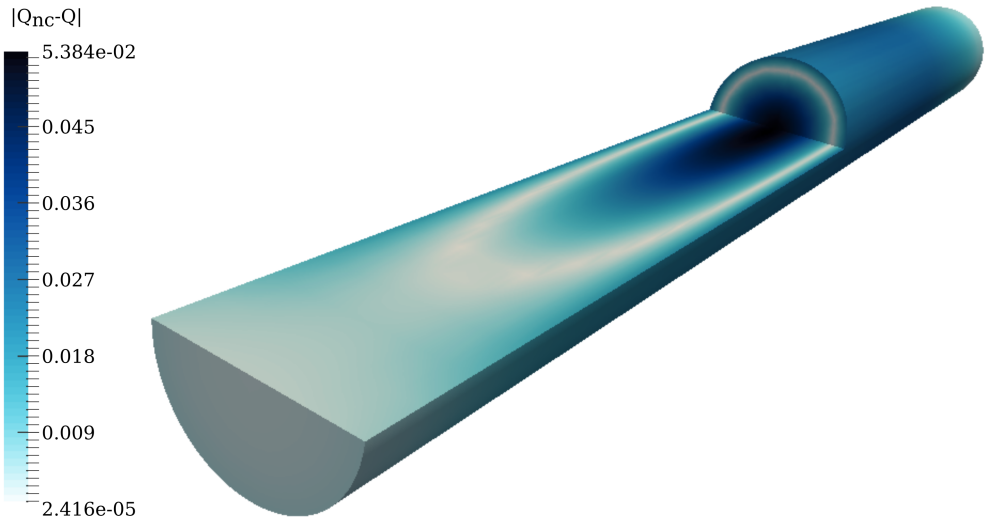


Figura 33 – Diferença de potência entre o cálculo acoplado e não acoplado (7,93 kW): vista isométrica



6 Conclusões

Nesta tese foi desenvolvido um sistema livre, aberto e gratuito para cálculos neutrônicos e termo-hidráulicos de forma acoplada utilizando malhas idênticas. Além do sistema propriamente dito, foi também desenvolvida uma metodologia para a construção deste sistema acoplado baseada em sistemas já existentes livres e utilizando memória compartilhada como forma de intercâmbio de dados.

Este sistema é inovador ao utilizar o *framework OpenFOAM*, baseado em volumes finitos, como ferramenta de cálculos termo-hidráulicos e o código de cálculos de física de reatores aberto *milonga* como ferramenta para cálculos neutrônicos. A utilização de memória compartilhada como espaço de intercâmbio de dados, apesar de conhecida (MACIEL, 2011; THELER et al., 2013), é praticamente ignorada pela comunidade de Engenharia Nuclear Computacional na implementação de sistemas acoplados. Tal situação é de se estranhar, já que o uso de memória compartilhada permite a comunicação entre os sistemas acoplados de forma confiável, robusta e tão rápida quanto qualquer acesso à memória do computador. O sistema desenvolvido nesta tese faz uso de memória compartilhada, sendo o único sistema, no melhor conhecimento do autor desta tese, de cálculos acoplados baseados em volumes finitos.

O sistema desenvolvido, por acoplar dois sistemas de cálculos que utilizam a formulação de volumes finitos para solução de sistemas de equações diferenciais, explora a característica de domínio idêntico para os dois sistemas. Em outras palavras, os domínios de solução, ou malhas, são os mesmos para ambos o que evita a utilização de funções de mapeamento, ou seja, evita o *overhead* do mapeamento geométrico, um problema de geometria computacional não-trivial no caso de malhas não-estruturadas.

Há, naturalmente, vantagens e desvantagens no uso de malha idêntica. Uma desvantagem é o custo computacional para malhas refinadas. Além disso, o fluxo neutrônico geralmente não precisa ser conhecido em detalhes na maioria dos cálculos de núcleo de reatores, o que fundamenta eventuais críticas ao custo computacional do sistema desenvolvido. Entretanto, o objetivo dos cálculos por volumes finitos é permitir identificar pequenas ocorrências imperceptíveis em cálculos por métodos menos granulares. O sistema desenvolvido nesta tese, cuja prova conceitual foi feita com uma malha relativamente pobre, permite conhecer fenômenos locais calculados a partir de elementos da multi-física envolvida, e não apenas de perfis de potência genéricos ou temperaturas médias, por exemplo.

O desenvolvimento desta tese levou também a contribuições fora do seu escopo principal, ou seja, para além do sistema acoplado propriamente dito. Incidentalmente,

ao acessar o código-fonte dos sistemas utilizados para permitir seu uso acoplado, foram encontradas falhas de implementação em ambos os sistemas utilizados para neutrônica e termo-hidráulica. Os erros encontrados no *OpenFOAM* foram reportados à comunidade. Os erros encontrados no *milonga* não só foram reportados ao seu autor como, em um caso crítico, foi feita a correção diretamente e então enviado o *patch* ao autor. Essa correção foi incorporada numa versão posterior do *milonga*. Estas contribuições incidentais se dão devido à própria natureza do desenvolvimento baseado em *software* livre, que prevê e se baseia em contribuições e trocas entre a comunidade de usuários, desenvolvedores e autores.

É dessa corrente de desenvolvimento que surge outra característica inovadora do sistema desenvolvido nesta tese: ele é construído exclusivamente a partir de sistemas abertos e livres[1].

6.1 Trabalhos Futuros

Em um trabalho de prova de conceito, a discussão sobre perspectivas futuras ganha uma importância adicional. O sistema desenvolvido nesta tese possui diversas limitações, apresentadas na seção 3.2.5. Uma vez provado o conceito, o próximo passo consiste em estender o alcance do sistema a casos mais complexos. Para isso devem ser superadas as limitações técnicas apresentadas. Além disso, sendo o sistema acoplado nada mais do que ambos os sistemas utilizados modificados internamente, eventuais melhorias no sistema acoplado passam, obrigatoriamente, por evolução, modificações ou expansão nos sistemas utilizados. Sendo assim, nesta seção são apresentadas formas de superar as limitações já apresentadas bem como opções adicionais com o objetivo de ampliar a utilização do sistema acoplado atual.

Hoje, tanto o *OpenFOAM* quanto o *milonga* são distribuídos exclusivamente para o sistema operacional Linux (BRITANNICA., 2014). Para fazer com que sejam utilizados em outras plataformas, são necessárias modificações em seus códigos-fonte e em seus sistemas de compilação e instalação. O *OpenFOAM*, dada sua ampla rede de usuários, já possui iniciativas neste sentido. O *milonga*, entretanto, possui apenas pequenas partes adaptadas à compilação multiplataforma. De forma a expandir o alcance de ambos, é necessário fazer com que ambos sejam passíveis de compilação multiplataforma. Há, disponíveis, ferramentas com esse intuito (MARTIN; HOFFMAN, 2008). Com ambos os sistemas funcionais em multiplataforma, as modificações no sistema acoplado seriam pequenas e factíveis.

[1] A metodologia utilizada na geração de seções de choque utiliza um *software* restrito. Entretanto, a geração de seções de choque, apesar de fundamental para os cálculos neutrônicos, não é, propriamente dita, parte do sistema acoplado, já que qualquer ferramenta pode ser utilizada para este propósito. Neste tema, começam também a surgir opções em *software* livre visando a geração de seções de choque em multi-grupos a partir de bibliotecas contínuas (MCFARLANE et al., 2016).

Um trabalho futuro que traria enormes impactos na capacidade de atacar grandes problemas em relação malhas refinadas com muitos elementos é a implementação de uma versão paralela do *milonga*. A implementação em paralelo dos cálculos em volumes finitos e de cálculo das matrizes do problema de autovalores permitira uma escalabilidade fundamental para os cálculos neutrônicos. Na versão atual, todo o processo é feito sequencialmente: toda a malha é percorrida, todas as interpolações célula a célula feitas, então é construída a matriz de solução e então resolvido o sistema na matriz. A execução em paralelo permite dividir o domínio em cada núcleo[2] e resolver paralelamente uma escala menor do problema. Isso se aplica tanto ao problema de construção das matrizes a partir dos volumes finitos quanto à solução da matriz propriamente dita. Algoritmos de solução de matrizes de vários tipos, amplamente conhecidos e utilizados, estão disponíveis sendo alguns dos mais robustos deles (HERNANDEZ et al., 2005; BALAY et al., 2016) distribuídos livremente. Neste caso, seriam necessárias adaptações no sistema acoplado dependendo da forma de divisão do domínio. Estas adaptações teriam um certo grau de complexidade. Entretanto, a implementação do sistema acoplado já foi feita com vistas à execução em paralelo, de modo que laços e elementos de programação paralela estão implementados para distintos núcleos utilizando MPI (QUINN, 2004).

Ainda no tocante à implementação em paralelo, outra candidata a melhorias é a função de percurso de células atualmente implementada no *milonga*. Levantamentos preliminares mostraram que a atual implementação do *milonga* gasta grande parte do tempo de execução nesta função, devido à interpolação de temperaturas por célula. A otimização desta função levaria a ganhos no tempo total de execução. A otimização desta função não está diretamente ligada à implementação em paralelo. Entretanto, antes da paralelização de qualquer algoritmo, é usual que se trabalhe na sua versão ótima (ou tão boa quanto possível).

A aplicação utilizada para a prova de conceito se restringiu à utilização da aproximação por difusão para os cálculos de fluxo de nêutrons. O *milonga* oferece ainda a opção de utilização do método de ordenadas discretas, também chamado de método S_N (HÉBERT, 2009), para a solução da equação de transporte. Entretanto, para problemas maiores a demanda por memória do método citado inviabiliza seu uso. A reimplementação deste método com objetivo de otimizar o uso de memória pode ser um caminho para a solução mais precisa do fluxo de nêutrons se comparado ao método de aproximação por difusão. Além disso, está em curso a implementação do método de características (HÉBERT, 2009) no *milonga* por membros da comunidade argentina de Física de Reatores.

A expansão do *solver OpenFOAM* para o cálculo de transientes além de estado estacionário também é possível. Para isso, uma abordagem é adaptar um novo *solver*

[2] A arquitetura dos processadores atuais implementa unidades de processamento independentes, denominadas *core*. Isso significa que a capacidade de multiprocessamento é inerente a esses processadores.

OpenFOAM já existente, capaz de lidar com variações em relação ao tempo utilizando os arquivos e modificações já feitas no sistema atual. Este seria um trabalho mais elaborado.

Outra linha de trabalho consiste em alterar a utilização de malhas em ambos os códigos, otimizando o uso de memória compartilhada. Neste caso, as classes utilizadas no *OpenFOAM* na implementação da utilização de malhas podem ser estendidas - servindo-se do conceito de herança existente no paradigma de programação orientada a objetos e no qual o *OpenFOAM* se baseia - para que o armazenamento de toda a estrutura de dados se dê em memória compartilhada. Isso exigiria ainda que toda a implementação do *milonga* no tratamento de malhas fosse, também, modificada de acordo. Esta abordagem já consiste num grande trabalho de projeto e implementação de software. A consequência deste trabalho seria a utilização de uma única macroestrutura de dados para representação do domínio em memória, utilizando, a grosso modo, metade da memória utilizada na implementação atual. Essa nova implementação poderia, ainda, ser projetada para funcionar em paralelo.

Ainda no tocante à implementação dos sistemas envolvidos, há a possibilidade de utilizar a capacidade ociosa de placas gráficas para, por exemplo, a solução das matrizes de autovalores. Esta abordagem, entretanto, necessita de mais estudos sobre sua viabilidade.

Fora dos trabalhos futuros relativos à implementações de novas funcionalidades, estão simulações numéricas mais elaboradas utilizando-se o sistema atual. É fato que, com as limitações atuais, em especial relativas ao cálculo sequencial, não é possível utilizar o sistema para problemas elaborados. Porém, estudos de validação utilizando problemas conhecidos (*benchmarks*) são fundamentais para que o sistema desenvolvido possa ser, eventualmente, considerado para simulações em nível de licenciamento ou aplicações de missão crítica.

Os possíveis caminhos de desenvolvimento futuro e aplicações não estão limitados aos apresentados neste capítulo. A expectativa ao fim deste trabalho de tese é continuar contribuindo para o desenvolvimento do ecossistema de ferramentas computacionais com foco nas vantagens do desenvolvimento baseado em *software* livre, para todos os envolvidos e interessados em Computação Científica aplicada à Engenharia Nuclear.

Referências

ANDROUTSELLIS-THEOTOKIS, S.; SPINELLIS, D.; KECHAGIA, M.; GOUSIOS, G. Open source software: A survey from 10,000 feet. *Foundations and Trends® in Technology, Information and Operations Management*, v. 4, n. 3–4, p. 187–347, 2010. ISSN 1571-9545. Disponível em: <http://dx.doi.org/10.1561/0200000026>. Citado 2 vezes nas páginas 22 e 23.

ATLIDAKIS, V.; ANDRUS, J.; GEAMBASU, R.; MITROPOULOS, D.; NIEH, J. POSIX abstractions in modern operating systems: The old, the new, and the missing. In: *Proceedings of the Eleventh European Conference on Computer Systems*. New York, NY, USA: ACM, 2016. (EuroSys '16), p. 19:1–19:17. ISBN 978-1-4503-4240-7. Disponível em: <http://doi.acm.org/10.1145/2901318.2901350>. Citado na página 41.

BAGLIETTO, E. New perspectives for nuclear reactor design. *atw - International Journal for Nuclear Power*, v. 57, n. 11, p. 652–656, Novembro 2012. Citado 2 vezes nas páginas 21 e 26.

BALAY, S.; ABHYANKAR, S.; ADAMS, M. F.; BROWN, J.; BRUNE, P.; BUSCHELMAN, K.; DALCIN, L.; EIJKHOUT, V.; GROPP, W. D.; KAUSHIK, D.; KNEPLEY, M. G.; MCINNES, L. C.; RUPP, K.; SMITH, B. F.; ZAMPINI, S.; ZHANG, H.; ZHANG, H. *PETSc Users Manual*. [S.l.], 2016. Disponível em: <http://www.mcs.anl.gov/petsc>. Citado 2 vezes nas páginas 47 e 91.

BARBER, D. A.; DOWNAR, T. J. *Software Design and Implementation Document for the General Interface in the Coupled Code*. [S.l.], 1998. Citado na página 27.

BEAUDOIN, M.; JASAK, H. Development of a generalized grid interface for turbomachinery simulations with OpenFOAM. In: *Open Source CFD International Conference*. [S.l.: s.n.], 2008. Citado na página 29.

BENNETT, A.; AVRAMOVA, M.; IVANOV, K. Coupled mcnp6/ctf code: Development, testing, and application. *Annals of Nuclear Energy*, v. 96, p. 1 – 11, 2016. ISSN 0306-4549. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0306454916302596>. Citado na página 36.

BOYD, W.; SHANER, S.; LI, L.; FORGET, B.; SMITH, K. The openmoc method of characteristics neutral particle transport code. *Annals of Nuclear Energy*, v. 68, p. 43 – 52, 2014. ISSN 0306-4549. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0306454913006634>. Citado 2 vezes nas páginas 22 e 23.

BRITANNICA., E. *Linux Operating System*. 2014. Web. Acessada em 11 de janeiro de 2016. Disponível em: <http://www.britannica.com/technology/Linux>. Citado 2 vezes nas páginas 22 e 90.

BSD-3. *Berkeley Software Distribution*. 1999. Disponível em: <https://opensource.org/licenses/BSD-3-Clause>. Citado na página 44.

DORVAL, E.; LEPPÄNEN, J. Monte carlo current-based diffusion coefficients: Application to few-group constants generation in serpent. *Annals of Nuclear*

Energy, v. 78, p. 104 – 116, 2015. ISSN 0306-4549. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0306454914006598>. Citado na página 30.

DOWNAR, T.; XU, Y.; SEKER, V.; CARLSON, D. *PARCS v2.7*: User manual. [S.l.], 2006. Citado na página 33.

EYMARD, R.; GALLOUËT, T.; HERBIN, R. *Finite volume methods*. 2003. Web. Disponível em: <http://www.cmi.univ-mrs.fr/~herbin/PUBLI/bookevol.pdf>. Citado na página 22.

FAGHIHI, F.; KHALAFI, H.; MIRVAKILI, S. M. A literature survey of neutronics and thermal-hydraulics codes for investigating reactor core parameters; artificial neural networks as the VVER-1000 core predictor. In: TSVETKOV, D. P. (Ed.). *Nuclear Power - System Simulations and Operation*. [S.l.]: InTech, 2011. p. 103–122. Citado na página 31.

FIORINA, C.; CLIFFORD, I.; AUFIERO, M.; MIKITYUK, K. GeN-Foam: a novel OpenFOAM® based multi-physics solver for 2D/3D transient analysis of nuclear reactors. *Nuclear Engineering and Design*, v. 294, p. 24–37, 2015. Citado na página 32.

FREE SOFTWARE FOUNDATION. *GNU General Public License*. 2007. Disponível em: <http://www.gnu.org/licenses/gpl.html>. Citado 2 vezes nas páginas 46 e 47.

FRIEDMAN, E. Serpent monte-carlo code: an advanced tool for few-group cross section generation. *ATW - International Journal for Nuclear Power*, v. 58, p. 156–157, 2013. Citado na página 30.

GALASSI, M.; DAVIES, J.; THEILER, J.; GOUGH, B.; JUNGMAN, G. *GNU Scientific Library - Reference Manual, Third Edition, for GSL Version 1.12 (3. ed.)*. [S.l.]: GSL Team, 2009. 1-573 p. ISBN 978-0-9546120-7-8. Citado na página 47.

GASTON, D.; NEWMAN, C.; HANSEN, G.; LEBRUN-GRANDIé, D. Moose: A parallel computational framework for coupled systems of nonlinear equations. *Nuclear Engineering and Design*, v. 239, n. 10, p. 1768 – 1778, 2009. ISSN 0029-5493. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0029549309002635>. Citado na página 35.

GEIST, A.; BEGUELIN, A.; DONGARRA, J.; JIANG, W.; MANCHEK, R.; SUNDERAM, V. *PVM: Parallel Virtual Machine*: A users' guide and tutorial for networked parallel computing. [S.l.]: MIT Press, 1994. ISBN 0262571080. Citado 2 vezes nas páginas 27 e 28.

GEUZAINE, C.; REMACLE, J.-F. Gmsh: A 3-D finite element mesh generator with built-in pre- and post-processing facilities. *International Journal for Numerical Methods in Engineering*, v. 79, p. 1309 – 1331, 2009. Citado 2 vezes nas páginas 45 e 47.

GLEICHER, F. N.; ORTENSI, J.; SPENCER, B. W.; WANG, Y.; NOVASCONE, S. R.; HALES, J. D.; GASTON, D.; WILLIAMSON, R. L.; MARTINEAU, R. C. The coupling of the neutron transport applications RATTLES$_N$AKE to the nuclear fuels performance application BISON under the MOOSE framework. In: *PHYSOR-2014 - The Role of reactor Physics toward a Sustainable Future*. [S.l.: s.n.], 2014. Citado na página 35.

GRAHAM, R. L.; KNUTH, D. E.; PATASHNIK, O. *Concrete Mathematics: A Foundation for Computer Science*. 2nd. ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1994. ISBN 0201558025. Citado na página 36.

GRAHAM, S. L.; SNIR, M.; PATTERSON, C. A. *Getting Up to Speed: The Future of Supercomputing*. [S.l.]: National Research Council, 2004. ISBN 0309546796. Citado na página 27.

HALSALL, M.; TAUBMAN, C. *The 1986 WIMS Nuclear Data Library*. 1986. AEEW-R 2133. Citado na página 70.

HÉBERT, A. *Applied Reactor Physics*. [S.l.]: Presses Internationales Polytechnique, 2009. ISBN 9782553014369. Citado na página 91.

HERMAN, B. R.; FORGET, B.; SMITH, K. Progress toward monte carlo–thermal hydraulic coupling using low-order nonlinear diffusion acceleration methods. *Annals of Nuclear Energy*, v. 84, p. 63 – 72, 2015. ISSN 0306-4549. Multi-Physics Modelling of LWR Static and Transient Behaviour. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0306454914005684>. Citado na página 37.

HERNANDEZ, V.; ROMAN, J. E.; VIDAL, V. SLEPc: A scalable and flexible toolkit for the solution of eigenvalue problems. *ACM Trans. Math. Software*, v. 31, n. 3, p. 351–362, 2005. Citado 2 vezes nas páginas 47 e 91.

HOSSAIN, A. S. K. *Development of a fast running multidimensional thermal-hydraulic code to be readly coupled with multidimensional neutronic tools, applicable to modular High Temperature Reactors*. Tese (Doutorado) — Institut für Kernenergetik und Energiesysteme - Universität Stuttgart, Fevereiro 2011. Citado na página 30.

HUFF, K. D.; GIDDEN, M. J.; CARLSEN, R. W.; FLANAGAN, R. R.; MCGARRY, M. B.; OPOTOWSKY, A. C.; SCHNEIDER, E. A.; SCOPATZ, A. M.; WILSON, P. P. Fundamental concepts in the cyclus nuclear fuel cycle simulation framework. *Advances in Engineering Software*, v. 94, p. 46 – 59, 2016. ISSN 0965-9978. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0965997816300229>. Citado 3 vezes nas páginas 22, 23 e 37.

HUMMEL, D. W.; NOVOG, D. R. Coupled 3d neutron kinetics and thermalhydraulic characteristics of the canadian supercritical water reactor. *Nuclear Engineering and Design*, v. 298, p. 78 – 89, 2016. ISSN 0029-5493. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0029549315005919>. Citado 2 vezes nas páginas 34 e 42.

IVANOV, K.; AVRAMOVA, M. Challenges in coupled thermal-hydraulics and neutronics simulations for LWR safety analysis. *Annals of Nuclear Energy*, v. 34, p. 501–513, 2007. Citado 4 vezes nas páginas 28, 31, 36 e 42.

JARETEG, K. *Development of an Integrated deterministic neutronic/thermal-hydraulic model using a CFD solver*. Tese (Doutorado) — Department of Applied Physics - Chalmers University of Technology, 2012. Citado 3 vezes nas páginas 31, 33 e 34.

JARETEG, K.; VINAI, P.; DEMAZIÈRE, C. Fine-mesh deterministic modeling of PWR fuel assemblies: Proof-of-principle of coupled neutronic/thermal–hydraulic calculations. *Annals of Nuclear Energy*, v. 68, p. 247–256, 2014. ISSN 0306-4549. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0306454914000036>. Citado 5 vezes nas páginas 30, 32, 33, 43 e 51.

JARETEG, K.; VINAI, P.; SASIC, S.; DEMAZIèRE, C. Coupled fine-mesh neutronics and thermal-hydraulics – modeling and implementation for PWR fuel assemblies. *Annals of Nuclear Energy*, v. 84, p. 244 – 257, 2015. ISSN 0306-4549. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0306454915000869>. Citado 2 vezes nas páginas 29 e 34.

JASAK, H. Openfoam: Open source CFD in research and industry. *International Journal of Naval Architecture and Ocean Engineering*, v. 1, n. 2, p. 89 – 94, 2009. ISSN 2092-6782. Disponível em: <http://www.sciencedirect.com/science/article/pii/S2092678216303879>. Citado na página 46.

JASAK, H.; JEMCOV, A.; TUKOVIĆ Željko. OpenFOAM: A C++ library for complex physics simulations. In: *International Workshop on COUPLED METHODS IN NUMERICAL DYNAMICS*. [S.l.: s.n.], 2007. Citado 2 vezes nas páginas 26 e 46.

KRAEVOY, V.; SHEFFER, A. Cross-parameterization and compatible remeshing of 3d models. *ACM Trans. Graph.*, ACM, v. 23, n. 3, p. 861–869, 2004. ISSN 0730-0301. Citado na página 43.

KREPPER, E.; KONCAR, B.; EGOROV, Y. CFD modelling of subcooled boiding - Concept, validation and application to fuel assembly design. *Nuclear Engineering and Design*, v. 237, p. 716–731, 2007. Citado na página 25.

LARSSON, V.; DEMAZIÈRE, C. A coupled neutronics/thermal-hydraulics tool for calculating fluctuations in pressurized water reactors. *Annals of Nuclear Energy*, v. 1, n. 43, p. 68–76, 2012. Citado na página 33.

LAUNDER, B.; SPALDING, D. The numerical computation of turbulent flows. *Computer Methods in Applied Mechanics and Engineering*, v. 3, n. 2, p. 269 – 289, 1974. ISSN 0045-7825. Disponível em: <http://www.sciencedirect.com/science/article/pii/0045782574900292>. Citado 2 vezes nas páginas 47 e 66.

LAZARO, A.; STANISZ, P. S.; AMMIRABILE, L.; TSIGE-TAMIRAT, H.; MARTORELL, S.; VERDÚ, G. Development of a three-dimensional thermohydraulics-neutronic coupling scheme for a transient analysis of liquid metal cooled fast reactor technologies suing the system code TRACE-PARCS. In: *The 15th International Topical Meeting on Nuclear Reactor Thermal-Hydraulics, NURETH-15*. Pisa, Italy: [s.n.], 2013. Citado na página 32.

LEPPÄNEN, J. *Serpent - a Continuous-energy Monte Carlo reactor physics burnup Calculation Code*. [S.l.], 2013. Citado 3 vezes nas páginas 30, 32 e 33.

LEPPÄNEN, J.; VIITANEN, T. Multi-physics coupling scheme in the Serpent 2 Monte Carlo code. In: *Transactions of the American Nuclear Society*. [S.l.: s.n.], 2012. p. 1165–1168. Citado 3 vezes nas páginas 21, 27 e 34.

LEPPÄNEN, J.; HOVI, V.; IKONEN, T.; KURKI, J.; PUSA, M.; VALTAVIRTA, V.; VIITANEN, T. The numerical multi-physics project (NUMPS) at VTT technical research centre of finland. *Annals of Nuclear Energy*, v. 84, p. 55–62, 2015. ISSN 0306-4549. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0306454914005532>. Citado na página 28.

LETHBRIDGE, P. Multiphysics analysis. *The Industrial Physicist*, v. 10, n. 6, p. 26–29, Janeiro 2005. Citado na página 27.

MACIEL, F. *Coupling RELAP Code with External Calculation Programs (Shared Memory Version)*. [S.l.], 2011. Citado 2 vezes nas páginas 33 e 89.

MARIĆ, T.; HÖPKINS, J.; MOONEY, K. *The OpenFOAM Technology Primer*. [S.l.]: Sourceflux UG, 2014. ISBN 9783000467578. Citado na página 46.

MARTIN, K.; HOFFMAN, B. *Mastering CMake 4th Edition*. 4th. ed. USA: Kitware, Inc., 2008. ISBN 1930934203, 9781930934207. Citado na página 90.

MARTÍNEZ, M.; MIRÓ, R.; VERDÚ, G.; PEREIRA, C.; MESQUITA, A. Z.; CHIVA, S. Assessment of a computational fluid dynamic (CFD) model for the IPR-R1 TRIGA research reactor. In: *American Nuclear Society Annual Meeting*. [S.l.: s.n.], 2012. Citado na página 25.

MCFARLANE, R. E.; MUIR, D. W.; BOICOURT, R. M.; KAHLER, A. C.; CONLIN, J. L. *The NJOY Nuclear Data Processing System, Version 2016*. [S.l.], 2016. Citado 2 vezes nas páginas 44 e 90.

NAVARRO, M. A.; SANTOS, A. A. C. Evaluation of a numerico procedure for flow simulation of a 5x5 PWR rod bundle with a mixing vane spacer. *Progress in Nuclear Energy*, v. 53, n. 8, p. 1190–1196, 2011. ISSN 0149-1970. International Nuclear Atlantica Conference - INAC 2009. Citado na página 25.

NORRIS, J. S. Mission-critical development with open source software: lessons learned. *Software, IEEE*, v. 21, n. 1, p. 42–49, Jan 2004. ISSN 0740-7459. Citado na página 22.

NOURBAKHSH, H. *Review and Evaluation of the Nuclear Regulatory Commission Safety Research Program*. [S.l.], 2010. Citado na página 25.

OPENFOAM - the Open Source CDF Toolbox: User guide. [S.l.], 2013. Citado na página 31.

OPENFOAM - the Open Source CDF Toolbox: Programmer's guide. [S.l.], 2015. Citado na página 67.

OPENFOAM C++ Documentation. [S.l.], 2015. Disponível em: <http://www.openfoam.org/docs/cpp/>. Acesso em: 25 de janeiro de 2016. Citado 2 vezes nas páginas 32 e 47.

POPE, M. A.; MOUSSEAU, V. A. Accuracy and efficiency of a coupled neutronics and thermal hydraulics model. *Nuclear Engineering and Technology*, v. 41, n. 7, p. 885–892, Setembro 2009. Citado na página 31.

QUINN, M. J. *Parallel programming in C with MPI and OpenMP*. 1st. ed. [S.l.]: McGraw-Hill, 2004. (Higher Education). ISBN 0072822562. Citado 2 vezes nas páginas 28 e 91.

REIS, P. A. L.; PEREIRA, C.; COSTA, A. L.; GONZÁLEZ-MANTECÓN, J.; VELOSO, M. A. F.; SOARES, H. V.; MIRÓ, R. Thermal hydraulic and neutron kinetic simulation of the TRIGA IPR-R1 research reactor using RELAP5-PARCS coupled model. In: *European Research Reactor Conference 2015 - Transactions*. [S.l.]: European Nuclear Society, 2015. p. 259–269. Citado 2 vezes nas páginas 33 e 70.

RELAP5/MOD3.3 Code Manual Volume I: Code structure, system models, and solution methods. [S.l.], 2003. Citado na página 33.

RICHARD, J.; GALLOWAY, J.; FENSIN, M.; TRELLUE, H. Smithers: An object-oriented modular mapping methodology for mcnp-based neutronic–thermal hydraulic multiphysics. *Annals of Nuclear Energy*, v. 81, p. 150 – 163, 2015. ISSN 0306-4549. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0306454915001619>. Citado 2 vezes nas páginas 29 e 36.

ROBBINS, K.; ROBBINS, S. *UNIX Systems Programming: Communication, Concurrency, and Threads*. [S.l.]: Prentice Hall PTR, 2003. ISBN 9780130424112. Citado na página 41.

ROMANO, P. K.; FORGET, B. The openmc monte carlo particle transport code. *Annals of Nuclear Energy*, v. 51, p. 274 – 281, 2013. ISSN 0306-4549. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0306454912003283>. Citado 3 vezes nas páginas 22, 23 e 37.

SCHMIDT, R.; BELCOURT, K.; HOOPER, R.; PAWLOWSKI, R.; CLARNO, K.; SIMUNOVIC, S.; SLATTERY, S.; TURNER, J.; PALMTAG, S. An approach for coupled-code multiphysics core simulations from a common input. *Annals of Nuclear Energy*, v. 84, p. 140 – 152, 2015. ISSN 0306-4549. Multi-Physics Modelling of LWR Static and Transient Behaviour. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0306454914005957>. Citado 2 vezes nas páginas 29 e 35.

SHER, R.; FIARMAN, S. *Studies of thermal-reactor benchmark-data interpretation: experimental corrections*. [S.l.], 1976. Citado na página 69.

SILVA, V. V. A.; SANTOS, A. A. C.; MESQUITA, A. Z.; BERNAL, A.; MIRÓ, R.; VERDÚ, G.; PEREIRA, C. Finite volume thermal-hydraulics and neutronics coupled calculations. In: *Proceedings of ICAPP 2015, Nice, France*. [S.l.: s.n.], 2015. Citado na página 28.

SLAYBAUGH, R. *The PyNE Software Library, A Framework for ENSDF?* Figshare, 2014. Disponível em: <https://doi.org/10.6084/M9.FIGSHARE.1254163.V1>. Citado na página 44.

STALLMAN, R. M. *Free Software, Free Society: Selected Essays of Richard M. Stalmann*. [S.l.]: Free Software Foundation, 2002. ISBN 1882114981. Citado na página 22.

THELER, G. Unstructured grids and the multigroup neutron diffusion equation. *Science and Technology of Nuclear Installations*, v. 2013, p. 26, 2013. Citado 3 vezes nas páginas 43, 49 e 68.

THELER, G. Transporte y difusión de neutrones en mallas no estructuradas. Não publicado. 2016. Citado na página 49.

THELER, G.; OMIL, J. P. G.; PELLEGRINO, E. A shared-memory-based coupling scheme for modeling the behavior of a nuclear power plant core. In: *Mecánica Computacional*. [S.l.: s.n.], 2013. v. 32, p. 1501–1517. Citado 2 vezes nas páginas 42 e 89.

TRAHAN, T. J. *An asymptotic, homogeneized, anisotropic, multigroup diffusion approximation to the neutron transport equation*. Tese (Doutorado) — Nuclear Engineering and Radiological Sciences Department, University of Michigan, 2014. Citado na página 48.

TURINSKY, P. J.; KOTHE, D. B. Modeling and simulation challenges pursued by the consortium for advanced simulation of light water reactors (casl). *Journal of Computational Physics*, v. 313, p. 367 – 376, 2016. ISSN 0021-9991. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0021999116001133>. Citado na página 28.

USTINENKO, V.; SAMIGULIN, M.; IOILEV, A.; LO, S.; TENTNER, A.; LYCHAGIN, A.; RAZIN, A.; GIRIN, V.; VANYUKOV, Y. Validation of cfd-bwr, a new two-phase computational fluid dynamics model for boiling water reactor analysis. *Nuclear Engineering and Design*, v. 238, n. 3, p. 660 – 670, 2008. ISSN 0029-5493. Benchmarking of {CFD} Codes for Application to Nuclear Reactor Safety. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0029549307003421>. Citado na página 21.

VAZQUEZ, M.; TSIGE-TAMIRAT, H.; AMMIRABILE, L.; MARTIN-FUERTES, F. Coupled neutronics thermal-hydraulics analysis using Monte Carlo and sub-channel codes. *Nuclear Engineering and Design*, v. 250, p. 403–411, 2012. Citado na página 34.

VELOSO, M. A. *Avaliação termo-hidráulica do reator TRIGA IPR-R1 a 150kW*. [S.l.], 2005. Citado 7 vezes nas páginas 59, 60, 61, 66, 70, 73 e 74.

VELOSO, M. A. F. *Análise Termofluidodinâmica de Reatores Nucleares de Pesquisa Refrigerados à Água em Regime de Convecção Natural*. Tese (Doutorado) — Faculdade de Engenharia Química da Universidade Estadual de Campinas, 2004. Citado na página 59.

VERSTEEG, H. K.; MALALASEKERA, W. *An Introduction to Computational Fluid Dynamics*: The finite volume method. 2nd. ed. [S.l.]: Pearson Education Limited, 2007. ISBN 9780131274983. Citado na página 55.

WALLI, S. R. The posix family of standards. *StandardView*, ACM, New York, NY, USA, v. 3, n. 1, p. 11–17, mar. 1995. ISSN 1067-9936. Disponível em: <http://doi.acm.org/10.1145/210308.210315>. Citado na página 41.

YAN, J.; KOCHUNAS, B.; HURSIN, M.; DOWNAR, T.; KAROUTAS, Z.; BAGLIETTO, E. Coupled computational fluid dynamics and MOC neutronic simulations of Westinghouse PWR fuel assemblies with grid spacers. In: *The 14th International Topical Meeting on Nuclear Reactor Thermalhydraulics, NURETH-14*. [S.l.: s.n.], 2011. Citado na página 30.

ZERKAK, O.; KOZLOWSKI, T.; GAJEV, I. Review of multi-physics temporal coupling methods for analysis of nuclear reactors. *Annals of Nuclear Energy*, v. 84, p. 225 – 233, 2015. ISSN 0306-4549. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0306454915000298>. Citado na página 29.

Anexos

ANEXO A – Lista das publicações

Publicações originadas deste trabalho de tese, incluindo trabalhos completos em conferências e em revista.

- SILVA, V. V. A.; SCHWEIZER, F. L. A.; SANTOS, A. A. C.; MESQUITA, A. Z.; PEREIRA, C.; COSTA, A. L.; VELOSO, M. A. F.; MIRÓ, R.; VERDÚ, G. *An Initial Verification and Validation Study of CFD Numerical Simulations of the IPR-R1 TRIGA Reactor*. In: RRFM 2012, 2012, PRAGA. REACTOR FUEL PERFORMACE 2012, 2012. p. 1-6.

- SILVA, V. V. A.; SANTOS, A. A. C.; SILVA, P. S. B. L.; MESQUITA, A. Z.; PEREIRA, C. *TRIGA Fuel Simulation Using OpenFOAM*. In: EUROPEAN RESEARCH REACTOR CONFERENCE (RRFM 2013), 2013, SAINT PETERSBURG. TRANSACTIONS OF EUROPEAN RESEARCH REACTOR CONFERENCE (RRFM 2013), 2013. v. ÚNICO. p. 569-616.

- SILVA, V. V. A.; SANTOS, A. A. C.; MESQUITA, A. Z.; SILVA, P. S. B. L.; PEREIRA, C. *CFD Simulation of IPR-R1 TRIGA Subchannels fluid flow*. In: INAC - International Nuclear Atlantic Conference, 2013, Recife. The Benefits of Nuclear Technology for Social Inclusion. Rio de Janeiro: ABEN, 2013. v. unico. p. 1-5.

- SILVA, V. V. A.; SANTOS, A. A. C.; Zacarias A. M.; BERNAL, A. ; MIRÓ, R.; VERDÚ, G.; PEREIRA, C. *Finite Volume thermal-hydraulics and neutronics coupled calculations*. In: ICAPP - International Congress on Advances in Nuclear Power Plants, 2015, Nice. Proceedings of ICAPP 2015, 2015. v. único. p. 1-8.

- SILVA, V. V. A; SANTOS, A. A. C.; THELER, G.; PEREIRA, C. *Open Software One-Step Coupled Neutronics and CFD Thermal-hydraulics calculation*. International Journal of Emerging Technology and Advanced Engineering. Volume 6, issue 12, Dezembro de 2016. (Artigo aceito para publicação)

ANEXO B – Registro de programa de computador

Uma biblioteca capaz de gerar um perfil cossenoidal de potências como condição de contorno *OpenFOAM* foi desenvolvida durante esta tese e registrada junto ao INPI.

- SILVA, V. V. A.; PEREIRA, C.; MESQUITA, A. Z.; SANTOS, A. A. C.; SILVA, S. B. L. TRIGAFUEL. 2013. Patente: Programa de Computador. Número do registro: BR512013000400-4, data de registro: 22/04/2013, título: "TRIGAFUEL", Instituição de registro: INPI - Instituto Nacional da Propriedade Industrial

ANEXO C – Código-fonte desenvolvido

O código-fonte desenvolvido é apresentado neste anexo. A listagem apresenta o código-fonte *OpenFOAM* bem como o *script milonga* de referência utilizado no sistema acoplado.

```
/*---------------------------------------------------------------------------* \
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
  \\    /   O peration     |
  \\  /    A nd           | Copyright (C) 2011-2013 OpenFOAM Foundation
  \\/     M anipulation  |
  -------------------------------------------------------------------------------
-
  License
  This file is part of OpenFOAM.

  OpenFOAM is free software: you can redistribute it and/or modify it
  under the terms of the GNU General Public License as published by
  the Free Software Foundation, either version 3 of the License, or
  (at your option) any later version.

  OpenFOAM is distributed in the hope that it will be useful, but WITHOUT
  ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
  FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
  for more details.

  You should have received a copy of the GNU General Public License
  along with OpenFOAM.  If not, see <http://www.gnu.org/licenses/>.

  Application
  chtMultiRegionSimpleFoam

  Description
  Steady-state version of chtMultiRegionFoam

  ------------------------------------------------------------------------------
  ---
  --- This file was modified as part of
  --- Vitor Vasconcelos AraÃºjo Silva Ph.D thesis work.
  ---
  contact: vitors@cdtn.br

  Application
  thesisCoupledFoam

  Description
  Steady-state version of chtMultiRegionFoam adapted to get the source-term for
  the energy equation from an external source. The source term is defined as a
  volScalarField and read from a C array structure. This version is implemented
  and tested to run in parallel and sequentialy.

  IMPORTANT:
  ----------
  OpenFOAM and Milonga communication is based in POSIX semaphores.
  Milonga should be started before OpenFOAM. Otherwise OpenFOAM will
  hold soon after initialization waiting for the first semaphore.
  (A call to sem_wait(calcOf) in createCouplingFields.H)

  \*---------------------------------------------------------------------------*
/

#include "fvCFD.H"
#include "rhoThermo.H"
#include "turbulenceModel.H"
#include "fixedGradientFvPatchFields.H"
#include "regionProperties.H"
#include "solidThermo.H"
#include "radiationModel.H"
#include "fvIOoptionList.H"
#include "OFstream.H"
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
#include "SortableList.H"

// C++ stdlib
```



```
#include <fstream>
#include <vector>
#include <iostream>
#include <cstdio>
#include <ctime>
#include <cstring>          // Used by memcpy when coupling

// C Posix (Shared memory communication)
#include <semaphore.h>
#include <sys/shm.h>
#include <sys/mman.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <errno.h>

using namespace std;

// To be able to call the Fortran function heatflux_
extern "C" void heatflux_(float*);

int main(int argc, char *argv[])
{

#include "setRootCase.H"

    Info << nl << nl
         << " ---> thesisCoupledFoam. Versao no Brasil." << nl
         << " ---> Vitor Vasconcelos - vitors@cdtn.br" << nl
         << " ---> Built at " << __TIME__ << ", " << __DATE__ << nl << endl;

#include "createTime.H"

    // Number of CFD iterations before neutronics call
    // hardcoded:
    unsigned int nIterations = 0;
    unsigned int cFact = 100;

    regionProperties rp(runTime);

#include "createFluidMeshes.H"
#include "createSolidMeshes.H"

#include "createFluidFields.H"
#include "createSolidFields.H"

#include "initContinuityErrs.H"

#include "createCouplingFields.H"

    // Get the number of processes
    //  label nProcs = Pstream::nProcs();

    while (runTime.loop())
    {
        nIterations++;

        Info<< "Time = " << runTime.timeName() << nl << endl;

        forAll(fluidRegions, i)
        {
            Info << "\nSolving for fluid region "
                 << fluidRegions[i].name() << endl;

#include "setRegionFluidFields.H"
#include "readFluidMultiRegionSIMPLEControls.H"
#include "solveFluid.H"

        }
```

```
        forAll(solidRegions, i)
        {
            Info<< "\nSolving for solid region "
                << solidRegions[i].name() << endl;

#include "setRegionSolidFields.H"
#include "readSolidMultiRegionSIMPLEControls.H"
#include "solveSolid.H"

            runTime.write();

            Info<< "ExecutionTime = " << runTime.elapsedCpuTime() << " s"
                << "  ClockTime = " << runTime.elapsedClockTime() << " s"
                << nl << endl;
        }

        // Time to couple ------------------------------------------
        //
        // How neutronics is called?
        //
        // - check if the defined number of iterations (defined in nIterCFD)
        //   ran using the std::fmod() function. If yes:
        // - copy all data from all processors to completeLists[]
        // - call neutronics
        // - There is NO normalization. Data comes from Milonga as power.
        // - scatter data from Q to all processors back. (This only happens
        //   for region 'fuel')
        // - bool values updated to make

        Info << "------ Neutronics call ratio (iterations): " << nIterations%cFact
             << endl;

        // Recall: coupling is created in createCouplingFields and it is false
        // if no shm files are correctly read.

        if(!(nIterations%cFact) && coupling) // Controls neutronics call
        {
            forAll(fluidRegions, i)
            {
                if(!Pstream::parRun())
                {
                    // Force cells references ordering.
                    // It *seems* that OpenFOAM mix relative cell
                    // values when splitting the mesh among regions
                    SortableList<label> *tmp;
                    tmp = new SortableList<label>(fluidRegionsLists[i]);

                    solidRegionsLists[i].clear();
                    solidRegionsLists[i].append(*tmp);

                    delete(tmp);
                }

                // Structures to hold data and cell addressing
                // for all processors
                List<scalarList> dataT(Pstream::nProcs());
                List<scalarList> dataRho(Pstream::nProcs());

                scalarList localDataT(thermoFluid[i].T().size());
                scalarList localDataRho(thermoFluid[i].T().size());

                // Since T, rho and Q have the same size
                // the processing of all data is performed
                // inside the same loop
                for(int k=0; k<Pstream::nProcs(); k++)
                {
                    dataT[k].setSize(thermoFluid[i].T().size(), 0.0);
                    dataRho[k].setSize(thermoFluid[i].rho().size(), 0.0);
                }
```



```
                // The if structure below gathers data from all processors
                if(Pstream::master())
                {
                    // For master processor data is copied directly
                    dataT[0] = thermoFluid[i].T();
                    dataRho[0] = thermoFluid[i].rho();

                    // Gather data from all processors to master
                    for(label j=1; j<Pstream::nProcs(); j++)
                    {
                        IPstream inputMasterStream(Pstream::blocking, j);
                        inputMasterStream >> dataT[j] >> dataRho[j];

                    }
                    // Copy from dataT and dataRho to the completeList for FLUID
 Regions
                    for(int k=0; k<Pstream::nProcs(); k++)
                    {
                        unsigned int offset = 0;

                        if(fluidRegions[i].name() == "coolant")
                        {
                            // The order is: fuel, cladding, coolant
                            // thermos[1] is the fuel region
                            offset = thermos[0].T().size()+thermos[1].T().size()
;
                        }

                        for(int m=0; m<dataT[k].size(); m++)
                        {
                            // ATTENTION:
                            // Trick mapping among the complete vector, using th
e fluid Regions data
                            // which maps to the data coming from processors
                            temperatureCompleteList[fluidRegionsLists[i][(k*data
T[k].size())+m]] = dataT[k][m];
                            densityCompleteList[fluidRegionsLists[i][(k*dataRho[
k].size())+m]] = dataRho[k][m];

                            // First test to send data to Milonga
                            shmTarray[offset+m] = dataT[k][m];
                        }
                    }
                }

                else
                {
                    localDataT = thermoFluid[i].T().internalField();
                    localDataRho = thermoFluid[i].rho().internalField();

                    // 0 references de master process
                    OPstream outputSlavesStream(Pstream::blocking, 0);
                    outputSlavesStream << thermoFluid[i].T().internalField()
                                       << thermoFluid[i].rho().internalField()
                                       << fluidList[Pstream::myProcNo()];
                }
            } // End forAll fluid loop

            forAll(solidRegions, i)
            {
                // Force cells references ordering.
                // For now, only if running sequentialy
                if(!Pstream::parRun())
                {
                    // It *seems* that OpenFOAM mix relative cell
                    // values when splitting the mesh among regions
                    SortableList<label> *tmp;
                    tmp = new SortableList<label>(solidRegionsLists[i]);
```

```
                    solidRegionsLists[i].clear();
                    solidRegionsLists[i].append(*tmp);

                    delete(tmp);
                }

                // Structures to hold data and cell addressing
                // for all processors
                List<scalarList> dataT(Pstream::nProcs());
                List<scalarList> dataRho(Pstream::nProcs());
                List<scalarList> dataQ(Pstream::nProcs());

                scalarList localDataT(thermos[i].T().size());
                scalarList localDataRho(thermos[i].rho().size());
                scalarList localDataQ(qVol[i].internalField().size());

                // Since T, rho and Q have the same size
                // the processing of all data is performed
                // inside the same loop
                dataT[Pstream::myProcNo()].setSize(thermos[i].T().size());
                dataRho[Pstream::myProcNo()].setSize(thermos[i].rho().size(), 0.
0);
                dataQ[Pstream::myProcNo()].setSize(qVol[i].size(), 0.0);

                // The if structure below gathers data from all processors
                if(Pstream::master())
                {
                    // For master, data is copied directly
                    dataT[0] = thermos[i].T();
                    dataRho[0] = thermos[i].rho();
                    dataQ[0] = qVol[i].internalField();

                    // Gather data from all processors to master
                    for(label j=1; j<Pstream::nProcs(); j++)
                    {
                        IPstream inputMasterStream(Pstream::blocking, j);
                        inputMasterStream >> dataT[j] >> dataRho[j] >> dataQ[j]
>> solidList[i][j];
                    }

                    // Copy from dataT and dataRho to the completeList for SOLID
 Regions
                    for(int k=0; k<Pstream::nProcs(); k++)
                    {

                        // This loop invariant is the size of dataT, which is th
e same of dataRho and dataQ.
                        // Use of any of these lists yields the same result

                        unsigned int offset = 0;

                        if(solidRegions[i].name() == "fuel")
                        {
                            offset = 0;
                        }
                        if(solidRegions[i].name() == "cladding")
                        {
                            // The order is: fuel, cladding, coolant
                            offset = thermos[1].T().size();
                        }

                        for(int m=0; m<dataT[k].size(); m++)
                        {
                            // ATTENTION:
                            // Trick mapping among the complete vector, using th
e solid Regions data
                            // which maps to the data coming from processors
                            temperatureCompleteList[solidRegionsLists[i][(k*data
```



```
T[k].size())+m]] = dataT[k][m];
                            densityCompleteList[solidRegionsLists[i][(k*dataRho[
k].size())+m]] = dataRho[k][m];
                            powerCompleteList[solidRegionsLists[i][(k*dataQ[k].s
ize())+m]] = dataQ[k][m];

                            shmTarray[offset+m] = dataT[k][m];
                        }
                    }
                }
                else
                {
                    localDataT = thermos[i].T().internalField();
                    localDataRho = thermos[i].rho().internalField();
                    localDataQ = qVol[i].internalField();

                    // 0 references de master process
                    OPstream outputSlavesStream(Pstream::blocking, 0);
                    outputSlavesStream << localDataT << localDataRho << localDat
aQ << solidList[i][Pstream::myProcNo()];
                }

            } // End forAll solid loop - data is written to SHM

            // milonga must be called
            Info << nl << "--- Calling NEUTRONICS... ";
            sem_post(calcOf);

            sem_wait(calcMil);
            Info << "DONE!" << nl << endl;

            // ------------------------------------------------------------------
------------------------
            // After neutronics call, Q data must be scattered to all processors
 and regions

            forAll(solidRegions, i) // Loop to scatter Q data from neutronics to
 all regions and processors
                                    // Note that only the 'fuel' region is used.
            {

                // Calling neutronics
                if(solidRegions[i].name() == "fuel" && Pstream::master())
                {

                    for(int o=0; o<solidRegionsLists[i].size(); o++)
                    {
                        powerCompleteList[solidRegionsLists[i][o]] = shmQarray[s
olidRegionsLists[i][o]];
                    }
                    Info << " --- Q data read from milonga." << nl << endl;
                }

                // Initialize dataQ field with the right size
                List<scalarList> dataQ(Pstream::nProcs());

                // Used to scatter data to each processor
                scalarList localDataQ(qVol[i].internalField().size(), 0.0);

                if(solidRegions[i].name() == "fuel")
                {
                    for(int k=0; k<Pstream::nProcs(); k++)
                    {
                        dataQ[k].setSize(qVol[i].internalField().size(), 1.0);
                    }
                    // The if structure below scatters data to all processors
                    if(Pstream::master())
                    {
                        for(int k=0; k<Pstream::nProcs(); k++)
```

```
                        {
                            // This loop control is the size of dataT, which is
the same of dataRho and dataQ.
                            // Use of any of these lists yields the same result
                            for(int m=0; m<dataQ[k].size(); m++)
                            {
                                // ATTENTION:
                                // Tricky mapping between the complete vector, u
sing the solid Regions data
                                // which maps to the data coming from processors
                                dataQ[k][m] = powerCompleteList[solidRegionsList
s[i][(k*dataQ[k].size())+m]];
                            }
                        }

                        // Scatter data from all processors to master
                        for(label j=1; j<Pstream::nProcs(); j++)
                        {
                            OPstream outputMasterStream(Pstream::blocking, j);
                            outputMasterStream << dataQ[j];
                        }

                        // For the main process or in a sequential run
                        // update qVol data from neutronics
                        qVol[i].internalField() = dataQ[0];
                    }
                    else
                    {
                        localDataQ = dataQ[Pstream::myProcNo()];

                        // 0 references de master process
                        IPstream inputSlavesStream(Pstream::blocking, 0);
                        inputSlavesStream >> localDataQ;

                        // Copy data to local scalarField qVol[fuel]
                        qVol[i].internalField() = localDataQ;

                    }
                    solidRegions[i].write();
                }
            }
        }

    } // runTime.loop()

    // End of runTime.loop(), free milonga

    return 0;
}

// ************************************************************************* //
```

createCouplingFields.H Page 1/2

```
// Create Lists for data interchange with neutronics

// Total number of cells is the number of elements in the
// createSolidFields/solidRegionsLists structure and
// createFluidFields/fluidRegionsLists structure
unsigned int totalNumberOfCells = 0;
//unsigned int totalNumberOfRegions = solidsNames.size() + fluidNames.size();

// ------------------------------------------------------------------
// Notes on shared memory coupling:
//
// - Shared memory files are created by OpenFoam.
// - Files must be deleted by OpenFOAM.
// (changed from the last implementation)

bool coupling = true;

// Three C standard arrays are created to shared data with Milonga
double *shmTarray = NULL;
double *shmQarray = NULL;

void *shmT;
void *shmQ;

int shmTfile = 0;
int shmQfile = 0;

// Posix C semaphores
sem_t *calcOf;
sem_t *calcMil;

// Posix structures initialization
calcOf = sem_open("calcOf", O_CREAT, 0666);
calcMil = sem_open("calcMil", O_CREAT, 0666);

if(Pstream::master())
{
// Semaphores and shared memory files are tested at this point.
// If milonga is not running, OpenFOAM runs uncoupled.
    shmTfile = shm_open("temperaturas", O_RDWR, 0666);
    shmQfile = shm_open("potencias", O_RDWR, 0666);

    if(shmTfile == -1 || shmQfile == -1)
    {
      Info << nl << " ---: Error getting shared memory: " << strerror(errno) << ". NON-COU
PLED calculations." << nl << endl;
        coupling = false;
    }
    else
      Info << nl << " ---: milonga shared memory data detected! COUPLED calculations. " << nl <<
endl;
}

if(Pstream::master())
{
  for (int i=0; i<solidsNames.size(); ++i)
    {
      totalNumberOfCells += solidRegionsLists[i].size();
    }

  for (int i=0; i<fluidNames.size(); ++i)
    {
      totalNumberOfCells += fluidRegionsLists[i].size();
    }
}

Info << " ---: totalNumberOfCells: " << totalNumberOfCells << endl;

List<double> temperatureCompleteList(totalNumberOfCells, 300.0);
```

createCouplingFields.H Page 2/2

```
List<double> densityCompleteList(totalNumberOfCells, 1000.0);
List<double> powerCompleteList(totalNumberOfCells, 0.0);

if(Pstream::master())
{
// After reading the shared memory files, they must be mapped to data
    shmT = mmap(NULL, totalNumberOfCells*sizeof(double), PROT_WRITE, MAP_SHARED,
 shmTfile, 0);
    shmQ = mmap(NULL, totalNumberOfCells*sizeof(double), PROT_WRITE, MAP_SHARED,
 shmQfile, 0);

// Check if all files were properly mapped
    if((shmT == MAP_FAILED || shmQ == MAP_FAILED) && (coupling))
    {
        Info << " --- Error mapping shared memory: " << strerror(errno) << ". Exiting..." <<
endl;
        exit(errno);
    }

// Make a C++ cast
    shmTarray = reinterpret_cast<double*>(shmT);
    shmQarray = reinterpret_cast<double*>(shmQ);
}
```

```
// Initialise fluid field pointer lists
PtrList<rhoThermo> thermoFluid(fluidRegions.size());
PtrList<volScalarField> rhoFluid(fluidRegions.size());
PtrList<volVectorField> UFluid(fluidRegions.size());
PtrList<surfaceScalarField> phiFluid(fluidRegions.size());
PtrList<uniformDimensionedVectorField> gFluid(fluidRegions.size());
PtrList<compressible::turbulenceModel> turbulence(fluidRegions.size());
PtrList<volScalarField> p_rghFluid(fluidRegions.size());
PtrList<volScalarField> ghFluid(fluidRegions.size());
PtrList<surfaceScalarField> ghfFluid(fluidRegions.size());
PtrList<radiation::radiationModel> radiation(fluidRegions.size());

List<scalar> initialMassFluid(fluidRegions.size());
List<label> pRefCellFluid(fluidRegions.size(),0);
List<scalar> pRefValueFluid(fluidRegions.size(),0.0);

PtrList<dimensionedScalar> rhoMax(fluidRegions.size());
PtrList<dimensionedScalar> rhoMin(fluidRegions.size());

PtrList<fv::IOoptionList> fluidFvOptions(fluidRegions.size());

List<labelList> fluidList(Pstream::nProcs());

// Populate fluid field pointer lists
forAll(fluidRegions, i)
{
    Info<< "*** Reading fluid mesh thermophysical properties for region "
        << fluidRegions[i].name() << nl << endl;

    Info<< "  Adding to thermoFluid" << endl;

    // Selecting thermodynamics package is done right here
    thermoFluid.set
    (
        i,
        rhoThermo::New(fluidRegions[i]).ptr()
    );

    Info<< "  Adding to rhoFluid" << endl;
    rhoFluid.set
    (
        i,
        new volScalarField
        (
            IOobject
            (
                "rho",
                runTime.timeName(),
                fluidRegions[i],
                IOobject::NO_READ,
                IOobject::AUTO_WRITE
            ),
            thermoFluid[i].rho()
        )
    );

    Info<< "  Adding to UFluid" << endl;
    UFluid.set
    (
        i,
        new volVectorField
        (
            IOobject
            (
                "U",
                runTime.timeName(),
                fluidRegions[i],
                IOobject::MUST_READ,
                IOobject::AUTO_WRITE
```



```
            ),
            fluidRegions[i]
        )
    );

    Info<< "  Adding to phiFluid" << endl;
    phiFluid.set
    (
        i,
        new surfaceScalarField
        (
            IOobject
            (
                "phi",
                runTime.timeName(),
                fluidRegions[i],
                IOobject::READ_IF_PRESENT,
                IOobject::AUTO_WRITE
            ),
            linearInterpolate(rhoFluid[i]*UFluid[i])
                & fluidRegions[i].Sf()
        )
    );

    Info<< "  Adding to gFluid" << endl;
    gFluid.set
    (
        i,
        new uniformDimensionedVectorField
        (
            IOobject
            (
                "g",
                runTime.constant(),
                fluidRegions[i],
                IOobject::MUST_READ,
                IOobject::NO_WRITE
            )
        )
    );

    Info<< "  Adding to turbulence" << endl;
    turbulence.set
    (
        i,
        compressible::turbulenceModel::New
        (
            rhoFluid[i],
            UFluid[i],
            phiFluid[i],
            thermoFluid[i]
        ).ptr()
    );

    Info<< "  Adding to ghFluid" << endl;
    ghFluid.set
    (
        i,
        new volScalarField("gh", gFluid[i] & fluidRegions[i].C())
    );

    Info<< "  Adding to ghfFluid" << endl;
    ghfFluid.set
    (
        i,
        new surfaceScalarField("ghf", gFluid[i] & fluidRegions[i].Cf())
    );

    p_rghFluid.set
```

```
    (
        i,
        new volScalarField
        (
            IOobject
            (
                "p_rgh",
                runTime.timeName(),
                fluidRegions[i],
                IOobject::MUST_READ,
                IOobject::AUTO_WRITE
            ),
            fluidRegions[i]
        )
    );

    // Force p_rgh to be consistent with p
    p_rghFluid[i] = thermoFluid[i].p() - rhoFluid[i]*ghFluid[i];

    radiation.set
    (
        i,
        radiation::radiationModel::New(thermoFluid[i].T())
    );

    initialMassFluid[i] = fvc::domainIntegrate(rhoFluid[i]).value();

    setRefCell
    (
        thermoFluid[i].p(),
        p_rghFluid[i],
        fluidRegions[i].solutionDict().subDict("SIMPLE"),
        pRefCellFluid[i],
        pRefValueFluid[i]
    );

    rhoMax.set
    (
        i,
        new dimensionedScalar
        (
            fluidRegions[i].solutionDict().subDict("SIMPLE").lookup
            (
                "rhoMax"
            )
        )
    );

    rhoMin.set
    (
        i,
        new dimensionedScalar
        (
            fluidRegions[i].solutionDict().subDict("SIMPLE").lookup
            (
                "rhoMin"
            )
        )
    );

    Info<< "  Adding fvOptions" << endl;
    fluidFvOptions.set
    (
        i,
        new fv::IOoptionList(fluidRegions[i])
    );

    // 23/04/2015
    // This is only necessary if running in parallel
```



```
    //
    // If running in parallel, the decomposed meshes must have
    // a function to map parallel meshes cells to the region original
    // mesh. The cellProcAddressing file
    // (processor<n>/constant/<region>/polyMesh/cellProcAddressing)
    // This labelIOList will be used to map rho and T values in one processo
r
    // to the complete region mesh.
    //
    // Note: for fluid regions, values are only read from processors.

    if(Pstream::parRun())
    {
      fileName cellCorr(runTime.rootPath()+"/"+runTime.caseName()+
                        "/constant/"+fluidRegions[i].name()+"/polyMesh");

      // Read cellProcAddressing for each processor
      fluidList[Pstream::myProcNo()] = labelIOList(IOobject
         (
          "cellProcAddressing",      // Filename
          cellCorr,
          fluidRegions[i],           // Registry
          IOobject::MUST_READ,       // Read option
          IOobject::NO_WRITE         // Write Option
          ));

      procLabelList[Pstream::myProcNo()] = procList;

      Pstream::gatherList<scalarField>(procScalarFieldList);
      Pstream::scatterList<scalarField>(procScalarFieldList);
    }

}
```

createFluidMeshes.H Page 1/2

```
    const wordList fluidNames(rp["fluid"]);
    PtrList<fvMesh> fluidRegions(fluidNames.size());
    List<labelList> fluidRegionsLists(fluidNames.size());

    forAll(fluidNames, i)
    {
        Info<< "Create fluid mesh for region " << fluidNames[i]
            << " for time = " << runTime.timeName() << nl << endl;

            fluidRegions.set
                (
                    i,
                    new fvMesh
                    (
                        IOobject
                        (
                            fluidNames[i],
                            runTime.timeName(),
                            runTime,
                            IOobject::MUST_READ
                            )
                        )
                    );

        Info << "Create correspondence to the complete mesh for region "
             << fluidNames[i] << " for time = " << runTime.timeName()
             << nl << endl;


        if(Pstream::master())
        {
          std::string caseConst(runTime.caseConstant());

          // If running in parallel, the runTime.caseConstant()
          // adds a relative path to the sequential value.
          // Since the directory where sets file is located is the same,
          // the relative path is removed from the string
          if(Pstream::parRun())
             caseConst.erase(0,3);

          std::string
              regSetPath(caseConst+"/polyMesh/sets/"+fluidNames[i]);
          std::string sedCommand("sed -i 's/cellSet/labelList/g' ");
          std::string command(sedCommand+regSetPath);

          fileName pM("polyMesh");
          fileName setS("sets");
          fileName setsPath(runTime.caseConstant());

          // Information about sets are stored in a cellSet class, which
          // IOobject read as a hashTable. This is a problem to keep the
          // information on data position in the coupling vector.
          //
          // Solution:
          // call to sed using system, replacing cellSet in the set
          // files of each region by labelList. Then OpenFOAM IOobject
          // will be able to direct read data faster.
          system(command.c_str());

          fluidRegionsLists[i] =  labelIOList(IOobject
                         (
                          fluidNames[i],
                          setsPath/pM/setS,
                          runTime,
                          IOobject::MUST_READ,
                          IOobject::NO_WRITE
                          )
            );
        }
```

createFluidMeshes.H Page 2/2

```
    }
```

createSolidFields.H — Page 1/3

```
    // Initialise solid field pointer lists
    PtrList<solidThermo> thermos(solidRegions.size());
    PtrList<radiation::radiationModel> radiations(solidRegions.size());
    PtrList<volScalarField> betavSolid(solidRegions.size());
    PtrList<volScalarField> qVol(solidRegions.size());

    List<List<labelList> > solidList(solidRegions.size(), List<labelList>(Pstrea
m::nProcs()));

    // Populate solid field pointer lists
    forAll(solidRegions, i)
    {
        Info<< "*** Reading solid mesh thermophysical properties for region "
            << solidRegions[i].name() << nl << endl;

        Info<< "  Adding to thermos\n" << endl;
        thermos.set(i, solidThermo::New(solidRegions[i]));

        Info<< "  Adding to radiations\n" << endl;
        radiations.set(i, radiation::radiationModel::New(thermos[i].T()));

        // Read Q if it exists. If not, create a null (zero)
        // volScalarField
        IOobject Qfile
          (
           "Q",
           runTime.timeName(),
           solidRegions[i],
           IOobject::READ_IF_PRESENT,
           IOobject::AUTO_WRITE
           );

        // At this point, Q is zero if the file does not exist
        // or is filled with values from file "Q"

        // Must check it before creating the field
        if(Qfile.headerOk())
          {
            // Create a qvol field from dictionary
            qVol.set
              (
               i,
               new volScalarField (Qfile, solidRegions[i])
               );

          }
        else
          {
            // If file is not there, create a 'dummy' IOobject
            // setting the dimensions of the field
            qVol.set
              (
               i,
               new volScalarField
               (
                IOobject
                (
                 "Q",
                 runTime.path(),
                 solidRegions[i],
                 IOobject::NO_READ,
                 IOobject::AUTO_WRITE
                 ),
                solidRegions[i],
                dimensionedScalar("2", dimensionSet(1, -1, -3, 0, 0),
                                  scalar(0.0))
                )
               );
          }
```

createSolidFields.H — Page 2/3

```
        // Test if Q must be filled for the first iteration
//        forAll(mesh.cells(), i)
//        {
//              Q[i] = 1.0e9;
//        }


        IOobject betavSolidIO
        (
            "betavSolid",
            runTime.timeName(),
            solidRegions[i],
            IOobject::MUST_READ,
            IOobject::AUTO_WRITE
        );

        if (betavSolidIO.headerOk())
        {
            betavSolid.set
            (
                i,
                new volScalarField(betavSolidIO, solidRegions[i])
            );
        }
        else
        {
            betavSolid.set
            (
                i,
                new volScalarField
                (
                    IOobject
                    (
                        "betavSolid",
                        runTime.timeName(),
                        solidRegions[i],
                        IOobject::NO_READ,
                        IOobject::NO_WRITE
                    ),
                    solidRegions[i],
                    dimensionedScalar("1", dimless, scalar(1.0))
                )
            );
        }

        // 23/04/2015
        // This is only necessary if running in parallel
        //
        // If running in parallel, the decomposed meshes must have
        // a function to map parallel meshes cells to the region original
        // mesh. The cellProcAddressing file
        // (processor<n>/constant/<region>/polyMesh/cellProcAddressing)
        // This labelIOList will be used to map rho and T values in one processo
r
        // to the complete region mesh.
        //
        // Note: for solid regions, values are read and written.

        if(Pstream::parRun())
        {
          fileName cellCorr(runTime.rootPath()+"/"+runTime.caseName()+
                            "/constant/"+solidRegions[i].name()+"/polyMesh");

          // Read cellProcAddressing for each processor
          solidList[i][Pstream::myProcNo()] = labelIOList(IOobject
             (
              "cellProcAddressing", // Filename
              cellCorr,
```

```
            solidRegions[i],  // Registry
            IOobject::MUST_READ,      // Read option
            IOobject::NO_WRITE        // Write Option
            ));

        // Create a mapping from each processor to the solidListsRegions
    }
}
```

createSolidMeshes.H


```
    const wordList solidsNames(rp["solid"]);
    PtrList<fvMesh> solidRegions(solidsNames.size());
    List<labelList> solidRegionsLists(solidsNames.size());

    forAll(solidsNames, i)
    {
        Info<< "Create solid mesh for region " << solidsNames[i]
            << " for time = " << runTime.timeName() << nl << endl;

            solidRegions.set
                (
                    i,
                    new fvMesh
                    (
                        IOobject
                        (
                            solidsNames[i],
                            runTime.timeName(),
                            runTime,
                            IOobject::MUST_READ
                            )
                        )
                    );

        Info << "Create correspondence to the complete mesh for region "
             << solidsNames[i] << " for time = " << runTime.timeName()
             << nl << endl;

        if(Pstream::master())
        {
          std::string caseConst(runTime.caseConstant());

          // If running in parallel, the runTime.caseConstant()
          // adds a relative path to the sequential value.
          // Since the directory where sets file is located is the same,
          // the relative path is removed from the string
          if(Pstream::parRun())
             caseConst.erase(0,3);

          std::string
            regSetPath(caseConst+"/polyMesh/sets/"+solidsNames[i]);
          std::string sedCommand("sed -i 's/cellSet/labelList/g' ");
          std::string command(sedCommand+regSetPath);

          fileName pM("polyMesh");
          fileName setS("sets");
          fileName setsPath(runTime.caseConstant());

          // Information about sets are stored in a cellSet class, which
          // IOobject read as a hashTable. This is a problem to keep the
          // information on data position in the coupling vector.
          //
          // Solution:
          // call to sed using system, replacing cellSet in the set
          // files of each region by labelList. Then OpenFOAM IOobject
          // will be able to direct read data faster.
          system(command.c_str());

          solidRegionsLists[i] = labelIOList(IOobject
                         (
                          solidsNames[i],
                          setsPath/pM/setS,
                          runTime,
                          IOobject::MUST_READ,
                          IOobject::NO_WRITE
                          )
              );
        }
```

createSolidMeshes.H


```
    }


// Force calculation of geometric properties to prevent it being done
// later in e.g. some boundary evaluation
//(void)solidRegions[i].weights();
//(void)solidRegions[i].deltaCoeffs();
```

EEqn.H


```
{
    volScalarField& he = thermo.he();

    fvScalarMatrix EEqn
    (
        fvm::div(phi, he)
      + (
            he.name() == "e"
          ? fvc::div(phi, volScalarField("Ekp", 0.5*magSqr(U) + p/rho))
          : fvc::div(phi, volScalarField("K", 0.5*magSqr(U)))
        )
      - fvm::laplacian(turb.alphaEff(), he)
     ==
        rad.Sh(thermo)
      + fvOptions(rho, he)
    );

    EEqn.relax();

    fvOptions.constrain(EEqn);

    solverPerformance sp = EEqn.solve();
//    Info << " --- ADDED: sp.finalResidual = " << sp.finalResidual() << endl;

    fvOptions.correct(he);

    thermo.correct();
    rad.correct();

    Info<< "(Energy) Min/max T:" << min(thermo.T()).value() << ' '
        << max(thermo.T()).value() << endl;
}
```

pEqn.H Page 1/2

```
{
    // Lembrete: rho Ã© um volScalarField criado em createFields
    rho = thermo.rho();
    rho = max(rho, rhoMin[i]);
    rho = min(rho, rhoMax[i]);
    rho.relax();


    volScalarField rAU(1.0/UEqn().A());

    // Interpolate rAu to faces giving it "Dp" name
    surfaceScalarField rhorAUf("Dp", fvc::interpolate(rho*rAU));

    // H by A
    // Remembering that U is a volVectorField defined in setRegionFluidFields.H
    volVectorField HbyA("HbyA", U);
    HbyA = rAU*UEqn().H();

    // Libera o tmp velocidade criado em UEqn.H
    UEqn.clear();

    surfaceScalarField phig(-rhorAUf*ghf*fvc::snGrad(rho)*mesh.magSf());

    // phiHbyA Ã© um a densidade interpolada nas faces vezes
    // o produto interno do vetor HbyA interpolado e
    // os vetores de area das faces na malha
    surfaceScalarField phiHbyA
    (
        "phiHbyA",
        fvc::interpolate(rho)*(fvc::interpolate(HbyA) & mesh.Sf())
    );

    // OpenFOAM 2.2.2
    // fvOptions.relativeFlux(fvc::interpolate(rho), phiHbyA);

    // OpenFOAM 2.4.0
    fvOptions.makeRelative(fvc::interpolate(rho), phiHbyA);

    bool closedVolume = adjustPhi(phiHbyA, U, p_rgh);

    phiHbyA += phig;

    dimensionedScalar compressibility = fvc::domainIntegrate(psi);
    bool compressible = (compressibility.value() > SMALL);

    // Solve pressure
    for (int nonOrth=0; nonOrth<=nNonOrthCorr; nonOrth++)
    {
        fvScalarMatrix p_rghEqn
        (
            fvm::laplacian(rhorAUf, p_rgh) == fvc::div(phiHbyA)
        );

        p_rghEqn.setReference
        (
            pRefCell,
            compressible ? getRefCellValue(p_rgh, pRefCell) : pRefValue
        );

        p_rghEqn.solve();

        if (nonOrth == nNonOrthCorr)
        {
            // Calculate the conservative fluxes
            phi = phiHbyA - p_rghEqn.flux();

            // Explicitly relax pressure for momentum corrector
            p_rgh.relax();
```

pEqn.H Page 2/2

```
            // Correct the momentum source with the pressure gradient flux
            // calculated from the relaxed pressure
            U = HbyA + rAU*fvc::reconstruct((phig - p_rghEqn.flux())/rhorAUf);
            U.correctBoundaryConditions();
            fvOptions.correct(U);
        }
    }

    p = p_rgh + rho*gh;

    #include "continuityErrs.H"

    // For closed-volume cases adjust the pressure level
    // to obey overall mass continuity
    if (closedVolume && compressible)
    {
        p += (initialMass - fvc::domainIntegrate(thermo.rho()))
            /compressibility;
        p_rgh = p - rho*gh;
    }

    rho = thermo.rho();
    rho = max(rho, rhoMin[i]);
    rho = min(rho, rhoMax[i]);
    rho.relax();

    Info<< "Min/max rho:" << min(rho).value() << ' '
        << max(rho).value() << endl;
}
```

setRegionFluidFields.H


```
    const fvMesh& mesh = fluidRegions[i];

    rhoThermo& thermo = thermoFluid[i];
    thermo.validate(args.executable(), "h", "e");

    volScalarField& rho = rhoFluid[i];
    volVectorField& U = UFluid[i];
    surfaceScalarField& phi = phiFluid[i];

    compressible::turbulenceModel& turb = turbulence[i];

    volScalarField& p = thermo.p();
    const volScalarField& psi = thermo.psi();

    fv::IOoptionList& fvOptions = fluidFvOptions[i];

    const dimensionedScalar initialMass
    (
        "initialMass",
        dimMass,
        initialMassFluid[i]
    );

    radiation::radiationModel& rad = radiation[i];

    const label pRefCell = pRefCellFluid[i];
    const scalar pRefValue = pRefValueFluid[i];

    volScalarField& p_rgh = p_rghFluid[i];
    const volScalarField& gh = ghFluid[i];
    const surfaceScalarField& ghf = ghfFluid[i];
```

setRegionSolidFields.H — Page 1/2

```
    solidThermo& thermo = thermos[i];

    tmp<volScalarField> trho = thermo.rho();

    tmp<volScalarField> tcp = thermo.Cp();

    tmp<volScalarField> talpha = thermo.alpha();
    const volScalarField& alpha = talpha();
    tmp<volScalarField> tkappa = thermo.kappa();

    volScalarField& h = thermo.he();

    const volScalarField& betav = betavSolid[i];

    // OpenFoam's original IOoptionsList introduced in version 2.2.2 is removed
    // fv::IOoptionList& fvOptions = solidHeatSources[i];

    // Added to use my own source-term
    // When reading from another file, it should be done at
    // this point, before atributing the qVol to const Q field

    // 13/04/2015
    // ------------------------------------------------------------------
    // CHANGE from former implementation.
    // Q (former qvol) is only read as a default file in OpenFOAM format.
    // if it is not present, Q is assumed 0 in the first step.
    // ------------------------------------------------------------------

    // First of all, only do that if the solid is the fuel
if (solidRegions[i].name() == "fuel")
  {
    // Create a scalar field from the values read from Q file in
    // solids directories
    scalarField sF = scalarField(qVol[i].internalField());
    scalarField completeSF(qVol[i].internalField().size()*Pstream::nProcs(), 0.0
);

    // Values in qVol[i] are dimensioned values.
    // Pay attention on this before assigning it to the vector.

    // This is only necessary if running in parallel
    if(Pstream::parRun())
      {
        fileName cellCorr(runTime.rootPath()+"/"+runTime.caseName()+
                          "/constant/fuel/polyMesh");

        // Read cellProcAddressing for each processor
        labelIOList procList
          (
           IOobject
           (
            "cellProcAddressing",                                    // Name
            cellCorr,
            mesh,                          // Registry
            IOobject::MUST_READ,           // Read option
            IOobject::NO_WRITE             // Write Option
            ),
           procList
           );

        procLabelList[Pstream::myProcNo()] = procList;

        Pout << " --- ADDED: procLabelList[" << Pstream::myProcNo()
             << "]: " << procList;

        Pstream::gatherList<scalarField>(procScalarFieldList);
        Pstream::scatterList<scalarField>(procScalarFieldList);

        Pout << " --- ADDED: procList: " << procList << endl;
```

setRegionSolidFields.H — Page 2/2

```
      }

// If the file is not present, use current qVol[i] (the volScalarField)
const volScalarField& Q = qVol[i];
```

solveFluid.H


```
//  Pressure-velocity SIMPLE corrector

    p_rgh.storePrevIter();
    rho.storePrevIter();
    {
        #include "UEqn.H"
        #include "EEqn.H"
        #include "pEqn.H"
    }

    turb.correct();
```

solveSolid.H


```
{
    for (int nonOrth=0; nonOrth<=nNonOrthCorr; nonOrth++)
    {
        fvScalarMatrix hEqn
        (
           - fvm::laplacian(betav*alpha, h, "laplacian(alpha,h)")
           - Q
//           + fvOptions(rho, h)
//           Retorna um tmp<fvMatrix<type> > depois de fazer operaÃ§Ãµes
//           na matriz do campo e na matriz criada internamente.
//           implementado em fvOptionListTemplates.C
        );

        hEqn.relax();
        hEqn.solve();
    }
}

thermo.correct();

Info<< "Min/max T:" << min(thermo.T()) << ' ' << max(thermo.T()) << endl;
```

UEqn.H


```
// Solve the Momentum equation
tmp<fvVectorMatrix> UEqn
(
    fvm::div(phi, U)
  + turb.divDevRhoReff(U)
 ==
    fvOptions(rho, U)
);

UEqn().relax();

fvOptions.constrain(UEqn());

solve
(
    UEqn()
 ==
    fvc::reconstruct
    (
        (
          - ghf*fvc::snGrad(rho)
          - fvc::snGrad(p_rgh)
        )*mesh.magSf()

        // Important:
        // ghf*fvc::snGrad(rho) comes from pressure adjustment.
        // p_rgh is p adjusted by density.
        // When the gradient is calculated for p_rgh the spurious
        // non-physical term which arises is compensated by
        // subtracting p_rgh
    )
);

fvOptions.correct(U);
```

milonga.mil — Page 1/2

```
MILONGA_DEBUG

# Set transient time to infinite, so milonga will
# wait for OpenFOAM until it ends.
#end_time = infinite
static_steps = 1000

# Make dt = 1. This does not affect milonga calculations
# since it solves a steady-state problem.
# Set for making time-steps named files.
# dt = 1

IF step_static=1

   MESH NAME boa FILE_PATH ../malhas/celula.msh

   MILONGA_PROBLEM FORMULATION diffusion SCHEME volumes DIMENSIONS 3 GROUPS 2

   PRINT "\# Number of cells: " %.0f cells

   VECTOR vec_d SIZE cells

   FUNCTION T(x,y,z) MESH boa CELLS VECTOR vec_T
   FUNCTION Q(x,y,z) MESH boa CELLS VECTOR vec_Q

   WRITE SHM_OBJECT potencias vec_Q
   WRITE SHM_OBJECT temperaturas vec_T
   WRITE SHM_OBJECT densidades vec_d

   SEM calcMil POST
   SEM calcMil WAIT

ELSE

# NÃ£o se pode dar valores para os vetores neste ponto
# ou serÃ£o re-escritos nas outras iteraÃ§Ãµes.
# Na primeira vez, o OpenFOAM tem que passar os
# perfis ou o OpenFOAM Ã© quem deve escrever os arquivos
# de memÃ³ria compartilhada.


SEM calcOf WAIT

READ SHM_OBJECT temperaturas vec_T

PRINT TEXT "\# temperatures correctly read from OpenFOAM."

# Inclusao das funcoes de interpolacao
INCLUDE functions.was

MATERIAL fuel {
D1          d_1_fuel(T(x,y,z))
SigmaA1     SigmaA_1_fuel(T(x,y,z))
nuSigmaF1   SigmanuF_1_fuel(T(x,y,z))
eSigmaF1    1
SigmaS1.1   SigmaS1.1_fuel(T(x,y,z))
SigmaS1.2   SigmaS1.2_fuel(T(x,y,z))
D2          d_2_fuel(T(x,y,z))
SigmaA2     SigmaA_2_fuel(T(x,y,z))
nuSigmaF2   SigmanuF_2_fuel(T(x,y,z))(T(x,y,z))
eSigmaF2    1
SigmaS2.1   SigmaS2.1_fuel(T(x,y,z))
SigmaS2.2   SigmaS1.2_fuel(T(x,y,z))
}

MATERIAL cladding {
D1          d_1_cladding(T(x,y,z))
SigmaA1     SigmaA_1_cladding(T(x,y,z))
SigmaS1.1   SigmaS1.1_cladding(T(x,y,z))
```

milonga.mil — Page 2/2

```
SigmaS1.2   SigmaS1.2_cladding(T(x,y,z))
D2          d_2_cladding(T(x,y,z))
SigmaA2     SigmaA_2_cladding(T(x,y,z))
SigmaS2.1   SigmaS2.1_cladding(T(x,y,z))
SigmaS2.2   SigmaS1.2_cladding(T(x,y,z))
}

MATERIAL coolant {
D1          d_1_coolant(T(x,y,z))
SigmaA1     SigmaA_1_coolant(T(x,y,z))
SigmaS1.1   SigmaS1.1_coolant(T(x,y,z))
SigmaS1.2   SigmaS1.2_coolant(T(x,y,z))
D2          d_2_coolant(T(x,y,z))
SigmaA2     SigmaA_2_coolant(T(x,y,z))
SigmaS2.1   SigmaS2.1_coolant(T(x,y,z))
SigmaS2.2   SigmaS1.2_coolant(T(x,y,z))
}

PHYSICAL_ENTITY NAME walls BC mirror
PHYSICAL_ENTITY NAME extremes BC vacuum
PHYSICAL_ENTITY NAME inlet BC vacuum
PHYSICAL_ENTITY NAME outlet BC vacuum

# O mesmo do OpenFOAM
# Q/m^3

power = [x]

# PRINT TEXT "These two should be equal: " SigmaT1(0,0,0) SigmaA_1_fuel(T(0,0,0))
# PRINT TEXT "This cannot be negative: " SigmaT1(0,0,0)-SigmaS1.1(0,0,0)-SigmaS1.2(0,0,0
)

PRINT TEXT "\# building..." NONEWLINE
MILONGA_STEP JUST_BUILD
PRINT TEXT " done."

PRINT "\# solving..." NONEWLINE
MILONGA_STEP JUST_SOLVE
PRINT TEXT " done."

PRINT "\# cpu time [sec] = "  %.2f time_cpu_build "(build) "  %.2f time_cpu_solve "(solve)"
 SEP " "

MESH_FILL_VECTOR MESH boa CELLS VECTOR vec_Q FUNCTION pow

WRITE SHM_OBJECT potencias vec_Q

SEM calcMil POST

MESH_POST FILE_PATH milonga.vtk pow phi1

ENDIF
```