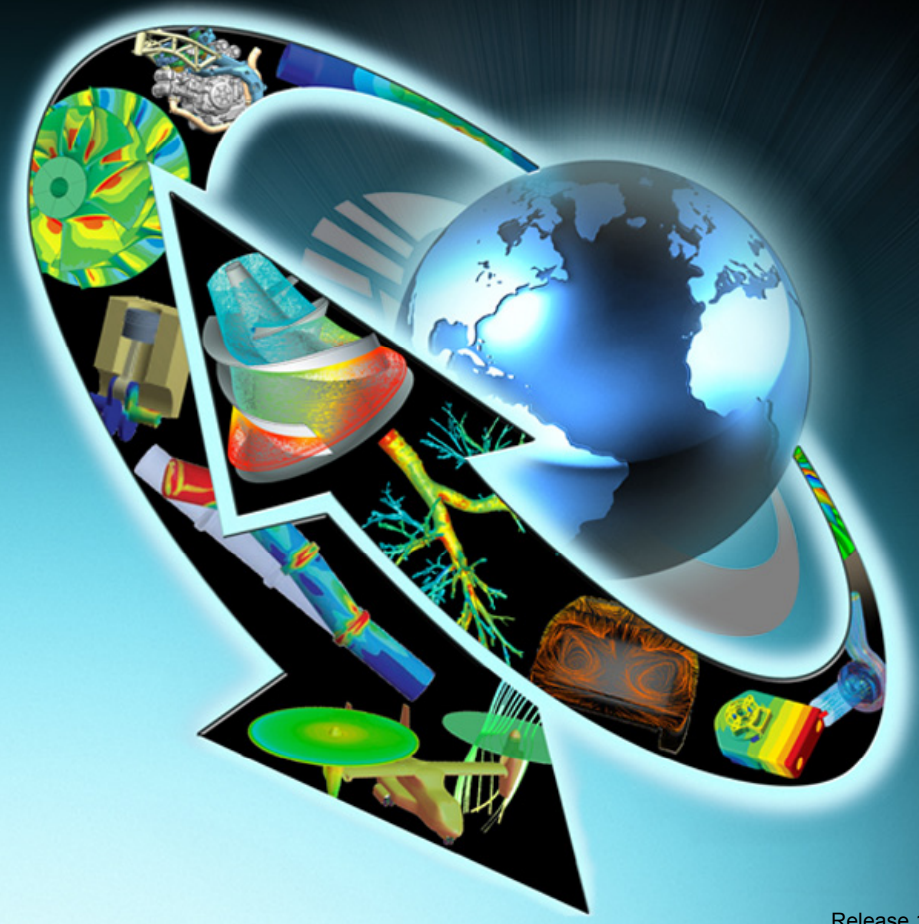
ANSYS®

Chapter 7

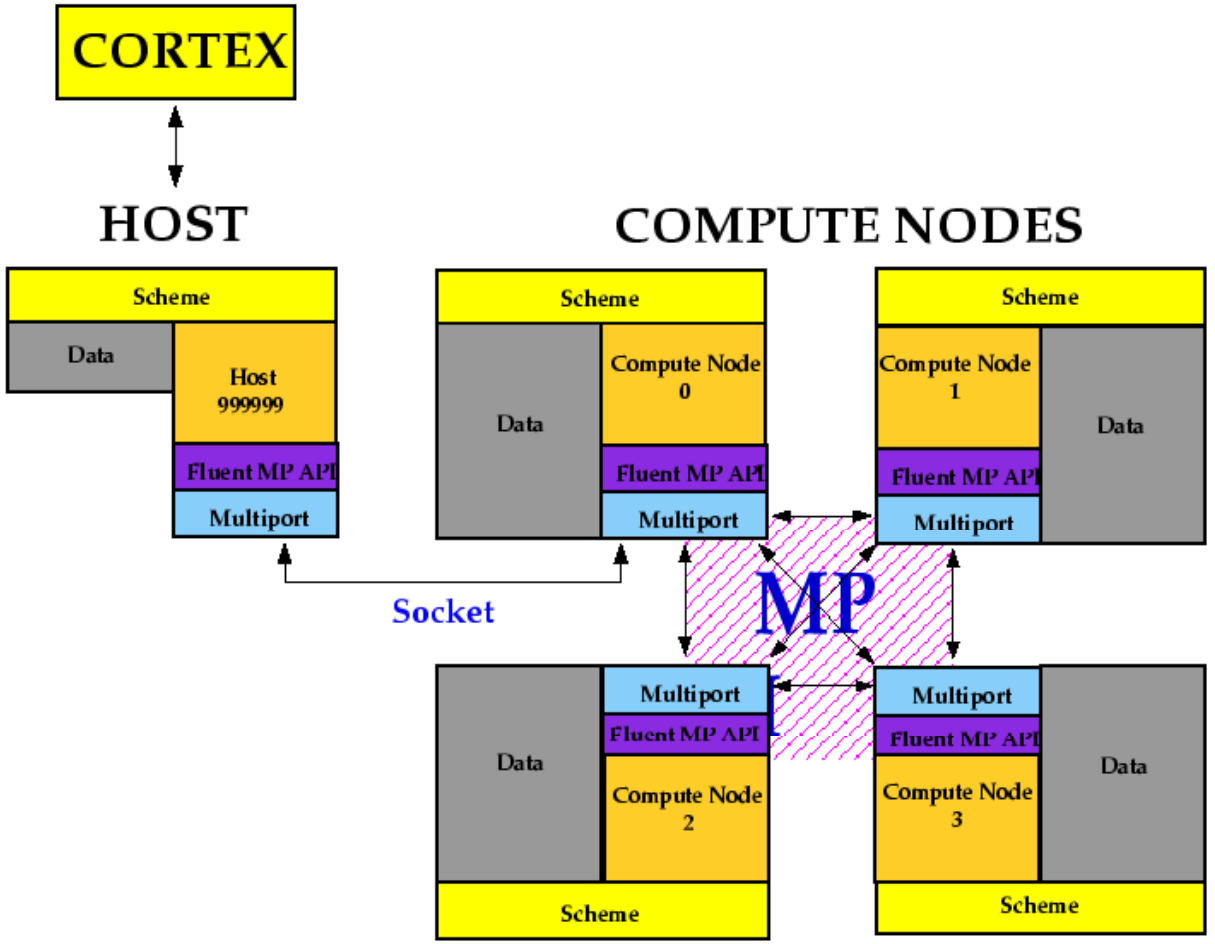
Parallelizing UDFs



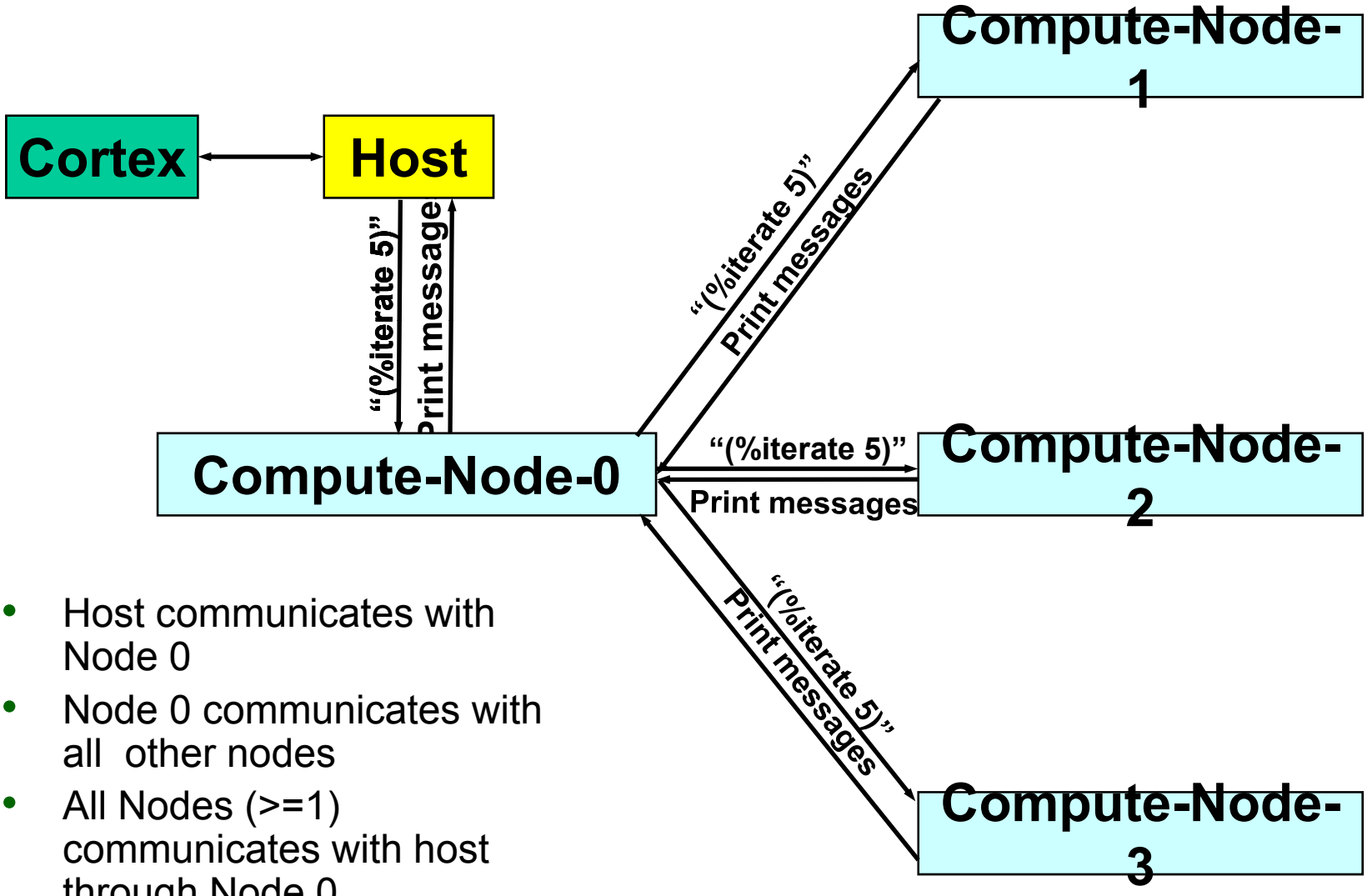
Advanced FLUENT User-Defined Functions

Parallel Fluent Architecture

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

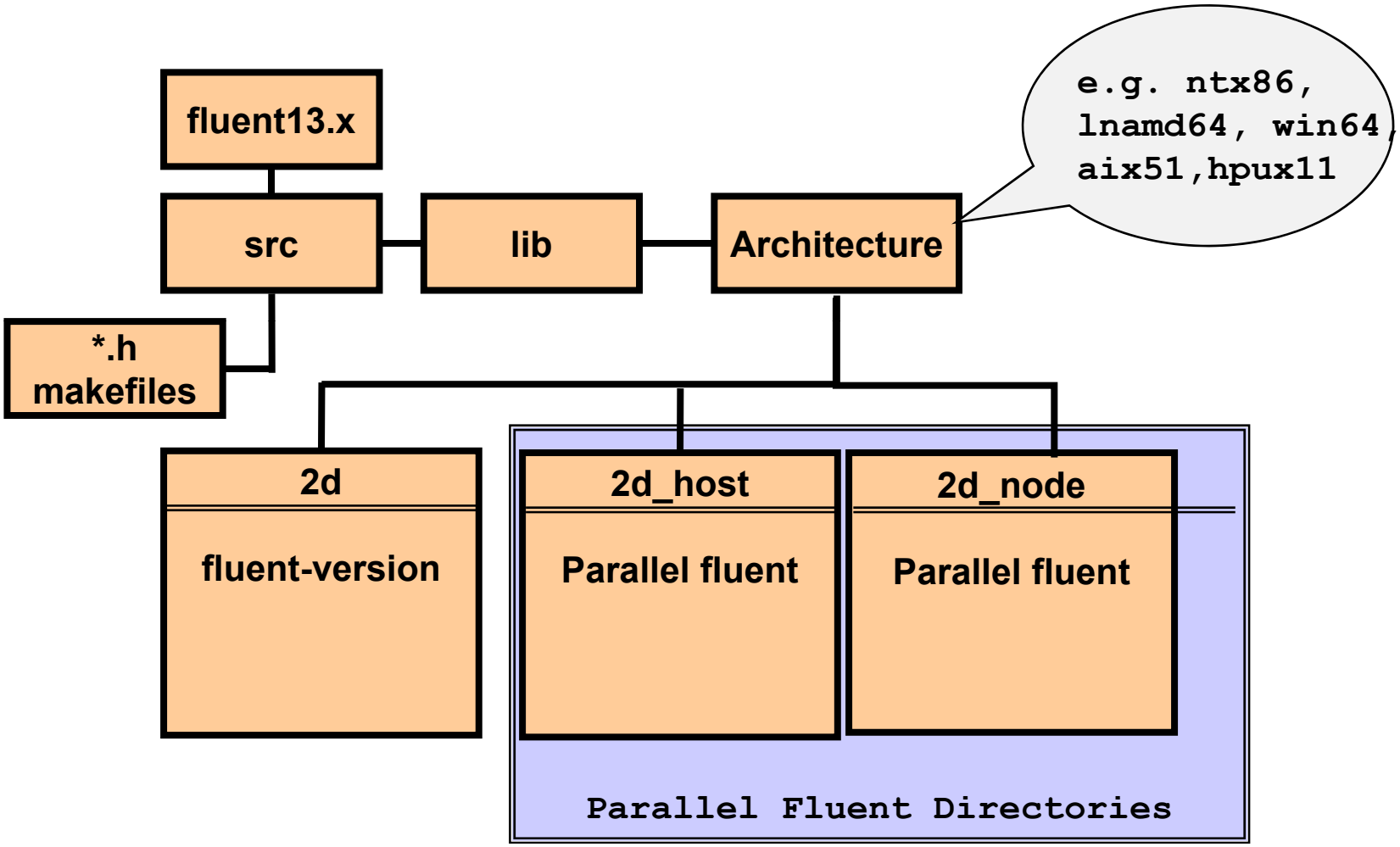


Commands Transfer in Parallel Fluent



- Host communicates with Node 0
- Node 0 communicates with all other nodes
- All Nodes (>=1) communicates with host through Node 0

fluent13.x Directory



Parallelizing a Serial UDF code

- In general, UDF codes need to be parallelized in order to make sure the codes are executed properly as either a host or node process
- To parallelize a UDF is to modify a serial code which will run successfully on both serial and parallel modes
- We use compiler directives (preprocessor commands) inserted into the original C code to distinguish which part of the program is executed by the host or by the node(s) when the UDF is compiled
- The syntax of the compiler directive is quite simple:

`#if` is a compiler directive (similar to "`#define`")

`#endif` is used to close an `#if`

Basic Compiler Directives

Examples:

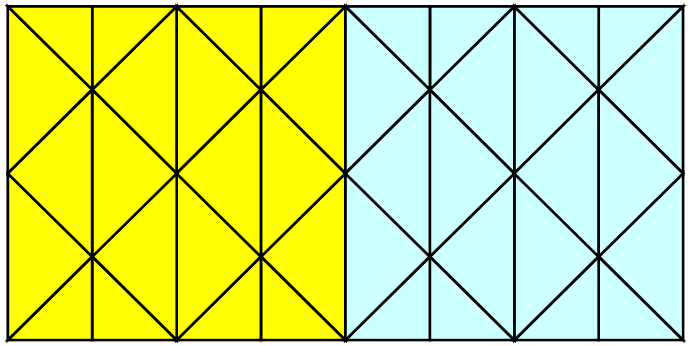
```
#if RP_HOST
 /* only host process is involved */
#endif
#if RP_NODE
/* only compute nodes are involved */
#endif

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

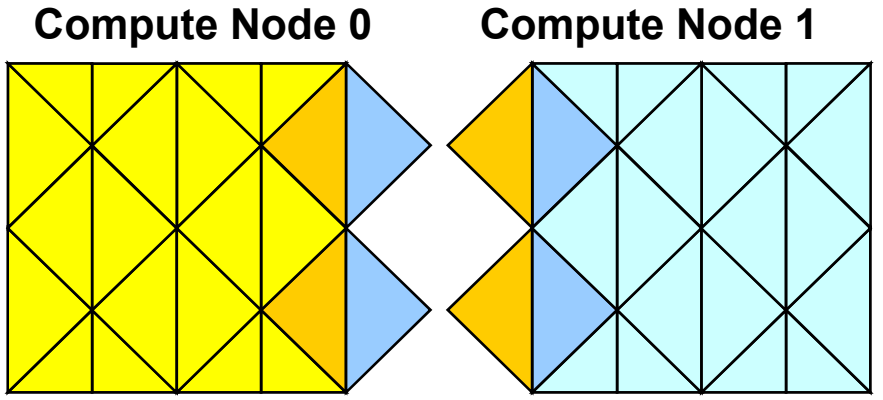
More directives can be found in the UDF Manual

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



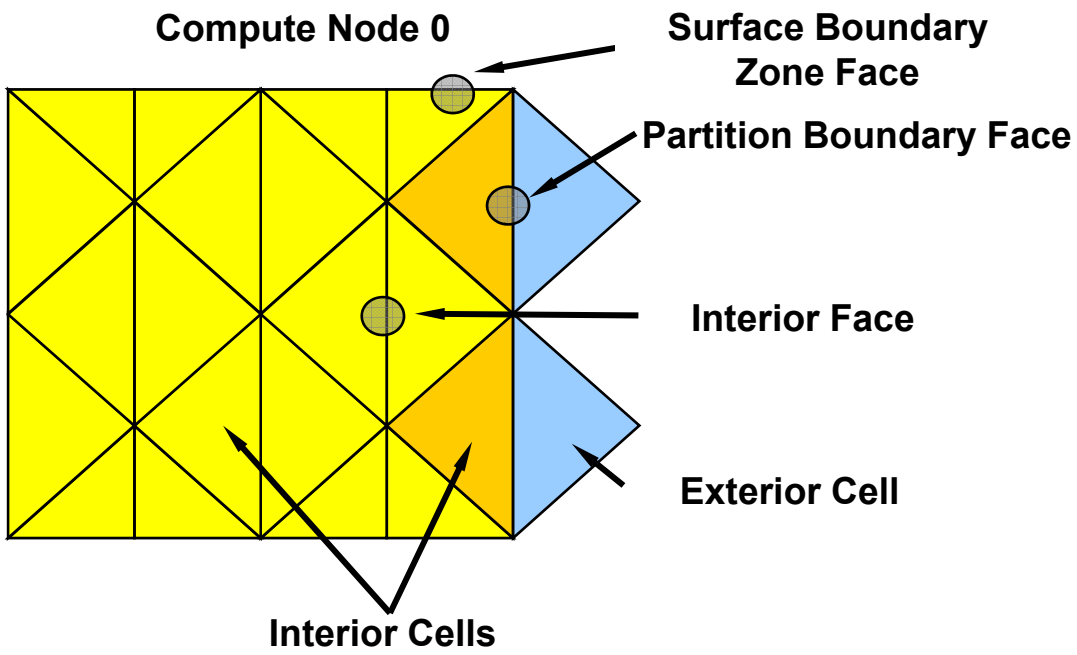
Domain Decomposition



Distribution across Compute Nodes

Partitioning (2)

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



Partitioned Thread Loop (1)

- The loop macro for cells:

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

- In parallel use, above loop construct loops through both interio and exterior cells
- 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)
```

but 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 face thread by using PRINCIPAL_FACE_P:

```
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)

- 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. (In parallel, it is executed on *both* the host and nodes.)

- The node to host data transfer macro is

  node_to_host_real_1(temp);

  Note that this command only sends the value of temp from Node0 to the Host; it does nothing for serial version
- Node to host data transfer occurs most frequently when passing the "global reduction" results from Node 0 to Host

Global Reduction

• Operations that collect data from all compute nodes and reduce the numbers into a single value or an array of values are called *Global Reduction* macros

Depending the objectives,

1) If you want the total value over all the Nodes, 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 ( )

Message0 ( ) function:

- **A function running on Node0 only. It prints directly to the cortex window. It is ignored on other compute nodes**
- **Also works for the serial processes**

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

- **Note the identical syntax of the function "Message0" with Message and printf commands**

Summary

- UDF Manual for additional information for parallel FLUENT and UDFs
- Many easy-to-read examples are available in the UDF Manual