Instructional workshop on OpenFOAM programming
LECTURE # 0

Pavanakumar Mohanamuraly

April 17, 2014

Outline

Introduction

Compiling, linking and executing

Aims of the workshop

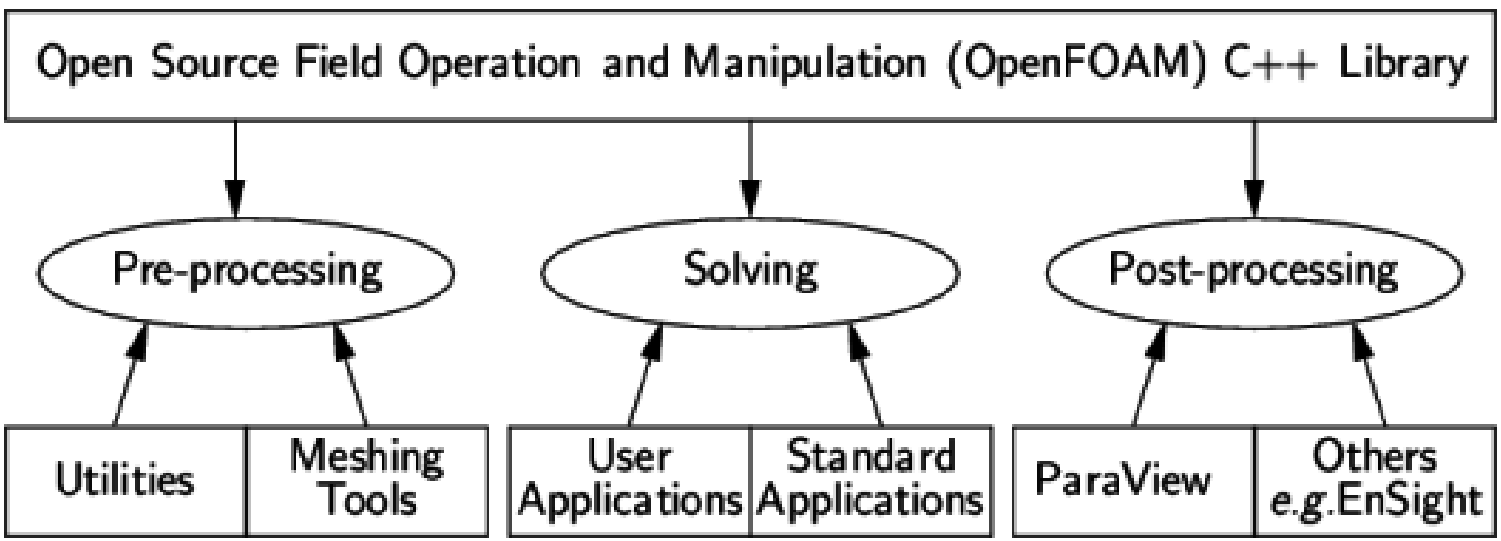
- Broad overview of OpenFOAM library and data-strucutres
- Hands on experience writing codes using the library
- Basics of operators, boundary conditions and parallelization
- Implementing solvers from scratch

Disclaimer

- Teach only OpenFOAM 1.7.x version
- Many changes/deprecations introduced in 2.x versions
- Will not cover problem/domain specific information
- Will not cover C++ programming or CFD fundamentals
- Complete hands-on approach

Overview of OpenFOAM [1]

- A oversized open-source CFD solver, library and tools
- Lacks good documentation (except the Doxygen docs)
- Solvers for complex EM and fluid problems
- Implement new or adapt existing solvers
- Mostly low fidelity solvers ($<= 2^{nd}$ order)
- (new) Adjoint solvers

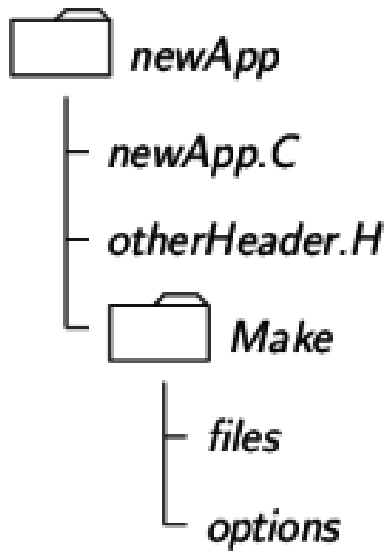


[1] figure source: http://openfoam.org/docs/user/userch1.php

Compiling source - *wmake* [2]

OpenFOAM uses the *wmake* compile system similar to *make*

- Create folder *Make* in source root
- Dependency for executables provided in *Make/files*
- Compile and link library flags in file *Make/options*



[2] figure source:
http://openfoam.org/docs/user/img/user261x.png

Make/files

Example **files**

```
source.cpp
another_source.cpp

EXE = my_executable
```

- First line onwards - the source files to be compiled
- *EXE* macro gives the name of the final executable or libso

Make/options

Example **options**

```
EXE_INC = \
    -I$(LIB_SRC)/finiteVolume/lnInclude \
    -I$(HOME)/boost/include/ \
    -g

EXE_LIBS = \
    -lfiniteVolume
```

- Include search path using *EXE_INC* macros
- Library search path using *EXE_LIB* macros
- *LIB_SRC* specifies the root path to OF source

Hands on # 0

Compile a simple example using OF *wmake* utility

Example code - **hello_foam.cpp**

```
/// Finite volume CFD application header
#include "fvCFD.H"

int main( int argc, char *argv[] ) {
  Info << "Hello FOAM !\n";
  return 0;
}
```

- Info object is the output stream (similar to stdout/err)
- Specializes output of FOAM data-structures

Make/files and *Make/option*

files

```
hello_foam.cpp

EXE = hello_foam
```

options

```
EXE_INC = \
    -I$(LIB_SRC)/finiteVolume/lnInclude \
    -g

EXE_LIBS = \
    -lfiniteVolume
```

Time to dive in !

OpenFOAM data-structures - List # 0

- Primitive types
- Dimensioned types
- **Info** stream output
- **Time** object
- **argList** Command-line parsing
- **IOdictionary** Input file parsing

OpenFOAM classes - Primitive types

- *label* - usually an integer type but depending on the compiler options it can be long integer
- *scalar* - double or float depending on the version of OF compiled
- *vector* - 3D scalar variable a $= [a_x, a_y, a_z]$
- *tensor* - $[a_{xx}, a_{xy}, a_{xz}; a_{yx}, a_{yy}, a_{yz}; a_{zx}, a_{zy}, a_{zz}]$
- *point* - same as vector
- *prefixList* - Array of type *prefix* (e.g., labelList)
- *prefixListList* - Array of arrays of type *prefix*
- *word* - Inherited from C++ string object
- *fileName* - *word* list which understands folder hierarchy

OpenFOAM classes - Operations on primitive types

Table : Vector/Tensor primitive operations

Operator	FOAM notation
Addition	a+b
Inner Product	a & b
Cross Product	a ^ b
Outer Product	a * b
Vector magnitude	mag(a)
Transpose	A.T()
Determinant	det(A)

OpenFOAM classes - Dimensioned types

- OpenFOAM operators automatically checks dimension consistency using *dimensionSet* object
- Primitives have dimensioned counter part (e.g., *dimensionedScalar*, *dimensionedVector*, etc)

```
/// Pressure units kgm^{-1}s^{-2}
dimensionSet pressureUnits( 1, -1, -2, 0, 0, 0, 0)
///          Dimension No.    1   2   3  4  5  6  7
```

No.	Property
1	Mass
2	Length
3	Time
4	Temperature
5	Quantity
6	Current
7	Luminous intensity

Hands on

Example: dim_test.cpp

```
/// Finite volume CFD application header
#include "fvCFD.H"

int main( int argc, char *argv[] ) {
  dimensionedScalar inputPressure =
  dimensionedScalar
  (
     "pressure", /// A name field
     dimensionSet( 1, -1, -2, 0, 0, 0, 0 ),
     1.0 /// Value to initialize
  );
  Info << inputPressure << "\n";
  return 0;
}
```

Try implementing velocity vector U with dimensions ms^{-1}.

Hands on

Non-dimensional constants

```
dimensionedScalar Mach =
  dimensionedScalar
  (
     "dimless", /// A name field
     dimless,
     1.0 /// Value to initialize
  );
  Info << Mach << "\n";
  return 0;
}
```

- `const dimensionSet dimless(0, 0, 0, 0, 0, 0, 0);`
- Other pre-defined dims available *dimMass*, *dimLength*, *dimTime*, *dimArea*, *dimVolume*, *dimDensity*, *etc)*

Hands on

A common problem encountered

Non-dimensional constants

```
inputPressure = Mach; /// error in dimension
    consistency
inputPressure.value() = Mach.value(); /// A hack to
    set the value by force
```

- Hack the *dimensioned* $<$ *Type* $>$ object to set values, which are dimensionally inconsistent using the *value*() member function

OpenFOAM classes - **Info** stream output

Already familiar with this so skipping ...

OpenFOAM classes - **argList** command-line parsing

- Parse command line inputs and options
- Register the command line option using *validOptions*

```
Foam::argList::validOptions.set( "mach", "Mach" );
Foam::argList::validOptions.set( "boolean", "" );
```

- Suppress printing OpenFOAM banner

```
Foam::argList::noBanner();
```

- Create an object instance of *argList*

```
Foam::argList args(argc, argv);
```

- Check for argument presence

```
args.optionFound("mach");
```

OpenFOAM classes - **argList** command-line parsing

- Read option from command line

```
scalar M;
args.optionReadIfPresent("mach", M);   /// or
args.optionRead("mach", M);
```

OpenFOAM classes - **Time** object and *control dictionary*

- Solver time and iteration control
- Controls all other allied operations tied to the above
  - Writing variable values with iteration
  - Reading variable values with iteration
- Necessary to create FOAM *objectRegistry*
  - Necessary for almost all derived classes (mesh, fields, etc)
- Constructor requires an input file called *control dictionary*
  - Dictionary (input) files are read/written using *IOdictionary* objects
  - All FOAM applications use the string

```
Foam::Time::controlDictName = "controlDict";
```

IOdictionary objects discussed after Time object - will postpone some things for later for clarity sake

OpenFOAM classes - **Time** object and *control dictionary*

controlDict file contents

```
startFrom       startTime;
startTime       0;
stopAt          endTime;
endTime         10.0;
deltaT          0.0005;
writeControl    timeStep;
writeInterval   1000;
purgeWrite      0;
writeFormat     ascii;
writePrecision  6;
writeCompression uncompressed;
timeFormat      general;
timePrecision   6;
runTimeModifiable yes;
```

OpenFOAM classes - **Time** object and *control dictionary*

Creating *Foam* :: *Time* object

```
Foam::Time runTime
(
  Foam::Time::controlDictName, /// Dictionary file
  args.rootPath(), /// Case root
  args.caseName() /// Case Name (cavity, etc)
);
```

- Advice to stick to *runTime* for *Time* object name
- and *Foam* :: *Time* :: *controlDictName* for *control dictionary*
- Possible to have multiple *runTime* objects in the same code

OpenFOAM classes - **objectRegistry** class

- Hierarchical database that FOAM uses to organize its model-related data
- Complemented by *IOobject*, and *regIOobject*
- *IOobject* provides standardized input / output support
- also gives access to *Foam* :: *Time*, which is at the root of *objectRegistry*
- *regIOobject* automatically manages the registration and deregistration of objects to the *objectRegistry*

OpenFOAM classes - **IOdictionary** input file parsing

- FOAM has elegant class for file I/O called *IOdictionary*
- Derived types like fvMesh, fields, etc use this object for I/O
- We saw *Foam* :: *Time* object using this to read the *control dictionary* file

Creating *IOdictionary* object

```
IOdictionary ioDictObj
(
  IOobject
  (
    "myDictFile", /// The dictionary file
    "", /// Relative path (from case root)
    runTime, /// The Time object
    IOobject::MUST_READ, /// Read for constructor
    IOobject::NO_WRITE /// Foam::Time writeControl
  )
);
```

OpenFOAM classes - Elements of a **dictionary** file

dictionary header

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "";
    object      myDictFile;
}
```

Dictionary file entry

```
keyword value;
```

OpenFOAM classes - **IOdictionary** parsing *vector*

- Parse inputs using *lookup*() member function
- Use casting to cast to the correct data-type

Parsing *vector* data-type

```
vector vec_in = vector( ioDictObj.lookup("vec_in") );
```

Dictionary file entry

```
vec_in (200.0 400.0 800.0);
```

OpenFOAM classes - **IOdictionary** parsing *sunDict*

- Parse inputs using *lookup*() member function
- Use casting to cast to the correct data-type

Parsing *vector* data-type from *subDict*

```
vec_in = vector( ioDictObj.subDict("subDict").lookup("
    vec_in") );
```

Dictionary file entry

```
subDict
{
  vec_in (0.0 0.0 0.0);
}
```

OpenFOAM classes - **List** object

- Equivalent of C++ vector class
- **IOdictionary** provides easy file I/O of *List*

Parsing *vector* data-type from *subDict*

```
List<vector> myList = List<vector>( ioDictObj.lookup("
    myList") );
```

Dictionary file entry

```
myList 3
(
   ( 0.0 0.0 0.0 )
   ( 1.0 0.0 0.0 )
   ( 0.0 1.0 0.0 )
);
```

IOdictionary hands-on

Create argList and Time objects

```
#include "fvCFD.H"

int main(int argc, char *argv[])
{
  /// Init the args object
  Foam::argList args(argc, argv);
  /// Foam Time object
  Foam::Time runTime
  (
    Foam::Time::controlDictName,
    args.rootPath(),
    args.caseName()
  );
```

IOdictionary hands-on

Create IOdictionary object

```
/// Input file dictionary
IOdictionary ioDictObj
(
    IOobject
    (
        "myDictFile", "",
        runTime,
        IOobject::MUST_READ,
        IOobject::NO_WRITE
    )
);
```

IOdictionary hands-on

Refer previous slides and

- Write code to parse a vector and List of vector from file
- Write code to parse vector from a *subDict*

IOdictionary hands-on

Preparing input files

- *Foam* :: *Time* requires one to define *controlDict* dictionary file
- *myDictObj* requires one to define corresponding dictionary file

IOdictionary hands-on - Bummer !

- Read in a simple scalar value
- Use casting to cast to the correct data-type

Parsing *scalar* data-type

```
scalar scalar_in( ioDictObj.lookup("scalar_in") );
```

Dictionary file entry

```
scalar_in 200.0;
```

IOdictionary hands-on - Bummer !

- Error in previous example (simple scalar not implemented)
- Use *dimensionedScalar* instead

Parsing *scalar* data-type

```
dimensionedScalar scalar_in( ioDictObj.lookup("scalar_in
    ") );
```

Dictionary file entry

```
scalar_in dimless [ 0 0 0 0 0 0 0 ] 200.0;
```

Note that input for dimensioned type same as that of output

End of day 1