User-Defined Functions

ADVANCED
CONVERGECFD
TRAINING

CONVERGE
CFD SOFTWARE

Introduction

- Via user-defined functions (UDFs), you can control
  - Boundary conditions
  - Combustion modeling
  - Discrete phase modeling
  - Events
  - Initialization
  - Turbulence modeling
  - Input/output
- Future additions
  - Material properties
  - Load balancing
  - Sources
  - Time-steps
  - AMR
  - Boundary motion

Outline

- UDF structure
  - Header files
  - UDF macros
  - Variables
  - Loops
- UDF setup in Linux
- UDF examples

Structure of UDF Source Code

```
#include <CONVERGE/udf.h>
#include <standard_headers.h>
#include “user_headers.h”
```

UDF headers

```
CONVERGE_XXX(<name>, IN(...), OUT(...))
```

UDF macros

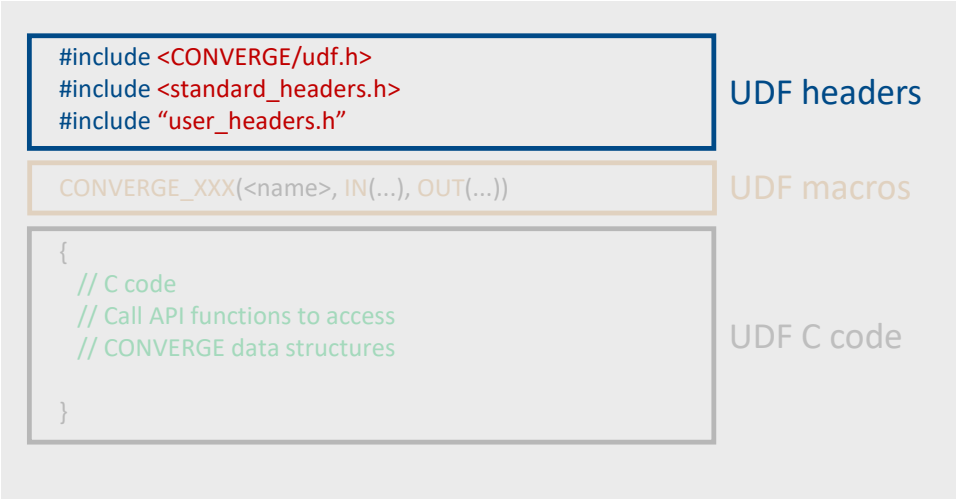
```
{
  // C code
  // Call API functions to access
  // CONVERGE data structures

}
```

UDF C code

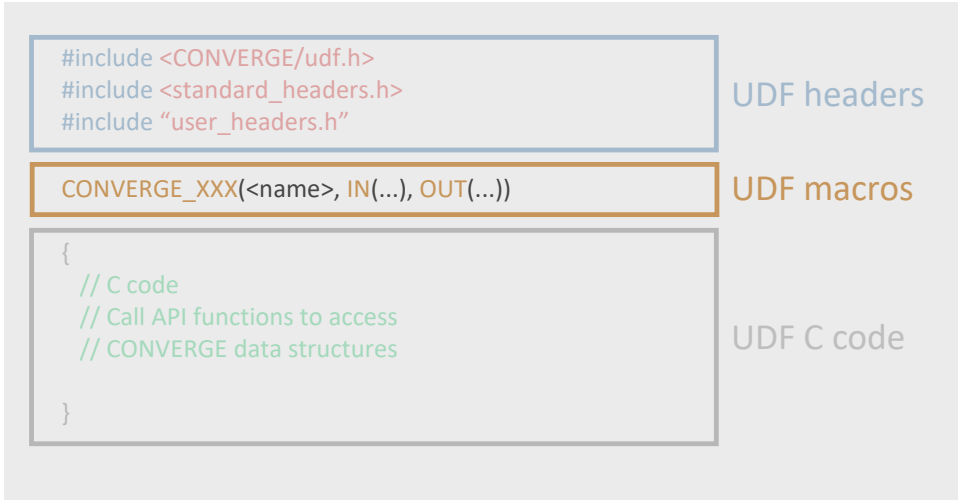
Header Files

- Include header files for any library functions used in your UDF
- *<CONVERGE/udf.h>* grants access to the CONVERGE API and is **required**
- Standard libraries and user-defined headers are **optional**



UDF Macros (1/5)

- UDF macros translate the UDF C functions for the CONVERGE solver
- Examples of UDF macros
  - *CONVERGE_UDF*
  - *CONVERGE_ONLOAD*
  - *CONVERGE_POST*
  - *CONVERGE_REACTION_RATE*



UDF Macros (2/5)

- *CONVERGE_UDF(<name>, IN(…), OUT(…))*
  - *UDF* function names are set internally and cannot be modified by users
  - The *IN* and *OUT* macros contain the list of variables used by the UDF
- Multiple examples in the *examples* directory and the UDF Manual

UDF Macros (3/5)

- *CONVERGE_ONLOAD(<name>, IN(…))*
  - *ONLOAD* functions can have any user-specified name
  - They are called when the UDF library is loaded
  - Use these functions for global UDF environment configuration (*e.g.*, setting up variables that can be used by the UDFs)
- Examples of *ONLOAD* functions
  - */examples/lagrangian/load_lag_env.c*
  - */examples/turbulence/turbmodel/user_turb_model_input_setup.c*

UDF Macros (4/5)

- *CONVERGE_POST(<name>, IN(…))*
  - *POST* functions can have any user-specified name
  - For visualization, manually add the names of the *POST* functions to *post.in* (they are not currently shown in CONVERGE Studio *Case Setup*)
  - CONVERGE populates a dynamic buffer with the *POST* function name
  - CONVERGE calls *POST* functions when writing 3D *post*.h5* output files only if those variables are included in *post.in*
- Examples of *POST* functions: */examples/io/post/post.c*

UDF Macros (5/5)

- *CONVERGE_REACTION_RATE(<name>, IN(…), OUT(…))*
  - *REACTION_RATE* functions provide user-specified reaction rates for certain reactions in the mechanism
  - Naming convention: *<reaction type>_rxn_<#>*
    - *<reaction type>* is *surface* or *gas*
    - *<#>* is a positive integer for a reaction in *mech.dat* or *surface_mech.dat*
- Examples of *REACTION_RATE* functions*:*
  - */examples/combustion/rxn_rate/reaction_rate.c*
  - */examples/combustion/rxn_rate/surface_reaction_rate.c*

Variables (1/4)

- UDFs can access CONVERGE variables using three macros
  - *FIELD()* is used for arrays that are specified as pointers
    - Arrays can be cell-centered, face-centered, or boundary-face-centered variables
    - Examples: *density*, *face_normal, boundary_temperature*
  - *VALUE()* is used for variables that are passed by value
  - *REF()* is used for variables that are passed by reference
- Each variable has a defined type (*e.g.*, *CONVERGE_precision_t*)
  - You must use the correct type when writing UDFs

```
CONVERGE_UDF(calc_yplus,
        IN(FIELD(CONVERGE_precision_t *, tke),
           FIELD(CONVERGE_precision_t *, density),
           FIELD(CONVERGE_precision_t *, mol_viscosity),
           VALUE(CONVERGE_index_t, cell_index),
           VALUE(CONVERGE_precision_t, passed_walldist)),
        OUT(REF(CONVERGE_precision_t, passed_bound_yplus),
              REF(CONVERGE_precision_t, passed_bound_law_shear_speed)))
```

Example: Input and output variables for *calc_yplus*

Variables (2/4)

- To see all *FIELD()* variables available to you, run a simulation and then check the *active_variables.log* file
  - *u* indicates that the variable can be used in UDFs

```
DENSITY:
        avail:                  [a,  p,  t,  u]
        type:                   CONVERGE_precision_t
```

- To access the variable in the UDF, use *FIELD(type *, name)* in the *IN()* or *OUT()* macro
- Example: *FIELD(CONVERGE_precision_t *, density)*

Variables (3/4)

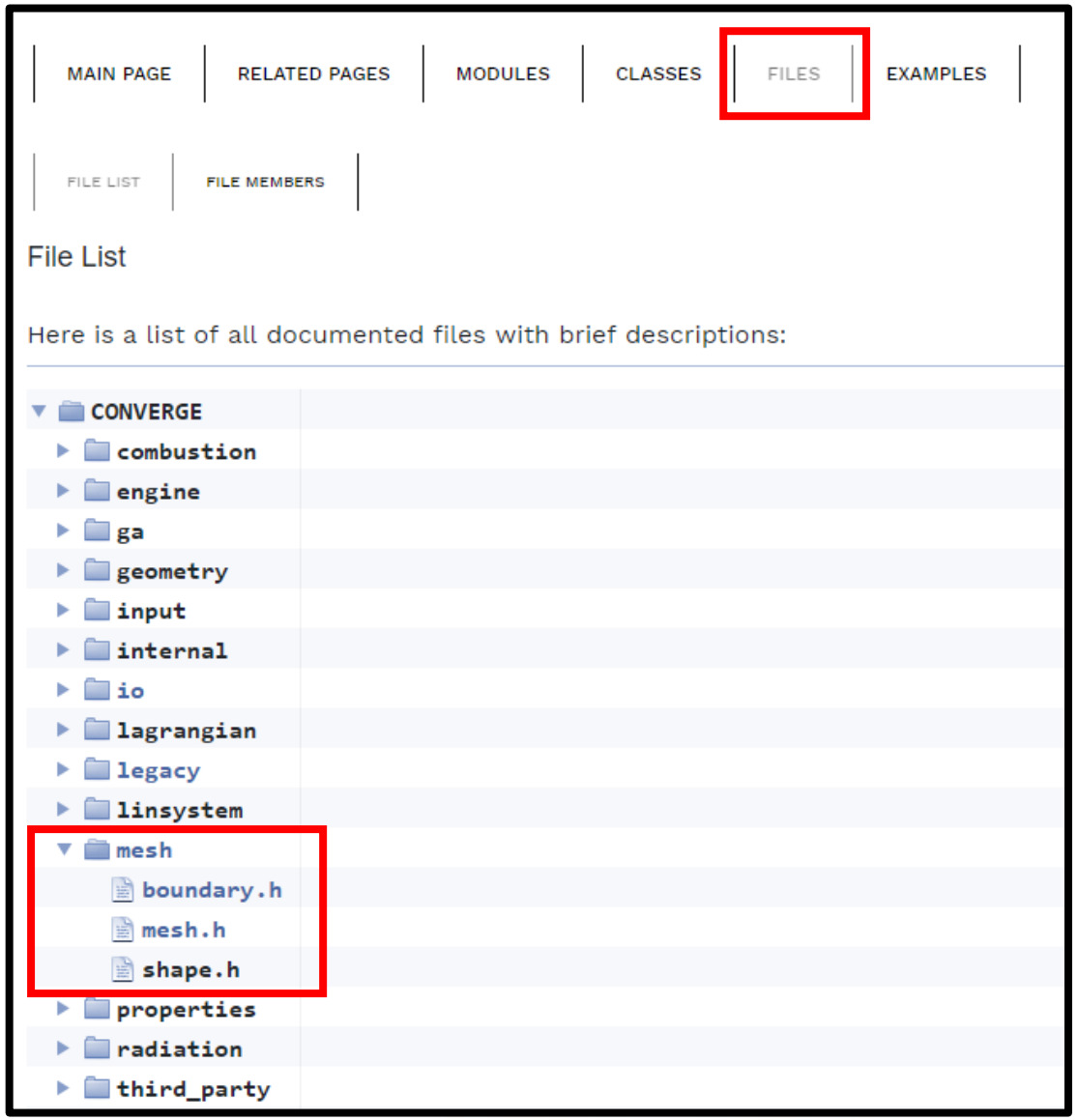
- Available variables for *VALUE()* and *REF()* are defined per UDF
  - Refer to the UDF Manual for input and output variables
- Inputs are always passed by *VALUE()*
- For outputs that have been changed by the UDF:
  - Scalar quantities are passed by *REF()*
  - Arrays are passed by *VALUE()*

Variables (4/4)

- All data accessed from CONVERGE has a defined type
- You must use the correct type for each variable
- Commonly used types are listed below
  - *CONVERGE_mesh_t*
  - *CONVERGE_table_t*
  - *CONVERGE_iterator_t*
  - *CONVERGE_species_t*
  - *CONVERGE_flag_t*
  - *CONVERGE_int_t*
  - *CONVERGE_index_t*
  - *CONVERGE_precision_t*
  - *CONVERGE_id_t*
  - *CONVERGE_vec3_t*
  - *CONVERGE_mat3_t*
  - *CONVERGE_injector_t*
  - *CONVERGE_nozzle_t*
  - *CONVERGE_cloud_t*
  - *CONVERGE_cloud_list_t*
  - *CONVERGE_outfile_header_t*
  - *CONVERGE_outfile_row_template_t*

CONVERGE API

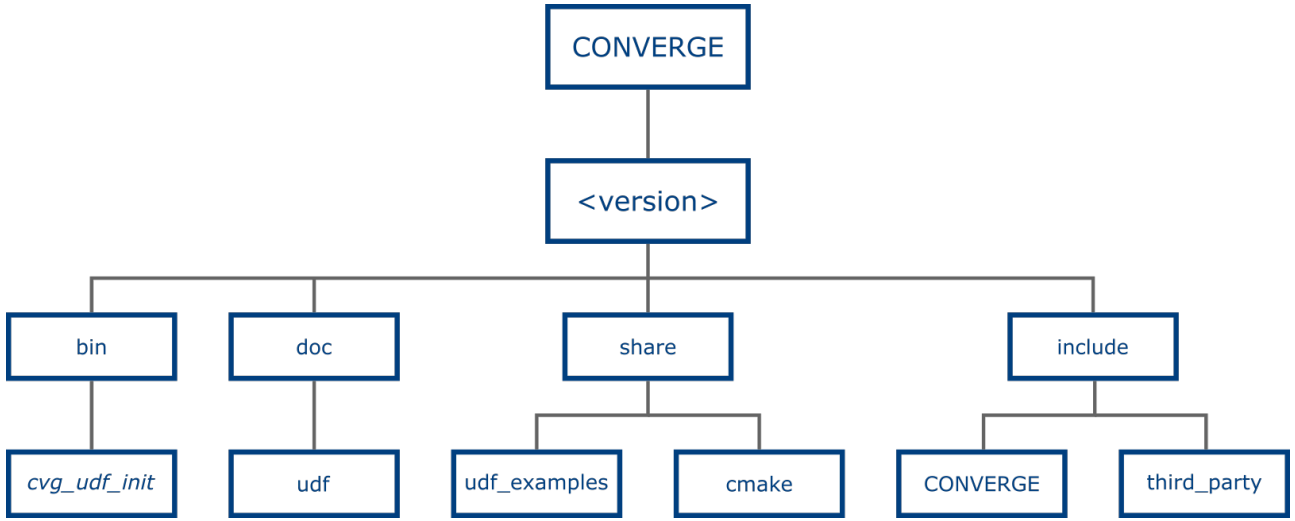
- Use API functions to retrieve information from CONVERGE data structures
- The installation package includes an *index.html* file with API documentation (in */CONVERGE/<version>/doc/udf/html*)
  - Open *index.html* in a browser
  - Go to *Modules* or *Files* tab
  - Header files/APIs are organized by functional area
  - APIs can be searched as well.



CONVERGE CFD SOFTWARE

UDF Setup for Linux (1/3)

- UDFs are available for CONVERGE 3.0.6 and later versions
- UDF-related files and directories are automatically included when you install the CONVERGE solver



UDF Setup for Linux (2/3)

1. Use *module load* to set environment variables

MPI Package	Command
Intel MPI	`module load <install_root>/Convergent_Science/Environment/modulefiles/CONVERGE/CONVERGE-IntelMPI/<version>`
MPICH	`module load <install_root>/Convergent_Science/Environment/modulefiles/CONVERGE/CONVERGE-MPICH/<version>`
Open MPI	`module load <install_root>/Convergent_Science/Environment/modulefiles/CONVERGE/CONVERGE-OMPI/<version>`

- Scripts are available if you don't use modules (refer to CONVERGE 3.0 Getting Started Guide for more information)

UDF Setup for Linux (3/3)

2. Run `cvg_udf_init` to create *examples*, *src*, *include*, and *build* directories
3. Copy UDF source file to *src* directory (you can use any source file in the *examples* directory)
4. Change to *build* directory and run the `cmake` and `make` commands to compile the source files in the *src* directory and to generate the shared dynamic library *libconverge_udf.so*
   - Minimum required versions: *cmake* 3.9.1, *make* 3.82
5. To link the **.so* library to CONVERGE, set `LD_LIBRARY_PATH` to point to the subdirectory containing the library
6. Activate any desired UDFs in CONVERGE Studio via *Case Setup > User Defined Functions > UDF Selection* (or set *feature_control > udf_flag* = 1 in *inputs.in* and set the desired flags [*i.e.*, *user_*_flag*] to 1 in *udf.in*)

Loops (1/9)

- Loop over cells

```
#include <CONVERGE/udf.h>
#include <standard_headers.h>
#include “user_headers.h”
```
UDF headers

```
CONVERGE_XXX(<name>, IN(...), OUT(...))
```
UDF macros

```
{
  // C code
  // Call API functions to access
  // CONVERGE data structures

}
```
UDF C code

```
CONVERGE_mesh_t mesh = <get mesh>
CONVERGE_iterator_t it;
CONVERGE_mesh_iterator_create(mesh, &it);
for(CONVERGE_index_t i = CONVERGE_iterator_first(it);
                i != -1;
                i = CONVERGE_iterator_next(it))
{
  // Access node i
}
CONVERGE_iterator_destroy(&it);
```

Loops (2/9)

- Loop over boundary faces

```
CONVERGE_mesh_t mesh = <get mesh>
CONVERGE_boundary_t bound;
CONVERGE_iterator_t mb_it;
CONVERGE_mesh_boundary_index_iterator_create(mesh, &mb_it);
for(CONVERGE_index_t i = CONVERGE_iterator_first(mb_it);
                i != -1;
                i = CONVERGE_iterator_next(mb_it))
{
  bound = CONVERGE_mesh_boundary(mesh, i);
  CONVERGE_iterator_t b_it;
  CONVERGE_boundary_iterator_create(bound, &b_it);

  for(CONVERGE_index_t j = CONVERGE_iterator_first(b_it);
                  j != -1;
                  j = CONVERGE_iterator_next(b_it))
  {
    // Access boundary node j of boundary i
  }
  CONVERGE_iterator_destroy(&b_it);
}
CONVERGE_iterator_destroy(&mb_it);
```

Loops (3/9)

- Loop over regions

```
CONVERGE_mesh_t mesh = <get mesh>
CONVERGE_iterator_t it;
CONVERGE_mesh_region_index_iterator_create(mesh, &it);
for(CONVERGE_index_t i = CONVERGE_iterator_first(it);
                i != -1;
                i = CONVERGE_iterator_next(it))
{
  // Access region i
}
CONVERGE_mesh_region_index_iterator_destroy(&it);
```

Loops (4/9)

- Loop over cells of a region

```
CONVERGE_mesh_t mesh = <get mesh>
CONVERGE_iterator_t it;
CONVERGE_mesh_region_iterator_create(mesh, ireg, &it);
for(CONVERGE_index_t i = CONVERGE_iterator_first(it);
                  i != -1;
                  i = CONVERGE_iterator_next(it))
{
  // Access node i of region ireg
}
CONVERGE_mesh_region_iterator_destroy(&it);
```

Loops (5/9)

- Loop over species (species array contains all phases)

```
CONVERGE_species_t species = <get species structure>
CONVERGE_iterator_t sp_it;
CONVERGE_species_euler_iterator_create(species, &sp_it);
for(CONVERGE_index_t isp = CONVERGE_iterator_first(sp_it);
                isp != -1;
                isp = CONVERGE_iterator_next(sp_it))
{
  // index species array with isp
}
CONVERGE_iterator_destroy(&sp_it);
```

Loops (6/9)

- Loop over species (species array contains only one phase)

```
CONVERGE_species_t species = <get species structure>
CONVERGE_index_t num_parcel_species = CONVERGE_species_num_parcel(species);
for(CONVERGE_index_t isp = 0;
                isp < num_parcel_species;
                isp++)
{
  // index parcel species array with isp
}
```

Loops (7/9)

- Loop over species (multiple species arrays)

```
CONVERGE_species_t species = <get species structure>
CONVERGE_iterator_t sp_it;
CONVERGE_species_gas_iterator_create(species, &sp_it);
for(CONVERGE_index_t multi_phase_isp = CONVERGE_iterator_first(sp_it), gas_isp = 0;
                multi_phase_isp != -1;
                multi_phase_isp = CONVERGE_iterator_next(sp_it), gas_isp++)
{
  // index species array with multi_phase_isp
  // index gas species array with gas_isp
}
CONVERGE_iterator_destroy(&sp_it);
```

Loops (8/9)

- Loop over spray or film parcels

```
CONVERGE_id_t LAG_RADIUS_ID = CONVERGE_lagrangian_field_id("LAGRANGIAN_RADIUS");
CONVERGE_cloud_list_t cloud_list = <Get cloud list. May either be a spray
cloud list or a film cloud list.>

CONVERGE_iterator_t cl_it;
CONVERGE_cloud_list_iterator_create(cloud_list, &cl_it);
for(CONVERGE_index_t i = CONVERGE_iterator_first(cl_it);
                i != -1;
                i = CONVERGE_iterator_next(cl_it))
{
  // Get the cloud with index i from cloud_list
  // If cloud_list is a spray cloud_list then the cloud will be a spray cloud
  // if cloud_list is a film cloud_list, then the cloud will be a film cloud
```

Loops (9/9)

- Loop over spray or film parcels (continued)

```
  CONVERGE_cloud_t cloud = CONVERGE_cloud_list_get_cloud_at(cloud_list, i);
  // Get the parcel radius array from the cloud.
  CONVERGE_precision_t* parcel_radius =
    CONVERGE_cloud_get_field_data_precision(cloud, LAG_RADIUS_ID);
  CONVERGE_iterator_t c_it;
  CONVERGE_cloud_iterator_create(cloud, &c_it);
  for(CONVERGE_index_t j = CONVERGE_iterator_first(c_it);
                  j != -1;
                  j = CONVERGE_iterator_next(c_it))
  {
    // Access the parcel radius for parcel j in cloud i
    (void)parcel_radius[j];
  }
  CONVERGE_iterator_destroy(&c_it);
}
CONVERGE_iterator_destroy(&cl_it);
```

UDF Examples

- 3D post output for local CFL number
- Soot mass calculation in the cylinder region and cells adjacent to liner boundary
- Inflow boundary conditions in cylindrical coordinates
- SMD calculation of spray parcels on a plane
- Custom reaction rate for a two-step mechanism
- AMR based on progress variable created dynamically by the UDF
- Events based on pressure difference

Example 1: Local CFL Calculation

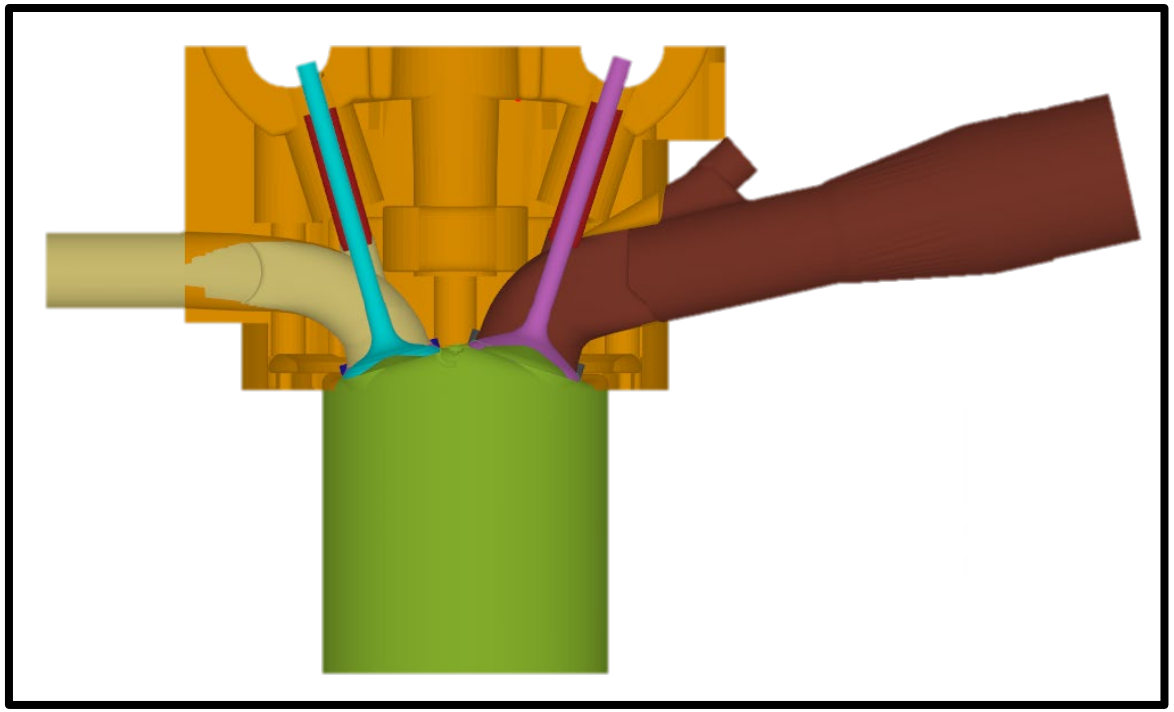
- Task: Calculate the local CFL number and write it to a *post*.h5* file
  - The CFL number is given by

$$\mathrm{CFL} = \frac{u\Delta t}{\Delta x}$$

- Analysis: This is an input/output (I/O) UDF with 3D output
- Plan: Start with *post.c* (*CONVERGE_POST* macro) from the default examples

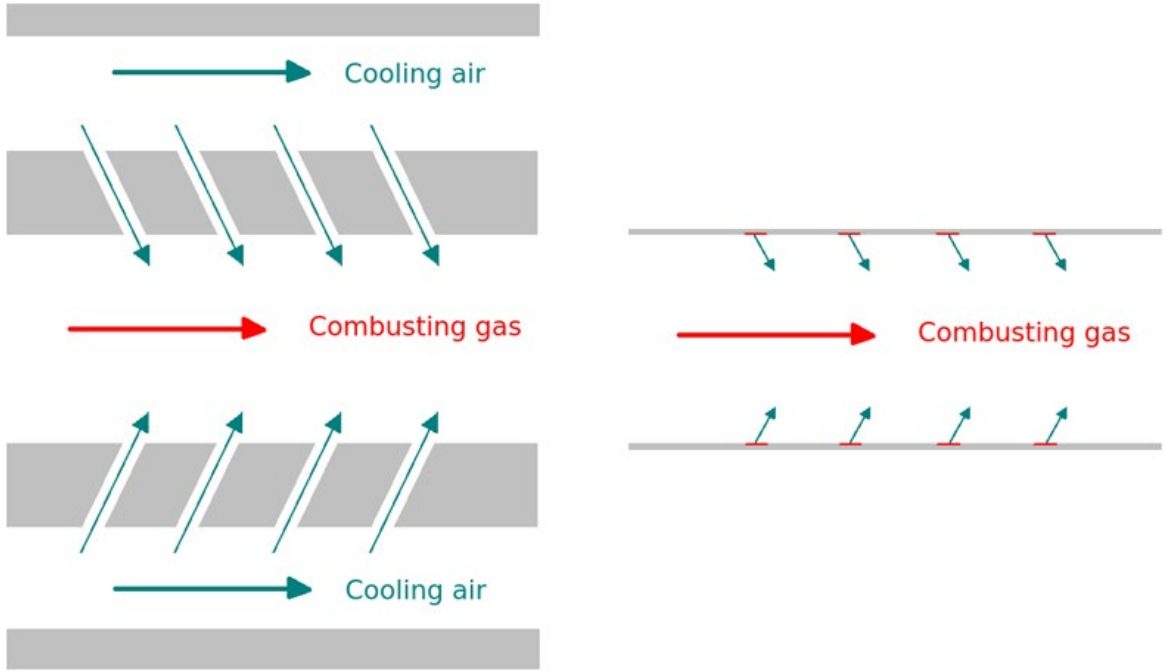
Example 2: Soot Mass Calculation

- Task: Calculate soot mass adjacent to the liner boundary, calculate total soot mass in the cylinder region, and write both to a **.out* file
- Analysis: This is an I/O UDF
  - Read the cylinder region ID and liner boundary name from an input file
  - Write 0D output
- Plan: Start with *read_input.c* and *output.c* from default examples



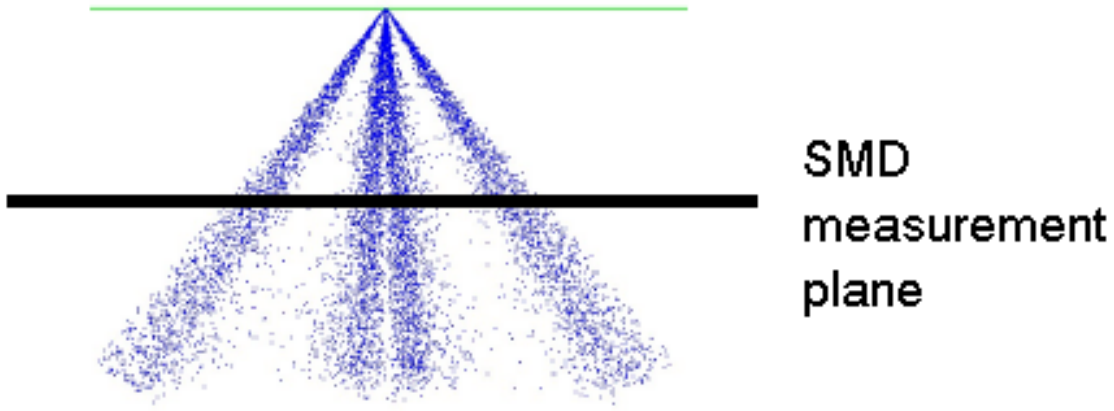
Example 3: Cylindrical Inflow Velocity Conditions

- Task: Provide inflow velocity in Cartesian coordinates based on velocity inputs in cylindrical coordinates
- Analysis: Need to read cylindrical velocity components from input file and convert to Cartesian coordinates
- Plan:
  - Use I/O UDF to read inputs
  - Use *CONVERGE_PROFILE* macro to convert cylindrical velocity components to Cartesian velocity components for each inflow boundary face



Example 4: SMD on a Plane

- Task: Measure SMD of spray parcels on a given plane
- Analysis: This is an I/O UDF
  - Read plane information from an input file
  - Write SMD of parcels on the plane at every **.out* file interval
- Plan: Start with *read_input.c* and *output.c* from default examples



Example 5: Custom Reaction Rate

- Task: Dynamically control the Arrhenius coefficient of a two-step mechanism to match the flamespeed with a detailed mechanism
- Analysis: Need custom reaction rate for the flamespeed-controlling reaction in *mech.dat*
- Plan: Use *CONVERGE_REACTION_RATE* macro to control the Arrhenius coefficient based on local conditions
  - Requires tagging the flamespeed-controlling reaction as a user reaction

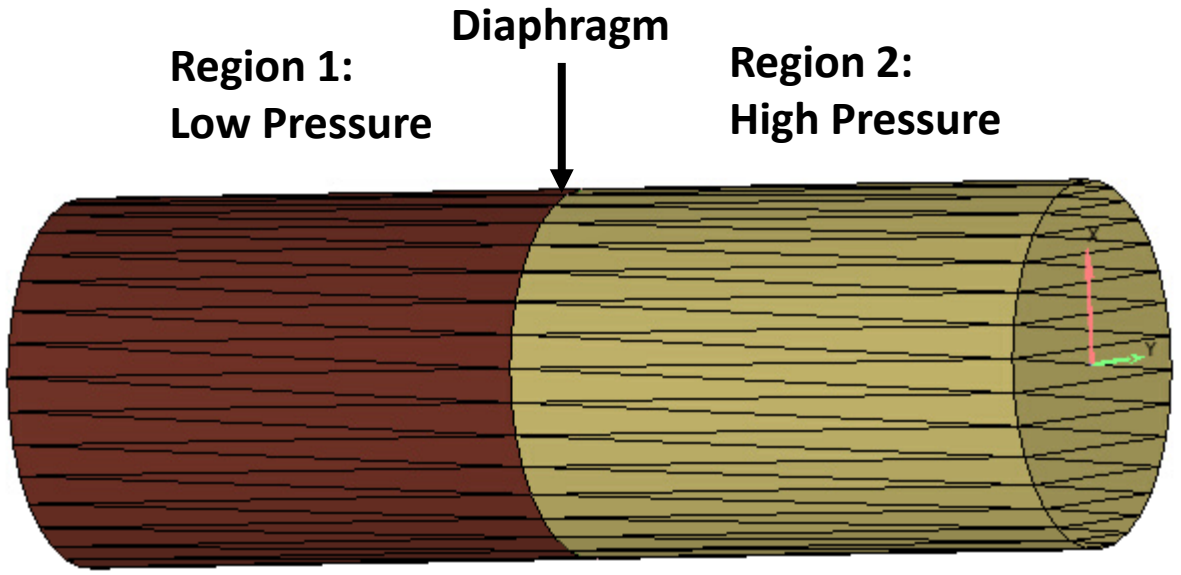
```
IC8H18 + 8.5 O2 => 8.0 CO + 9.0 H2O     A     0.0  3.4e+04
user/#/
CO + 0.5 O2 = CO2                              2.0e+09 0.0 1.4e+04
```

Example 6: AMR Based on Progress Variable

- Task: Activate AMR based on progress variable
  - AMR based on temperature can create finer cells near the wall in the burnt region because of temperature gradient
  - This can be remedied by using a progress variable instead of temperature
- Analysis: This is an *ONLOAD* UDF (global variable setup)
- Plan
  - Use *CONVERGE_ONLOAD* macro to create dynamic buffer for progress variable
  - Activate progress variable for AMR

Example 7: Events Based on Pressure Difference

- Task: Simulate the rupture of a diaphragm due to the pressure difference across the diaphragm
- Analysis: Need to create an OPEN event based on pressure difference
- Plan: Use *CONVERGE_EVENT* macro to define a custom OPEN event



Need Help Writing a UDF?

- Contact the Convergent Science Applications team for help with custom development
  - support@convergecfd.com (US)
  - supportEU@convergecfd.com (EU)
  - support.in@convergecfd.com (India)

Please fill out the training survey for this session.
Before you leave for the day, please place the survey sheet in one of the folders (available in each training room).
We appreciate your feedback!

THANK YOU!

CONVERGECFD.COM

CONVERGECFD TRAINING

CONVERGE CFD SOFTWARE