The **RocfluMP** Book

A. Haselbacher
Assistant Professor
Department of Mechanical and Aerospace Engineering
University of Florida
222 MAE-B, P.O. Box 116300
Gainesville, FL 32608-6300

Contact information:
Andreas Haselbacher
Assistant Professor
Department of Mechanical and Aerospace Engineering
University of Florida
222 MAE-B, P.O. Box 116300
Gainesville, FL 32608-6300
Phone: (352) 392-9459
Fax: (352) 392-1071
Email: haselbac@ufl.edu

Version 1.13.0 of 04/20/08

This manual documents:
rfluclone version 0.11.1 of 04/06/08
rfluconv version 6.6.1 of 04/06/08
rfluextr version 1.5.1 of 04/06/08
rfluinit version 5.6.1 of 04/06/08
rflumap version 4.34.1 of 04/06/08
rflump version 15.1.0 of 04/06/08
rflupart version 3.33.1 of 04/06/08
rflupick version 7.34.1 of 04/06/08
rflupost version 11.11.1 of 04/06/08

Contents

Notation **11**

I Preliminaries **13**

1 Introduction **14**
1.1 Objective . . . 14
1.2 Overview of RocfluMP . . . 14
1.3 Contributors . . . 17
1.4 Overview of the RocfluMP Book . . . 18

2 Nomenclature, Conventions, and Restrictions **21**
2.1 Nomenclature . . . 21
2.2 Conventions . . . 21
2.3 Restrictions . . . 22

3 Installation and Compilation **23**
3.1 Installation . . . 23
3.1.1 Installation from CVS Repository . . . 23
3.1.2 Installation from `.tar.gz` File . . . 24
3.2 Compilation . . . 24
3.2.1 Overview of Compilation Process . . . 24
3.2.2 Description of Compilation Options . . . 25

II User Manual **27**

4 Capability Descriptions **28**
4.1 One-Dimensional Computations . . . 28
4.2 Two-Dimensional Computations . . . 28
4.3 Axisymmetric Computations . . . 28
4.4 Periodic and Symmetry Boundary Conditions . . . 29
4.5 Mass, Pressure, Skin-Friction, and Heat-Transfer Coefficient Computation . . . 29

4.6 Force and Moment Computation . . . 29
4.7 Thrust and Specific-Impulse Computation . . . 30
4.8 Visualization of Discontinuities . . . 30
4.9 Visualization and Identification of Vortical Structures . . . 31

5 File Content and Format Specifications 33
5.1 Filename Conventions . . . 33
5.2 Input File . . . 33
5.2.1 ACCELERATION Section . . . 34
5.2.2 FLOWMODEL Section . . . 34
5.2.3 FORCES Section . . . 35
5.2.4 FORMATS Section . . . 35
5.2.5 GRIDMOTION Section . . . 36
5.2.6 INITFLOW Section . . . 37
5.2.7 MIXTURE Section . . . 39
5.2.8 MVFRAME Section . . . 40
5.2.9 NUMERICS Section . . . 41
5.2.10 POST Section . . . 45
5.2.11 PREP Section . . . 49
5.2.12 PROBE Section . . . 50
5.2.13 REFERENCE Section . . . 51
5.2.14 TIMESTEP Section . . . 51
5.2.15 TRANSFORM Section . . . 53
5.2.16 VISCMODEL Section . . . 54
5.3 Patch-Mapping Files . . . 54
5.3.1 VGRIDns Patch-Mapping File (.vgi File) . . . 55
5.3.2 MESH3D Patch-Mapping File (.mgi File) . . . 56
5.3.3 TETMESH Patch-Mapping File (.tmi File) . . . 57
5.3.4 Cobalt Patch-Mapping File (.cgi File) . . . 57
5.3.5 StarCD Patch-Mapping File (.sgi File) . . . 57
5.4 Boundary-Condition File . . . 57
5.4.1 Physical Boundary Conditions . . . 58
5.4.1.1 Farfield Boundary: BC_FARF Section . . . 58
5.4.1.2 Inflow Boundary: BC_INFLOW/BC_INFLOW_TOTANG Section . . . 59
5.4.1.3 Inflow Boundary: BC_INFLOW_VELTEMP Section . . . 62
5.4.1.4 Injection Boundary: BC_INJECT Section . . . 63
5.4.1.5 No-Slip Boundary: BC_NOSLIP_HFLUX Section . . . 64
5.4.1.6 No-Slip Boundary: BC_NOSLIP_TEMP Section . . . 66
5.4.1.7 Outflow Boundary: BC_OUTFLOW Section . . . 67
5.4.1.8 Periodic Boundary: BC_PERIODIC Section . . . 68
5.4.1.9 Slip Boundary: BC_SLIPW Section . . . 70
5.4.1.10 Symmetry Boundary: BC_SYMMETRY Section . . . 71

5.4.1.11 Virtual Boundary: `BC_VIRTUAL` Section . . . 71
5.4.2 Time-Dependent Boundary Conditions . . . 72
5.4.2.1 `TBC_PIECEWISE` Section . . . 72
5.4.2.2 `TBC_SINUSOIDAL` Section . . . 73
5.4.2.3 `TBC_STOCHASTIC` Section . . . 74
5.4.2.4 `TBC_WHITENOISE` Section . . . 74
5.4.3 Grid-Motion Boundary Conditions . . . 74
5.5 Region-Mapping File . . . 75
5.6 Restart-Information File . . . 76
5.7 Convergence File . . . 77
5.8 Probe File . . . 78
5.9 Moving-Reference-Frame File . . . 78
5.10 Mass-Conservation Check File . . . 79
5.11 Statistics File . . . 79
5.12 GENx Control File . . . 79

6 Problem Setup **81**

7 Execution **85**
7.1 rfluclone . . . 85
7.1.1 Invocation . . . 85
7.1.2 Input Files . . . 85
7.1.3 Output Files . . . 86
7.2 rfluconv . . . 86
7.2.1 Invocation . . . 86
7.2.2 Input Files . . . 87
7.2.3 Output Files . . . 87
7.2.4 Interactive Input . . . 87
7.3 rfluextr . . . 87
7.3.1 Invocation . . . 87
7.3.2 Input Files . . . 88
7.3.3 Output Files . . . 88
7.3.4 Interactive Input . . . 88
7.4 rfluinit . . . 88
7.4.1 Invocation . . . 88
7.4.2 Input Files . . . 89
7.4.3 Output Files . . . 89
7.5 rflumap . . . 89
7.5.1 Invocation . . . 89
7.5.2 Input Files . . . 90
7.5.3 Output Files . . . 90
7.6 rflump . . . 90

7.6.1 Invocation . . . 90
7.6.2 Input Files . . . 91
7.6.3 Output Files . . . 91
7.6.4 Profiling and Performance Analysis Guidelines . . . 92
7.7 rflupart . . . 93
7.7.1 Invocation . . . 93
7.7.2 Input Files . . . 93
7.7.3 Output Files . . . 94
7.7.4 Batch-Job Submission Guidelines . . . 94
7.8 rflupick . . . 94
7.8.1 Invocation . . . 96
7.8.2 Input Files . . . 96
7.8.3 Output Files . . . 96
7.8.4 Interactive Input . . . 97
7.9 rflupost . . . 98
7.9.1 Invocation . . . 98
7.9.2 Input Files . . . 98
7.9.3 Output Files . . . 99

8 Example Cases **101**
8.1 Shocktube . . . 101
8.2 GAMM Bump . . . 102
8.2.1 Serial Computation . . . 104
8.2.2 Parallel Computation . . . 105

9 Visualization **109**
9.1 Plotting Variables . . . 109
9.1.1 Visualization of Discontinuities . . . 109
9.1.2 Visualization and Identification of Vortical Structures . . . 110
9.1.3 Visualization of Eulerian Particle Fields . . . 111
9.2 TECPLOT Output . . . 111
9.2.1 File Naming Convention . . . 111
9.2.2 File Content . . . 112
9.2.2.1 Solution Variables . . . 112
9.2.2.2 Plotting Variables . . . 112
9.2.2.3 Patch-Coefficient Variables . . . 112
9.2.3 Zone Naming Conventions . . . 112
9.2.3.1 Volume Zones . . . 113
9.2.3.2 Boundary Patch Zones . . . 114
9.2.3.3 Border Face Zones . . . 114
9.2.3.4 Special Cell Zones . . . 116
9.2.3.5 Special Face Zones . . . 116

9.3 ENSIGHT Output . . . 117
9.3.1 File Naming Convention . . . 117
9.3.2 Part Naming Conventions . . . 117
9.3.2.1 Volume Parts . . . 118
9.3.2.2 Boundary Patch Parts . . . 120

10 Troubleshooting **121**
10.1 General Considerations . . . 121
10.2 Explanations of Warnings . . . 122
10.3 Explanations of Errors . . . 122
10.4 Other Problems . . . 123
10.5 Locating Troublespots . . . 123

III Developer and Reference Manual **125**

11 Governing Equations **126**
11.1 The Navier-Stokes Equations . . . 126
11.2 The Geometric Conservation Law . . . 127
11.3 Gas Models . . . 128
11.3.1 Calorically and Thermally Perfect Gas . . . 128
11.3.2 Mixture of Calorically and Thermally Perfect Gases . . . 128
11.3.3 Pseudo Gas . . . 128
11.4 Thermodynamic Properties . . . 128
11.4.1 Calorically and Thermally Perfect Gas . . . 128
11.4.2 Mixture of Calorically and Thermally Perfect Gases . . . 128
11.4.3 Pseudo Gas . . . 128
11.5 Transport Properties . . . 128
11.5.1 Viscosity . . . 128
11.5.1.1 Sutherland Model . . . 128
11.5.1.2 Antibes Model . . . 128
11.5.2 Conductivity . . . 129
11.6 Boundary Conditions . . . 129

12 Algorithms and Methods **131**
12.1 Geometry Definition . . . 131
12.1.1 Computation of Face Properties . . . 131
12.1.2 Computation of Volume Properties . . . 132
12.2 Spatial Discretization . . . 132
12.2.1 Stencil Construction . . . 132
12.2.2 Interpolation Operators . . . 133
12.2.3 Gradient Operators . . . 134
12.2.4 Inviscid Fluxes . . . 135

12.2.4.1 Limiter Functions . . . 135
12.2.4.2 Numerical Flux Functions . . . 135
12.2.4.3 Entropy Fixes . . . 135
12.2.5 Viscous Fluxes . . . 135
12.2.6 Optimal LES Discretization . . . 135
12.2.6.1 Computation of Integrals . . . 135
12.2.6.2 Computation of Stencil Weights . . . 135
12.2.7 Source Terms . . . 135
12.3 Boundary Conditions . . . 135
12.3.1 Simple Fluid-Solid Boundary Conditions . . . 135
12.3.1.1 Slip Wall Boundaries . . . 135
12.3.1.2 No-Slip Wall Boundaries . . . 135
12.3.1.3 Injection Wall Boundaries . . . 135
12.3.2 Simple Fluid-Fluid Boundary Conditions . . . 135
12.3.2.1 Inflow Boundaries . . . 135
12.3.2.2 Outflow Boundaries . . . 135
12.3.3 Non-Reflecting Boundary Conditions . . . 135
12.3.3.1 Overall Approach . . . 136
12.3.3.2 Inflow Boundary Conditions . . . 139
12.3.3.3 Outflow Boundary Conditions . . . 141
12.3.4 Virtual Boundary Conditions . . . 143
12.3.4.1 Periodic Boundaries . . . 143
12.3.4.2 Symmetry Boundaries . . . 143
12.4 Temporal Discretization . . . 143
12.4.1 Runge-Kutta Methods . . . 143
12.4.2 Computation of Time Step . . . 143
12.5 Grid Motion . . . 143
12.5.1 Grid Smoothing . . . 143
12.5.2 Discrete Geometric Conservation Law . . . 143
12.5.3 Implementation Details . . . 143
12.6 Mass, Pressure, Skin-Friction, and Heat-Transfer Coefficient Computation . . . 143
12.7 Force and Moment Computation . . . 144

13 Implementation Details 147
13.1 NSCBC Implementation . . . 147
13.1.1 Data-structure . . . 147
13.1.2 Initialization procedures . . . 148
13.1.3 Data read/write routines . . . 148
13.1.4 Gradient computation at boundary . . . 148
13.1.5 Characteristic wave amplitude computation . . . 148
13.1.6 Time integration . . . 149
13.1.7 Flux computation using boundary data . . . 149

13.2 Parallelization 149

14 Data Structures **151**
14.1 Philosophy and Abstraction 151
14.2 Region Data Structure 152
14.3 Grid Data Structure 154
14.3.1 Module `ModGrid.F90` 154
14.3.2 Module `RFLU_ModGrid.F90` 162
14.4 Boundary Data Structure 162
14.5 Mixture Data Structure 167
14.5.1 Data Type `t_mixt_input` 167
14.5.2 Data Type `t_mixt` 169

15 GENx Integration **173**
15.1 Definition of Attributes 173
15.1.1 Grid Attributes 173
15.1.1.1 Volume Panes 173
15.1.1.2 Surface Panes 173
15.1.2 Solution Attributes 173
15.1.3 Interface Attributes 173
15.1.3.1 Non-Interacting Panes 173
15.1.3.2 Non-Burning Panes 173
15.1.3.3 Burning Panes 173

16 File Content and Format Specifications **177**
16.1 Grid Files 177
16.1.1 RocfluMP Grid File 177
16.1.2 CENTAUR Grid File 179
16.1.3 VGRIDns Grid Files 179
16.1.3.1 `.vbc` File 179
16.1.3.2 `.cgosg` File 179
16.1.4 MESH3D Grid File 179
16.1.5 TETMESH Grid File 179
16.1.5.1 `.noboite` File 179
16.1.6 Cobalt Grid File 179
16.2 Flow-Solution File 180
16.3 Dimension File 181
16.4 Cell-Mapping File 183
16.5 Renumbering File 183
16.6 Communication-Lists File 183

References **185**

Index **187**

Notation

E	total energy per unit mass
E^P	energy source from particles
E^S	energy source from smoke
E^{SGS}	subgrid-scale energy flux
$\vec{f_e}$	vector of external volume forces
f^P	momentum source from particles
f^S	momentum source from smoke
$\vec{F_c}$	vector of convective fluxes
$\vec{F_v}$	vector of viscous fluxes
F^R	energy flux due to radiation
H	total (stagnation) enthalpy
k	thermal conductivity coefficient
m^P	mass source from particles
n_x, n_y, n_z	components of unit normal vector in $x-, y-, z-$direction
p	static pressure
$\dot{q}_h$	heat source
$\vec{Q}$	source term
dS	surface element
t	time
u, v, w	Cartesian velocity components
$\vec{v}$	velocity vector
V	contravariant velocity
$\vec{W}$	vector of conservative variables
x, y, z	Cartesian coordinates
μ	dynamic viscosity coefficient
ρ	density
τ	viscous stress
Ω	control volume
$\partial\Omega$	boundary of a control volume

Part I

Preliminaries

Chapter 1

Introduction

1.1 Objective

The objective of the RocfluMP book is four-fold:

1. To enable users to compile and install the RocfluMP source code on computer systems.
2. To enable users to run the RocfluMP code for the example cases.
3. To enable users to run the RocfluMP code for their own cases.
4. To enable users to develop RocfluMP to suit their own needs.

1.2 Overview of RocfluMP

RocfluMP solves the three-dimensional time-dependent compressible Navier-Stokes equations on moving and/or deforming unstructured grids. The grids may consist of arbitrary combinations of tetrahedra, hexahedra, prisms, and pyramids. The spatial discretization is carried out using the finite-volume method. The inviscid fluxes are approximated by upwind schemes to allow for capturing of shock waves and contact discontinuities. Steady flows can be computed using an explicit multi-stage method tuned for fast convergence. Unsteady flows are computed with the fourth-order accurate Runge-Kutta method.

To solve fluid-dynamics problems in which processes other than those described by the Navier-Stokes equations are important, RocfluMP is designed to interface with so-called *multi-physics modules*. The multi-physics modules model phenomena such as turbulence, particles, chemical reactions, and radiation and their interaction. At present, the multi-physics modules are under development and have not yet been integrated with RocfluMP. When considering fluid flow problems with several chemical species, RocfluMP may be regarded as solving transport equations for the mixture variables.

RocfluMP may be used to solve problems involving fluid-structure interactions. More specifically, RocfluMP is designed to operate as a solution module inside CSAR's coupled

rocket-simulation code GENx. To accommodate dynamically changing fluid domains arising from the deformation predicted by a structural simulation, RocfluMP allows for moving grids. The Geometric Conservation Law (GCL) is satisfied in a discrete sense to machine-precision to avoid the introduction of spurious sources of mass, momentum, and energy due to grid motion.

The relationship of RocfluMP and the other codes is depicted schematically in Fig. 1.1. A brief description of the multi-physics modules follows (they are described in detail in their respective manuals).

- Rocturb is the turbulence module which implements a variety of models for Reynolds-averaged Navier-Stokes (RANS) simulations and Large-Eddy Simulation (LES).
- Rocpart is the Lagrangian particle tracking module.
- Rocspecies is the module simulating the evolution of chemical species and Equilibrium Eulerian particles.
- Rocrad is the radiation module.

RocfluMP consists of several modules:

- rfluclone is the cloning module of RocfluMP. Starting from an appropriate single-region grid for serial computations, rfluclone "clones" grid files a specified number of times to generate a perfectly balanced data set for parallel performance studies.
- rfluconv is the conversion module of RocfluMP. It converts RocfluMP solution and grid files from ASCII to binary format and vice versa, and converts RocfluMP grid files into a format supported by YAMS and TETMESH. rfluconv requires interactive user input.
- rfluextr is a specialized data-extraction module of RocfluMP. It allows the extraction of data from solutions produced RocfluMP along user-specified straight lines and writes the data to ASCII files for visualization.
- rfluinit is the initialization module of RocfluMP. It generates RocfluMP solution files for each region based on the information contained in the user input file.
- rflumap is the processor-mapping module of RocfluMP. It generates the mapping file which is required for parallel computations. It also generates the Rocin control files for computations with GENx. rflumap requires interactive user input.
- rflump is the actual solution module of RocfluMP.
- rflupick is used in conjunction with rflupost to visualize only specific cells in the grid, such as cells near boundaries or cells sharing faces or vertices. For parallel computations, rflupick can also be used to instruct rflupost to convert only specific regions for visualization. This allows the visualization of nominally large cases on small machines. rflupick requires interactive user input.

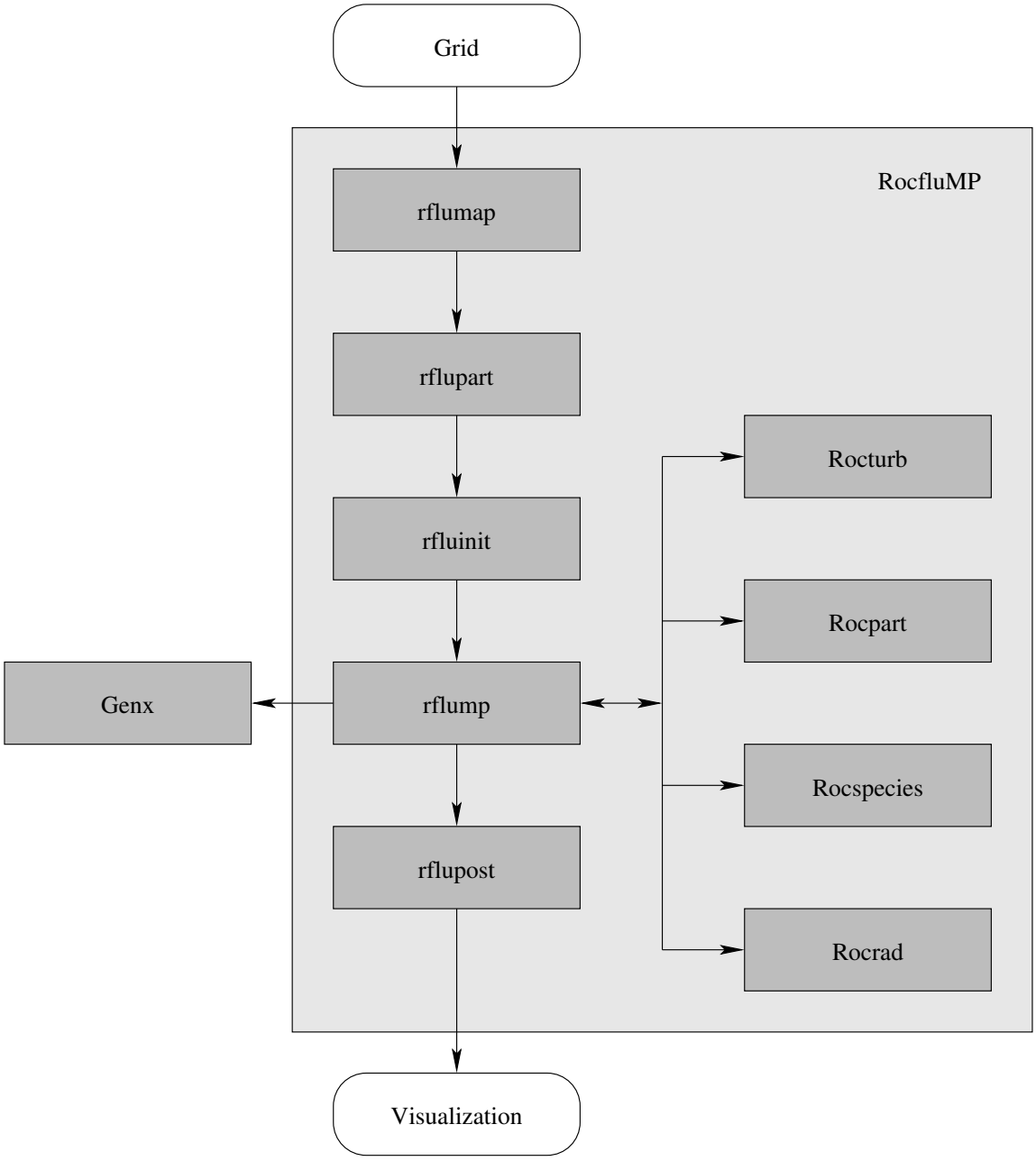

Figure 1.1: Overview of RocfluMP and related codes.

- rflupost is the postprocessing module of RocfluMP. It converts grid and solution files from the RocfluMP format into the formats recognized by visualization packages. At present, the following formats are supported:
  - TECPLOT format (Tecplot, Inc., Bellevue, WA)
  - ENSIGHT format (Computational Engineering International, Apex, NC)
- rflupart is the partitioning module of RocfluMP. It converts grid files from outside formats into binary or ASCII files in RocfluMP format and partitions the grids into regions. At present, the following formats are supported:
  - HYBRID format (CENTAUR grid generator, CentaurSoft, Austin, TX)
  - VGRIDns format (VGRIDns grid generator, Shahyar Pirzadeh, NASA Langley)
  - MESH3D format (MESH3D grid generator, Tim Baker, Princeton University)
  - TETMESH format (TETMESH grid generator, SIMULOG, France)
  - Cobalt format (Cobalt flow solver, Cobalt Solutions LLC, Springfield, OH)
  - GAMBIT format (GAMBIT grid generator, Fluent, Lebanon, NH)
  - StarCD format (Star-CCM++ grid generator, CD-Adapco, London, UK)

1.3 Contributors

The following people, listed in alphabetical order, have contributed to the development and testing of RocfluMP:

Mark Brandyberry: Testing, application of RocfluMP to fluid-structure interaction problems using GENx

Michael Campbell (Research Programmer, CSAR): Performance analysis and tuning, command-line interface, assistance with porting of RocfluMP to new computing platforms

Heath Dewey (Graduate Student, Department of Aeronautical Engineering, UIUC): Implementation of Newton-Krylov solver

Jim Ferry: Time-dependent boundary conditions, initial implementation of Equilibrium Eulerian method in Rocspecies

Robert Fiedler (Technical Program Manager, CSAR): Testing, application of RocfluMP to fluid-structure interaction problems using GENx

Andreas Haselbacher (Assistant Professor, University of Florida, formerly: Principal Research Scientist, CSAR): Lead developer of RocfluMP

Xiangmin Jiao (Assistant Professor, State University of New York at Stony Brook, formerly: Research Scientist, CSAR): Assistance with integration of RocfluMP into GENx

Randy Lagumbay (Research Scientist, Alden, formerly: Graduate Student, University of Colorado at Boulder): Formulation and implementation of (gas-liquid-vapor) multiphase flow capability

Fady Najjar (Research Scientist, Lawrence Livermore National Laboratory, formerly: Senior Research Scientist, CSAR): Lead developer of Rocpart module, integration of Rocpart with RocfluMP, testing, verification and validation

Adam Moody (Lawrence Livermore National Laboratory): Assistance with tuning of RocfluMP

Manoj Parmar (Graduate Student, Department of Mechanical and Aerospace Engineering, University of Florida, formerly: Graduate Student, Department of Aeronautical Engineering, UIUC): Validation for shock diffraction over cylinders and spheres, implementation of Schlieren and Interferogram image capability in rflupost, implementation of limiter functions, non-reflecting boundary conditions, implementation of moving-reference frame capability

Charles Shereda (Lawrence Livermore National Laboratory): Assistance with tuning of RocfluMP

Oleg Vasilyev (Professor, University of Colorado at Boulder): Formulation of (gas-liquid-vapor) multiphase flow capability

1.4 Overview of the RocfluMP Book

The RocfluMP book is divided into three parts.

Part I, which you are currently reading, is designed to provide preliminary information so that users get an overview of RocfluMP, become familiar with the nomenclature, conventions, and restrictions of RocfluMP, and can install and compile RocfluMP on their computer.

Part II contains information for users of RocfluMP. This entails a detailed description of the capabilities of RocfluMP, the content and specification of the various files which either must be provided or may have to be edited by the user, explanations of how to set up a problem to be solved with RocfluMP, how to execute RocfluMP, how to visualize results produced by RocfluMP, and how to resolve problems which may arise when running RocfluMP.

Part III provides detailed information for developers of RocfluMP and serves as a reference manual for users. It includes the precise form of the governing equations, the algorithms and methods used to solve them, their parallel implementation, a detailed description of the data structures, and the integration of RocfluMP with GENx, and the content and specification of the various files which may have to changed by developers.

To convey information in a clear and concise manner, the following graphical aids are employed:

Important information, such as restrictions, dependencies, or other subtleties which are not immediately apparent are emphasized through colored boxes such as this one.

Because RocfluMP continues to evolve, the information contained in this book is also changing. To draw attention to changes in the book, so-called change bars are positioned in the margins. More specifically, change bars indicate that either a new capability was added or that the corresponding description was changed.

Chapter 2

Nomenclature, Conventions, and Restrictions

2.1 Nomenclature

The following conventions are used in this document:

1. A *grid level*, or simply *level*, represents the entire solution domain of RocfluMP. A grid level can consist of one or more *solution regions*.

2. A *solution region*, or simply *region*, is obtained from a grid level by partitioning it for parallel processing. For sequential processing, the region encompasses the entire grid level.

3. A *grid* is defined to be an arbitrary collection of *grid cells*, or simply *cells*.

4. A *grid cell* is defined to be a convex polyhedron. Each cell is composed of *faces*.

5. A *face* is defined to be a polygon. Each face is composed of *edges*.

6. A *patch* is a collection of faces on the boundary of a region on which the same boundary condition is applied.

7. An *edge* is defined to be a straight line linking two *vertices*.

8. A *vertex* is defined to be a point in space. A vertex belongs to at least one cell. A vertex is not necessarily equivalent to a node.

2.2 Conventions

1. SI (Système International) units are used in RocfluMP and all documents relating to RocfluMP.

2. All coordinate systems are right-handed.
3. Normal vectors point out of the solution domain.

2.3 Restrictions

1. Cells must be hexahedra, prisms, pyramids, or tetrahedra.
2. Cells must be conforming, i.e., hanging edges or vertices are not allowed.
3. Faces must be triangles or quadrilaterals.

Chapter 3

Installation and Compilation

3.1 Installation

The following assumes that RocfluMP is to be installed either from the CSAR CVS repository or from a `gzip`ped `tar` file.

3.1.1 Installation from CVS Repository

To be able to access the CSAR CVS repository, set the `CVSROOT` environment variable to (taking the `bash` shell as an example)

```
export CVSROOT=:pserver:user@machine.uiuc.edu:/cvsroot
```

and either open a new terminal or type

```
[user@machine ~]$ source .bashrc
```

Then type

```
[user@machine ~]$ cvs login
```

and hit the `Enter` key at the prompt.

Now move into the directory where you want to install RocfluMP. In the following, this is assumed to be `directory`. Then type

```
[user@machine ~/directory]$ cvs co genx/Codes/RocfluidMP
```

which will check out the source code for RocfluMP from the repository.

Assuming the checkout command has completed successfully, you are now ready to compile the code for serial computations, and you can proceed to Sec. 3.2.

3.1.2 Installation from `.tar.gz` File

Move into the directory where you want to install RocfluMP. In the following, this is assumed to be `directory`. Move or copy the `gzipped` `tar` file, assumed to be `<file>.tar.gz` in the following, into `directory`. Then type

```
[user@machine ~/directory]$ gzip -d <file>.tar.gz
[user@machine ~/directory]$ tar -xvf <file>.tar
```

which will unpack the source code.

Assuming these commands to have completed successfully, you are now ready to compile the code for serial computations, and you can proceed to Sec. 3.2.

3.2 Compilation

3.2.1 Overview of Compilation Process

The compilation process for RocfluMP is automatic in the sense that the `Makefiles` determine the machine type and set the suitable compilation options. If you intend to run on Apple, IBM, Linux, SGI, or Sun machines, you do not need to modify any `Makefiles`. If you intend to run on other machines, you will need to create your own `Makefile`. You can pattern it after the existing machine-dependent `Makefiles`.

RocfluMP is compiled with MPI by default, which means that you must have installed MPI on your machine before attempting to compile RocfluMP.

The compilation process consists of two stages. The first stage is the actual computation, as described below. The output of the compilation process are several executables:

rfluclone The cloning module of RocfluMP.

rfluconv The conversion module of RocfluMP.

rfluextr The extraction module of RocfluMP.

rfluinit The initialization module of RocfluMP.

rflumap The region mapping module of RocfluMP.

rflupick The region and cell picking module of RocfluMP.

rflupost The postprocessing module of RocfluMP.

rflupart The partitioning module of RocfluMP.

rflump The flow solution module of RocfluMP.

The second stage consists of copying these executables into your `$(HOME)/bin` directory by typing

```
[user@machine ~/directory]$ gmake RFLU=1 install
```

3.2.2 Description of Compilation Options

To compile RocfluMP, type the following at the prompt:

```
[user@machine ~/directory]$ gmake RFLU=1 <options>
```

where the currently supported `<options>` are any of the following:

`CHECK_DATASTRUCT=1` Activates checking of data structures. This option will print out the content of the important data structures used by RocfluMP. Note that activating this option will lead to substantial screen output, so it should only be activated for small cases.

`DEBUG=1` Activates debugging compiler options. If this option is not specified, optimizing compiler options are chosen by default.

`PLAG=1` Activates compilation of Rocpart. This option must be specified if you wish to run computations with Lagrangian particles.

`ROCPROF=<Path to` **Rocprof** `library>` Activates profiling of selected routines in RocfluMP. Note that activating this option entails an overhead, so it should not be used for production runs.

`SPEC=1` Activates compilation of Rocspecies. This option must be specified if you wish to run computations with chemical species and/or Equilibrium Eulerian particles.

Part II

User Manual

Chapter 4

Capability Descriptions

This chapter describes the capabilities of RocfluMP and points the user to the relevant sections in Chapter 5.

4.1 One-Dimensional Computations

RocfluMP can compute one-dimensional flows. Grids for one-dimensional computations may contain only a single stack of hexahedral cells in the x-coordinate direction. In other words, each cell must have two faces each on patches with y = constant and z = constant. A virtual boundary condition as specified by the BC_VIRTUAL Section must be applied to these patches. One-dimensional computations are activated by the keyword DIMENS in the NUMERICS Section.

4.2 Two-Dimensional Computations

RocfluMP can compute two-dimensional flows. Grids for two-dimensional computations may contain only a single plane of prismatic and/or hexahedral cells in the z-coordinate direction. In other words, each cell must have two faces on patches with z = constant. A virtual boundary condition as specified by the BC_VIRTUAL Section must be applied to these patches. Two-dimensional computations are activated by the keyword DIMENS in the NUMERICS Section. A sample grid which could be used for two-dimensional computations with RocfluMP is shown in Fig. 4.1.

4.3 Axisymmetric Computations

RocfluMP can compute axisymmetric flows. Axisymmetric computations are carried out as part of RocfluMP's capability of running two-dimensional computations described in Section 4.2 and are activated by the keyword AXIFLAG in the NUMERICS Section.

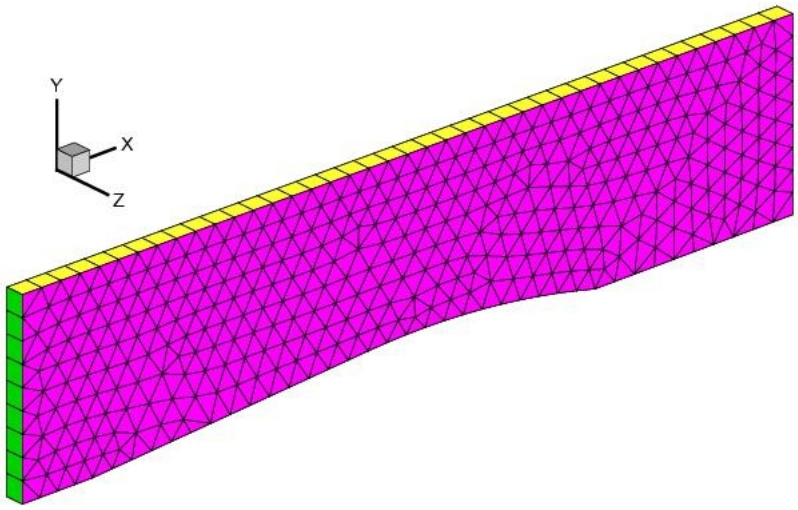

Figure 4.1: Sample grid which can be used for two-dimensional computations with RocfluMP.

4.4 Periodic and Symmetry Boundary Conditions

RocfluMP can impose periodic and symmetry boundary conditions. The periodic boundary condition is general and allows both linear and rotational periodicity to be imposed, thus allowing channel flows or flows in turbomachinery geometries, see, e.g., Fig. 4.2 , to be computed. The imposition of periodic and symmetry boundary conditions is described in the `BC_PERIODIC` and `BC_SYMMETRY` sections.

4.5 Mass, Pressure, Skin-Friction, and Heat-Transfer Coefficient Computation

RocfluMP can compute mass, pressure, skin-friction, and heat-transfer coefficients for faces on patches. Definitions of these coefficients can be found in Section 12.6.

4.6 Force and Moment Computation

RocfluMP can compute forces and moments exerted by the fluid on the patches. Definitions of the force and moment coefficients and their computation can be found in Section 12.7. The computation of forces and moments is governed by the `FORCES` and the `REFERENCE` Sections in the input file.

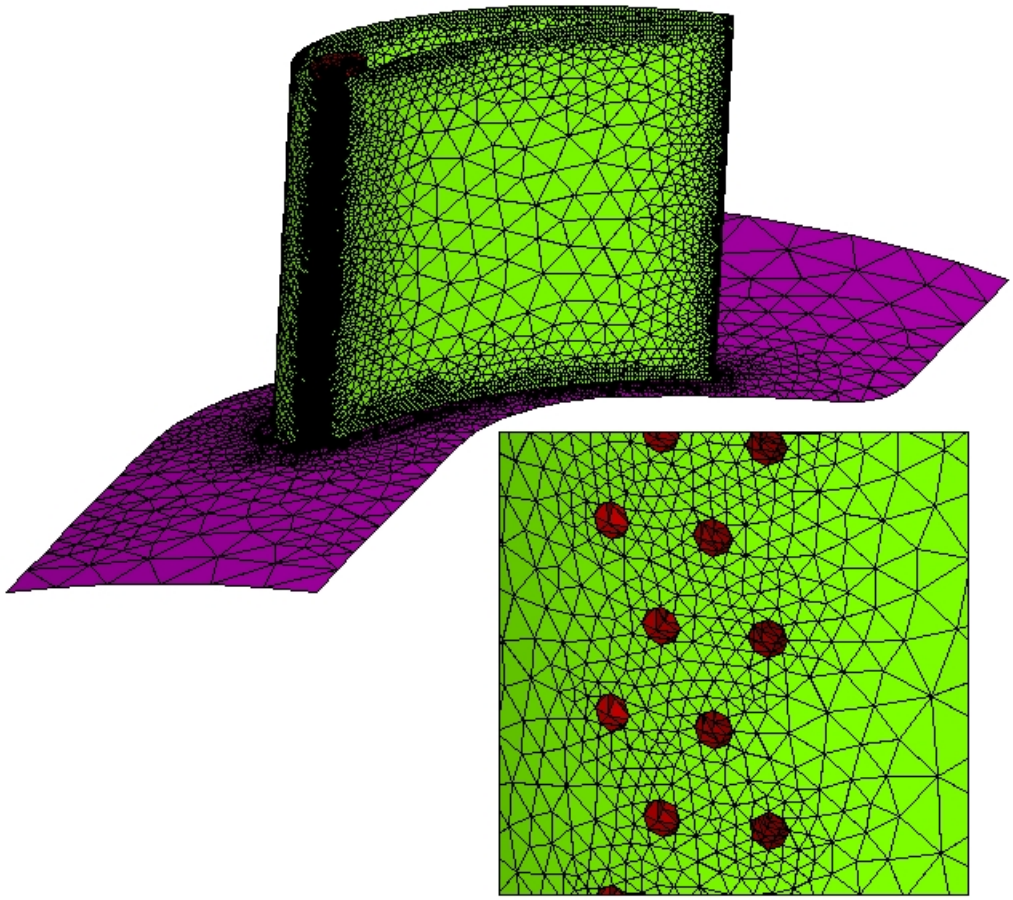

Figure 4.2: Generic turbine guide vane with leading-edge cooling. Such geometries can be computed with RocfluMP using periodic boundary conditions. Inset shows detail view of cooling holes at leading edge.

4.7 Thrust and Specific-Impulse Computation

RocfluMP can compute the thrust and specific impulse generated by the fluid on the patches. The computation of thrust and specific impulse is governed by the `FORCES` and the `REFERENCE` Sections in the input file.

4.8 Visualization of Discontinuities

RocfluMP can compute Schlieren pictures, Shadowgraphs, and Interferograms to assist in the visualization of compressible flows with discontinuities. The computation of Schlieren pictures, Shadowgraphs, and Interferograms is governed by the keyword `DISCFLAG` in the `POST` Section of the input file.

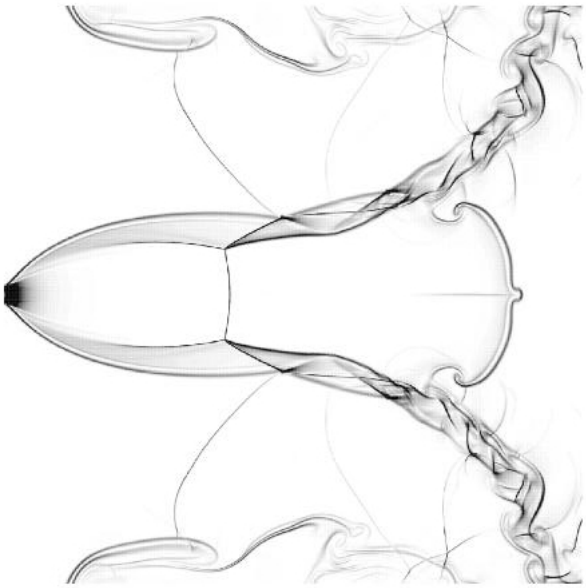

Figure 4.3: Sample Schlieren picture of flow from open-ended shocktube 20 ms after rupture of diaphragm. Dark regions indictate high gradients of the density and can thus be used to locate shock waves and contact discontinuities.

4.9 Visualization and Identification of Vortical Structures

RocfluMP can compute variables which help visualize and identify vortical structures. The computation of these variables is described in Subsection 9.1.2 and is governed by keywords in the POST Section of the input file.

Chapter 5

File Content and Format Specifications

This chapter describes the content and format of RocfluMP files which require or may require editing by a user to prepare a computation or interpret results obtained from a computation. The visualization files produced by rflupost are described in Chapter 9.

The content and format of RocfluMP files which cannot or must not be modified by a user, but may have to be modified by a developer, are described in Chapter 16.

5.1 Filename Conventions

The majority of files share a user-specified string, the so-called 'case name,' represented by `casename` below. Many of the files whose format is described below consist of a region index and either an iteration or a time index. The region index indicates the global region number with which a given file is associated. A region index of zero indicates that the given file is associated with a serial or unpartitioned data set. A region index of one or greater indicates that the given file is part of parallel data set.

5.2 Input File

The input file is called `<casename>.inp`. The input file is divided into sections. Each section contains several lines, each of which consists of a keyword and a value, as shown below.

```
# SECTION_NAME
KEYWORD_1 VALUE_1
KEYWORD_2 VALUE_2
KEYWORD_3 VALUE_3
#
```

Comments may be inserted after the specification of the values; they are ignored by the routines reading the input file.

The following sections describe each section in the input file and the associated keywords in detail. For simplicity, the sections are listed in alphabetical order, but they may appear in any order in the input file.

5.2.1 ACCELERATION Section

The ACCELERATION section contains the following keywords:

TYPE Specifies whether acceleration is specified This keyword can take the following values.

- 0 Acceleration is not specified.
- 1 Acceleration is specified. The acceleration vector is given by the values assigned to the keywords ACCELX, ACCELY, and ACCELZ.

ACCELX Acceleration in x-coordinate direction [m/s^2].

ACCELY Acceleration in y-coordinate direction [m/s^2].

ACCELZ Acceleration in z-coordinate direction [m/s^2].

5.2.2 FLOWMODEL Section

The FLOWMODEL section contains the following keywords:

MODEL Specifies which equations are to be solved. It can take the following values:

- 0 rflump solves the Euler equations.
- 1 rflump solves the Navier-Stokes equations.

MOVEGRID Specifies whether grid motion is active or not. It can take the following values:

- 0 Grid motion is inactive.
- 1 Grid motion is active.

It is important to note that this flag influences the names of the grid file and dimension file. If grid motion is active and the flow is unsteady, these files aquire a time stamp, see Sec. 16.1.1.

5.2.3 FORCES Section

The FORCES section governs the computation of forces on moments on the patches of the geometry. The computation of the forces and moments and the necessary conventions are described in Section 12.7. The FORCES section contains the following keywords:

FLAG Specifies whether forces and moments are to be computed. It can take the following values:

- 0 Do not compute forces and moments.
- 1 Compute forces and moments.

Activating the computation of forces and moments leads to the forces and moments being printed on the screen and written to files.

PATCHFLAG Specifies whether pressure, skin-friction, and heat-transfer coefficients are to be written to output files. It can take the following values:

- 0 Do not write coefficients to output files.
- 1 Write coefficients to output files.

REFAREA Specifies value of reference area [m^2].

REFLENGTH Specifies value of reference length [m].

REFXCOORD Specifies value of reference x-coordinate [m].

REFYCOORD Specifies value of reference y-coordinate [m].

REFZCOORD Specifies value of reference z-coordinate [m].

It is important to note that the computation of force and moment coefficients uses values such as DENS, PRESS, and ABSVEL from the REFERENCE section, but it does *not* use the LENGTH parameter from that section! Instead the reference length is given by the value assigned to the keyword REFLENGTH.

5.2.4 FORMATS Section

The FORMATS section contains the following keywords:

GRID Specifies the format of the RocfluMP grid file. It can take the following values:

- 0 Grid file is in ASCII format.
- 1 Grid file is in binary format.

GRIDSRC Specifies the format of the grid file read by rflupart. It can take the following values:

- 0 Grid file is in CENTAUR ASCII format.
- 1 Grid file is in VGRIDns format.
- 2 Grid file is in MESH3D format.
- 3 Grid file is in TETMESH format.
- 4 Grid file is in Cobalt format.
- 5 Grid file is in GAMBIT format. Only GAMBIT files in ASCII neutral file format are supported.
- 6 Grid file is in StarCD format. Only StarCD files in ASCII format are supported.
- 10 Grid file is in CENTAUR binary format.

SOLUTION Specifies the format of the RocfluMP flow file. It can take the following values:

- 0 Flow file is in ASCII format.
- 1 Flow file is in binary format.

5.2.5 GRIDMOTION Section

The GRIDMOTION section contains the following keywords:

NITER Specifies the number of smoothing iterations. Only applicable if TYPE=1 or TYPE=2.

SFACT Specifies the smoothing coefficient. The values for the number of smoothing iterations and the smoothing coefficient should be chosen together. The recommended values are, for moving the grid by smoothing the boundary displacements, NITER=4 and SFACT=0.25, and for moving the grid by smoothing the coordinates, NITER=10 and SFACT=0.1. Only applicable if TYPE=1 or TYPE=2.

TYPE Specifies the type of grid motion. It can take the following values:

- 1 Move the grid by smoothing the boundary displacements. This option is only available for serial computations.
- 2 Move the grid by smoothing the coordinates. This option is only available for serial computations.
- 3 Move the grid using the MESQUITE package. This option is only available when running RocfluMP in GENx.

Moving the grid based on smoothing boundary displacements has the advantage that the vertices are not moved for vanishing displacements. This is not true if the grid is moved by smoothing coordinates. In particular, non-uniform or distorted grids can be strongly affected by smoothing the coordinates. Therefore, moving the grid by smoothing the boundary displacements is the recommended option.

5.2.6 INITFLOW Section

The INITFLOW section is relevant only to rfluinit. It contains the following keywords:

DENS The density of the initial solution, [kg/m^3]. Only relevant if FLAG=1.

FLAG Specifies whether initial solution is to be generated. It can take the following values:

1 Generate initial solution using the values assigned to the keywords DENS, VELX, VELY, VELZ, and PRESS.

2 Generate initial solution using the data contained in a file.

3 Generate initial solution using a hardcode. The hardcode depends on the casename. This option can only be used if appropriate code to initialize the solution was added to the file RFLU_InitFlowHardCode.F90 in rfluinit for the given casename. If the appropriate code does not exist, rfluinit will return an error. At present, hardcoded initial solutions for the following casenames are provided:

onera_c0 ONERA C0 case [6].

pipeacoust Pipe acoustics cases.

ringleb Ringleb flow [9].

st_sod1 Sod shock tube [13], case 1.

st_sod2 Sod shock tube [13], case 2.

ssvort<t><mxn>l<p> Supersonic free vortex.

<t>=[h|p] h and p denote hexahedral and prismatic grids, respectively.

<mxn> denotes the grid resolution for both hexahedral and prismatic grids. It can take the following values: 20x5, 40x10, 80x20, 160x40, 320x80.

<p> denotes the number of layers in the z-coordinate direction. It can take values 1 or 3.

IVAL1-IVAL6 Integer variable used to set hard-coded initial conditions. Its meaning is dependent on the casename as specified below. Only relevant if FLAG=3.

PRESS The static pressure of the initial solution, [Pa]. Only relevant if FLAG=1.

RVAL1-RVAL6 Real variable used to set hard-coded initial conditions. Its meaning is dependent on the casename as specified below. Only relevant if FLAG=3.

VELX The x-component of the velocity vector of the initial solution, [m/s]. Only relevant if FLAG=1.

VELY The y-component of the velocity vector of the initial solution, [m/s]. Only relevant if FLAG=1.

VELZ The z-component of the velocity vector of the initial solution, [m/s]. Only relevant if FLAG=1.

The meaning of the variables `IVAL1-IVAL6` and `RVAL1-RVAL6` is dependent on the `casename` and specified in the following list:

`onera_c0`

`pipeacoust`

`IVAL1` Specifies wave number of circumferential mode, [1/rad].

`IVAL2` Specifies wave number of axial mode, [−].

`IVAL3` Specifies root of derivative of Bessel function of order given by `IVAL1`, [−].

`IVAL4` Specifies boundary conditions in axial directions:

`0` The pressure disturbance vanishes.

`1` The gradient of pressure disturbance vanishes.

`RVAL1` Specifies amplitude of pressure disturbance, [Pa].

It is important to note that initial solutions are defined after geometric transformations.

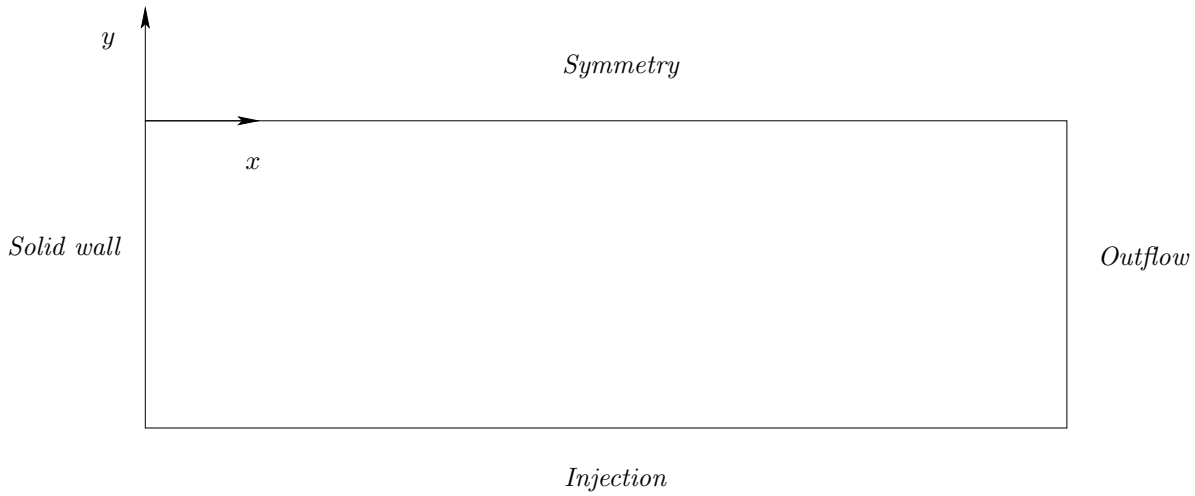

Figure 5.1: Configuration and boundary conditions for `onera_c0` case.

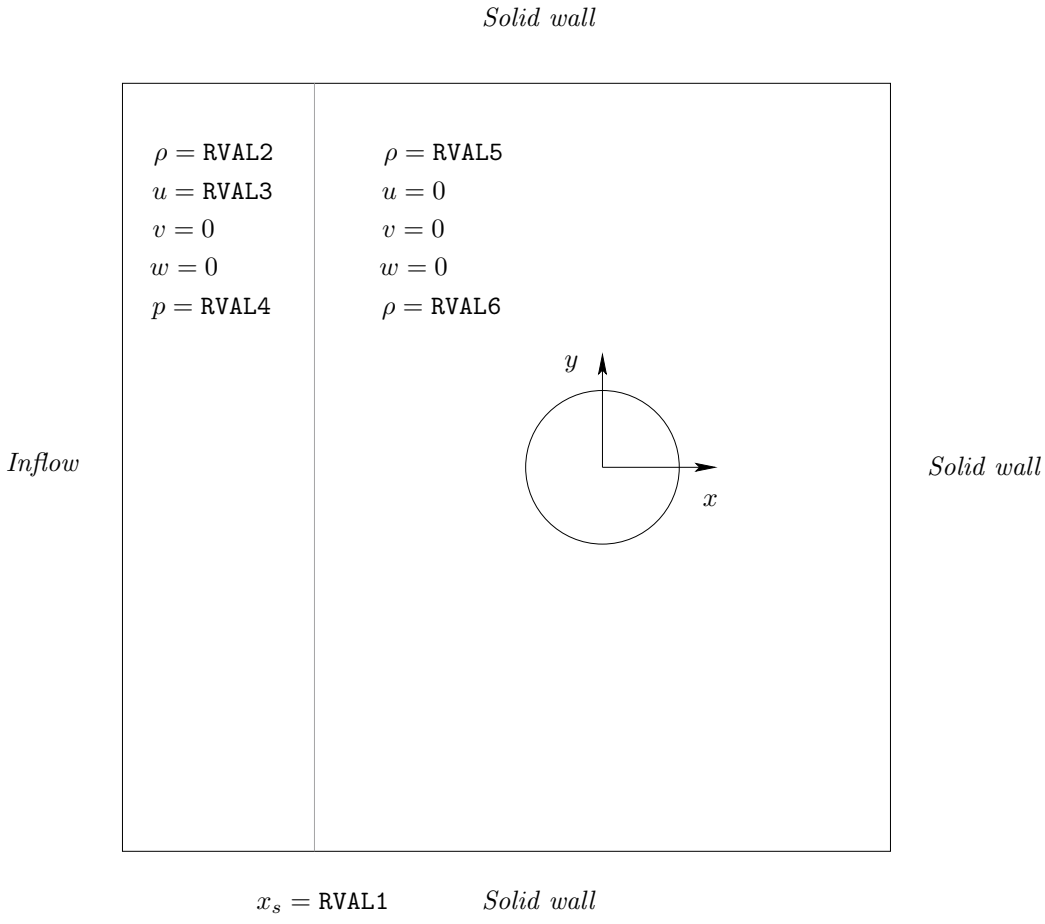

Figure 5.2: Configuration and boundary conditions for `cylds` case.

5.2.7 `MIXTURE` Section

The `MIXTURE` section contains the following keywords:

`FROZENFLAG` Specifies whether mixture conservation are to be updated. It can take the following values:

- `0` Do not update mixture conservation equations.
- `1` Update mixture conservation equations (default).

`GASMODEL` Specifies the gas model. The gas model determines how the molecular weight, the specific heat at constant pressure, and the transport coefficients are computed. It can take the following values:

- `1` The gas is considered to be thermally and calorically perfect, i.e., the molecular weight and specific heat at constant pressure are constant (default). The latter is given by the value of `CP` from the `REFERENCE` section. The former is computed from

the specific heat at constant pressure and the value of GAMMA from the REFERENCE section.

3 The gas is considered to be a mixture of thermally and calorically perfect gases. The molecular weight and specific heat at constant pressure are computed from mass-fraction weighted averages of the molecular weights and specific heats at constant pressure of the mixture constituents, see Section 11.3.2. This gas model is available only if species are solved, i.e., if the Rocspecies module is activated.

5 The gas is considered to be a pseudo gas. The molecular weight and specific heat at constant pressure are computed from appropriately defined averages of the molecular weights and specific heats at constant pressure of the mixture constituents, see Section 11.3.3. This gas model is available only if species are solved, i.e., if the Rocspecies module is activated.

5.2.8 MVFRAME Section

The MVFRAME section contains the following keywords:

ACCFLAG Specifies whether acceleration is computed or imposed. It can take the following values:

0 Acceleration is computed (default). The acceleration is computed through Newton's second law from the force exerted by the fluid on the body defined by patch with global index IPATCH and mass MASS.

1 Acceleration is imposed. The acceleration vector is given by the values assigned to ACCX, ACCY, and ACCZ during the time interval ACCTS-ACCTE.

ACCTS Starting value of time interval during which acceleration is applied [s]. Before the starting value, the acceleration is zero. Only relevant if ACCFLAG=1, when it must be set by the user; there is no default value.

ACCTE Ending value of time interval during which acceleration is applied [s]. After the starting value, the acceleration is zero. Only relevant if ACCFLAG=1, when it must be set by the user; there is no default value.

ACCX x-component of acceleration vector [m^2/s]. Only relevant if ACCFLAG=1, when it must be set by the user; there is no default value.

ACCY y-component of acceleration vector [m^2/s]. Only relevant if ACCFLAG=1, when it must be set by the user; there is no default value.

ACCZ z-component of acceleration vector [m^2/s]. Only relevant if ACCFLAG=1, when it must be set by the user; there is no default value.

FLAG Specifies whether the moving reference is active.

0 Acceleration of moving reference frame is not active (default).

1 Acceleration of moving reference frame is active.

If the moving reference-frame option is activated, the instantaneous values of the location, velocity, and acceleration of the body are written to the moving-reference-frame file. The format of the moving-reference frame file is described in Section 5.9.

IPATCH Global index of patch defining the body being accelerated. This patch index should define a patch with slip or no-slip boundary conditions.

LOCX x-component of initial position of body defined by IPATCH [m]. Only relevant if ACCFLAG=0, when it must be set by the user; there is no default value.

LOCY y-component of initial position of body defined by IPATCH [m]. Only relevant if ACCFLAG=0, when it must be set by the user; there is no default value.

LOCZ z-component of initial position of body defined by IPATCH [m]. Only relevant if ACCFLAG=0, when it must be set by the user; there is no default value.

MASS Mass of body defined by IPATCH [kg/m^3]. Only relevant if ACCFLAG=0.

VELX x-component of initial velocity of body defined by IPATCH [m]. Only relevant if ACCFLAG=0, when it must be set by the user; there is no default value.

VELY y-component of initial velocity of body defined by IPATCH [m]. Only relevant if ACCFLAG=0, when it must be set by the user; there is no default value.

VELZ z-component of initial velocity of body defined by IPATCH [m]. Only relevant if ACCFLAG=0, when it must be set by the user; there is no default value.

5.2.9 NUMERICS Section

The NUMERICS section contains the following keywords:

AXIFLAG Specifies whether to carry out axisymmetric computations. It can take the following values:

0 Do not carry out axisymmetric computation (default).

1 Carry out axisymmetric computation.

Note that AXIFLAG=1 is only permissible for two-dimensional computations, i.e., if DIMENS=2.

CFL Specifies the CFL number to be used during computations.

`CRECONSTC` Specifies whether constrained reconstruction is to be used to compute cell gradients. It can take the following values:

- `0` Do not use constrained reconstruction (default).
- `1` Use constrained reconstruction.

`CRECONSTCW` Weight for constrained cell-gradient reconstruction. Larger values weight constraints more heavily. Default value: 1.0; should not be changed without good reason.

`CRECONSTF` Specifies whether constrained reconstruction is to be used to compute face gradients. It can take the following values:

- `0` Do not use constrained reconstruction (default).
- `1` Use constrained reconstruction.

If you wish to compute viscous flows, you must set `CRECONSTF=1`. Otherwise, the no-slip condition is only enforced weakly and you may observe non-zero velocities on solid boundaries.

`CRECONSTFW` Weight for constrained face-gradient reconstruction. Larger values weight constraints more heavily. Default value: 1.0; should not be changed without good reason.

`DIMENS` Specifies the dimensionality of the computation. It can take the following values:

- `1` Run one-dimensional computation.
- `2` Run two-dimensional computation.
- `3` Run three-dimensional computation (default).

The dimensionality flag is provided so that truly one- and two-dimensional computations can be performed, i.e., a computation in which the grid contains only one cell in the y and/or z-directions. Three-dimensional computations must contain at least three cells in each coordinate direction to be able to compute gradients properly.

Note the following:

- If you wish to run truly one-dimensional computations, your grid may consist only of hexahedra, there must be two boundary patches each that coincide with y = constant and z = constant planes, and you should specify the boundary condition on those patches to be `BC_VIRTUAL`, see Sect. 5.4.1.11.
- If you wish to run truly two-dimensional computations, your grid may consist only of prisms and/or hexahedra, there must be two boundary patches that coincide with z = constant planes, and you should specify the boundary condition on those patches to be `BC_VIRTUAL`, see Sect. 5.4.1.11.

`DISCR` Specifies the discretization scheme for the inviscid fluxes. It can take the following values:

- `1` The flux-difference splitting scheme of Roe [10].
- `3` The HLLC approximate Riemann solver of Batten et al. [3]
- `4` The AUSM+ scheme of Liou [7].

The flux-difference splitting scheme of Roe works well in general. Because Roe's scheme is not positively conservative, it can lead to negative densities and pressures for flows with strong transients. For this reason, such flows should be computed with either the HLLC solver or the AUSM+ scheme. For shock waves with perfect or near-perfect alignment on uniform hexahedral grids, the Roe scheme and HLLC solver can suffer from transverse shock instability, so the AUSM+ scheme should be used instead. Viscous flows near solid walls with strong flow-to-grid alignment can occasionally lead to mild forms of decoupling with the AUSM+ scheme.

`DISSFACT` Factor multiplying the dissipation terms of the flux-difference splitting of Roe. The dissipation may need to be reduced to capture marginally resolved flow features such as separation or vortices. It is important to note however, that reducing the dissipation can compromise the stability of the computation. The default value of `DISSFACT` is `1`.

`ENTROPY` Specifies value of entropy correction coefficient. Applies only to the flux-difference splitting scheme of Roe.

`ORDER` The order of accuracy of the flux discretization. It can take the following values:

- `1` Compute fluxes with first-order accuracy.
- `2` Compute fluxes with second-order accuracy.

For runs with GENx, rflupart should be executed with `ORDER=2` even if the actual computation is carried out with `ORDER=1`. This is because MESQUITE requires virtual cells and vertices beyond those available in first-order accurate runs for proper smoothing across region boundaries.

`ORDERBF` The order of accuracy of the flux discretization for boundary faces. It can take the following values:

- `1` Compute boundary fluxes with first-order accuracy.
- `2` Compute boundary fluxes with second-order accuracy.

Note that the order of accuracy of the flux discretization for boundary faces can be overridden for a specific patch by specifying the order of accuracy through the keyword `ORDER` in appropriate section in the boundary-condition file.

`RECONST` Specifies the way in which the gradients computed by the reconstruction method are modified. It can take the following values:

- `0` The gradients are not modified.
- `1` Weighted essentially non-oscillatory (WENO) reconstruction in which gradients are weighted according to the inverse square of their magnitude.
- `2` Weighted essentially non-oscillatory (WENO) reconstruction in which gradient components are weighted according to the inverse square of their magnitude.
- `10` Gradients are modified using limiter function of Barth and Jespersen [2].
- `11` Gradients are modified using limiter function of Venkatakrishnan [15].

For flows without sharp gradients (such as shock waves, contact discontinuities, or material interfaces) or computations in which sharp gradients are resolved (such as the above in inviscid flows), the gradients need not be modified. For flows with sharp gradients or computations in which sharp gradients are not resolved, the gradients must be modified to avoid numerical instability.

`SDIMENSC` Specifies the dimensionality of the cell stencil. It can take the following values:

- `2` use two-dimensional cell stencil.
- `3` use three-dimensional cell stencil (default).

`SDIMENSF` Specifies the dimensionality of the face stencil. It can take the following values:

- `2` use two-dimensional face stencil.
- `3` use three-dimensional face stencil (default).

`SDIMENSBF` Specifies the dimensionality of the boundary face stencil. It can take the following values:

- `2` use two-dimensional boundary face stencil.
- `3` use three-dimensional boundary face stencil (default).

`TOLERICT` Specifies the value of the tolerance used in the in-cell test for locating particles and probes. The tolerance is typically important only if the grid contains cell types with quadrilateral faces, i.e., hexahedra, prisms, or pyramids, and the quadrilateral faces are not planar. Typical values should be in the range 10^{-8} to 10^{-6}. RocfluMP will check whether the value is appropriate and, if necessary, increase it.

5.2.10 POST Section

The POST section contains the following keywords:

COREFLAG Specifies whether variables for vortex identification are to be written to visualization files. It can take the following values:

- 0 Do not compute variables for vortex identification and do not write to visualization files.
- 1 Compute variables for vortex identification and write to visualization files.

See also Sec. 9.1.

At present, vortex-identification variables can only be written to TECPLOT visualization files.

DISCFLAG Specifies whether variables for Schlieren images, shadowgraphs, and interferograms are to be written to visualization files. It can take the following values:

- 0 Do not compute variables for Schlieren images, shadowgraphs, and interferograms and do not write to visualization files.
- 1 Compute variables for Schlieren images, shadowgraphs, and interferograms and write to visualization files.

See also Sec. 9.1.

At present, the variables for Schlieren images, shadowgraphs, and interferograms can only be written to TECPLOT visualization files, and the variables for interferograms are only computed correctly when MERGEFLAG=1.

ERRFLAG Specifies whether errors are to be computed. This can only be done if a hard-coded initial solution is used. See also FLAG in the INITFLOW section.

EXTRFLAG Specifies whether data is to be extracted from the solution. This option can only be used if appropriate code to exctract data was added to the file RFLU_ModExtractFlowData.F90 in rflupost for the given casename. If the appropriate code does not exist, rflupost will not extract data, write a warning, and continue executing. At present, extracting solution data is possible for the following casenames:

- onera_c0_2d_100x50 ONERA C0 case [6].
- skews_ms2p0, skews_ms3p0, skews_ms4p0 Shock diffraction problem of Skews [11, 12].
- st_sod1, st_sod1_mp2 Sod shock tube [13], case 1.
- st_sod2, st_sod2_mp2 Sod shock tube [13], case 2.

`stg_2d` Generic shock tube.

`GRADFLAG` Compute gradient-reconstruction errors and write as plotting variable into output file. It can take the following values:

`0` Do not compute gradient-reconstruction errors (default).

`1` Compute gradient-reconstruction errors.

Note that `GRADFLAG=1` is only valid for the following casenames: `gtlin` and `gttri`. Note also that `GRADFLAG=1` is only relevant for TECPLOT output, i.e., if `PLTTYPE=1`

`INTERORDER` Specifies the polynomial order of interpolation. This keyword is only relevant if `INTERTYPE=2`. It can take the following values:

`1` Use linear interpolation.

`2` Use quadratic interpolation.

`INTERTYPE` Specifies whether and how data is to be interpolated from the cell-centers to the vertices. It can take the following values:

`0` Do not transfer the solution from cell centers to vertices. This option avoids the introduction of interpolation errors and can be useful if it is necessary to visualize small solution differences which may be smoothed out by the interpolation process. TECPLOT and ENSIGHT allow the visualization of cell-centered solution data.

`1` Use simple arithmetic averaging.

`2` Use k-exact averaging. This method interpolates polynomials of order k exactly on arbitrary grids. The order k is specified by the value assigned to the keyword `INTERORDER`.

Choosing between `INTERTYPE=1` and `INTERTYPE=2` is largely a matter of balancing the accuracy and cost of the interpolation. Simple arithmetic averaging is much faster but also less accurate than the k-exact interpolation method. In practice, the differences between the two methods are usually negligible unless solution gradients are very large or grids are highly distorted.

`MERGEFLAG` Specifies whether the regions from a parallel computation are to be merged for postprocessing. It can take the following values:

`0` Do not merge the regions for postprocessing. Each partition will be written separately to the output file. Virtual cells and boundary faces will be written separately for each region and patch, respectively.

`1` Merge the regions for postprocessing.

Not merging regions for postprocessing can be useful for several reasons: First, one may want to make sure that solution contours are well-behaved across partition boundaries. Second, when using this option in conjunction with rflupick, the amount of data to be visualized can be reduced drastically by selecting only specific regions to be postprocessed in rflupost. Third, it becomes possible to visualize virtual cells and boundary faces which can be useful during debugging.

`MIXTCVFLAG` Specifies whether the conserved variables of the mixture module should be written to the output file. It can take the following values:

- `0` Do not write mixture conserved variables to output file.
- `1` Write mixture conserved variables to output file (default).

The primary purpose of not writing the mixture conserved variables are to reduce the size of the output file.

`MIXTDVFLAG` Specifies whether the derived variables of the mixture module should be written to the output file. It can take the following values:

- `0` Do not mixture derived write variables to output file.
- `1` Write mixture derived variables to output file (default).

The primary purpose of not writing the mixture conserved variables are to reduce the size of the output file.

`MIXTGVFLAG` Specifies whether the gas variables of the mixture module should be written to the output file. It can take the following values:

- `0` Do not mixture gas write variables to output file.
- `1` Write mixture gas variables to output file (default).

The primary purpose of not writing the mixture conserved variables are to reduce the size of the output file.

`NSERVERS` Specifies the number of servers when writing output files for parallel runs and visualization with ENSIGHT. Only relevant if `OUTFORMAT=2`.

`OUTFORMAT` Specifies the format of the visualization files. It can take the following values:

- `1` Write visualization files in TECPLOT format (default).
- `2` Write visualization files in ENSIGHT format

`PEULFLAG` Specifies whether the Lagrangian particle fields are to be converted to Eulerian particle fields and written to the visualization files. It can take the following values:

- `0` Do not convert Lagrangian fields to Eulerian fields.

1 Convert Lagrangian fields to Eulerian fields.

See also Sec. 9.1.

At present, Eulerian particle fields can only be written to TECPLOT visualization files.

PLAGFRAC Specifies the fraction of Lagrangian particles to be written to output files. The default value is 1.0. At present, PLAGFRAC is only relevant for TECPLOT output, i.e., if PLTTYPE=1.

PLTPATFLAG Specifies whether only the first virtual patch is to be written to output file. It can take the following values:

0 Write all patches to output file.

1 Write only the first patch virtual to the output file (default).

Because virtual patches only exist for one- and two-dimensional computations, PLTPATFLAG is only relevant if DIMENS=1 or DIMENS=2. The main use of setting PLTPATFLAG=1 is to reduce the size of the output file for two-dimensional computations. At present, PLTPATFLAG is only relevant for TECPLOT files, i.e., if OUTFORMAT=1.

PLTTYPE Specifies the data to be written to output file. It can take the following values:

1 Write only the grid to the output file.

2 Write the grid and the solution to output file.

PLTVOLFLAG Specifies whether volume data are to be written to output file. It can take the following values:

0 Do not write volume data for postprocessing; only surface data is written.

1 Write volume and surface data for postprocessing.

The advantage of not writing volume data to the output file is to reduce the amount of data to be visualized.

SPECFLAG Specifies whether the postprocessor-information file produced by rflupick is to be read. It can take the following values:

0 Do not read the postprocessor-information file.

1 Read the postprocessor-information file.

Reading the postprocessor-information file allows only specific regions to be postprocessed. This option is only relevant if MERGEFLAG=0.

Note that special faces can only be written by rflupost to output files if only the grid is written out, i.e., PLTTYPE=1, or if the solution is interpolated to the vertices, i.e., INTERTYPE=1 or INTERTYPE=2.

WRIMERGE Specifies whether merged grid and solution files are to be written. See also MERGEFLAG.

VIRTFLAG Specifies whether the virtual cells and virtual faces on patches should be written to the output file. It can take the following values:

0 Do not write virtual cells and virtual boundary faces to output file (default).

1 Write virtual cells and virtual boundary faces to output file.

VORTFLAG Specifies whether vorticity is to be computed and written to visualization files. It can take the following values:

0 Do not compute vorticity and do not write to visualization files.

1 Compute vorticity and write to visualization files.

See also Sec. 9.1.

At present, vorticity can only be written to TECPLOT visualization files.

5.2.11 PREP Section

The PREP section is relevant only to rflupart. It contains the following keywords:

PARTMODE Specifies the partitioning mode. It can take the following values:

1 Partition the grid with a general method. This will in general lead to well-balanced, but not perfectly balanced, partitions.

2 Partition the grid with an imposed mapping to get perfect load balancing. Note that this will work only if the cells in the grid are numbered like the cells in a structured grid and if the number of regions is a divisor of the number of cells of the unpartitioned grid. Furthermore, it is important to note that the ordering of the cells will have a very strong impact on the resulting partitioning, see Fig. 5.3. Depending on the numbering, it is possible to generate non-contiguous partitions and/or partitions with large surface-to-volume ratios.

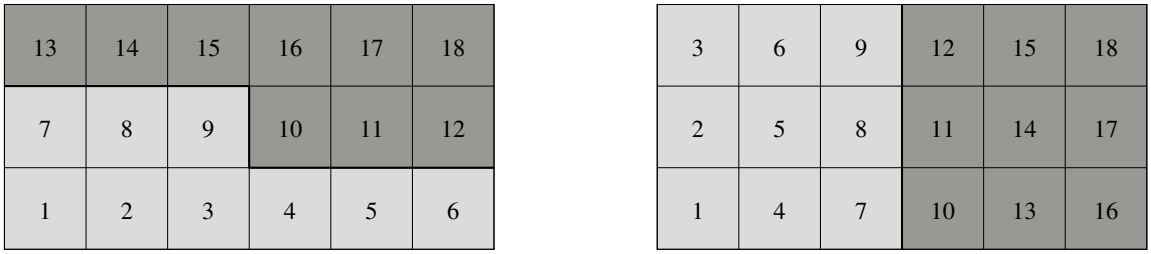

Figure 5.3: Partitioning of a quadrilateral grid into two regions using `PARTMODE=2`, demonstrating importance of cell numbering.

5.2.12 `PROBE` Section

The `PROBE` section consists of two subsections separated by the character `#`. The first subsection contains the following keyword:

`NUMBER` Specifies the number of probes.

Immediately following the keyword `NUMBER`, there must be n lines, where $n =$ `NUMBER`. Each line must contain three real values, which represent the x-, y-, and z-coordinate of a given probe. rflump attempts to find the cell whose centroid is closest to the specified coordinates.

The second part contains the following keywords:

`WRITIME` Offset in seconds at which data is written to probe files.

`WRIITER` Offset between iterations at which data is written to probe files.

`OPENCLOSE` Specifies whether probe files are to closed and opened after writing data. It can take the following values:

`0` Do not close and open probe file after writing data.

`1` Close and open probe file after writing data.

Closing and opening the probe file after writing data can be useful because this forces write buffers to be flushed.

It is important to note that the separation character `#` must be present even if the keywords in the second subsection are not included in the input file. Otherwise the remainder of the input file will not be read correctly.

An example `PROBE` section for unsteady flow is given below.

```
# PROBE
NUMBER 4
0.001 0.000  0.499
0.001 0.000 -0.499
9.999 0.000  0.499
9.999 0.000 -0.499
#
WRITIME 5.0E-4
OPENCLOSE 1
#
```

5.2.13 REFERENCE Section

The REFERENCE section contains the following keywords:

ABSVEL Reference velocity magnitude, [m/s].

CP Reference specific heat coefficient at constant pressure, [J/kg K].

DENS Reference density, [kg/m^3].

GAMMA Reference ratio of specific heats, [-].

LENGTH Reference length, [m].

PRESS Reference static pressure, [Pa].

PRLAM Reference laminar Prandtl number, [-].

PRTURB Reference turbulent Prandtl number, [-].

RENUM Reference Reynolds number, [-].

SCNLAM Reference laminar Schmidt number, [-].

SCNTURB Reference turbulent Schmidt number, [-].

5.2.14 TIMESTEP Section

The TIMESTEP section contains the following keywords:

DTMINLIMIT The time step in seconds below which information will be printed out about the region and cell in which the minimum time step occurs. The additional information can be helpful in diagnosing whether small time steps are due to unexpectedly small cells or cells of poor quality. Once the region and cell with the minimum time step are known, rflupick can be used in conjunction with rflupost to visualize that cell. Only relevant if FLOWTYPE=1.

DTMINLIMIT should only be set to representative positive value if the additional information is actually desired because determining the additional information incurs a performance penalty for parallel runs.

FLOWTYPE Specifies whether flow steady or unsteady. It can take the following values:

- 0 Flow is steady.
- 1 Flow is unsteady.

Note that the value of this keyword influences the name of the flow-solution files.

MAXITER The iteration number at which the computation is to be stopped. A calculation is stopped if either the maximum number of iterations is reached, or if the norm of the density residual has fallen below the residual tolerance. Only relevant if FLOWTYPE=0.

MAXTIME The time in seconds at which the computation is to be stopped. Only relevant if FLOWTYPE=1.

PRNITER Offset between iterations at which convergence information is printed on screen and written to the convergence file. Only relevant if FLOWTYPE=0.

PRNTIME Offset in time in seconds at which convergence information is printed on screen and written to the convergence file. Only relevant if FLOWTYPE=1.

RESTOL The tolerance for the density residual below which the computation is judged to be converged. A calculation is stopped if either the maximum number of iterations is reached, or if the norm of the density residual has fallen below the residual tolerance. Only relevant if FLOWTYPE=0.

RKSCHEME Specifies the Runge-Kutta method used to integrate the discrete equations in time for unsteady flows. It can take the following values:

- 1 Classical RK4 scheme.
- 2 Three-stage Runge-Kutta method derived by Wray XXX ADD REF XXX.

If the Rocpart module is activated, RKSCHEME=2 must be chosen.

STARTITER The iteration number from which the computation is to be started. Only relevant if FLOWTYPE=0.

STARTTIME The time in seconds from which the computation is to be started. Only relevant if FLOWTYPE=1.

TIMESTEP The maximum time step in seconds to be used in the computation. Only relevant if FLOWTYPE=1.

WRIITER Offset between iterations at which flow files are to be written. Only relevant if FLOWTYPE=0.

WRITIME Offset in time in seconds at which flow files are to be written. For unsteady flows with moving grids, the grid files are written also. Only relevant if FLOWTYPE=1.

5.2.15 TRANSFORM Section

The TRANSFORM section is relevant only to rflupart. It contains the following keywords:

ANGLE_X Angle of rotation around x-axis, in degrees. The angle of rotation is positive in the counter-clockwise direction when looking down the x-axis.

ANGLE_Y Angle of rotation around y-axis, in degrees. The angle of rotation is positive in the counter-clockwise direction when looking down the y-axis.

ANGLE_Z Angle of rotation around z-axis, in degrees. The angle of rotation is positive in the counter-clockwise direction when looking down the z-axis.

DIST_FLAG Specifies whether to distort the grid. It can take the following values:

0 Do not distort the grid (default).

1 Distort the grid.

Distorting the grid amounts to adding random perturbations to the coordinates of the vertices in the grid. To preserve the boundary of the computational domain, only the coordinates of interior vertices are perturbed. The distortion is specified by the values assigned to the keywords DIST_X, DIST_Y, and DIST_Z. These represent scaling factors in the three coordinate directions and must be chosen carefully to avoid perturbing vertices to the extent that the cells have negative volumes. The primary use of this option is to test the performance of algorithms on distorted grids.

DIST_X Scaling factor for x-component of grid distortion (default is 0.0). Only relevant if DIST_FLAG=1.

DIST_Y Scaling factor for y-component of grid distortion (default is 0.0). Only relevant if DIST_FLAG=1.

DIST_Z Scaling factor for z-component of grid distortion (default is 0.0). Only relevant if DIST_FLAG=1.

FLAG Specifies whether the grid is to be scaled rotated. It can take the following values:

0 Do not scale and rotate the grid (default).

1 Scale and rotate the grid.

`SCALE_X` Scaling factor for x-component of coordinates.

`SCALE_Y` Scaling factor for y-component of coordinates.

`SCALE_Z` Scaling factor for z-component of coordinates.

It is important to note that initial solutions are defined after geometric transformations.

5.2.16 `VISCMODEL` Section

The `VISCMODEL` section contains the following keywords:

`MODEL` Specifies the viscosity model. It can take the following values:

Sutherland viscosity model, see Section 11.5.1.1.

1 Fixed viscosity. The viscosity value is given by the value assigned to the keyword `VISCOSITY`.

2 Antibes viscosity model see Section 11.5.1.2.

`REFTEMP` Specifies the value of the Sutherland constant [K].

`SUTHCOEF` Specifies the value of the reference temperature T_{ref} in the Sutherland and Antibes models [K].

`VISCOSITY` Reference value for the dynamic viscosity [kg/(ms)].

The reference value for the dynamic viscosity must be chosen carefully because it is used to compute the response times of Lagrangian and Eulerian particles if the Rocpart and/or Rocspecies modules are activated. Otherwise, unphysical response times can result which may lead to instability of the computation.

5.3 Patch-Mapping Files

Patch-mapping files are used to define a mapping between the patches defined in the VGRIDns, MESH3D, TETMESH, and Cobalt grid files and that desired in RocfluMP simulations. The motivation for mapping patches arises because grids are usually generated with more patches than are required to impose boundary conditions for a simulation. For this reason, the patch-mapping files are typically used to reduce the number of patches and therefore the number of boundary conditions.

5.3.1 VGRIDns Patch-Mapping File (.vgi File)

The format of the `.vgi` file is:

Line 1: The number of patches after the mapping.

Line 2: The number of mappings (hereafter referred to as `nMappings`).

The remaining `nMappings` lines: Each line contains three integers. The first two integers represent the lower and upper limits of the patches in the VGRIDns file which are to be mapped to the patch indicated by the third integer. The lower limit must be less than or equal to the upper limit.

The following is an example `.vgi` file:

```
 5
 8
 5  6  1
 1  4  2
12 12  2
 7  7  3
10 10  3
 8  9  4
11 11  4
17 22  5
```

This file indicates that five patches will exist after merging and that eight mappings are listed. For example, the first mapping specifies that patches five and six are mapped to patch one, and the last mapping specifies that patches 17 to 22 are mapped to patch five. Note that it is possible to map several original patches to a given new patch in separate mappings. That is, the following example is equivalent to the one given above:

```
 5
10
 5  5  1
 6  6  1
 1  4  2
12 12  2
 7  7  3
10 10  3
 8  9  4
11 11  4
17 18  5
19 22  5
```

It is important to note that the extrema of the lower and upper original patch indices must be equal to unity and the number of patches specified in the `.bc` file, respectively. Furthermore, the extrema of the new patch indices must be equal to unity and the number of patches specified on the first line, respectively.

In practice, it is usually impossible to specify the mappings without having visually inspected the grid. It is therefore recommended that in a first try, a one-to-one mapping is specified. By inspecting the grid, it is possible to merge appropriate patches by writing the proper mapping file.

5.3.2 MESH3D Patch-Mapping File (`.mgi` File)

The `.mgi` file serves the same purpose as the `.vgi` file. The format of the `.mgi` file is:

Line 1: The number of patches in the `.m3d` file.

Line 2: The number of patches after the mapping.

Line 3: The number of mappings (hereafter referred to as `nMappings`).

The remaining `nMappings` lines: Each line contains three integers. The first two integers represent the lower and upper limits of the patches in the MESH3D file which are to be mapped to the patch indicated by the third integer. The lower limit must be less than or equal to the upper limit.

The first line contains the original number of patches because this information is not contained the `.m3d` file.

One important difference between the `.vgi` and `.mgi` files is that the extrema of the lower and upper limits of the original patches in the `.mgi` file do not have to be equal to unity and the number of patches specified on the second line, respectively. This is because boundary faces in the `.m3d` are grouped by arbitrary flags and not by patch numbers. Hence the following is a valid `.mgi` file:

```
1   22
2    7
3   10
4    5   6   1
5    1   4   2
6   12  12   2
7    7   7   3
8   10  10   3
9    8   9   4
10  11  11   4
11  17  22   5
12 200 600   6
13 100 100   7
```

Note in particular the last two lines: They illustrate that the upper limit of the original patches does not have to be equal to the number of patches in the `.m3d` file.

5.3.3 TETMESH Patch-Mapping File (`.tmi` File)

5.3.4 Cobalt Patch-Mapping File (`.cgi` File)

The `.cgi` file serves the same purpose as the `.vgi` and `.mgi` files. The format of the `.cgi` file is:

Line 1: The number of patches after the mapping.

Line 2: The number of mappings (hereafter referred to as `nMappings`).

The remaining `nMappings` lines: Each line contains three integers. The first two integers represent the lower and upper limits of the patches in the Cobalt file which are to be mapped to the patch indicated by the third integer. The lower limit must be less than or equal to the upper limit.

5.3.5 StarCD Patch-Mapping File (`.sgi` File)

The `.sgi` file serves the same purpose as the `.vgi`, `.mgi`, and `.cgi` files. The format of the `.sgi` file is:

Line 1: The number of patches after the mapping.

Line 2: The number of mappings (hereafter referred to as `nMappings`).

The remaining `nMappings` lines: Each line contains three integers. The first two integers represent the lower and upper limits of the patches in the StarCD file which are to be mapped to the patch indicated by the third integer. The lower limit must be less than or equal to the upper limit.

5.4 Boundary-Condition File

The boundary-condition file is called `<casename>.bc`. Like the input file, the boundary-condition file is divided into sections. Each section contains several lines, each of which consists of a keyword and a value, with the exception of the line containing the keyword `PATCH`, on which two values are listed.

```
# SECTION_NAME
PATCH      PATCH_1  PATCH_2
KEYWORD_1  VALUE_1
KEYWORD_2  VALUE_2
#
```

Each section assigns a boundary condition to either a single patch or to a range of patches. The boundary-condition file must be terminated by the string:

```
# END
```

The sections may be listed in any order in the boundary-condition file, but are listed below in alphabetical order for simplicity.

5.4.1 Physical Boundary Conditions

5.4.1.1 Farfield Boundary: `BC_FARF` Section

The `BC_FARF` section contains the following keywords:

`ATTACK` Angle of attack [deg]. See Fig. 5.4 for the definition of `ATTACK`.

`CORR` Specifies whether farfield point vortex correction is to be applied. It can take the following values:

- `0` Do not apply point-vortex correction.
- `1` Apply point-vortex correction.

Note that at present the correction can only be used for two-dimensional computations and has not been thoroughly tested.

`KIND` Specifies the treatment of the boundary condition. It can take the following values:

- `0` Simple boundary-condition treatment.
- `1` Characteristic boundary-coundition treatment.

`MACH` Mach number [-].

`NAME` Specifies the name of the boundary.

`ORDER` Specifies the order of accuracy of computing fluxes on this boundary. It can take the following values:

- `1` First-order accuracy.
- `2` Second-order accuracy.

The order with which boundary fluxes are computed must be less or equal to the order of accuracy with which interior fluxes are computed as specified by the keyword `ORDER` in the `NUMERICS` section. It can be useful to compute boundary fluxes with first-order accuracy even though interior fluxes are computed with second-order accuracy if strong gradients are swept across the boundary, such as shock waves or jet boundaries.

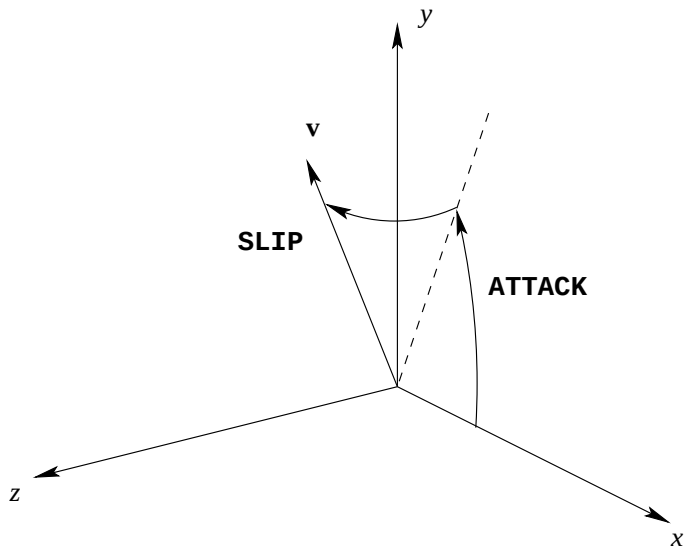

Figure 5.4: Definition of angles `ATTACK` and `SLIP` for farfield boundary condition. Angles are positive in direction of arrows.

`PATCH` Specifies the range of patches to which the data in this section is to be applied using two integers.

`PRESS` Static pressure [Pa].

`SLIP` Sideslip angle [deg]. See Fig. 5.4 for the definition of `SLIP`.

`TEMP` Static temperature [K].

`THRUSTFLAG` Specifies whether thrust and specific impulse are computed for this patch. This flag is relevant only when `FLAG=1` in the `FORCE` section. It can take the following values:

`0` Do not compute thrust and specific impulse for this patch.

`1` Compute thrust and specific impulse for this patch.

5.4.1.2 Inflow Boundary: `BC_INFLOW/BC_INFLOW_TOTANG` Section

The `BC_INFLOW` or `BC_INFLOW_TOTANG` section contains the following keywords:

`BETAH` Angle between velocity vector and its projection onto xz-plane [deg]. See Fig. 5.5 for the definition of `BETAH`.

`BETAV` Angle between the projection of the velocity vector onto the xz-plane and the positive x-axis [deg]. See Fig. 5.5 for the definition of `BETAV`.

`FIXED` Specifies whether the flow is assumed to be normal to the boundary. It can take the following values:

0 Inflow not assumed to be normal to boundary.

1 Inflow assumed to be normal to boundary.

Specifying the flow to be normal to the boundary can be particularly helpful if the inflow boundary represents a reservoir condition or other conditions in which the flow velocity is very small. It is important to note that the motivation for and effect of FIXED is different from specifying the flow direction through BETAH and BETAV. The keyword FIXED influences the angle computed from the extrapolated velocity vector. For vanishing velocity, the determination of this angle becomes ill-conditioned which can lead to failure of computations. Instead, the angle can be fixed so that the velocity, no matter how small, is always normal to the boundary.

KIND Specifies the treatment of the boundary condition. It can take the following values:

0 Simple boundary-condition treatment.

1 Characteristic boundary-coundition treatment.

MACH Mach number [-]. The Mach number must be specified only if the inflow is supersonic.

NAME Specifies the name of the boundary.

ORDER Specifies the order of accuracy of computing fluxes on this boundary. It can take the following values:

1 First-order accuracy.

2 Second-order accuracy.

The order with which boundary fluxes are computed must be less or equal to the order of accuracy with which interior fluxes are computed as specified by the keyword ORDER in the NUMERICS section. It can be useful to compute boundary fluxes with first-order accuracy even though interior fluxes are computed with second-order accuracy if strong gradients are swept across the boundary, such as shock waves or jet boundaries.

PATCH Specifies the range of patches to which the data in this section is to be applied using two integers.

PTOT Total pressure [Pa].

THRUSTFLAG Specifies whether thrust and specific impulse are computed for this patch. This flag is relevant only when FLAG=1 in the FORCE section. It can take the following values:

0 Do not compute thrust and specific impulse for this patch.

1 Compute thrust and specific impulse for this patch.

TYPE Specifies whether inflow is supersonic or subsonic. It can take the following values:

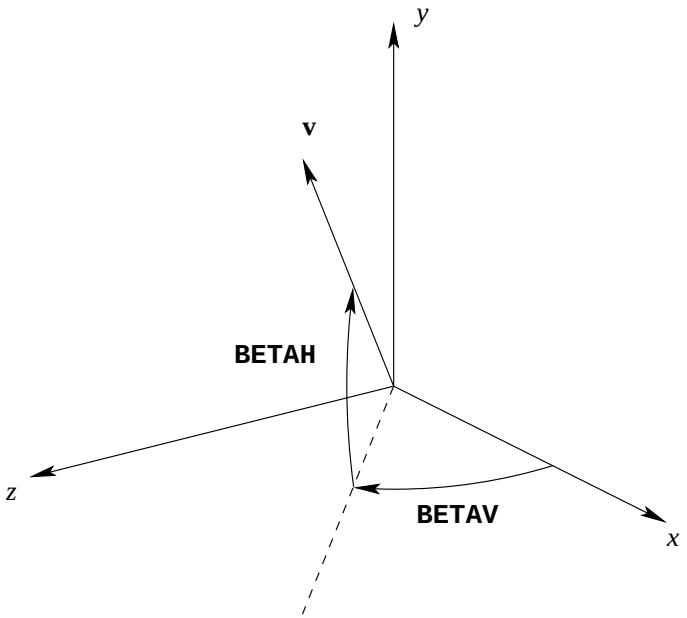

Figure 5.5: Definition of angles `BETAH` and `BETAV` for inflow boundary condition. Angles are positive in direction of arrows.

`0` Inflow is supersonic. The Mach number must be specified by the keyword `MACH`.

`1` Inflow is subsonic.

`TTOT` Total temperature [K].

Specification of Inflow Angles. For planar boundary patches with known normal vector $\mathbf{n} = \{n_x, n_y, n_z\}^t$ and inflow normal to the boundary, the inflow angles are given by

$$\beta_v = \tan^{-1}\left(\frac{n_z}{n_x}\right), \tag{5.1}$$

$$\beta_h = -\sin^{-1} n_y. \tag{5.2}$$

For imposed velocities and flow which is not necessarily normal to the boundary, the inflow angles are given by

$$\beta_v = \tan^{-1}\left(\frac{w}{u}\right), \tag{5.3}$$

$$\beta_h = \sin^{-1}\left(\frac{v}{\|\mathbf{v}\|}\right). \tag{5.4}$$

The section name `BC_INFLOW` is obsolete and should be replaced by `BC_INFLOW_TOTANG`. For backward compatibility, RocfluMP treats the two boundary conditions in the same way.

5.4.1.3 Inflow Boundary: BC_INFLOW_VELTEMP Section

The BC_INFLOW_VELTEMP section contains the following keywords:

KIND Specifies the treatment of the boundary condition. It can take the following values:

- 0 Simple boundary-condition treatment.
- 1 Characteristic boundary-coundition treatment.

NAME Specifies the name of the boundary.

ORDER Specifies the order of accuracy of computing fluxes on this boundary. It can take the following values:

- 1 First-order accuracy.
- 2 Second-order accuracy.

The order with which boundary fluxes are computed must be less or equal to the order of accuracy with which interior fluxes are computed as specified by the keyword ORDER in the NUMERICS section. It can be useful to compute boundary fluxes with first-order accuracy even though interior fluxes are computed with second-order accuracy if strong gradients are swept across the boundary, such as shock waves or jet boundaries.

PATCH Specifies the range of patches to which the data in this section is to be applied using two integers.

PRESS Static pressure [Pa]. Only read if TYPE=0.

REFLECT Specifies whether the boundary condition is reflecting or non-reflecting. It can take following values:

- 0 Non-reflecting boundary condition.
- 1 Reflecting boundary condition.

This option is relevant only if KIND=1.

TEMP Static temperature [K].

THRUSTFLAG Specifies whether thrust and specific impulse are computed for this patch. This flag is relevant only when FLAG=1 in the FORCE section. It can take the following values:

- 0 Do not compute thrust and specific impulse for this patch.
- 1 Compute thrust and specific impulse for this patch.

TYPE Specifies whether inflow is supersonic or subsonic. It can take the following values:

- 0 Inflow is supersonic. The static pressure must be specified by the keyword PRESS.

1 Inflow is subsonic.

VELX x-component of fluid velocity [m/s].

VELY y-component of fluid velocity [m/s].

VELZ z-component of fluid velocity [m/s].

5.4.1.4 Injection Boundary: BC_INJECT Section

The BC_INJECT section contains the following keywords:

BFLAG Specifies whether this patch is burning at $t = 0$. Only relevant for computations with GENx. It can take the following values:

0 Patch is not burning at $t = 0$.

1 Patch is burning at $t = 0$.

COUPLED Specifies whether patch is interacting during a computation with GENx. It can take the following values:

1 Patch is interacting.

2 Patch is not interacting.

CRECONST Specifies whether constrained reconstruction is to be used when computing face gradients for this boundary. It can take the following values:

0 Do not use constrained reconstruction (default).

1 Use constrained reconstruction.

KIND Specifies the treatment of the boundary condition. It can take the following values:

0 Simple boundary-condition treatment.

1 Characteristic boundary-coundition treatment.

MFRATE Injection mass flux [kg/(m^2 s)].

NAME Specifies the name of the boundary.

ORDER Specifies the order of accuracy of computing fluxes on this boundary. It can take the following values:

1 First-order accuracy.

2 Second-order accuracy.

The order with which boundary fluxes are computed must be less or equal to the order of accuracy with which interior fluxes are computed as specified by the keyword ORDER in the NUMERICS section. It can be useful to compute boundary fluxes with first-order accuracy even though interior fluxes are computed with second-order accuracy if strong gradients are swept across the boundary, such as shock waves or jet boundaries.

PATCH Specifies the range of patches to which the data in this section is to be applied using two integers.

STATS Specifies whether particle impact statistics are to be gathered. It can take the following values:

0 Do not gather paticle impact statistics.

1 Gather paticle impact statistics.

Particle impact statistics are written into the patch-coefficient file for visualization with TECPLOT.

TEMP Injection static temperature [K].

THRUSTFLAG Specifies whether thrust and specific impulse are computed for this patch. This flag is relevant only when FLAG=1 in the FORCE section. It can take the following values:

0 Do not compute thrust and specific impulse for this patch.

1 Compute thrust and specific impulse for this patch.

5.4.1.5 No-Slip Boundary: BC_NOSLIP_HFLUX Section

The BC_NOSLIP_HFLUX section is used to specify boundary conditions for patches with a no-slip boundary condition and imposed heat fluxes and contains the following keywords:

COUPLED Specifies whether patch is interacting during a computation with GENx. It can take the following values:

1 Patch is interacting.

2 Patch is not interacting.

CRECONST Specifies whether constrained reconstruction is to be used when computing face gradients for this boundary. It can take the following values:

0 Do not use constrained reconstruction (default).

1 Use constrained reconstruction.

If you wish to compute viscous flows, you must set `CRECONST=1`. Otherwise, the no-slip condition is only enforced weakly and you may observe non-zero velocities on solid boundaries.

`HFLUX` Specifies the value of the heat flux [W/(m^2 K)].

`KIND` Specifies the treatment of the boundary condition. It can take the following values:

- `0` Simple boundary-condition treatment.
- `1` Characteristic boundary-coundition treatment.

`NAME` Specifies the name of the boundary.

`ORDER` Specifies the order of accuracy of computing fluxes on this boundary. It can take the following values:

- `1` First-order accuracy.
- `2` Second-order accuracy.

The order with which boundary fluxes are computed must be less or equal to the order of accuracy with which interior fluxes are computed as specified by the keyword `ORDER` in the `NUMERICS` section. It can be useful to compute boundary fluxes with first-order accuracy even though interior fluxes are computed with second-order accuracy if strong gradients are swept across the boundary, such as shock waves or jet boundaries.

`PATCH` Specifies the range of patches to which the data in this section is to be applied using two integers.

`STATS` Specifies whether particle impact statistics are to be gathered. It can take the following values:

- `0` Do not gather paticle impact statistics.
- `1` Gather paticle impact statistics.

Particle impact statistics are written into the patch-coefficient file for visualization with TECPLOT.

`THRUSTFLAG` Specifies whether thrust and specific impulse are computed for this patch. This flag is relevant only when `FLAG=1` in the `FORCE` section. It can take the following values:

- `0` Do not compute thrust and specific impulse for this patch.
- `1` Compute thrust and specific impulse for this patch.

5.4.1.6 No-Slip Boundary: BC_NOSLIP_TEMP Section

The BC_NOSLIP_TEMP section is used to specify boundary conditions for patches with an isothermal no-slip boundary condition and contains the following keywords:

COUPLED Specifies whether patch is interacting during a computation with GENx. It can take the following values:

- 1 Patch is interacting.
- 2 Patch is not interacting.

CRECONST Specifies whether constrained reconstruction is to be used when computing face gradients for this boundary. It can take the following values:

- 0 Do not use constrained reconstruction (default).
- 1 Use constrained reconstruction.

If you wish to compute viscous flows, you must set CRECONST=1. Otherwise, the no-slip condition is only enforced weakly and you may observe non-zero velocities on solid boundaries.

KIND Specifies the treatment of the boundary condition. It can take the following values:

- 0 Simple boundary-condition treatment.
- 1 Characteristic boundary-coundition treatment.

NAME Specifies the name of the boundary.

ORDER Specifies the order of accuracy of computing fluxes on this boundary. It can take the following values:

- 1 First-order accuracy.
- 2 Second-order accuracy.

The order with which boundary fluxes are computed must be less or equal to the order of accuracy with which interior fluxes are computed as specified by the keyword ORDER in the NUMERICS section. It can be useful to compute boundary fluxes with first-order accuracy even though interior fluxes are computed with second-order accuracy if strong gradients are swept across the boundary, such as shock waves or jet boundaries.

PATCH Specifies the range of patches to which the data in this section is to be applied using two integers.

STATS Specifies whether particle impact statistics are to be gathered. It can take the following values:

0 Do not gather paticle impact statistics.

1 Gather paticle impact statistics.

Particle impact statistics are written into the patch-coefficient file for visualization with TECPLOT.

TEMP Specifies the value of the temperature [K].

THRUSTFLAG Specifies whether thrust and specific impulse are computed for this patch. This flag is relevant only when FLAG=1 in the FORCE section. It can take the following values:

0 Do not compute thrust and specific impulse for this patch.

1 Compute thrust and specific impulse for this patch.

5.4.1.7 Outflow Boundary: BC_OUTFLOW Section

The BC_OUTFLOW section contains the following keywords:

KIND Specifies the treatment of the boundary condition. It can take the following values:

0 Simple boundary-condition treatment.

1 Characteristic boundary-coundition treatment.

NAME Specifies the name of the boundary.

NSCBCK Specifies the degree of reflection at boundary. The boundary is non-reflecting if NSCBK=0.0, partially reflecting if NSCBK>0.0 and NSCBK<1.0, and reflecting if NSCBK=1.0. This option is relevant only if KIND=1 and REFLECT=0.

ORDER Specifies the order of accuracy of computing fluxes on this boundary. It can take the following values:

1 First-order accuracy.

2 Second-order accuracy.

The order with which boundary fluxes are computed must be less or equal to the order of accuracy with which interior fluxes are computed as specified by the keyword ORDER in the NUMERICS section. It can be useful to compute boundary fluxes with first-order accuracy even though interior fluxes are computed with second-order accuracy if strong gradients are swept across the boundary, such as shock waves or jet boundaries.

PATCH Specifies the range of patches to which the data in this section is to be applied using two integers.

PRESS Static pressure [Pa]. The static pressure must only be specified if the outflow is subsonic or mixed subsonic/supersonic.

`REFLECT` Specifies whether the boundary condition is reflecting or non-reflecting. It can take following values:

- `0` Non-reflecting boundary condition.
- `1` Reflecting boundary condition.

This option is relevant only if `KIND=1`.

`THRUSTFLAG` Specifies whether thrust and specific impulse are computed for this patch. This flag is relevant only when `FLAG=1` in the `FORCE` section. It can take the following values:

- `0` Do not compute thrust and specific impulse for this patch.
- `1` Compute thrust and specific impulse for this patch.

`TYPE` Specifies whether outflow is supersonic or subsonic. It can take the following values:

- `0` Outflow is supersonic.
- `1` Outflow is subsonic. The static pressure must be specified by the keyword `PRESS`.
- `2` Outflow is mixed subsonic/supersonic. The static pressure must be specified by the keyword `PRESS`.

5.4.1.8 Periodic Boundary: `BC_PERIODIC` Section

`ANGLE` Specifies the angle between two related periodic patches [deg]. For linear periodicity, `ANGLE=0.0`. For rotational periodicity, the angle is that which transform the current patch to be coincident with its related patch (as specified by the value assigned to the keyword `RELPATCH`). The convention is that positive angles are defined according to the right-hand rule assuming that the rotation axis (as specified by the value assigned to the keyword `AXIS`) is pointing in the respective positive coordinate direction; see Fig. 5.6 for an example.

`AXIS` For linear periodicity, specifies the direction of the translation vector which transforms the current patch with its related patch. For rotational periodicity, specifies the direction of the axis about which the current patch is rotated to coincide with its related patch. It can take the following values:

- `1` x-coordinate direction.
- `2` y-coordinate direction.
- `3` z-coordinate direction.

`NAME` Specifies the name of the boundary.

`PATCH` Specifies the range of patches to which the data in this section is to be applied using two integers.

`RELPATCH` Specifies the patch to which the current patch is related.

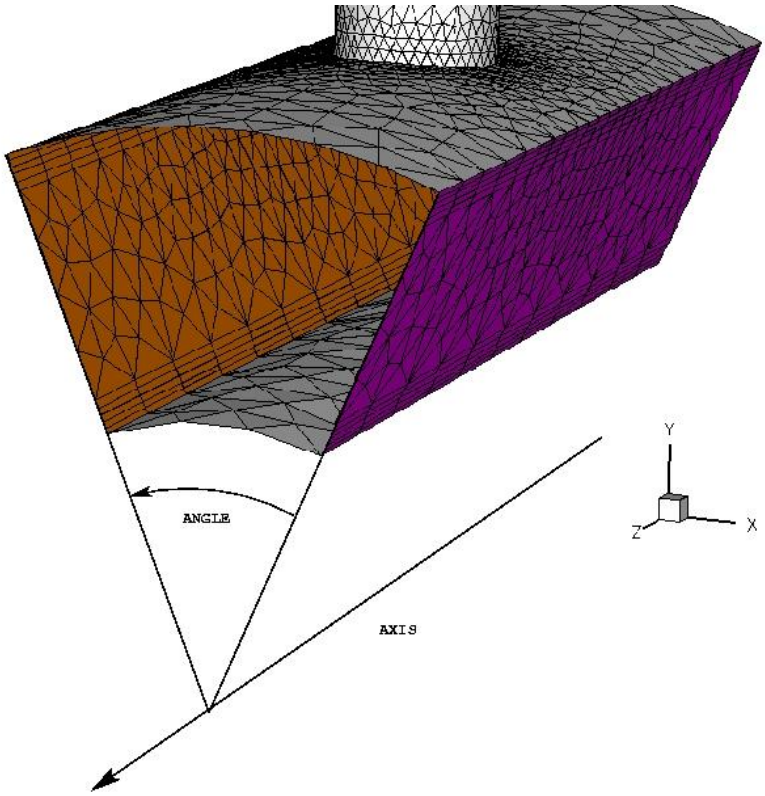

Figure 5.6: Definition of rotation angle for periodic boundaries. In this example, the rotation angle for the purple patch would be positive while that for the orange patch would be negative.

5.4.1.9 Slip Boundary: BC_SLIPW Section

The BC_SLIPW section contains the following keywords:

COUPLED Specifies whether patch is interacting during a computation with GENx. It can take the following values:

- 0 Patch is interacting.
- 2 Patch is not interacting.

CRECONST Specifies whether constrained reconstruction is to be used when computing face gradients for this boundary. It can take the following values:

- 0 Do not use constrained reconstruction (default).
- 1 Use constrained reconstruction.

KIND Specifies the treatment of the boundary condition. It can take the following values:

- 0 Simple boundary-condition treatment.
- 1 Characteristic boundary-coundition treatment.

NAME Specifies the name of the boundary.

ORDER Specifies the order of accuracy of computing fluxes on this boundary. It can take the following values:

- 1 First-order accuracy.
- 2 Second-order accuracy.

The order with which boundary fluxes are computed must be less or equal to the order of accuracy with which interior fluxes are computed as specified by the keyword ORDER in the NUMERICS section. It can be useful to compute boundary fluxes with first-order accuracy even though interior fluxes are computed with second-order accuracy if strong gradients are swept across the boundary, such as shock waves or jet boundaries.

PATCH Specifies the range of patches to which the data in this section is to be applied using two integers.

STATS Specifies whether particle impact statistics are to be gathered. It can take the following values:

- 0 Do not gather paticle impact statistics.
- 1 Gather paticle impact statistics.

Particle impact statistics are written into the patch-coefficient file for visualization with TECPLOT.

THRUSTFLAG Specifies whether thrust and specific impulse are computed for this patch. This flag is relevant only when FLAG=1 in the FORCE section. It can take the following values:

0 Do not compute thrust and specific impulse for this patch.

1 Compute thrust and specific impulse for this patch.

5.4.1.10 Symmetry Boundary: BC_SYMMETRY Section

NAME Specifies the name of the boundary.

PATCH Specifies the range of patches to which the data in this section is to be applied using two integers.

The symmetry boundary condition can only be applied to flat patches. The symmetry boundary condition can be applied to patches which are not xy-, xz-, or xy-coordinate planes.

5.4.1.11 Virtual Boundary: BC_VIRTUAL Section

The BC_VIRTUAL section contains the following keywords:

NAME Specifies the name of the boundary.

PATCH Specifies the range of patches to which the data in this section is to be applied using two integers.

This boundary condition should only be applied with truly one or two-dimensional computations, see the description of the keyword DIMENS in the NUMERICS Section. It is important to note that virtual boundaries can only be applied to y = constant and/or z = constant planes and must come in pairs. Virtual patches with y = constant are only allowed in truly one-dimensional computations. Virtual patches with z = constant are only allowed in truly one- or two-dimensional computations.

The effect of this boundary condition is that no fluxes are computed on the specified patch. Because virtual boundaries always come in pairs and each cell in a truly one or two-dimensional computation must have one face each on the virtual boundaries, the effect of computing no fluxes on this boundary is to simulate constant properties in the y- and/or z-direction.

5.4.2 Time-Dependent Boundary Conditions

Any of the physical quantities specified by the user in the above-described sections can be specified to vary in time. The time variations can be specified to be piecewise linear, sinusoidal, or stochastic using additional sections in the boundary-condition file. Each of these sections can be used to modify one user-specified physical quantity on one boundary patch. If more than one physical quantity on a given patch or physical quantities on several patches are to be modified, multiple additional sections have to be included in the boundary-condition file.

5.4.2.1 `TBC_PIECEWISE` Section

The `TBC_PIECEWISE` section allows the specification of piecewise constant and piecewise linear variations. Consider the following example section:

```
1  # TBC_PIECEWISE
2  INJECT    MFRATE ! BC and variable to which TBC applies
3  PATCH     1  1   ! applies to patch
4  ONTIME   -1.0E6  ! time to start using this TBC
5  OFFTIME   1.0E6  ! time to stop  using this TBC
6  ORDER     0      ! 0 = piecewise constant (default), 1 = piecewise linear
7  NJUMPS    4      ! number of points at which behavior changes
8  #
9  FRAC  0.0        ! fraction of input value of variable before first time
10 TIME  0.001      ! first time at which behavior changes
11 FRAC  0.1        ! next fraction attained (constant) or ramped to (linear)
12 TIME  0.002      ! second time at which behavior changes
13 FRAC  0.3
14 TIME  0.003
15 FRAC  0.6
16 TIME  0.004      ! final time at which behavior changes
17 FRAC  1.0        ! final value for constant case; *ignored* for linear case
18 #
```

Line 2: Specifies that the variable `MFRATE` on an `INJECT` boundary is to be modified. Note that this does not yet specify which injection boundary is to be modified.

Line 3: Specifies that variables on patch one are to be modified.

Line 4: Specifies the lower bound on the time window in which the values are to be modified.

Line 5: Specifies the upper bound on the time window in which the values are to be modified.

Line 6: Specifies the polynomial order of interpolation. It can take the following values:

`0` For piecewise constant interpolation.

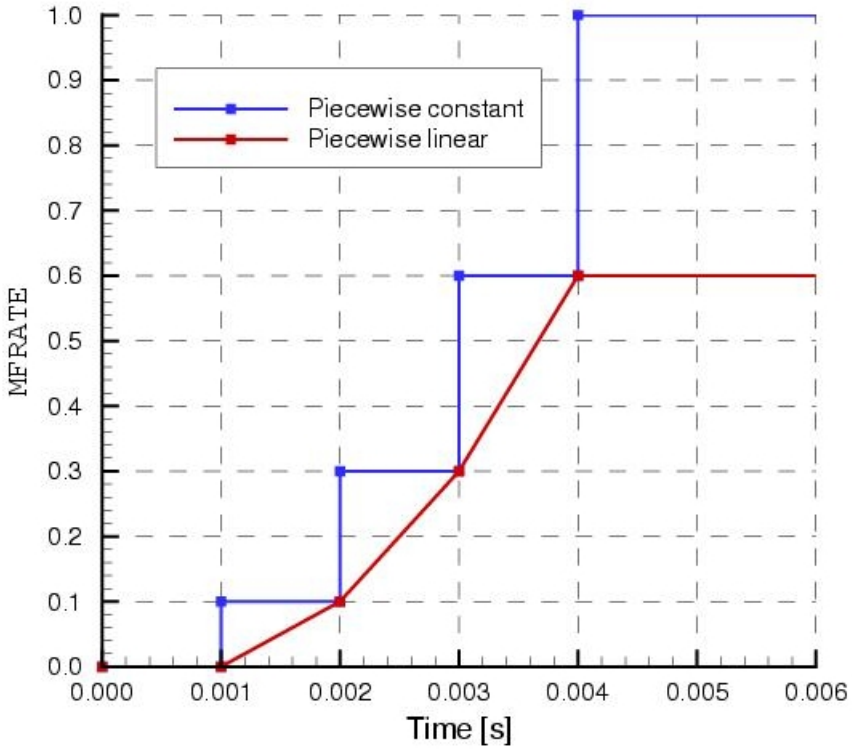

Figure 5.7: Illustration of piecewise constant and linear interpolations for `TBC_PIECEWISE` time-dependent boundary conditions.

`1` For piecewise linear interpolation.

Line 7: Specifies the number of data points through which the time-dependent behavior is specified.

Lines 9-17: Specify the behaviour of the user-specified variable in time. The behaviour is specified through pairs of values for the user-specified value and the time at which the variation changes. The user-specified value of the variable `MFRATE` is modified by the fraction `FRAC`; i.e., at any time, the actual time-dependent value is given by the product of `MFRATE` and `FRAC`. Figure 5.7 illustrates the piecewise constant and linear variations arising from the input of the above example. It is important to note that the piecewise constant and linear representations differ in their final value, because the final value of `FRAC` is ignored for linear variations.

5.4.2.2 `TBC_SINUSOIDAL` Section

The `TBC_SINUSOIDAL` section allows the specification of sinusoidal variations of the form

$$\alpha(t) = c\,(1 + A\sin(\omega t + \phi))$$

where c is the user-specified constant value, A is the amplitude if the sinusoid, ω is the angular frequency, and ϕ is the phase.

Consider the following example section:

```
1 # TBC_SINUSOIDAL
2 OUTFLOW   PRESS  ! BC and variable to which TBC applies
3 PATCH     2  2   ! applies to patch
4 ONTIME    1.0E-3 ! time to start using this TBC
5 OFFTIME   2.0E-3 ! time to stop  using this TBC
6 AMP       0.2    ! amplitude of sinusoid
7 FREQ      1.0E4  ! frequency of sinusoid
8 PHASE     0.0    ! argument of sin() for t=0
9 #
```

Line 2: Specifies that the variable `PRESS` of an `OUTFLOW` boundary is to be modified. Note that this does not yet specify which outflow boundary is to be modified.

Line 3: Specifies that variables on patch two are to be modified.

Line 4: Specifies the lower bound on the time window in which the values are to be modified.

Line 5: Specifies the upper bound on the time window in which the values are to be modified.

Line 6: Specifies the amplitude A of the sinusoidal variation.

Line 7: Specifies the angular frequency ω of the sinusoidal variation [rad/s].

Line 8: Specifies the phase ϕ of the sinusoidal variation [deg].

5.4.2.3 `TBC_STOCHASTIC` Section

5.4.2.4 `TBC_WHITENOISE` Section

5.4.3 Grid-Motion Boundary Conditions

Computations with moving boundaries require the specification of boundary conditions for the grid-motion algorithm. The following keywords can be specified in any of the above sections:

`MVPATCH` Specifies whether patch is moving. It can take the following values:

- `0` Patch is not moving.
- `1` Patch is moving.

`SMGRID` Specifies whether grid on patch is to be smoothed. It can take the following values:

- `0` Do not smooth surface grid.

1 Smooth surface grid.

It is important to note that surface grids can only be smoothed if the associated patches are flat. rflump checks whether patches with active smoothing are flat. If such patches are not flat, smoothing is deactivated automatically and a warning is printed to the screen.

MOVEDIR Specifies in which direction(s) the vertices on the patch are allowed to move. The direction(s) in which movement is allowed are indicated by integers:

0 Vertices on patch are not allowed to be moved in any direction.

1 Vertices on patch are allowed to be moved in x-coordinate direction.

2 Vertices on patch are allowed to be moved in y-coordinate direction.

4 Vertices on patch are allowed to be moved in z-coordinate direction.

The non-zero values can be combined to specify movement in planes normal to coordinate axes or arbitrary movement:

3 Vertices on patch are allowed to be moved in xy-plane, i.e., movement normal to the z-coordinate direction is not allowed.

5 Vertices on patch are allowed to be moved in xz-plane, i.e., movement normal to the y-coordinate direction is not allowed.

6 Vertices on patch are allowed to be moved in yz-plane, i.e., movement normal to the x-coordinate direction is not allowed.

7 Vertices on patch are allowed to be moved in xyz-space.

5.5 Region-Mapping File

The region-mapping file contains the mappings between regions and processes. The file is always in ASCII format and is called <casename>.map.

Consider the following example region-mapping file:

```
1  # ROCFLU region mapping file
2  # Number of regions
3        4
4  # Number of processes
5        4
6  # Process 000001
7        1
8        1
9  # Process 000002
10       1
```

```
11        2
12  # Process 000003
13        1
14        3
15  # Process 000004
16        1
17        4
18  # End
```

Line 1: The first line must contain the string shown, otherwise reading of the file will fail.

Lines 2-5: Specify the number of regions and processes.

Lines 6-8: Specify that process one will run one region and that that region's index is one.

Lines 9-17: Specify that processes two to four will run one region each and that the corresponding region indices are two, three, and four, respectively.

Line 18: The last line must contain the string shown, otherwise reading of the file will fail.

It is instructive to consider the two additional examples shown below. On the left is the region-mapping file for the case in which four regions are to be run on two processes. On the right is the region-mapping file for the case in which four regions are to be run on a single process.

```
1   # ROCFLU region mapping file
2   # Number of regions
3         4
4   # Number of processes
5         2
6   # Process 000001
7         2
8         1
9         2
10  # Process 000002
11        2
12        3
13        4
14  # End
```

```
# ROCFLU region mapping file
# Number of regions
      4
# Number of processes
      1
# Process 000001
      4
      1
      2
      3
      4
# End
```

5.6 Restart-Information File

The restart-information file contains the iteration numbers or time stamps at which restart files were written by rflump. It is read by rflump to determine the iteration number or time stamp from which a restart should be made. Restarts are always made from the iteration

number or time stamp on the last line in the restart-information file. The initial restart-information file is written by rfluprep.

The restart-information file allows jobs to be restarted automatically without user intervention. This is particularly useful when running on computers which require jobs to be submitted through batch queues.

5.7 Convergence File

The convergence file is called `<casename>.con` and is written in ASCII format. For steady flows, the convergence file contains the following information:

Column 1: The iteration number.

Column 2: The residual.

Column 3: The x-component of the net force on solid walls [N].

Column 4: The y-component of the net force on solid walls [N].

Column 5: The z-component of the net force on solid walls [N].

Column 6: The mass flow entering the solution domain [kg/s].

Column 7: The mass flow exiting the solution domain [kg/s].

For unsteady flows, the convergence file contains the following information:

Column 1: The time [s].

Column 2: The time step [s].

Column 3: The x-component of the net force on solid walls [N].

Column 4: The y-component of the net force on solid walls [N].

Column 5: The z-component of the net force on solid walls [N].

Column 6: The mass flow entering the solution domain [kg/s].

Column 7: The mass flow exiting the solution domain [kg/s].

5.8 Probe File

The probe file is written by rflump if the input file contains the PROBE section and setting the variable NUMBER to an integer greater than 0.

The name of the probe file is <casename>.prb_mmmmm, where mmmmm is the number of the probe.

Each line of the probe file consists of seven columns, which contain the following pieces of data:

Column 1: For steady flows, the iteration number; for unsteady flows, the time [s].

Column 2: The density [kg/m^3].

Column 3: The x-velocity component [m/s].

Column 4: The y-velocity component [m/s].

Column 5: The z-velocity component [m/s].

Column 6: The static pressure [Pa].

Column 7: The static temperature [K].

5.9 Moving-Reference-Frame File

The moving-reference frame file is written by rflump if the input file contains a MVFRAME Section and if the FLAG variable in that section is set to 1.

The name of the moving-reference-frame file is <casename>.vel_mmmm where mmmm is the global index of the patch defining the body being accelerated, see MVFRAME Section.

Each line of the moving-reference frame consists of ten columns, which contain the following pieces of data:

Column 1: The time.

Column 2: The x-component of the location of the body [m].

Column 3: The y-component of the location of the body [m].

Column 4: The z-component of the location of the body [m].

Column 5: The x-component of the velocity of the body [m/s].

Column 6: The y-component of the velocity of the body [m/s].

Column 7: The z-component of the velocity of the body [m/s].

Column 8: The x-component of the acceleration of the body [m/s^2].

Column 9: The y-component of the acceleration of the body [m/s^2].

Column 10: The z-component of the acceleration of the body [m/s^2].

5.10 Mass-Conservation Check File

The mass-conservation check file is written by rflump for unsteady flows with grid motion outside of GENx. The file is called `<casename>.mass` and contains the following information:

Column 1: The time [s].

Column 2: The mass contained in the solution domain [kg].

Column 3: The mass flow entering the solution domain [kg/s].

Column 4: The mass flow exiting the solution domain [kg/s].

Column 5: The volume of the solution domain [m^3].

As implied by its name, the mass-conservation check file is used to check that mass is conserved during moving-grid computations.

5.11 Statistics File

5.12 GENx Control File

When rflump is run within GENx, an additional input file is required because rflump is not invoked from the command line. This so-called GENx control file is always called `RocfluControl.txt` and contains the following information:

Line 1: The case name.

Line 2: The directory name containing the rflump input files, relative to the directory from which GENx is invoked.

Line 3: The directory name containing the rflump output files, relative to the directory from which GENx is invoked. written by Roccom.

Line 4: The verbosity level.

Line 5: The checking level.

Chapter 6

Problem Setup

This chapter gives a summary of the problem setup. The objective is to assist users in setting up a run, especially for coupled runs within GENx.

1. Generate a grid. The names and number of output files will differ depending on which grid generator is used:

   - CENTAUR: Output file is `*.hyb.asc` or `*.hyb.bin`. Rename the output files to `<casename>.hyb.asc` or `<casename>.hyb.bin`.
   - VGRIDns: Output files are `*.bc` and `*.cgosg`. Rename `*.bc` to `<casename>.vbc`. This is important because rflump uses the extension `.bc` for its boundary-condition file. Rename `*.cgosg` to `<casename>.cgosg`. Generate a patch-mapping file `<casename>.vgi` as described in Sec. 5.3.1.
   - MESH3D: Output file is `*.m3d`. Rename to `<casename>.m3d`. Generate a patch-mapping file `<casename>.mgi` as described in Sec. 5.3.2.
   - TETMESH: Output file is `*.noboite`. Rename to `<casename>.noboite`. Generate a patch-mapping file `<casename>.tmi` as described in Sec. 5.3.3.
   - GRIDGEN: Specify Cobalt as the output format. Output files are `*.bc` and `*.inp`. Because these extensions are used by rflump, it is recommended that you rename these files as `*-COBALT.bc` and `*-COBALT.inp`. Link `<casename>.cgr` to `*-COBALT.inp`,

     ```
     ln -s *-COBALT.inp <casename>.cgr
     ```

     Generate a patch-mapping file `<casename>.cgi` as described in Sec. 5.3.4.

2. Generate a boundary-condition file `<casename>.bc`. The format of the boundary-condition file is described in 5.4. Pay particular attention to the following:

   - Make sure that the patch-mapping files generated above are consistent with the boundary-condition file, i.e., each patch in the grid is mapped to a patch in

the boundary-condition file. Changing the patch-mapping file means that the preprocessor will need to be rerun.

- For coupled simulations: Make sure that each patch has the correct values for the following parameters: `COUPLED`, `MVPATCH`, and `SMGRID`. Incorrect values of the `COUPLED` parameter are likely to lead to failure of Surfdiver.

3. Generate an input file `<casename>.inp`. Pay particular attention to the following sections:

   - Format of grid file: Make sure you set the `GRIDSRC` keyword in the `FORMATS` section to the correct value.
   - Initial condition: Make sure that the initial condition is correct because changing the initial condition means that the preprocessor will need to be rerun. In particular, make sure that the initial condition is sensible and compatible with the boundary conditions.
   - Reference values: Make sure that the values for `CP` and `GAMMA` are correct. They have a very strong effect on steady-state pressure levels for typical rocket problems. Changing these values influences the initial solution and means that the preprocessor will need to be rerun.
   - For coupled computations, make sure that the `PREP` section contains the keyword `SURFFLAG` set to `1` if rfluprep was not compiled as part of GENx.
   - For coupled computations, make sure that the units are correct. If the grid was generated in units other than meters (as is often the case), add a `TRANSFORM` section with the appropriate scaling factors. Inconsistent units will lead to failure of Surfdiver.

4. Generate a processor-mapping file if parallel runs are to be made by executing

   ```
   rflumap -c <casename> -m 1 -p <nprocs> -r <nregions> -v <verbosity>
   ```

   Note that the number of regions can be larger than the number of processes. See Sec. 7.5.

5. Run the partitioning module by executing

   ```
   rflupart -c <casename> -v <verbosity>
   ```

   See Sec. 7.7. The partitioning module generates grid files in rflump format.

6. Run the initialization module by executing

```
rfluinit -c <casename> -v <verbosity>
```

See Sec. 7.4. The initialization module generates solution files in rflump format.

7. For coupled computations, you need to generate an input file for Rocin by running rflumap again,

```
rflumap -c <casename> -m 2 -p <nprocs> -r <nregions> -v <verbosity>
```

See Sec. 7.5.

8. For coupled computations, generate the `RocfluControl.txt` file, see Sec. 5.12.

9. You are ready to run either rflump or GENx.

Chapter 7

Execution

This chapter contains detailed information on the command-line arguments and input and output files of rfluclone, rfluconv, rfluextr, rfluinit, rflumap, rflump, rflupart, rflupick, and rflupost.

7.1 rfluclone

7.1.1 Invocation

rfluclone is invoked by typing

```
rfluclone -c <casename> -v <verbosity>
```

The following command-line arguments are read by rfluconv:

`<casename>` A character string used to label the input and output files.

`<verbosity>` An integer indicating the desired verbosity level of rfluconv. The verbosity level can take the following values:

- `0` No output. rfluconv will not write any information to standard output.
- `1` Low level of output. rfluconv will write some information to standard output.
- `2` High level of output. rfluconv will write detailed information to standard output.

7.1.2 Input Files

The following input files are read by rfluclone:

- A grid file in RocfluMP ASCII or binary format.
- A dimension file.

- A boundary condition file.
- A region mapping file.

7.1.3 Output Files

The following output files are written by rfluclone:

- A grid file in RocfluMP ASCII or binary format for each region.
- A communication file for each region.
- A version file called `rfluclone.vrs`. It contains the version number and date of the executable. Successive runs append to the version file.

7.2 rfluconv

7.2.1 Invocation

rfluconv is invoked by typing

```
rfluconv -c <casename> -s <stamp> -v <verbosity>
```

The following command-line arguments are read by rfluconv:

`<casename>` A character string used to label the input and output files.

`<stamp>` A variable indicating the iteration or time from which the grid and solution files are to be read.

`<verbosity>` An integer indicating the desired verbosity level of rfluconv. The verbosity level can take the following values:

- 0 No output. rfluconv will not write any information to standard output.
- 1 Low level of output. rfluconv will write some information to standard output.
- 2 High level of output. rfluconv will write detailed information to standard output.

rfluconv expects interactive user input after invokation, as described in Sect. 7.2.4.

7.2.2 Input Files

The following input files are read by rfluconv:

- A grid file in RocfluMP ASCII or binary format.
- A flow solution file in RocfluMP ASCII or binary format.
- A dimension file.
- A boundary condition file.

7.2.3 Output Files

The following output files are written by rfluconv:

- A grid file in RocfluMP ASCII or binary format.
- A flow solution file in RocfluMP ASCII or binary format.
- A surface-grid file for TETMESH or YAMS.
- A version file called `rfluconv.vrs`. It contains the version number and date of the executable. Successive runs append to the version file.

7.2.4 Interactive Input

The interactive input after invokation is self-explanatory and will not be described here.

7.3 rfluextr

7.3.1 Invocation

rfluextr is invoked by typing

```
rfluextr -c <casename> -s <stamp> -v <verbosity>
```

The following command-line arguments are read by rfluextr:

`<casename>` A character string used to label the input and output files.

`<stamp>` A variable indicating the iteration or time from which the grid and solution files are to be read.

`<verbosity>` An integer indicating the desired verbosity level of rfluconv. The verbosity level can take the following values:

0 No output. rfluconv will not write any information to standard output.

1 Low level of output. rfluconv will write some information to standard output.

2 High level of output. rfluconv will write detailed information to standard output.

7.3.2 Input Files

The following input files are read by rfluextr:

- A grid file in RocfluMP ASCII or binary format for each region.
- A solution file in RocfluMP ASCII or binary format for each region.
- A dimension file for each region.
- A boundary condition file.
- A region mapping file.

7.3.3 Output Files

The following output files are written by rfluextr:

- An ASCII file called `<casename>.lin` containing the data extracted along the specified line.
- A version file called `rfluextr.vrs`. It contains the version number and date of the executable. Successive runs append to the version file.

7.3.4 Interactive Input

The interactive input after invokation is self-explanatory and will not be described here.

7.4 rfluinit

7.4.1 Invocation

rfluinit is invoked by typing

```
rfluinit -c <casename> -v <verbosity>
```

The command-line arguments read by rfluinit are:

`<casename>` A character string used to label the input and output files.

<verbosity> An integer indicating the desired verbosity level of rfluinit. The verbosity level can take the following values:

0 No output. rfluinit will not write any information to standard output.

1 Low level of output. rfluinit will write some information to standard output.

2 High level of output. rfluinit will write detailed information to standard output.

7.4.2 Input Files

The following input files are read by rfluinit:

- An input file called <casename>.inp.
- A grid file in RocfluMP format.
- A boundary-condition file. The name of the file is <casename>.bc.
- A dimension file.
- A restart-information file.

7.4.3 Output Files

The following input files are written by rfluinit:

- A flow solution file in rfluinit format.
- A version file called rfluinit.vrs. It contains the version number and date of the executable. Successive runs append to the version file.

7.5 rflumap

7.5.1 Invocation

rflumap is invoked by typing:

```
rflumap -c <casename> -m <mode> -p <nprocs> -r <nregions> -v <verbosity>
```

The following command-line arguments are read by rflumap:

<casename> A character string used to label the input and output files.

<mode> An integer indicating the mode of invokation. The mode can take the following values:

1 Initial mode. rflumap will only create a mapping file.

2 Final mode. rflumap will read an existing mapping file and create an input file for Rocin.

<nprocs> An integer indicating the number of processes. The number of processes must be less or equal to the number of regions.

<nregions> An integer indicating the number of regions. The number of regions must be greater or equal to the number of processes.

<verbosity> An integer indicating the desired verbosity level of rflumap. The verbosity level can take the following values:

0 No output. rflumap will not write any information to standard output.

1 Low level of output. rflumap will write some information to standard output.

2 High level of output. rflumap will write detailed information to standard output.

7.5.2 Input Files

rflumap does not read any input files.

7.5.3 Output Files

rflumap writes the region mapping file. For coupled runs, rflumap also produces the input file for Rocin.

7.6 rflump

7.6.1 Invocation

rflump is an MPI code and hence it's invokation is dependent on the type of machine and may also be dependent on the MPI distribution. For typical MPI distributions, rflump is invoked by typing

```
mpirun -np <n> rflump -c <casename> -v <verbosity>
```

where <n> is the number of processes.

The command-line arguments read by rflump are:

<casename> A character string used to label the input and output files.

<verbosity> An integer indicating the desired verbosity level of rflump. The verbosity level can take the following values:

0 No output. rflump will not write any information to standard output.

1 Low level of output. rflump will write some information to standard output.

2 High level of output. rflump will write detailed information to standard output.

It is important to be aware of the automatic restart capability of rflump. When flow-solution files are written, rflump writes the iteration number or time stamp into the so-called restart-information file. Whenever rflump is invoked, it checks for the existence of the restart-information file. If the restart-information file exists, rflump reads the last iteration number or time stamp at which flow-solution files were written. rflump reads in the flow-solution files corresponding to that iteration number or time stamp and starts the computation. Therefore, successive invokations of the above commands lead to different behaviour of rflump! This is done to simplify the execution of rflump within batch jobs.

7.6.2 Input Files

The following input files are read by rflump:

- An input file called `<casename>.inp`.
- A grid file in RocfluMP format.
- A flow-solution file in RocfluMP format.
- A boundary-condition file. The name of the file is `<casename>.bc`.
- A dimension file.
- A restart-information file.

7.6.3 Output Files

The following output files are written by rflump:

- A grid file in RocfluMP format for each region if grid motion is active.
- A flow solution file in RocfluMP format for each region.
- A dimension file for each region if the flow is unsteady and grid motion is active.
- A restart-information file.
- A probe file for each probe.
- A convergence file.

- A mass-conservation check file if grid motion is active and rflump is not run within GENx.
- A version file called `rflump.vrs`. It contains the version number and date of the executable. Successive runs append to the version file.

7.6.4 Profiling and Performance Analysis Guidelines

To profile RocfluMP, it should be compiled with the Rocprof library as described in Subsection 3.2.2. Important routines in RocfluMP will be instrumented automatically. If additional routines are to be included in the instrumentation, the source code must be edited accordingly. The execution of RocfluMP is unchanged by the additional instrumentation. On completing execution, dedicated output files are written by the Rocprof library for analysis with the Profane tool.

The performance of RocfluMP is strongly affected by many factors. If the goal of the analysis is to determine the outright speed of the code as a whole or selected portions of it, the following guidelines should be considered:

1. Create a balanced data set for parallel runs. In other words, the dimensions of each region should be identical or nearly identical. It is important to note that this applies not only to real cells, but also to virtual cells because the latter determine the amount of communication between the regions.
2. Deactivate the printing and writing of output, i.e.,
   - Deactivate printing to screen by setting the verbosity level to zero.
   - Deactivate checking by setting the checking level to zero.
   - Deactivate the computation of forces. (This is particularly important for parallel runs since the computation of forces and moments is likely to introduce a load imbalance and requires additional communication.)
   - Do not specify the extraction of data through probes. (This is particularly important for parallel runs since the writing of probe data is likely to introduce a load imbalance.)
3. Choose an appropriate number of regions for a given number of processors and machine configuration. Many parallel computers are divided into so-called nodes, each of which may hold a given number of processors. For example, assume a given machine contains 16 processors on each node. If a run is to be made on 1024 processors, 64 nodes of this machine would be required. In measuring outright performance of the code, however, it is not a good idea to use all the processors on a given node. This is because the operating system will require some resources also and then slow down the computation of the region(s) assigned to that processor. It is therefore recommended that one processor on each dataset be left unused on such machines. Hence it is necessary to

request more nodes or to create datasets which are multiples of $N_p - 1$, where N_p is the number of processors per node.

7.7 rflupart

7.7.1 Invocation

For serial computations, rflupart is invoked by typing

```
rflupart -c <casename> -v <verbosity>
```

The command-line arguments read by rflupart are:

`<casename>` A character string used to label the input and output files.

`<verbosity>` An integer indicating the desired verbosity level of rflupart. The verbosity level can take the following values:

`0` No output. rflupart will not write any information to standard output.

`1` Low level of output. rflupart will write some information to standard output.

`2` High level of output. rflupart will write detailed information to standard output.

7.7.2 Input Files

The following input files are read by rflupart:

- An input file called `<casename>.inp`.
- A grid file. rflupart supports the following formats:
  - CENTAUR format. The CENTAUR grid file may be in ASCII or binary format. The file in ASCII format is called `<casename>.hyb.asc`, and the file in binary format is called `<casename>.hyb.bin`.
  - VGRIDns format. The file is called `<casename>.cgosg`.
  - MESH3D format. The file is called `<casename>.m3d`.
  - TETMESH format. The file is called `<casename>.noboite`.
  - Cobalt format. The file is called `<casename>.cgr`.
  - GAMBIT format. The file is called `<casename>.neu`.
- A boundary condition file called `<casename>.bc`.

- A mapping file called `<casename>.map` specifying how many processors are to be used for parallel computations, and how the regions are mapped to the processors. This file is only required for parallel computations. If the mapping file does not exist, rflupart assumes that a serial computation will be made.

- A file specifying the mapping between the boundary patches used during grid generation and how these patches translate to the patches to be used in RocfluMP. This file is only needed if a VGRIDns, MESH3D, Cobalt, or StarCD grid file is read. The file is called `<casename>.vgi`, `<casename>.mgi`, or `<casename>.cgi`, depending on whether the grid file is in VGRIDns, MESH3D, Cobalt, or StarCD format.

7.7.3 Output Files

The following output files are written by rflupart:

- A grid file in RocfluMP format for each region. For parallel computations, the number of grid files written out depends on the number of regions specified in the mapping file.

- A dimension file for each region. For parallel computations, the number of dimension files written out depends on the number of regions specified in the mapping file.

- A communication map file for each region. For parallel computations, the number of communication map files written out depends on the number of regions specified in the mapping file.

- A version file called `rflupart.vrs`. It contains the version number and date of the executable. Successive runs append to the version file.

7.7.4 Batch-Job Submission Guidelines

Because rflupart is a serial code, the partitioning of large grids can be memory-intensive. The memory required by rflupart is approximately independent of the number of partitions provided that the number of partitions is not too small. This is because rflupart stores only one partition at the same time as the serial grid.

To aid in the preparation of batch jobs, Fig. 7.1 summarizes the memory required to partition a grid of a given number of cells.

7.8 rflupick

When running rflump in parallel, the output from rflupick can only be used if rflupost is instructed not to merge regions.

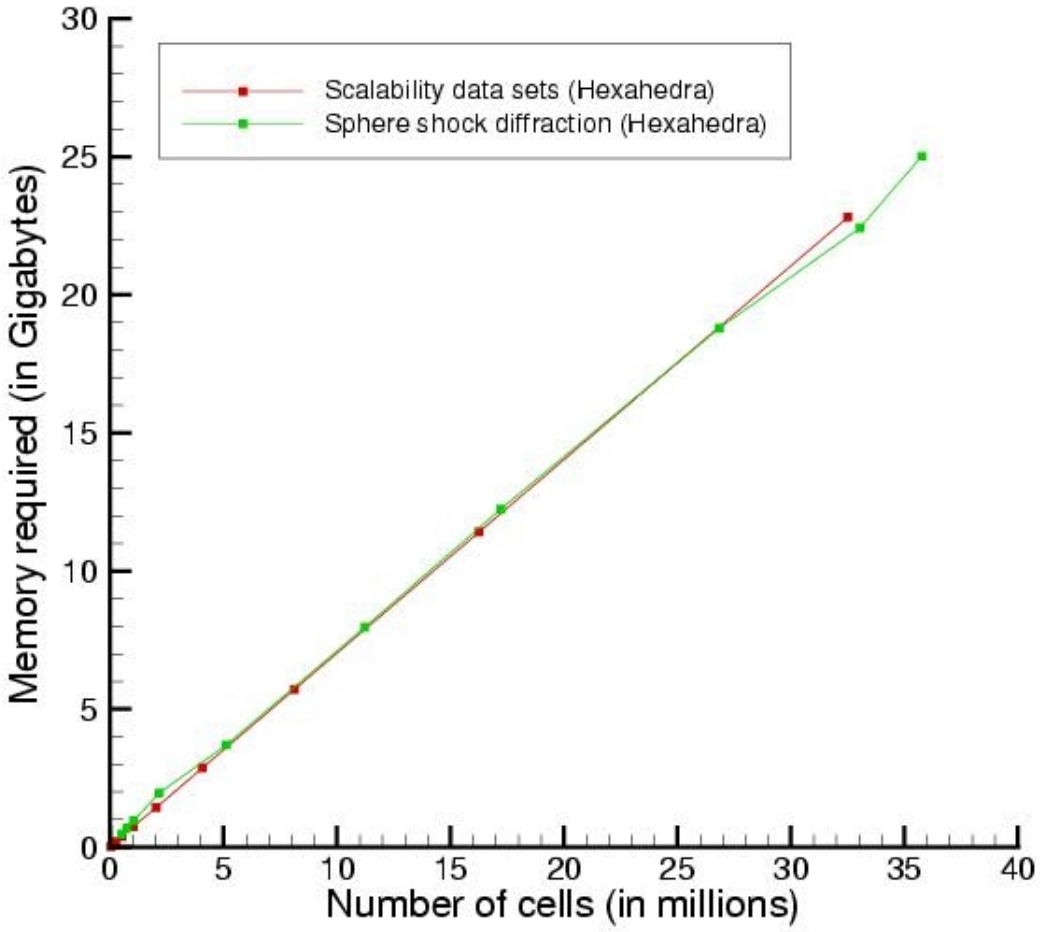

Figure 7.1: Memory requirements of rflupart to partition a grid of given number of cells.

7.8.1 Invocation

rflupick is invoked by typing:

```
rflupick -c <casename> -s <stamp> -v <verbosity>
```

The following command-line arguments are read by rflupick:

`<casename>` A character string used to label the input and output files.

`<stamp>` A variable indicating the iteration or time from which the grid and solution files are to be read.

`<verbosity>` An integer indicating the desired verbosity level of rflupick. The verbosity level can take the following values:

- `0` No output. rflupick will not write any information to standard output.
- `1` Low level of output. rflupick will write some information to standard output.
- `2` High level of output. rflupick will write detailed information to standard output.

rflupick expects interactive user input after invokation, as described in Sect. 7.8.4.

7.8.2 Input Files

The following input files are read by rflupick:

- A grid file in RocfluMP ASCII or binary format.
- A flow solution file in RocfluMP ASCII or binary format.
- A dimension file.
- A boundary condition file.
- A mapping file.

7.8.3 Output Files

The following output files are written by rflupick:

- A postprocessor info file.
- A version file called `rflupick.vrs`. It contains the version number and date of the executable. Successive runs append to the version file.

7.8.4 Interactive Input

After invokation, rflupick presents the following screen output:

```
Picking regions manually...
  Enter information on regions:
    a - Pick all regions
    s - Pick some regions
    n - Pick no regions
  Enter information type:
```

The input determines which regions will be postprocessed by rflupost.

Picking all regions. Assuming that the user enters option a, rflupick will loop over all regions, and for each region present the following screen output (the following output assumes that the region index is 00001):

```
Picking special cells...
  Global region: 00001
    Enter information on special cells:
      b - cell adjacent to boundary face
      c - single cell
      f - cells adjacent to interior face
      s - stencil members
      v - cells meeting at vertex
      q - quit
  Enter information type:
```

This output is self-explanatory. Once special cells were picked, rflupick will present the following screen output:

```
Picking special faces...
  Global region: 00001
    Enter information on special faces:
      b - boundary face
      i - interior face
      q - quit
  Enter information type:
```

Once again, the output is self-explanatory. Once special faces were picked, rflupick will repeat the same procedure for the next region.

Picking some regions. Assuming that the user enters option `s`, rflupick will present the following screen output:

```
Enter global region index (< 1 to exit):
```

rflupick will repeat the same request until the user enters an integer smaller than unity. Once some regions were picked, rflupick will loop over the picked regions and present the same output as described above.

7.9 rflupost

7.9.1 Invocation

For serial computations, rflupost is invoked by typing

```
rflupost -c <casename> -s <stamp> -v <verbosity>
```

The following command-line arguments are read by rflupost:

`<casename>` A character string used to label the input and output files.

`<stamp>` A variable indicating the iteration or time from which the grid and solution files are to be read.

`<verbosity>` An integer indicating the desired verbosity level of rflupost. The verbosity level can take the following values:

- `0` No output. rflupost will not write any information to standard output.
- `1` Low level of output. rflupost will write some information to standard output.
- `2` High level of output. rflupost will write detailed information to standard output.

7.9.2 Input Files

The following input files are read by rflupost:

- A grid file in RocfluMP format.
- A flow solution file in RocfluMP format.
- A dimension file.
- A boundary condition file.
- A file generated by rflupick detailing which regions are to be postprocessed and whether individual cells are to be visualized. If this file does not exist, all regions are postprocessed.

7.9.3 Output Files

The following output files are written by rflupost:

- A file in binary TECPLOT format called `<casename>.plt`.
- A version file called `rflupost.vrs`. It contains the version number and date of the executable. Successive runs append to the version file.

Chapter 8

Example Cases

This chapter illustrates the execution of rflumap, rflupart, rfluinit, rflump, and rflupost for several example cases. For the sake of brevity, the output produced by these program is not shown.

8.1 Shocktube

The flow in a shocktube is used to test the shock-capturing capabilities of the spatial discretization and the accuracy of the temporal discretization. The CVS repository contains coarse and fine grids for the first and second cases specified by Sod [13]. The grid is not shown on account of the simple geometry.

The following files are required for this case:

```
shocktube.bc
shocktube.inp
shocktube.hyb.bin
```

The `GRIDSRC` variable in the `FORMATS` section of the `shocktube.inp` file indicates that the original grid file is in CENTAUR format.

By typing

```
rflupart -c shocktube -v 2
```

the partitioner runs and writes the following output files:

```
shocktube.dim
shocktube.grd_00000
```

The initialization module is executed by typing

```
rfluinit -c shocktube -v 2
```

which produces the file

```
shocktube.mixt.cv_00000_0.00000E+00
```

The solver may be run by typing

```
rflump -c shocktube -v 2
```

With the settings contained in the files from the CVS repository, rflump will take as many time steps as needed to reach a physical time of $t = 1.0 \cdot 10^{-3}$ s.

The results may be visualized with TECPLOT. The postprocessor is used to interpolate the solution variables from cell centroids to vertices and to write the TECPLOT data file by typing

```
rflupost -c shocktube -s 1.0E-3 -v 2
```

The resulting data file may be read into TECPLOT by typing

```
tecplot shocktube_00000_1.00000E-03.plt
```

The contours of density are depicted in Fig. 8.1
The CVS repository contains a TECPLOT macro file called `lineplots.mcr` which can be used to extract line plots of density, velocity, and pressure along the x-axis as shown in Fig. 8.2.

8.2 GAMM Bump

The transonic flow over a circular arc bump in a straight-walled channel is often used to test the inflow and outflow boundary conditions and the shock-capturing capabilities of the spatial discretization. This case will be used to illustrate both serial and parallel computations.

The grid used in this computation is shown in Fig. 8.3. The green surface indicates the inflow boundary. The outflow boundary is hidden from view in this figure.

The following files are required for this case:

```
gamm8.bc
gamm8.inp
gamm8.hyb.bin
```

The `GRIDSRC` variable in the `FORMATS` section of the `gamm8.inp` file indicates that the original grid file is in CENTAUR format.

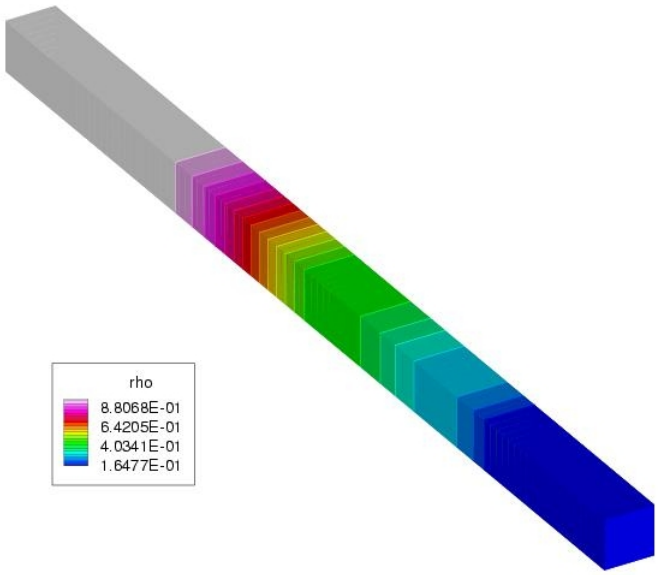

Figure 8.1: Density contours for shock-tube computation at $t = 1.0 \cdot 10^{-3}$ s with initial conditions given by Sod's first case.

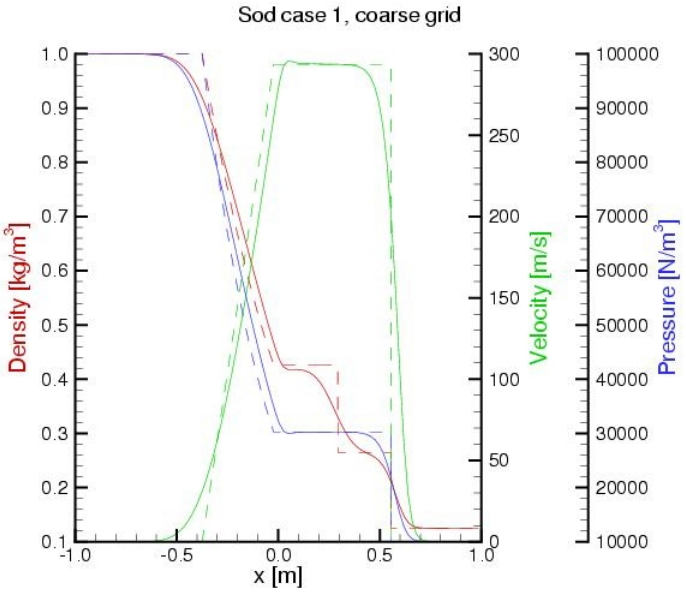

Figure 8.2: Line plots for shock-tube computation at $t = 1.0 \cdot 10^{-3}$ s with initial conditions given by Sod's first case.

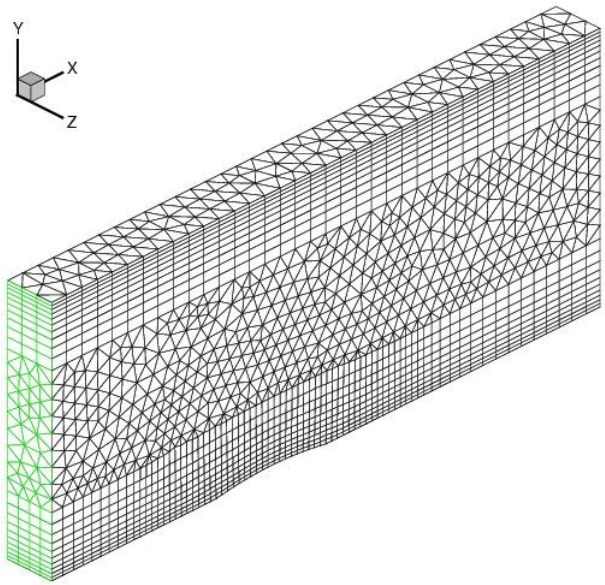

Figure 8.3: Grid used for GAMM bump computation.

8.2.1 Serial Computation

By typing

```
rflupart -c gamm8 -v 2
```

the partitioner runs and writes the following output files:

```
gamm8.cmp_00000
gamm8.dim_00000
gamm8.grd_00000
gamm8.rnm_00000
```

By typing

```
rfluinit -c gamm8 -v 2
```

the partitioner runs and writes the following output files:

```
gamm8.mixt.cv_00000_000000
```

The solver may be run by typing

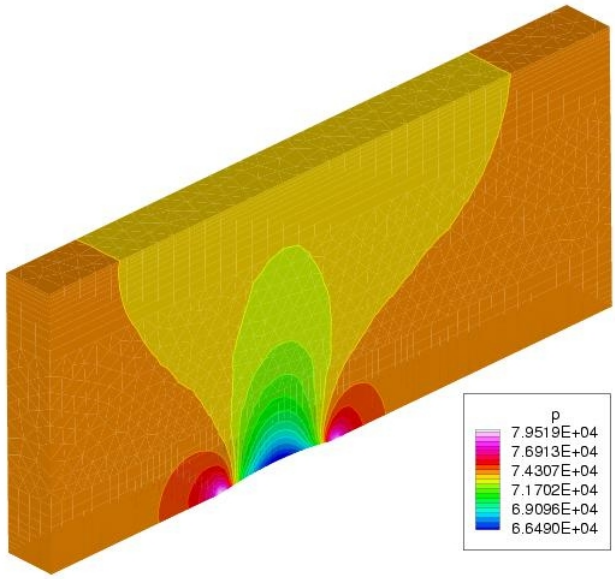

Figure 8.4: Static pressure contours for transonic GAMM bump computation.

```
rflump -c gamm8 -v 2
```

With the settings contained in the files from the CVS repository, rflump will take 500 steps to reach the convergence tolerance.

The results may be visualized with TECPLOT. The postprocessor is used to interpolate the solution variables from cell centroids to vertices and to write the TECPLOT data file by typing

```
rflupost -c gamm8 -s 500 -v 2
```

The resulting data file may be read into TECPLOT by typing

```
tecplot gamm8_00000_000500.plt
```

The contours of static pressure are depicted in Fig. 8.4

8.2.2 Parallel Computation

The first step in preparing for a parallel computation is to decide how many regions should be used. For this example, we will use five regions and hence type

```
rflumap -c gamm8 -m 1 -p 5 -r 5 -v 2
```

which leads to the region-mapping file

```
gamm8.map
```

To partition the grid files, the partitioner is invoked by typing

```
rflupart -c gamm8 -v 2
```

which produces the following files:

```
gamm8.cmp_00001  gamm8.com_00001  gamm8.dim_00001  gamm8.grd_00001
gamm8.cmp_00002  gamm8.com_00002  gamm8.dim_00002  gamm8.grd_00002
gamm8.cmp_00003  gamm8.com_00003  gamm8.dim_00003  gamm8.grd_00003
gamm8.cmp_00004  gamm8.com_00004  gamm8.dim_00004  gamm8.grd_00004
gamm8.cmp_00005  gamm8.com_00005  gamm8.dim_00005  gamm8.grd_00005

gamm8.rnm_00001
gamm8.rnm_00002
gamm8.rnm_00003
gamm8.rnm_00004
gamm8.rnm_00005
```

To initialize the solution, the initializer is invoked by typing

```
rfluinit -c gamm8 -v 2
```

which produces the following files:

```
gamm8.mixt.cv_00001_000000
gamm8.mixt.cv_00002_000000
gamm8.mixt.cv_00003_000000
gamm8.mixt.cv_00004_000000
gamm8.mixt.cv_00005_000000
```

The parallel computation is initiated by

```
mpirun -np 5 rflump -c gamm8 -v 2
```

The precise command line may vary depending on which computer is used

The results of the computation will not be shown again. Instead, the capability of rflupost to allow the user to visualize the partitioned geometry will be demonstrated. Generate the TECPLOT file through

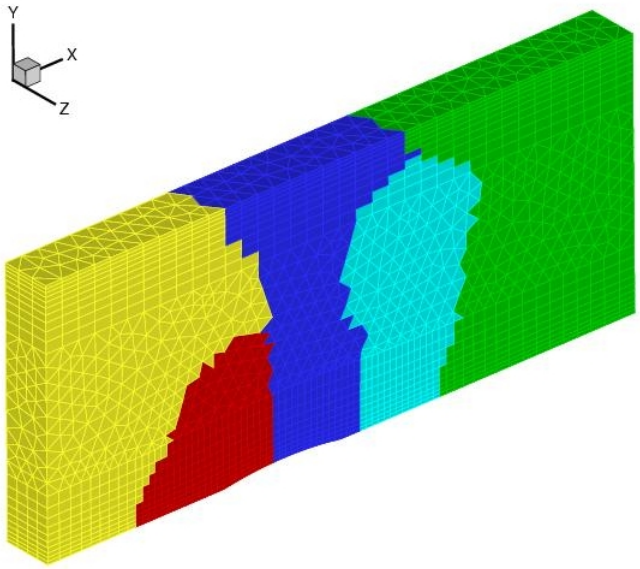

Figure 8.5: Partitioned grid for parallel GAMM bump computation.

```
rflupost -c gamm8 -s 500 -v 2
```

Figure 8.5 depicts the five regions generated for this particular run. More importantly, rflupost can be used to visualize individual regions, the interregion faces, the virtual faces associated with particular regions and boundary patches, as well as the virtual cells of any region. Some of these capabilities are illustrated in Fig. 8.6.

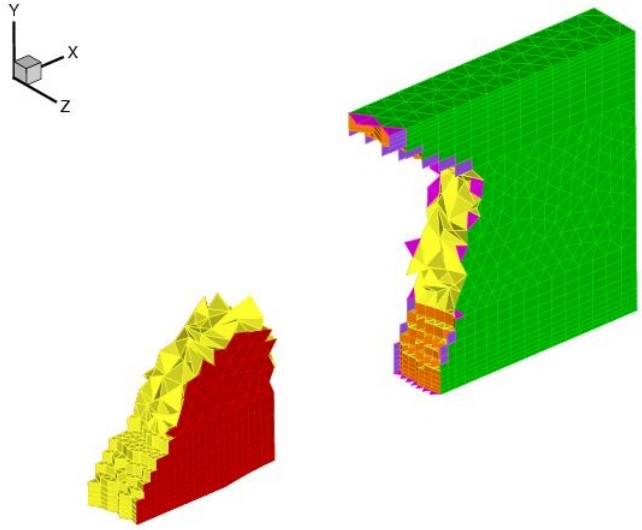

Figure 8.6: Partitioned grid for parallel GAMM bump computation, showing interregion faces as well as virtual boundary faces.

Chapter 9

Visualization

This chapter describes how results obtained with rflump can be visualized. The focus is on presenting information which will be helpful in visualizing results obtained specifically with rflump. Consult the appropriate manuals for information on how the visualization tools themselves should be used.

This chapter assumes that the user has executed rflupost already and thereby produced one or more files for visualization. The behavior of rflupost is influenced by the values assigned to the keywords in the POST Section of the input file.

rflupost writes output files for TECPLOT and ENSIGHT. TECPLOT is best used to visualize small to medium-sized cases and to aid in debugging, whereas ENSIGHT is best used to visualize large cases.

9.1 Plotting Variables

rflupost can be instructed to write so-called plotting variables to the output files. Plotting variables are variables derived from the solution, usually through non-trivial additional computations in rflupost, to gain insight into particular features of the flowfield.

At present, rflupost writes plotting variables only to TECPLOT files.

9.1.1 Visualization of Discontinuities

The computation of plotting variables which allow the visualization of discontinuities is activated by the keyword DISCFLAG in the POST Section of the input file. Three variables are computed to allow the visualization of discontinuities:

1. The magnitude of the density gradient. This variable is used to make Schlieren pictures, in which highly localized regions of large density gradients indicate shock waves and contact discontinuities. For an example of a Schlieren picture, see Fig. 4.3.

2. The Laplacian of the density. This variable is used to make Shadowgraphs.

At present, this variable is set to zero by rflupost.

3. An indicator function of the normalized density,

$$I_i = \left\lfloor n_f \left(\frac{\rho_i - \min_i \rho_i}{\max_i \rho_i - \min_i \rho_i} \right) + \frac{1}{2} \right\rfloor \bmod 2, \tag{9.1}$$

where i represents the index of a given cell and n_f is the number of fringes. The indicator function can be used to plot interferograms.

At present, rflupost computes correctly the indicator function only if `MERGEFLAG=1` in the `POST` Section of the input file.

If `DISCFLAG=1`, rflupost writes three plotting variables to the visualization files, i.e., the magnitude of the density gradient, the Laplacian of the density, and I_i given by Eq. (9.1).

9.1.2 Visualization and Identification of Vortical Structures

The computation of plotting variables which allow the visualization of vortical structures is activated by the keywords `VORTFLAG` and `COREFLAG` in the `POST` Section of the input file.

The keyword `VORTFLAG` activates the computation of the Cartesian components of the vorticity field,

$$\omega_x = \frac{\partial w}{\partial y} - \frac{\partial v}{\partial z}, \tag{9.2}$$

$$\omega_y = \frac{\partial u}{\partial z} - \frac{\partial w}{\partial x}, \tag{9.3}$$

$$\omega_z = \frac{\partial v}{\partial x} - \frac{\partial u}{\partial y}. \tag{9.4}$$

If `VORTFLAG=1`, rflupost writes three plotting variables to the visualization files, i.e., ω_x, ω_y, and ω_z.

The keyword `COREFLAG` activates the computation of variables which allow identification of vortices. These variables are used in three criteria for vortex identification, which are described and compared in detail by Chakraborty et al. [5]. The criteria are based on the velocity gradient tensor $\nabla \mathbf{v}$ and its symmetric and antisymmetric components

$$\mathbf{S} = \frac{1}{2} \left(\nabla \mathbf{v} + (\nabla \mathbf{v})^t \right), \tag{9.5}$$

$$\mathbf{Q} = \frac{1}{2} \left(\nabla \mathbf{v} - (\nabla \mathbf{v})^t \right). \tag{9.6}$$

The criteria are:

1. *Q Criterion:* Vortices are identified by regions with $Q > 0$, where Q is the second invariant of $\nabla\mathbf{v}$, i.e.,
$$Q = \frac{1}{2}\left(\|\mathbf{Q}\|^2 - \|\mathbf{S}\|^2\right). \tag{9.7}$$

2. λ_2 *Criterion:* Vortices are identified by regions in which $\lambda_2 < 0$, where λ_2 is the second eigenvalue of $\mathbf{S}^2 + \mathbf{Q}^2$, assuming an ordering of $\lambda_1 \geqslant \lambda_2 \geqslant \lambda_3$.

3. *Swirling-Strength Criterion:* The eigenvalues of the velocity-gradient tensor are assumed to be given by λ_r and $\lambda_{cr} \pm \mathrm{i}\,\lambda_{ci}$. Then a vortex is identified by regions in which $\lambda_{ci} \geqslant \varepsilon$ and $\lambda_{cr}/\lambda_{ci} \leqslant \delta$, where ε and δ are positive thresholds.

If `COREFLAG=1`, rflupost writes four plotting variables to the visualization files, i.e., Q given by Eq. (9.7), λ_2, and λ_{cr} and λ_{ci} from the swirling-strength criterion.

9.1.3 Visualization of Eulerian Particle Fields

The computation of plotting variables which allow the visualization of Eulerian fields of the particle data is activated by the keyword `PEULFLAG` in the `POST` Section of the input file.

If `PEULFLAG=1`, rflupost writes eight plotting variables to the visualization files.

9.2 TECPLOT Output

The TECPLOT files produced by rflupost are in binary format to reduce file size and loading time. The binary file are compatible across machines, i.e., files written on little-endian machines can be read on big-endian machines and vice versa. The files produced by rflupost make use of TECPLOT features which require version 10 or newer.

9.2.1 File Naming Convention

The name of the TECPLOT files produced by rflupost depends on whether a steady or unsteady flow is computed. The name of the TECPLOT file is:

- `<casename>_<iteration stamp>.plt` for steady flows, where `<iteration stamp>` is a six-digit string denoting the iteration number.

- `<casename>_<time stamp>.plt` for unsteady flows, where `<time stamp>` is a string of the form `n.nnnnnnnE+nn` denoting time.

It should be noted that the TECPLOT files produced for unsteady flows cannot be loaded into TECPLOT from the command-line because TECPLOT interprets the character `+` as combining two file names. Instead, these files need to be loaded into TECPLOT using the `File -> Load Data File(s)` option after having started TECPLOT from the command line.

Table 9.1: Names and meaning of solution variables written to TECPLOT files.

Variable	Meaning	Units
r	Density	kg/m^3
ru	x-component of momentum	$kg/(m^2\,s)$
rv	y-component of momentum	$kg/(m^2\,s)$
rw	z-component of momentum	$kg/(m^2\,s)$
rE	Total internal energy	J/m^3
p	Pressure	Pa
T	Temperature	K
a	Speed of sound	m/s

9.2.2 File Content

9.2.2.1 Solution Variables

rflupost writes all conserved and dependent variables into the TECPLOT files. In the TECPLOT file, the variables are denoted by dedicated names as listed in Table 9.1.

9.2.2.2 Plotting Variables

Plotting variables are labelled PVnn where nn is a two-digit integer starting with 01, 02, and so on. The meaning of a given plotting variable is dependent on the user input for the flags DISCFLAG, VORTFLAG, COREFLAG, and PEULFLAG. Each variable is numbered consecutively depending on whether the associated flag is active.

9.2.2.3 Patch-Coefficient Variables

9.2.3 Zone Naming Conventions

TECPLOT organizes data by zones. Each zone is assigned a name and may consist of either volume or surface data. Because TECPLOT allows each zone to be activated (displayed) or deactived (not displayed) separately, the separation of data into zones is useful in allowing detailed and selective investigation of data. For this reason, the TECPLOT files produced by rflupost make extensive use of zones. The following explains the zone naming conventions used by rflupost.

The zone naming convention adopted by rflupost is best explained with reference to Fig. 9.1. Data written to the TECPLOT file is split into volume and boundary patch data. Volume data is further split according to the cell type (tetrahedron, hexahedron, prism, or pyramid), and, for each cell type, according to the cell kind (actual or virtual cell). Boundary patch

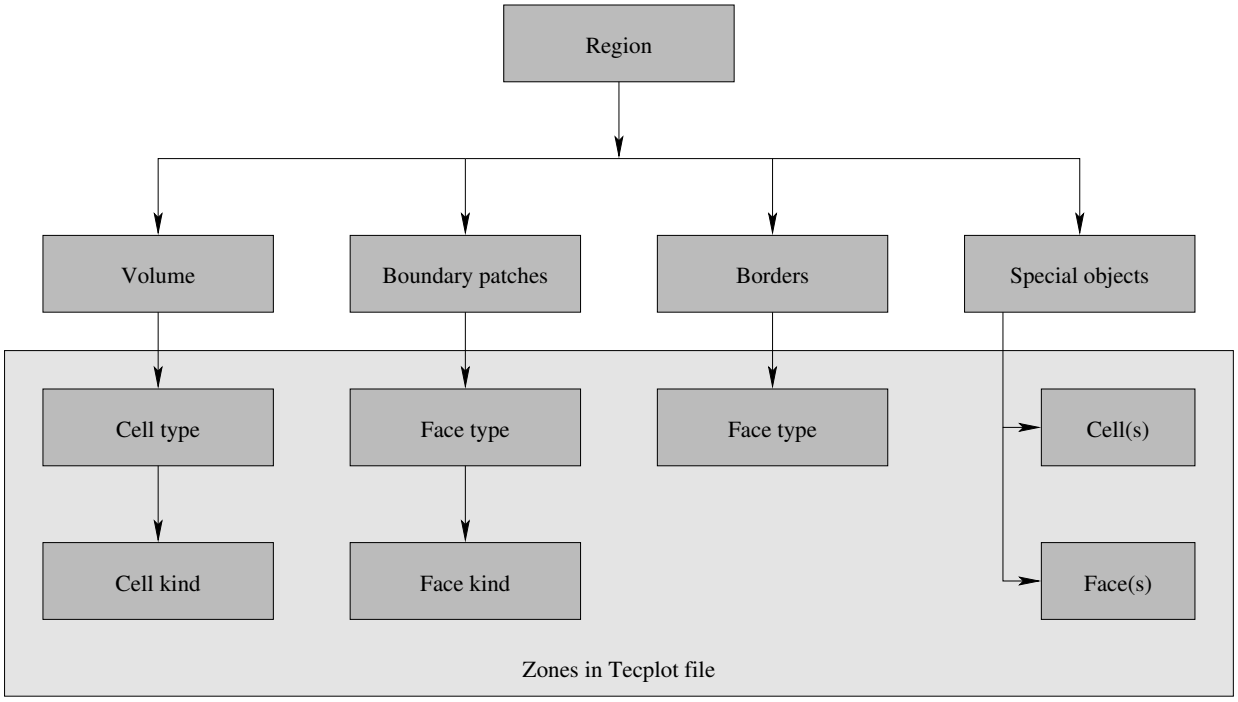

Figure 9.1: Illustration of zones written to TECPLOT files by rflupost.

data is further split according to the face type (triangular or quadrilateral), and, for each face type, according to the face kind (actual or virtual face).

9.2.3.1 Volume Zones

The names of volume zones are:

```
<cell type>-<cell kind>_<region index>
```

where

`<cell type>` is a four-letter string indicating the cell type:

- `TET` if the zone consists of tetrahedral cells.
- `HEX` if the zone consists of hexahedral cells.
- `PRI` if the zone consists of prismatic cells.
- `PYR` if the zone consists of pyramidal cells.

`<cell kind>` is a one-letter string indicating the cell kind:

- `A` if the zone consists of actual cells.

`V` if the zone consists of virtual cells.

`<region index>` is a five-digit string indicating the global region index.

Examples are shown in Fig. 9.2, which represents a screen dump of part of the zone style menu in TECPLOT. Zone number 1 contains the actual hexahedra of region 1 while zone number 23 contains the virtual prisms of region 2.

9.2.3.2 Boundary Patch Zones

The names of boundary patch zones are:

```
PAT_<patch index>_<face type>-<face kind>_<region index>
```

where

`<patch index>` is a three-digit string indicating the global patch index. It is important to note that this is a *global* patch index.

`<face type>` is a string indicating the face type:

`TRI` if the zone consists of triangular cells.

`QUAD` if the zone consists of quadrilateral cells.

`<face kind>` is a one-letter string indicating the face kind:

`A` if the zone consists of actual faces.

`V` if the zone consists of virtual faces.

`<region index>` is a five-digit string indicating the global region index.

Examples are shown in Fig. 9.2, which represents a screen dump of part of the zone style window in TECPLOT. Zone number 9 contains the actual quadrilateral faces on global patch number 1 in region 1. Zone number 13 contains the virtual triangular faces on global patch number 5 in region 1. Note once again that the patch numbers are *global* patch numbers. Global patch number 5 corresponds to local patch number 3 in region 1.

9.2.3.3 Border Face Zones

The names of border face zones are:

```
INT-<face type>_<region index>
```

where

Zone Style

Mesh | Contour | Vector | Scatter | Shade | Bounda

Zone Num	Zone Name	Zone Grp	Zone Show	Mesh Show	Mesh Type
1	HEX-A_00001	1	Yes	Yes	Overlay
2	HEX-V_00001	1	Yes	Yes	Overlay
3	PRI-A_00001	1	Yes	Yes	Overlay
4	PRI-V_00001	1	Yes	Yes	Overlay
5	C_00001247_00001	1	Yes	Yes	Overlay
6	F_00000871_PAT_000_00001	1	Yes	Yes	Overlay
7	F_00000013_PAT_001_00001	1	Yes	Yes	Overlay
8	INT_QUAD_00001	1	Yes	Yes	Overlay
9	PAT_001_QUAD-A_00001	1	Yes	Yes	Overlay
10	PAT_001_QUAD-V_00001	1	Yes	Yes	Overlay
11	PAT_002_QUAD-A_00001	1	Yes	Yes	Overlay
12	PAT_005_TRI-A_00001	1	Yes	Yes	Overlay
13	PAT_005_TRI-V_00001	1	Yes	Yes	Overlay
14	PAT_005_QUAD-A_00001	1	Yes	Yes	Overlay
15	PAT_005_QUAD-V_00001	1	Yes	Yes	Overlay
16	PAT_006_TRI-A_00001	1	Yes	Yes	Overlay
17	PAT_006_TRI-V_00001	1	Yes	Yes	Overlay
18	PAT_006_QUAD-A_00001	1	Yes	Yes	Overlay
19	PAT_006_QUAD-V_00001	1	Yes	Yes	Overlay
20	HEX-A_00002	1	Yes	Yes	Overlay
21	HEX-V_00002	1	Yes	Yes	Overlay
22	PRI-A_00002	1	Yes	Yes	Overlay
23	PRI-V_00002	1	Yes	Yes	Overlay
24	INT_QUAD_00002	1	Yes	Yes	Overlay
25	PAT_001_QUAD-A_00002	1	Yes	Yes	Overlay
26	PAT_001_QUAD-V_00002	1	Yes	Yes	Overlay
27	PAT_005_TRI-A_00002	1	Yes	Yes	Overlay
28	PAT_005_TRI-V_00002	1	Yes	Yes	Overlay
29	PAT_005_QUAD-A_00002	1	Yes	Yes	Overlay
30	PAT_005_QUAD-V_00002	1	Yes	Yes	Overlay
31	PAT_006_TRI-A_00002	1	Yes	Yes	Overlay
32	PAT_006_TRI-V_00002	1	Yes	Yes	Overlay
33	PAT_006_QUAD-A_00002	1	Yes	Yes	Overlay
34	PAT_006_QUAD-V_00002	1	Yes	Yes	Overlay
35	HEX-A_00003	1	Yes	Yes	Overlay
36	HEX-V_00003	1	Yes	Yes	Overlay
37	PRI-A_00003	1	Yes	Yes	Overlay
38	PRI-V_00003	1	Yes	Yes	Overlay
39	INT_QUAD_00003	1	Yes	Yes	Overlay

Figure 9.2: Screen dump of part of zone style window in TECPLOT illustrating zone naming convention used in rflupost.

`<face type>` is a string indicating the face type:

`TRI` if the zone consists of triangular cells.

`QUAD` if the zone consists of quadrilateral cells.

`<region index>` is a five-digit string indicating the global region index.

An example is shown in Fig. 9.2, which represents a screen dump of part of the zone style window in TECPLOT. Zone number 8 contains the quadrilateral border faces of region 1. In this particular case, the zone contains only quadrilateral faces.

9.2.3.4 Special Cell Zones

The names of special cell zones are:

```
C_<cell index>_<region index>
```

where

`<cell index>` is an eight-digit string indicating the cell index.

`<region index>` is a five-digit string indicating the global region index.

Note that the names of special cell zones do not distinguish between cell types and cell kinds. An example is shown in Fig. 9.2, which represents a screen dump of part of the zone style window in TECPLOT. Zone number 5 contains cell number 1247 of region 1.

9.2.3.5 Special Face Zones

The names of special cell zones are:

```
F_<face index>_PAT_<patch index>_<region index>
```

where

`<face index>` is an eight-digit string indicating the face index.

`<patch index>` is a three-digit string indicating the global patch index. It is important to note that this is a *global* patch index. The patch number is zero if the face is an interior face.

`<region index>` is a five-digit string indicating the global region index.

Note that the names of special face zones do not distinguish between faces types. Examples are shown in Fig. 9.2, which represents a screen dump of part of the zone style menu in TECPLOT. Zone number 5 contains cell number 1247 of region 1. Zone number 7 contains face number 13 of global patch 1 of region 1.

Table 9.2: Names and meaning of `variable` string in ENSIGHT file name.

`variable`	Meaning	Units
`r`	Density	kg/m^3
`rv`	Momentum vector	$kg/(m^2\,s)$
`rE`	Total internal energy	J/m^3
`p`	Pressure	Pa
`T`	Temperature	K
`a`	Speed of sound	m/s

9.3 ENSIGHT Output

The ENSIGHT files produced by rflupost are in binary format to reduce file size and loading time. The binary file are compatible across machines, i.e., files written on little-endian machines can be read on big-endian machines and vice versa. The files produced by rflupost make use of ENSIGHT features which require version 8 or newer. To allow visualization of large datasets, rflupost can write files suitable for use with the client-server version of ENSIGHT Gold. The number of servers is specified by the keyword `NSERVERS` in the `POST` Section.

9.3.1 File Naming Convention

The name of the ENSIGHT files produced by rflupost depends on whether a steady or unsteady flow is computed. One variable, which may be a scalar or vector, is written to each file. The name of an ENSIGHT file is:

- `<casename>.<variable>_<server index>_<iteration stamp>` for steady flows, where `<iteration stamp>` is a six-digit string denoting the iteration number.
- `<casename>.<variable>_<server index>_<time stamp>` for unsteady flows, where `<time stamp>` is a string of the form `n.nnnnnnnE+nn` denoting time.

For steady or unsteady flows, `<variable>` is a character string denoting the variable contained in the file and `<server index>` is the index of the server which will read the file. The meaning of the `<variable>` character string is as listed in Table 9.2.

9.3.2 Part Naming Conventions

ENSIGHT organizes data by parts. Each part is assigned a name and may consist of either volume or surface data. Because ENSIGHT allows each part to be activated (displayed) or deactived (not displayed) separately, the separation of data into parts is useful in allowing

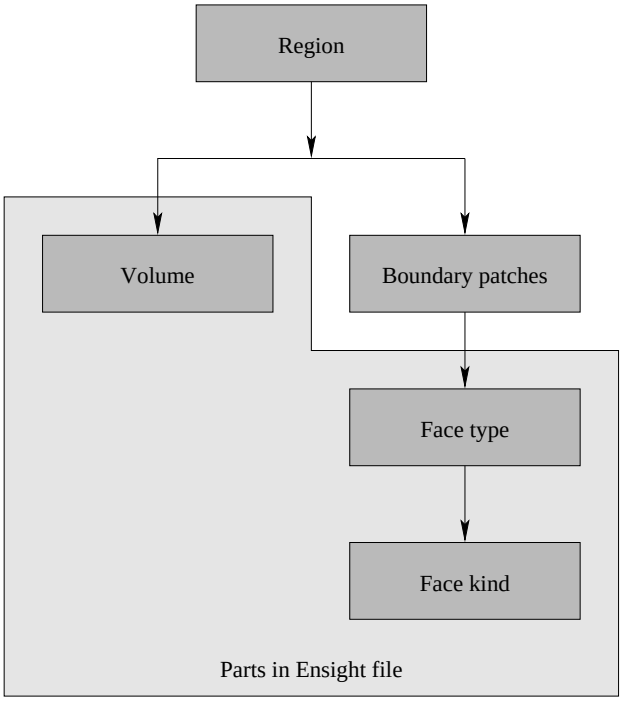

Figure 9.3: Illustration of parts written to ENSIGHT files by rflupost.

detailed and selective investigation of data. For this reason, the ENSIGHT files produced by rflupost make extensive use of parts. The following explains the part naming conventions used by rflupost.

The part naming convention adopted by rflupost is best explained with reference to Fig. 9.3. Data written to the ENSIGHT file is split into volume and boundary patch data. Volume data is further split according to the cell type (tetrahedron, hexahedron, prism, or pyramid), and, for each cell type, according to the cell kind (actual or virtual cell). Boundary patch data is further split according to the face type (triangular or quadrilateral), and, for each face type, according to the face kind (actual or virtual face).

9.3.2.1 Volume Parts

The names of volume parts are:

```
VOL_<region index>
```

where `<region index>` is a five-digit string indicating the global region index.

Examples are shown in Fig. 9.4, which represents a screen dump of the data part loader window in ENSIGHT.

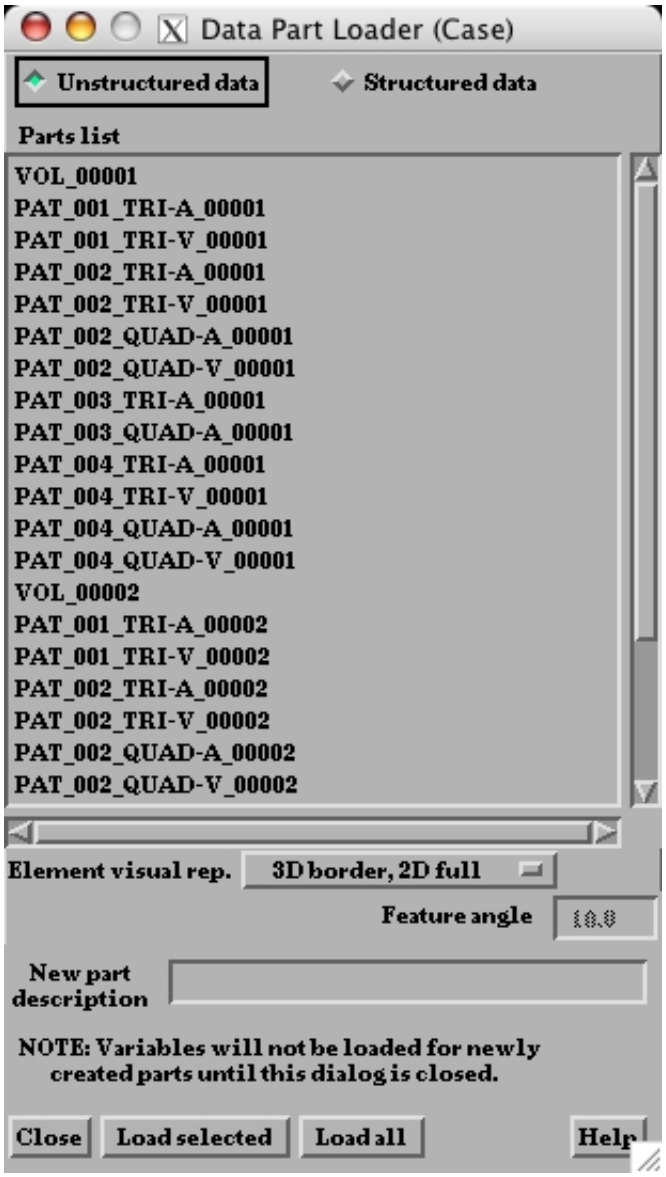

Figure 9.4: Screen dump of data part loader window in ENSIGHT illustrating part naming convention used in rflupost.

9.3.2.2 Boundary Patch Parts

The names of boundary patch parts are:

```
PAT_<patch index>_<face type>-<face kind>_<region index>
```

where

`<patch index>` is a three-digit string indicating the global patch index. It is important to note that this is a *global* patch index.

`<face type>` is a string indicating the face type:

`TRI` if the zone consists of triangular cells.

`QUAD` if the zone consists of quadrilateral cells.

`<face kind>` is a one-letter string indicating the face kind:

`A` if the zone consists of actual faces.

`V` if the zone consists of virtual faces.

`<region index>` is a five-digit string indicating the global region index.

Examples are shown in Fig. 9.4, which represents a screen dump of the data part loader window in ENSIGHT.

Chapter 10

Troubleshooting

rflump classifies problems arising during execution into two categories:

1. Non-critical problems: rflump attempts to make an automatic recovery. It will print a warning message and continue with the execution. The recovery attempt may entail changing input variables specified by the user.

2. Critical problems: rflump cannot recover without user intervention. It will print an error message and stop execution.

10.1 General Considerations

If a problem is encountered and it proves difficult to determine the source, the following suggestions may prove helpful:

1. Repeat the run or restart from the last output dump with maximum verbosity level. The additional output printed by activating the maximum verbosity level can indicate possible problem areas, see Sec. 10.5

2. Repeat the run with smaller CFL number/time step. Decreasing the CFL number/time step can mitigate the effect of strong transients. It is possible that some computations have to be started with smaller a CFL number/time step to ensure stability.

3. Repeat the run with different flux function. Some flux functions exhibit pathological behavior under certain conditions or do not guarantee preservation of positive-definite quantities. For example, of the flux functions available in RocfluMP, the HLLC scheme often exhibits more robustness in strong transients than the Roe scheme.

4. Repeat the run with a first-order scheme. Failing first-order accurate computations often indicate problems with problem setup.

5. Repeat the run with modified boundary conditions. For example, if problems persist during strong transients at an outflow boundary, it may prove helpful to attempt a new run with supersonic or subsonic conditions.

10.2 Explanations of Warnings

`Inflow detected at outflow boundary!`
`Outflow detected at inflow boundary!`
One or more faces on an inflow/outflow boundary were detected with outflow/inflow, respectively. This is often caused by the precise values of the boundary conditions imposed by the user. For example, too high a value of the static pressure at outflow boundaries can lead to inflow in some conditions. Reverse flow on inflow/outflow boundaries can be a temporary problem, i.e., the expected flow direction becomes established as the flow develops. In some cases, however, reverse flow on boundaries can indicate a more serious problem, namely a poorly chosen location of boundaries. This may be reflected in the persistence of inflow at an outflow boundary despite lowering the static pressure, the persistence of outflow at an inflow boundary despite increasing the stagnation pressure, or recirculation regions across outflow boundaries. In these cases, it is necessary to reposition the relevant boundary patches.

These warnings will be accompanied by additional output which provides more information about the precise location of the boundary faces with reverse flow. Section 10.5 describes how this output can be used to visualize the cells affected by reverse flow.

10.3 Explanations of Errors

`Absolute difference in volumes larger than specified limit.`
rflump computes the total volume of the computational domain using two methods. The first method simply sums the volumes of all the cells in a given region. The second method computes the total volume from the boundary and interregion faces. If the two methods disagree by more than a (presently hardcoded) tolerance, it is likely that the interior grid in invalid and the above error message is printed.

`Invalid quantity detected.`
rflump detected solution variables with the value `NaN`. This indicates a serious problem. This is often the result of using a CFL number or a time step which is too large. For coupled simulations, invalid variables may also arise if the geometry becomes heavily deformed. This error message may also indicate problems with the specification of boundary conditions.

`Negative positive-definite quantity detected.`
Negative positive-definite solution variables, such as density, pressure, or temperature were detected. This is often the result of using a CFL number or a time step which is too large.

For coupled simulations, negative positive-definite variables may also arise if the geometry becomes heavily deformed.

`Negative volume(s) detected.`
One or more cells with negative volumes were detected. For computations without moving grids, this indicates that the cell or boundary-face connectivity is incorrect. For computations with moving grids, negative volumes indicate that either the boundary deformation is too strong given the grid-motion parameters specified by the user, or the grid quality is poor and deteriorates with grid motion. In the latter case, the grid quality needs to be improved before a new run is started.

`Face sum greater than minimum face area.`
rflump checks whether cells are closed by computing the sum of the face vectors and comparing them against a (presently hardcoded) tolerance. It the sum of the face vectors exceeds the tolerance, it is likely that the cell or boundary-face connectivity is incorrect.

10.4 Other Problems

This section lists problems and suggested remedies for problems which do not lead to warnings or errors.

Cells near moving boundaries become highly stretched.
Highly stretched cells near moving boundaries indicate that the boundary motion is not propagated well enough into the interior. This problem can be remedied by increasing the number of smoothing interations `NITER` or by increasing the smoothing coefficient `SFACT` in the `GRIDMOTION` section of the input file.

10.5 Locating Troublespots

If problems are encountered, rflump prints additional information to allow the user to determine precisely the locations at which problems occur. For example, consider the following output:

```
Printing location information...
  Global region: 00001
  Cell location information:
     #      Cell   x-coordinate    y-coordinate    z-coordinate   Location
     1        51  0.32081289E-02 -0.83698854E-01 -0.12040992E-01  Global patches:   1   6
     2       300  0.12485188E-01 -0.83750698E-01  0.61473528E-03  Global patches:   1   6
     3       357  0.35957776E-02 -0.83795613E-01 -0.74193030E-02  Global patch:   6
     4      1696  0.11128962E-01 -0.83783063E-01 -0.56762047E-02  Global patches:   1   6
     5      2048  0.10507133E-01 -0.83728027E-01 -0.66900307E-02  Global patches:   1   6
     6      2066  0.87561363E-02 -0.83784932E-01 -0.89096344E-02  Global patches:   1   6
```

```
     7      2116  0.47333590E-02 -0.83785244E-01 -0.97767197E-02  Global patch:   6
     8      2574  0.78548965E-03 -0.83765772E-01 -0.12463422E-01  Global patches:   1   6
     9      2995  0.48426506E-02 -0.83799824E-01 -0.90511979E-02  Global patch:   6
    10      6461  0.45099568E-02 -0.83819846E-01 -0.11576659E-01  Interior
Printing location information done.
```

The output lists the cells and the coordinates of the cell centroids along with their locations. The location indicates whether the cell shares one or more faces with boundary patches. In some cases, the above information may suffice because all or a majority of cells lie on a given boundary patch, indicating that the problem originates with the boundary conditions applied to this patch. If the location of the cells is not immediately apparent or scattered around the computational domain, it can be helpful to select individual cells for visualization using rflupick. This can be useful to determine whether the problems originate in a clusters of cells of poor quality, or along the interface between layers of different cell types.

If the source of the problem is not readily apparent from the additional output, it is often helpful to place probes at the location of the cell centroids before restarting the computation or starting a new computation. Taking the first three cells of above example, the following `PROBE` may be inserted into the `<casename>.inp` file:

```
# PROBE
NUMBER 3
0.32081289E-02 -0.83698854E-01 -0.12040992E-01
0.12485188E-01 -0.83750698E-01  0.61473528E-03
0.35957776E-02 -0.83795613E-01 -0.74193030E-02
#
#
```

The probe files produced by repeating the run can be visualized with TECPLOT (or other packages) and may help in determining the cause of the problem.

Part III

Developer and Reference Manual

Chapter 11

Governing Equations

11.1 The Navier-Stokes Equations

RocfluMP solves the three-dimensional time-dependent compressible Navier-Stokes equations in integral form on moving and/or deforming grids,

$$\frac{\partial}{\partial t}\int_{\Omega} \vec{W}\, d\Omega + \oint_{\partial\Omega} \vec{F}_c\, dS = \oint_{\partial\Omega} \vec{F}_v\, dS \int_{\Omega} \vec{Q}\, d\Omega. \tag{11.1}$$

The vector of the conservative variables $\vec{W}$ is

$$\vec{W} = \begin{bmatrix} \rho \\ \rho u \\ \rho v \\ \rho w \\ \rho E \end{bmatrix}. \tag{11.2}$$

The vector of convective fluxes reads

$$\vec{F}_c = \begin{bmatrix} \rho V \\ \rho u V + n_x p \\ \rho v V + n_y p \\ \rho w V + n_z p \\ \rho H V + V_g p + F^R \end{bmatrix} \tag{11.3}$$

with the face-normal velocity given by

$$V = n_x u + n_y v + n_z w - V_g, \tag{11.4}$$

where V_g is the grid speed, i.e., the grid velocity normal to the control-volume face. The vector of the viscous fluxes can be written as

$$\vec{F}_v = \begin{bmatrix} 0 \\ n_x\tau_{xx} + n_y\tau_{xy} + n_z\tau_{xz} \\ n_x\tau_{yx} + n_y\tau_{yy} + n_z\tau_{yz} \\ n_x\tau_{zx} + n_y\tau_{zy} + n_z\tau_{zz} \\ n_x\Theta_x + n_y\Theta_y + n_z\Theta_z \end{bmatrix}, \tag{11.5}$$

where

$$\begin{aligned}\Theta_x &= u\tau_{xx} + v\tau_{xy} + w\tau_{xz} + k\frac{\partial T}{\partial x} + E_x^{SGS} \\ \Theta_y &= u\tau_{yx} + v\tau_{yy} + w\tau_{yz} + k\frac{\partial T}{\partial y} + E_y^{SGS} \\ \Theta_z &= u\tau_{zx} + v\tau_{zy} + w\tau_{zz} + k\frac{\partial T}{\partial z} + E_z^{SGS}\end{aligned} \tag{11.6}$$

are the terms describing the work of the viscous stresses and the heat conduction in the fluid. In the case of LES, the viscous stresses read

$$\tau_{ij} = 2\mu\, S_{i,j} - \frac{2}{3}\mu\frac{\partial v_k}{\partial x_k}\delta_{ij} + \tau_{ij}^{SGS} \tag{11.7}$$

with τ_{ij}^{SGS} being the subgrid stresses. In the case of RANS equations, the subgrid terms E^{SGS} in Eq. (11.6) and τ_{ij}^{SGS} in Eq. (11.7) are omitted. Instead, the dynamic viscosity and the thermal conductivity are split into a laminar and a turbulent part, i.e.,

$$\mu = \mu_L + \mu_T \tag{11.8}$$

and

$$k = k_L + k_T = c_p\left(\frac{\mu_L}{Pr_L} + \frac{\mu_T}{Pr_T}\right), \tag{11.9}$$

where c_p stands for the specific heat coefficient at constant pressure, and Pr is the Prandtl number.

The source term is given by

$$\vec{Q} = \begin{bmatrix} m^P \\ \rho f_{e,x} + f_x^P + f_x^S \\ \rho f_{e,y} + f_y^P + f_y^S \\ \rho f_{e,z} + f_z^P + f_z^S \\ \rho\vec{f_e}\cdot\vec{v} + \dot{q}_h + E^P + E^S \end{bmatrix}, \tag{11.10}$$

where m^P, f^P, f^S, E^P, and E^S represent the source terms introduced by the particle and smoke modeling. The vector of external volume forces reads

$$\vec{f_e} = \vec{g} - \vec{a} \tag{11.11}$$

with $\vec{g}$ being the gravitational acceleration and $\vec{a}$ the acceleration of the rocket.

11.2 The Geometric Conservation Law

For uniform flow, Eq. (11.1) reduces to the Geometric Conservation Law (GCL),

$$\frac{\partial}{\partial t}\int_\Omega d\Omega - \oint_{\partial\Omega} V_t\, dS = 0\,. \tag{11.12}$$

The GCL ensures that the motion of the grid does not alter a uniform flow. It must be satisfied in a discrete sense, independent of the deformation of the grid and the numerical solution method. The discretization of the GCL in RocfluMP is described in Sec. 12.5.2.

11.3 Gas Models

11.3.1 Calorically and Thermally Perfect Gas

11.3.2 Mixture of Calorically and Thermally Perfect Gases

11.3.3 Pseudo Gas

11.4 Thermodynamic Properties

11.4.1 Calorically and Thermally Perfect Gas

11.4.2 Mixture of Calorically and Thermally Perfect Gases

11.4.3 Pseudo Gas

11.5 Transport Properties

11.5.1 Viscosity

11.5.1.1 Sutherland Model

The viscosity is computed from

$$\frac{\mu}{\mu_{\text{ref}}} = \left(\frac{T}{T_{\text{ref}}}\right)^{\frac{3}{2}} \frac{T_{\text{ref}} + S_{\text{ref}}}{T + S_{\text{ref}}}$$

where μ is the dynamic viscosity, μ_{ref} is the reference dynamic viscosity, T is the static temperature, T_{ref} is the reference temperature, and S_{ref} is the Sutherland constant.

11.5.1.2 Antibes Model

The viscosity is computed from

$T_{\text{ref}} \geq 120\,\text{K}$:

$$\mu = \begin{cases} \mu_{\text{ref}} \left(\dfrac{T}{T_{\text{ref}}}\right)^{\frac{3}{2}} \dfrac{T_{\text{ref}} + S_{\text{ref}}}{T + S_{\text{ref}}} & \text{if} \quad T \geq 120\,\text{K} \\ \mu_{120}\,\dfrac{T}{120} & \text{if} \quad T < 120\,\text{K} \end{cases}$$

$T_{\text{ref}} < 120\,\text{K}$:

$$\mu = \begin{cases} \mu_{\text{ref}} \dfrac{T}{T_{\text{ref}}} & \text{if} \quad T < 120\,\text{K} \\ \mu_{120} \left(\dfrac{T}{120}\right)^{\frac{3}{2}} \dfrac{120 + S_{\text{ref}}}{T + S_{\text{ref}}} & \text{if} \quad T \geq 120\,\text{K} \end{cases}$$

This formula was specified by the organizers of the Workshop on Hypersonic Flows for Reentry Problems [1]. S_{ref} is given the value 110 K and μ_{120} denotes the value of the viscosity at 120 K.

11.5.2 Conductivity

11.6 Boundary Conditions

Chapter 12

Algorithms and Methods

12.1 Geometry Definition

RocfluMP uses a method originally due to Bruner [4] and improved by Wang [16] to compute geometrical properties of faces and volumes. The method is particularly convenient for finite-volume schemes because volume properties are expressed in terms of face properties. This means that the face and volume properties can be computed in a single loop over faces.

12.1.1 Computation of Face Properties

The face properties of interest are the normal vector, the area, and its centroid.

For a triangular face, the scaled normal vector is given by

$$\mathbf{n} = \frac{1}{2}\left(\mathbf{r}_{12} \times \mathbf{r}_{23}\right). \tag{12.1}$$

where $\mathbf{r}_{ij} = \mathbf{r}_j - \mathbf{r}_i$. For a quadrilateral face, the normal vector is given by the average of the average normal vectors obtained by subdividing the face into two triangles.

The area of triangular and quadrilateral faces follows from Eq. 12.1 as

$$S = \|\mathbf{n}\|_2. \tag{12.2}$$

The centroid of a triangular face is computed from

$$\mathbf{r}^c = \frac{1}{3}\left(\mathbf{r}_1 + \mathbf{r}_2 + \mathbf{r}_3\right). \tag{12.3}$$

For a quadrilateral face, the centroid is given by the average of the average centroids obtained by subdividing the face into two triangles.

12.1.2 Computation of Volume Properties

The volume properties of interest are the volume and its centroid.

For a polyhedron composed of N triangular faces, the volume may be computed from

$$V = \frac{1}{3} \sum_{i=1}^{N} \mathbf{r}_i^c \cdot \mathbf{n}_i \, S_i. \tag{12.4}$$

The centroid of a polyhedron composed of N triangular faces may be computed from

$$\mathbf{r}_c = \frac{1}{4V} \sum_{i=1}^{N} (\mathbf{r}_i^c \cdot \mathbf{n}_i) \; \mathbf{r}_i^c \, S_i \tag{12.5}$$

For polyhedra with quadrilateral faces, Eqs. (12.4) and (12.5) can be applied given that they only involve face properties which have already been computed.

It is interesting to note that Eq. (12.5) expresses the volume centroid as a weighted sum of face centroids, and that the weights are not guaranteed to be positive. Positive weights can be guaranteed by first computing an approximate cell centroid $\tilde{\mathbf{r}}_c$ as the average of the vertex position vectors, and then replacing $\mathbf{r}_i^c \cdot \mathbf{n}_i$ by $(\mathbf{r}_i^c - \widetilde{\mathbf{r}}_c) \cdot \mathbf{n}_i$ in Eq. (12.5).

12.2 Spatial Discretization

12.2.1 Stencil Construction

Explicit stencils only need to be constructed for the interpolation and gradient operators. The minimum extent or size of these stencils is determined by their order of accuracy.

For an interpolation operator of order p on an arbitrary grid, the minimum extent is given by

$$\overline{N}_{\min} = \frac{(p+1)(p+2)(p+3)}{6} + 1. \tag{12.6}$$

In the present work, filter operators are regarded as low-order interpolation operators. For a gradient operator of order q on an arbitrary grid, the minimum extent is given by

$$N_{\min} = \frac{(q+1)(q+2)(q+3)}{6}. \tag{12.7}$$

For both interpolation and gradient operators, it may be advantageous to increase the stencil extent to include more cell centroids than necessary. For filter operators, this can be used to minimize the imaginary part of the transfer function. The larger-than-necessary support necessitates the use of least-squares techniques to determine the interpolated value or gradients.

Gradient operators are required at cell and face centroids. Interpolation operators are required at cell and face centroids and at vertices. Hence cell-to-cell, face-to-cell, and vertex-to-cell stencils must be constructed, as depicted schematically in Fig. 12.1.

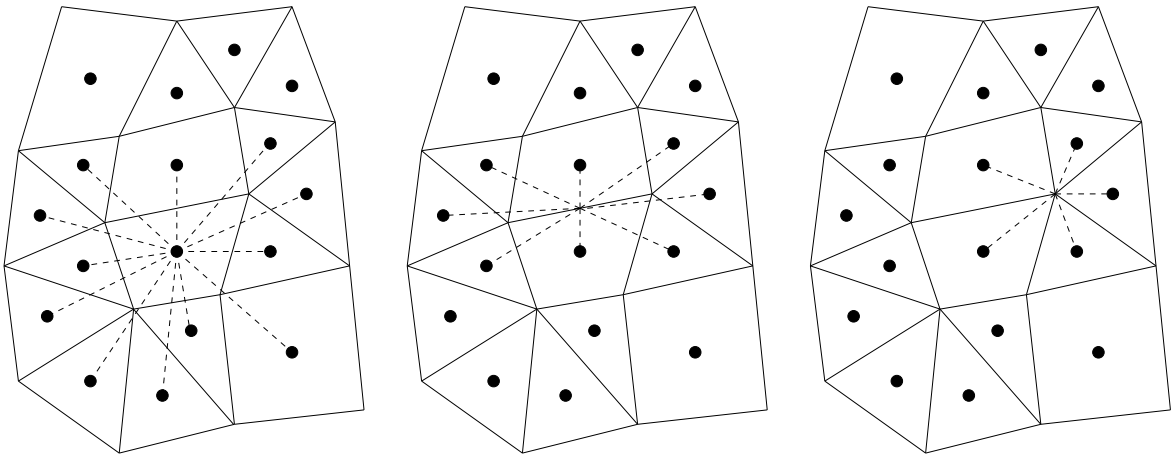

Figure 12.1: Schematic illustration stencils in two dimension. **(a)** Cell-to-cell stencil, **(b)** face-to-cell stencil, and **(c)** vertex-to-cell stencil.

At present, these stencils are constructed using an Octree-based approach. The Octree is initialized using cell-centroid coordinates, and queried with the locations at which the interpolation or gradient operators are to be constructed.

12.2.2 Interpolation Operators

The interpolation operators are constructed using a least-squares approach based on a modified Taylor series. Assuming linear interpolation of a scalar variable ϕ, this gives

$$\phi_i = \overline{\phi}_0 + (\nabla\phi)_0 \cdot \Delta\mathbf{r}_{0i}, \tag{12.8}$$

where $\overline{\phi}_0$ is the interpolated value at location 0, which may be a cell or face centroid or a vertex. Assuming that Eq. (12.8) is applied to the d_0 points in the stencil, an overdetermined system of linear equations is obtained,

$$\begin{bmatrix} \Delta x_{01} & \Delta y_{01} & 1 \\ \Delta x_{02} & \Delta y_{02} & 1 \\ \vdots & \vdots & \vdots \\ \Delta x_{0d_0} & \Delta y_{0d_0} & 1 \end{bmatrix} \begin{Bmatrix} \partial_x\phi \\ \partial_y\phi \\ \overline{\phi} \end{Bmatrix}_0 = \begin{Bmatrix} \phi_1 \\ \phi_2 \\ \vdots \\ \phi_{d_0} \end{Bmatrix}, \tag{12.9}$$

or

$$\mathbf{A}\mathbf{x} = \mathbf{b}. \tag{12.10}$$

The system can be inverted using the Singular Value Decomposition (SVD), which gives a interpolation formula of the form,

$$\overline{\phi}_0 = \sum_{i=1}^{d_0} \overline{\omega}_{0i}\phi_i, \tag{12.11}$$

where the stencil weights are given by

$$\overline{\omega}_{0i} = \mathbf{A}^{-1}_{3,i} \quad \text{for} \quad 1 \leq i \leq d_0. \tag{12.12}$$

12.2.3 Gradient Operators

The gradient operators are also constructed using a least-squares approach. Assuming linear gradient reconstruction of a scalar variable ϕ, this gives

$$\phi_i = \phi_0 + (\nabla\phi)_0 \cdot \Delta\mathbf{r}_{0i}, \tag{12.13}$$

where ϕ_0 is now the value at location 0, which may be a cell or face centroid or a vertex. Assuming that Eq. (12.13) is applied to the d_0 points in the stencil, an overdetermined system of linear equations is obtained,

$$\begin{bmatrix} \Delta x_{01} & \Delta y_{01} \\ \Delta x_{02} & \Delta y_{02} \\ \vdots & \vdots \\ \Delta x_{0d_0} & \Delta y_{0d_0} \end{bmatrix} \begin{Bmatrix} \partial_x\phi \\ \partial_y\phi \end{Bmatrix}_0 = \begin{Bmatrix} \Delta\phi_{01} \\ \Delta\phi_{02} \\ \vdots \\ \Delta\phi_{0d_0} \end{Bmatrix}, \tag{12.14}$$

or

$$\mathbf{A}\mathbf{x} = \mathbf{b}. \tag{12.15}$$

The system can be inverted using the Singular Value Decomposition (SVD), which gives a formula of the form,

$$(\nabla\phi)_0 = \sum_{i=1}^{d_0} \boldsymbol{\omega}_{0i}\Delta\phi_{0i}, \tag{12.16}$$

where the vector of stencil weights is given by

$$\boldsymbol{\omega}_{0i} = \mathbf{A}^{-1}_{1:2,i} \quad \text{for} \quad 1 \leq i \leq d_0. \tag{12.17}$$

12.2.4 Inviscid Fluxes

12.2.4.1 Limiter Functions

12.2.4.2 Numerical Flux Functions

12.2.4.3 Entropy Fixes

12.2.5 Viscous Fluxes

12.2.6 Optimal LES Discretization

12.2.6.1 Computation of Integrals

12.2.6.2 Computation of Stencil Weights

12.2.7 Source Terms

12.3 Boundary Conditions

12.3.1 Simple Fluid-Solid Boundary Conditions

12.3.1.1 Slip Wall Boundaries

12.3.1.2 No-Slip Wall Boundaries

12.3.1.3 Injection Wall Boundaries

12.3.2 Simple Fluid-Fluid Boundary Conditions

12.3.2.1 Inflow Boundaries

12.3.2.2 Outflow Boundaries

12.3.3 Non-Reflecting Boundary Conditions

This section describes formulations to implement Navier-Stokes Characteristic Boundary-Conditions (NSCBC) for Euler and Navier-Stokes equations. Such implementation controls the spurious wave reflection at the computational boundaries and provide means for reflecting and non-reflecting boundary conditions treatments.

Subject matter is this document is partly taken from 2 papers on characteristics based boundary conditions (see [8] and [14]).

12.3.3.1 Overall Approach

Euler Equations are

$$\frac{\partial \rho}{\partial t} + \frac{\partial \rho u}{\partial x} + \frac{\partial \rho v}{\partial y} + \frac{\partial \rho w}{\partial z} = 0 \tag{12.18}$$

$$\frac{\partial \rho u}{\partial t} + \frac{\partial \rho u u}{\partial x} + \frac{\partial \rho u v}{\partial y} + \frac{\partial \rho u w}{\partial z} = -\frac{\partial p}{\partial x} \tag{12.19}$$

$$\frac{\partial \rho v}{\partial t} + \frac{\partial \rho v u}{\partial x} + \frac{\partial \rho v v}{\partial y} + \frac{\partial \rho v w}{\partial z} = -\frac{\partial p}{\partial y} \tag{12.20}$$

$$\frac{\partial \rho w}{\partial t} + \frac{\partial \rho w u}{\partial x} + \frac{\partial \rho w v}{\partial y} + \frac{\partial \rho w w}{\partial z} = -\frac{\partial p}{\partial z} \tag{12.21}$$

$$\frac{\partial \rho E}{\partial t} + \frac{\partial \rho H u}{\partial x} + \frac{\partial \rho H v}{\partial y} + \frac{\partial \rho H w}{\partial z} = 0 \tag{12.22}$$

These equations can be written in any orthogonal coordinate reference frame. Their form remain invariant. lets consider a reference frame (*n-s-r*) instead of (*x-y-z*). flow velocities would now be denoted by u_n,u_s,u_r (instead of *u,v,w*), which refers to velocity components in *n,s,r* coordinate directions. Euler equations are now,

$$\frac{\partial \rho}{\partial t} + \frac{\partial \rho u_n}{\partial n} + \frac{\partial \rho u_s}{\partial s} + \frac{\partial \rho u_r}{\partial r} = 0 \tag{12.23}$$

$$\frac{\partial \rho u_n}{\partial t} + \frac{\partial \rho u_n u_n}{\partial n} + \frac{\partial \rho u_n u_s}{\partial s} + \frac{\partial \rho u_n u_r}{\partial r} = -\frac{\partial p}{\partial n} \tag{12.24}$$

$$\frac{\partial \rho u_s}{\partial t} + \frac{\partial \rho u_s u_n}{\partial n} + \frac{\partial \rho u_s u_s}{\partial s} + \frac{\partial \rho u_s u_r}{\partial r} = -\frac{\partial p}{\partial s} \tag{12.25}$$

$$\frac{\partial \rho u_r}{\partial t} + \frac{\partial \rho u_r u_n}{\partial n} + \frac{\partial \rho u_r u_s}{\partial s} + \frac{\partial \rho u_r u_r}{\partial r} = -\frac{\partial p}{\partial r} \tag{12.26}$$

$$\frac{\partial \rho E}{\partial t} + \frac{\partial \rho H u_n}{\partial n} + \frac{\partial \rho H u_s}{\partial s} + \frac{\partial \rho H u_r}{\partial r} = 0 \tag{12.27}$$

Consider a boundary such that normal direction *n* is perpendicular to face and *s,r* direction are tangential to boundary. Using characteristic analysis to modify the Hyperbolic terms of Euler equations (12.23)-(12.27) corresponding to waves propagating in the *n* direc-

tion, Euler equations can be written in following form,

$$\frac{\partial \rho}{\partial t} + d_1 + \frac{\partial \rho u_s}{\partial s} + \frac{\partial \rho u_r}{\partial r} = 0 \tag{12.28}$$

$$\frac{\partial \rho u_n}{\partial t} + u_n d_1 + \rho d_3 + \frac{\partial \rho u_n u_s}{\partial s} + \frac{\partial \rho u_n u_r}{\partial r} = 0 \tag{12.29}$$

$$\frac{\partial \rho u_s}{\partial t} + u_s d_1 + \rho d_4 + \frac{\partial \rho u_s u_s}{\partial s} + \frac{\partial \rho u_s u_r}{\partial r} = -\frac{\partial p}{\partial s} \tag{12.30}$$

$$\frac{\partial \rho u_r}{\partial t} + u_r d_1 + \rho d_5 + \frac{\partial \rho u_r u_s}{\partial s} + \frac{\partial \rho u_r u_r}{\partial r} = -\frac{\partial p}{\partial r} \tag{12.31}$$

$$\frac{\partial \rho E}{\partial t} + \frac{1}{2}(u_n^2 + u_s^2 + u_r^2)d_1 + \frac{d_2}{\gamma - 1} + \rho u_n d_3 + \rho u_s d_4 + \rho u_r d_5 + \frac{\partial \rho H u_s}{\partial s} + \frac{\partial \rho H u_r}{\partial r} = 0 \tag{12.32}$$

The vector $\mathbf{d}$, which contains normal derivative terms $(\partial/\partial n)$, is given by characteristic analysis as follows:

$$\mathbf{d} = \begin{Bmatrix} d_1 \\ d_2 \\ d_3 \\ d_4 \\ d_5 \end{Bmatrix} = \begin{Bmatrix} \frac{1}{c^2}\left[L_2 + \frac{1}{2}(L_5 + L_1)\right] \\ \frac{1}{2}(L_5 + L_1) \\ \frac{1}{2\rho c}(L_5 - L_1) \\ L_3 \\ L_4 \end{Bmatrix} = \begin{Bmatrix} \frac{\partial \rho u_n}{\partial n} \\ \rho c^2 \frac{\partial u_n}{\partial n} + u_n \frac{\partial p}{\partial n} \\ u_n \frac{\partial u_n}{\partial n} + \frac{1}{\rho}\frac{\partial p}{\partial n} \\ u_n \frac{\partial u_s}{\partial n} \\ u_n \frac{\partial u_r}{\partial n} \end{Bmatrix} \tag{12.33}$$

In (n-s-r) coordinate frame L_i's can be written as:

$$\begin{Bmatrix} L_1 \\ L_2 \\ L_3 \\ L_4 \\ L_5 \end{Bmatrix} = \begin{Bmatrix} \lambda_1 \left(\frac{\partial p}{\partial n} - \rho c \frac{\partial u_n}{\partial n}\right) \\ \lambda_2 \left(c^2 \frac{\partial \rho}{\partial n} - \frac{\partial p}{\partial n}\right) \\ \lambda_3 \frac{\partial u_s}{\partial n} \\ \lambda_4 \frac{\partial u_r}{\partial n} \\ \lambda_5 \left(\frac{\partial p}{\partial n} + \rho c \frac{\partial u_n}{\partial n}\right) \end{Bmatrix} \tag{12.34}$$

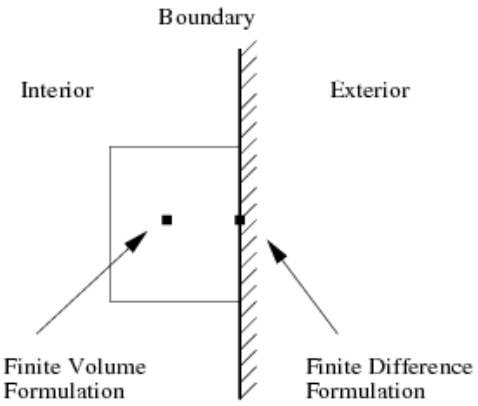

Figure 12.2: Finite difference at boundary.

where λ_i's are defined as:

$$\begin{Bmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \\ \lambda_4 \\ \lambda_5 \end{Bmatrix} = \begin{Bmatrix} u_n - c \\ u_n \\ u_n \\ u_n \\ u_n + c \end{Bmatrix} \tag{12.35}$$

Care should be given to the fact that at the interior of domain finite-volume formulation is used. Flux integration for interior domain uses cell averaged quantity (see equations 12.36-12.37). While at the boundary, flow equations are solved using Finite-difference formulation and are solved for actual variables on boundary face, see Fig. 12.2.

$$\frac{\partial}{\partial t}\int_\Omega \phi dV + \oint_{\partial\Omega} \phi \mathbf{v}\cdot\mathbf{n} ds = 0 \tag{12.36}$$

$$V\frac{d\bar{\phi}}{dt} + \oint_{\partial\Omega} \phi \mathbf{v}\cdot\mathbf{n} ds = 0 \tag{12.37}$$

where $\bar{\phi} = \frac{1}{V}\oint_\Omega \phi dV$ is the average value of ϕ over cell.

The Local One-Dimensional Inviscid (LODI) Relations. At any point on the boundary we can obtain a LODI system by considering the system of Eqs (12.23)-(12.27) and neglecting transverse terms (setting 's' and 'r' direction terms to zero). This is one-dimensional Euler equation in 'n' direction. In terms of primitive variables LODI relation can be written

as:

$$\frac{\partial \rho}{\partial t} = -\frac{1}{c^2}\left[L_2 + \frac{1}{2}(L_5 + L_1)\right] \tag{12.38}$$

$$\frac{\partial p}{\partial t} = -\frac{1}{2}(L_5 + L_1) \tag{12.39}$$

$$\frac{\partial u_n}{\partial t} = -\frac{1}{2\rho c}(L_5 - L_1) \tag{12.40}$$

$$\frac{\partial u_s}{\partial t} = -L_3 \tag{12.41}$$

$$\frac{\partial u_r}{\partial t} = -L_4 \tag{12.42}$$

These relations may be combined to express the time derivatives of all the other quantities of interest, e.g., Temperature, T.

$$\frac{\partial T}{\partial t} + \frac{T}{\rho c^2}\left[-L_2 + \frac{1}{2}(\gamma - 1)(L_5 + L_1)\right] = 0 \tag{12.43}$$

The NSCBC Strategy for the Euler Equations. Characteristics based boundary conditions are implemented in following 3 steps.

1. For each physical boundary condition imposed, corresponding equation is eleminated from the system (12.23)-(12.27). like if p or T is imposed, then there is no need to solve energy equation at boundary.

2. At each boundary some of the characteristics are going out of domain and some are coming in. Characteristic amplitudes L_i's corresponding to outgoing waves can be computed using one sided differencing. Incoming wave amplitudes can not be computed using one sided differencing. Using LODI relations and imposed boundary conditions, these incoming wave amplitudes can be expressed in terms of outgoing wave amplitudes.

3. Use the remaining conservation equations of the system (12.23)-(12.27) combined with the values of the L_i's obtained from Step 2 to compute all variables which are not imposed by boundary conditions.

12.3.3.2 Inflow Boundary Conditions

Inflow boundary conditions discussed in this section are the ones where velocity and temperature are specified. Inflow can be subsonic or supesonic. subsonic inflow can be reflecting or non-reflecting.

At a generic boundary face u_n has to be negative for it to be inflow. $u_n < 0$ means flow is incoming i.e. flow is entering the domain from outside.

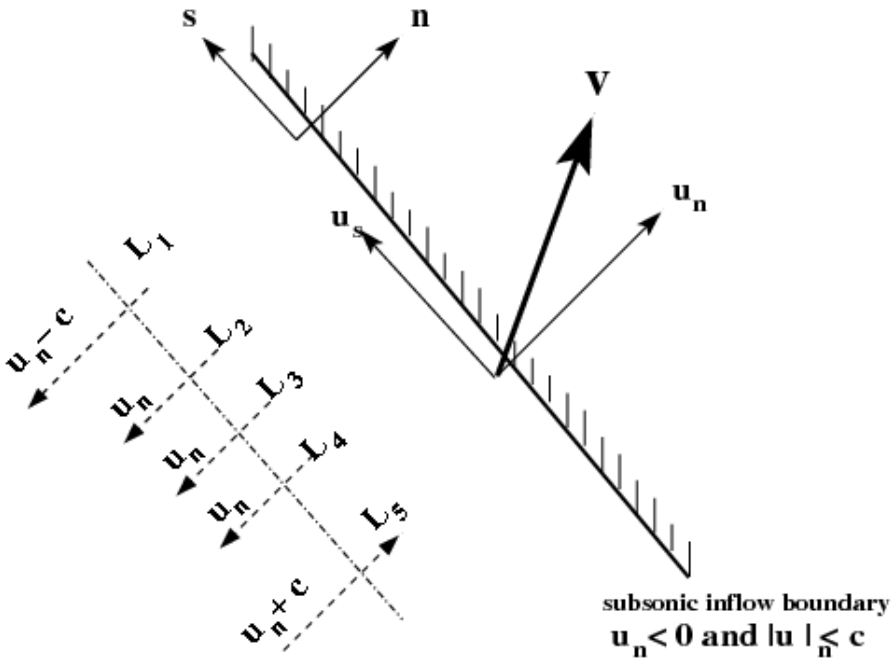

Figure 12.3: Subsonic inflow boundary

Subsonic Inflow For subsonic inflow ($|u_n| < c$) 4 characteristics (corresponding to $\lambda_1, \lambda_2, \lambda_3, \lambda_4$) are entering the domain and one characteristic (λ_5) is leaving the domain (see Fig. 12.3). This implies, L_5 would always be computed from interior solution while L_1, L_2, L_3, L_4 need to be specified. L_5 can be computed as given by equation (12.34).

$$L_5 = \lambda_1 \left(\frac{\partial p}{\partial n} - \rho c \frac{\partial u_n}{\partial n} \right)$$

because u_n, u_s, u_r, T are imposed at inflow, momentum and energy equations are eleminated at boundary. Only density need to be solved at inflow boundary.

Subsonic Reflecting Inflow. For reflecting subsonic inflow, LODI relations for 4 imposed physical quantities (u_n, u_s, u_r, T) can be used to express unknown wave amplitudes (L_1, L_2, L_3, L_4) in terms of known wave amplitude L_5 and boundary conditions as follows:

$$L_5 = L_1 - 2\rho c\frac{du_n}{dt} \tag{12.44}$$

$$L_2 = \frac{1}{2}(\gamma - 1)(L_5 + L_1) + \frac{\rho c^2}{T}\frac{dT}{dt} \tag{12.45}$$

$$L_3 = \frac{du_s}{dt} \tag{12.46}$$

$$L_4 = \frac{du_r}{dt} \tag{12.47}$$

$$\tag{12.48}$$

Subsonic Non-Reflecting Inflow. For non-reflecting subsonic inflow, all incoming wave amplitudes are set to zero.

$$L_5 = L_2 = L_3 = L_4 = 0 \tag{12.49}$$

Supersonic Inflow. For supersonic inflow ($|u_n| > c$) all 5 characteristics are entering the domain. No characteristic wave can travel upstream. For supersonic inflow all characteristic amplitudes are set to zero.

$$L_1 = L_2 = L_3 = L_4 = L_5 = 0 \tag{12.50}$$

12.3.3.3 Outflow Boundary Conditions

Outflow boundary conditions discussed in this section are the ones where pressure is specified. Outflow can be subsonic or supesonic. subsonic outflow can be reflecting or non-reflecting.

At a generic boundary face u_n has to be positive for it to be outflow. $u_n > 0$ means flow is outgoing.

Subsonic Outflow. For subsonic outflow ($|u_n| < c$) one characteristic (corresponding to λ_1) is entering the domain and 4 characteristics ($\lambda_2, \lambda_3, \lambda_4, \lambda_5$) are leaving the domain (see Fig. 12.4).This implies, L_2, L_3, L_4, L_5 would always be computed from interior solution while L_1 need to be specified. L_2, L_3, L_4, L_5 can be computed as given by equation (12.34).

$$L_2 = \lambda_2\left(c^2\frac{\partial \rho}{\partial x} - \frac{\partial p}{\partial x}\right)$$

$$L_3 = \lambda_3\frac{\partial v}{\partial x}$$

$$L_4 = \lambda_4\frac{\partial w}{\partial x}$$

$$L_5 = \lambda_5\left(\frac{\partial p}{\partial x} + \rho c\frac{\partial u}{\partial x}\right)$$

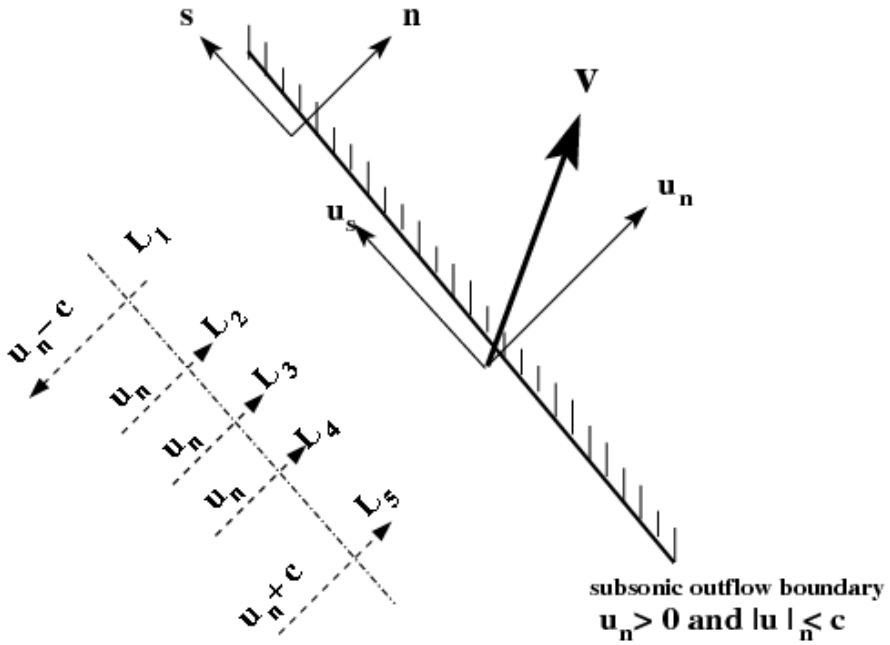

Figure 12.4: Subsonic outflow boundary

because p is imposed at outflow, energy equation need not be solved at the boundary.

Subsonic Reflecting Outflow. For reflecting subsonic outflow, LODI relations for pressure can be used to express unknown wave amplitude L_1 in terms of L_5 and boundary condition as follows:

$$L_1 = -L_5 - 2\frac{dp}{dt} \tag{12.51}$$

Subsonic Non-Reflecting Outflow. For non-reflecting subsonic outflow, all incoming wave amplitudes are set to zero.

$$L_1 = 0 \tag{12.52}$$

This is perfectly non-reflecting outflow. However sometimes little reflection should be allowed for to simulate real conditions. In such case L_1 can be expressed as:

$$L_1 = K(p - p_\infty) \tag{12.53}$$

Supersonic Outflow. For supersonic outflow ($|u_n| > c$) all 5 characteristics are leaving the domain. No characteristic wave can travel downstream. For supersonic outflow all characteristic amplitudes can be computed using relation (12.39).

12.3.4 Virtual Boundary Conditions

12.3.4.1 Periodic Boundaries

12.3.4.2 Symmetry Boundaries

12.4 Temporal Discretization

12.4.1 Runge-Kutta Methods

12.4.2 Computation of Time Step

12.5 Grid Motion

12.5.1 Grid Smoothing

12.5.2 Discrete Geometric Conservation Law

12.5.3 Implementation Details

12.6 Mass, Pressure, Skin-Friction, and Heat-Transfer Coefficient Computation

RocfluMP computes mass, pressure, skin-friction, and heat-transfer coefficients for faces on patches. In the following, the subscript i denotes variables associated with a face with normal vector $\mathbf{n}_i$.

The mass-flux coefficient is defined as

$$C_{m,i} = \frac{\rho_i \mathbf{v}_i \cdot \mathbf{n}_i}{\frac{1}{2}\rho_{\text{ref}} V_{\text{ref}}}. \tag{12.54}$$

The pressure coefficient is defined as

$$C_{p,i} = \frac{p_i - p_{\text{ref}}}{\frac{1}{2}\rho_{\text{ref}} V_{\text{ref}}^2}. \tag{12.55}$$

The skin-friction coefficients are defined as

$$C_{fx,i} = \frac{(\boldsymbol{\tau}_i \cdot \mathbf{n}_i)_x}{\frac{1}{2}\rho_{\text{ref}} V_{\text{ref}}^2}, \tag{12.56a}$$

$$C_{fy,i} = \frac{(\boldsymbol{\tau}_i \cdot \mathbf{n}_i)_y}{\frac{1}{2}\rho_{\text{ref}} V_{\text{ref}}^2}, \tag{12.56b}$$

$$C_{fz,i} = \frac{(\boldsymbol{\tau}_i \cdot \mathbf{n}_i)_z}{\frac{1}{2}\rho_{\text{ref}} V_{\text{ref}}^2}. \tag{12.56c}$$

where $\boldsymbol{\tau}_i \cdot \mathbf{n}_i$ is the viscous stress acting on the face.

The heat-transfer coefficient is defined as

$$C_{h,i} = \frac{\mathbf{q}_i \cdot \mathbf{n}_i}{\frac{1}{2}\rho_{\text{ref}} V_{\text{ref}}^3}. \tag{12.57}$$

where $\mathbf{q}_i$ is the heat flux for the face.

12.7 Force and Moment Computation

RocfluMP computes forces and moments exerted by the fluid on the patches. The force and moment on a patch are computed from the sum of the forces and moments on the faces of that patch.

The force on a face i with unit normal vector $\mathbf{n}_i$ and area S_i is

$$\mathbf{F}_i = \left[(\rho_i \mathbf{v}_i \cdot \mathbf{n}_i)\, \mathbf{v}_i + (p_i - p_{\text{ref}})\, \mathbf{n}_i - \boldsymbol{\tau}_i \cdot \mathbf{n}_i \right] S_i, \tag{12.58}$$

where ρ_i is the density, $\mathbf{v}_i$ is the velocity vector, p_i is the pressurem, and $\boldsymbol{\tau}_i \cdot \mathbf{n}_i$ is the viscous stress. The force components are given by

$$F_{x,i} = \Big[(\rho_i \mathbf{v}_i \cdot \mathbf{n}_i)\, u_i + (p_i - p_{\text{ref}})\, n_{x,i} - (\boldsymbol{\tau}_i \cdot \mathbf{n}_i)_x \Big] S_i, \tag{12.59a}$$

$$F_{y,i} = \Big[(\rho_i \mathbf{v}_i \cdot \mathbf{n}_i)\, v_i + (p_i - p_{\text{ref}})\, n_{y,i} - (\boldsymbol{\tau}_i \cdot \mathbf{n}_i)_y \Big] S_i, \tag{12.59b}$$

$$F_{z,i} = \Big[(\rho_i \mathbf{v}_i \cdot \mathbf{n}_i)\, w_i + (p_i - p_{\text{ref}})\, n_{z,i} - (\boldsymbol{\tau}_i \cdot \mathbf{n}_i)_z \Big] S_i. \tag{12.59c}$$

The moment about a reference location $\mathbf{r}_{\text{ref}}$ created by the force on a face is

$$\mathbf{M}_i = (\mathbf{r}_i - \mathbf{r}_{\text{ref}}) \times \mathbf{F}_i, \tag{12.60}$$

where $\mathbf{r}_i$ is the position vector of the face centroid. The convention for positive moments is shown in Fig. 12.5: Moments around a given coordinate axis are defined to be positive if they lead to a counter-clockwise rotation when looking in the negative direction of that coordinate axis. The moments components are given by

$$M_{x,i} = F_{z,i}(y_i - y_{\text{ref}}) - F_{y,i}(z_i - z_{\text{ref}}), \tag{12.61a}$$

$$M_{y,i} = F_{x,i}(z_i - z_{\text{ref}}) - F_{z,i}(x_i - x_{\text{ref}}), \tag{12.61b}$$

$$M_{z,i} = F_{y,i}(x_i - x_{\text{ref}}) - F_{x,i}(y_i - y_{\text{ref}}). \tag{12.61c}$$

Non-dimensional force coefficients are defined by

$$C_{F_{x,i}} = \frac{F_{x,i}}{\frac{1}{2}\rho_{\text{ref}} V_{\text{ref}}^2 S_{\text{ref}}} = \left(C_{m,i} \frac{u_i}{V_{\text{ref}}} + C_{p,i} n_{x,i} - C_{fx,i} \right) \frac{S_i}{S_{\text{ref}}}, \tag{12.62a}$$

$$C_{F_{y,i}} = \frac{F_{y,i}}{\frac{1}{2}\rho_{\text{ref}} V_{\text{ref}}^2 S_{\text{ref}}} = \left(C_{m,i} \frac{v_i}{V_{\text{ref}}} + C_{p,i} n_{y,i} - C_{fy,i} \right) \frac{S_i}{S_{\text{ref}}}, \tag{12.62b}$$

$$C_{F_{z,i}} = \frac{F_{z,i}}{\frac{1}{2}\rho_{\text{ref}} V_{\text{ref}}^2 S_{\text{ref}}} = \left(C_{m,i} \frac{w_i}{V_{\text{ref}}} + C_{p,i} n_{z,i} - C_{fz,i} \right) \frac{S_i}{S_{\text{ref}}}, \tag{12.62c}$$

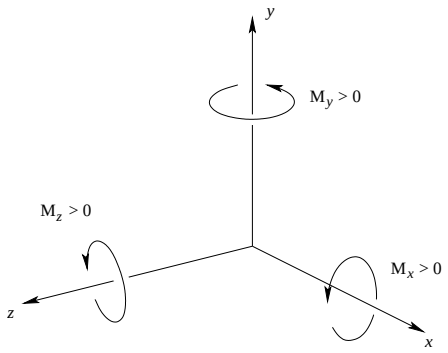

Figure 12.5: Definition for positive moments.

where the mass coefficient $C_{m,i}$ is defined by Eq. (12.54), the pressure coefficient $C_{p,i}$ is defined by Eq. (12.55) and the skin-friction coefficients $C_{fx,i}$, $C_{fy,i}$, and $C_{fz,i}$ are defined by Eqs. (12.56).

Non-dimensional moment coefficients are defined by

$$C_{M_{x,i}} = \frac{M_{x,i}}{\frac{1}{2}\rho_{\text{ref}} V_{\text{ref}}^2 S_{\text{ref}} L_{\text{ref}}} = C_{F_{z,i}} \frac{y_i - y_{\text{ref}}}{L_{\text{ref}}} - C_{F_{y,i}} \frac{z_i - z_{\text{ref}}}{L_{\text{ref}}}, \tag{12.63a}$$

$$C_{M_{y,i}} = \frac{M_{y,i}}{\frac{1}{2}\rho_{\text{ref}} V_{\text{ref}}^2 S_{\text{ref}} L_{\text{ref}}} = C_{F_{x,i}} \frac{z_i - z_{\text{ref}}}{L_{\text{ref}}} - C_{F_{z,i}} \frac{x_i - x_{\text{ref}}}{L_{\text{ref}}}, \tag{12.63b}$$

$$C_{M_{z,i}} = \frac{M_{z,i}}{\frac{1}{2}\rho_{\text{ref}} V_{\text{ref}}^2 S_{\text{ref}} L_{\text{ref}}} = C_{F_{y,i}} \frac{x_i - x_{\text{ref}}}{L_{\text{ref}}} - C_{F_{x,i}} \frac{y_i - y_{\text{ref}}}{L_{\text{ref}}}. \tag{12.63c}$$

The force and moment coefficients for an entire patch are simply given by the summation of the force and moment coefficients for the faces on that patch,

$$C_{F_x} = \frac{1}{S_{\text{ref}}} \sum_i \Big(C_{m,i} \frac{u_i}{V_{\text{ref}}} + C_{p,i} n_{x,i} - C_{fx,i} \Big) S_i, \tag{12.64a}$$

$$C_{F_y} = \frac{1}{S_{\text{ref}}} \sum_i \Big(C_{m,i} \frac{v_i}{V_{\text{ref}}} + C_{p,i} n_{y,i} - C_{fy,i} \Big) S_i, \tag{12.64b}$$

$$C_{F_z} = \frac{1}{S_{\text{ref}}} \sum_i \Big(C_{m,i} \frac{w_i}{V_{\text{ref}}} + C_{p,i} n_{z,i} - C_{fz,i} \Big) S_i, \tag{12.64c}$$

and

$$C_{M_x} = \sum_i \Big(C_{F_{z,i}} \frac{y_i - y_{\text{ref}}}{L_{\text{ref}}} - C_{F_{y,i}} \frac{z_i - z_{\text{ref}}}{L_{\text{ref}}} \Big), \tag{12.65a}$$

$$C_{M_y} = \sum_i \Big(C_{F_{x,i}} \frac{z_i - z_{\text{ref}}}{L_{\text{ref}}} - C_{F_{z,i}} \frac{x_i - x_{\text{ref}}}{L_{\text{ref}}} \Big), \tag{12.65b}$$

$$C_{M_z} = \sum_i \Big(C_{F_{y,i}} \frac{x_i - x_{\text{ref}}}{L_{\text{ref}}} - C_{F_{x,i}} \frac{y_i - y_{\text{ref}}}{L_{\text{ref}}} \Big). \tag{12.65c}$$

Chapter 13

Implementation Details

13.1 NSCBC Implementation

This document describes implementation of Navier-Stokes Characterstics based Boundary Conditions in Rocflu (a three-dimensional time-dependent compressible Navier-Stokes equations solver on moving and/or deforming unstructured grids).

NSCBC implementation would require following procedures:

- Data-structure
- Initialization procedures
- Data read/write routines
- Gradient computation at boundary
- Characteristic wave amplitude computation
- Time integration
- Flux computation using boundary data

following sections would describe each of above in detail

13.1.1 Data-structure

NSCBC involves solving Navier-Stokes equations at boundary. following variables are needed for proper implementation of NSCBC.

- Conservative and Dependent variables are needed at each boundary face. These are updated by solving flow equations at boundary and these describes complete state of flow system at boundary at all time.

- A flag variable, `bcKind`, is needed to determine if a particular boundary is to be treated as NSCBC. NSCBC routines are called if this variable `bcKind == BC_KIND_NSCBC`.
- Boundary condition data is required at each boundary. It is read from user specified boundary condition inputfile containing details of boundary type `bcType,bcName` and other details needed for complete implementation of boundary condition.
- Gradient arrays at boundary. These are computed and stored for each time step.

13.1.2 Initialization procedures

Navier-Stokes equations are initial-boundary value problems (IBV), which require an initial solution which is evolved in time. Flow variables at NSCBC boundaries are initialized using routine `RFLU_BXV_InitVars`, which further calls specific initialization routine depending on type of boundary condition like, `RFLU_NSCBC_InitSlipWall` for slip walls, `RFLU_NSCBC_InitOutflow` for outflow boundary and so on.

13.1.3 Data read/write routines

Boundary variables are written in seperate files than interior solution files. Routines used to read/write boundary solution files are `RFLU_BXV_ReadVarsWrapper/RFLU_BXV_WriteVarsWrapper`. These wrapper routines further call corresponding routines to read/write binary or ASCII datafiles, e.g., reading routines are `RFLU_BXV_WriteVarsASCII` and `RFLU_BXV_WriteVarsBinary`. similarly there are writing routines.

13.1.4 Gradient computation at boundary

Gradients of the primitive variables at boundary are computed using `RFLU_ComputeGradBFacesWrapper`. Gradients are computed using least-square gradient reconstruction technique which need $2d/3d$ boundary face-to-cell stencils. When boundary is aligned with one of axis and interior cells are lying in curvilinear direction normal to boundary face, then $1d$ stencil can be used to compute normal derivatives of variables. This is useful only when there is no flow variation in transverse directions. Stencils are computed using `RFLU_BuildBF2CStencilWrapper` which further calls routines for $1d/2d$ or $3d$ stencil computations. After stencils are computed, weights are computed using `RFLU_ComputeWtsF2CWrapper`. Once stencil and weights are available gradients are computed through `RFLU_ComputeGradBFacesWrapper`, which further uses `RFLU_ComputeGradBFaces_1D` to compute gradient using $1d$ stencils and `RFLU_ComputeGradBFaces` to compute gradients using $2d/3d$ stencils.

13.1.5 Characteristic wave amplitude computation

This is the most important step in implementing NSCBC. Local One-Dimensional Inviscid (LODI) relations are used at boundary face which uses derivatives of flow variables in

face-normal direction. There is need to resolve x, y, z components of gradients in normal and transverse directions at boundary face. Time derivatives of conservative variables are expressed in terms of wave amplitudes (incoming or outgoing wave) and space derivatives in transverse directions. This task is carried out in `RFLU_NSCBC_CompRhs`, which loops over each boundary patch and depending on boundary type a specific subroutine is called to compute right hand side of system of equation at boundary, e.g., `RFLU_NSCBC_CompRhsOutflow` for outflow, `RFLU_NSCBC_CompRhsFarf` for farfield boundary condition and so on.

13.1.6 Time integration

RK-4 is used for time integration. It is point quantity at boundary face (unlike averaged cell quantity in interior domain), which is being solved for. routine for carrying out RK-4 integration is `RungeKuttaMP` which calls `RKInitMP` and `RKUpdateMP`.

13.1.7 Flux computation using boundary data

Flux computation at the boundary faces is done making use of boundary variables and interior cell variables. Convective flux computation using second-order accurate approximation for flow variables are carried out using `RFLU_NSCBC_CompSecondPatchFlux` subroutine and `RFLU_NSCBC_CompFirstPatchFlux` can be used for first-order approximation.

figure 13.1 describes NSCBC implementation in the form of a flowchart.

13.2 Parallelization

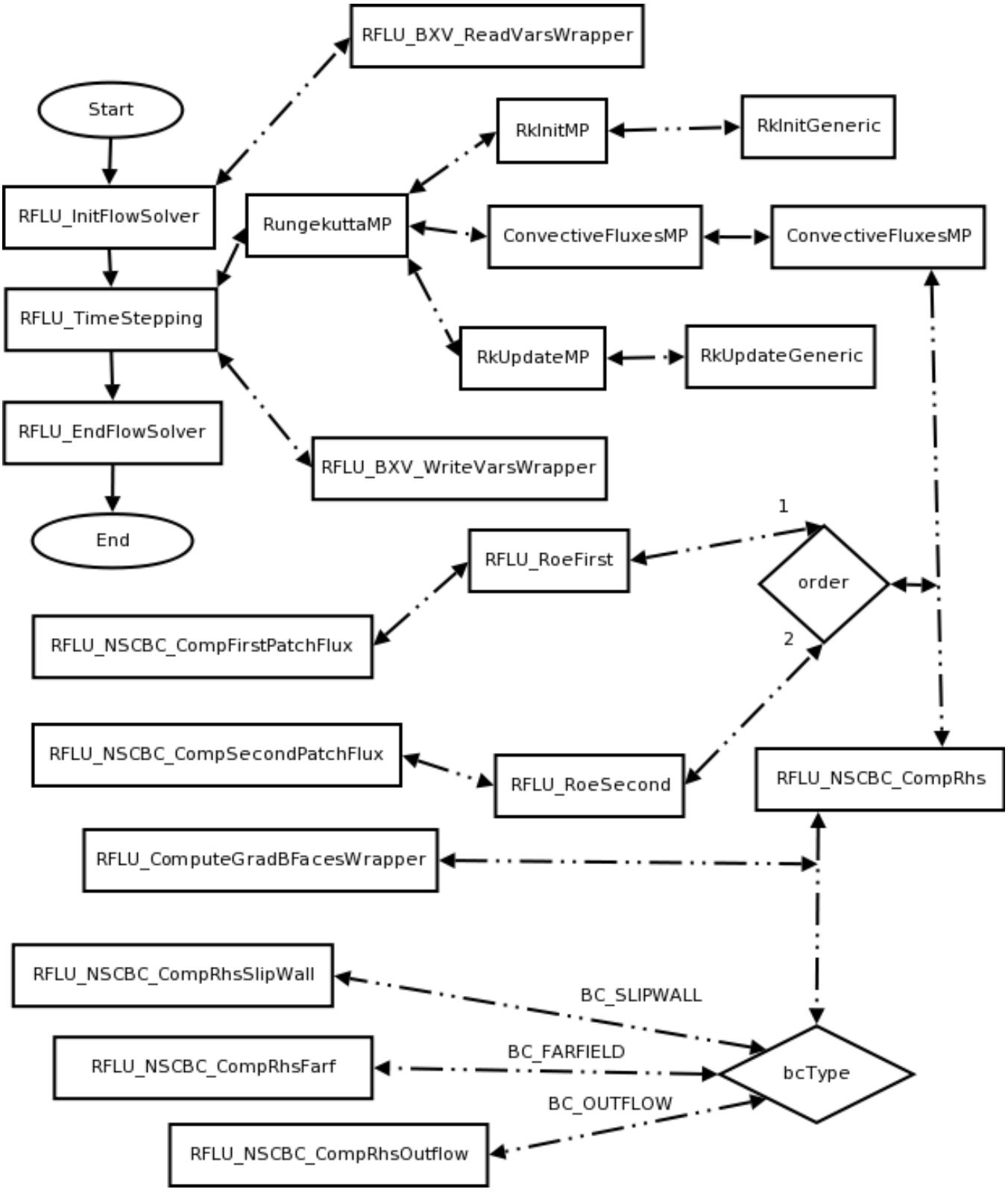

Figure 13.1: NSCBC implementation in Rocflu

Chapter 14

Data Structures

RocfluMP makes extensive use of Fortran 90 user-defined types for the definition of data structures.

14.1 Philosophy and Abstraction

The top layer of the data structure developed for RocfluMP is depicted schematically in Fig. 14.1.

The top layer of the data structure consists of two main layers of abstraction:

1. At the highest layer are multigrid levels constructed from the finest grid. Each layer can contain an arbitrary number of regions.

2. At the second level are solution region. A solution region is defined as the entire solution region for sequential processing applications, or a single partition for parallel processing applications. Each multigrid level can consist of an arbitrary number of domains.

Note that the multigrid levels are located atop the solution regions. This means that each multigrid level is partitioned separately for parallel processing. Because intra-layer commu-

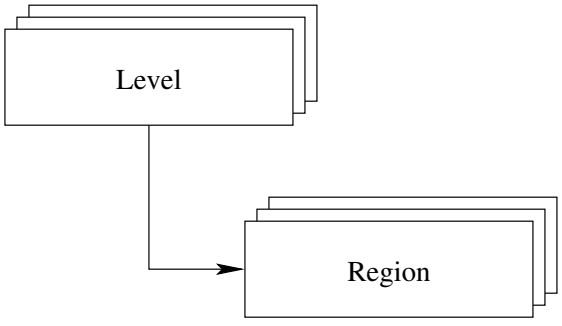

Figure 14.1: Overview of data-structure layers.

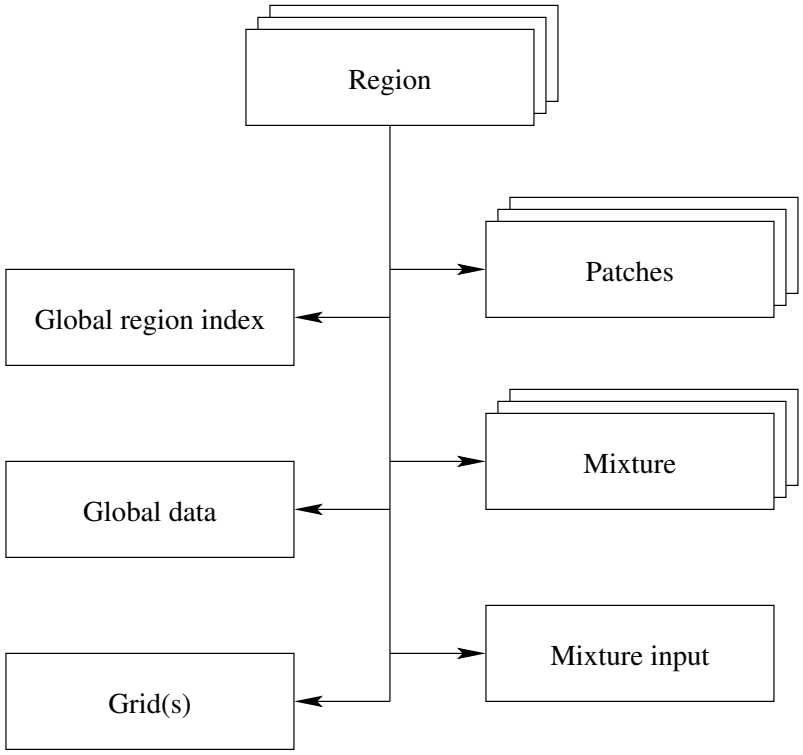

Figure 14.2: Overview of region data structure.

nication is more important than inter-level communication, the possibility of optimizing each multigrid level for load balancing separately should allow for better parallel performance.

The types `t_level` and `t_region` are defined in `ModDataStruct.F90`.

14.2 Region Data Structure

The region data structure contains all the information required to solve the governing equations in a given region. A schematic overview of the region data structure is given in Fig. 14.2.

The region data structure itself is defined to be an array. This gives additional flexibility in allowing several regions to be assigned to a single processor.

The components of the user-defined data type `t_region` are defined as follows:

`iRegionGlobal` is the global index of the local region.

`irkStep` is the index of the Runge-Kutta step.

`fieldFlagMixt` is a field flag used to communicate the conserved variables for parallel calculations using the FEM framework.

`dt` contains the timestep.

```
TYPE t_region
  INTEGER :: iRegionGlobal,irkStep
  INTEGER :: fieldFlagMixt

  REAL(RFREAL), POINTER :: dt(:)

  TYPE(t_grid) :: grid,gridOld
  TYPE(t_mixt) :: mixt
  TYPE(t_turb) :: turb
  TYPE(t_spec) :: spec
  TYPE(t_radi) :: radi
  TYPE(t_peul) :: peul

  TYPE(t_plag)  , POINTER :: plags(:)
  TYPE(t_patch) , POINTER :: patches(:)
  TYPE(t_global), POINTER :: global

  TYPE(t_mixt_input) :: mixtInput
  TYPE(t_turb_input) :: turbInput
  TYPE(t_spec_input) :: specInput
  TYPE(t_peul_input) :: peulInput
  TYPE(t_plag_input) :: plagInput
  TYPE(t_radi_input) :: radiInput
END TYPE t_region
```

Figure 14.3: Definition of region data structure.

`grid` is the user-defined data type containing all the information relating to the grid. See Sect. 14.3 for a description of `t_grid`.

`gridOld` is the user-defined data type containing all the information relating to the old grid when using grid motion. See Sect. 14.3 for a description of `t_grid`.

`mixt` is the user-define data type containing all the information relating to mixture. It is described in Sec. 14.5.2.

`patches` is the user-defined data type containing all the information relating to boundary patches. See Sect. 14.4 for a description of `t_patch`.

`global` is a pointer to global data.

`mixtInput` is a user-defined data type containing all the user-defined input for the solution of the mixture equations. It is described in Sec. 14.5.1.

14.3 Grid Data Structure

The grid data structure contains information relating to the description of the grid. An overview of the grid data structure is given in Fig. 14.4. The grid data structure is defined in `ModGrid.F90`. Some additional (RocfluMP-specific) data is defined in `RFLU_ModGrid.F90`, which is mainly used in converting from exterior grid formats to that used by RocfluMP, and in helping to construct some data structures. The two modules are discussed in detail below.

14.3.1 Module `ModGrid.F90`

In understanding the grid data structure, the following points are important:

1. RocfluMP can operate on grids consisting of arbitrary combinations of tetrahedral, hexahedral, prismatic, and pyramidal cells. As indicated in Sec. 2.1, these are referred to as instances of different *cell types*. When running RocfluMP in parallel, one also has to distinguish between actual and virtual cells, so RocfluMP introduces the concept of a *cell kind* to distinguish between these.

2. RocfluMP categorizes faces according to types and kinds also. A face can be of triangular of quadrilateral type, and can be of different kinds depending on whether the adjacent cells are actual or virtual ones, and whether the face is on a boundary.

3. Since RocfluMP is based on the cell-centered method, the computation of fluxes is most easily carried out by looping over faces of the grid. Because boundary conditions are conveniently enforced by modifying the computation of fluxes on boundary patches, the grid data structure only stores internal faces, i.e., faces which do not lie on boundary patches.

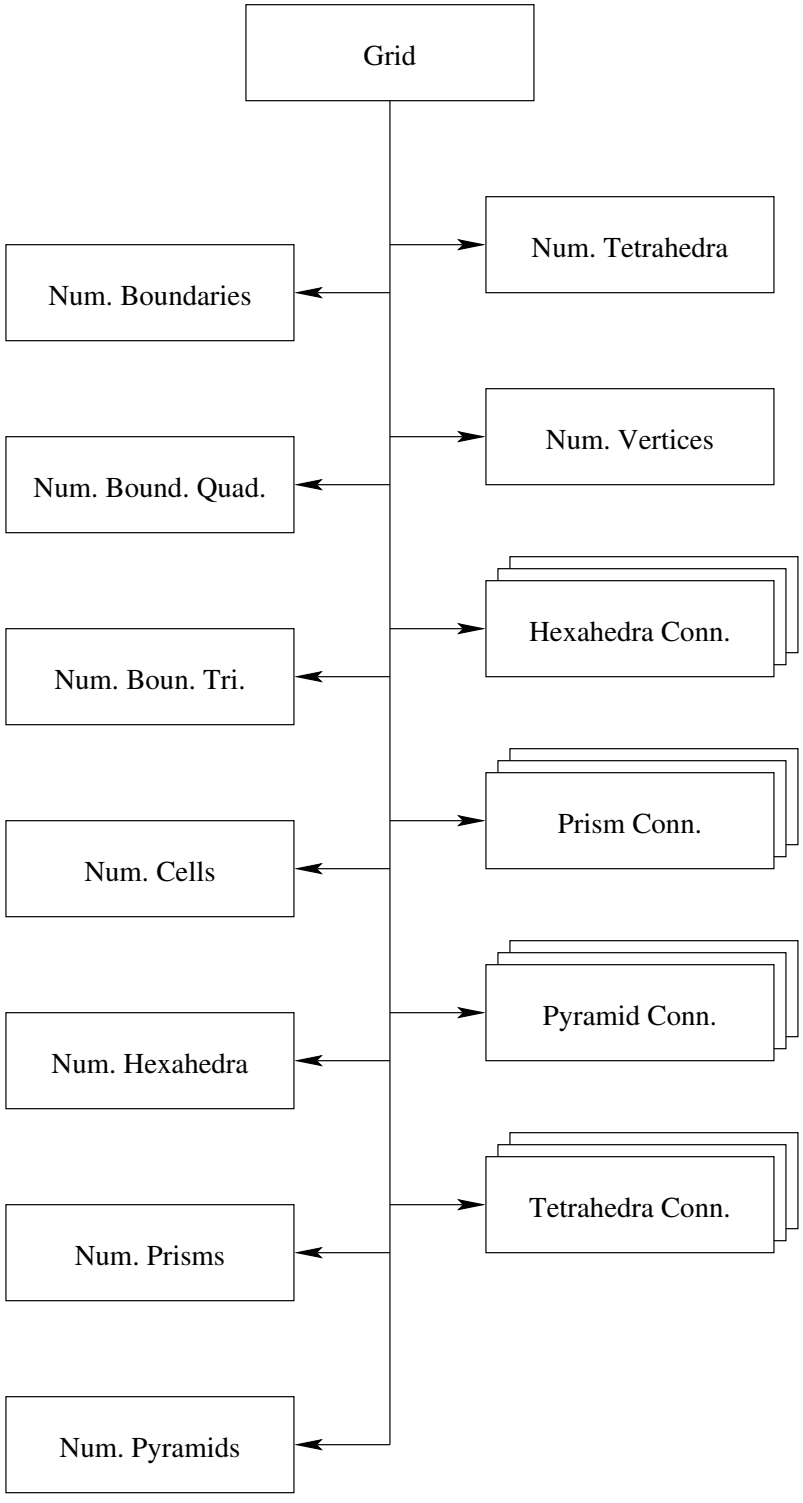

Figure 14.4: Overview of grid data structure.

```
  TYPE t_grid
! - Basic grid quantities -------------------------------------------------

    INTEGER :: indGs,nBFaces,nBQuads,nBTris,nCells,nCellsTot,nEdges, &
               nEdgesEst,nEdgesTot,nFaces,nFacesEst,nFacesTot,nHexs, &
               nHexsTot,nPatches,nPris,nPrisTot,nPyrs,nPyrsTot, &
               nTets,nTetsTot,nVert,nVertTot
    INTEGER, DIMENSION(:), POINTER :: hexFlag,hex2CellGlob, &
                                      priFlag,pri2CellGlob,pyrFlag, &
                                      pyr2CellGlob,tetFlag,tet2CellGlob, &
                                      vertFlag,v2c
    INTEGER, DIMENSION(:,:), POINTER :: cellGlob2Loc,c2cs,e2v,e2vTemp,f2c, &
                                        f2cTemp,f2cs,f2v,f2vTemp,hex2v,pri2v, &
                                        pyr2v,tet2v,v2cInfo
    REAL(RFREAL), DIMENSION(:,:), POINTER :: fc,fn
    REAL(RFREAL), DIMENSION(:,:,:), POINTER :: cgwt,fgwt

! - Grid Motion ------------------------------------------------------------

    INTEGER, DIMENSION(:), POINTER :: degr
    REAL(RFREAL), DIMENSION(:), POINTER :: gs,volMin
    REAL(RFREAL), DIMENSION(:,:), POINTER :: rhs

! - Geometric information --------------------------------------------------

    REAL(RFREAL), POINTER :: xyz(:,:)
    REAL(RFREAL), POINTER :: vol(:),cofg(:,:)
  END TYPE t_grid
```

Figure 14.5: Definition of grid data structure.

The components of the user-defined type `t_grid` are defined as follows:

`indGs` is a flag used to allocate the array for the grid speeds. If grid motion is active, the grid speeds need to be computed, and hence `indGs=1`, otherwise `indGs=0`. This allows the grid speed array `gs` (see below) to be accessed even if grid motion is not active, which simplifies the code because conditional statements can be avoided. The array `gs` will typically be accessed through a statement such as `gs(indGs*ifc)`, where `ifc` is an integer variable used in a loop over interior faces.

`nBFaces` is the total number of triangular and quadrilateral faces on all boundary patches.

`nBQuads` is the total number of quadrilateral faces on all boundary patches.

`nBTris` is the total number of triangular faces on all boundary patches.

`nCells` is the number of interior cells in the grid.

`nCellsTot` is the total number of cells in the grid, i.e., interior and dummy cells.

`nEdges` is the number of interior edges in the grid.

`nEdgesEst` is the estimated total number of edges in the grid. It is used only in the construction of the edge list.

`nEdgesTot` is the total number of edges in the grid, i.e., interior and dummy edges.

`nFaces` is the number of interior triangular and quadrilateral faces in the grid.

`nFacesEst` is the estimated total number of interior triangular and quadrilateral faces in the grid. It is used only in the construction of the face list.

`nFacesTot` is the total number of triangular and quadrilateral faces in the grid, i.e., interior and dummy faces.

`nHexs` is the number of interior hexahedral cells in the grid.

`nHexsTot` is the total number of hexahedral cells in the grid, i.e., interior and dummy hexahedral cells.

`nPris` is the number of prismatic cells in the grid.

`nPrisTot` is the total number of prismatic cells in the grid, i.e., interior and dummy prismatic cells.

`nPyrs` is the number of pyramidal cells in the grid.

`nPyrsTot` is the total number of pyramidal cells in the grid, i.e., interior and dummy pyramidal cells.

`nTets` is the number of tetrahedral cells in the grid.

`nTetsTot` is the total number of tetrahedral cells in the grid, i.e., interior and dummy tetrahedral cells.

`nVert` is the number of vertices in the grid.

`nVertTot` is the total number of vertices in the grid, i.e., interior and dummy vertices.

`hexFlag` contains a flag indicating the kind of a given hexahedral cell. It is read in from the RocfluMP grid file, and can only take the values: `CELL_KIND_BNDRY`, `CELL_KIND_ACTUAL`, and `CELL_KIND_VIRTUAL` (defined in `ModParameters.F90`).

`hex2CellGlob` contains the mapping of a given hexahedral cell to a global cell.

`priFlag` contains a flag indicating the kind of a given prismatic cell. It is read in from the RocfluMP grid file, and can only take the values: `CELL_KIND_BNDRY`, `CELL_KIND_ACTUAL`, and `CELL_KIND_VIRTUAL` (defined in `ModParameters.F90`).

`pri2CellGlob` contains the mapping of a given prismatic cell to a global cell.

`pyrFlag` contains a flag indicating the kind of a given pyramidal cell. It is read in from the RocfluMP grid file, and can only take the values: `CELL_KIND_BNDRY`, `CELL_KIND_ACTUAL`, and `CELL_KIND_VIRTUAL` (defined in `ModParameters.F90`).

`pyr2CellGlob` contains the mapping of a given pyramidal cell to a global cell.

`tetFlag` contains a flag indicating the kind of a given tetrahedral cell. It is read in from the RocfluMP grid file, and can only take the values: `CELL_KIND_BNDRY`, `CELL_KIND_ACTUAL`, and `CELL_KIND_VIRTUAL` (defined in `ModParameters.F90`).

`tet2CellGlob` contains the mapping of a given tetrahedral cell to a global cell.

`vertFlag` contains a flag indicating the kind of a given vertex. It is read in from the RocfluMP grid file, and can only take the values `VERT_KIND_ACTUAL` and `VERT_KIND_VIRTUAL` (defined in `ModParameters.F90`).

`cellGlob2Loc` contains the mapping of a global cell to the local cell of a given type.

`c2cs` contains the cell stencils for each cell. This array is used in computing cell gradients and averaged variables.

`e2v` contains the two vertices defining an edge. This array is used only for grid motion.

`e2vTemp` is a temporary array used to construct `e2v`.

`f2c` contains the two cells adjacent to a face.

`f2cTemp` is a temporary array used to construct `f2c`.

`f2cs` contains the face stencils for each face. This array is used in computing face gradients.

`f2v` contains the vertices defining a face.

`f2vTemp` is a temporary array used to construct `f2v`.

`hex2v` contains the connectivity information for the hexahedral cells. The vertices must be numbered as shown in Fig. 14.6(a). The face to vertex, edge to vertex, and edge to face connectivity arrays for hexahedral cells are shown in Table 14.1.

`pri2v` contains the connectivity information for the prismatic cells. The vertices must be numbered as shown in Fig. 14.6(b). The face to vertex, edge to vertex, and edge to face connectivity arrays for prismatic cells are shown in Table 14.2.

`pyr2v` contains the connectivity information for the pyramidal cells. The vertices must be numbered as shown in Fig. 14.6(c). The face to vertex, edge to vertex, and edge to face connectivity arrays for pyramidal cells are shown in Table 14.3.

`tet2v` contains the connectivity information for the tetrahedral cells. The vertices must be numbered as shown in Fig. 14.6(d). The face to vertex, edge to vertex, and edge to face connectivity arrays for tetrahedral cells are shown in Table 14.4.

`fc` contains the face-centroid coordinates.

`fn` contains the components of the face-normal unit vector and the area of the face.

`cgwt` contains the cell-gradient weights.

`fgwt` contains the face-gradient weights.

`degr` contains the degree of each vertex.

`gs` contains the grid speed of each face.

`volMin` contains the minimum volume of all cells incident to a vertex. The variable is used in altering the effect of the grid-smoothing algorithm to avoid inverting cells and hence negative volumes.

`rhs` contains the residual for grid smoothing.

`xyz` contains the x-, y-, and z-coordinates of the vertices.

`vol` contains the volumes of the cells.

`cofg` contains the centroids of the cells.

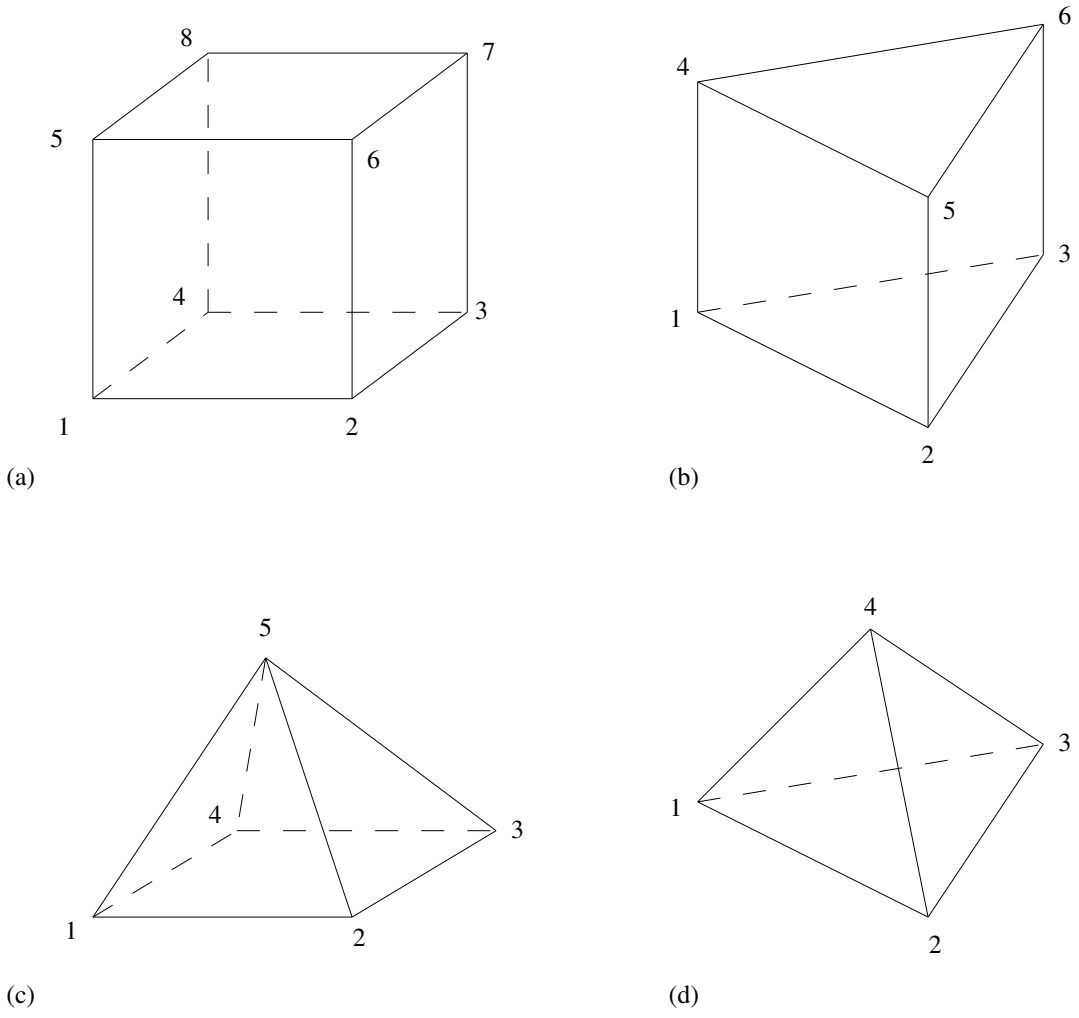

Figure 14.6: Definition of cell connectivity.

Face	Vertices			
1	1	4	3	2
2	1	2	6	5
3	2	3	7	6
4	3	4	8	7
5	1	5	8	4
6	5	6	7	8

Table 14.1: Face-to-vertex connectivity arrays for hexahedral cells.

Face	Vertices			
1	1	3	2	
2	1	2	5	4
3	2	3	6	5
4	1	4	6	3
5	4	5	6	

Table 14.2: Face-to-vertex connectivity arrays for prismatic cells.

Each cell type has not only a clearly defined numbering of its vertices, but also for its edges and faces. These numberings are listed in Tables 14.1-14.4. In reading these tables, it is to be understood that edges and faces have an orientation. This is a crucial point if the routines which construct data structures are to be understood properly. Therefore, the rows in these tables are to be read only from left to right. Thus, edge 10 of an hexahedron is pointing from vertex 2 to vertex 6. Furthermore, faces are oriented such that their normal vectors are pointing out of the cell. This corresponds to anti-clockwise ordering of the vertices when viewing the face of a cell from the outside of that cell, and to clockwise ordering when viewing the face through the cell.

Face	Vertices			
1	1	4	3	2
2	1	2	5	
3	2	3	5	
4	3	4	5	
5	1	5	4	

Table 14.3: Face-to-vertex connectivity arrays for pyramidal cells.

Face	Vertices		
1	1	2	3
2	2	4	3
3	1	3	4
4	1	4	2

Table 14.4: Face-to-vertex connectivity arrays for tetrahedral cells.

Figure 14.7: Overview of boundary data structure.

14.3.2 Module `RFLU_ModGrid.F90`

The module `RFLU_ModGrid.F90` contains some data structures used in the conversion of exterior grid formats to that used in RocfluMP, and some data structures used in the generation of other data structures.

14.4 Boundary Data Structure

An overview of the boundary data structure is given in Fig. 14.7. The definition of the boundary data structure is shown in Fig. 14.8.

The components of the user-defined type for the boundary data structure are defined as follows:

`bcType` is the type of the boundary patch. It is used to identify which boundary conditions is to be set on that boundary patch.

`bcCoupled` is a flag indicating whether the boundary patch is coupled to another code. It can have the values `BC_NOT_COUPLED`, `BC_NOT_BURNING`, and `BC_BURNING` (defined in `ModParameters.F90`).

`iPatchGlobal` is the global index of the boundary patch. For serial computations, `iPatchGlobal` is equal to the index of the boundary patch. The variable is needed to access the correct boundary condition information when reading the boundary condition file.

`nBFaces` is the total number of triangular and quadrilateral faces on a boundary patch.

`nBTris` is the number of triangular faces on a boundary patch, and is read from the grid file.

`nBQuads` is the number of quadrilateral faces on a boundary patch, and is read from the grid file.

`nBVert` is the number of vertices on a boundary patch.

```
  TYPE t_patch
    INTEGER :: bcType,bcCoupled
    INTEGER :: iPatchGlobal
    INTEGER :: nBFaces,nBTris,nBQuads,nBVert,nBVertEst
    INTEGER, DIMENSION(:), POINTER :: bf2bg,bf2c,bv,bvTemp
    INTEGER, DIMENSION(:,:), POINTER :: bf2cs,bf2ct,bf2v,bf2vl,bTri2v, &
                                        bTri2vl,bQuad2v,bQuad2vl
    LOGICAL :: movePatch,smoothGrid
    REAL(RFREAL), DIMENSION(:), POINTER :: gs
    REAL(RFREAL), DIMENSION(:,:), POINTER :: bvn,dXyz,fc,fn
    REAL(RFREAL), DIMENSION(:,:,:), POINTER :: bfgwt
    CHARACTER*(CHRLEN) :: bcName
#ifdef GENX
    INTEGER, DIMENSION(:), POINTER :: bFlag,bcFlag
    REAL(RFREAL), DIMENSION(:), POINTER :: mdotAlp,pf,qc,qr,rhofAlp,tempf, &
                                           tflmAlp
    REAL(RFREAL), DIMENSION(:,:), POINTER :: duAlp,nfAlp,rhofvfAlp,tracf, &
                                             xyz
#endif
    TYPE(t_bcvalues) :: valMixt,valTurb,valSpec,valPeul,valRadi
    TYPE(t_bcvalues), POINTER :: valPlag(:)
    TYPE(t_tile_plag) :: tilePlag
    TYPE(t_buffer_plag) :: bufferPlag
  END TYPE t_patch
```

Figure 14.8: Definition of boundary data structure.

`nBVertEst` is the estimated number of vertices on a boundary patch. It is used only in the construction of the boundary vertex list.

`bf2bg` is an access array which maps a face of a patch to an address in the array `bGradFace` contained in the mixture data type `t_mixt`. It is needed because the boundary face gradients of all boundary patches are stored in a single array for convenience.

`bf2ct` is an access array which maps a given boundary face to a cell type. It can only take the values `CELL_TYPE_TET`, `CELL_TYPE_HEX`, `CELL_TYPE_PRI`, and `CELL_TYPE_PYR` (defined in `ModParameters.F90`).

`bf2v` contains the vertices defining a boundary patch face.

`bf2vl` contains the vertices defining a boundary patch face, locally numbered for each boundary patch.

`bTri2v` contains the vertices defining a triangular boundary patch face. The vertices are oriented such that the normal vector is pointing out of the computational domain.

`bTri2vl` contains the vertices defining a triangular boundary patch face, locally numbered for each boundary patch. The vertices are oriented such that the normal vector is pointing out of the computational domain.

`bQuad2v` contains the vertices defining a quadrilateral boundary patch face. The vertices are oriented such that the normal vector is pointing out of the computational domain.

`bQuad2vl` contains the vertices defining a quadrilateral boundary patch face, locally numbered for each boundary patch. The vertices are oriented such that the normal vector is pointing out of the computational domain.

`movePatch` is a logical variable indicating whether the boundary patch is moving.

`smoothGrid` is a logical variable indicating whether the boundary patch grid is to be smoothed.

`gs` contains the grid speed of each boundary patch face.

`bvn` contains the components of the unit normal vector at the boundary patch vertices.

`dXyz` contains the imposed displacement of the boundary vertices.

`fc` contains the face-centroid coordinates.

`fn` contains the components of the face-normal unit vector and the area of the face.

`bfgwt` contains the face-gradient weights.

`bcName` contains the name of the boundary patch.

`bFlag` is a flag whether a burning face has ignited or not when running RocfluMP inside GENx. This is used to avoid faces which have ignited from extinguishing. (Used only if `GENX=1`.)

`bcFlag` is a flag indicating the type of interaction with other codes when running RocfluMP inside GENx. It can only assume the values `BC_NOT_COUPLED`, `BC_NOT_BURNING`, and `BC_BURNING` (defined in `ModParameters.F90`). (Used only if `GENX=1`.)

`mdotalp` contains the mass flux for each boundary patch face. It is allocated only for burning boundary patches. (Used only if `GENX=1`.)

`pf` contains the face pressure. (Used only if `GENX=1`.)

`qc` contains the convective heat flux. It is allocated only for burning boundary patches. (Used only if `GENX=1`.)

`qr` contains the radiative heat flux. It is allocated only for burning boundary patches. (Used only if `GENX=1`.)

`rhofalp` contains the fluid density for each boundary patch face. (Used only if `GENX=1`.)

`tempf` contains the fluid temperature for each boundary patch face. It is allocated only for burning boundary patches. (Used only if `GENX=1`.)

`tflmAlp` contains the static temperature of the injected fluid. It is allocated only for burning boundary patches. (Used only if `GENX=1`.)

`duAlp` contains the incremental displacement. (Used only if `GENX=1`.)

`nfAlp` contains the components of the unit face-normal vector. (Used only if `GENX=1`.)

`rhofvfAlp` contains the components of the product of the fluid density times the fluid velocity. Note that this is *not* the same as `mdotalp`, as the former also includes the effect of boundary motion due to deformation. (Used only if `GENX=1`.

`tracf` contains the fluid traction for each boundary patch face. (Used only if `GENX=1`.)

`xyz` contains the x-, y-, and z-coordinates of the vertices. (Used only if `GENX=1`.)

`valMixt` contains the user-specified values for the enforcement of boundary conditions on the mixture.

```
  TYPE t_mixt_input

    INTEGER :: flowModel
    LOGICAL :: moveGrid, externalBc
    INTEGER :: nDv, nTv, nGv, nGrad, indCp, indMol
    REAL(RFREAL) :: prLam, prTurb, scnLam, scnTurb

! - turbulence modeling

    INTEGER :: turbModel

! - species

    INTEGER :: specModel

! - continuum particles

    LOGICAL :: conPartUsed

! - discrete particles

    LOGICAL :: disPartUsed

! - radiation

    LOGICAL :: radiUsed

! - numerics

    INTEGER      :: spaceDiscr, spaceOrder, pSwitchType
    INTEGER      :: timeScheme, nrkSteps, ldiss(5)
    REAL(RFREAL) :: cfl, smoocf, vis2, vis4, pSwitchOmega, limfac, epsentr
    REAL(RFREAL) :: ark(5), grk(5), trk(5), betrk(5)

! - flow initialization (used within preprocessor)

    REAL(RFREAL) :: iniVelX, iniVelY, iniVelZ, iniPress, iniDens

! - flow initialization (for uniform flow preservation check)

    REAL(RFREAL) :: unifDens, unifEner, unifMomX, unifMomY, unifMomZ, unifPres

  END TYPE t_mixt_input
```

Figure 14.9: Definition of data type `t_mixt_input`.

14.5 Mixture Data Structure

14.5.1 Data Type t_mixt_input

flowModel is a flag indicating which flow model is used. It can only take the values FLOW_EULER or FLOW_NAVST (defined in ModParameters.F90).

moveGrid is a logical variable indicating whether the volume grid is to be moved. Note that the movement of interior points does not necessarily have to be activated when boundary patches are moving.

nDv contains the number of dependent variables. It is used to determine the size of the array dv (see below).

nTv contains the number of transport variables. It is used to determine the size of the array tv (see below).

nGv contains the number of gas variables. It is used to determine the size of the array gv (see below).

indCp is a flag used to allocate the array for specific heat in the gas-variable array. If the specific heat is to vary in space, indCp=1, otherwise indCp=0. This allows the gas-variable array gv (see below) to be accessed even if the specific heat is constant, which simplifies the code because conditional statements can be avoided.

indMol is a flag used to allocate the array for molar mass in the gas-variable array. If the molar mass is to vary in space, indMol=1, otherwise indMol=0. This allows the gas-variable array gv (see below) to be accessed even if the molar mass is constant, which simplifies the code because conditional statements can be avoided.

prLam contains the value of the laminar Prandtl number.

prTurb contains the value of the turbulent Prandtl number.

scnLam contains the value of the laminar Schmidt number.

scnTurb contains the value of the turbulent Schmidt number.

turbModel is a flag indicating which turbulence model is used.

specModel is a flag indicating which gas model is used. Currently, it can only take the value SPEC_MODEL_NONE.

conPartUsed is a logical variable indicating whether continuum particles are used.

disPartUsed is a logical variable indicating whether discrete particles are used.

radiUsed is a logical variable indicating whether radiation modeling is used.

spaceDiscr is a flag indicating which spatial discretization model is used. It can only take the values DISCR_UPW_ROE or DISCR_OPT_LES.

spaceOrder is a flag indicating the order of accuracy of the spatial discretization. Currently, it can only take the value DISCR_ORDER_1.

nrkSteps is a flag indicating the number of steps of the explicit-multistage or the Runge-Kutta scheme.

ldiss is a flag indicating whether the dissipation terms are to be computed in a given stage of the explicit multistage scheme.

cfl contains the value of the CFL number.

epsentr contains the value of the constant in the entropy fix.

ark contains coefficients used in the explicit-multistage and the Runge-Kutta scheme.

grk contains coefficients used in the Runge-Kutta scheme.

trk contains coefficients used in the Runge-Kutta scheme.

betrk contains coefficients used in the explicit-multistage scheme.

iniVelX contains the x-component of the velocity vector for the initial condition. It is only used in rfluprep and written into the solution file.

iniVelY contains the y-component of the velocity vector for the initial condition. It is only used in rfluprep and written into the solution file.

iniVelZ contains the z-component of the velocity vector for the initial condition. It is only used in rfluprep and written into the solution file.

iniPress contains the static pressure for the initial condition. It is only used in rfluprep and written into the solution file.

iniDens contains the density for the initial condition. It is used only in rfluprep and written into the solution file.

unifDens contains the density value when checking RocfluMP for uniform flow preservation. The check for uniform flow preservation is activated by compiling RocfluMP with CHECK_UNIFLOW=1.

unifEner contains the total internal energy value when checking RocfluMP for uniform flow preservation. The check for uniform flow preservation is activated by compiling RocfluMP with CHECK_UNIFLOW=1.

`unifMomX` contains the x-component of momentum when checking RocfluMP for uniform flow preservation. The check for uniform flow preservation is activated by compiling RocfluMP with `CHECK_UNIFLOW=1`.

`unifMomY` contains the y-component of momentum when checking RocfluMP for uniform flow preservation. The check for uniform flow preservation is activated by compiling RocfluMP with `CHECK_UNIFLOW=1`.

`unifMomZ` contains the z-component of momentum when checking RocfluMP for uniform flow preservation. The check for uniform flow preservation is activated by compiling RocfluMP with `CHECK_UNIFLOW=1`.

14.5.2 Data Type `t_mixt`

The data type `t_mixt` contains data related to the mixture and the solution of the associated transport equations. The variables associated with the mixture are divided into several types:

1. *Conserved variables*, i.e., dependent variables for which transport equations are solved, are stored in the array `cv`. For RocfluMP, the conserved variables are $\{\rho, \rho u, \rho v, \rho w, \rho E\}^t$.

2. *Dependent variables*, i.e., dependent variables for which no transport equations are solved, are stored in the array `dv`. For RocfluMP, the dependent variables are $\{p, T, c\}^t$.

3. *Transport variables*, i.e., dependent variables such as the coefficients of viscosity and conductivity.

4. *Gas variables*, i.e., dependent variables such as the specific heat at constant pressure and the molar mass.

The dependent, transport, and gas variables are updated after the update of the conserved variables by calling the routine `mixtureProperties.F90`.

Because it is convenient to work with different state variables at times, RocfluMP provides routines to change the "state" of the state vector from conserved variables to two different sets of primitive variables. This is advantageous when computing gradients for the viscous fluxes and printing information on the solution. The possible states are as follows:

1. Conserved variables given by $\{\rho, \rho u, \rho v, \rho w, \rho E\}^t$. This is the default state and indicated by `cvState` having the value `CV_MIXT_STATE_CONS`. The value of the integer parameter `CV_MIXT_STATE_CONS`, and the corresponding parameters for the other states, is defined in `ModParameters.F90`.

2. Primitive variables given by $\{\rho, u, v, w, p\}^t$. This state is indicated by `cvState` having the value `CV_MIXT_STATE_DUVWP`.

3. Primitive variables given by $\{\rho, u, v, w, T\}^t$. This state is indicated by `cvState` having the value `CV_MIXT_STATE_DUVWT`.

```
  TYPE t_mixt
    REAL(RFREAL), POINTER :: cv(:,:), cvOld(:,:), dv(:,:), tv(:,:), gv(:,:)
#ifdef STATS
    REAL(RFREAL), POINTER :: tav(:,:)
#endif
    REAL(RFREAL), POINTER :: rhs(:,:), rhsSum(:,:), diss(:,:), fterm(:,:)
    INTEGER :: cvState
    REAL(RFREAL), DIMENSION(:,:), POINTER :: cvVrtx
    REAL(RFREAL), DIMENSION(:,:,:), POINTER :: bGradFace,gradCell,gradFace
  END TYPE t_mixt
```

Figure 14.10: Definition of data type `t_mixt`.

Changes of the state are effected by `USE`ing the module `RFLU_ModConvertCv.F90`, and calling the routines:

`RFLU_ConvertCvCons2Prim(pRegion,cvStateFuture)` to convert from conserved variables to primitive variables. `cvStateFuture` must be set to either `CV_MIXT_STATE_DUVWP` or `CV_MIXT_STATE_DUVWT`; any other value will generate an error. An error will also be generated if `cvState` is not equal to `CV_MIXT_STATE_CONS`.

`RFLU_ConvertCvPrim2Cons(pRegion,cvStateFuture)` to convert from a primitive variable state to conserved variables. `cvStateFuture` must be set to `CV_MIXT_STATE_CONS`; any other value will generate an error. An error will also be generated if `cvState` is equal to `CV_MIXT_STATE_CONS`.

The strict checking of `cvState` upon calling the conversion routines is carried out to catch programming errors, where the state was changed on entering a routine, but not changed back on exiting the routine. Additional statements such as

```
IF ( pRegion%mixt%cvState == CV_MIXT_STATE_CONS ) THEN
  CALL ErrorStop(global,ERR_CV_STATE_INVALID,__LINE__)
END IF ! region
```

may be placed at the top of routines to catch such errors.

The data defined in the data type `t_mixt` is shown in Fig. 14.10, and explained in detail below.

`cv` contains the vector of conserved variables.

`cvOld` contains the vector of old conserved variables, i.e., from a previous timestep.

`dv` contains the vector of dependent variables.

`tv` contains the vector of transport variables.

`gv` contains the vector of gas variables.

`rhs` contains the residual vector.

`rhsSum` contains a weighted sum of residual vectors for the Runge-Kutta scheme.

`diss` contains the residual vector due to dissipative terms of the spatial discretization.

`cvState` is a flag indicating which state is stored in the conservative state vector. It can only take the values `CV_MIXT_STATE_CONS`, `CV_MIXT_STATE_DUVWP`, and `CV_MIXT_STATE_DUVWT`.

`cvVrtx` contains the state vector at the vertices. It is used only in rflupost after having interpolated the cell-centered values to the vertices.

`bGradFace` contains the boundary-face gradients. It is accessed using the array `bf2bg` in the data type `t_patch`.

`gradCell` contains the cell gradients.

`gradFace` contains the face gradients.

Chapter 15

GENx Integration

15.1 Definition of Attributes

15.1.1 Grid Attributes

15.1.1.1 Volume Panes

The volume-grid pane attributes are defined in Table 15.1.

15.1.1.2 Surface Panes

The surface-grid pane attributes are defined in Table 15.2. Note that multiple connectivity lists exist for each face type, e.g., `:t3:real` and `:t3g:real`. The former is required by Roccom and its entries are used to access the coordinate array `nc` on each surface pane. The latter is required by rflump and its entries are used to access the coordinate array `nc` on the volume pane.

15.1.2 Solution Attributes

The solution attributes are defined in Table 15.3.

15.1.3 Interface Attributes

15.1.3.1 Non-Interacting Panes

15.1.3.2 Non-Burning Panes

The attributes of non-burning panes are defined in Table 15.4.

15.1.3.3 Burning Panes

The attributes of burning panes are defined in Table 15.5.

Table 15.1: Definitions of volume-grid pane attributes.

Attribute	Meaning	Units	Type
`nc`	Coordinates	m	Real
`pconn`	Pane connectivity	–	Integer
`:T4:real`	Connectivity of actual tetrahedra	–	Integer
`:T4:virtual`	Connectivity of virtual tetrahedra	–	Integer
`:H8:real`	Connectivity of actual hexahedra	–	Integer
`:H8:virtual`	Connectivity of virtual hexahedra	–	Integer
`:W6:real`	Connectivity of actual prisms	–	Integer
`:W6:virtual`	Connectivity of virtual prisms	–	Integer
`:P5:real`	Connectivity of actual pyramids	–	Integer
`:P5:virtual`	Connectivity of virtual pyramids	–	Integer

Table 15.2: Definitions of surface-grid pane attributes.

Attribute	Meaning	Units	Type
`bcflag`	Boundary-condition flag	–	Integer
`patchNo`	Patch number	–	Integer
`constr_type`	Constraint type	–	Integer
`nc`	Coordinates	m	Real
`:t3:real`	Connectivity of actual triangles, local numbering	–	Integer
`:t3:virtual`	Connectivity of virtual triangles, local numbering	–	Integer
`:t3g:real`	Connectivity of actual triangles, global numbering	–	Integer
`:t3g:virtual`	Connectivity of virtual triangles, global numbering	–	Integer
`:q4:real`	Connectivity of actual quadrilaterals, local numbering	–	Integer
`:q4:virtual`	Connectivity of virtual quadrilaterals, local numbering	–	Integer
`:q4g:real`	Connectivity of actual quadrilaterals, global numbering	–	Integer
`:q4g:virtual`	Connectivity of virtual quadrilaterals, global numbering	–	Integer

Table 15.3: Definitions of solution attributes.

Attribute	Meaning	Units	Type
`rhof`	Density	kg/m^3	Real
`rhovf`	Momentum per unit volume	$kg/(m^2\,s)$	Real
`rhoEf`	Energy per unit volume	J/m^3	Real
`pf`	Pressure	Pa	Real
`Tf`	Temperature	K	Real
`af`	Speed of sound	m/s	Real

Table 15.4: Definitions of attributes on non-burning panes.

Attribute	Meaning	Units	Intent	Location	Type
`du_alp`	Displacement	m	Incoming	Vertex	Real
`rhofvf_alp`	Momentum flux	$kg/(m^2\,s)$	Incoming	Face	Real
`Tb_alp`	Boundary temperature	K	Incoming	Face	Real
`nf_alp`	Normal vector	–	Outgoing	Face	Real
`rhof_alp`	Density	kg/m^3	Outgoing	Face	Real
`pf`	Pressure	Pa	Outgoing	Face	Real
`qc`	Convective heat flux	W/m^2	Outgoing	Face	Real
`qr`	Radiative heat flux	W/m^2	Outgoing	Face	Real
`tf`	Traction	Pa	Outgoing	Face	Real

Table 15.5: Definitions of attributes on burning panes.

Attribute	Meaning	Units	Intent	Location	Type
`du_alp`	Displacement	m	Incoming	Vertex	Real
`rhofvf_alp`	Momentum	$kg/(m^2\,s)$	Incoming	Face	Real
`mdot_alp`	Injection mass flux	$kg/(m^2\,s)$	Incoming	Face	Real
`Tflm_alp`	Flame temperature	K	Incoming	Face	Real
`nf_alp`	Normal vector	–	Outgoing	Face	Real
`rhof_alp`	Density	kg/m^3	Outgoing	Face	Real
`pf`	Pressure	Pa	Outgoing	Face	Real
`tf`	Traction	Pa	Outgoing	Face	Real
`qc`	Convective heat flux	W/m^2	Outgoing	Face	Real
`qr`	Radiative heat flux	W/m^2	Outgoing	Face	Real
`Tf`	Temperature	K	Outgoing	Face	Real
`bflag`	Burning flag	–	Outgoing	Face	Integer

Chapter 16

File Content and Format Specifications

This chapter describes the content and format of RocfluMP files which may require modifications by a developer. The content and format of RocfluMP files which requires or may require editing by a user is described in Chapter 5. The naming of most files follows the conventions outlined in Sect. 5.1.

16.1 Grid Files

16.1.1 RocfluMP Grid File

The grid file contains the coordinates and connectivity information of the grid at a given iteration or time. The name of the grid file depends on whether it is written in ASCII or binary format and whether the grid is moving. The name of the grid file in ASCII format is:

- `<casename>.grda_<mmmmm>` for steady flows
- `<casename>.grda_<mmmmm>_<n.nnnnnnnE+nn>` for unsteady flows with moving grids

where `<mmmmm>` is the region number and `<n.nnnnnnnE+nn>` is the time stamp. The name of the grid file in binary format is:

- `<casename>.grd_<mmmmm>` for steady flows
- `<casename>.grd_<mmmmm>_<n.nnnnnnnE+nn>` for unsteady flows with moving grids

The format of the grid file in ASCII format is illustrated by the following code fragment:

```
1 WRITE(iFile,'(A)') '# ROCFLU grid file'
2 WRITE(iFile,'(A)') '# Precision and range'
3 WRITE(iFile,'(2(I8))') PRECISION(1.0_RFREAL),RANGE(1.0_RFREAL)
```

```
4  WRITE(iFile,'(A)') '# Physical time'
5  WRITE(iFile,'(E23.16)') global%currentTime
6  WRITE(iFile,'(A)') '# Dimensions'
7  WRITE(iFile,'(5(I8))') pGrid%nVertTot,pGrid%nTetsTot,pGrid%nHexsTot, &
8                         pGrid%nPrisTot,pGrid%nPyrsTot
9  WRITE(iFile,'(A)') '# Coordinates'
10 DO i = 1,3
11   WRITE(iFile,'(5(E23.16))') (pGrid%xyz(i,j),j=1,pGrid%nVertTot)
12 END DO ! i
13 IF ( pGrid%nTetsTot > 0 ) THEN
14   WRITE(iFile,'(A)') '# Tetrahedra'
15   DO i = 1,4
16     WRITE(iFile,'(10(I8))') (pGrid%tet2v(i,j),j=1,pGrid%nTetsTot)
17   END DO ! i
18 END IF ! pGrid%nTetsTot
19 IF ( pGrid%nHexsTot > 0 ) THEN
20   WRITE(iFile,'(A)') '# Hexahedra'
21   DO i = 1,8
22     WRITE(iFile,'(10(I8))') (pGrid%hex2v(i,j),j=1,pGrid%nHexsTot)
23   END DO ! i
24 END IF ! pGrid%nHexsTot
25 IF ( pGrid%nPrisTot > 0 ) THEN
26   WRITE(iFile,'(A)') '# Prisms'
27   DO i = 1,6
28     WRITE(iFile,'(10(I8))') (pGrid%pri2v(i,j),j=1,pGrid%nPrisTot)
29   END DO ! i
30 END IF ! pGrid%nPrisTot
31 IF ( pGrid%nPyrsTot > 0 ) THEN
32   WRITE(iFile,'(A)') '# Pyramids'
33   DO i = 1,5
34     WRITE(iFile,'(10(I8))') (pGrid%pyr2v(i,j),j=1,pGrid%nPyrsTot)
35   END DO ! i
36 END IF ! pGrid%nPyrsTot
37 WRITE(iFile,'(A)') '# Boundaries'
38 WRITE(iFile,'(I8)') pGrid%nPatches
39 DO iPatch = 1,pGrid%nPatches
40   pPatch => pRegion%patches(iPatch)
41   WRITE(iFile,'(3(I8))') pPatch%nBTrisTot,pPatch%nBQuadsTot
42   IF ( pPatch%nBTrisTot > 0 ) THEN
43     DO j = 1,3
44       WRITE(iFile,'(10(I8))') (pPatch%bTri2v(j,k),k=1,pPatch%nBTrisTot)
45     END DO ! j
46   END IF ! pPatch%nBTrisTot
47   IF ( pPatch%nBQuadsTot > 0 ) THEN
48     DO j = 1,4
```

```
      WRITE(iFile,'(10(I8))') (pPatch%bQuad2v(j,k),k=1,pPatch%nBQuadsTot)
    END DO ! j
  END IF ! pPatch%nBQuadsTot
END DO ! iPatch
WRITE(iFile,'(A)') '# End'
```

16.1.2 CENTAUR Grid File

16.1.3 VGRIDns Grid Files

The `.mapbc` file produced by VGRIDns is not read.

16.1.3.1 `.vbc` File

The `.vbc` file corresponds to the `.bc` file written by VGRIDns. It is renamed because the boundary-condition file of RocfluMP has the extension `.bc`.

16.1.3.2 `.cgosg` File

16.1.4 MESH3D Grid File

16.1.5 TETMESH Grid File

16.1.5.1 `.noboite` File

The format of the `.noboite` file is described in the TETMESH user's manual.

16.1.6 Cobalt Grid File

The first line of the Cobalt grid file contains three integers,

```
nDimensions  nZones  nBoundaryPatches
```

where the meaning is self-explanatory. For use within rflump, Cobalt grid files must satisfy the following restrictions: `nDimensions` must be equal to `3` and `nZones` must be equal to `1`.

The next line contains five integers,

```
nVertices  nFaces  nCells  nVerticesPerFaceMax  nFacesPerCellMax
```

where the last two quantities represent the maximum number of vertices defining a face and the maximum number of faces defining a cell. For example, if the grid consisted purely of tetrahedra, `nVerticesPerFaceMax` and `nFacesPerCellMax` would be equal to `3` and `4`, respectively. On the other hand, if the grid consisted of tetrahedra, prisms, and pyramids, `nVerticesPerFaceMax` and `nFacesPerCellMax` would be equal to `4` and `5`, respectively.

The next `nVertices` lines contain the x-, y-, and z-coordinates for each vertex.
The next `nFaces` lines contain the face-connectivity information,

```
nVerticesPerFace  <nVerticesPerFace> vertices  Cell1  Cell2
```

where `Cell1` and `Cell2` are the two cells which share a given face. If a face lies on a boundary patch, the respective cell is given by the negative patch index.

16.2 Flow-Solution File

The flow-solution file contains the conserved variables of the solution at a given iteration or time. The name of the flow-solution file depends on whether it is written in ASCII or binary format and whether a steady or unsteady flow is computed. The name of the flow-solution file in ASCII format is:

- `<casename>.floa_<mmmmm>_<nnnnnn>` for steady flows
- `<casename>.floa_<mmmmm>_<n.nnnnnnnE+nn>` for unsteady flows

where `<mmmmm>` is the region number, `<nnnnnn>` is the iteration number, and `<n.nnnnnnnE+nn>` is the time stamp. The name of the flow-solution file in binary format is:

- `<casename>.flo_<mmmmm>_<nnnnnn>` for steady flows
- `<casename>.flo_<mmmmm>_<n.nnnnnnnE+nn>` for unsteady flows

The format of the flow-solution file in ASCII format is illustrated by the following code fragment:

```
1  WRITE(iFile,'(A)') '# ROCFLU flow file'
2  WRITE(iFile,'(A)') '# Precision and range'
3  WRITE(iFile,'(2(I8))') PRECISION(1.0_RFREAL),RANGE(1.0_RFREAL)
4  WRITE(iFile,'(A)') '# Initial residual'
5  WRITE(iFile,'(E23.16)') global%resInit
6  WRITE(iFile,'(A)') '# Physical time'
7  WRITE(iFile,'(E23.16)') global%currentTime
8  WRITE(iFile,'(A)') '# Dimensions'
9  WRITE(iFile,'(2(I8))') pGrid%nCellsTot,pRegion%mixtInput%nCv
10 WRITE(iFile,'(A)') '# Mixture density'
11 WRITE(iFile,'(5(E23.16))') (pCv(CV_MIXT_DENS,j),j=1,pGrid%nCellsTot)
12 WRITE(iFile,'(A)') '# Mixture x-momentum'
13 WRITE(iFile,'(5(E23.16))') (pCv(CV_MIXT_XMOM,j),j=1,pGrid%nCellsTot)
14 WRITE(iFile,'(A)') '# Mixture y-momentum'
15 WRITE(iFile,'(5(E23.16))') (pCv(CV_MIXT_YMOM,j),j=1,pGrid%nCellsTot)
16 WRITE(iFile,'(A)') '# Mixture z-momentum'
```

```
WRITE(iFile,'(5(E23.16))') (pCv(CV_MIXT_ZMOM,j),j=1,pGrid%nCellsTot)
WRITE(iFile,'(A)') '# Mixture total internal energy'
WRITE(iFile,'(5(E23.16))') (pCv(CV_MIXT_ENER,j),j=1,pGrid%nCellsTot)
WRITE(iFile,'(A)') '# End'
```

16.3 Dimension File

The dimension file contains information about the grid and grid-related quantities for a given region. It is read by the various programs to determine the sizes of arrays. The file is called

- <casename>.dim_<mmmmm> for steady flows, and
- <casename>.dim_<mmmmm>_<nnnnnnE+nn> for unsteady flows,

where <mmmmm> is the region index and <nnnnnnE+nn> is the time stamp. At present, the time stamp of the dimension file is always zero.

The dimension file is always in ASCII format and consists of sections whose significance is indicated by character strings. The sections can be written in any order. Consider the following example dimensions file:

```
# ROCFLU dimensions file
# Vertices
    1797    4102    4922
# Cells
    8194   19697   23636
# Tetrahedra
    8194   19697   23636
# Hexahedra
       0       0       0
# Prisms
       0       0       0
# Pyramids
       0       0       0
# Patches
       3       6
       1     219     468       0       0
       3     360     708       0       0
       5      82     155       0       0
# Borders
       7
       4       4     988    1184     254     301      48
       9       2    2904    2457     617     516     141
      10       4    2121    2206     477     500     107
      12       3    3709    3358     770     690     191
      13       4      80      67      51      41       0
```

```
26        15        3      611      750      168      216       31
27        16        3     1477     1481      340      340       70
28 # End
```

Line 1: The first line must contain the string shown, otherwise reading of the file will fail.

Line 2: Specifies the beginning of the vertex dimension section.

Line 3: Specifies the number of actual vertices, the total number of vertices, and the maximum allowed number of vertices. The maximum allowed number of vertices must be equal to or greater than the total number of vertices.

Line 4: Specifies the beginning of the cell dimension section.

Line 5: Specifies the number of actual cells, the total number of cells, and the maximum allowed number of cells. The maximum allowed number of cells must be equal to or greater than the total number of cells.

Lines 6-13: Specify the dimensions of tetrahedral, prismatic, pyramidal, and hexahedral cells.

Line 14: Specifies the beginning of the patch dimension section.

Line 15: Specifies the number of patches local to the given region and the global number of patches. In this case, the given region contains three patches while there exist six global patches. The remainder of the patch section contains one line for each local patch.

Lines 16-18: Specifies, for each local patch, its global index, the number of actual triangles, the total number of triangles, the number of actual quadrilaterals, and the total number of quadrilaterals.

Line 19: Specifies the beginning of the border dimension section.

Line 20: Specifies the number of borders in the given region. The remainder of the border section contains one line for each border.

Lines 21-27: Specifies, for each border, the global index of the region with which data must be exchanged, the corresponding index of the border of the region with which data must be exchanged, the number of cells for which data must be sent, the number of cells for which data must be received, the number of vertices for which data must be sent, the number of vertices for which data must be received, and the number of vertices for which data must be shared.

Line 28: The last line must contain the string shown, otherwise reading of the file will fail.

16.4 Cell-Mapping File

The cell-mapping file contains the mappings between the numbering local to a given cell-type and the global numbering over all cell types. The file is always in ASCII format and is called <casename>.cmp_<mmmmm> where <mmmmm> is the region index.

The format of the cell-mapping file is illustrated by the following code fragment:

```
WRITE(iFile,'(A)') '# ROCFLU cell mapping file'
WRITE(iFile,'(A)') '# Dimensions'
WRITE(iFile,'(4(I8))') pGrid%nTetsTot,pGrid%nHexsTot,pGrid%nPrisTot,pGrid%nPyrsTot
WRITE(iFile,'(A)') '# Tetrahedra'
WRITE(iFile,'(10(I8))') (pGrid%tet2CellGlob(icg),icg=1,pGrid%nTetsTot)
WRITE(iFile,'(A)') '# Hexahedra'
WRITE(iFile,'(10(I8))') (pGrid%hex2CellGlob(icg),icg=1,pGrid%nHexsTot)
WRITE(iFile,'(A)') '# Prisms'
WRITE(iFile,'(10(I8))') (pGrid%pri2CellGlob(icg),icg=1,pGrid%nPrisTot)
WRITE(iFile,'(A)') '# Pyramids'
WRITE(iFile,'(10(I8))') (pGrid%pyr2CellGlob(icg),icg=1,pGrid%nPyrsTot)
WRITE(iFile,'(A)') '# End'
```

16.5 Renumbering File

The renumbering file contains the mapping between the global cell and vertex numbers of a given region and the global cell and vertex numbers of the serial region. In addition, the file contains the mapping between the boundary faces of a given region and the boundary faces of the serial region. The file is always in ASCII format and is called <casename>.rnm_<mmmmm> where <mmmmm> is the region index.

The format of the renumbering file is illustrated by the following code fragment:

```
WRITE(iFile,'(A)') '# ROCFLU renumbering file'
WRITE(iFile,'(A)') '# Dimensions'
WRITE(iFile,'(2(I8))') pGrid%nVertTot,pGrid%nCellsTot
WRITE(iFile,'(A)') '# Vertices'
WRITE(iFile,'(10(I8))') (pGrid%pv2sv(ivg),ivg=1,pGrid%nVertTot)
WRITE(iFile,'(A)') '# Cells'
WRITE(iFile,'(10(I8))') (pGrid%pc2sc(icg),icg=1,pGrid%nCellsTot)
WRITE(iFile,'(A)') '# Boundary faces'
WRITE(iFile,'(10(I8))') (pGrid%pbf2sbfCSR(ifl),ifl=1,pGrid%nBFacesTot)
WRITE(iFile,'(A)') '# End'
```

16.6 Communication-Lists File

The communication-lists file contains the communications lists required by each region to send data to and receive data from other regions. The file is always in ASCII format and is

called <casename>.com_<mmmmm> where <mmmmm> is the region index.

The format of the communication-lists file is illustrated by the following code fragment:

```
WRITE(iFile,'(A)') '# ROCFLU communication lists file'
WRITE(iFile,'(A)') '# Dimensions'
WRITE(iFile,'(I8)') pGrid%nBorders
DO iBorder = 1,pGrid%nBorders
  pBorder => pGrid%borders(iBorder)
  WRITE(iFile,'(2(I8))') pBorder%iRegionGlobal,pBorder%iBorder
END DO ! iBorder
WRITE(iFile,'(A)') '# Cells'
DO iBorder = 1,pGrid%nBorders
  pBorder => pGrid%borders(iBorder)
  WRITE(iFile,'( 2(I8))') pBorder%nCellsSend,pBorder%nCellsRecv
  WRITE(iFile,'(10(I8))') (pBorder%icgSend(icl),icl=1,pBorder%nCellsSend)
  WRITE(iFile,'(10(I8))') (pBorder%icgRecv(icl),icl=1,pBorder%nCellsRecv)
END DO ! iBorder
WRITE(iFile,'(A)') '# Vertices'
DO iBorder = 1,pGrid%nBorders
  pBorder => pGrid%borders(iBorder)
  WRITE(iFile,'( 3(I8))') pBorder%nVertSend,pBorder%nVertRecv,pBorder%nVertShared
  WRITE(iFile,'(10(I8))') (pBorder%ivgSend(ivl),ivl=1,pBorder%nVertSend)
  WRITE(iFile,'(10(I8))') (pBorder%ivgRecv(ivl),ivl=1,pBorder%nVertRecv)
  WRITE(iFile,'(10(I8))') (pBorder%ivgShared(ivl),ivl=1,pBorder%nVertShared)
END DO ! iBorder
WRITE(iFile,'(A)') '# End'
```

References

[1] Abgrall R., Désidéri J.-A., Glowinski R., Mallet M., and Périaux J. (Eds.), *Hypersonic Flows for Reentry Problems*, Vol. III, Proceedings of the INRIA-GAMNI/SMAI Workshop on Hypersonic Flows for Reentry Problems, Part II. Antibes, France, April 15-19, 1991, Springer Verlag.

[2] Barth T.J. and Jespersen D.C., *The Design and Application of Upwind Schemes on Unstructured Meshes*, AIAA Paper 89-0366, 27th Aerospace Sciences Meeting, Reno, NV, January 1989

[3] Batten P., Clarke N., Lambert C., and Causon D.M., *On the Choice of Wavespeeds for the HLLC Riemann Solver*, SIAM J. Sci. Comp., 18(6):1553-1570, 1997.

[4] Bruner C.W.S., *Geometric Properties of Arbitrary Polyhedra in Terms of Face Geometry*, AIAA J., 33(7):1350, 1995.

[5] Chakraborty P., Balachandar S., and Adrian R.J., *On the Relationships between Local Vortex Identification Schemes*, J. Fluid Mech., Vol. 535, pp. 189-214, 2005.

[6] Ciucci A., Iafrati A., and Schettino A., *Numerical Analysis of Pressure Oscillations in a Duct - Test Case C0*, Technical Report TR-96-102, Centro Italiano Ricerche Aerospaziali, September 1996.

[7] Liou M.-S., *A Sequel to AUSM: AUSM+*, J. Comp. Phys., 129:364-382, 1996.

[8] Poinsot T.J. and Lele S.K., *Boundary Conditions for Direct Simulations of Compressible Viscous Flows*, J. Comp. Phys., 101:104-129, 1992.

[9] Ringleb F., *Exakte Lösungen der Differentialgleichung einer adiabaten Gasströmung*, ZAMM, 20(4):185-198, 1940.

[10] Roe P.L., *Approximate Riemann Solvers, Parameter Vectors, and Difference Schemes*, J. Comp. Phys., 43:357-372, 1981.

[11] Skews B.W., *The Shape of a Diffracting Shock Wave*, J. Fluid Mech., 29(2):297-304, 1967

[12] Skews B.W., *The Perturbed Region Behind a Diffracting Shock Wave*, J. Fluid Mech., 29(4):705-719, 1967

[13] Sod G., *A Survey of Several Finite Difference Methods for Systems of Nonlinear Hyperbolic Conservation Laws*, J. Comp. Phys., 27:1-31, 1978.

[14] Thompson K.W., *Time Dependent Boundary Conditions for Hyperbolic Systems*, J. Comp. Phys., 68:1-24, 1987.

[15] Venkatakrishnan V., *Convergence to Steady-State Solutions of the Euler Equations on Unstructured Grids with Limiters*, J. Comp. Phys., Vol. 118, No. 1, pp. 120-130, April 1995

[16] Wang Z.J., "Improved Formulation for Geometric Properties of Arbitrary Polyhedra", AIAA J., 37(10):1326, 1999.

Index

CENTAUR grid file, 179
    format of, 179
Cobalt grid file, 179
    format of, 179
ENSIGHT files
    File Naming Convention, 117
    Part Naming Conventions, 117
GENx control file, 79
    Format of, 79
MESH3D grid file, 179
    format of, 179
rfluclone
    Execution, 85
    Input files, 85
    Invocation, 85
    Output files, 86
rfluconv
    Execution, 86
    Input files, 87
    Interactive Input, 87
    Invocation, 86
    Output files, 87
rfluextr
    Execution, 87
    Input files, 88
    Interactive Input, 88
    Invocation, 87
    Output files, 88
rfluinit
    Execution, 88
    Input files, 89
    Invocation, 88
    Output files, 89
rflumap
    Execution, 89
    Input files, 90
    Invocation, 89
    Output files, 90
rflump
    Automatic restart capability, 91
    Execution, 90
    Input files, 91
    Invocation, 90
    Output files, 91
rflupart
    Batch-Job Submission Guidelines, 94
    Execution, 93
    Input files, 93
    Invocation, 93
rflupick
    Execution, 94
    Input files, 96
    Interactive Input, 97
    Invocation, 96
    Output files, 96
rflupost
    Execution, 98
    Input files, 98
    Invocation, 98
    Output files, 99
rfluprep
    Output files, 94
TECPLOT files
    File Naming Convention, 111
    Zone Naming Conventions, 112
TETMESH grid file, 179
    format of, 179
PROBE Section
    Example, 50
    Format of, 50

VGRIDns grid file, 179
format of, 179

Automatic restart capability
rflump, 91

Batch-Job Submission Guidelines
rflupart, 94
Boundary-condition file, 57
Format of, 57

Cell-mapping file, 183
Format of, 183
Communication-lists file, 183
Format of, 183
Convergence file, 77
Format of, 77

Dimension file, 181
Format of, 181

Execution, 85
rfluclone, 85
rfluconv, 86
rfluextr, 87
rfluinit, 88
rflumap, 89
rflump, 90
rflupart, 93
rflupick, 94
rflupost, 98

File Naming Convention
ENSIGHT files, 117
TECPLOT files, 111
Flow-solution file, 180
Format of, 180
Format of
PROBE Section, 50

Input files
rfluclone, 85
rfluconv, 87
rfluextr, 88
rfluinit, 89
rflumap, 90
rflump, 91
rflupart, 93
rflupick, 96
rflupost, 98
Interactive Input
rfluconv, 87
rfluextr, 88
rflupick, 97
Invocation
rfluclone, 85
rfluconv, 86
rfluextr, 87
rfluinit, 88
rflumap, 89
rflump, 90
rflupart, 93
rflupick, 96
rflupost, 98

Moving-reference-frame file, 78
Format of, 78

Output files
rfluclone, 86
rfluconv, 87
rfluextr, 88
rfluinit, 89
rflumap, 90
rflump, 91
rflupick, 96
rflupost, 99
rfluprep, 94

Part Naming Conventions
ENSIGHT files, 117
Probe file, 78
Format of, 78

Region-mapping file, 75
Format of, 75
Renumbering file, 183
Format of, 183
Restart-information file, 76

Format of, 76

Zone Naming Conventions
TECPLOT files, 112