ANSYS

Chapter 1

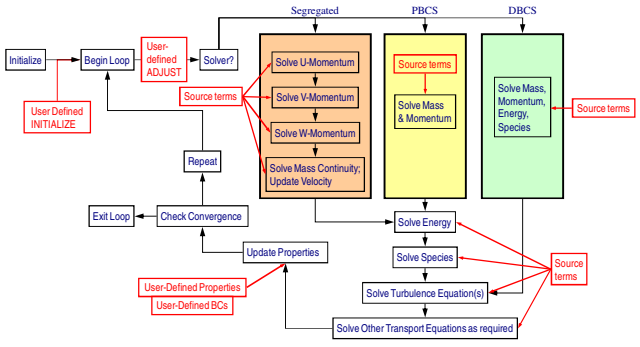
Introduction and FLUENT Data Structure

Advanced FLUENT User-Defined Functions

Introduction

- The FLUENT solver is a general-purpose code. In order to customize the FLUENT solver, users can use their own C-codes called user-defined functions (UDFs) to accomplish:
  - Special boundary conditions
  - Customized or solution dependent material properties
  - New physical models
  - Reaction rates
  - Source terms
  - Customized post-processing
  - Solving user-supplied partial differential equations
  - More ….

User Access to the FLUENT Solver



C Programming (1)

- Basic syntax rules:
  - each statement must be terminated with a semicolon ;
  - comments can be inserted anywhere between /* and */
  - variables must be explicitly declared (unlike in FORTRAN)
  - **compound statements** must be enclosed by braces { }
  - functions have the following format:

```
return-value-type function-name (parameter-list)
        { function body }
```

  - Macros are defined in the header files, they can be used just like functions
- Built-in data types: int, float, double, enum, boolean:

```
int niter, a; /* declaring 'niter' and 'a' as integers */
float dx[10]; /* 'dx' is a real array with 10 members, the
                 array index always starts from dx[0] */

enum {X, Y, Z}; /* X, Y, Z are enumeration constants 0, 1, 2 */
```


C Programming (2)

- **`pointer`** is a special kind of variable that contains the memory **`address`**, not content, of another variable
- **`Pointers`** are declared using the * notation:

```
int *ip; /* ip is declared as a pointer to integer */
```

- We can make a **`pointer`** point to the address of predefined variable as follows:

```
int a=1;
int *ip;
ip = &a; /* &a returns the address of variable a */
printf("content of address pointed to by ip = %d\n", *ip);
```

- **`Pointers`** are also used to point to the beginning of an array
  - Thus **`pointers`** are used to address **`arrays`** in C
- **`Array`** variables can be defined using the notation **`name[size]`** where **`name`** is the variable name and **`size`** is an integer which defines the number of elements in the array (from 0 to size-1)

C Programming (3)

- Operators

`=` (assignment)

`+, -, *, /, %` (modulus)

`<, >, >=, <= , ==, !=`

`++` increment; **`i++`** is "post-increment": use the *current* value of **`i`** in the expression, then increment i by 1 (**`i=i+1`**)

`--` decrement: **`j--`** post-decrement, use the current value of **`j`**, then decrement j by 1 **`(j=j-1)`**

`+=` addition assignment:

```
agg += single; /* it means agg=agg+single;  */
```

`*=` multiplication assignment, `-=` subtraction assignment,
`/=` division assignment

C Programming (4)

- **Basic control structures**

```
if ( … )
  <statement>;

if ( …)
  <statement>;
else
  <statement>;

if ( …)
   <statement>;
else if ( … )
  <statement>;
```

For Loops:

```
for ( k=0; k < NUM; k++ )
    <statement>;
```

While Loops:

```
while ( … )
<statement>;
```

Conditional Operator (? :)

(condition ? operand a : operand b)

example:

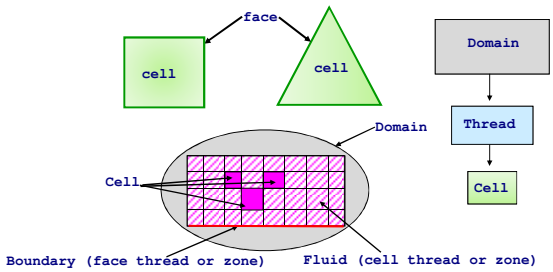
```
real At = (rp_axi ? At*2*M_PI : At );
```

(false: At; true: At*2*M_PI)

CFD Programming in FLUENT

- We (as CFD programmers in FLUENT) want to know how FLUENT organizes data so that we know:
  - How to access mesh information for a particular cell zone or a face zone
    - Cell centroids, cell volumes, the neighbors, etc.
    - Face centroids, face areas, face normal directions (vectors), etc.
  - How to access field data for cells (and faces): pressure, velocity, density, etc.
  - How to efficiently loop through these cells or faces in the codes
- How to supply customized source terms, boundary conditions, and fluid properties, etc., to the solver
- How to modify the behaviors or specific model parameters for various physical models as in turbulence, reactions kinetics, multiphase, and dynamic mesh, etc.
- How to implement user's own governing equations in the finite-volume framework of FLUENT solver

Data Structure Overview



The Domain

- Domain is the set of connectivity and hierarchy info for the entire data structure in a given problem for single phase flows. It includes:
  - all fluid zones ('fluid threads')
  - all solid zones ('solid threads')
  - all boundary zones ('boundary threads')
- Cell: Cell is the computational unit, conservation equations are solved over each cell
- Face: direction is in the outward normal
- Threads: represent the collection of cells or faces; a Thread represents a fluid or solid or boundary zone
- multiphase simulations (singlephase simulations use single domain only)
  - Each phase has its own "Domain-structure"
  - Geometric and common property information are shared among 'sub-domains'
  - Multiphase UDF will be discussed later

The Threads

- A **`Thread`** is a sub-set of the **`Domain`** structure
- Individual '**`fluid`**', '**`solid`**' and each '**`boundary`**' zones are identified as '**`zones`**' and their datatype is **`Thread`**
- '**`Zone`**' and '**`Thread`**' terms are often used interchangeably
- Some further details about **`Zone/Thread ID`** and **`Thread-datatype:`**
  - Zones are identified at mesh level with an integer **`ID`** in the **`Define/Boundary Condition`** panel
  - **`Threads`**, a Fluent-specific datatype, store structured information about the mesh, connectivity, models, property, etc. all in one place
  - Users identify zones through the **`ID`**'s
  - **`Zone/Thread-ID`** and **`Thread`**s are correlated through UDF macro's

Cell and Face Datatypes

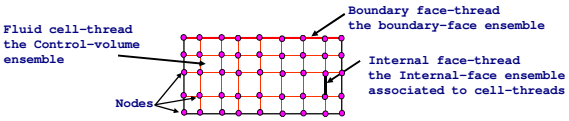
- Control volumes of fluid and solid zones are also called '**`cell`**s' in FLUENT
  - The data structure for the cell zones is typed as '**`cell_t`**' (the cell thread index)
  - The data structure for the cell faces is typed as '**`face_t`**' (the face thread index)
- A fluid or solid zone is called a cell zone, which can be accessed by using cell threads
- Boundary or internal faces can be accessed by using face threads

Some additional info on Faces

- Each Control volume has a finite number of faces
  - Faces on the boundary are also typed '**face_t**'; their ensemble are listed as boundary **face-threads** with the fluid & solid cell-threads under **Define-Boundary_Condition** panel
  - Those faces which are inside the flow-domain and do not share any external boundary are not accessible from GUI (because you do not need them)
  - They can still be accessed from User-Defined-Functions

Fluent UDF Data Structure Summary

- The data structure for accessing a cell zone is typed as '**cell_t**' (the cell thread index); the data structure for faces is typed as '**face_t**' (the face thread index)



Type	Example	Declaration
Domain	*d	**d** is a pointer to domain thread
Thread	*t	**t** is a pointer to thread
cell_t	c	**c** is cell thread index
face_t	f	**f** is a face thread index
Node	*node	**node** is pointer to a node

Fluent UDF Data Structure Summary (2)

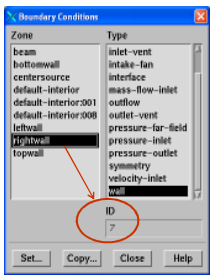
- Each thread (zone) has a unique integer ID available in the boundary condition panel (or can be listed by the list-zone TUI command: **/grid/modify-zones/list-zones**)
- Given a correct ID, the **Lookup_Thread** macro can retrieve the *thread pointer*

```
int ID=7;
Thread *tf=Lookup_Thread(domain, ID);
```

- Conversely, given a thread pointer tf, the zone ID can be retrieved

```
ID=THREAD_ID(tf);
```

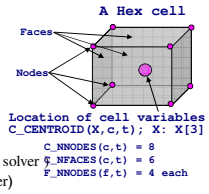
- Once we have the correct pointer (for a specific zone), we can access the members belonging to the zone without any problem. Thread pointer provides the leading address of the thread (zone)



Geometry Macros

- **C_NNODES(c, t)** Number of nodes in a cell
- **C_NFACES(c, t)** No. of faces in a cell
- **F_NNODES(f, t)** No. of nodes in a face
- **C_CENTROID(x, c, t)** x, y, z-coords of cell centroid
- **F_CENTROID(x, f, t)** x, y, z-coords of face centroid
- **F_AREA(A, f, t)** Area vector of a face;
- **NV_MAG(A)** Area-magnitude
- **C_VOLUME(c, t)** Volume of a cell
- **C_VOLUME_2D(c, t)** Volume of a 2D cell
  (Depth is 1m in 2D; 2*π m in axi-symmetric solver
- **NODE_X(nn)** Node x-coord; (nn is a node pointer)
- **NODE_Y(nn)** Node x-coord;
- **NODE_Z(nn)** Node x-coord;

Many more are available. See the FLUENT UDF Manual



Macros for Cell Variables

- **C_R(c,t)** Density
- **C_P(c,t)** Pressure
- **C_U(c,t)**, **C_V(c,t)**, **C_W(c,t)** Velocity components
- **C_T(c,t)** Temperature
- **C_H(c,t)** Enthalpy
- **C_K(c,t)** Turbulent kinetic energy
- **C_D(c,t)** Turbulent energy dissipation
- **C_YI(c,t,i)** Species mass fraction
- **C_UDSI(c,t,i)** User defined scalar

t is a cell-thread pointer, **c** is a cell thread index, **i** is an integer for indexing

Macros for Cell Variables (2)

- More cell variables
  - **C_DUDX(c,t)**
  - **C_DUDY(c,t)**
  - **C_DUDZ(c,t)**
  - **C_DVDX(c,t)**
  - **C_DVDY(c,t)**
  - **C_DVDZ(c,t)**
  - **C_DWDX(c,t)**
  - **C_DWDY(c,t)**
  - **C_DWDZ(c,t)**

  Velocity derivatives
  - **C_MU_L(c,t)** Laminar viscosity
  - **C_MU_T(c,t)** Turbulent viscosity
  - **C_MU_EFF(c,t)**
  - **C_DP(c,t)[i]** Pressure derivatives
  - **C_D_DENSITY(c,t)[i]** Density derivatives

Loop Macros in UDF

- `thread_loop_c(ct,d) { }` for loop over **cell** threads in domain d
- `thread_loop_f(ft,d) { }` for loop over **face** threads in domain d

(Note: ct, ft and d are pointers to Thread)

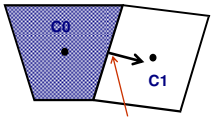
- `begin_c_loop(c, t)`
  `{...}`
  `end_c_loop (c,t)` for loop over cells in a given cell thread t

- `begin_f_loop(f, f_thread)`
  `{ ... }`
  `end_f_loop(f, f_thread)` for loop over all faces in a given face thread f_thread

Macros for Accessing Cells/Faces

- For any given face, the cell from which the face normal vector points *away* is called the **C0** cell; and the cell which the face normal vector points at is called the **C1** cell. The following program fragment calculate the total cell volume next to a given face zone with zone ID on the **C0** side:

```
Thread *tf, *t0;
face_t f;
cell_t c0;
real totalV=0.;
tf = Lookup_Thread(domain, ID);
t0 = THREAD_T0(tf);
begin_f_loop(f, tf)
  { c0=F_C0(f, tf);     /*get the c0 thread */
    totalV += C_VOLUME(c0, t0);
  }
end_f_loop(f, tf)
```



a face normal vector for the C0 cell

Top-Level UDF Macros

- User's own codes must use the top-level UDF macros in order to communicate with the FLUENT solver
  - Profiles : `DEFINE_PROFILE`
  - Source terms : `DEFINE_SOURCE`
  - Properties : `DEFINE_PROPERTY`
  - User-defined Scalars : `DEFINE_UNSTEADY`, `DEFINE_FLUX`, `DEFINE_DIFFUSIVITY`
  - Initialization : `DEFINE_INIT`
  - Global Functions : `DEFINE_ADJUST`, `DEFINE_ON_DEMAND`, `DEFINE_RW_FILE`
  - Wall-heat-flux : `DEFINE_HEAT_FLUX`
  - Model-Specific Functions : `DEFINE_DPM_…`, `DEFINE_SR_RATE`, `DEFINE_VR_RATE`, …

Refer to the UDF Manual for a complete list

The Header Files

- The udf-macros are defined in the `'udf.h'` file
- `udf.h` is a fluent header file in the `{Fluent installed directory}/Fluent12.y/src/` directory
- `udf.h` must be included at the top in each and every udf file
  - A file may contain more than one UDF
  - User can use multiple files for UDF
- Any UDF you might write **must** use one of the `'DEFINE_…'` macros from this `udf.h` file
- There are many more header files stored in the same directory can be browsed by users

```
#define DEFINE_PROFILE(name, t, i) void name(Thread *t, int i)
#define DEFINE_PROPERTY(name,c,t) real name(cell_t c, Thread *t)
#define DEFINE_SOURCE(name, c, t, dS, i) \
        real name(cell_t c, Thread *t, real dS[], int i)
#define DEFINE_INIT(name, domain) void name(Domain *domain)
#define DEFINE_ADJUST(name, domain) void name(Domain *domain)
#define DEFINE_DIFFUSIVITY(name, c, t, i) \
        real name(cell_t c, Thread *t, int i)
```