# DEVELOPMENT OF COMPUTATIONAL FLUID DYNAMICS (CFD) BASED TOPOLOGY OPTIMIZATION CODES IN OPENFOAM

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

NIYAZI ŞENOL

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
ENGINEERING SCIENCES

FEBRUARY 2019

Approval of the thesis:

**DEVELOPMENT OF COMPUTATIONAL FLUID DYNAMICS (CFD) BASED TOPOLOGY OPTIMIZATION CODES IN OPENFOAM**

submitted by **NIYAZI ŞENOL** in partial fulfillment of the requirements for the degree of **Master of Science in Engineering Sciences Department, Middle East Technical University** by,

Prof. Dr. Halil Kalıpçılar
Dean, Graduate School of **Natural and Applied Sciences** _____

Prof. Dr. Murat Dicleli
Head of Department, **Engineering Sciences** _____

Prof. Dr. Ahmet Nedim Eraslan
Supervisor, **Engineering Sciences Department, METU** _____

Prof. Dr. Hasan Umur Akay
Co-supervisor, **Mechanical Eng. Dept., Atılım University** _____

**Examining Committee Members:**

Prof. Dr. Tolga Akış
Civil Engineering Department, Atılım University _____

Prof. Dr. Ahmet Nedim Eraslan
Engineering Sciences Department, METU _____

Assist. Prof. Dr. Hamdullah Yücel
Institute of Applied Mathematics, METU _____

Date: 01.02.2019

**I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.**

Name, Surname: Niyazi Şenol

Signature :

# ABSTRACT

## DEVELOPMENT OF COMPUTATIONAL FLUID DYNAMICS (CFD) BASED TOPOLOGY OPTIMIZATION CODES IN OPENFOAM

Şenol, Niyazi
M.S., Department of Engineering Sciences
Supervisor: Prof. Dr. Ahmet Nedim Eraslan
Co-Supervisor : Prof. Dr. Hasan Umur Akay

February 2019, 64 pages

Optimization has been commonly used by engineers in industry to design and fabricate products efficiently, since it has a great impact on the performance. The recent decades have witnessed an important amount of work in design optimization in structural mechanics. However, topology optimization is less commonly used in fluid mechanics, heat transfer, and thermal-fluid flow problems, especially this thesis aimed to improve its effects on those areas. This work is also dedicated to develop open source codes for topology optimization of different cases. Investigated cases are ducted flow case, pure heat conduction case, elbow flow case, and thermal-fluid flow case. Optimization is also interdisciplinary area that combines various engineering branches and mathematics. In this work different types of topology optimization using adjoint method problems are also investigated. Codes are developed in OpenFOAM environment and added to thesis for researchers willing to improve them. Various libraries of OpenFOAM are used to develop solvers. Also parallel computing study of one of the cases in the thesis is examined in OpenFOAM.

Keywords: Topology Optimization, Multiphysics Optimization, Continuous Adjoint Method, OpenFOAM, Computational Fluid Dynamics (CFD)

# ÖZ

## OPENFOAM'DA HESAPLAMALI AKIŞKANLAR DİNAMİĞİ (HAD) ESASLI TOPOLOJİ OPTİMİZASYON KODLARININ GELİŞTİRİLMESİ

Şenol, Niyazi
Yüksek Lisans, Mühendislik Bilimleri Bölümü
Tez Yöneticisi: Prof. Dr. Ahmet Nedim Eraslan
Ortak Tez Yöneticisi : Prof. Dr. Hasan Umur Akay

Şubat 2019 , 64 sayfa

Optimizasyon endüstride daha verimli üretim ve tasarım yapabilmek için, performansında büyük etkisi olmasından dolayı mühendisler tarafından yaygın bir şekilde kullanılmaktadır. Son yıllarda yapısal mekaniği topoloji optimizasyonunda önemli miktarda çalışma yapıldı. Fakat akışkanlar mekaniği ve ısı transferi konularında çok sık kullanılmıyor, bu tezde bu alanlardaki etkisini artırma amaçlanıyor. Ayrıca bu çalışmada farklı alanlardaki vakalar için açık kaynaklı kodlar geliştirmek planlanıyor. Araştırılacak vakalar, kanallı akış, salt ısı iletimi, dirsek akışı ve termal akış olacaktır. Optimizasyon matematiği ve mühendisliğin farklı dallarını birleştiren çokludisiplinli bir alan. Farklı vakaları çözerken topoloji optimizasyonu ile eklenik metodu kullanıyor. Geliştirilen kodlar OpenFOAM'da derleniyor ve bu konuyu geliştirmek isteyen araştırmacılara yardımcı olması için teze eklendi. Ayrıca tezdeki vakalardan birinin OpenFOAM'da paralel hesaplama çalışması yapılmıştır.

Anahtar Kelimeler: Topoloji Optimizasyonu, Çoklu-fizik Optimizasyonu, Sürekli Ek-

lenik Metodu, OpenFOAM, Hesaplamalı Akışkanlar Dinamiği (HAD)

To the Advancement of Science

## ACKNOWLEDGMENTS

I would like to express my sincere gratitude to my thesis advisor, Prof. Ahmet N. Eraslan. He has been a tremendous force in shaping my attitude toward research over the past two years. I am incredibly lucky to have had such an insightful and understanding mentor, and this work would not have been possible without his understanding attitude. I am indebted to Prof. Hasan U. Akay, my co-advisor of my thesis. He has been continuous source of inspiration and help over the years. He gave me the vision of Open Source and so this thesis could be come to exist. He has also unconditionally supported me the entire time. His love of teaching, studying and ability to explain almost anything are things that I will always admire. Thank you for making my time here truly enjoyable. I would also like to extend my thanks to my committee members: Prof. Tolga Akış and Assist. Prof. Dr. Hamdullah Yücel for taking the time to serve on my thesis committee. Also, I wish to express my sincere thanks to my family and friends for their thrust and understanding throughout the years.

## TABLE OF CONTENTS

APPENDICES

## LIST OF TABLES

TABLES

## LIST OF FIGURES

FIGURES

## LIST OF ABBREVIATIONS

| **Symbol** | **Explanation** |
|---|---|
| $\gamma$ | Design Variable |
| $\alpha$ | Design Variable/Darcy's Term |
| $k$ | Thermal Conductivity Coefficient |
| $\mathbf{v}$ | Primal Velocity |
| $\mathbf{u}$ | Adjoint Velocity |
| $p$ | Primal Pressure |
| $q$ | Adjoint Pressure |
| $T$ | Primal Temperature |
| $T_a$ | Adjoint Temperature |
| $\nu$ | Kinematic Viscosity |
| $\nabla$ | Nabla Operator |
| $\mathbf{n}$ | Normal Vector |
| $\Omega$ | Computational Domain |
| $\Gamma$ | Computational Domain Boundary |
| $J$ | Cost Function |
| $V$ | Volume |

# CHAPTER 1

# INTRODUCTION

## 1.1 Design Optimization

An important aspect of being competitive in engineering design is to reduce development time while using a minimum amount of resources and efforts. Using a Computer-Aided Engineering (CAE) software with design optimization is a good strategy that helps to meet these needs. Traditionally, Computer-Aided Engineering (CAE) software without design optimization has given chance to engineers for analysing only a fixed volume within fixed boundaries in each analysis. In contrast, design optimization is a way to develop optimized geometric designs with volume and shape changes by minimizing or maximizing cost (or objective) functions defined by designer subject to some constraints.

There are two main approaches in design optimization: shape optimization and topology optimization. The latter will be focused on in this thesis, whereas the former is explained briefly in order to compare them.

## 1.2 Shape Optimization

Shape optimization [1, 2] procedure starts with a proposed shape; points on the surface are selected as a design variable. At the last the optimization cycle, the parameters of the shape are determined to ensure the optimal quantity of interest. Optimality quantities of interest/s are formulated continuously and called cost functions. For instance, an example may be given from an aerospace application. In Fig. 1.1, comparison of pressure coefficient, $C_p$, values of based and optimized airfoil can be seen.

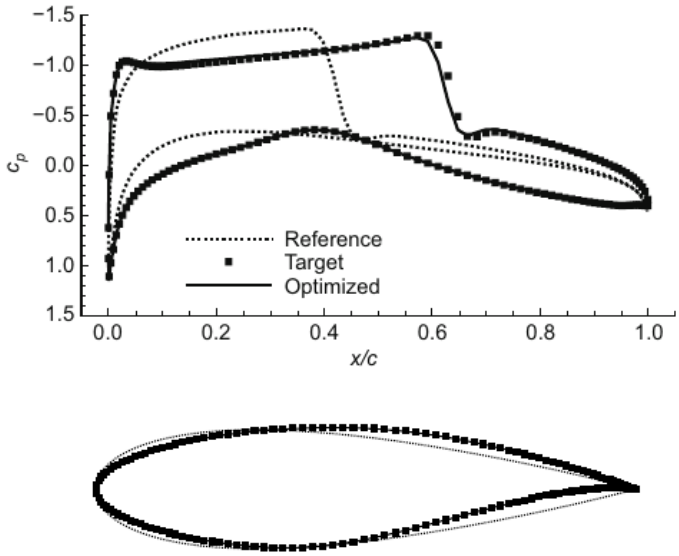


Figure 1.1: An example of airfoil shape optimization [3].

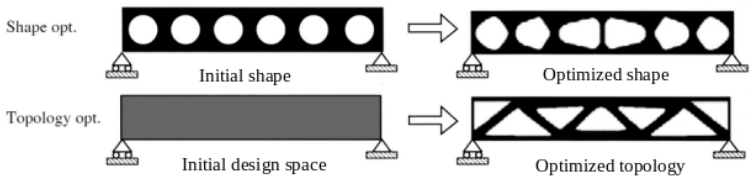


Figure 1.2: An example of structural optimization [2].

Several articles have been published on shape optimization in the area of aerodynamics [1, 3, 4], structural mechanics [5, 6], and manufacturing [7].

## 1.3 Topology Optimization

As opposed to shape optimization, topology optimization begins without a predesigned shape. Therefore, it is different from shape optimization in a fundamental manner. The idea of topology optimization is to determine the template shape itself, as illustrated in Figure 1.2 . Here, there is no predefined shape, instead there is a design space and the constraints of the problem.

Topology optimization is generally formulated as a material distribution problem.

Every cell assigns as a material value function changing between 0 and 1. The name of this approach is 'Solid Isotropic Microstructure with Penalization' (SIMP) [8, 9, 10].

In the algorithm of topology optimization, the domain is changing iteratively by placing or removing solids in a given design space or domain, targeting to minimize/-maximize cost function/s subjected to a constraint such as volume of solid, boundary conditions and other constraints.

The power of topology optimization is on its ability to realize optimal solutions that are not initially obvious to the engineer/designer.

### 1.3.1 Current status of topology optimization

In field of structural mechanics, usage of topology optimization has become a well-established procedure, a compulsory step in some cases (e.g., large scale problems), and has been used to obtain initial conceptual designs. Here, cost function/s to be minimized/maximized are typically the stiffness of the structure [11]. In the last decade, most commercial Computer-Aided Engineering (CAE) software (e.g., OptiStruct, TopoSLang, Comsol, Genesis, MSC/Nastran, Matlab, Ansys, GS Engineering, Tosca, etc.) have modules for topology optimization [12], and thus the technique is now beginning to be used in industry.

The use of topology optimization in industrial applications including flow, heat or thermal-coupled flow is less common. In the recent years a moderate number of articles in flow [13, 14, 15] and heat transfer applications [10, 16] has been published; however, the industry has not yet embraced this area widely. One of the motivations of this thesis is to make progress in this direction, especially using open-source software.

Design procedures are based on numerical analysis methods that evaluate the relative change of value of a set of feasible design variables. The value of a design variable is based on the value of a cost (objective) function that is evaluated using discretization methods such as finite volume and finite element in Computational Fluid Dynamics (CFD) as well as Computational Mechanics areas. The choice of the objective function is essential and requires a knowledge of the design problem at hand. Classical

processes of engineering design using numerical simulations have been carried out by trial and error, relying on the intuition and experience of the designer/engineer to select design parameters. However, while the number of design variables is increasing, the designer's/engineer's ability to make the correct choices decreases. In order to deal with a design space of large dimensionality, numerical analyses need to be combined with optimization procedures.

Topology optimization using CFD is generally a computationally expensive task and traditionally in optimization methods applied in this field the number of necessary flow simulations in the process is highly dependent on the number of the design parameters. To overcome this issue there are some methods available, e.g. , deterministic algorithms [1], evolutionary algorithms [17] and gradient based algorithms [5, 9]. In the present work, for topology optimization a gradient based algorithm known as adjoint method is used. This algorithm will be investigated in Chapter 2.

In the literature, the solution methodology used for analysis part of structural optimization is traditionally based on finite element methods [11, 18]. Finite volume methods are less common in topology optimization for fluid flow, heat transfer or thermal-fluid problems [19, 20].

Computational Fluid Dynamics, CFD, is a numerical analysis method. In this method Numerical analysis, Partial or Ordinary differential equations are transfered into systems of algebraic equations so that they can be solved numerically using computer. To perform this transfer process, there are various methods. The most popular ones are Finite Difference method (FDM), Finite Volume method (FVM) and Finite Element method (FEM). The ideas of these methods are different. FEM includes the partition of the domain into small elements which have points known as nodes. The form of an unknown function (say temperature) is obtained in terms of a basis function expansion. The purpose is to calculate the unknown coefficients of the function expansion so that the residual is minimum. The basic idea of FDM is to change the differential equations by approximations obtained by Taylor expansions near point of interests. FVM involves the division of the computational domain into small volumes. Then conservation laws are applied on these volumes. Thereafter the divergence theorem is employed to change volume integrals to boundary integrals. And finally, the

fluxes are calculated at boundaries. Fluid dynamics involves conservation laws such as conservation of mass, energy and momentum.

OpenFOAM uses finite volume method to solve systems of partial differential equations ascribed on any 3D unstructured mesh of polyhedral cells. The fluid flow solvers are developed within a robust, implicit, pressure-velocity, iterative solution framework, although alternative techniques are applied to other continuum mechanics solvers.

In this thesis, topology optimization of multi-disciplinary thermal fluid problems [10, 23, 24] are addressed, which are less common in literature. For this, incompressible Navier-Stokes and heat transfer equations are solved together with adjoint method with finite volume method. The governing equations for fluid flow and convective heat transfer are called primal equations in optimization context. A set of corresponding adjoint equations and boundary conditions are derived through variational formulations. The obtained equations are implemented in OpenFOAM and example problems are solved to demonstrate the applicability of developed method to different class of problems consisting only flow, only heat transfer and coupled thermal-fluid flow problems. Moreover, since these problems take a lot of computational time and memory, parallel computation capabilities have been tested with the modifications made to flow as well as heat transfer modules of OpenFOAM. The OpenFOAM subpackage developed in this thesis is named 'multiTopOptFoam'

# CHAPTER 2

# MATHEMATICAL FORMULATION

Topology optimization has been a subject of fluid mechanics and heat transfer during the last decade and generally includes three main components. The first component consists of constraints which assign the type of flow or heat transfer being searched. Constraint can imply the conduction heat transfer equation in the devices of MEMS (Microelectromechanical Systems) [16], or Navier-Stokes equations in the aerospace industry [5]. The second component is the objective functions (cost functions) to be maximized or minimized. Typical objectives for airfoil design are minimizing drag force or maximizing lift force [4, 5]. For internal flow objective functions could be minimizing pressure drop through the pipe or obtaining uniform flow at the outlet [15, 19]. The third component is controlling some design parameters. The last one is key actor changing the domain or shape to meet the objective functions, such as the mean temperature of a heat system [16, 20].

As mentioned in Chapter 1, one of the optimization methods is the gradient-based optimization. This kind of algorithm generally needs a continuous objective function that converges to a local minimum. A typical flow chart of gradient based optimization algorithm can be seen in Fig. 2.1. The process starts by evaluating objective function and requires one flow simulation followed by one evaluation of the objective function. Then the gradients of the objective function with respect to the design parameters are calculated. The result of gradient evaluation represents how the objective functions respond to changes in the design variable. The geometry of the domain is modified using values obtained from gradient evaluation. The loop starts again and continues until a solution convergence criterion is satisfied.

**Steepest Descent Method:** This is a commonly used method of gradient based algo-

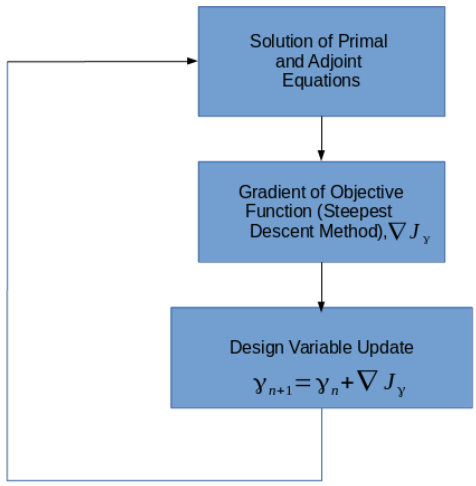


Figure 2.1: Flow chart for gradient based optimization method.

rithm to update design parameters in the optimization step. Steepest Descent Method is aimed to alternate a design parameter (say $\gamma$) by moving in the direction of negative gradient of the objective function. This is mathematically shown as

$$\gamma_{n+1} = \gamma_n - \lambda \nabla J_\gamma = \gamma_n - \lambda \frac{\partial J(\gamma_n)}{\partial \gamma_n}, \tag{2.1}$$

where $\gamma$ is the design variable, $J$ is the objective function, $n$ is the current iteration, and $\lambda$ is the step size. The last term of Eq. 2.1 can be defined by finite difference method using forward differencing:

$$\frac{\partial J}{\partial \gamma} \approx \lim_{h \to 0} \frac{J(\gamma + h) - J(\gamma)}{h}. \tag{2.2}$$

Using Eq. 2.2 to compute gradient of objective functions, it is relatively easier than other algorithms to implement in computer. However, it is not efficient unless the number of design parameters is very few. Adjoint method overcomes this drawback because computational cost is independent of the number of design variables. Besides the cost of computation of the gradients of objective functions is nearly equal to the cost of the solution of primal flow equations.

## 2.1 The Adjoint Method

The adjoint method has been used in various engineering applications. This method started to appear in control theory articles around 1950s. Its range varies from aerospace applications [4, 11] to financial applications [21]. The first appearances of adjoint method in fluid mechanics was for Stokes Flow and introduced by Pironneau [3]. Jameson [4] has made a huge contribution in optimization of airfoils and wings using adjoint methods. At the same time automotive industry has met adjoint method. Othmer [15] used this method for optimization of some components of cars for internal flow.

The main idea behind the adjoint method is to obtain a set of adjoint equations corresponding to a set of primal equations defining the conservation laws of the physical problem, such as structural equilibrium, flow and heat transfer. There are two different approaches to derive adjoint equations: the discrete adjoint method and the continuous adjoint method. In the continuous adjoint method, the primal equations are first linearied, then the adjoint equations are derived from these linearized primal equations; after that the primal and adjoint equations are discretized. In the case of discrete adjoint method, the primal equations are first discretizated of then the adjoint operation is applied to the discretized form of primal equations from which discrete adjoint equations are obtained. A comparison of these steps is illustrated in Fig. 2.2.

## 2.2 The Derivation of Continuous Adjoint Method

In this section, mathematical part of derivation of the continuous adjoint equations will be explained. Then, in the following section, adjoint equations of internal flows will be derived.

The total variation of $J$, where $J$ is an objective function, can be shown as the sum of the variations with respect to the design variable vector, $\gamma$, and the flow variables

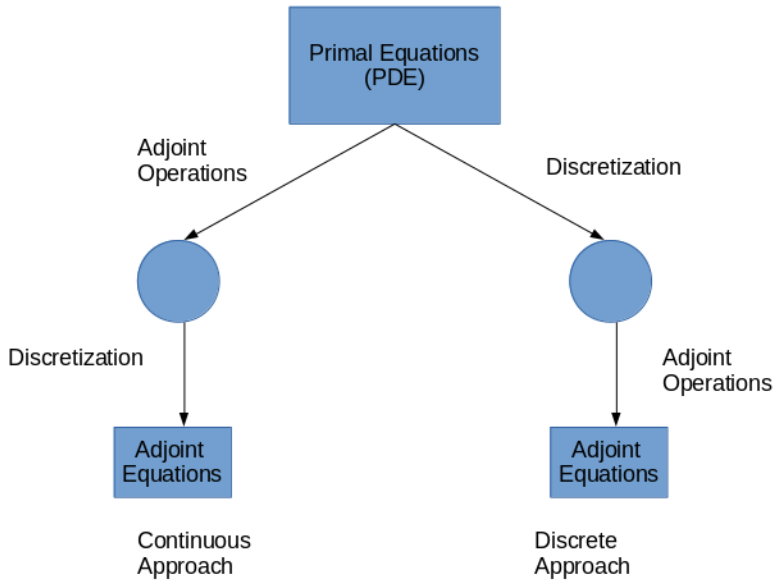


Figure 2.2: Comparison of the continuous adjoint and the discrete adjoint method.

vector, $\boldsymbol{w}$,

$$\delta J = \underbrace{\frac{\partial J}{\partial \boldsymbol{w}} \delta \boldsymbol{w}}_{primalvariable} + \underbrace{\frac{\partial J}{\partial \boldsymbol{\gamma}} \delta \boldsymbol{\gamma}}_{design}. \tag{2.3}$$

The flow equations can be written in residual form as

$$\boldsymbol{R}(\boldsymbol{w}, \gamma) = \boldsymbol{0}. \tag{2.4}$$

Similarly, the total variation of $\boldsymbol{R}$ can be expressed as sum of variations of design parameters and flow variables:

$$\delta \boldsymbol{R} = \frac{\partial \boldsymbol{R}}{\partial \boldsymbol{w}} \delta \boldsymbol{w} + \frac{\partial \boldsymbol{R}}{\partial \boldsymbol{\gamma}} \delta \boldsymbol{\gamma} = \boldsymbol{0}. \tag{2.5}$$

Now we introduce a vector of Lagrange multipliers, $\boldsymbol{\lambda}$, and multiply Eq. 2.5 with Lagrange multiplier as follows:

$$\boldsymbol{\lambda}^T \delta \boldsymbol{R} = \boldsymbol{\lambda}^T \frac{\partial \boldsymbol{R}}{\partial \boldsymbol{w}} \delta \boldsymbol{w} + \boldsymbol{\lambda}^T \frac{\partial \boldsymbol{R}}{\partial \boldsymbol{\gamma}} \delta \boldsymbol{\gamma} = 0. \tag{2.6}$$

By adding Eq. 2.6 to Eq. 2.3, the total variation of $J$, can be expressed as

$$\delta L \equiv \delta J = \left[\frac{\partial J}{\partial \boldsymbol{w}} + \boldsymbol{\lambda}^T \frac{\partial \boldsymbol{R}}{\partial \boldsymbol{w}}\right]\delta \boldsymbol{w} + \left[\frac{\partial J}{\partial \boldsymbol{\gamma}} + \boldsymbol{\lambda}^T \frac{\partial \boldsymbol{R}}{\partial \boldsymbol{\gamma}}\right]\delta \boldsymbol{\gamma}. \tag{2.7}$$

For the adjoint approach, we try to avoid solving for $\delta w$, for this reason the term on the first parenthesis of Eq. 2.7 is equated to zero as follows:

$$\frac{\partial J}{\partial \boldsymbol{w}} + \boldsymbol{\lambda}^T \frac{\partial \boldsymbol{R}}{\partial \boldsymbol{w}} = 0. \tag{2.8}$$

After a basic mathematical operation, the following system of equation are obtained:

$$\boldsymbol{\lambda}^T \frac{\partial \boldsymbol{R}}{\partial \boldsymbol{w}} = -\frac{\partial J}{\partial \boldsymbol{w}} \tag{2.9}$$

Eq. 2.9 is called the adjoint equation. To determine Lagrange multipliers, $\boldsymbol{\lambda}$, Eq. 2.9 has to be solved. Once the Lagrange Multipliers are found, it may now be used to determine the total variation of cost function $J$ in Eq. 2.7 as:

$$\delta L \equiv \delta J = \left[\frac{\partial J}{\partial \boldsymbol{\gamma}} + \boldsymbol{\lambda}^T \frac{\partial \boldsymbol{R}}{\partial \boldsymbol{\gamma}}\right]\delta \boldsymbol{\gamma} \tag{2.10}$$

$$\nabla J = \frac{dJ}{d\boldsymbol{\gamma}} = \frac{\partial J}{\partial \boldsymbol{\gamma}} + \boldsymbol{\lambda}^T \frac{\partial \boldsymbol{R}}{\partial \boldsymbol{\gamma}} \tag{2.11}$$

Consequently we arrived to $\nabla J$ update design variable $\boldsymbol{\gamma}$ on Eq. 2.1.

The steps of continuous adjoint algorithm can be outlined as:

i) Choose an initial design variable vector $\boldsymbol{\gamma}$

ii) Solve primal equations, Eq. 2.4, for flow variables, $\boldsymbol{w}$

iii) Solve adjoint equations, Eq. 2.9, for adjoint variables $\boldsymbol{\lambda}$

iv) Calculate $\nabla J$ from Eq. 2.11

v) Update each design variable $\gamma$ using Eq. 2.1

vi) If $\nabla J$ close to zero, stop the algorithm. Otherwise go to step ii and repeat.

## 2.3 Primal Equations of Thermal-Fluid Flow Problems with Darcy Term

In this thesis, we examine the topology optimization of internal thermal-fluid flow problems in incompressible porous media, where the primal governing equations involve continuity, momentum and heat transfer equations as follows:

$$-\boldsymbol{\nabla} \cdot \boldsymbol{v} = 0, \tag{2.12}$$

$$(\boldsymbol{v} \cdot \boldsymbol{\nabla})\boldsymbol{v} = -\boldsymbol{\nabla} p + \boldsymbol{\nabla}(2\nu \boldsymbol{D}(\boldsymbol{v})) - \alpha(\gamma)\boldsymbol{v}, \tag{2.13}$$

$$C\boldsymbol{v} \cdot (\boldsymbol{\nabla} T) = \nabla \cdot (k(\boldsymbol{\gamma})\boldsymbol{\nabla} T) - Q, \tag{2.14}$$

where $\nabla$ is the gradient vector, $\boldsymbol{v}$ the velocity vector, $p$ pressure, $T$ temperature, $\nu$ kinematic viscosity, $\boldsymbol{D}(\boldsymbol{v}) = \frac{1}{2}(\nabla(\boldsymbol{v}) + \nabla(\boldsymbol{v})^T)$ rate of strain tensor, $\alpha(\boldsymbol{\gamma})$ porosity, $k(\boldsymbol{\gamma})$ thermal conductivity, and the $Q$ volumetric heat source. It is noted that in Eq. 2.13, pressure $p$ and porosity $\alpha$ and in Eq. 2.14 the conductivity $k$ and heat source $Q$ are scaled with respect to $\rho$ for convenience, and $C$ is the specific heat constant. The Darcy's porosity term in the momentum equations above allows the formation of topology as the design parameter $\boldsymbol{\gamma}$ is solved to satisfy the cost function requirements.

**Darcy Term:** This term can be also called porosity term and brings into the momentum equations as a sink term using Darcy's law [5]. Porosity values ($\alpha$) are identified in every cell. Cell porosities act as the design variable of the optimization problem. Porosity values allow a continuous gradation between fluid ($\alpha_{fluid} = 0$) domain and solid ($\alpha_{solid} = \alpha_{max}$) domain. This means that if a cell has a minimum $\alpha$ value, which equal to zero, on momentum equations in Eq. 2.13 Darcy term is zero and so momentum equations consequently become standard Navier-Stokes equations, that is the element will act as a fluid. On the contrary, if a cell has maximum $\alpha$ value, Darcy term will be higher than the other terms on the momentum equations relatively and they are neglected, that is the element will act as a solid. Selecting $\alpha_{max}$ value

depends on Darcy dimensionless number [5]:

$$Da = \frac{v}{\alpha_{max} l^2}, \tag{2.15}$$

where $v$ is inlet velocity, $l$ is a characteristic length, and $Da \leq 10^{-5}$ for nonpermeable domain. So $\alpha_{max}$ can be approximately found with following inequality

$$\alpha_{max} \geq 10^5 \frac{v}{l^2} \tag{2.16}$$

## 2.4 Derivation of Adjoint Equations of Thermal-Flow with Darcy Term

Equations to be solved for internal flows here are incompressible, steady, and modified Navier-Stokes equations. Modified part is extra Darcy term which has a basic role on topology optimization method. Their residuals form are listed on Eq. 2.17. First three equations represent momentum equations, and the last one is continuity equation. The momentum equations are divided by mass density $\rho$ for convenience.

$$\begin{aligned} (R_1, R_2, R_3)^T &= v_j v_{i,j} + p_{,i} - (\nu(v_{i,j} + v_{j,i}))_{,j} + \alpha v_i \\ R_4 &= -v_{i,i} \\ R_5 &= v_i T_{,i} + Q - k T_{,i,i} \end{aligned} \tag{2.17}$$

where $v_i = (v_x, v_y, v_z)$ primal velocity components and $T$ primal temperature variable. Primal pressure divided by density is denoted $p$. The reason we divided the momentum equations by density is that pressure for incompressible flows is treated as divided by density in OpenFOAM. $\nu$ denotes kinematic viscosity, which is same as dynamic viscosity divided by density. Throughout the derivations Einstein's summation notation is used. $\alpha v_i$ term is a Darcy Term, where $\alpha$ is porosity divided by density.

We define an augmented cost function $L$ for Navier-Stokes equations as:

$$L = J + \int_{\Omega} \boldsymbol{\lambda}^T \boldsymbol{R} d\Omega \tag{2.18}$$

where $J$ is cost function to be optimized, $\boldsymbol{\lambda}$ is vector of adjoint variables, e.g. $\boldsymbol{\lambda} = (u_x, u_y, u_z, q, T_a)$.

The total variation of L is

$$\delta L = \underbrace{\delta_{\boldsymbol{w}} L}_{primalvariables} + \underbrace{\delta_{\boldsymbol{\gamma}} L}_{design}, \tag{2.19}$$

where $\boldsymbol{w} = (\boldsymbol{v}, p, T) = (v_x, v_y, v_z, p, T)$. The total variations of primal equations and cost function can be separated with contributions from primal variables and design parameters, respectively as follows:

$$\delta \boldsymbol{R} = \underbrace{\delta_{\boldsymbol{w}} \boldsymbol{R}}_{primalvariables} + \underbrace{\delta_{\boldsymbol{\gamma}} \boldsymbol{R}}_{design} \tag{2.20}$$

$$\delta J = \underbrace{\delta_{\boldsymbol{w}} J}_{primalvariables} + \underbrace{\delta_{\boldsymbol{\gamma}} J}_{design} \tag{2.21}$$

By summing up the above equations we get:

$$\begin{aligned} \delta L &= \delta_{\boldsymbol{w}} L + \delta_{\boldsymbol{\gamma}} L \\ &= \delta J + \int_{\Omega} \boldsymbol{\lambda}^T \delta \boldsymbol{R} d\Omega \\ &= \delta_{\boldsymbol{w}} J + \delta_{\boldsymbol{\gamma}} J + \int_{\Omega} \boldsymbol{\lambda}^T \delta_{\boldsymbol{w}} \boldsymbol{R} d\Omega + \int_{\Omega} \boldsymbol{\lambda}^T \delta_{\boldsymbol{\gamma}} \boldsymbol{R} d\Omega \end{aligned} \tag{2.22}$$

after picking up similar term we obtain:

$$\begin{aligned}\delta L &= \delta_{\boldsymbol{w}} L + \delta_{\boldsymbol{\gamma}} L \\ &= \underbrace{\delta_{\boldsymbol{w}} J + \int_{\Omega} \boldsymbol{\lambda}^T \delta_{\boldsymbol{w}} \boldsymbol{R} d\Omega}_{primalvariables} + \underbrace{\delta_{\boldsymbol{\gamma}} J + \int_{\Omega} \boldsymbol{\lambda}^T \delta_{\boldsymbol{\gamma}} \boldsymbol{R} d\Omega}_{design}.\end{aligned} \tag{2.23}$$

Based on Eq. 2.8, the first term of 2.23 which is coming from flow variable has to be equal to zero

$$\delta_{\boldsymbol{w}} L = \delta_{\boldsymbol{w}} J + \int_{\Omega} \boldsymbol{\lambda}^T \delta_{\boldsymbol{w}} \boldsymbol{R} d\Omega = 0. \tag{2.24}$$

The variation with respect to flow field can also be separated contributions from primal velocities, $\boldsymbol{v}$, primal pressure, $p$, and primal temperature $T$ and expressing $\boldsymbol{\lambda} = (u_i, q, T_a)$

$$\begin{aligned}\delta_{\boldsymbol{w}} L = \delta_{\boldsymbol{v}} J + \delta_p J + \delta_T J + \int_{\Omega} (u_i, q, T_a) \delta_{\boldsymbol{v}} \boldsymbol{R} d\Omega + \\ \int_{\Omega} (u_i, q, T_a) \delta_p \boldsymbol{R} d\Omega + \int_{\Omega} (u_i, q, T_a) \delta_T \boldsymbol{R} d\Omega = 0\end{aligned} \tag{2.25}$$

By taking the variations of $R_i$ with respect to primal velocities and primal pressure, we obtain

$$\begin{aligned}\delta_{\boldsymbol{v}} (R_1, R_2, R_3)^T &= \delta v_j v_{i,j} - (\nu(\delta v_{i,j} + \delta v_{j,i}))_{,j} + \alpha \delta v_i \\ \delta_{\boldsymbol{v}} R_4 &= -\delta v_{i,i} \\ \delta_{\boldsymbol{v}} R_5 &= \delta v_i T_{,i}\end{aligned} \tag{2.26}$$

$$\begin{aligned}\delta_p (R_1, R_2, R_3)^T &= \delta p_{,i} \\ \delta_p R_4 &= 0 \\ \delta_p R_5 &= 0\end{aligned} \tag{2.27}$$

$$
\begin{aligned}
\delta_p(R_1, R_2, R_3)^T &= 0 \\
\delta_p R_4 &= 0 \\
\delta_p R_5 &= u_i \delta T_{,i} - k \delta T_{,ii}
\end{aligned}
\tag{2.28}
$$

Inserting Eq. 2.26, Eq. 2.27 and Eq. 2.28 into Eq. 2.25 gives

$$
\begin{aligned}
\delta_{\boldsymbol{w}} L = 0 &= \delta_{\boldsymbol{v}} J + \delta_p J + \delta_T J \\
&+ \int_\Omega (u_i \delta v_j v_{i,j} - u_i(\nu(\delta v_{i,j} + \delta v_{j,i}))_{,j} + u_i \alpha \delta v_i) d\Omega \\
&- \int_\Omega q \delta v_{i,i} d\Omega + \int_\Omega v_i \delta p_{,i} d\Omega \\
&+ \int_\Omega T_a \delta v_i T_{,i} d\Omega + \int_\Omega T_a \delta T_{,i} v_i d\Omega \\
&- \int_\Omega T_a k \delta T_{,ii} d\Omega
\end{aligned}
\tag{2.29}
$$

The cost function can be split up into contributions from the boundaries $\Gamma$ and interior domain $\Omega$

$$
J = \int_\Gamma J_\Gamma d\Gamma + \int_\Omega J_\Omega d\Omega \tag{2.30}
$$

Collecting terms on above equations in the same integral form, the following equations are obtained.

$$
\begin{aligned}
&\int_\Omega \big(\frac{\partial J_\Omega}{\partial v_i} - v_j u_{j,i} - v_j u_{i,j} + q_{,i} - (\nu(u_{i,j} + u_{j,i}))_{,j} + \alpha u_i\big) \delta v_i d\Omega \\
&+ \int_\Omega \big(\frac{\partial J_\Omega}{\partial p} + u_{i,i}\big) \delta p d\Omega + \int_\Gamma \big(\frac{\partial J_\Gamma}{\partial p} + u_i n_i\big) \delta p d\Gamma \\
&+ \int_\Omega \big(\frac{\partial J_\Omega}{\partial T} - (T_a)_{,i} v_i - k(T_a)_{,ii}\big) \delta T d\Omega \\
&- \int_\Gamma \nu u_i \delta v_{i,j} n_j d\Gamma \\
&+ \int_\Gamma \big(\frac{\partial J_\Gamma}{\partial v_i} + n_i u_j v_j + u_i v_j n_j + \nu u_{i,j} n_j - q n_i\big) \delta v_i d\Gamma = 0
\end{aligned}
\tag{2.31}
$$

Considering that variations of flow variables cannot be zero, every integrand of Eq. 2.31 must be zero. So from this condition we get adjoint equations with volume integrals.

### 2.4.1 Obtained adjoint equations

We obtain the adjoint equations and their boundary conditions from the primal governing equations by following the procedure outlined in previous parts.

$$-\boldsymbol{\nabla} \cdot \boldsymbol{u} = \frac{\partial J_\Omega}{\partial p} \tag{2.32}$$

$$-(\boldsymbol{u} \cdot \boldsymbol{\nabla})\boldsymbol{v} - (\boldsymbol{v} \cdot \boldsymbol{\nabla})\boldsymbol{u} - \boldsymbol{\nabla} \cdot \boldsymbol{\nabla}\nu\boldsymbol{u} = -\boldsymbol{\nabla}\boldsymbol{q} - \alpha(\boldsymbol{\gamma})\boldsymbol{u} + CT(\boldsymbol{\nabla}T_a) - \frac{\partial J_\Omega}{\partial \boldsymbol{v}} \tag{2.33}$$

$$C\boldsymbol{v} \cdot (\boldsymbol{\nabla}T_a) + C\boldsymbol{\nabla} \cdot (k(\boldsymbol{\gamma})\boldsymbol{\nabla}T_a) = \frac{\partial J_\Omega}{\partial T} \tag{2.34}$$

where $\boldsymbol{u}$ is the adjoint velocity, $q$ adjoint pressure divided by $\rho$, $T_a$ adjoint temperature, and $J_\Omega$ total cost function in design domain $\Omega$.

### 2.4.2 Design variable dependent porosity and conductivity

Following the SIMP approach, the variations of porosity and conductivity as a function of the each component of the design variable vector $\boldsymbol{\gamma}$ can be expressed as follows [9]:

$$\alpha(\gamma) = \alpha_{solid} - (\alpha_{solid} - \alpha_{fluid})(1-\gamma)\frac{1+q}{1+q-\gamma}, \tag{2.35}$$

$$k(\gamma) = k_{fluid} + (k_{solid} - k_{fluid})\gamma^p, \tag{2.36}$$

where $q$ (typically 0.1) and $p$ (typically 3) are penalty constants, $\alpha_{fluid}$ and $\alpha_{solid}$ are the fluid (minimum) and solid (maximum) porosity values; $k_{fluid}$ and $k_{solid}$ are the fluid (minimum) and solid (maximum) thermal conductivity values. Optimum topologies are formed with the design variable $\gamma$, which varies from 0 to 1, with $\gamma = 0$ representing the fluid medium (flow passage), and $\gamma = 1$ representing the solid medium.

### 2.4.3 Boundary conditions

The boundary conditions of the primal and the derived adjoint equations are summarized in Table 2.1. The boundary conditions of the adjoint equations are deduced from the variational formulation used while deriving the adjoint equations.

### 2.4.4 Weights for cost functions

In the design optimizations, the total cost function plays an important role affecting the optimization procedure and the equations to be solved. There may be one or more cost functions involved in each problem. If the total cost function has at least two functions, the optimization is called 'multi-objective' optimization. In this paper, our multi-objective/total cost function is a combination of three different cost functions. The first contribution is the flow cost function $J_p$, which aims to minimize the pressure drop, while the second one is the heat transfer cost function $J_h$ intended to minimize the average temperature in the domain. The third cost function $J_v$ provides a fluid volume constraint as a fraction of the total volume of the design domain as $V_f = Vol_f/Vol_{total}$. These three contributions constitute a multi-objective function with varying weight factors as the following:

$$J = w_1 J_f + w_2 J_h + J_v, \tag{2.37}$$

where $w_1$ is the weight factor of the flow cost function and $w_2$ is the weight factor of the heat transfer cost function. The weight factors affect the nature of optimization, and adjusting them changes the character of such optimization. For instance, for $w_1 >> w_2$ values, the total cost function is dominated by the flow cost function and the optimization is focused on minimizing the pressure drop, while by setting the $w_2 >> w_1$ values, the heat transfer cost function becomes dominant. In the implementation of volume constraint $J_v$, the Augmented Lagrange Multiplier (ALM) [5] method is used, where the Lagrangian function $L$ is considered as $J_v$ and defined as:

$$L = J_v = -\lambda_k c_k + w {c_k}^2 \tag{2.38}$$

where $\lambda_k$ is the $k$-th Lagrange multiplier, and $w$ a scalar weight factor. In this way $c_k$ is defined as:

$$c_k = \left[ \frac{\int (1-\gamma) d\Omega}{\int d\Omega} - V_f \right]^2 \tag{2.39}$$

where $V_f$ is the volume fraction of the fluid part.

Table 2.1: Boundary conditions

| Type | Variable | Inlet | Outlet | Wall |
|---|---|---|---|---|
| 1 | $v_i$ | $(v_x\ v_y\ v_z)$ | $\partial v_n/\partial n = 0$, $v_t = 0$ | (0 0 0) |
| 2 | $p$ | $\partial p/\partial n = 0$ | 0 | $\partial p/\partial n = 0$ |
| 3 | T | 0 | $\partial T/\partial n = 0$ | $\partial T/\partial n = 0$ |
| 4 | $u_i$ | $u_n = -\partial J/\partial p$ , $u_t = 0$ | $\partial u_n/\partial n = 0$, $u_t = 0$ | (0 0 0) |
| 5 | $q$ | $\partial q/\partial n = 0$ | 0 | $\partial q/\partial n = 0$ |
| 6 | $T_a$ | 0 | $\partial T_a/\partial n = 0$ | $\partial T_a/\partial n = 0$ |

# CHAPTER 3

# OPENFOAM

## 3.1 Open Source Software

Wikipedia defines open source as "Open-source software (OSS) is a type of computer software in which source code is released under a license in which the copyright holder grants users the rights to study, change, and distribute the software to anyone and for any purpose. Open-source software may be developed in a collaborative public manner. According to scientists who have studied it, open-source software is a prominent example of open collaboration. The term is often written without a hyphen as open source software". New capabilities can be developed and made freely available to scientific community. There are many open source software licenses available in the literature, to name a few:

- GNU General Public License (GPL);
- Apache License;
- BSD License;
- Creative Commons;
- European Commission License (EUPL).

## 3.2 C++ : Object-Oriented Programming (OOP) Language

OpenFOAM uses C++ language, which is a general purpose object-oriented programming (OOP) language, developed by Bjarne Stroustrup in 1979 [22] as a part of his

Ph.D. project, and is an extension of the C language. Moreover it is possible to write code in a "C style" or "object-oriented style" in C++ language. It can be coded in either way and is thus a well-known example of a hybrid language.

C++ can be classified to be an intermediate-level language, as it involves both high level and low level language characteristics. At the beginning, the language was called "C with classes" as it had all the properties of the C language with an additional concept of "classes". However, it was renamed "C++" in 1983.

One of the main features of C++ is a collection of predefined classes, in which data types that can be used more than one. The language also enables declaration of user-defined classes. These classes can further replace member functions to implement specific functionality. Multiple objects of a particular class can be used to implement the functions within the class. Objects can be defined as instances created at run time. These classes can also be inherited by other new classes which place the public and protected functionalities by default.

C++ programming language includes several operators such as comparison, arithmetic, bit manipulation, and logical operators. One of the most attractive features of C++ is that it enables the overloading of certain operators such as vector manipulation or any mathematical operation. Some of the important concepts within the C++ programming language are namespaces, polymorphism, virtual functions, friend functions, pointers, and templates.

## 3.3 C++ and OpenFOAM

C++ is a complex programming language, rich in features. OpenFOAM uses all C++ features. OpenFOAM stands for Open Source Field Operation and Manipulation.

It is licensed under the GNU General Public License (GPL). That means it is freely available and distributed with the source code. It can be used in parallel computers, and there is no need to pay for separate licenses. It is under active development and counts with a wide-spread community around the world (industry, academia and research labs).

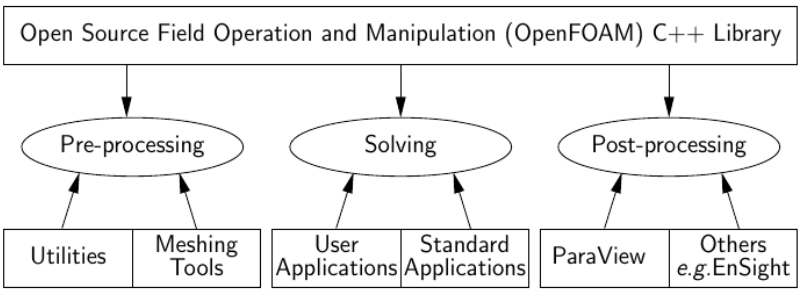


Figure 3.1: Overview of OpenFOAM structure.

OpenFOAM is the first and foremost combined C++ library, used primarily to create executables, known as applications. The applications fall into two categories: solvers, that are designed to solve partial differential equations (PDEs) or ordinary differential equations (ODEs) of a specific problem in continuum mechanics; and utilities, that are designed to perform tasks that involve data manipulation. New solvers and utilities can be created by its users with some prerequisite knowledge of the underlying method, physics and programming techniques involved. OpenFOAM is supplied with pre- and post-processing environments. The interface to the preprocessing and postprocessing are themselves OpenFOAM utilities, thereby ensuring consistent data handling across all environments. The overall structure of OpenFOAM is shown in Figure 3.1.

OpenFOAM has extensive multi-physics capabilities, among others:

- Computational fluid dynamics (compressible and incompressible flows);
- Computational heat transfer and conjugate heat transfer;
- Combustion and chemical reactions;
- Multiphase flows and mass transfer;
- Particle methods and Lagrangian particles tracking;
- Stress analysis and fluid-structure interaction;
- Rotating frames of reference, arbitrary mesh interface, dynamic mesh handling, and adaptive mesh refinement;

• 6 DOF solvers, computational aero-acoustics, computational electromagnetics, computational solid mechanics, etc.;

• Topology optimization with adjoint method for internal flow.

## 3.4 OpenFOAM Terminology

### 3.4.1 Forming the equations

OpenFOAM enables writing differential equations of modelled problem in computer environment by the help of C++ language. Consider following partial equation (momentum equation)

$$\frac{\partial \rho U}{\partial t} + \nabla \cdot \phi U - \nabla \cdot \mu \nabla U = -\nabla p. \tag{3.1}$$

Eq. 3.1 can be transformed into OpenFOAM as following

```
solve
(
    fvm::ddt(rho,U)
    + fvm::div(phi,U)
    - fvm::laplacian(mu,U)
    ==
    - fvc::grad(p)
);
```

Figure 3.2: OpenFOAM notation.

The considered field $U$ can be scalar, vector or tensor. Discretization does not have to be specified at this stage. Above syntax may be considered to be very close to Eq. 3.1.

### 3.4.2 Generating the mesh

Mesh is an important element for numerical analysis. The problem domain is divided to the meshes in order to solve differential equations. In OpenFOAM, meshes are objects defined by $points$, $faces$, $cells$, and $boundary\ patches$. To clarify, a $cell$ is described as a list of faces closing its volume, a $face$ is an ordered list of $points$ and $points$ are ordered list of locations $(x, y, z)$, $patches$ are defined as boundary faces to assign boundary conditions.

### 3.4.3 Time class

While domain is dividing into the meshes, temporal dimension is also breaking to a finite number of $time\ steps$. In OpenFOAM, the $time$ class is handling to track the time step count with time increment $\Delta t$ and to output in every $time\ step$.

### 3.4.4 Fields

Mathematical operators are valid on fields of $scalars$, $vectors$, and $tensors$. Each of them is special $tensor$ with different order. Their class consists of a list of typed values at predefined points of the mesh with tensorial type and associated value. For example, $vector$ class includes vector operations (addition, substraction, scalar multiplication, cross product, magnitude, $etc.$) and thus has three floating point numbers.

## 3.5 Contributions to OpenFOAM

We started to modify an existing solver called adjointShapeOptimizationFoam which optimizes incompressible internal flows. In order to solve pure heat conduction problem, a solver is developed based on adjointShapeOptimizationFoam solver. For thermo-fluid flow problems, we combined these two solvers by considering different features like volume constraints and weight factors and gave a name 'multiTopOptFoam'.

# CHAPTER 4

# NUMERICAL TEST CASES

The computational procedures of topology optimization, theory behind adjoint method and equations developed in Chapter 2 are implemented into OpenFOAM under the name multiTopOptFoam. This chapter includes cases to demonstrate the results and compare them with related references. These cases are Ducted Flow, Pure Heat Conduction Problem, and Thermal-Fluid Flow.

## 4.1 Case 1: Ducted Flow

This case is a topology optimization of a flow splitter manifold commonly arising in typical climatization systems. The problem domain consists of one inlet and four outlets as shown in Figure 4.1. The governing primal and adjoint equations from Eqs. 2.12-2.14 and Eqs. 2.32-2.34 reduce to:

Primal Equations:

$$-\boldsymbol{\nabla} \cdot \boldsymbol{v} = 0 \tag{4.1}$$

$$(\boldsymbol{v} \cdot \boldsymbol{\nabla})\boldsymbol{v} = -\boldsymbol{\nabla} p + \boldsymbol{\nabla}(2\nu \boldsymbol{D}(\boldsymbol{v})) - \alpha(\gamma)\boldsymbol{v} \tag{4.2}$$

Adjoint Equations:

$$-\boldsymbol{\nabla} \cdot \boldsymbol{u} = \frac{\partial J_\Omega}{\partial p} \tag{4.3}$$

$$-(\boldsymbol{u} \cdot \boldsymbol{\nabla})\boldsymbol{v} - (\boldsymbol{v} \cdot \boldsymbol{\nabla})\boldsymbol{u} - \boldsymbol{\nabla} \cdot \boldsymbol{\nabla} \nu \boldsymbol{u} = -\boldsymbol{\nabla} \boldsymbol{q} - \alpha(\boldsymbol{\gamma})\boldsymbol{u} - \frac{\partial J_\Omega}{\partial \boldsymbol{v}} \tag{4.4}$$

The boundary condition of type 1 at the inlet is $v_y = 20m/s$ with $v_x = v_y = 0$ and a

Reynolds number of approximately 300,000 based on characteristic length $0.225\ m$, kinematic viscosity $\nu = 1.5\ 10^{-4} \frac{m^2}{s}$. The zero-velocity boundary condition is used on the walls and the zero-velocity gradient boundary condition is used for the outlets. The boundary condition for pressure is of type 3, for which zero pressure-gradient boundary conditions are applied at inlet and walls, and zero pressure boundary conditions are imposed at the outlets. For the adjoint velocity, the boundary condition is of type 4 in Table 2.1, where $u_x$ and $u_z$ components are zero and $u_y = \frac{\partial J}{\partial p} = -1$ $\frac{m}{s}$. The adjoint velocities are zero for rest of the boundaries. For the adjoint pressure, the boundary condition is of type 5 and it states that it is zero at the outlets and zero-gradient at the inlet and walls.

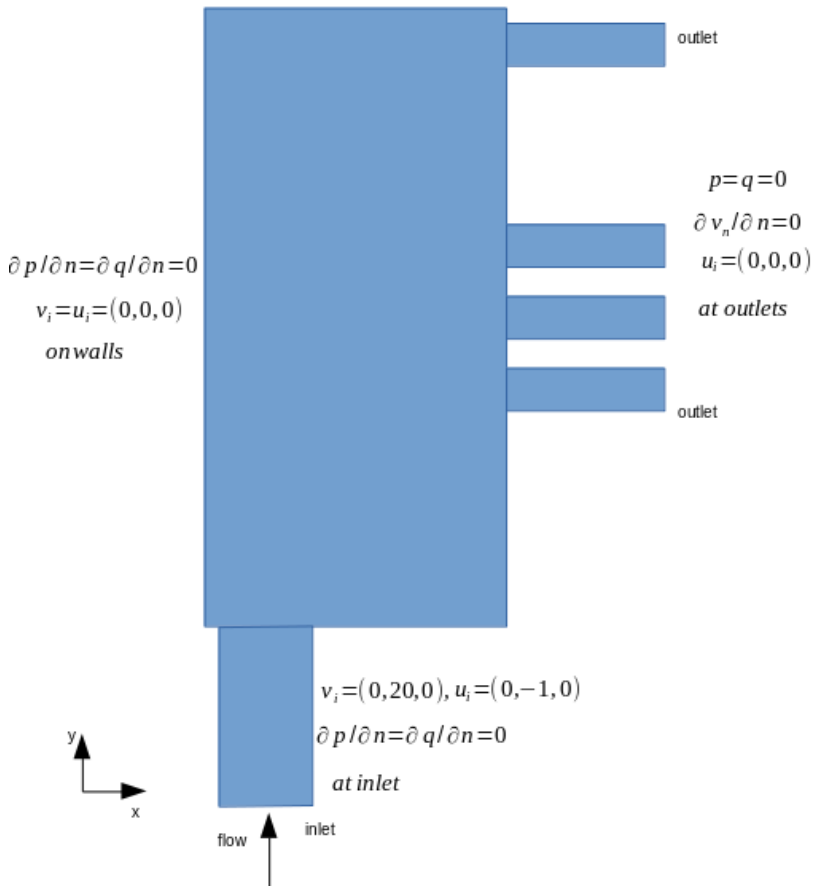


Figure 4.1: Problem domain of ducted flow.

A cost function $J_f$ for minimizing pressure losses between inlet and outlet boundaries in the form

$$J_\Gamma = J_f = d_1 \left(p - p_{outlet}\right)_{inlet} \tag{4.5}$$

where $d_1$ is $1m/s$. Since the cost function is defined as a difference in pressures between inlet and outlet $J_\omega$ is zero. The Eq. 2.14 and Eq. 2.34 are omitted due to the fact that heat transfer is not considered in this case. In Eq. 2.32, the derivative of the cost function with respect to primal pressure appears, and hence, its derivative has to be determined as follows:

$$\frac{\partial J_f}{\partial p} = \frac{\partial}{\partial p}\left(p - p_{outlet}\right)_{inlet} = 1 \tag{4.6}$$

In this case, optimization is done by considering only the flow and the related cost functions and minimizing the pressure drop between the inlet and the outlets. For this purpose, the SIMPLEFOAM solver in OpenFOAM is used. The SIMPLE (Semi-Implicit Method for Pressure Linked Equations) method is intended for constructing. Although the problem is 3D, the design space is a halved symmetry plane to avoid redundant computations. One million hexahedral elements are used for the computations. For this case, frozen turbulence and steady-state assumptions are made. The 'Frozen turbulence' assumption states that, during the calculation of the adjoint variables, turbulent kinetic viscosity $\nu_{eff}$ does not change. Figure 4.2a shows the primal velocity distribution, from which we can conclude that on the left-top and the right-bottom corners, there are recirculation regions. The volume fraction of solid region is satisfied as 0.4. Figure 4.3a depicts the final design variable distribution that implies the shape of the product. Comparison of the results with the results of Othmer [15] in Figures 4.2 (b) and 4.3 (b) are reasonable. In Table 4.1, decrease on cost function value can be seen:

Table 4.1: Cost function values of initial and optimized design, J [$kg/(ms^2)$].

| | |
|---|---|
| Initial Cost Value | $8.3\times10^8$ |
| Optimized Cost Value | $1.2\times10^7$ |

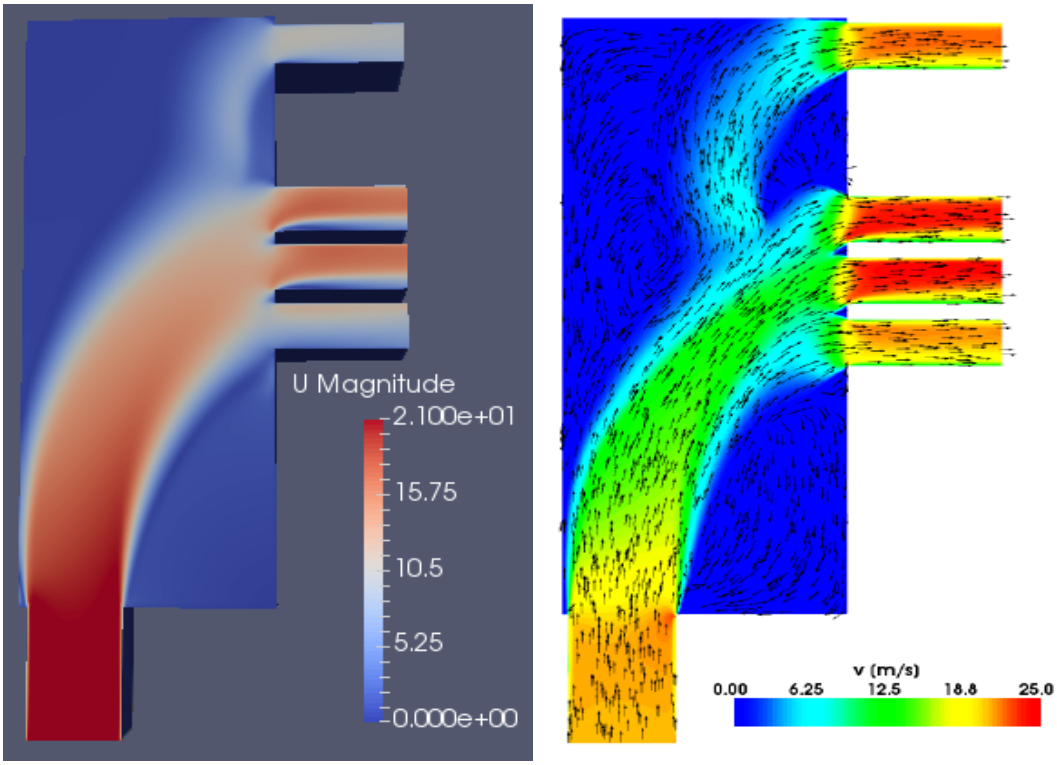


Figure 4.2: (a) Final velocity distribution of case 1, (b) Othmer [15] .

## 4.2 Case 2: Pure Heat Conduction

Although topology optimization of pure heat conduction case with other optimization methods has been studied in various studies, adjoint method has not been widely used in this kind of cases. Pure heat conduction case is a basis problem where to find optimal pattern to move away heat from 2D square plate into a heat sink. In other words, the objective is to minimize mean temperature in the design domain.

For this case, $\gamma$ is design variable and is assigned to each element of the domain. $\gamma$ has minimum ($\gamma_{min} = 0$) and maximum ($\gamma_{max} = 1$) value in this way domain may have different pattern. If an element has $\gamma_{min}$ value, element can be considered as void. If an element has $\gamma_{max}$ value, element can be considered as solid. $\gamma$ controls thermal conductivity, $k$. $k$ is defined as a function of $\gamma$

$$k(\gamma) = k_{min} + (k_{max} - k_{min})\gamma^p \tag{4.7}$$

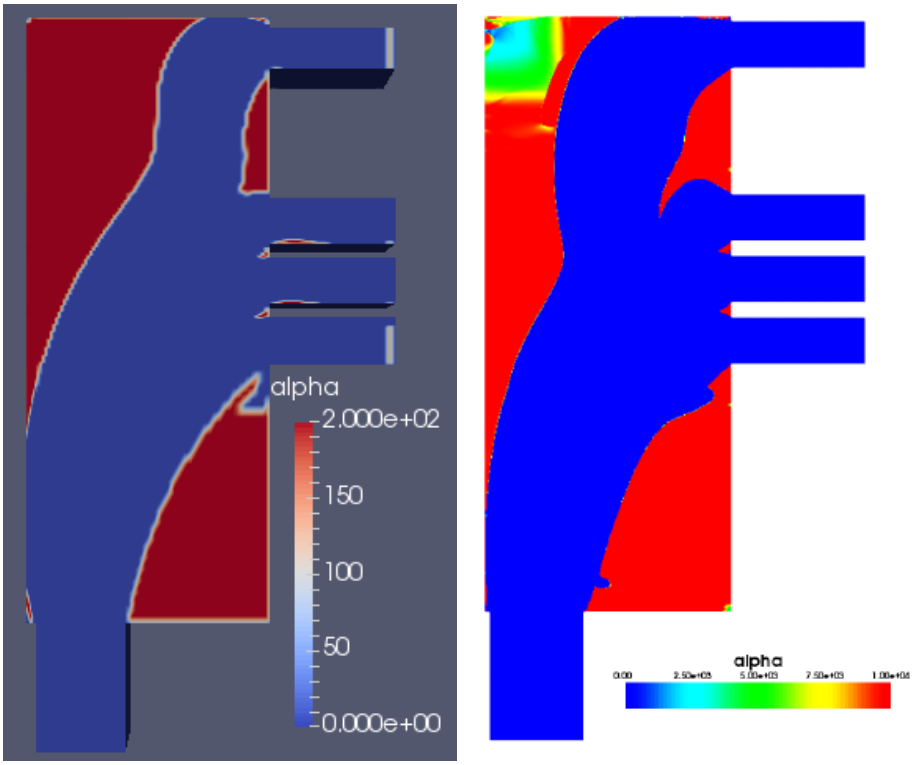


Figure 4.3: Design variable $\alpha$ final distribution of (a) case 1, (b) Othmer [15].

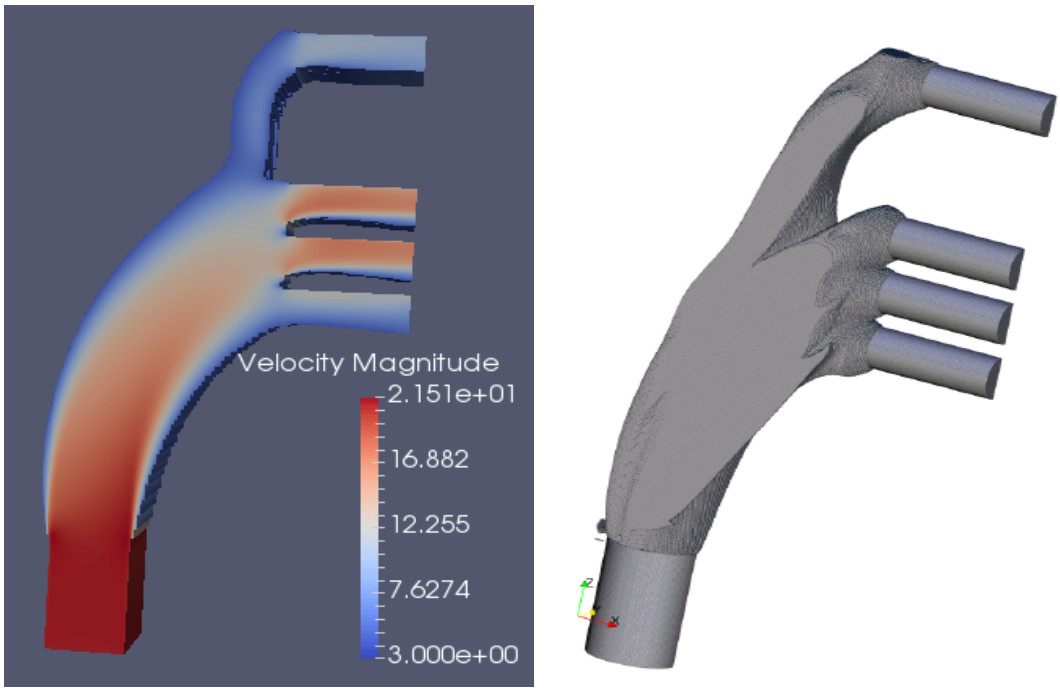


Figure 4.4: Design variable $\alpha$ final distribution of (a) case 1, (b) Othmer [15].

where $p$ is a penalty constant and chosen as $3$ in this case.

Constant and homogeneous heat source $Q$ is applied on whole domain. Boundary conditions are adiabatic(insulating) at walls and heat sink at bottom $T_1 = 273\ K$ for primal variable ($T$).

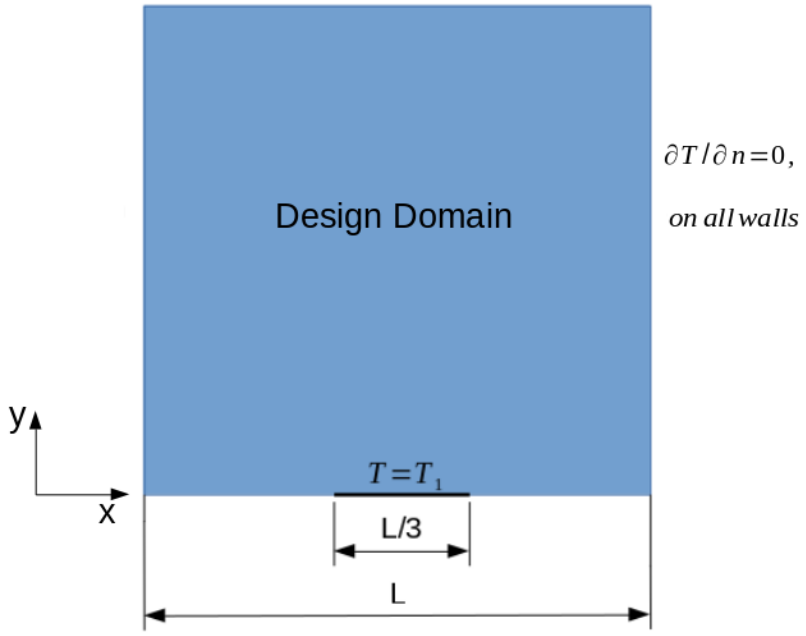


Figure 4.5: Design domain and primal variable boundary conditions.

| | |
|---|---|
| Domain Size | $L \times L = 3 \times 3\ m^2$ |
| $T_1$ | $273\ K$ |
| $k_{max}$ | $1\ m^2/K$ |
| $k_{min}$ | $0.001\ m^2/K$ |
| $Q$ | $0.3$ |
| Volume Fraction($V_f$) | $0.4$ |

Poisson's equation is the governing equation of this case

$$\nabla \cdot (k(\gamma)\nabla T) = -Q \tag{4.8}$$

Adjoint equation of this case is derived in Chapter 2. It can be seen in Eq. 4.9:

$$k(\gamma)\nabla^2 T_a = -\frac{\partial J_{total}}{\partial T} \tag{4.9}$$

where $T_a$ is adjoint variable and $J_{cost}$ is objective (cost) function. As already discussed at the beginning, objective (cost) function is formulated mathematically

$$J_{cost} = \frac{1}{2}\int_{\Omega}(T - T^*)^2 d\Omega \tag{4.10}$$

so total objective function becomes

$$J_{total} = J_{cost} + J_{volume} = \frac{1}{2}\int_{\Omega}(T - T^*)^2 d\Omega + J_{volume} \tag{4.11}$$

where $J_{volume}$ is defined as volume constraint and explained on previous section. Thus Adjoint equation (Eq. 4.9) becomes

$$k(\gamma)\nabla^2 T_a = -(T - T^*) \tag{4.12}$$

Adjoint boundary conditions of this case is derived on Chapter 2. They can be seen in Eq. 4.13:

$$\begin{aligned} k(\gamma)\nabla T_a &= 0 \qquad & wall \\ T_a &= 273 \qquad & heat - sink \end{aligned} \tag{4.13}$$

In order to update design variable values, $\gamma$, Eq. 2.1 from Chapter 2 needs $\nabla J$.

$$\nabla J_{total} = \frac{\partial J_{total}}{\partial \gamma} = \frac{\partial J_{cost}}{\partial \gamma} + \frac{\partial J_{volume}}{\partial \gamma} \tag{4.14}$$

First term on the RHS of Eq. 4.14 is obtained as following, where second term was explained on previous section.

$$\frac{\partial J_{cost}}{\partial \gamma} = 3(k_{max} - k_{min})\gamma^2 \nabla T \cdot \nabla T_a \tag{4.15}$$

**Results**: The domain is discretized using 30000 elements. The optimal design topologies corresponding the case with $V_f$ = 0.4 is shown to compare reasonably well with the results of Dede [17] shown in Figure 4.6b. Following results are obtained.

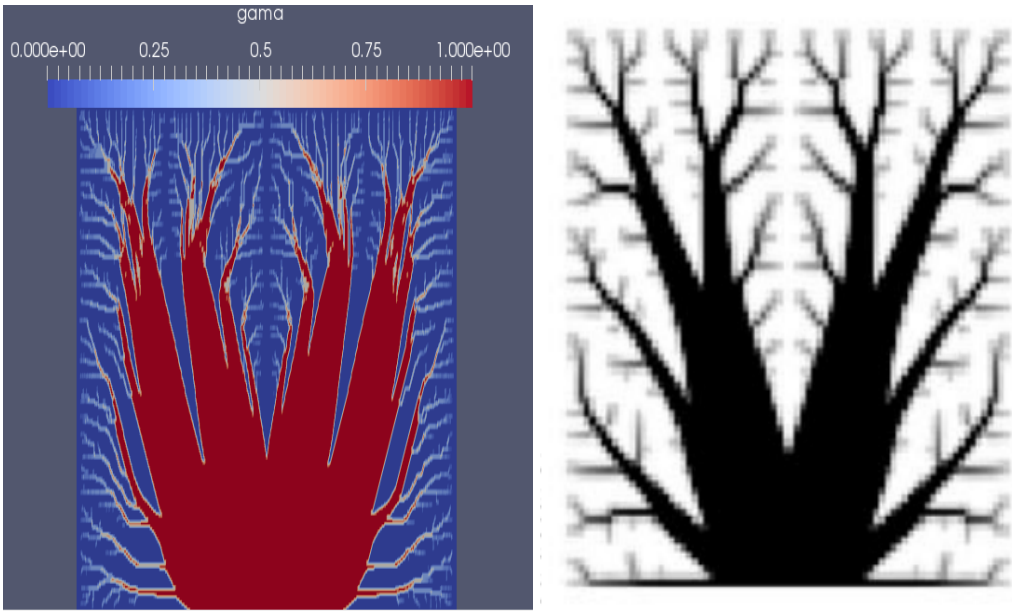


Figure 4.6: Design variable $\gamma$ distribution of (a) $V_f = 0.4$ of case 2, (b) Dede [9].

Table 4.2: Cost function values of initial and optimized design, J $[K]$.

| | |
|---|---|
| Initial Cost Value | 447 |
| Optimized Cost Value | 34 |

### 4.3 Case 3: Coupled Thermal-Fluid Flow

In this case, we solve the topology optimization of thermal-fluid flow case for which multi-objective functions are needed for solution of coupled Navier-Stokes and energy equations are coupled. The coupled characteristic of the problem requires the use of a multiple cost function including both heat transfer and fluid components. Primal equations given by Eqs. 2.12-14 and adjoint equations given by Eqs. 2.32-34 apply to this problem are:

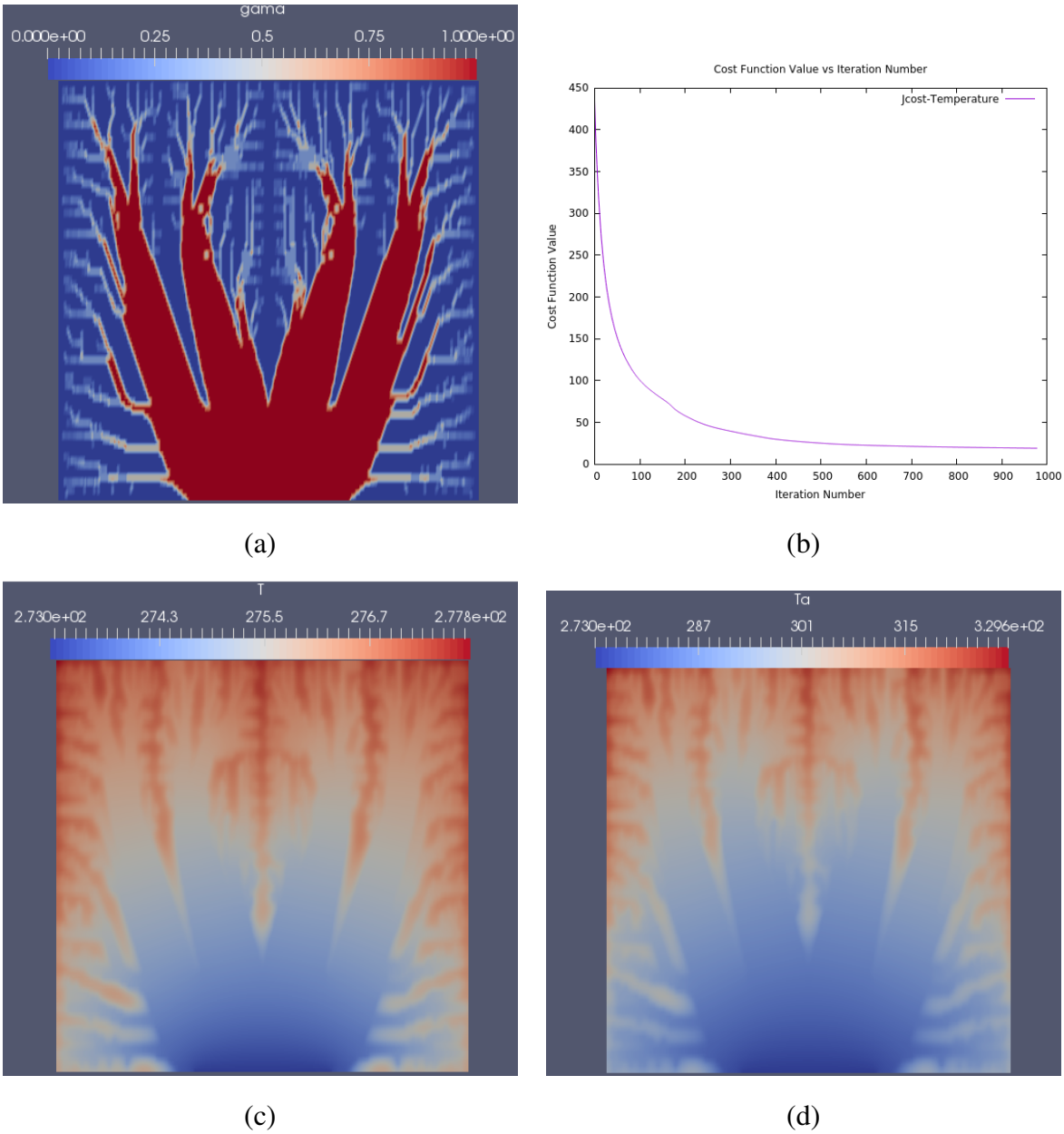


Figure 4.7: (a) $\gamma$ distribution (b) converge of cost function value as a function of iteration number (c) temperature distribution (d) adjoint temperature distribution at optimal solution of case 2.

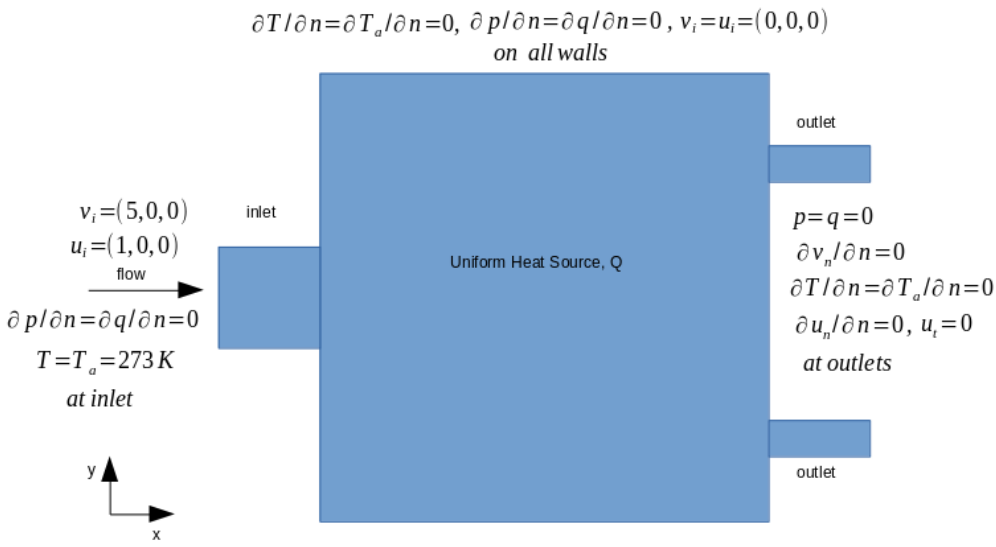


Figure 4.8: Problem domain of coupled thermal-fluid flow.

Primal Equations:

$$-\boldsymbol{\nabla} \cdot \boldsymbol{v} = 0 \tag{4.16}$$

$$(\boldsymbol{v} \cdot \boldsymbol{\nabla})\boldsymbol{v} = -\boldsymbol{\nabla} p + \boldsymbol{\nabla}(2\nu \boldsymbol{D}(\boldsymbol{v})) - \alpha(\gamma)\boldsymbol{v} \tag{4.17}$$

$$C\boldsymbol{v} \cdot (\boldsymbol{\nabla} T) = \nabla \cdot (k(\boldsymbol{\gamma})\boldsymbol{\nabla} T) - Q \tag{4.18}$$

Adjoint Equations:

$$-\boldsymbol{\nabla} \cdot \boldsymbol{u} = \frac{\partial J_\Omega}{\partial p} \tag{4.19}$$

$$-(\boldsymbol{u} \cdot \boldsymbol{\nabla})\boldsymbol{v} - (\boldsymbol{v} \cdot \boldsymbol{\nabla})\boldsymbol{u} - \boldsymbol{\nabla} \cdot \boldsymbol{\nabla}\nu\boldsymbol{u} = -\boldsymbol{\nabla}\boldsymbol{q} - \alpha(\boldsymbol{\gamma})\boldsymbol{u} + CT(\boldsymbol{\nabla} T_a) - \frac{\partial J_\Omega}{\partial \boldsymbol{v}} \tag{4.20}$$

$$C\boldsymbol{v} \cdot (\boldsymbol{\nabla} T_a) + C\boldsymbol{\nabla} \cdot (k(\boldsymbol{\gamma})\boldsymbol{\nabla} T_a) = \frac{\partial J_\Omega}{\partial T} \tag{4.21}$$

Convective heat transfer appears in most industrial applications in which thermal movement occurs. There is heat transfer mechanism between the fluid and the solid region. The focus is on developing the heat transfer between these two regions. Convection heat transfer mechanism is mainly divided into two groups. When movement of the fluid is driven by buoyancy by the reason of density, it is called natural convection. If the flow has gained movement by external forces, it is called forced convection.

The optimization aims at finding an optimum distribution for the design variable $\gamma$, which affects directly the porosity and thermal conductivity functions (Eqs. 2.35-2.36). The problem has a square domain with one inlet and two outlets for fluid flow and constant heat source $Q$ applied over the domain.

The flow enters the domain through the left inlet with a given velocity and temperature and goes out the domain through the right two outlets. On the walls, where the rest of the domain appears, there is a no-slip boundary condition for primal velocity and the adiabatic boundary condition for the primal temperature. For adjoint variables, type 4-6 boundary conditions are used was listed in Table 1. The governing equations for primal and adjoint variables are Eqs. 4.16-21. The multi-cost function is a combination of the flow and heat cost functions explained earlier and formulated in Eq. 2.38. After implementing the cost functions to this equation, we obtain:

$$\begin{aligned} J &= w_1 J_f + w_2 J_h + J_v \\ &= w_1 d_1 \left( \int_{inlet} p d\Gamma - \int_{outlet} p d\Gamma \right) + w_2 \left( \frac{d_2}{2} \int (T - T^*)^2 d\Omega \right) + (-\lambda_k c_k + w {c_k}^2) \end{aligned} \tag{4.22}$$

where $d_1$ and $d_2$ are constants for unit consistency with units $m/s$ and $1/s$, respectively. Another modification to this solver was adding the volume constraint cost function as mentioned before. The volume fraction of fluid region is set to 0.3 in this case. Structured mesh of 20.000 elements are used for computations. The objective of analyzing the combined flow is to optimize the domain based on the weights of cost functions of each problem. In this way, the topology optimization problem can be controlled by selecting the weight factors in the total cost function (Eq. 4.22). Moreover, it is vitally essential to choose right weight factors in the optimization procedure for desired level of optimization. Several optimization studies have been done on this selection process. The results of $w_1 = 1$ and $w_2 = 10$ case are shown as example in Figures 4.10 and 4.11, where the heat transfer cost function is beginning to dominate the solution over the flow cost function. The effect of the higher heat transfer weight is observed with formation of branches around the inlet, which meet again close to the outlets.

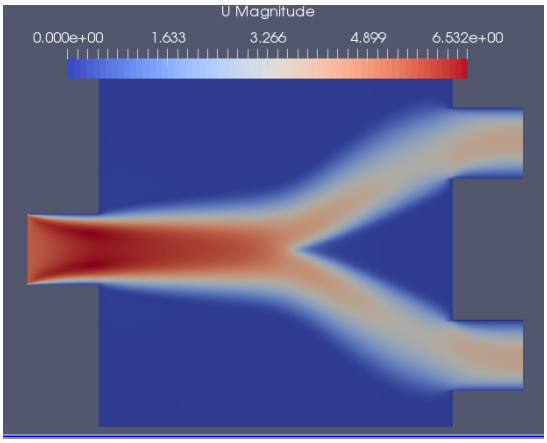


Figure 4.9: Velocity distribution of optimized solution when $w_1 = 1$ and $w_2 = 0$.

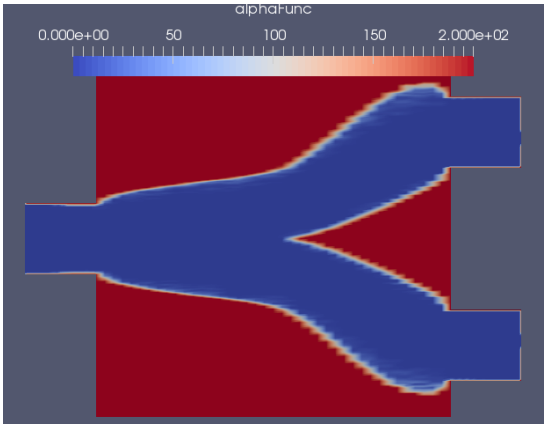


Figure 4.10: $\alpha(\gamma)$ distribution of optimized solution when $w_1 = 1$ and $w_2 = 0$.

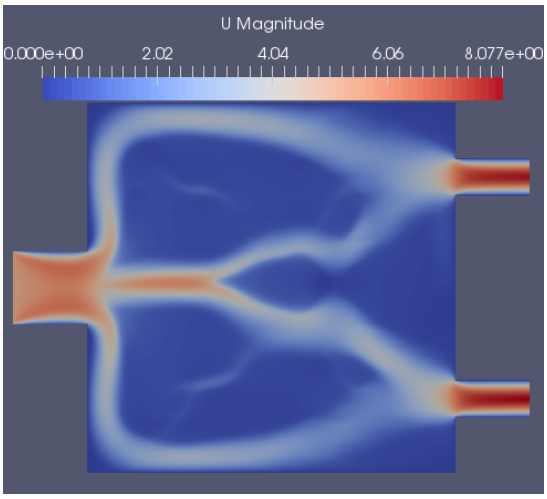


Figure 4.11: Velocity distribution of optimized solution of case 3 when $w_1 = 1$ and $w_2 = 10$.

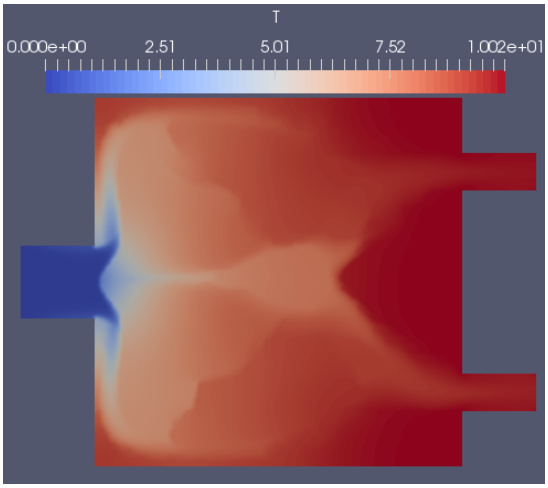


Figure 4.12: Temperature distribution of optimized solution of case 3 when $w_1 = 1$ and $w_2 = 10$.

# CHAPTER 5

# PARALLEL COMPUTING STUDY

Parallel computing study of ducted flow case and pure heat conduction case are examined and its results are shown in this chapter. Domain decomposition parallelism is fundamental to the design of OpenFOAM and integrated at a low level so that solvers can generally be developed without the need for any parallel-specific coding. The underlying aim is to break up the domain with minimal effort but in such a way to guarantee an economic solution. The geometry and fields are broken up according to a set of parameters specified in a dictionary named $decomposeParDict$ that must be located in the $system$ directory of the case of interest. The user has a choice of four methods of decomposition which are simple, hierarchical, scotch, and manual. Simple decomposition method in which the domain is split into pieces by direction, e.g. 3 pieces in the x direction, 2 in y etc. Hierarchical decomposition which is the same as simple method except the user specifies the order in which the directional split is done, e.g. first in the z-direction, then the x-direction and the y-direction etc. Manual decomposition, where the user directly specifies the allocation of each cell to a particular processor. Last method Scotch decomposition which requires no geometric input from the user and attempts to minimise the number of processor boundaries. The parallel computations in this study are carried out using the Message Passing Interface (MPI) library, which ensures proper communication between processors (cores) is used in OpenFOAM. Two different mesh sizes (1 million and 10 million) were used to investigate the speed-up and efficiency of cases 1 and 2 in Tables 5.2 and 5.3, respectively for up to 48 cores. It is seen that the performance of Case 2 is better than Case 1, which is somewhat expected since the mesh subdivisions and interface sizes of the unstructured mesh of Case 1 are almost impossible to balance, whereas the mesh and interface sizes of the structured mesh of Case 2

are easy to balance. We studied effect of using large-scale mesh of topology optimization of internal flow with adjoint method with various mesh decompositions in an open-source platform. Parallel computations have been conducted in Linux environment. For large-scale computing Xean Scalable-6148 processors with 40 CPU cores, 2.40 GHz clock speed and 2048 Gigaflops CPU speed have been used. We are grateful to Scientific and Technological Research Council of Turkey for supporting us by providing access to their cluster.

An example of $scotch$ with 8 decompositions is shown in Figure 5.1:

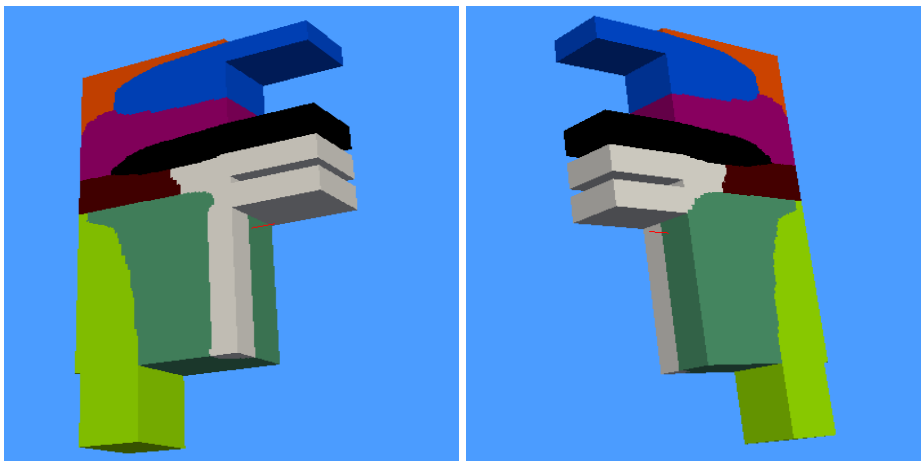

Figure 5.1: Front view of domain decompositions (left) and back view of domain decompositions (right).

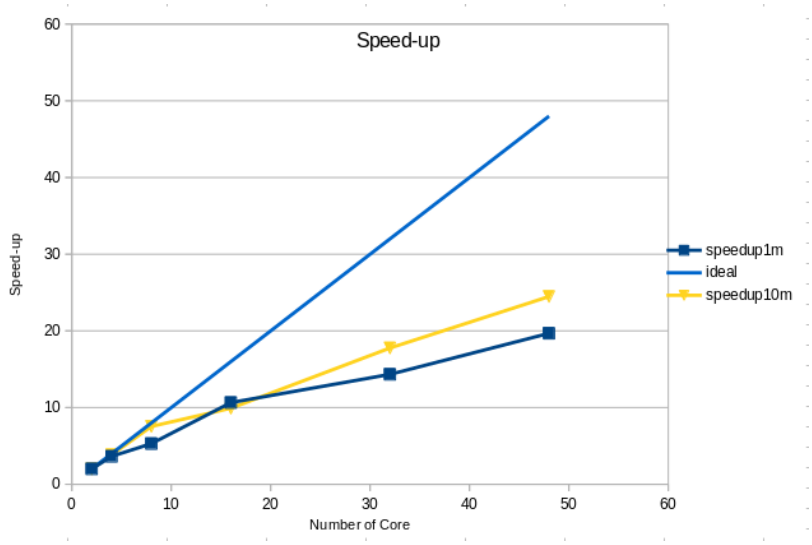


Figure 5.2: Speedup of ducted flow case

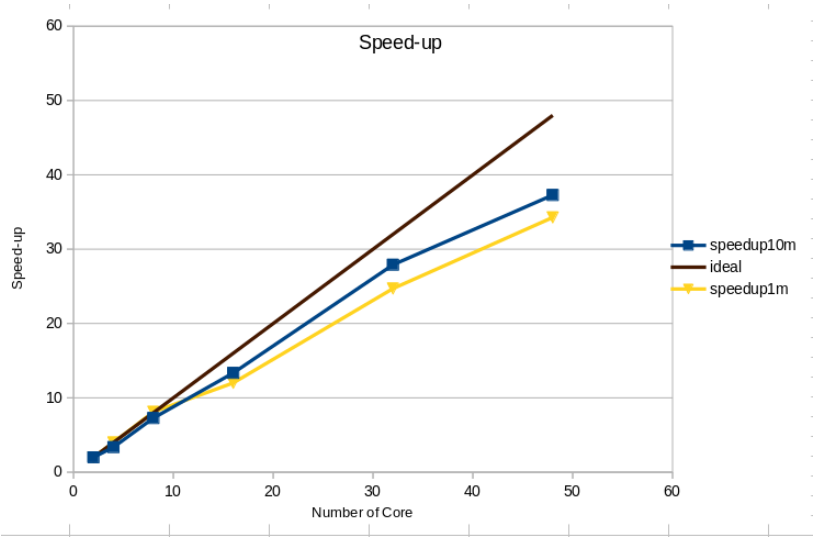


Figure 5.3: Speedup of pure heat conduction case

Table 5.1: Cost function values at $1^{st}$ and $1000^{st}$ iteration, J $[kg/(ms^2)]$.

| Number of Cores | $1^{st}$ iteration | $1000^{st}$ iteration |
|---|---|---|
| 1 | $830895\times10^3$ | $11688\ \times10^3$ |
| 4 | $830619\times10^3$ | $11476\times10^3$ |
| 16 | $830601\times10^3$ | $11224\times10^3$ |

Table 5.2: Speed-up and efficiency of ducted flow case

| Ducted Flow Case | 1M Mesh | | 10M Mesh | |
|---|---|---|---|---|
| Number of Cores | Speed-up | Efficiency (%) | Speed-up | Efficiency (%) |
| 2 | 2 | 100 | 2 | 100 |
| 4 | 3.64 | 91 | 3.85 | 96 |
| 8 | 5.3 | 66 | 7.54 | 94 |
| 16 | 10.65 | 66 | 9.92 | 62 |
| 32 | 14.33 | 44 | 17.77 | 55 |
| 48 | 19.68 | 41 | 24.48 | 51 |

Table 5.3: Speed-up and efficiency of heat conduction case

| Heat Conduct. Case | 1M Mesh | | 10M Mesh | |
|---|---|---|---|---|
| Number of Cores | Speed-up | Efficiency (%) | Speed-up | Efficiency (%) |
| 2 | 2 | 100 | 2 | 100 |
| 4 | 3.36 | 84 | 3.99 | 99 |
| 8 | 7.28 | 91 | 8.11 | 101 |
| 16 | 13.35 | 83 | 12.01 | 75 |
| 32 | 27.9 | 87 | 24.7 | 77 |
| 48 | 37.28 | 77 | 34.27 | 71 |

# CHAPTER 6

# CONCLUSIONS AND FUTURE WORK

## 6.1 Conclusions

The objective of this study has been to develop topology optimization procedures for thermal-fluid problems. The topology optimization employed here is based on the adjoint equation formulation coupled with SIMP formulation in conjunction with a gradient based optimization algorithm. The developed topology optimization methodology is demonstrated with application to ducted-flow, pure heat-transfer and thermal-fluid flow problems, we saw that consideration of forced convection naturally introduces a multi-objective cost function into the problem formulation. Topology optimization for forced convection problems is more complex due to the multi-objective nature of the cost function. Unless care is taken with the problem statement, unexpected (though mathematically defensible) solutions may be obtained. Furthermore, we have taken a relatively simple approach to multi-objective optimization, using a simple linear weighted sum of objective functions to drive the problem. This represents a rich area for future work, and is essential if industrially-relevant problems are to be solved using topology optimization. Finally, parallel study of Ducted-Flow case is investigated to demonstrate its performance.

## 6.2 Recommendations for Future Work

Choosing weight factors of cost functions is a subject that should be investigated. Additionally, models corresponding to example cases analyzed in this thesis can be built in laboratory. Thus, analyzed results can be validated by experimental ones.

Parallelization in multi-core systems is complicated and needs more investigation. It is desired that this thesis will help multi-disciplinary community using topology optimization for different applications in industry.

## REFERENCES


[1] R. N. Gauger. Efficient deterministic approaches for aerodynamic shape optimization. *Optimization and Computational Fluid Dynamics,* 111–145, 2008.

[2] J. Reuther and A. Jameson. Aerodynamic shape optimization of wing and wing-body configurations using control theory. *33rd Aerospace Sciences Meeting and Exhibit,* 1995.

[3] L. JiaQi, X. JunTao, and L. Feng. Aerodynamic design optimization by using a continuous adjoint method. *Physics, Mechanics and Astronomy,* 57(7):1363–1375, 2014.

[4] A. Jameson. Aerodynamic design via control theory. *Journal of Scientific Computing,* 3(3):233–260, 1988.

[5] E. M. Papoutsis-Kiachagias and K. C. Giannakoglou. Continuous adjoint methods for turbulent flows, Applied to Shape and Topology Optimization: Industrial applications. *Computational Methods in Engineering,* 23(2):255–299, 2016.

[6] M.P. Bendsoe and N. Kikuchi. Generating optimal topologies in structural design using a homogenization method. *Computer Methods in Applied Mechanics and Engineering,* 71(2):197–224, 1988.

[7] S. Liu, Q. Li, J. Liu, W. Chen, and Y. Zhang. A realization method for transforming a topology optimization design into additive manufacturing structures. *Engineering,* 4(2):277-285, 2018.

[8] M.P. Bendsoe and O. Sigmund. *Topology Optimization, Theory, Methods and Applications.* Springer, Berlin, 2002.

[9] E.M. Dede, J. Lee, and T. Nomura. *Multiphysics Simulation: Electromechanical System Applications and Optimization* Springer, London, 2014.

[10] A. A. Koga, E. C. C. Lopes, H. F. V. Nova, et al. Development of heat sink device by using topology optimization. *International Journal of Heat and Mass Transfer,* 64(1):759-772, 2013.

[11] E. Oktay, H.U. Akay, and O.T. Sehitoglu. Three-dimensional structural topology optimization of aerial vehicles under aerodynamic loads. *International Journal of Computational Fluid Dynamics,* 92(1):225–232, 2014.

[12] E.M. Dede. Multiphysics topology optimization of heat transfer and fluid flow systems. *The Proceedings of the COMSOL Conference,* 2009.

[13] O. Pironneau. On optimum design in fluid mechanics. *Journal of Fluid Mechanics,* 64(1):97–110, 1974.

[14] T. Borvall and J. Petersson. Topology optimization of fluids in stokes flow. *International Journal for Numerical Methods in Fluids,* 41(1):77–107, 2003.

[15] C. Othmer, E. de Villiers, and H.G. Weller. Implementation of a continuous adjoint for topology optimization of ducted flows. *18th AIAA Computational Fluid Dynamics Conference,* 2007.

[16] E.M. Dede. Optimization and design of a multipass branching microchannel heat sink for electronics cooling. *Journal of Electronic Packaging,* 134(4), 2012.

[17] N. Leon, E. Uresti, and W. Arcos. Fan-shape optimisation using CFD and genetic algorithms for increasing the efficiency of electric motors. *International Journal of Computer Applications in Technology,* 30(1):47–58, 2007.

[18] H.U. Akay, E. Oktay, M. Manguoglu, and A.A. Sivas. Improved parallel preconditioners for multidisciplinary topology optimisations *International Journal of Computational Fluid Dynamics,* 30(4):329-336, 2016.

[19] C. Othmer. Adjoint methods for car aerodynamics. *Journal of Mathematics in Industry,* 4(6), 2014.

[20] T. Matsumori, T. Kondoh, A. Kawamoto, and T. Nomura. Topology optimization for fluid-thermal interaction problems under constant input power. *Structural and Multidisciplinary Optimization,* 47(4):571–581, 2013.

[21] B. D. Craven and S. M. N. Islam. *Optimization in Economics and Finance.* Springer, Netherlands, 2005.

[22] B. Stroustrup. *The C++ Programming Language.* Murray Hill, New Jersey, 1986.

[23] K. Yaji, T. Yamada, S. Kubo, K. Izui, and S. Nishiwaki. A topology optimization method for a coupled thermal-fluid problem using level set boundary expressions. *International Journal of Heath and Mass Transfer,* 81(1):878-888, 2015.

[24] J. K. Guest and J. H. Prevost. Topology optimization of creeping fluid flows using a Darcy-Stokes finite element. *International journal for numerical methods in engineering,* 66(3):461-484, 2006.

# APPENDIX A

# OPENFOAM CODES

In this appendix, codes made in OpenFOAM solver multiTopOptFoam are shown for ducted flow, pure heat transfer and thermal-fluid flow cases, respectively.

## A.1 case 1 : Ducted Flow

```
#include "fvCFD.H"
#include "singlePhaseTransportModel.H"
#include "turbulentTransportModel.H"
#include "simpleControl.H"
#include "fvOptions.H"
template<class Type>
void zeroCells
(
GeometricField<Type, fvPatchField, volMesh>& vf,
const labelList& cells
)
{
forAll(cells, i)
{
vf[cells[i]] = 0;
}
}
```

```
int main(int argc, char *argv[])
{
#include "postProcess.H"
#include "setRootCase.H"
#include "createTime.H"
#include "createMesh.H"
#include "createControl.H"
#include "createFields.H"
#include "createFvOptions.H"
#include "initContinuityErrs.H"
#include "initAdjointContinuityErrs.H"
turbulence->validate();

Info<< "\nStarting time loop\n" << endl;
dimensionedScalar lambdaVol("lambdaVol",dimless,0.0);
dimensionedScalar weightFactor("weightFactor",dimless,1.01);
while (simple.loop())
{
Info<< "Time = " << runTime.timeName() << nl << endl;
laminarTransport.lookup("lambda") >> lambda;
laminarTransport.lookup("Vtarget") >> Vtarget;

scalar volFraction(0.0);
scalar totalVolume(0.0);

forAll(mesh.cells(),celli)
{
volFraction +=  ( one.value() - alpha[celli]/alphaMax.value() )
* mesh.V()[celli]  ;
totalVolume += mesh.V()[celli];

}
Info<< "totalVolume = " <<  totalVolume << "  " <<endl;
```

```
Info<< "volFraction = " <<  volFraction << "  " <<endl;
scalar cVol = Foam::sqr( volFraction/totalVolume
- Vtarget.value() );
Info<< "cVol = " <<  cVol << "  " <<endl;
scalar stepOne = -lambdaVol.value() + iki.value()
*weightFactor.value()*cVol;
Info<< "stepOne = " <<  stepOne << "  " <<endl;
scalar cVolDerivative = -iki.value()/(alphaMax.value()
*totalVolume)
* ( volFraction/totalVolume - Vtarget.value() ) ;
Info<< "cVolDerivative = " <<  cVolDerivative ;
scalar stepTwo =  stepOne * cVolDerivative;
Info<< "stepTwo = " <<  stepTwo << "  " <<endl;
alpha +=
mesh.fieldRelaxationFactor("alpha")
*(min(max(alpha + lambda
*(nn*( Ua & U)- stepTwo), zeroAlpha), alphaMax) - alpha);


zeroCells(alpha, inletCells);
zeroCells(alpha, outletCells);

forAll(mesh.boundary()["outlet"], faceI)
//patchI: label of your target patch
{
alpha[mesh.boundary()["outlet"].faceCells()[faceI]] = 0;
}

const labelList& ids = mesh.cellZones().findIndices("outlet");
forAll(ids, i)
{
const labelList& cells = mesh.cellZones()[ids[i]];
forAll(cells, j)
```

```
{
label celli = cells[j];
alpha[celli] = 0.0;
}
}

label inletPatchID = mesh.boundaryMesh()
.findPatchID("outlet");
//Patch identifier of the boundary condition
fvPatchScalarField wallP = alpha
.boundaryField()[inletPatchID];
// Obtain a patchScalar field on the BC

forAll(wallP, faceI)  // Loop over each face of the patch
{
alpha[faceI]=0;
}
lambdaVol += -two.value() * weightFactor.value()*cVol;
Info<< "lambdaVol = " <<  lambdaVol << "  " <<endl;
weightFactor = min
(stepp*weightFactor.value(),weightFactorMax);
Info<< "weightFactor = " <<  weightFactor ;

scalar aveAlpha(0.0);
forAll(mesh.cells(),celli)
{
aveAlpha += mesh.V()[celli] * alpha[celli];
}
scalar AlphaMean=
 aveAlpha / (totalVolume*alphaMax.value())  ;

Info<< "AlphaMean = " <<  AlphaMean << "  " <<endl;
Info<< "totalVolume = " <<  totalVolume << "  " <<endl;
```

```
label patchID = mesh.boundaryMesh().findPatchID("inlet");
scalar Jpressure(0.0);
forAll(p.boundaryField()[patchID],faceI)
{
Jpressure += p.boundaryField()[patchID][faceI];
}
Info<< "Jpressure = " <<  Jpressure << "  " <<endl;

// Pressure-velocity SIMPLE corrector
{
#include "UPEqn.H"
}
// Adjoint Pressure-velocity SIMPLE corrector
{
#include "UaPaEqn.H"
}
laminarTransport.correct();
turbulence->correct();
runTime.write();
Info<< "ExecutionTime = "
<< runTime.elapsedCpuTime()
<< " s\n\n" << endl;
}
Info<< "End\n" << endl;
return 0;
}
```

### A.2 case 2 : Pure Heat Conduction

```
#include "fvCFD.H"
#include "fvOptions.H"
#include "simpleControl.H"
```

```
int main(int argc, char *argv[])
{
#include "setRootCase.H"
#include "createTime.H"
#include "createMesh.H"
simpleControl simple(mesh);
#include "createFields.H"
// #include "createFvOptions.H"

Info<< "\nCalculating temperature distribution\n";
dimensionedScalar lambdaVol
("lambdaVol",dimless,0.0);
dimensionedScalar weightFactor
("weightFactor",dimless,1.01);

while (simple.loop())
{
Info<< "Time = " << runTime.timeName() << nl;
transportProperties.lookup("lambda") >> lambda;
transportProperties.lookup("Vtarget") >> Vtarget;
while (simple.correctNonOrthogonal())
{
volVectorField gradT(fvc::grad(T));
volVectorField gradTa(fvc::grad(Ta));
volScalarField gradTgradTa(gradTa & gradT);
volVectorField gradTgama(fvc::grad(gama));
volScalarField gradTgradgama(gradT & gradTgama);

volScalarField DthT(Dth*(0.001+0.999*gama*gama*gama)) ;
volScalarField DthTa(DthT*Tatt) ;


scalar costFunctionCell(0.0);
```

```
scalar volFraction(0.0);
scalar totalVolume(0.0);
forAll(mesh.cells(),celli)
{
volFraction +=
( bir.value() - DthT[celli] ) * mesh.V()[celli]  ;
totalVolume += mesh.V()[celli];
costFunctionCell += (T[celli]-273)* (T[celli]-273);
}
Info<< "totalVolume = " <<  totalVolume;

Info<< "volFraction = " <<  volFraction;
scalar costFunctionTotal = costFunctionCell
*totalVolume / 2;
Info<< "Totalvolume" << totalVolume;

scalar cVol = Foam::sqr( volFraction/totalVolume -
Vtarget.value() );
Info<< "cVol=" << cVol<<endl;
scalar stepOne = -lambdaVol.value()
+ iki.value()*weightFactor.value()*cVol;
Info<< "stepOne=" << stepOne<<endl;

scalar cVolDerivative = -iki.value()/(totalVolume)
* ( volFraction/totalVolume - Vtarget.value() ) ;
Info<< "cVolDerivative=" << cVolDerivative<<endl;
scalar stepTwo =  stepOne * cVolDerivative;
Info<< "stepTwo=" << stepTwo<<endl;

gama +=
mesh.fieldRelaxationFactor("gama")
*(min(max( gama+lambda*(Untless * Dth * gama * gama
* gradTgradTa*Tcnstt-stepTwo) , zeroGama)
```

```
, gamaMax) - gama);

lambdaVol += -iki.value() * weightFactor.value()*cVol;
Info<< "lambdaVol = " <<  lambdaVol;
weightFactor = min(stepp*weightFactor.value(),weightFactorMax);
Info<< "weightFactor = " <<  weightFactor;

solve
(
fvm::laplacian(DthT , T) + Q
);

solve
(
fvm::laplacian(DthTa , Ta) +   T-Ttt
);

Info<< "End\n" << endl;
Info<< "costFunctionTotal = " << costFunctionTotal<<endl;
}
runTime.write();
Info<< "ExecutionTime = " << runTime.elapsedCpuTime()
<< "  ClockTime = " << runTime.elapsedClockTime();

}
return 0;
}
```

### A.3 case 3 : Coupled Thermal-Fluid Flow

```
#include "fvCFD.H"
```

```
#include "singlePhaseTransportModel.H"
#include "turbulentTransportModel.H"
#include "simpleControl.H"
#include "fvOptions.H"
template<class Type>
void oneCells
(
GeometricField<Type, fvPatchField, volMesh>& vf,
const labelList& cells
)
{
forAll(cells, i)
{
vf[cells[i]] = 0;
}
}
template<class Type>
void zeroCells
(
GeometricField<Type, fvPatchField, volMesh>& vf,
const labelList& cells
)
{
forAll(cells, i)
{
vf[cells[i]] = Zero;

}
}
int main(int argc, char *argv[])
{
#include "postProcess.H"
#include "setRootCase.H"
```

```
#include "createTime.H"
#include "createMesh.H"
#include "createControl.H"
#include "createFields.H"
#include "createFvOptions.H"
#include "initContinuityErrs.H"
#include "initAdjointContinuityErrs.H"
turbulence->validate();
Info<< "\nStarting time loop\n" << endl;
while (simple.loop())
{
Info<< "Time = " << runTime.timeName() << nl << endl;
laminarTransport.lookup("lambda") >> lambda;


eta +=
mesh.fieldRelaxationFactor("eta")
*(min(max(eta - lambda * (ll*q * (1+q)
* (alMin-alMax)/(eta+q)/(eta+q)
* (Ua & U ) +therm*(ks-kf)
*Ta*fvc::laplacian(T)), zeroEta), etaMax)- eta);

zeroCells(eta, inletCells);
alpha = (alMax+(alMin-alMax) * eta * (1+q)/(eta+q)) ;
thermalK = (eta*kf+(1-eta)*ks) ;
// Pressure-velocity SIMPLE corrector
{
// Momentum predictor
tmp<fvVectorMatrix> tUEqn
(
fvm::div(phi, U)
+ turbulence->divDevReff(U)
+ fvm::Sp(alpha*hh, U)
```

```
==
fvOptions(U)
);
fvVectorMatrix& UEqn = tUEqn.ref();
UEqn.relax();
fvOptions.constrain(UEqn);
solve(UEqn == -fvc::grad(p));
fvOptions.correct(U);
volScalarField rAU(1.0/UEqn.A());
volVectorField HbyA(constrainHbyA
(rAU*UEqn.H(), U, p));
tUEqn.clear();
surfaceScalarField phiHbyA
("phiHbyA", fvc::flux(HbyA));
adjustPhi(phiHbyA, U, p);
// Update the pressure BCs to ensure flux consistency
constrainPressure(p, U, phiHbyA, rAU);
// Non-orthogonal pressure corrector loop
while (simple.correctNonOrthogonal())
{
fvScalarMatrix pEqn
(
fvm::laplacian(rAU, p) == fvc::div(phiHbyA)
);
pEqn.setReference(pRefCell, pRefValue);
pEqn.solve();
if (simple.finalNonOrthogonalIter())
{
phi = phiHbyA - pEqn.flux();
}
}
#include "continuityErrs.H"
// Explicitly relax
```

```
//pressure for momentum corrector
p.relax();


// Momentum corrector
U = HbyA - rAU*fvc::grad(p);
U.correctBoundaryConditions();
fvOptions.correct(U);

}
//Info<<"phi"<<" "<<phi<<endl;
solve
(
fvm::div(phi, T)
- fvm::laplacian(thermalK*qqqq, T)
);

solve
(
fvm::div(phi, Ta)
- fvm::laplacian(thermalK*qqqq, Ta)
);

Info<<"Temperature"<<" "<<T<<endl;
// Adjoint Pressure-velocity SIMPLE corrector
{
// Adjoint Momentum predictor
volVectorField adjointTransposeConvection((
fvc::grad(Ua) & U));
zeroCells(adjointTransposeConvection, inletCells);
tmp<fvVectorMatrix> tUaEqn
(
fvm::div(-phi, Ua)
```

```
- adjointTransposeConvection
+ turbulence->divDevReff(Ua)
+ fvm::Sp(alpha*hh, Ua)
==
fvOptions(Ua)
);
fvVectorMatrix& UaEqn = tUaEqn.ref();

UaEqn.relax();
fvOptions.constrain(UaEqn);
solve(UaEqn == fvc::grad(pa) - Ta
* fvc::grad(T)*ttt);
fvOptions.correct(Ua);
volScalarField rAUa(1.0/UaEqn.A());
volVectorField HbyAa("HbyAa", Ua);
HbyAa = rAUa*UaEqn.H();
tUaEqn.clear();
surfaceScalarField phiHbyAa
("phiHbyAa", fvc::flux(HbyAa));
adjustPhi(phiHbyAa, Ua, pa);

// Non-orthogonal pressure corrector loop
while (simple.correctNonOrthogonal())
{
fvScalarMatrix paEqn
(
fvm::laplacian(rAUa, pa) == fvc::div(phiHbyAa)
);
paEqn.setReference(paRefCell, paRefValue);
paEqn.solve();
if (simple.finalNonOrthogonalIter())
{
phia = phiHbyAa - paEqn.flux();
```

```
}
}

#include "adjointContinuityErrs.H"

// Explicitly relax
// pressure for adjoint momentum corrector
pa.relax();
// Adjoint momentum corrector
Ua = HbyAa - rAUa*fvc::grad(pa);
Ua.correctBoundaryConditions();
fvOptions.correct(Ua);
}
laminarTransport.correct();
turbulence->correct();

runTime.write();

Info<< "ExecutionTime = "
<< runTime.elapsedCpuTime()
<< " s\n\n" << endl;
}
Info<< "End\n" << endl;
return 0;
}
```