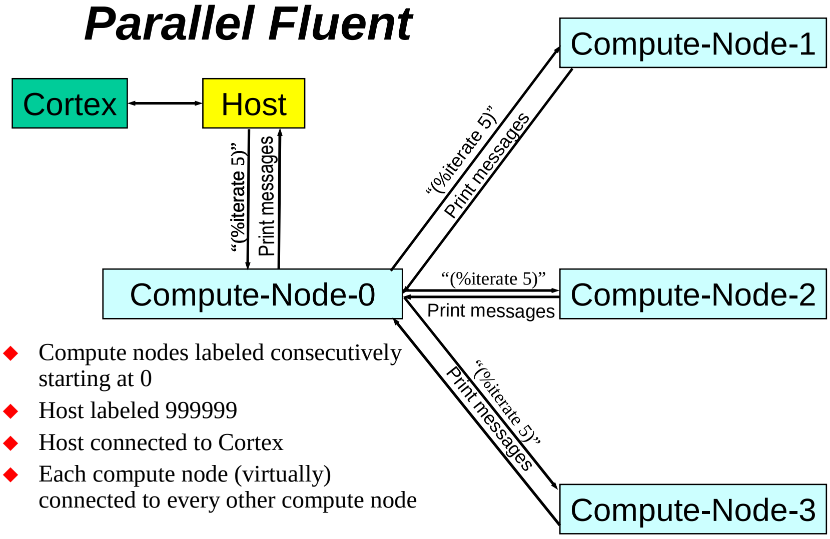
ANSYS

UDF in Parallel FLUENT

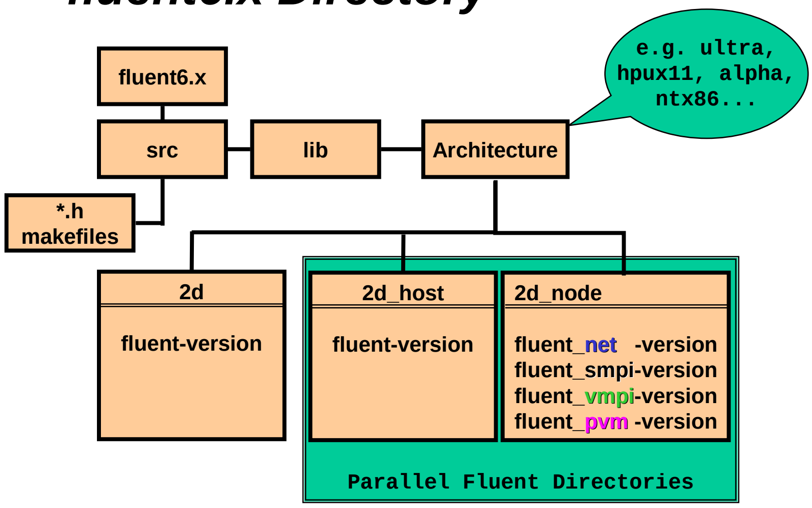
Advanced UDF
Modeling Course

Parallel Fluent



- Compute nodes labeled consecutively starting at 0
- Host labeled 999999
- Host connected to Cortex
- Each compute node (virtually) connected to every other compute node

fluent6.x Directory



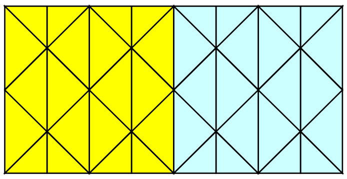
Intro to Compiler Directives

- "**#if**" is a compiler directive (similar to "**#define**")
- A "**#endif**" is used to close a "**#if**

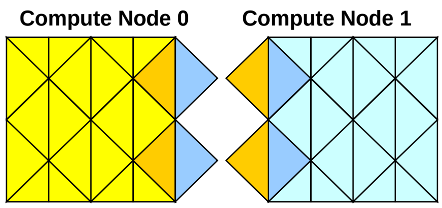
```
#if RP_NODE   /* Compute-Node */
#if RP_HOST   /* Host */
#if PARALLEL  /* Equivalent to #if RP_HOST||RP_NODE*/
#if !PARALLEL /* Serial */
#if RP_HOST
     Message("I'm the Host process \n");
#endif
#if RP_NODE
     Message("I'm the Node process number:%d \n", myid);
#endif
```

Partitioning (1)

- Domain Decomposition Technique: Splits the domain across Compute Nodes
- Because Fluent's algorithms expect a cell to be on both sides of an interior face, copies of the neighboring partition's cells are kept on each Node
- Compute Node 0 has copies of the cells on the other side of all partition faces and Compute Node 1 has corresponding cell copies from Node 0



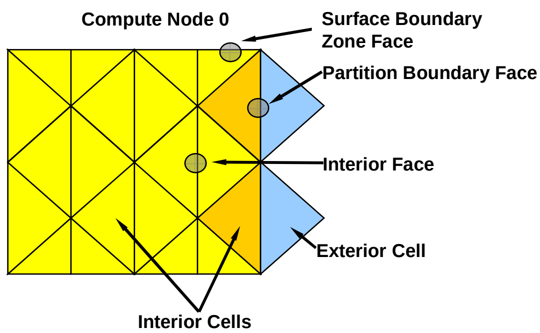
Domain Decomposition



Distribution across Compute Nodes

Partitioning (2)

The main cells of the partition are designated "Interior" cells and the additional copied cells from other Compute Nodes are designated "Exterior" cells
The Partition Boundary Faces are a special type of Interior face



Partitioned Thread Loop (1)

```
begin_c_loop(c,t)
  {
  }
end_c_loop(c,t)
```

- In parallel use, above loop construct loops through the Exterior cells too
- Use **begin_c_loop_int(c,t)** in all UDFs that are to be parallelized:

```
begin_c_loop_int(c,t)
  {
  }
end_c_loop_int(c,t)
```

- This loop excludes the exterior cells to replicate serial **begin_c_loop(c,t)**
- Another loop construct loops through the exterior cells only :

```
begin_c_loop_ext(c,t)
  {
  }
end_c_loop_ext(c,t)
```

- It is rarely used in UDFs and does nothing if compiled in serial version

Partitioned Thread Loop (2)

- Similar loops exist for faces:

```
begin_f_loop_all(f,t)
{…}
end_f_loop_all(f,t)

begin_f_loop_int(f,t)
{…}
end_f_loop_int(f,t)
```

- But you can simply use the standard loop and check to see if the face is “allocated” to this Thread using:

```
begin_f_loop (f,t)
{
  if(PRINCIPAL_FACE_P(f,t))
  {…}
}
end_f_loop(f,t)
```

Inter-Process Communication (1)

- Each compute node maintain local cache of individual variables
- Synchronization or make global reduction of such data involves communication in a particular order
- Consider the simple operation of passing a user defined cortex parameter that is set using scheme but is used in a UDF

Serial Code

```
DEFINE_INIT(set_temp,domain)
{
 real i_temp;
 i_temp = RP_Get_Real("user-temp");
 begin_c_loop(c,t)
  C_T(c,t)=i_temp;
 end_c_loop(c,t)
}
```

Combined (Serial & Parallel) Code

```
DEFINE_INIT(set_temp,domain)
{
 real i_temp;
#if !RP_NODE /* i.e. serial or host */
 i_temp = RP_Get_Real("user-temp");
#endif
host_to_node_real_1(i_temp);
#if !RP_HOST /* i.e. serial or node */
 begin_c_loop_int(c,t)
   C_T(c,t)= i_temp;
 end_c_loop_int(c,t)
#endif
}
```


Inter-Process Communication (2)

- To ensure same code to work for serial and parallel versions, negated compiler directives are mostly used:

```
#if !RP_NODE  /* i.e. serial or host */
#if !RP_HOST  /* i.e. serial or node */
#if !PARALLEL /* i.e. serial only    */
```

- The macro "`host_to_node_real_1(i_temp);`" is defined as a **Send** command in the Host version, a **Receive** command in the Compute Node versions and does **Nothing** in the Serial version
- The reciprocal command to `host_to_node_real_1()` is `node_to_host_real_1();`
- But this only sends the value of `temp` from `Node0` to the Host
- The formal broadcasting and host communication can be done as below:

```
temp = PRF_GRSUM1(temp);  /*This sums up temp over all nodes*/
                          /*All nodes now have temp=sum     */
node_to_host_real_1(temp);/*only Node0 sends data to Host   */
```

Global Reduction

This combination process is called "Reduction" and there are a number of ways to reduce your data depending on what you want:

1) If you want the total value over all the Nodes, you use a Summation Reduction
2) If you want the Max or Min over all the Nodes use a High or Low Reduction
3) If you want a logical test over all nodes use an And or Or Reduction

There are different macros depending on what data type you're sending:

```
count    = PRF_GISUM1(count);   /* Total Integer count */
min_temp = PRF_GRLOW1(min_temp);/* Global minimum */
PRF_GLOR(sonic_tests, 3, work); /* Arrays can be reduced too,
                                    needs a work array */
PRF_GRSUM4(v_x,v_y,v_z,v_mag);  /* 4 vars are reduced at a time */
```


Example UDF (1)

- Find totals and averages of a property over all the cells
- Purpose is to write an UDF that works for both Parallel and Serial solvers

```
#include "udf.h"
DEFINE_ON_DEMAND(av_pres_in_thread)
{int thread_id;
 real vol_sum=0.0, pres_sum=0.0;
#if !RP_HOST                                /* serial or node */
  cell_t c; Thread *t;
#endif /* !RP_HOST */
#if !RP_NODE                                /* serial or host */
  thread_id=RP_Get_Integer("udf/av_thread_id");
#endif /* !RP_NODE */
  host_to_node_int_1(thread_id);            /* Passes on  serial */
#if !RP_HOST                                /* serial or node */
  t= Lookup_Thread(Get_Domain(1), thread_id);
  begin_c_loop_int(c,t)                     /* Internal cells only*/
    {vol_sum  += C_VOLUME(c,t);
     pres_sum += C_P(c,t) * C_VOLUME(c,t);}
  end_c_loop_int(c,t)
#endif /* !RP_HOST */                       /* Continued */
```

Example UDF (2)

```
#if RP_NODE
  Message("Sub totals on Node %d: %f,%f\n",myid ,
                                  pres_sum ,vol_sum);
#endif /* RP_NODE */
  vol_sum  = PRF_GRSUM1(vol_sum);
  pres_sum = PRF_GRSUM1(pres_sum);
#if RP_NODE
  Message(”Reduced vals Node %d: %f,%f\n",myid ,
                                  pres_sum ,vol_sum);
#endif /* RP_NODE */
node_to_host_real_2(vol_sum,pres_sum);
#if !RP_NODE                              /* i.e., host or serial*/
  Message("Avg. pressure over Thread %d is %f Pa\n", thread_id,
                                             pres_sum/vol_sum);
#endif /* !RP_NODE */
}
```


Message0 ( )

- A function that can be run on node0 that prints directly to the cortex window
- Also works for serial processes

```
Message0("Average pressure over Thread %d ",thread_id);
Message0("is %f Pa\n",pres_sum/vol_sum);
```

- Note the exact similarity of the function “Message0” with Message and printf commands

Parallel File Output

- In a parallel session, file I/O can be done only through the **`Node_Zero`**

Example:

```
#if PARALLEL
if (I_AM_NODE_ZERO_P)
{ sprintf (ntim,"outfile-%d", ntime);
  if (fd == NULL)   /*Open a new file */
   {fd = fopen(ntim,"w");}
/* if new file "open" failed, try to append */
  if (fd == NULL) /* reopen the file in append-mode*/
   {fd = fopen(ntim,"a");
    Message( "Appending to existing file: %s", ntim);
    fprintf(fd,"\nAppend begins at: %f \n", f_time);}
}
#endif
```