HTX User's Manual

General Information

HTX is a computational tool for 3-D solid conduction heat transfer analysis. HTX uses a structured finite volume scheme to solve the transient energy conservation equation. HTX is written in FORTRAN 90. It can operate independently with user-specified boundary conditions, or it can exchange heat transfer and temperature data with CFD codes such as WIND.

Current Capabilities

CFD General Notation System (CGNS)

CGNS format is used for all data input/output, using calls to the CGNS library subroutines, except as explained in the next subsection. On the initial run, a single .cgns file contains grid information. During the computation, temperature field data are written to the same "*casefile.cgns"* in both cell-centered format for restarting the calculation, and in cell-vertex format, which is generally preferred for graphical display.

Surface Data Interchange (SDI)

Exchange of temperature and heat transfer data with another code, along solid/fluid boundaries, is implemented using the SDI library routines. The Wind-US code has been modified to include the SDI library for exchanging surface data. The user can specify any of the zone sides (imin, imax, etc.) for data interchange. Typically the WIND code will read surface temperature data, and write surface heat transfer data to an "*sdifile.*" HTX will read these heat transfer data, and write surface temperature data. The process is iterated until both codes converge to a steady solution. For specifying variable temperature data along a surface, a separate code may be supplied to generate the necessary interface information via the SDI library. Each interface is named and the exchange mode is specified in the Namelist input file.

The initial *sdifile*, consisting of surface grid and temperature data, is written by HTX. For the initial HTX run, constant-value approximations to the surface temperatures and heat transfer rate can be specified. The *sdifile* must contain a surface's grid data before its field data can be written.

Boundary conditions

Boundary conditions can be specified for any number of "faces," or sections of a block side. If the number of faces is not specified, the default value is one. The following boundary conditions are operational:

1. Constant user-specified temperature
2. Temperature field read from *sdifile*
3. Constant user-specified heat flux
4. Heat transfer field read from *sdifile*

Future or Partially Implemented Capabilities

Multiblocking

Currently, any number of zones (blocks) can be read from the *casefile.cgns*. The blocks can be in any (i,j,k) orientation with respect to one another. Zone interfacing is contiguous (one-to-one). The code will run with multiple zones, but as data exchange across "communication" boundaries is not yet operational, only single block operation is recommended. SDI is hard-coded for zone number one only.

Material Properties

Material properties are specified in the input file as constant and the same for all zones.

Timestepping

Currently, constant time steps are specified in the input file. Future revisions will include maximum stable timestepping, and local maximum stable timestepping, in order to speed convergence of steady-state calculations.

Getting Started

To run HTX, first set up a run directory containing the following files:

htx.exe	executable file
htx.inp	input file
casefile.cgns	"*casefile*" can be any name given by the user.

To compile HTX use the makefile in the source directory, modified as needed to invoke your compiler. Type "make," and then copy *htx.exe* to your run directory. The source directory must contain the pre-compiled CGNS and SDI library files *libcgns.a* and *libsdi.a*, and the CGNS header file *cgnslib_f.h*. The library files in the source distribution will probably not work on your machine, but will need to be recompiled from their respective sources. To remove all previously compiled object and module files and start over, type "make clean."

NAMELIST Input

Run Parameters Namelist Variables

NAMELIST RUN:

nmax	number of time steps

interval	time between SDI writes (no default)
ic	Initial condition flag (no default) [1: restart; 2: constant temperature start]
T_init	Initial temperature for ic=2 (no default)
mainfile	Grid and solution CGNS file (no default)
dt	Time step in seconds (no default)

Properties Parameters Namelist Variables

NAMELIST PROPERTIES

kappa_all	Heat conduction coefficient for all zones
C_all	Cp for all zones
rho_all	Density for all zones

Boundary Condition Parameters Namelist Variables

NAMELIST BOUND_CONS

libcs	Number of Boundary Conditions (faces) for lower I boundary
ljbcs	Number of Boundary Conditions (faces) for lower J boundary
lkbcs	Number of Boundary Conditions (faces) for lower K boundary
uibcs	Number of Boundary Conditions (faces) for upper I boundary
ujbcs	Number of Boundary Conditions (faces) for upper J boundary
ukbcs	Number of Boundary Conditions (faces) for upper K boundary

For each face (up to 20 faces):

libc(1:libcs)	Boundary condition for lower I boundary face 1: Constant user specified temperature: See li_temp 2: Temperature specified via Surface Data Interface (see SDI_IO) 3: Constant user specified heat flux: See li_q 4: Heat Flux specified via Surface Data Interface (see SDI_IO),
ljbc, lkbc, ljbc, lkbc, uibc, ujbc, ukbc	Similar to libc for other boundaries

lilj, lilk, liuj, liuk	Limiting gridlines for each lower I boundary face
ljli, ljlk, ljui, ljuk	Limiting gridlines for each lower J boundary face
lkli, lklj, lkui, lkuj	Limiting gridlines for each lower K boundary face
uilj, uilk, uiuj, uiuk	Limiting gridlines for each upper I boundary face
ujli, ujlk, ujui, ujuk	Limiting gridlines for each upper J boundary face
ukli, uklj, ukui, ukuj	Limiting gridlines for each upper K boundary face

li_temp	Constant temperature for libc=2 for each lower I boundary face
li_q	Constant heat flux for libc=4 for each lower I boundary face

ui_temp, ui_q, lj_temp, lj_q, uj_temp, uj_q, lk_temp, lk_q, uk_temp, uk_q Simlarly for other faces

Surface Data Interface (SDI) Namelist variables

NAMELIST SDI_IO:

Defaults in parentheses
Choices in brackets

num_sdi Number of Surface Data Interfaces (0)

For each surface interface, nsdi=1,num_sdi

interface(nsdi) Surface interface name

li_grid_write(nsdi) Write surface grid for lower I surface (.false.)
lj_grid_write(nsdi) Write surface grid for lower J surface (.false.)
lk_grid_write(nsdi) Write surface grid for lower K surface (.false.)
ii_grid_write(nsdi) Write surface grid for upper I surface (.false.)
uj_grid_write(nsdi) Write surface grid for upper J surface (.false.)
uk_grid_write(nsdi) Write surface grid for upper K surface (.false.)

li_temp_write(nsdi) Write temperature for lower I surface (.false.)
lj_temp_write(nsdi) Write temperature for lower J surface (.false.)
lk_temp_write(nsdi) Write temperature for lower K surface (.false.)

An example input file is shown below.

```
&run
nmax=1000,
interval=1000,
ic=1,
T_init=500.0,
mainfile='solid.cgns',
dt = 0.000002,
/
&properties
kappa_all=27.0,
C_all=460.0,
rho_all=7800.,
/
&sdi_io
num_sdi=1,
sdi(1)%interface='cht_ljheat_ujtemp',
sdi(1)%lj_grid_write(1)=.true.,
sdi(1)%uj_grid_write(1)=.true.,
```

```
sdi(1)%lj_heat_read(1)=.true.,
sdi(1)%uj_temp_read(1)=.true.,
/
&bound_cons
b_zone(1)%li_q(1)=0.0,
b_zone(1)%libc(1)=3,

b_zone(1)%ljbc(1)=4,
b_zone(1)%lj_q(1)=0.0,

b_zone(1)%lkbc(1)=3,
b_zone(1)%lk_q(1)=0.0,

b_zone(1)%uibc(1)=3,
b_zone(1)%ui_q(1)=0.0,

b_zone(1)%ujbc(1)=2,

b_zone(1)%ukbc(1)=3,
b_zone(1)%uk_q(1)=0.0,
/
```

On the initial run, *casefile.cgns* contains grid node coordinates and boundary indices, including connectivity data for multi-zone calculations. The *casefile.cgns* can be written using Gridgen.

How to Interface with Other Applications

The SDI (Surface Data Interface) library is used to provide a means to communicate with another application at a common surface. Each code must refer to the same Interface Name. For HTX this is the interface(nsdi) character string for each interface nsdi=1,num_sdi and num_sdi is the total number of interfaces.

Example applications which set a variable temperature and/or heat flux on an interface can be found in the cases directory under 2DNOZZLE and jpl_slice. For 2DNOZZLE, the specified temperature and heat flux are specified on the grid nodes, so they can be communicated directly to the surface. In the program SDI_add_Q_T.f90, the HTX NAMELIST is read to determine the interface information. For the jpl_slice example, the specified temperature is not on the surface nodes, so must be interpolated to these nodes before communicating the data via the SDI calls. In SDI_add_Touter.f90 a separate input file “htx_touter.inp” is called to read a NAMELIST for setting up the interface data. See the FORTRAN source for an example of how the communication is accomplished.

The Wind-US code can also communicate with the HTX code by setting the mode to CHT in the TTSPEC block and specifying the surfaces which will send/receive data from HTX. In the example below, the interface involves six surfaces and the communication will occur every 10 iterations:

```
TTSPEC
  type temperature
  mode CHT
  update interval 10
  zone 1
  surface jmax
  zone 2
  surface jmax
  zone 3
  surface jmax
  zone 4
  surface jmax
  zone 5
  surface jmax
  zone 6
  surface jmax
ENDTTSPEC
```

There is no requirement that the surface nodes be coincident between Wind-US and HTX, so an interpolation code is required to transfer the data between the two interfaces. For the interpolation coefficients to be generated, the HTX surface node coordinates must be created before any communication can take place. Therefore, an init argument to the HTX command is required to create the coordinate data, i.e. “htx.exe init” will create the coordinate data for the interfaces.

At each “update interval” Wind-US will execute the script “run_cht.pl”. This script must be in the directory where Wind-US was started. Example scripts are provided for each test case. The major function is to

1. Create the HTX interface coordinate data if necessary
2. Interpolate from the Wind-US interface to the HTX interface
3. Run the HTX code
4. Interpolate from the HTX interface to the Wind-US interface

An example of the Perl script “run_cht.pl” is shown below where the SDI surface is called “htx_1”.

```
#!/usr/bin/perl

#=============================================================================
==========
=comment

Runs a specified CHT code, and performs interpolations between the flow surface
and the solid surface.
```

```
=cut
#==========================================================================
==========

use strict;
use warnings;

#--------------------------------------------------------------------------
# User Options
#--------------------------------------------------------------------------

# The (relative or absolute) path to the CHT code to be run
my $cht_code                = "htx";

# Output file for the CHT code
my $cht_out                 = "cht.out";

# The command to create the CHT surface
my $cht_create_surface_cmd  = "$cht_code init";

# Output file for the interpolation code
my $interp_out              = "interp.out";

# The (relative or absolute) path to the interpolation code to be used
my $interp_code             = "sdi_interp_surface";

# CFD surface interface name
my $flow                    = "wind_cht";

# Heat conduction surface interface name
my $solid                   = "htx_1";

#--------------------------------------------------------------------------
# Local Variables
#--------------------------------------------------------------------------

my $cmd;

#--------------------------------------------------------------------------
# Create solid CGF file (if needed)
#--------------------------------------------------------------------------

if (not -e ".sdi/$solid.cgf") {
   print ".sdi/$solid.cgf doesn't exist, creating...";
```

```
    $cmd = "$cht_create_surface_cmd 1>>$cht_out 2>&1";
    system($cmd);

    print "ok\n";
}

#-------------------------------------------------------------------------
# Interpolate from $flowcgf -> $solidcgf
#-------------------------------------------------------------------------

print "interpolating HeatFlux from $flow to $solid...";

$cmd = "$interp_code $flow $solid HeatFlux Temp 1>$interp_out 2>&1";
system($cmd);

if (system("/bin/grep 'Done!' $interp_out 1>/dev/null 2>&1") == 0) {
    print "ok\n";
} else {
    print "failed; see $interp_out\n";
    exit(-1);
}

#-------------------------------------------------------------------------
# Run CHT code
#-------------------------------------------------------------------------

print "running cht code...";

$cmd = "$cht_code 1>>$cht_out 2>&1";
system($cmd);

print "ok\n";

#-------------------------------------------------------------------------
# Interpolate from $solid -> $flow
#-------------------------------------------------------------------------

print "interpolating Temp from $solid to $flow...";

$cmd = "$interp_code $solid $flow Temp 1>$interp_out 2>&1";
system($cmd);

if (system("/bin/grep 'Done!' $interp_out 1>/dev/null 2>&1") == 0) {
    print "ok\n";
} else {
    print "failed; see $interp_out\n";
```

```
    exit(-1);
}

#EOF
```