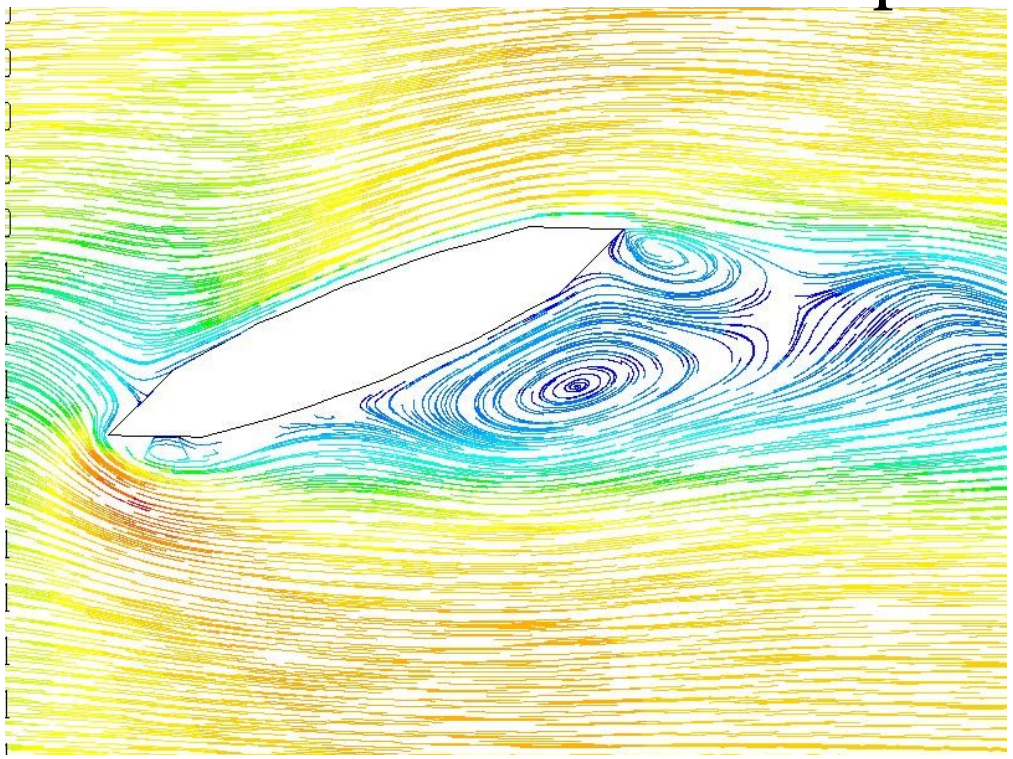
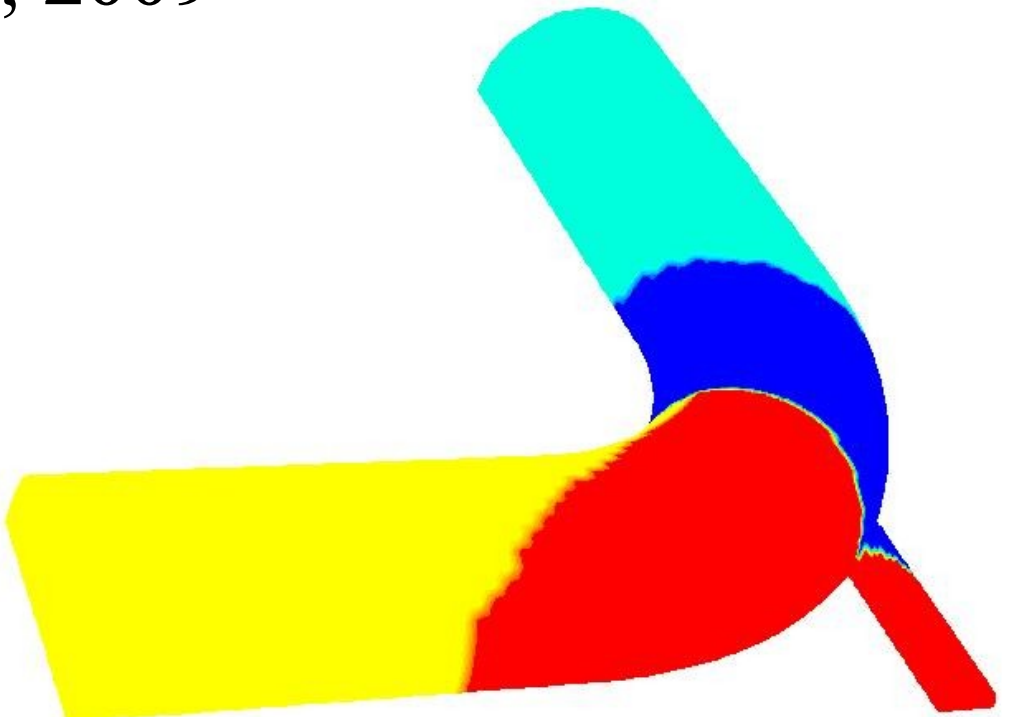
Special Topics In FLUENT

David H. Porter
Minnesota Supercomputing Institute
University of Minnesota

April 21, 2009





OUTLINE

Parallel FLUENT

- Faster time to solution
- Do larger problems

User Defined Functions

- Customize … almost anything

Hands-On

- Experiment & learn
- Try in your own application

Round Table Q&A

- Access experience of group
- Identify needs & problems
- Suggest future special topics

Running FLUENT in Parallel

- **Motivation**
- **New platform for FLUENT**
- **How to run in parallel**
  - Domain decomposition
  - Interactively
  - In background
  - In PBS queue

Why Run in Parallel?

Best way to do really large jobs

- Batch queues are designed for large parallel jobs
- Only supported way to access significant resources

Faster time to solution

- Parallel speedup is good
- Queue availability will dramatically improve soon

Enable larger problems

- More memory available
- Memory distributed across nodes
- Typically 2 GB per core
- Larger & faster disks available on core systems

New Platform for FLUENT: Blade

Best way to compute on Blade is in parallel batch queue

- Like lab PCs, login nodes are shared
- Login nodes are *NOT* intended for computation
- You can submit large jobs to queue & disconnect
- Individual jobs can run for as long as 2 days
- New procurement will work the same way

Dedicated compute nodes

- 8 GB per node
- 4 cores per node

Disk space

- 6 TB of fast scratch space
- Use project space for longer term storage

How To Run In Parallel

Parallel FLUENT based on "Domain Decomposition"

- Partition mesh: for distributed memory systems
- One MPI rank per partition

Interactively

- GUI driven parallel FLUENT
- Interactive PBS queue

In the background

- Command line driven FLUENT
- Run & input scripts

Batch jobs in PBS queue

- How to submit jobs
- PBS batch scripts
- Like running in the background

Partition Mesh

Setup problem as usual.

Module load fluent

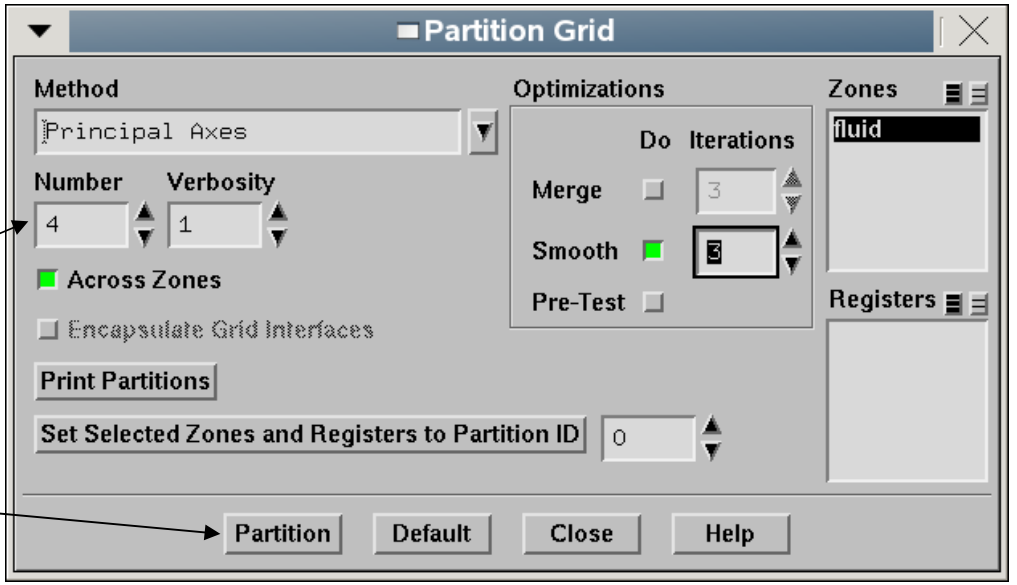
Fluent 3d

File → Read → Case & Data

Parallel → partition …

Set: Number=4

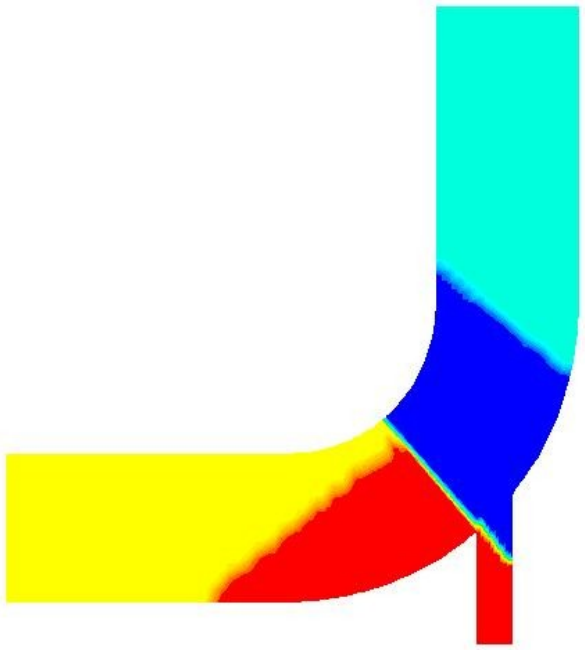
Partition

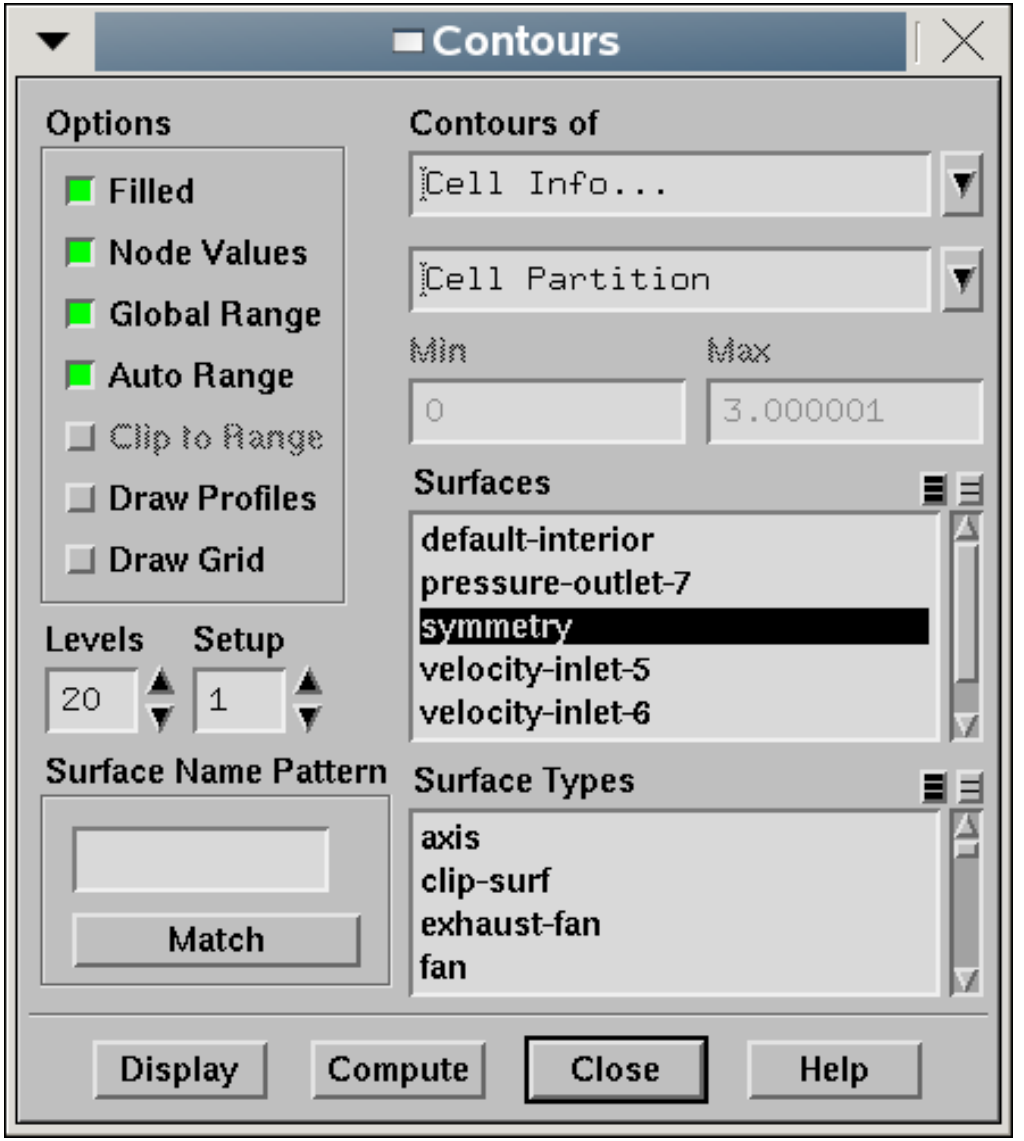


File → Write → Case & Data

Display Partitions

Display → Contours …
Filled
Contours of : Cell Info …
Cell Partion
Surfaces: Symmetry





Interactive Parallel FLUENT

Login with X forwarding

ssh -X <userid>@L6.msi.umn.edu

Start parallel fluent

- One MPI rank per partition
- Flag is -tN
- N = Number of MPI ranks
- All ranks on same node (limit to 4 on Blade)
- Can start from a pre-partitioned case

Fluent 3d -t4

File → Read → Case & Data

Solve → Iterate ...

Interactive in PBS QUEUE

FLUENT GUI requires X forwarding

- Flag for X forwarding –X
- Flag for interactive queue –I

Login with X forwarding, then submit interactive job with X

- ssh -X <userid>@blade.msi.umn.edu
- qsub –X –I r4i.pbs

```
#!/bin/bash -l
#PBS -l walltime=01:00:00,pmem=1750mb,nodes=1:ppn=4
```

Then wait for interactive prompt.

Command Line Parallel FLUENT

Setup Case as usual

Partition mesh & save Case+Data

- Includes mesh partition

Edit a text input file

- File: run4.in
- A leading ";" means comment

```
; Read case & data files
rcd elbow1.cas.gz
; Calculate 150 iterations
it 150
; Write case & data files
wcd elbow_150.cas.gz
; Exit FLUENT
exit
```

Run without GUI

- Flag -g means no GUI
- Flag -tN mean run N partitions (ranks)

fluent 3d -t4 -g < run4.in >& run4.out &

Run in PBS Queue

Setup for command line parallel run

- Generate Case+Data
- Partition mesh
- Edit input script

Setup PBS run

- Case & data in directory: <wdir>
- Edit PBS script: run.pbs
- qsub run.pbs

```
#!/bin/bash -l
#PBS -l walltime=08:00:00,pmem=1750mb,nodes=1:ppn=4
#
cd <wdir>
fluent 3d -t4 -g < run4.in >& run4.out
```

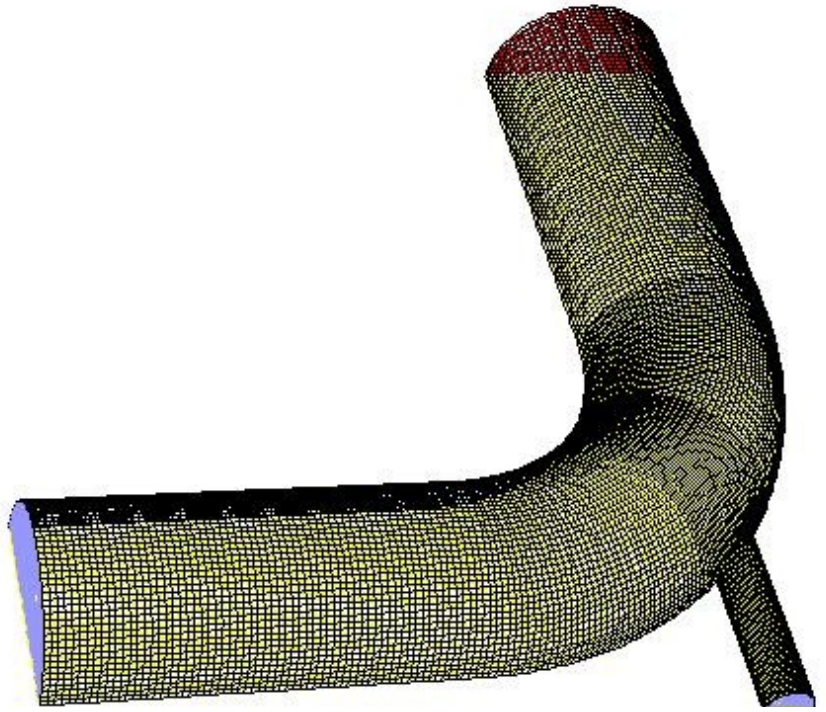
NO &

Parallel Speedup Test

Test case: 3D Mixing Elbow (from Tutorial 1)

Refined mesh once: Cells=110816; Faces=344224; Nodes=122935

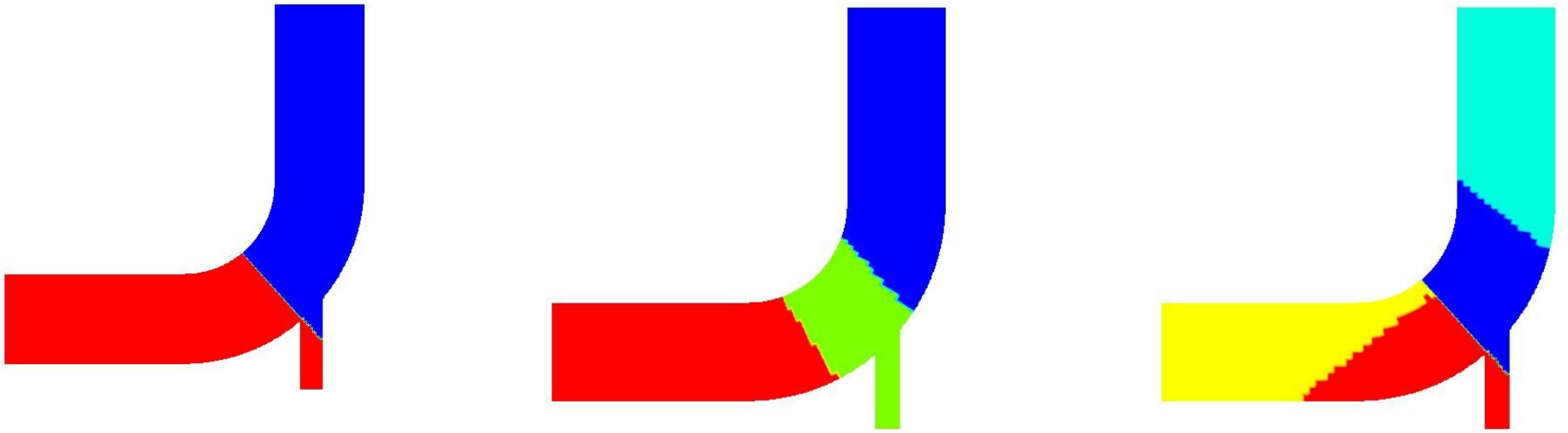
Test on Blade using 1, 2, 3, & 4 partitions



Parallel Speedup Results

Ran relaxation iterations for steady state from initial state.
Times for 20 & 120 iterations include startup, read, write, & shutdown
Internal 100 iterations representative of long calculation limit.
All Times are in sec.

N	20 iter	120 iter	Internal 100	Speedup
1	47.041	244.462	197.421	1.00
2	36.620	179.174	142.554	1.38
3	28.102	115.651	87.549	2.25
4	24.899	93.501	68.602	2.87



User Defined Functions

What is a UDF?

- What UDFs can do
- Kinds of UDFs
- Map of “Call Outs”

Macros

- **Define Macros**
- **Loop Macros**
- **Variable Macros**

Steps for creating & using

Examples

What is a User Defined Function?

A UDF is a routine that is called from FLUENT

- You write the routine
- Routine is in C
- Macros provide easy & uniform access FLUENT variables
- UDFs are compiled or interpreted then linked into FLUENT
- UDFs are “hooked” into appropriate parts of FLUENT

Use UDFs to customize many different aspects of FLUENT

- Boundary conditions & source terms
- What, when, and how data is read and written
- Models: fluid, martial, turbulence, species, …
- Control time step
- Post-processing

Kinds of User Defined Functions

Kind of UDF determines when it is called & what it can do

- Kind of UDF specified by "Define Macro"
- Define Macro specifies when and how UDF is called

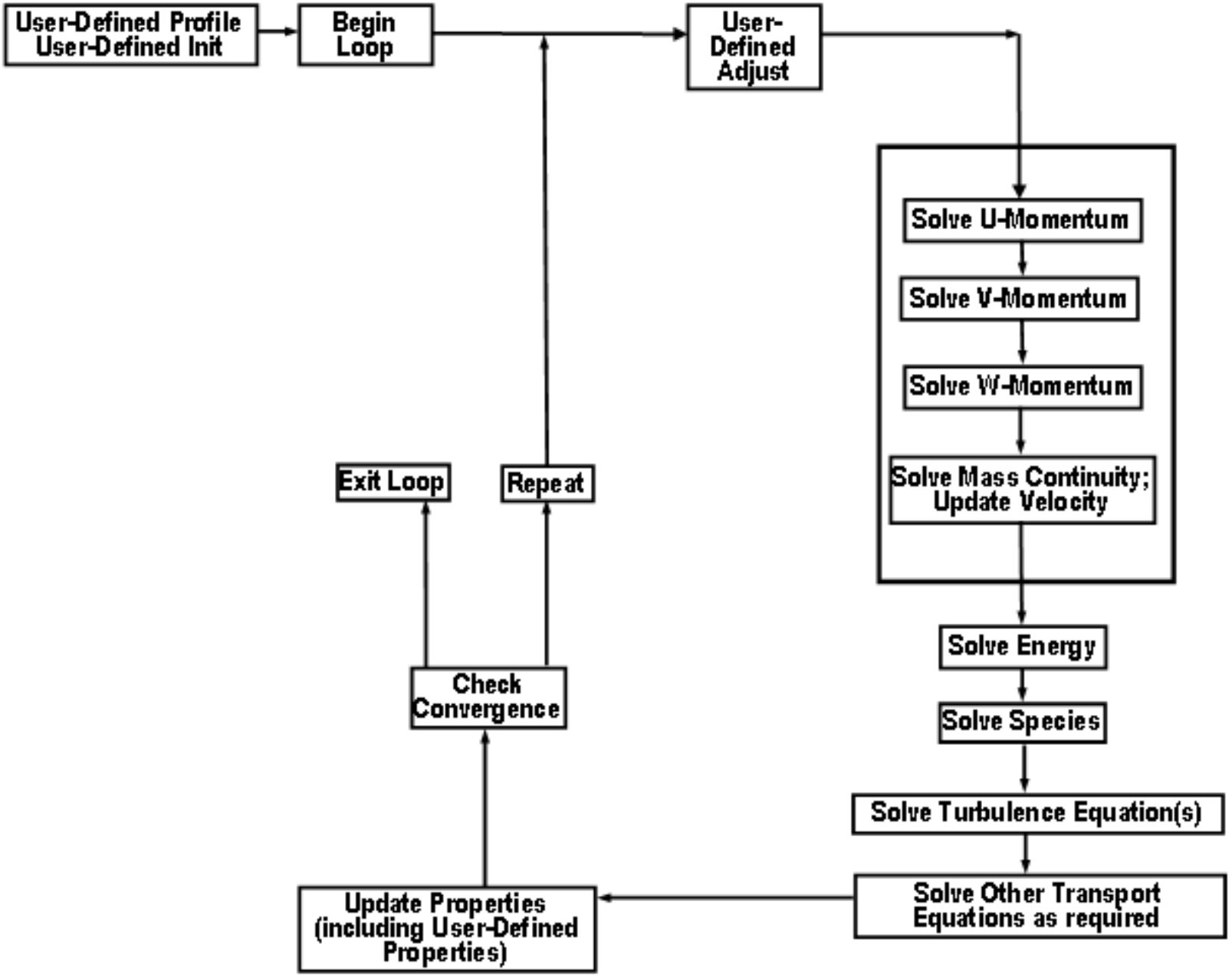
General Purpose UDFs (called outside of fluid update)

- On startup & on initialization
- On Read & Write
- Before or after each fluid iteration
- On exit
- On demand

Model specific UDFs (usually called during fluid iteration)

- Material properties
- Transport equations
- Turbulence models
- Dynamic mesh
- Mesh motion

Calling Procedure (Pressure Based)



General Purpose Define Macros

DEFINE_ADJUST	Manipulate variables before iteration
DEFINE_DELTAT	Control time step
DEFINE_EXECUTE_AT_END	At end of fluid update
DEFINE_EXECUTE_AT_EXIT	At end of FLUENT session
DEFINE_EXECUTE_FROM_GUI	User defined scheme routine
DEFINE_EXECUTE_ON_LOADING	When a UDF library is loaded
DEFINE_INIT	Initialize variables
DEFINE_ON_DEMAND	Asynchronously: on user input
DEFINE_RW_FILE	When Case & Data are read or written

For details, see UDF Manual, Sec 2.1

Model Specific Define Macros

Many different kinds of model specific define macros
Here are some categories

DEFINE_WALL_FUNCTIONS
DEFINE_PROFILE
DEFINE_SOURCE
DEFINE_DIFFUSIVITY
DEFINE_TURBULENT_VISCOSITY
DEFINE_TURB_PREMIX_SOURCE
DEFINE_HEAT_FLUX
DEFINE_PRANDTL UDFs
DEFINE_PROPERTY UDFs
DEFINE_SR_RATE
DEFINE_VR_RATE

DEFINE_CHEM_STEP
DEFINE_CPHI
DEFINE_DOM_DIFFUSE_REFLECTIVITY
DEFINE_DOM_SOURCE
DEFINE_DOM_SPECULAR_REFLECTIVITY
DEFINE_GRAY_BAND_ABS_COEFF
DEFINE_NET_REACTION_RATE
DEFINE_NOX_RATE
DEFINE_PR_RATE
DEFINE_SCAT_PHASE_FUNC
DEFINE_SOLAR_INTENSITY
DEFINE_SOX_RATE

For details & full lists of define macros See UDF Manual, Sec. 2

Example: Define Macro

```
#include "udf.h"

DEFINE_PROFILE(name_for_this_routine, thread, position)
{
  /* Local variable declarations */
  /* & content of routine */
}
```

- #include “udf.h” always needed
- Define_... specifies where UDF can be hooked and arguments
- You provide UDF name for reference in FLUENT
- Arguments (thread & position) passed into UDF by FLUENT

Thread is a set of cell faces passed by FLUENT

Position is which is set by fluent

Macros For Looping Over Mesh

Mesh Hierarchy

- Domain(s) made up of threads
- Threads made up of cells or faces
- Cells contain volume (3D) or surface (2d) variables
- Faces contain variables on 1D lower structures
- Nodes: points which define faces & cells

Loop constructs

- Loop over cell-threads or face-threads in a domain
- Loop over cells in a cell-thread
- Loop over faces in a face-thread
- Loop over nodes in cell or face

Example: Loop Over Cells

```
DEFINE_ADJUST(my_routine, domain)
{
  Thread *t;
  cell_t c;

  thread_loop_c (t,domain) {
    begin_c_loop (c,t) {
      /* Access or modify cell “c” variables */
    }
  }
}
```

- In a “DEFINE_ADJUST” UDF, the domain is passed in
- All threads “t” in the domain “domain” are looped over in outer loop
- All cells “c” in the thread “t” are looped over in the inner loop
- This routine will do what you specify to every cell in the domain

Macros For Data Access

Access data at

- Nodes: locations, numbers
- Cells: volume, flow variables, gradients
- Faces: vector area, flow variables, fluxs
- Connectivity: vectors between cell centroids
- Special utilities: get ID, set boundary profile
- Model specific:
- User Defined Scaler: set transport coefficients
- User Defined Memory:

Will only review some of the data access macros here.

For full lists & details see: UDF Manual, Sec 3.2

Macros For Data Access: Nodes

Node Macros

- NODE_X(node) = X-coordinate of node
- NODE_Y(node) = Y-coordinate of node
- NODE_Z(node) = Z-coordinate of node
- F_NNODES(f,t) = number of nodes in a face

For details see: UDF Manual, Sec 3.2.2

Macros For Data Access: Cells

Cell attributes (of cell_t c in thread *t)

- C_CENTROID(x,c,t): returns centroid in real x[ND_ND]
- C_VOLUME(c,t) = volume of cell
- C_NNODES(c,t) = number of nodes
- C_NFACES(c,t) = number of faces
- C_FACE(c,t,i) = global face index ,from local face index i

Cell flow variables: C_var(c,t) = flow variable “var”

Where: var={R,P,U,V,W,T,H}

For: density,pressure,velocity(XYZ),temperature,enthalpy

Gradients: C_<var>_G(c,t)

Reconstruction Gradients (monotonic): C_<var>_RG(c,t)

For complete lists and details see: UDF Manual, Sec 3.2.3

Macros For Data Access: Faces

Face attributes (of face_t f in thread *t)

- F_CENTROID(x,f,t): returns centroid in real x[ND_ND]
- F_AREA(a,c,t): returns face area vector in real a[ND_ND]

Boundary face flow variables: F_var(f,t) = flow variable "var"

Where: var={U,V,W,T,H}

For: XYZ-velocity,temperature,enthalpy

Interior and boundary faces

- F_P(f,t) = pressure
- F_FLUX(f,t) = mass flow rate through face

For details see: UDF Manual, Sec 3.2.4

User Defined Scalars

New mesh variables

- Cell variable: one for each cell
- Face variable: one for each face

Inputs for FLUENT procedures or use for your own purposes

- Set & used in UDFs
- Can be hooked into FLUENT procedures
- Custom diagnostics

Steps To Create & Use A UDF

Text edit a C source code file which defines the UDF

- Use a DEFINE macro for routine declaration
- Use loop macros to run over cells & faces
- Use data access macros to retrieve and modify FLUENT data

Compile or interpret source code & load into FLUENT project

- Build build library through FLUENT GUI
- Load library into project

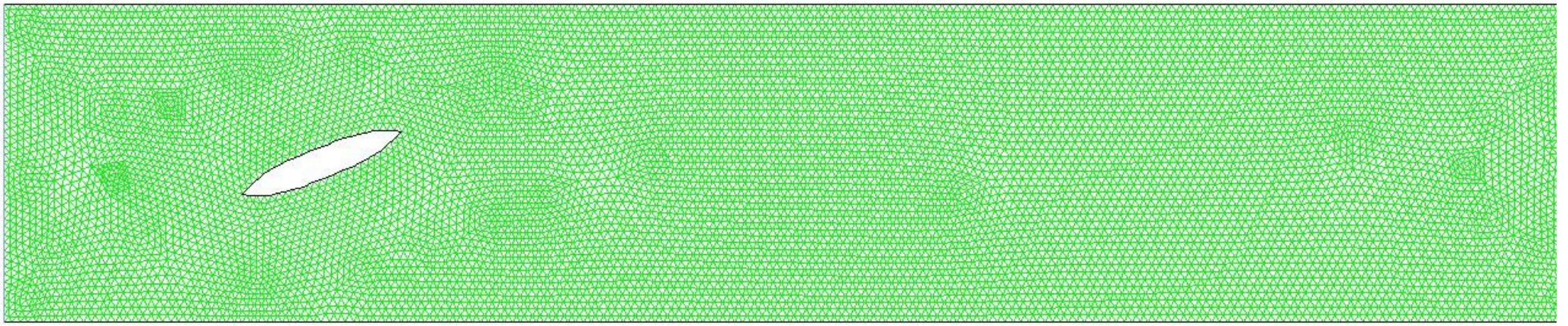
Hook UDF to FLUENT model

- Your UDF name will appear in appropriate dialogs
- Increase number of UDS as needed.

UDF Examples

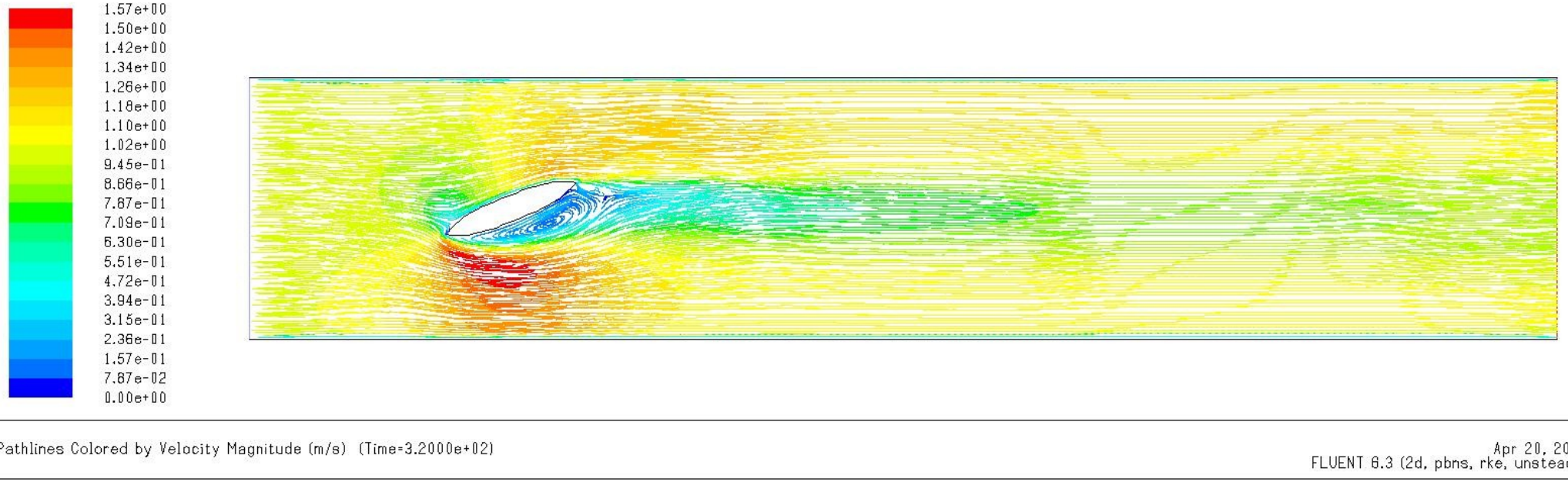
- **Two examples:**
  - Customize Inflow Boundary Conditions
  - Create a User Defined Scalar (UDS) for Post Processing

Start from: 2D channel flow past a blade



2D Channel Flow Past A Blade

Solver: 2D, unsteady, pressure based
Turbulence Model k-epsilon/realizable
Inlet boundary: constant Vx=1m/s
Mesh: Tri, 18,724 cells; 28,200 faces; 9576 nodes



Example 1: Modify Inlet Boudnary

Impose both temporally and spatially dependent inflow velocity

- Vx = 1.0 + 0.7*sin(t/2)*sin(pi*y/10)
- Units: MKS

Variation in inflow velocity

- Substantial: 70%
- 1 period across vertical range: y=[-10, 10]
- Period of 4*pi sec

Source Code For Boundary UDF

```
#include "udf.h"
DEFINE_PROFILE(sin_of_yt, thread, position)
{
  real x[ND_ND];           /* Position vector */
  real y;
  face_t f;
  real t = CURRENT_TIME;   /* Current time of simulation */

  begin_f_loop(f, thread)  /* Loops over faces in thread */
    {
      F_CENTROID(x,f,thread); /* Gets centroid of face f */
      y = x[1];
      F_PROFILE(f, thread, position) = 1.0 +
                        0.7*sin(0.5*t)*sin(0.3141592654*y);
    }
  end_f_loop(f, thread)

}
```

F_PROFILE: macro specifically designed for setting profiles
Position: Index of variable (like Vx) to be set (will be selected in GUI)
Thread: ID of boundary (will be selected in GUI)

Compile and Load UDF

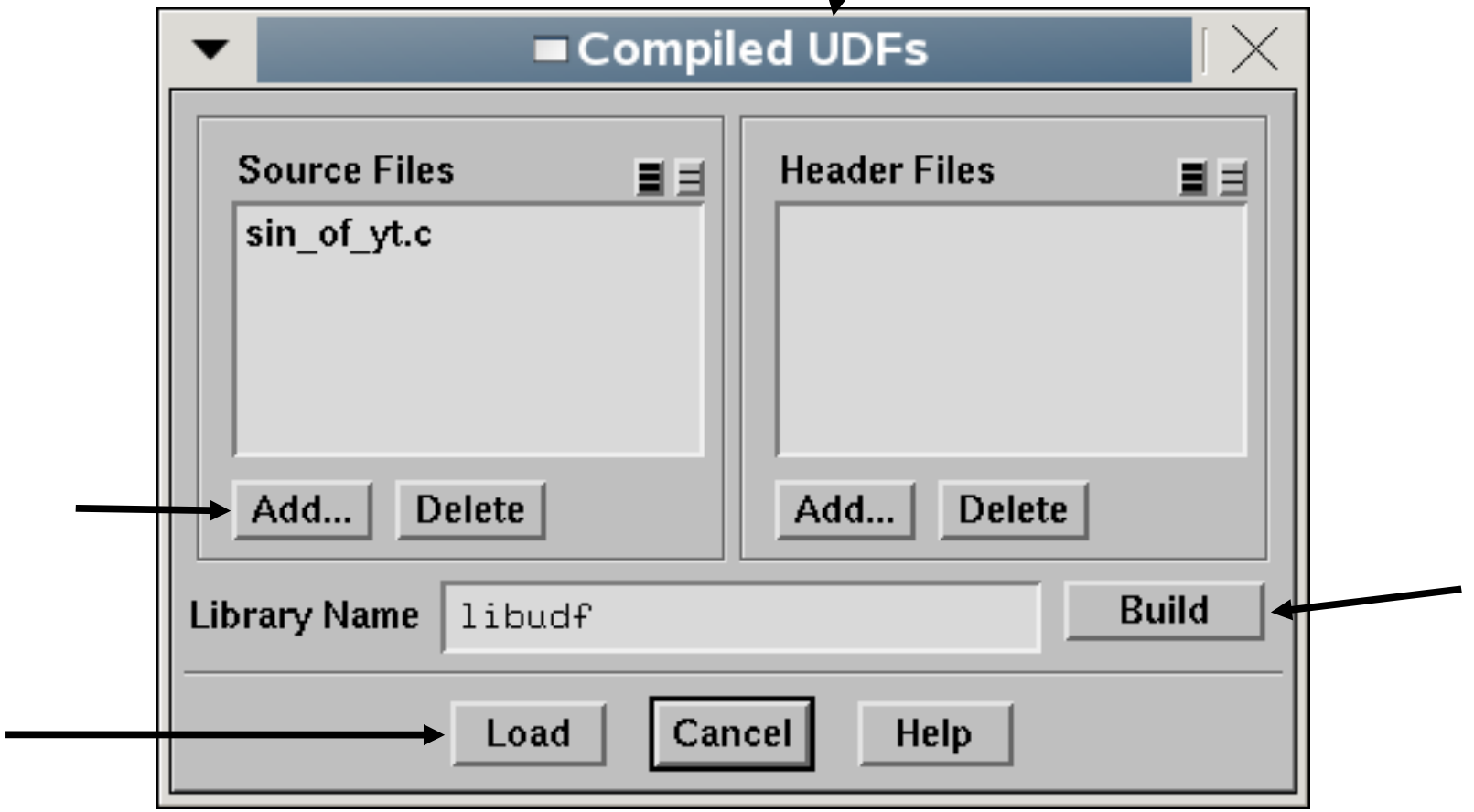
Define → User Defined → Functions → Compiled

Source File Add... "sin_of_yt.c"

Keep Library Name: "libudf"

Build

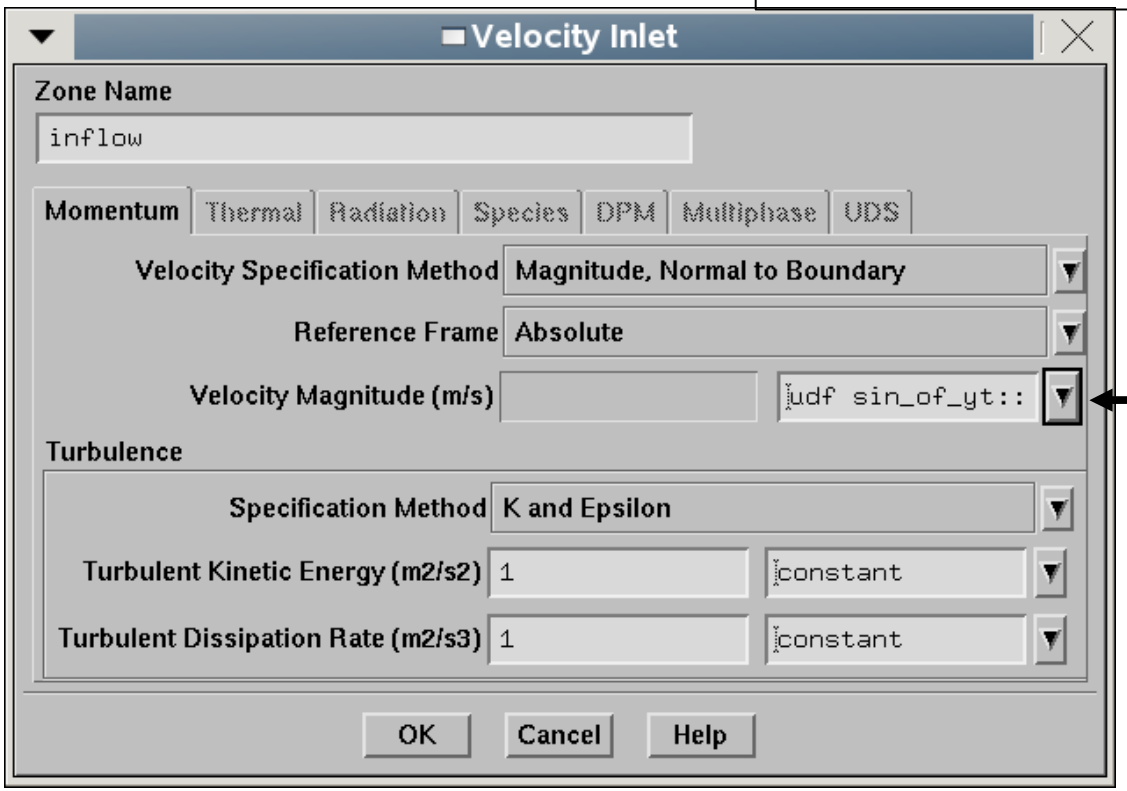
Load



Hook UDF Into Inlet Boundary

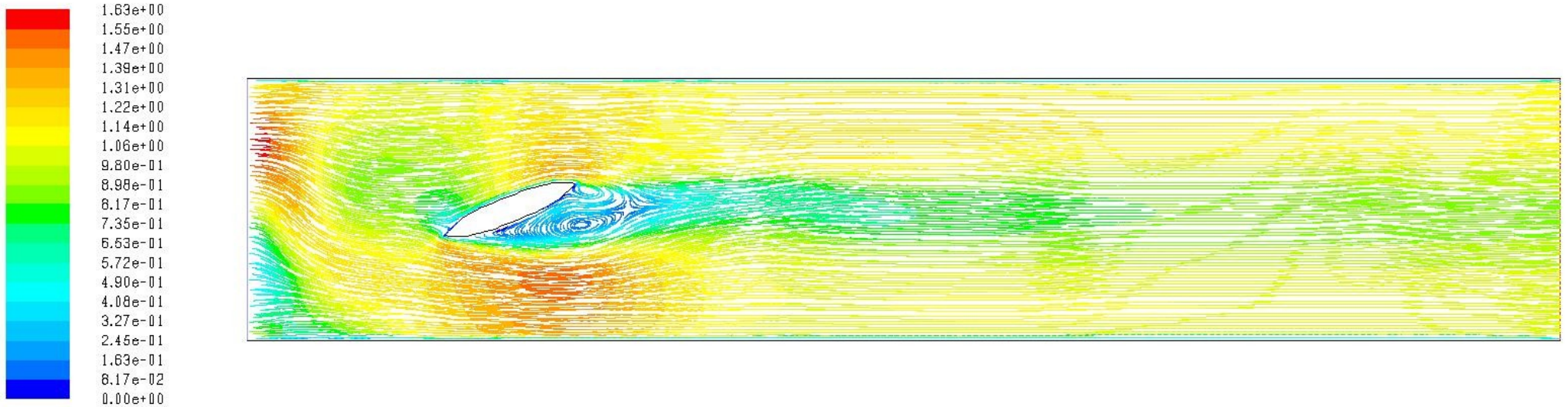
Define → Boundary Conditions
    Select Inflow (Velocity Inlet)
    Set …

Velocity Inlet (Inflow)
    Momentum Tab
    Velocity Magnitude (m/s)
        Select: udf sin_of_yt
    OK

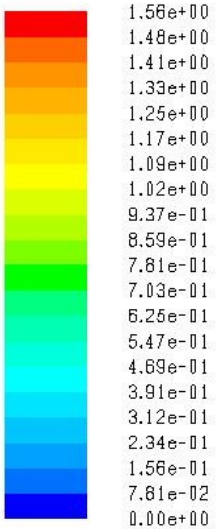


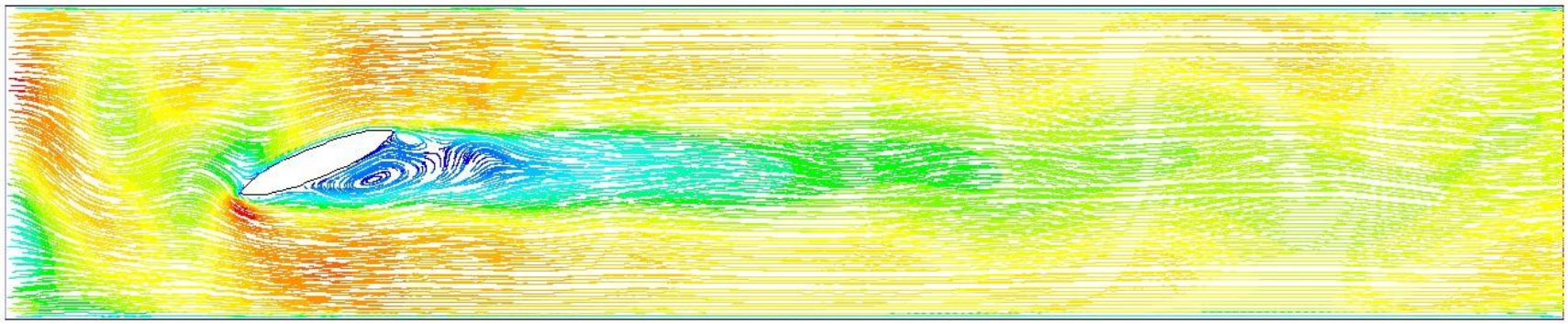
Run For 9 Sec

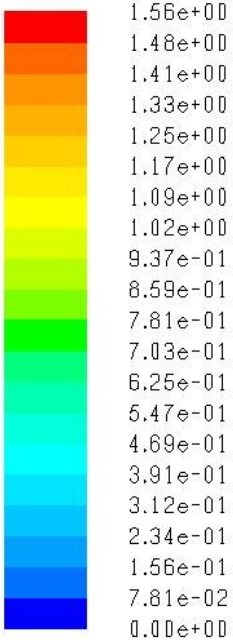
About 3/4ths of a period

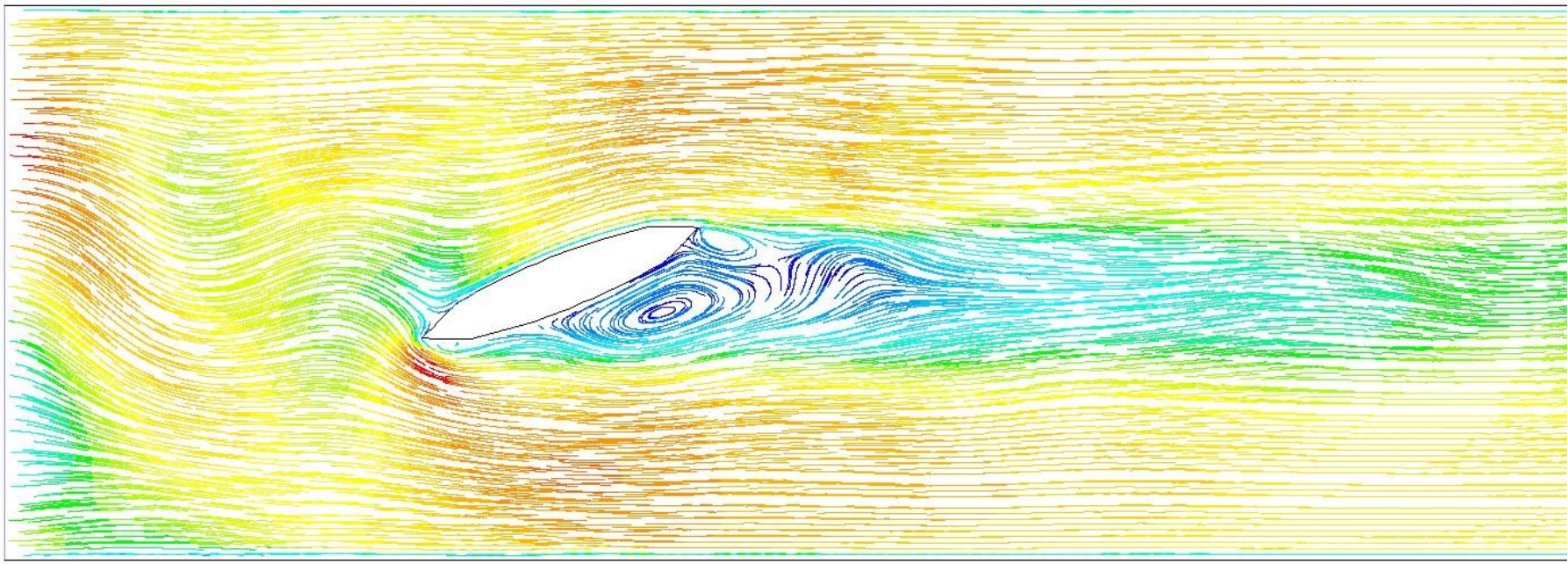


Run For Another 100 Sec.









Example 2: Create New Diagnostic

Goal: visualize Z-component of vorticity
    Shows eddies and slip surfaces
Signed quantity: direction of rotation

Formula: dV/dx - dU/dy

C Source Code: ON_DEMAND

```
#include "udf.h"
enum {
  VORT_Z,           /* Index of UDS: VORT_Z=0 */
  N_REQUIRED_UDS    /* # of UDS variables: N_REQUIRED_UDS=1 */
};

DEFINE_ON_DEMAND(vortz_on_demand)
{
  Domain d;         /* domain is not passed in */
  Thread t;
  cell_t c;
  face_t f;

  /* Make sure there are enough user-defined scalars. */
  if (n_uds < N_REQUIRED_UDS) {
    Internal_Error("not enough UDSs allocated");
  }

  d = Get_Domain(1);    /* Get domain using Fluent utility */
```

An ON_DEMAND UDF can be called at any time from GUI

C Source Code: C_DVDX

```
/* Fill  cells with UDS: Z-component of vorticity. */
thread_loop_c (t,d)
  {
    if (NULL != THREAD_STORAGE(t,SV_UDS_I(VORT_Z)))
      {
        begin_c_loop (c,t)
          {
            C_UDSI(c,t,VORT_Z) = C_DVDX(c,t) - C_DUDY(c,t);
          }
        end_c_loop (c,t)
      }
  }
```

Loop over all cell threads “t” in domain “d”
Make sure storage for the UDS is available for thread t
Loop over cells c in thread t
Use macros C_DVDX & C_DUDY to evaluate velocity derivatives.

C Source Code:

```
  /* Fill  faces with UDS: Z-component of vorticity. */
  thread_loop_f (t,d)
    {
      if (NULL != THREAD_STORAGE(t,SV_UDS_I(VORT_Z)) &&
          NULL != T_STORAGE_R_NV(t->t0,SV_UDSI_G(VORT_Z)))
       {
          begin_f_loop (f,t)
           {
            F_UDSI(f,t,VORT_Z) =
                                C_UDSI(F_C0(f,t),t->t0,VORT_Z);
           }
          end_f_loop (f,t)
       }
    }
}
```

Loop over face threads in d, and faces in those threads.
Set face value of UDS to the corresponding cell value.

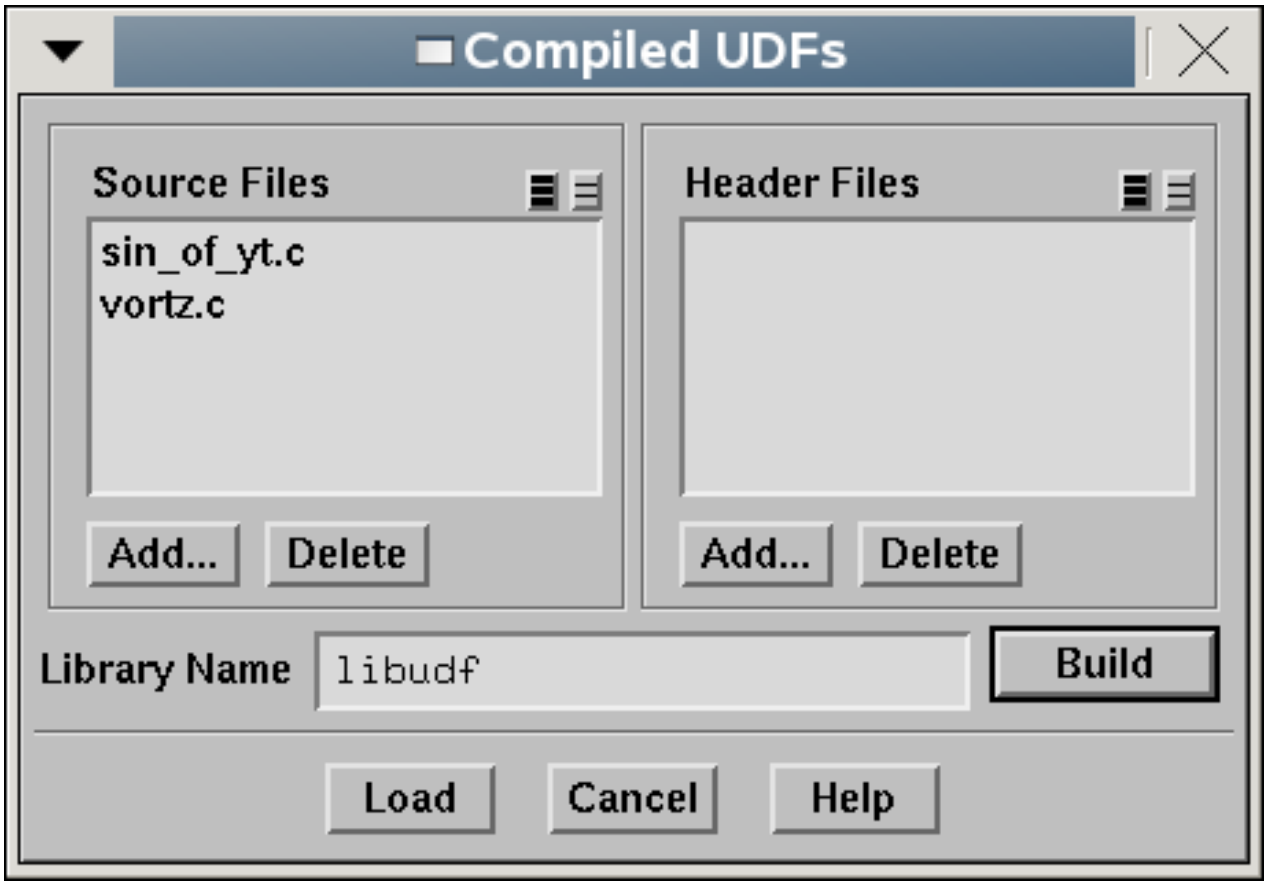
Compile & Load UDFs

Define → User-Defined → Functions → Compiled ...

Source Files: add both sin_of_yt.c & vortz.c

Build

Load

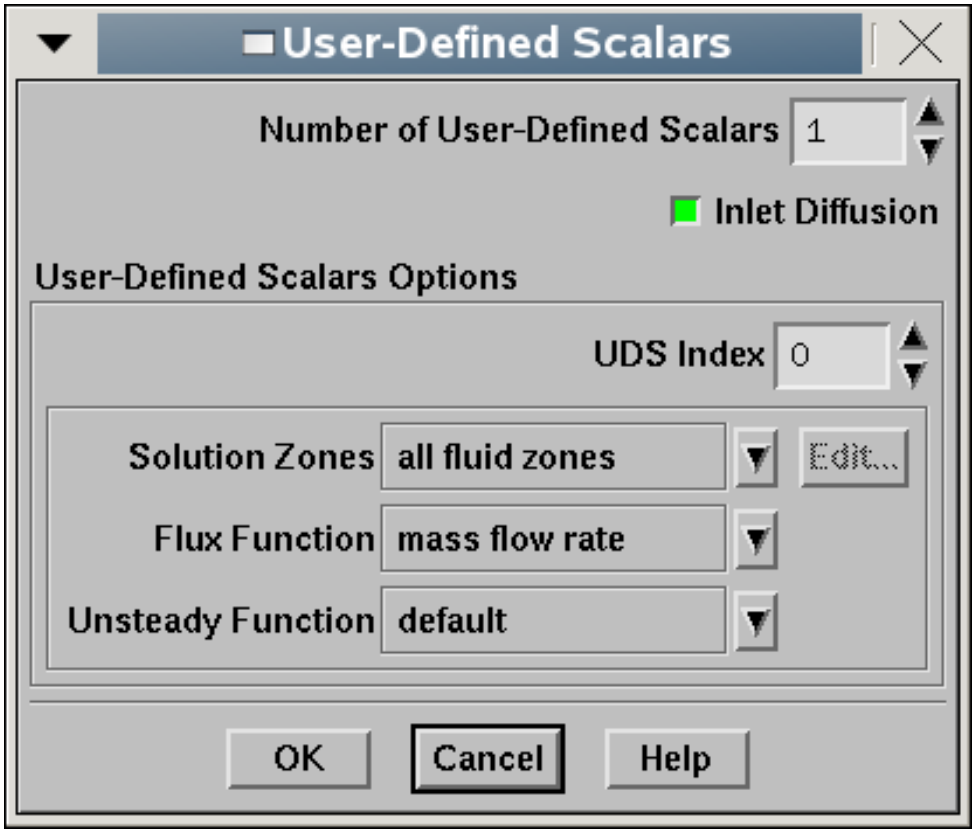


Increase Number of UDS

Define → User-Defined → Scalars …

Set: Number of User-Define Scalers = 1

OK

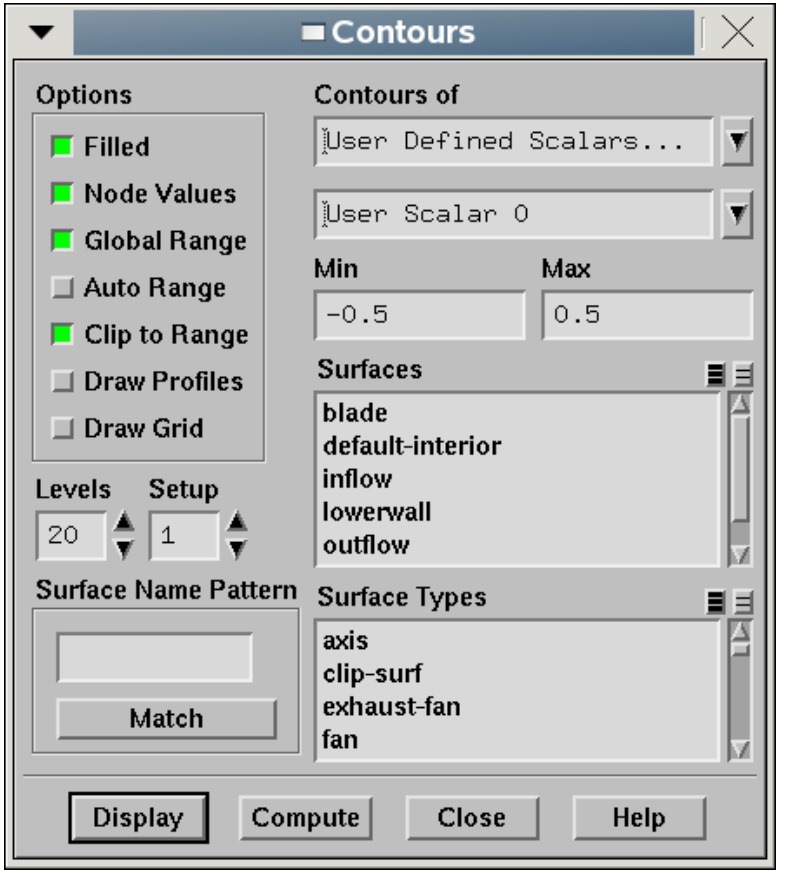


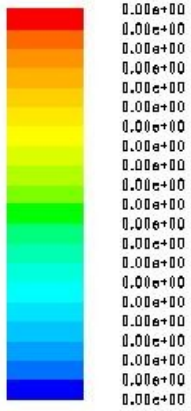
View Before Running UDF

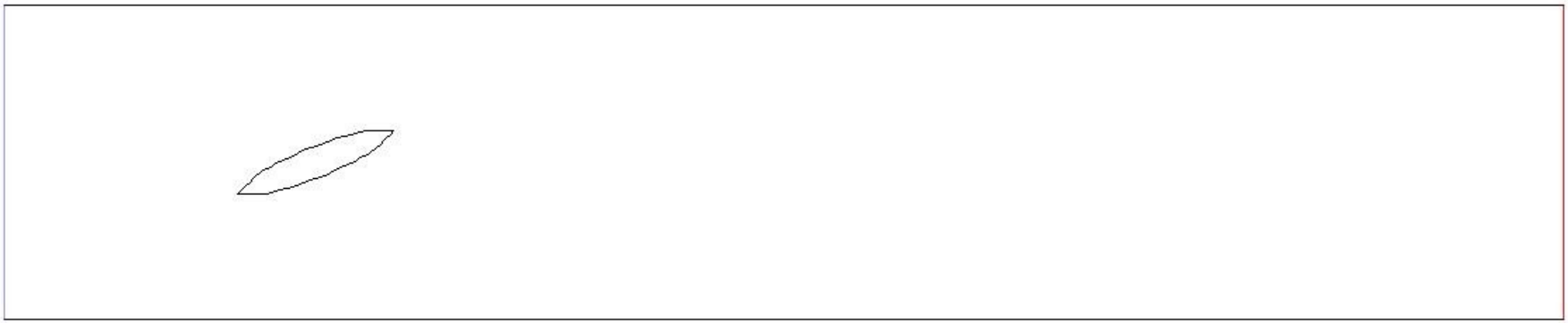
Can Select UDS in Contours

However, UDS has not been filled.

Need to run ON_DEMAND UDF.



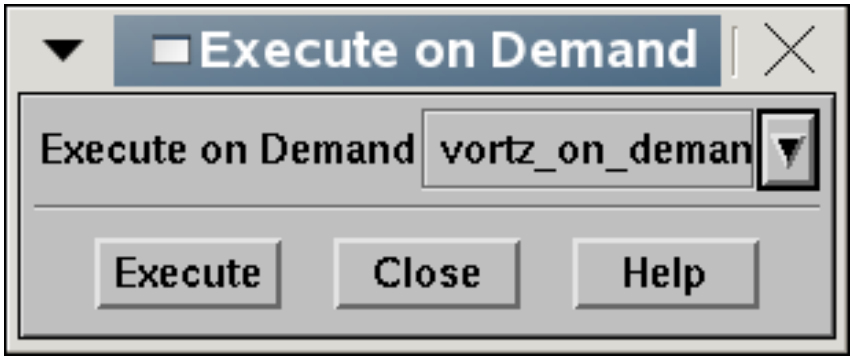




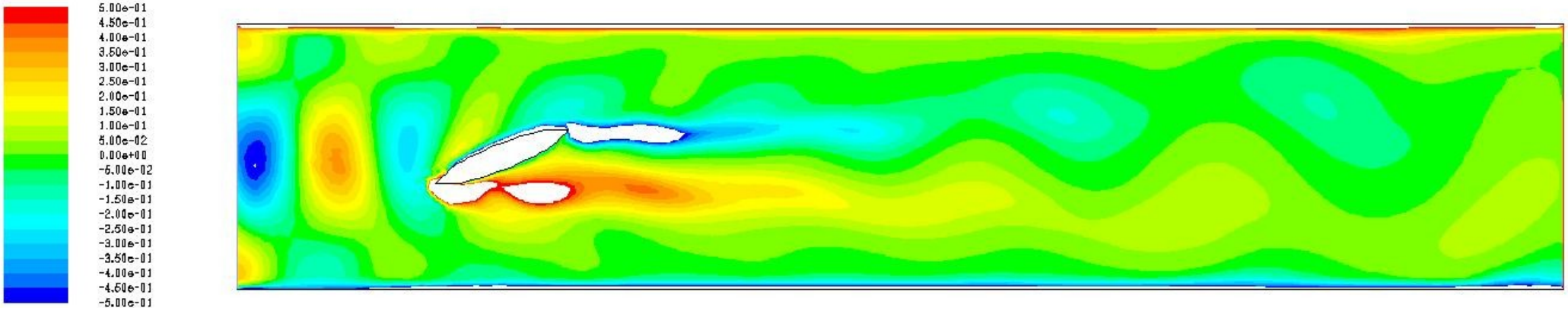
View After Running UDF

Iterate once to fill arrays.

Define → User-Defined
→ Execute on Demand …



Vort_z shows direction of rotation
Set range to [-0.5, 0.5]
White shows clipped values
Strong vorticity behind blade edges



Hands On

Tutorial logins on SDVL Linux PCs & Blade

Account names: temp01 – temp24

Template Case+Data in fluent directory

cd fluent
module load fluent
fluent 2d

Suggested Exercises

1) Create your own custom inlet and/or outflow boundaries
2) Run in parallel

To Get Help

On line Documentation: http://wwwr.msi.umn.edu/fluent/index.htm

Parallel Fluent: User's Guide Sec 31: Parallel Processing

User-Defined Functions: UDF Manual

MSI User Support:

help@msi.umn.edu

612-626-0802

Proposed: Computational Fluid Dynamics Focus Group

Network with people who share your interests

Access the large knowledge base here at the U. of M.

Identify common needs & problems

Propose initiatives

If your interested, contact me: porter@msi.umn.edu