Connecting Cantera and FLUENT

E. David Huckaby

Cantera Workshop

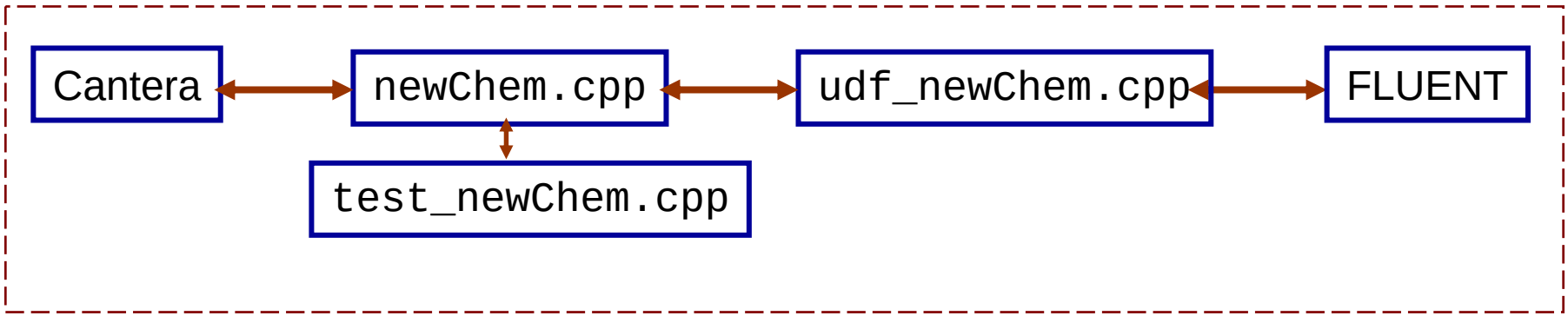
30th Symposium on Combustion, July 2004

The National Energy Technology Laboratory

Design of the Interface

Design:

- mimic “Cantera-clib” interface
- The FLUENT udf-functions have no explicit reference to Cantera



Why:

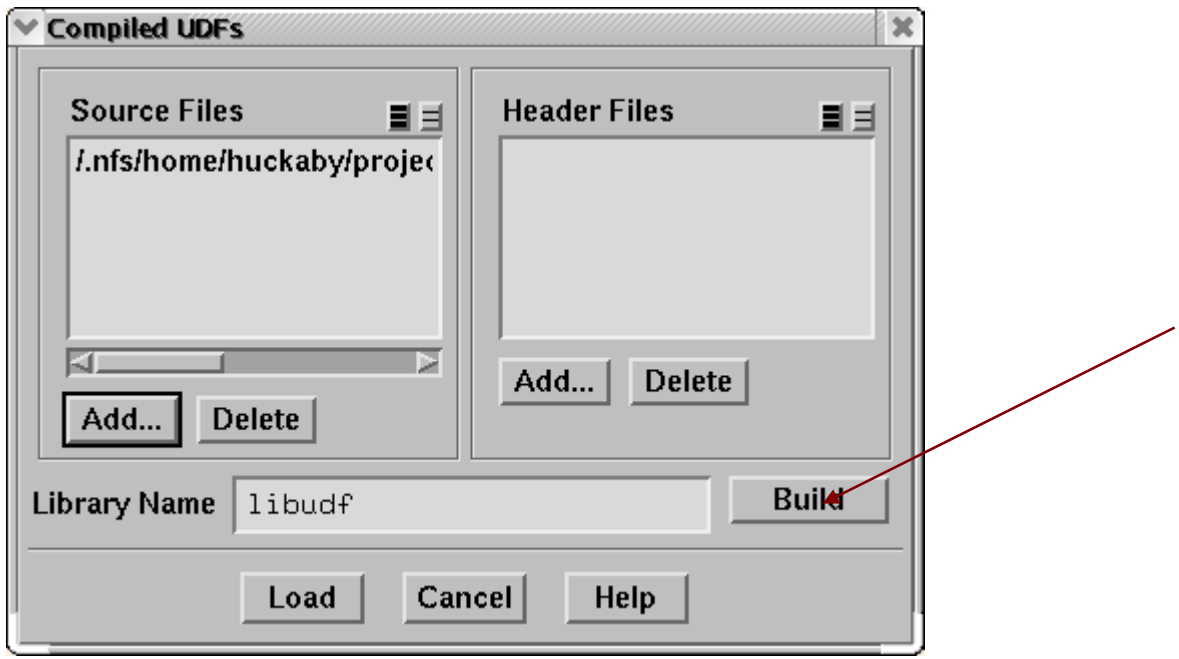
- interference between FLUENT and Cantera functions (CVode)
- offline testing and debugging
  - be careful of memory leaks (“ccmalloc”)
- functions can be used with other packages

Steps to connect FLUENT and Cantera

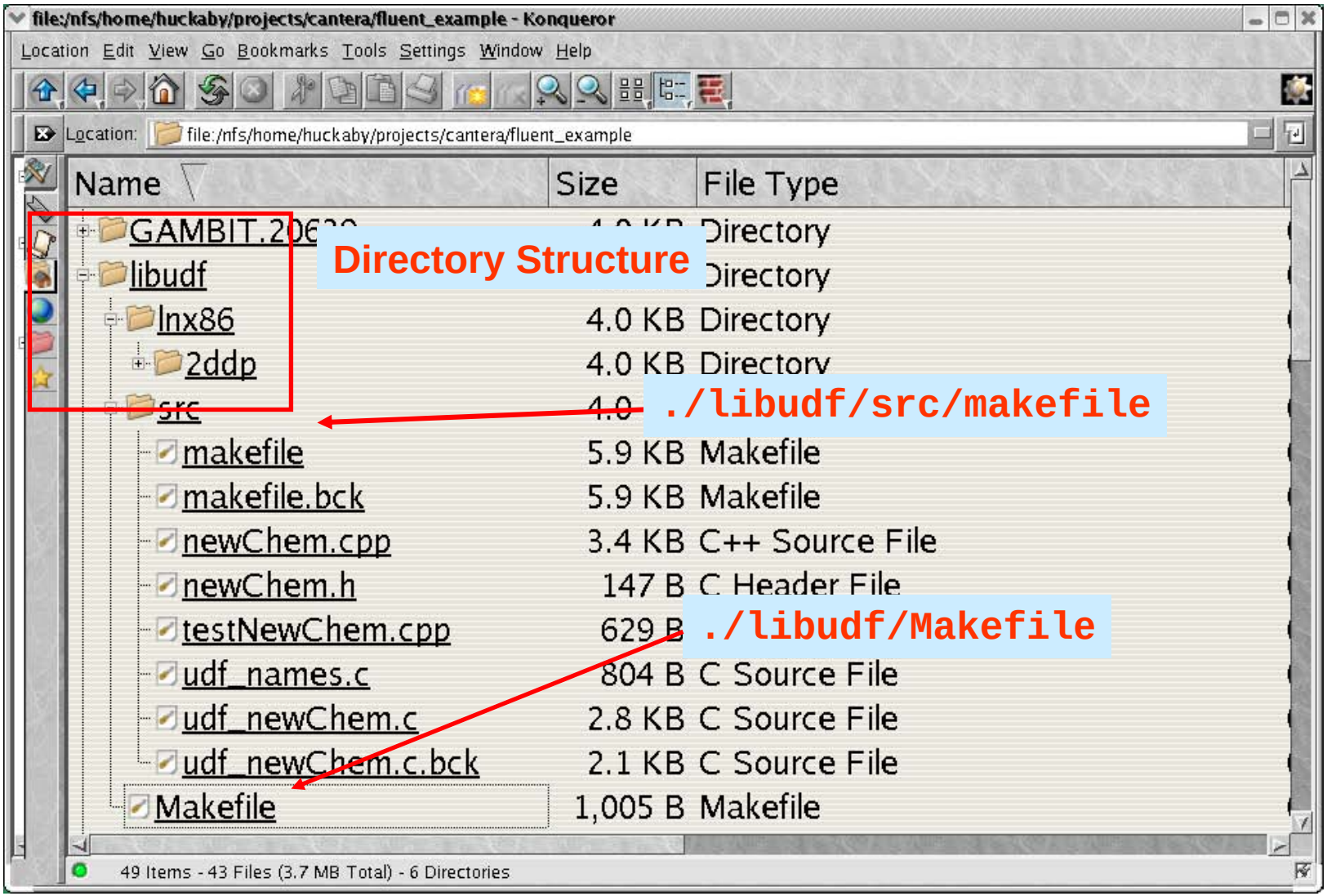
1. Create `libudf` directories structure
2. Create functions to manipulate Cantera objects (`newChem.cpp`, `newChem.hpp`)
3. Test functions (`test_newChem.cpp`)
4. Modify `libudf/Makefile`
5. Modify `libudf/src/makefile`
6. Create “hooks” between FLUENT and functions (`udf_newChem.c`)
7. Manually compile sources
8. Load udf_newChem into FLUENT
9. Execute udf’s

Create `libudf` directory structure

1. Create empty file named `udf_newChem.c`
2. Define/user-defined/compiled – build button
3. This will create the structure and “makefiles” on the next page



libudf Directory Structure



Create functions which access Cantera `newChem.cpp`

- Create `in libudf/src`
  - contains objects to access Cantera functionality
  - extern “C” { … to “de-mangle” c++ functions

```
static Cantera::IdealGasMix *_gas;
static vector<int> _speciesMap;
static vector<int> _speciesMapInv;
static vector<double> _massFractions;
static int _nCantera = 0;
static int _nFluent = 0;
static Cantera::ChemEquil *_equil;

extern "C" {
```

Create a header file for functions `newChem.h`

- Create in libudf/src

```
void newChem_newChem( const int nFluent );
void newChem_addToMap( const int fluent_index, const char *name);
void newChem_equil_HP(double *T, double *p, double Y[]);
```

- Extern “C” is not needed
- Three functions
  - Initialization
  - map between species indicies
  - equilibrate the composition

Test the newly created functions

- Create a directory `test`
  - Create/Edit `Makefile` (look at `cxx_examples`)

```
CANTERA = /nfs/home/huckaby/projects/cantera/cantera_dev
CANTERA_INC = $(CANTERA)/include/cantera
CANTERA_LIB = -L$(CANTERA)/lib/cantera/1.5.4 \
        -lzeroD -ltransport -lconverters -lcantera -lrecipes \
        -lcvode -lctlapack -lctmath -lctblas -ltpx -lctcxx \
        -L/usr/lib/gcc-lib/i386-redhat-linux/2.96 -lg2c -lstdc++\
        -lgcc -lm

CXX = g++
CXX_FLAGS = -g -O
CXX_INC = $(CANTERA_INC)
```

Makefile cont'd

```
default: test.exe

newChem.o: ../libudf/src/newChem.cpp
	echo $(CANTERA_INC)
	g++ -g -c -I$(CANTERA_INC) -I/usr/include/g++
../libudf/src/newChem.cpp

testNewChem.o: testNewChem.cpp
	g++ -g -c -I$(CANTERA_INC) -I../libudf/src testNewChem.cpp

test.exe: newChem.o testNewChem.o
	g++ -g testNewChem.o newChem.o $(CANTERA_LIB) -o test.exe

clean:
	rm -f *.so
	rm -f *.o
	rm -f *_wrap.cxx
```

Modify `libudf/Makefile`

- One change
  - Add `.cpp` extension to list of files copied
  - This makes file recursively copy the files in `libudf/src` to the appropriate library directory
    - `libudf/lnx86/2ddp`
    - `libudf/lnx86/3d`
    - `libudf/lnx86/2ddp_host (?)`

Modify libudf/src/makefile

- Add functionality of the test makefile to the udf makefile
- Define a few environment variables

  1. $(CANTERA_DIR) = <install location>
  2. $(CXX_SRC) = newChem.cpp
  3. $(CXX_INC) = $(CANTERA_DIR)/include/cantera
  4. $(CXX_LIB) = $(CANTERA_DIR)/lib/cantera/1.5.4
- Create Rule for compiling .cpp code
- Add $(CXX_LIB) to the link step

Functions which access FLUENT's function library

Set to one if init_newChem has been executed

```
int is_init = 0;
```

Local variables to hold the Gas State

```
real *y;
real T = 300;
real p_gauge = 0.0;
```

Get/Set T,p and Y to/from FLUENT

```
void getFluentComposition(cell_t cell,Thread *thread)
void getFluentComposition(cell_t cell,Thread *thread)
```

Initialize interface with Cantera calls: `newChem_newChem( )`

```
void init_newChem()
```

Execute equilibrate function on a single cell

```
void equil_HP(cell_t cell, Thread *thread)
```

Create “Hooks” into FLUENT’s execution loop

FLUENT “hook” for initialize function

```
DEFINE_ON_DEMAND(demand_initialize)
{
    init_newChem();
}
```

FLUENT “hook” for equilibrium function

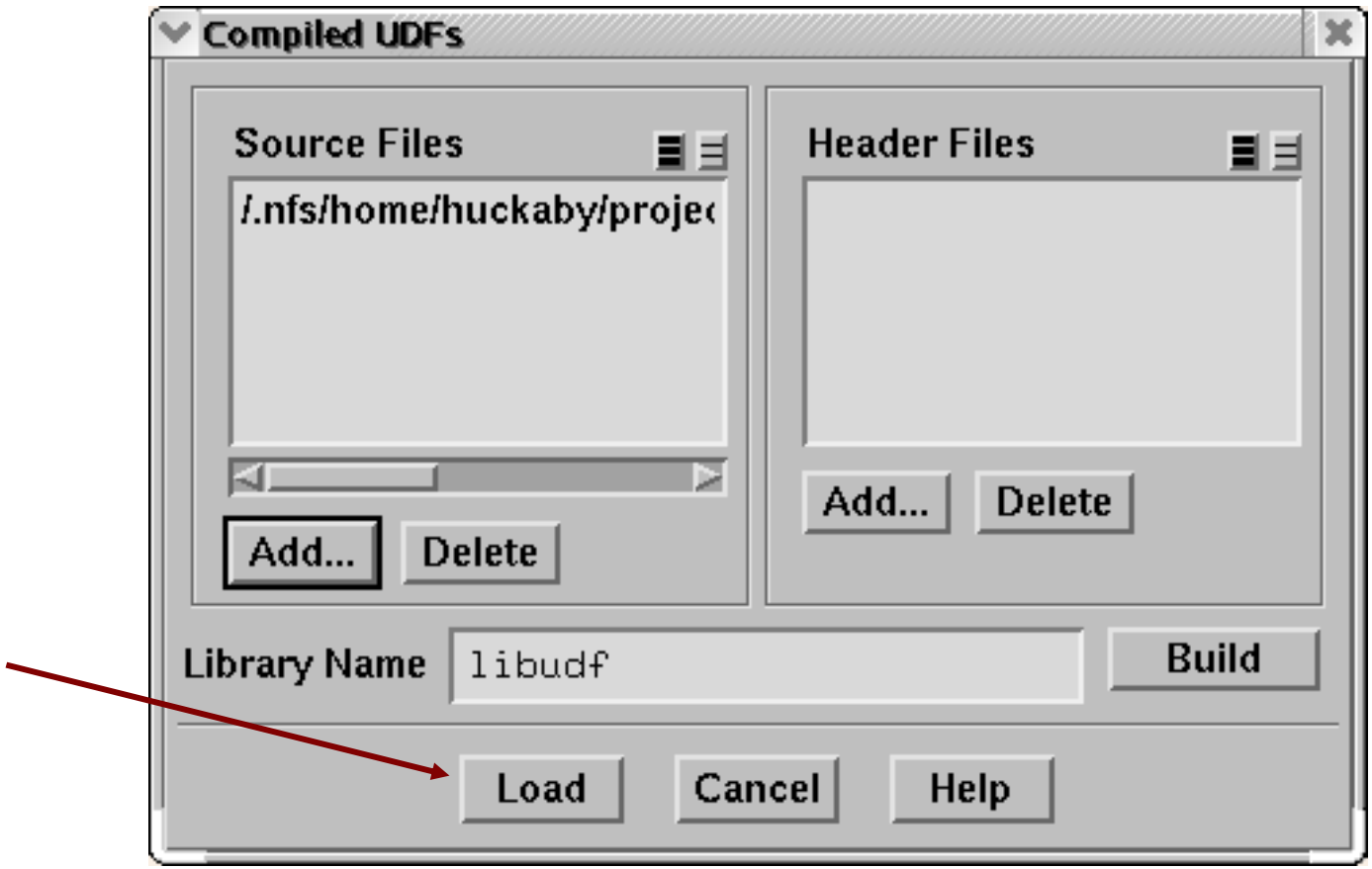
```
DEFINE_ON_DEMAND(demand_equilirbium)
{
    if (is_init == 0) init_newChem();
    thread_loop_c( thread, domain )  {
        begin_c_loop( cell, thread) {
            if ( FLUID_THREAD_P(thread) )
                equil_HP(cell,thread);
        end_c_loop(cell, thread)  }
    }
}
```

Compile the library

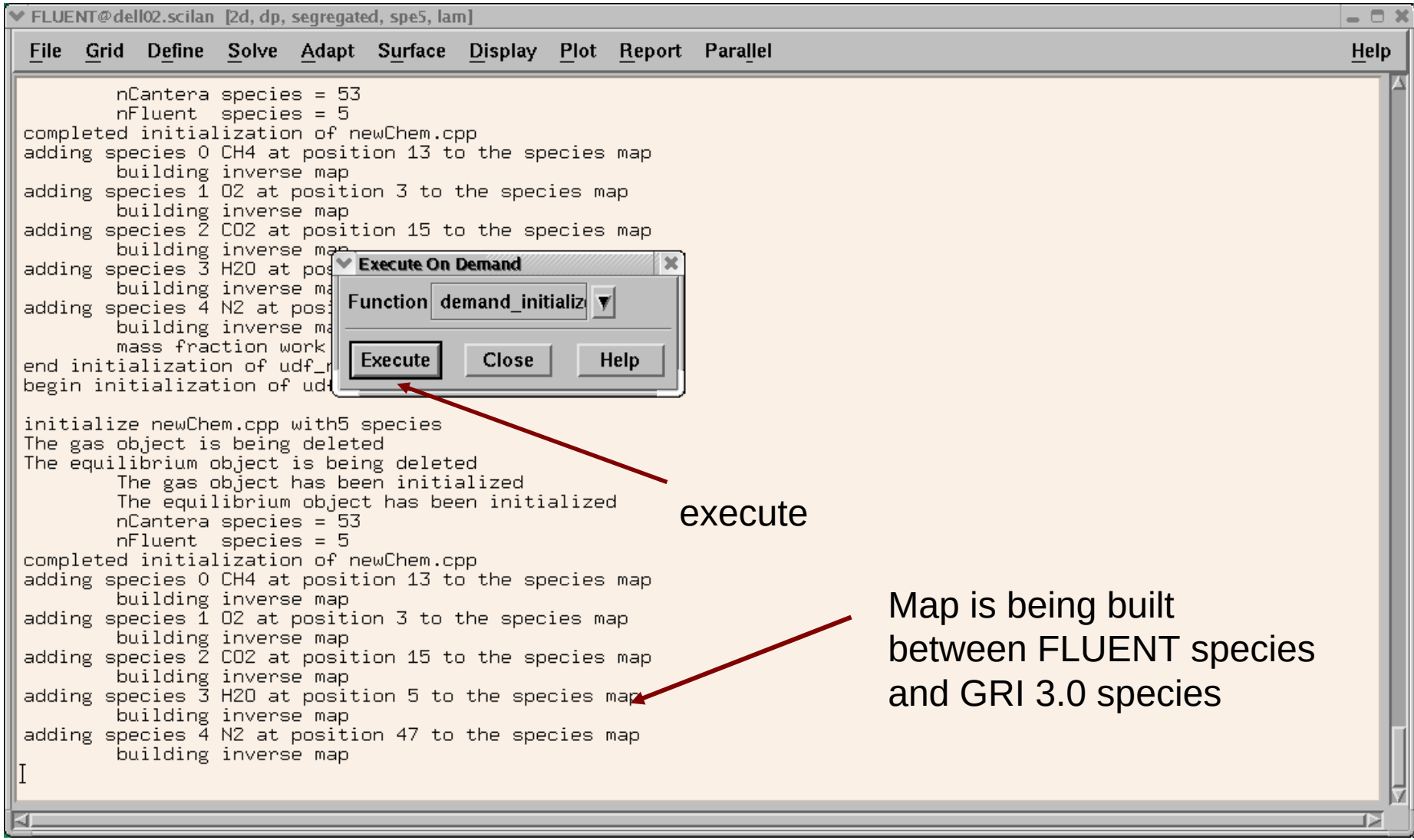
- Source must be compiled **manually** as opposed to the **build** button
- Type “make” in the directory libudf

Load the udf_newChem into FLUENT

`Define/user-defined/compiled` – hit load button



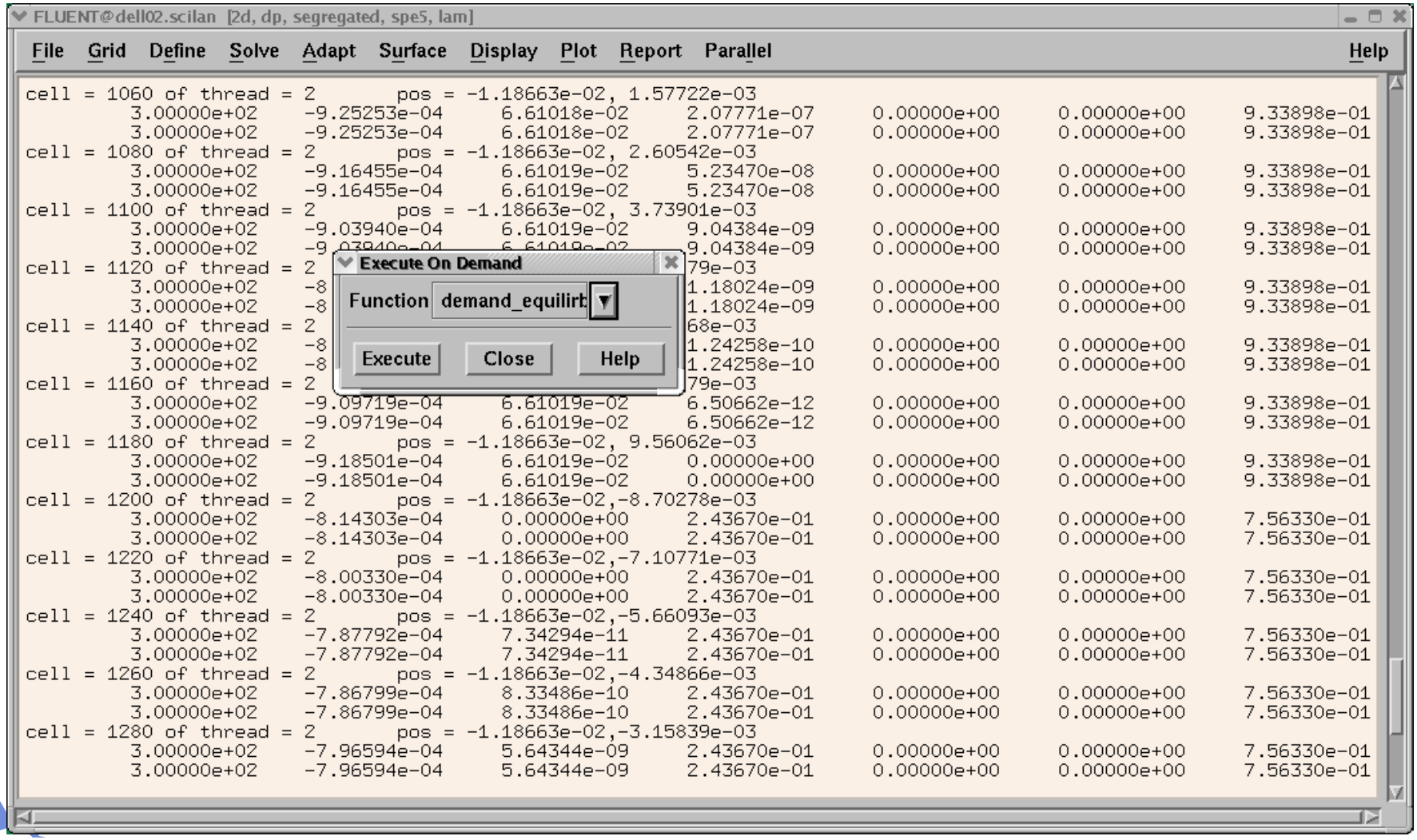
Execute demand_initialize



execute

Map is being built between FLUENT species and GRI 3.0 species

Execute demand_equil



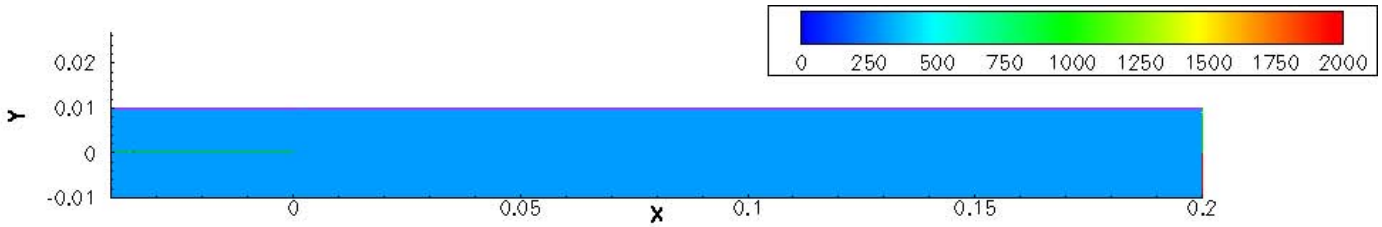
Confined Reacting Mixing Layer

- **Geometry:**
  - Injectors: H = 1 cm, L = 4 cm
  - Chamber: H = 2 cm, L = 20 cm
- **Chemistry:** 1 step – methane-air oxidation
- **Boundary Conditions:**
  - *Air inlet: V = 16 cm/s, Y_{O2} = 0.24367, T = 300K
  - *Fuel inlet: V = 16 cm/s, Y_{CH4} = 0.0661, T = 300K
  - Walls: adiabatic, non-reacting
  - Outlet: constant pressure, p = 1 atm
- **Solver:**
  - velocity-pressure coupling - SIMPLE
  - u, v, T, ρ, Y - QUICK
  - p - 2nd-order

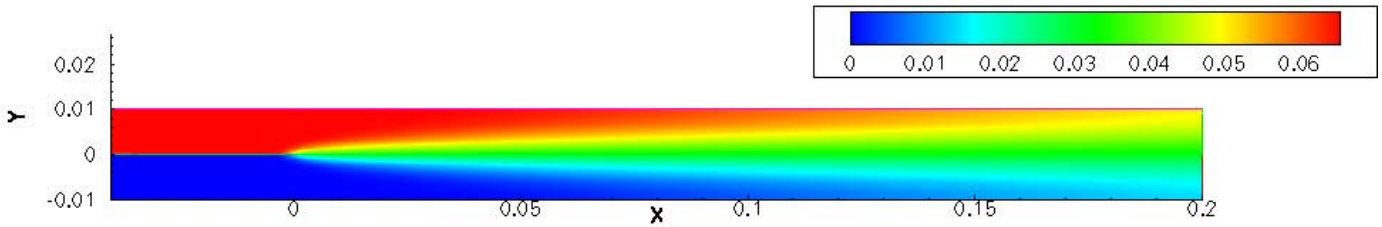
* Jones and Becker (Comb. Flame 19, 351)

Before demand_equil

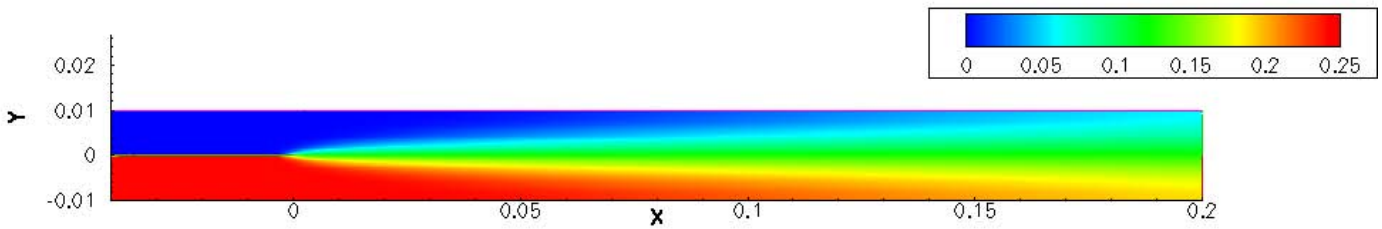
T



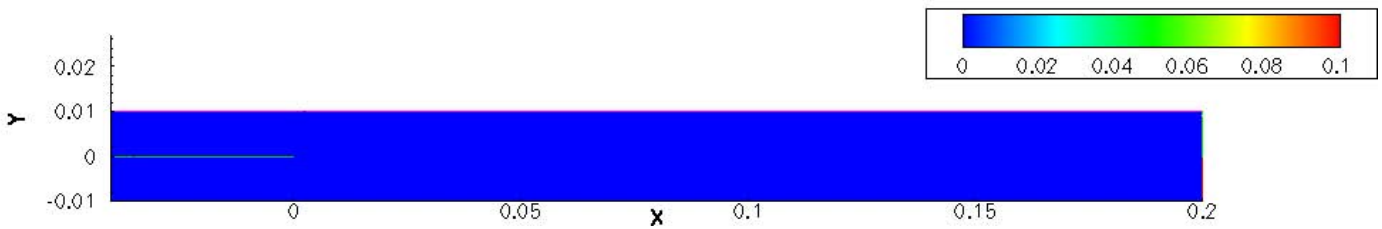
CH_4



O_2

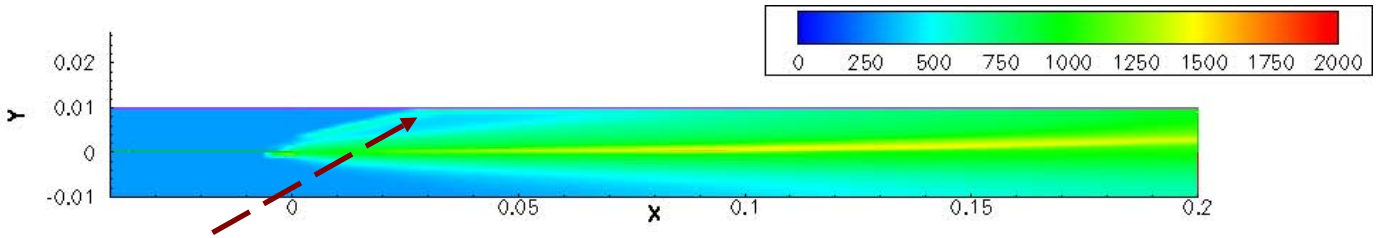


H_2O



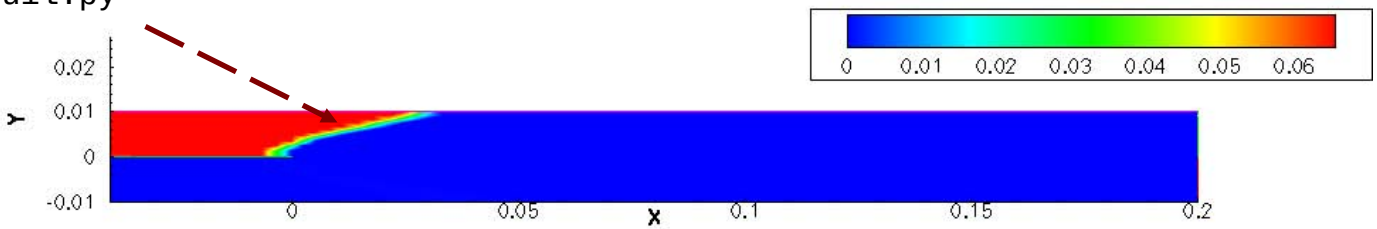
After `demand_equil`

T

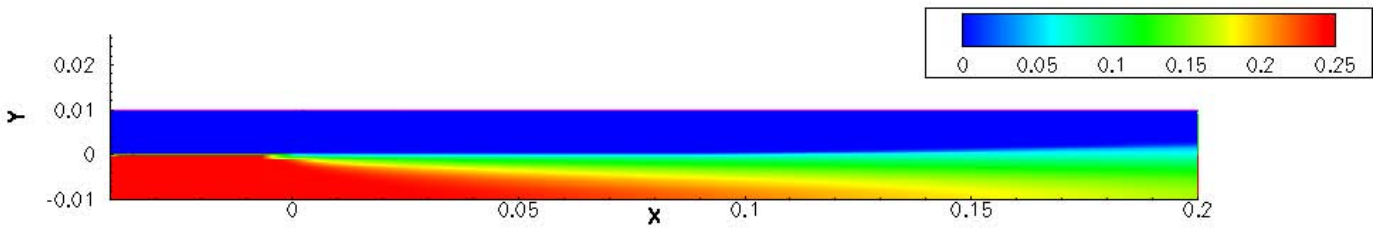


*see `testEquil.py`

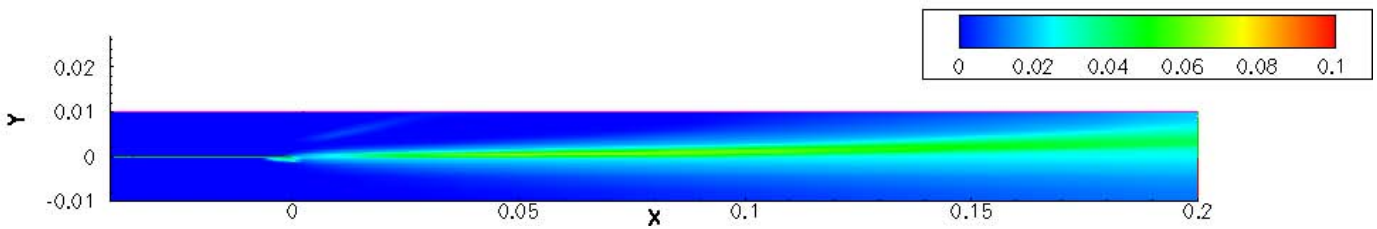
CH_4



O_2



H_2O



Contact Information

- Please contact me if you have any questions or find errors in the text or example files

- NETL – www.netl.doe.gov
- E. David Huckaby – huckaby@netl.doe.gov