ppOpen-HPC:
Open Source Infrastructure for Development and Execution of Large-Scale Scientific Applications on Post-Peta-Scale Supercomputers with Automatic Tuning (AT).

ppOpen-APPL/FVM

ver. 0.3.0

User's guide

License

 This software is one of the results of the JST CREST "ppOpen-HPC: Open Source Infrastructure for Development and Execution of Large-Scale Scientific Applications on Post-Peta-Scale Supercomputers with Automatic Tuning (AT)" project.

Change History

Changes in release 0.2.0

- ✓ Local data structure has been modified.
- ✓ Performance of computation and communication of ver.0.2.0 has been much improved compared to that of ver.0.1.0.
- ✓

Changes in release 0.3.0

- ✓ New rule for names is applied
- ✓ Hexahedral meshes are supported
- ✓ Interface for binary-type mesh files
- ✓ *heat3D* code with future ppOpen-APPL/FVM library's

Table of Contents

1. Introduction

1.1 ppOpen-APPL/FVM and Finite Volume Discretization

ppOpen-APPL/FVM is a set of libraries for the development of parallel code using the finite-volume method (FVM) with the hybrid mesh system with tetrahedra, prisms and hexahedra. Operations utilize the edge-based method, which is proposed, and implemented in [1]. Actually, most of the subroutines and functions of ppOpen-APPL/FVM are based on those of the code developed in [1]. Although both the explicit and implicit time-marching methods are supported in ppOpen-APPPL/FVM, the current version (ver.0.3.0) does not include linear solvers for the implicit method. Parallel conjugate gradient (CG) solver to be introduced to ppOpen-APPL/FVM in future is provided as a function of "heat3d" code developed on ppOpen-APPL/FVM ver.0.3.0. The heat3D provides such functions which will be introduced to future version of ppOpen-APPL/FVM.

ppOpen-APPL/FVM is written in Fortran 90 with MPI and OpenMP. The subroutines and functions of ppOpen-APPL/FVM can be called from simulation code written in Fortran 90. Both flat-MPI and OpenMP/MPI hybrid programming models are available. Hexahedral meshes are newly supported in ver.0.3.0. Local data structure has been also modified in ver.0.3.0.

In the hybrid mesh system with tetrahedra and prisms, the surface of the model is covered with triangles, which provide geometric flexibility, while the structure of the mesh in the direction normal to the surface provides thin prismatic elements suitable for the viscous region (Fig.1 and Fig.2). The outermost layer of the prismatic mesh is then used as the inner boundary surface for a tetrahedral mesh (Fig.2), which covers the rest of the computational domain. Tetrahedral meshes are also suitable for connecting different prismatic regions. Figure 3 shows an example of the hybrid meshes around a sphere. The spatial discretization proceeds by constructing a dual cell around each node N that represents the control volume over which the integral averages of the temporal derivatives are evaluated. The two-dimensional analogy of defining dual cells for different configurations in a triangular-quadrilateral hybrid mesh is illustrated in Fig.4. The duals are defined by connecting the midpoints of the edges and centroids of the triangular and/or quadrilateral faces that share the node. Dual cells for a three-dimensional hybrid grid are constructed along similar lines using the centroids of the faces and cells with which each node is associated. The surface area is computed using the dual mesh construction of Fig.4 and Fig.5, by accumulating the areas of each dual-mesh face that shares the edge. Thread parallelization of edge-based operations using OpenMP is not straight-forward due to data dependency. Therefore, a multicolor ordering procedure [2] is introduced for avoiding data dependency during parallel computations.

1.2 Local Data Structure

A proper definition of the layout of the distributed data structures is an important factor determining the efficiency of parallel computations with unstructured meshes. The local data

structures in GeoFEM are node-based with overlapping elements, and as such, are appropriate for the preconditioned iterative solvers used in GeoFEM [2]. In FEM and FVM, independent variables for linear equations (*e.g.,* velocity, temperature, etc.) are defined on nodes. From the viewpoint of efficiency in parallel computation, the number of nodes should be balanced among the domains. Therefore, GeoFEM adopts the node-based partitioning method. In the node-based manner of partitioning, overlapping elements among the domains are required in order to utilize *element-by-element* operations in FEM procedures such as matrix assembling. Fig.6 shows an example of overlapping elements. Each node requires information from all of the elements surrounding the node. In Fig.6, the elements in gray are shared by more than two domains and information on these elements is required in order to complete the process for each node. Each domain must consist of information on overlapping elements in order to conduct *element-by-element* procedures in a purely parallel manner.

Communication among processors occurs during computation. Subroutines for communications in structured grids are provided by MPI. However, users are required to design both the local data structure and communications for unstructured grids. In GeoFEM, each domain contains the following local data:

- Nodes originally assigned to the domain
- Elements that include the assigned nodes
- All nodes that form elements but are from external domains
- A communication table for sending and receiving data
- Boundary conditions and material properties

Nodes are classified into the following 3 categories from the viewpoint of message passing:

- Internal nodes (originally assigned to the domain)
- External nodes (forming the element in the domain but that are from external domains)
- Boundary nodes (external nodes of other domains)

Fig.7 shows a sample partitioning. If the PE #2 partition in Fig.7 and Fig.8 is considered, nodes are classified as follows:

- Internal nodes {1,2,3,4,5,6}
- External nodes {7,8,9,10,11,12}
- Boundary nodes {1,2,5,6}

In ppOpen-APPL/FVM, some additional information for mesh adaptation and grid hierarchy has been added to the original static GeoFEM data structure. In order to conform with the interface of library for adaptive mesh refinement and the data migration procedure, *double-numbering* of nodes, elements and edges has been implemented where items are identified by 2 types of ID (original partition and local ID) [1].

Communication tables between neighboring domains are also included in the local data. Values for *boundary* nodes in the domains are *sent* to the neighboring domains and are *received* as *external* nodes at the *destination* domain. This data structure, described in Fig.7, and the communication procedure described in Fig.8 provide excellent parallel efficiency. Fig.9 describes the Fortran subroutine of communication procedures based on GeoFEM [2]. In this figure, the arrays `EXPORT_INDEX` and `EXPORT_NODE` correspond to the communication table for the send-phase, and `IMPORT_INDEX` and `IMPORT_NODE` correspond to the receiving part of the communication table. This type of communication occurs in the procedure used for computing the matrix-vector product of Krylov iterative solvers described in the next subsection. The partitioning program of ppOpen-APPL/FVM works on a single PE, and divides the entire initial mesh into distributed local data. In ver.0.2.0, local data structure has been modified, and numbering of external nodes in each neighboring domain is continuous. Therefore, receiving part is much simpler in ver.0.2.0 compared to that in ver.0.1.0, as shown in Fig.9 (b) and (c).

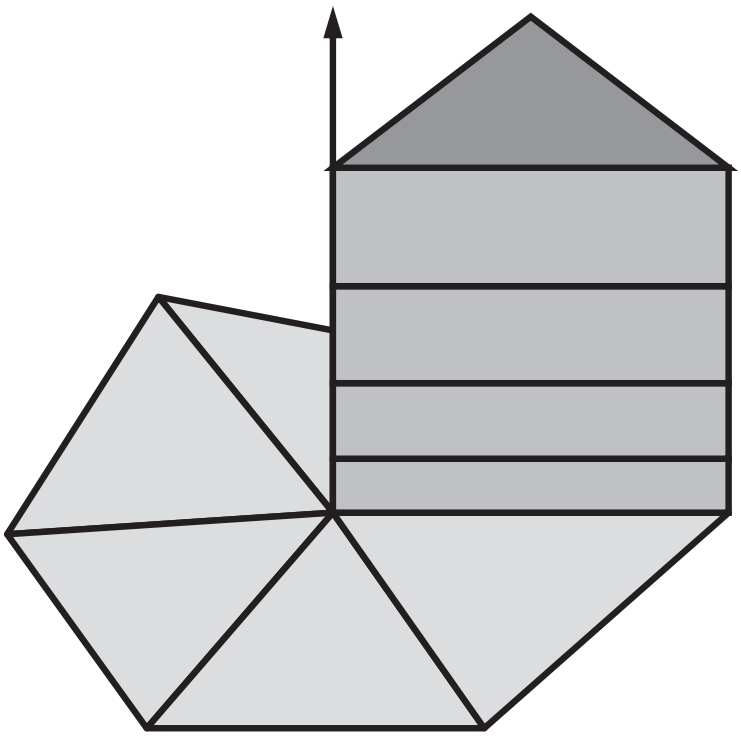


Fig. 1 Prismatic meshes generated from surface triangles in the normal-to-surface direction [1]

(a) Prisms

(b) Tetrahedra

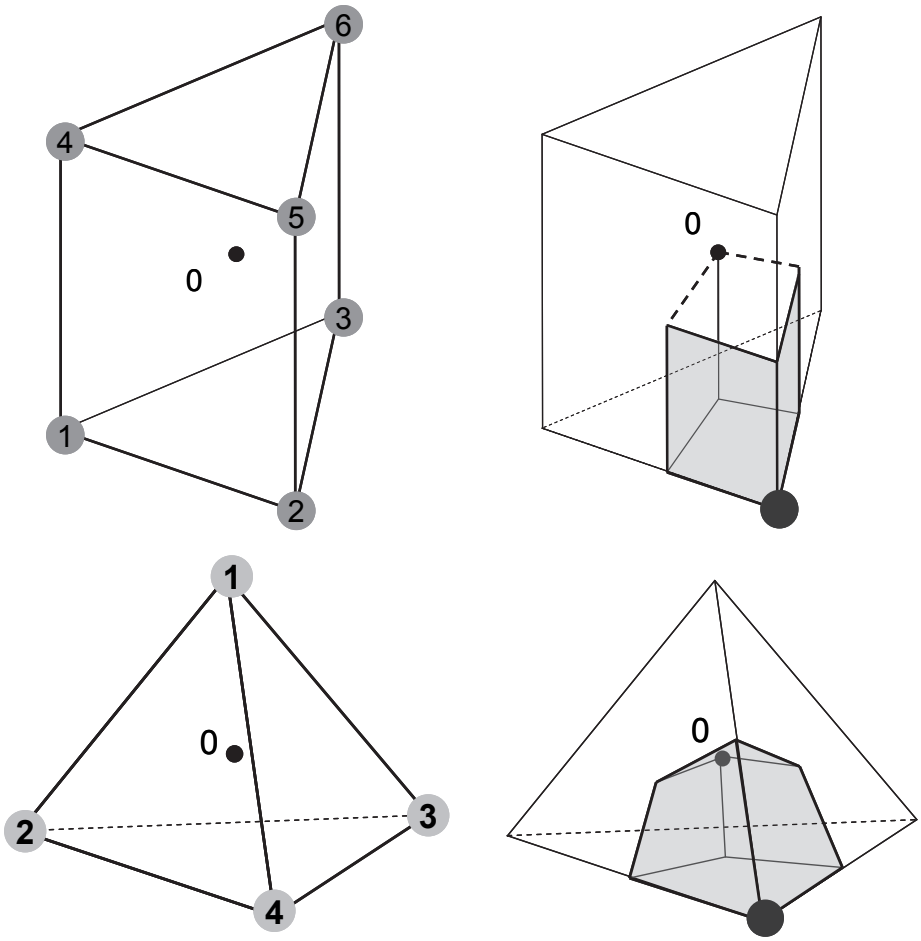


Fig. 2 Prismatic and tetrahedral meshes and dual-cells [1]

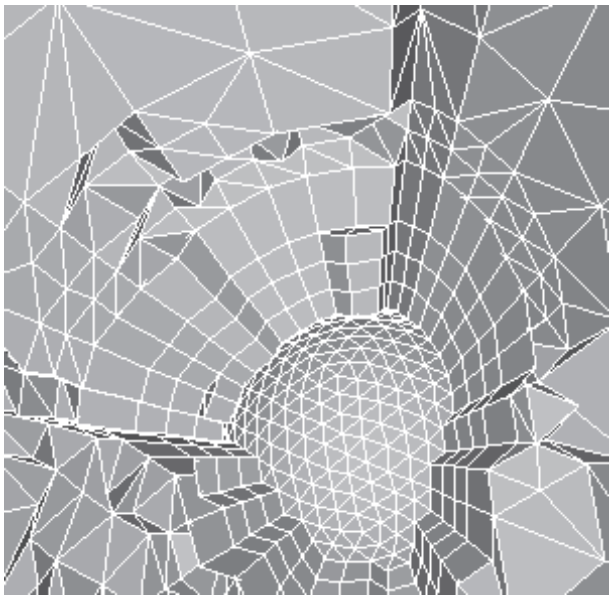


Fig. 3 An example of prismatic/tetrahedral hybrid meshes [1]

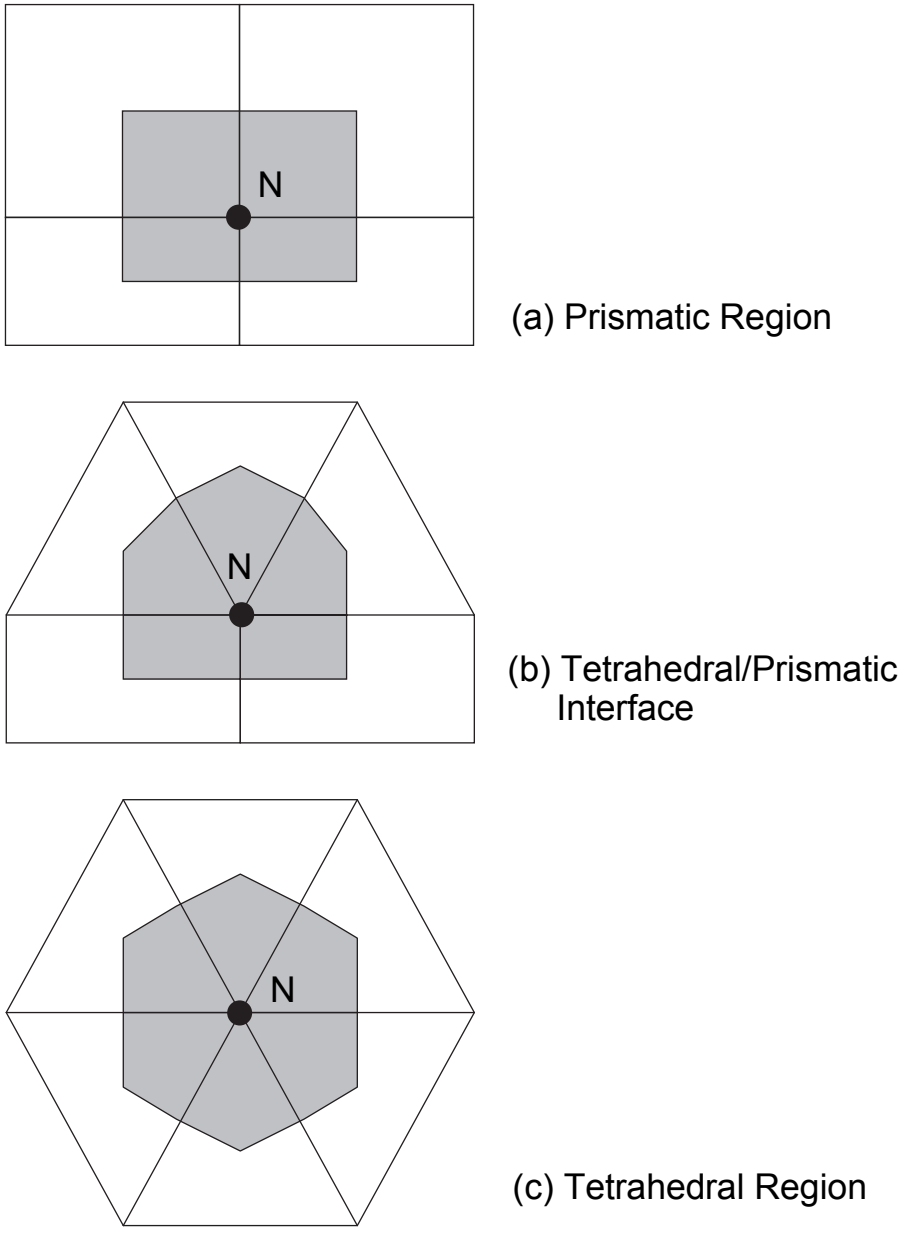


Fig. 4 Dual volume constructions for mixed-element topology. Two-dimensional analogies for dual mesh around a node in the (a) prismatic region, (b) tetrahedral-prismatic interface, and (c) tetrahedral region [1]

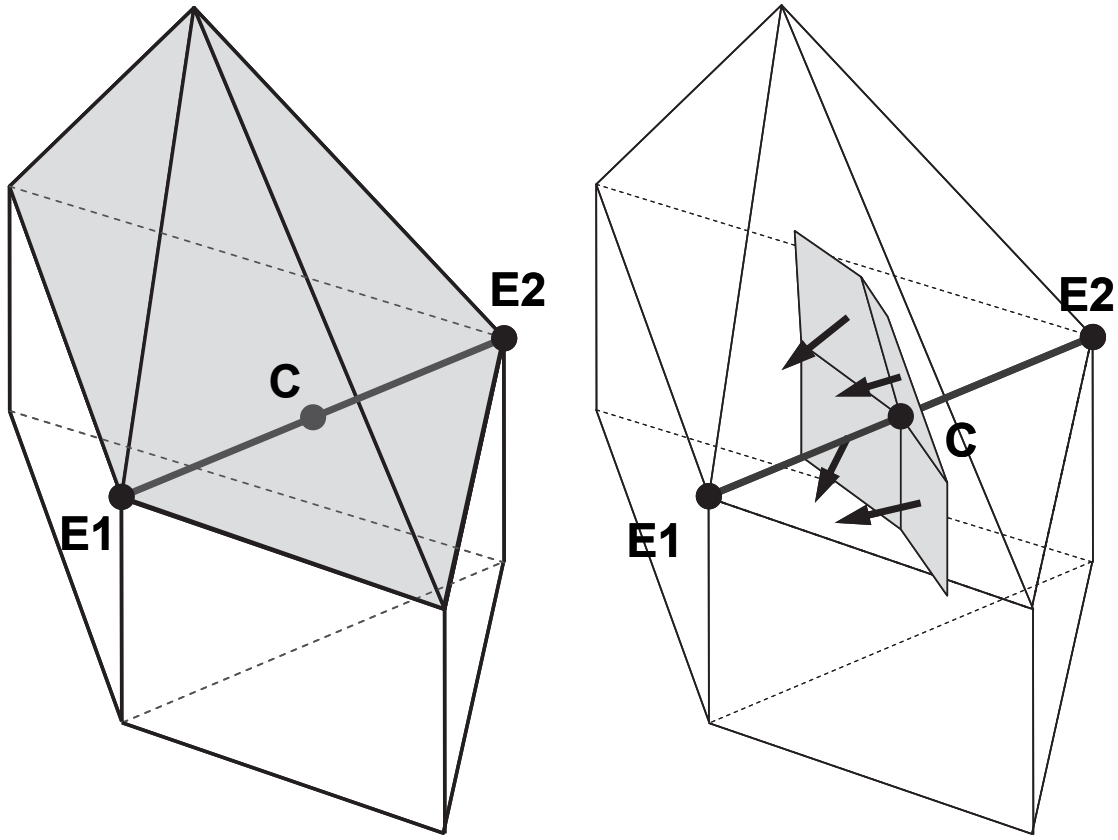


Fig. 5 Edge-dual volume defined around the edges for computing the gradients of primitive variables at the edge centers [1]

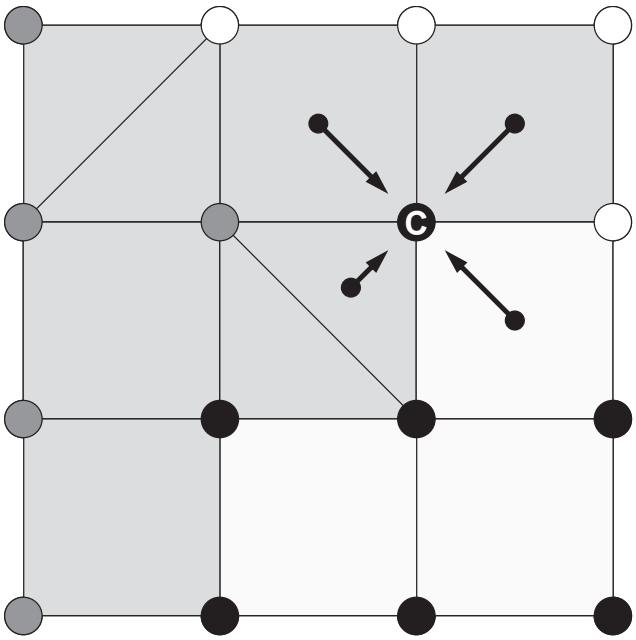


Fig. 6 *Element-by-element* operations around node *C*. Gray meshes are *overlapped* among domains

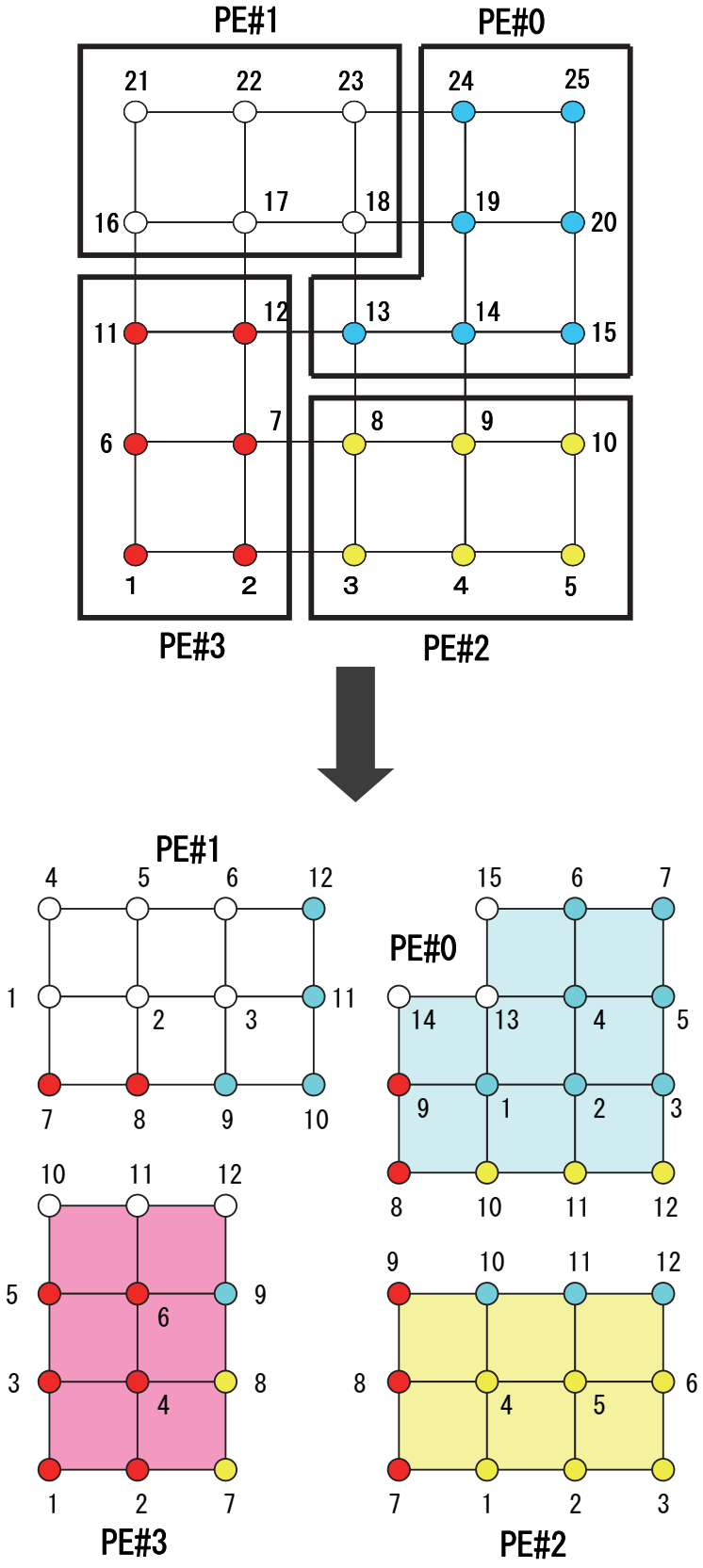
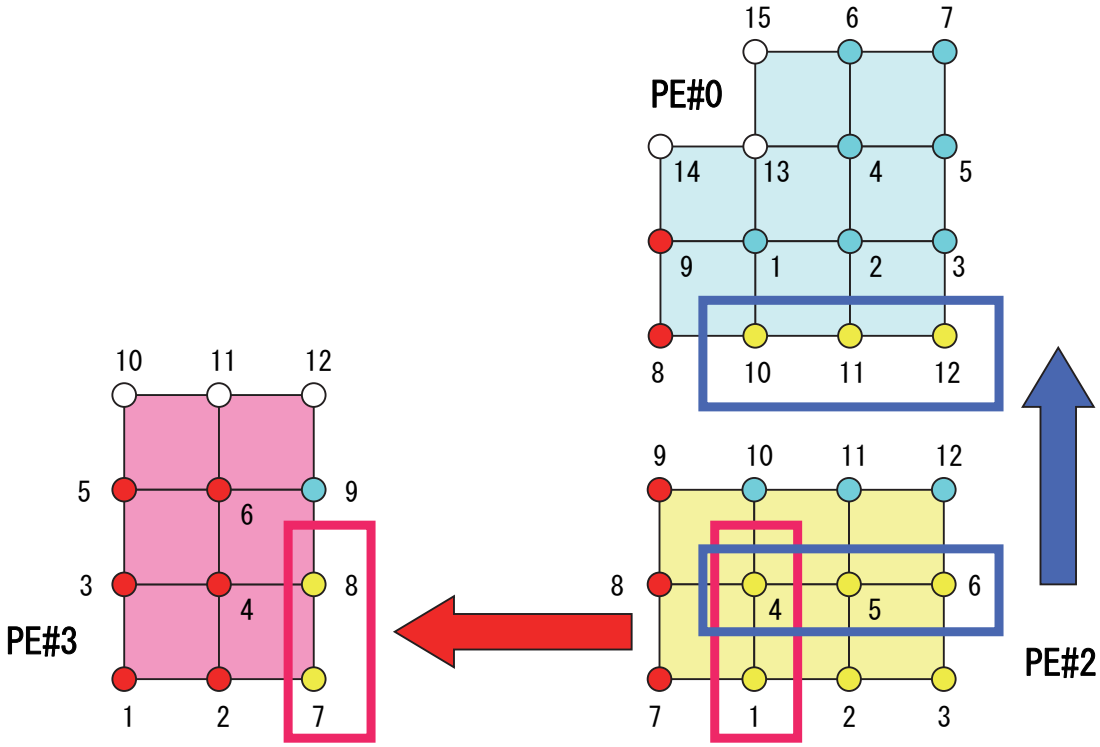


Fig. 7 Node-based partitioning into 4 PEs [2]

(a) SEND phase



(b) RECEIVE phase

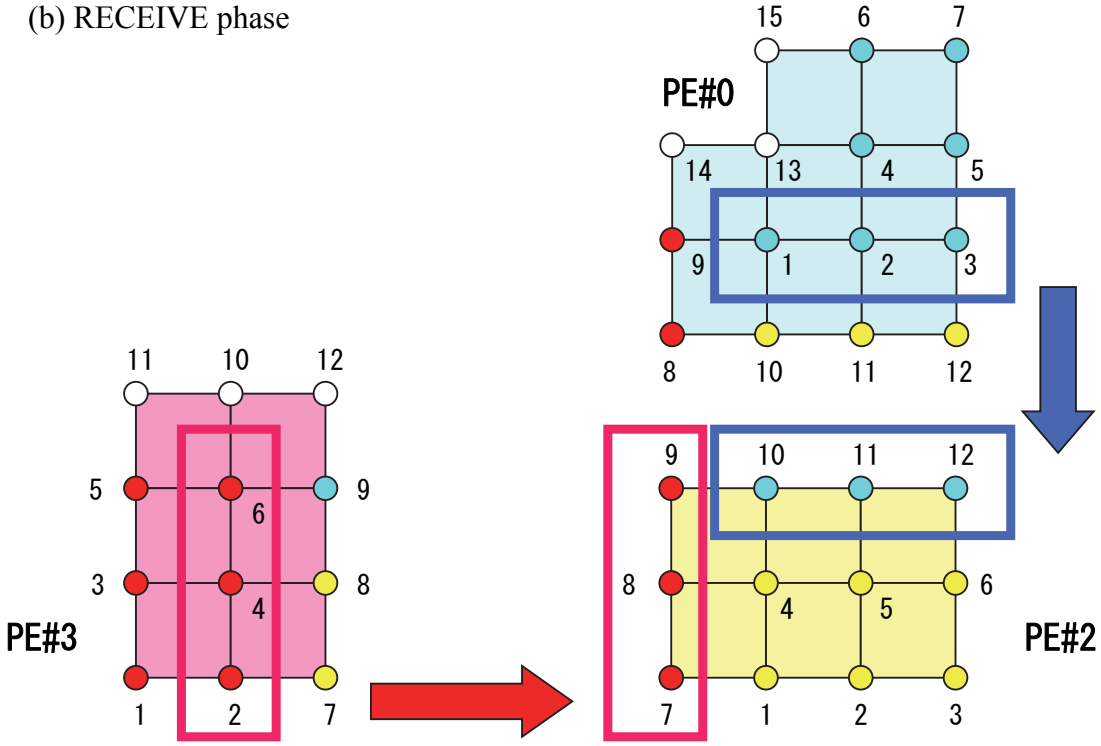


Fig. 8 Communication among processors [2]

(a) SEND phase

```
 do neib= 1, NEIBPETOT
   do k= export_index(neib-1)+1, export_index(neib)
     kk= export_item(k)
     SENDbuf(k)= VAL(kk)
   enddo
 enddo

 do neib= 1, NEIBPETOT
   iS_e= export_index(neib-1) + 1
   iE_e= export_index(neib  )
   BUFlength_e= iE_e + 1 - iS_e

   call MPI_ISEND
&       (SENDbuf(iS_e), BUFlength_e, MPI_DOUBLE_PRECISION,
&        NEIBPE(neib), 0, MPI_COMM_WORLD, request_send(neib),
&        ierr)
 enddo

 call MPI_WAITALL (NEIBPETOT, request_send, stat_recv, ierr)
```

(b) RECEIVE phase (ver.0.1.0)

```
 do neib= 1, NEIBPETOT
   iS_i= import_index(neib-1) + 1
   iE_i= import_index(neib  )
   BUFlength_i= iE_i + 1 - iS_i

   call MPI_IRECV
&       (RECVbuf(iS_i), BUFlength_i, MPI_DOUBLE_PRECISION,
&        NEIBPE(neib), 0, MPI_COMM_WORLD, request_recv(neib),
&        ierr)
  enddo

  call MPI_WAITALL (NEIBPETOT, request_recv, stat_recv, ierr)

  do neib= 1, NEIBPETOT
     do k= import_index(neib-1)+1, import_index(neib)
       kk= import_item(k)
       VAL(kk)= RECVbuf(k)
     enddo
  enddo
```

(c) RECEIVE phase (ver.0.2.0 or later)

```
 do neib= 1, NEIBPETOT
   iS_i= import_index(neib-1) + 1 + N
   iE_i= import_index(neib  ) +     N
   BUFlength_i= iE_i + 1 - iS_i

   call MPI_IRECV
&       (VAL(iS_i), BUFlength_i, MPI_DOUBLE_PRECISION,
&        NEIBPE(neib), 0, MPI_COMM_WORLD, request_recv(neib),
&        ierr)
  enddo
```

Fig. 9 Communication procedures among domains

2. Installation and Quick Start

2.1 ppohFVM_0.3.0.tar

The “ppohFVM_0.3.0.tar” archive includes the following:

- Source code files of “ppOpen-APPL/FVM ver.0.3.0”
- Source code files of “ppOpen-APPL/FVM-Tool/Partitioner” which is a partitioning utility for ppOpen-APPL/FVM
- Source code files of “hybNS”, which is a solver for 3D compressible Navier-Stokes equations with explicit time-marching, developed using ppOpen-APPL/FVM
- Source code files of “hybNS_mg” which is a mesh generator for hybNS
- Source code files of “heat3D”, which is a solver for 3D steady-state heat conduction equations, developed using ppOpen-APPL/FVM
- Source code files of “pmesh” and “pmesh_bin”, which are parallel mesh generator for heat3D
- Sample Makefiles
- Sample data files for “hybNS”

2.2 Structure of Directories

The “ppohFVM_0.3.0.tar” archive includes the following directories. `$(CUR)` denotes the directory where the “ppohFVM_0.3.0.tar” archive is unpacked.

Name of Directory	Contents
`$(CUR)/src`	source code files of ppOpen-APPL/FVM
`$(CUR)/utils/partitioner`	source code files of ppOpen-APPL/FVM-Tool/Partitioner
`$(CUR)/examples/hybNS/src`	source code files of hybNS using ppOpen-APPL/FVM
`$(CUR)/examples/hybNS/mg`	source code files of hybNS_mg, mesh generator for hybNS
`$(CUR)/examples/hybNS/data`	sample data sets of hybNS (mesh data)
`$(CUR)/examples/hybNS/run`	sample data sets of hybNS (control data)
`$(CUR)/examples/hybNS/run/results`	sample results of hybNS
`$(CUR)/examples/heat3D/src`	source code files of heat3D using ppOpen-APPL/FVM
`$(CUR)/examples/heat3D/pmesh`	source code files of pmesh and pmesh_bin, parallel mesh generators for heat3D
`$(CUR)/examples/heat3D/data`	sample data sets of heat3D (mesh data)
`$(CUR)/examples/heat3D/run`	sample data sets of heat3D (control data)
`$(CUR)/examples/heat3D/run/results`	sample results of heat3D
`$(CUR)/include`	directory that stores created module files
`$(CUR)/lib`	directory that stores created libraries
`$(CUR)/bin`	directory that stores created exec. files
`$(CUR)/doc`	documents
`$(CUR)/etc`	examples of ‘`Makefile.in`’

2.3 Quick Start

(1) Preparation

- Fortran 90 compilers (Operations have been confirmed with Intel, PGI, Fujitsu compilers)
- MPI library
- OpenMP must be supported if you want to develop OpenMP/MPI Hybrid code
- METIS library (v.4) (libmetis.a)

***** NOTICE *** The most recent version (METIS 5.0) does not work. Please obtain the previous version (v.4) from the following site:**

http://glaros.dtc.umn.edu/gkhome/fsroot/sw/metis/OLD

(2) Modify 'Makefile.in'

Samples of 'Makefile.in' are found in `$(CUR)/etc`

Examples of 'Makefile.in'	Compiler, Parallel Programming Models
`$(CUR)/etc/Makefile.in.fx10.flatmpi`	Flat MPI for Fujitsu FX10
`$(CUR)/etc/Makefile.in.intel.flatmpi`	Intel Compiler
`$(CUR)/etc/Makefile.in.pgi.flatmpi`	PGI Compiler
`$(CUR)/etc/Makefile.in.fx10.hybrid`	OpenMP/MPI Hybrid for Fujitsu FX10
`$(CUR)/etc/Makefile.in.intel.hybrid`	Intel Compiler
`$(CUR)/etc/Makefile.in.pgi.hybrid`	PGI Compiler

Options in 'Makefile.in'	Descriptions
`$(MPIF90), $(MPIF77)`	FORTRAN 90/77 with MPI
`$(F90), $(F77)`	FORTRAN 90/77 for single core
`$(sFFLAGS)`	Compiler options for Optimizations for ppOpen-APPL/FVM-Tool/Partitioner (DO NOT INCLUDE "OpenMP" flags)
`$(sMGFLAGS)`	Compiler options for Optimizations for hybNS_mg (DO NOT INCLUDE "OpenMP" flags)
`$(pMGFLAGS)`	Compiler options for Optimizations for ppOpen-APPL/FVM and hybNS
`$(METISDIR)`	Name of directory, where 'libmetis.a' is located
`$(PREFIX)/include`	directory that holds installed module files
`$(PREFIX)/lib`	directory that holds installed libraries
`$(PREFIX)/bin`	directory that holds installed exec. files

***** NOTICE ***: $(PREFIX) directory must be specified as ABSOLUTE/FULL path.**

(3) Complile/install ppOpen-APPL/FVM and ppOpen-APPL/FVM-Tool/Partitioner

Operations	Files created (libraries, module files, exec. files)
$> cd $(CUR)/** **$> make clean	
$> make	$(CUR)/lib/ppohFVMlib.a $(CUR)/bin/ppohFVM_part
$> make ppohFVM	ppOpen-APPL/FVM only $(CUR)/lib/ppohFVMlib.a
$> make part	ppOpen-APPL/FVM-Tool/Partitioner only $(CUR)/bin/ppohFVM_part
$> make install	$(PREFIX)/lib/ppohFVMlib.a $(PREFIX)/bin/ppohFVM_part
$> make ppohFVM_install	$(PREFIX)/lib/ppohFVMlib.a
$> make part_install	$(PREFIX)/bin/ppohFVM_part

(4) Compile/install hybNS and hybNS_mg

Operations	Files created (libraries, module files, exec. files)
$> cd $(CUR)/** **$> make hybNS_clean	
$> make hybNS	$(CUR)/bin/hybNS $(CUR)/bin/hybNS_mg
$> make bin_install	$(PREFIX)/bin/hybNS $(PREFIX)/bin/hybNS_mg

(5) Compile/install heat3D, pmesh and pmesh_bin

Operations	Files created (libraries, module files, exec. files)
$> cd $(CUR)/** **$> make heat3D_clean	
$> make heat_3D	$(CUR)/bin/heat3D $(CUR)/bin/pmesh, pmesh_bin
$> make bin_install	$(PREFIX)/bin/heat3D $(PREFIX)/bin/pmesh, pmesh_bin

***** NOTICE ***: Processes (4) and (5) must be done after (3).**

(6) Running the code

(hybNS)

```
$> cd $(CUR)/examples/hybNS/run
$> mpirun -np 8 <$PREFIX>/bin/hybNS
```

with appropriate thread number for OpenMP (or corresponding operations)

(heat3D)

```
$>  cd $(CUR)/examples/heat3D/run
$>  mpirun -np 8 <$PREFIX>/bin/heat3D
```

with appropriate thread number for OpenMP (or corresponding operations)

(7) Clean/Uninstall

```
$>  cd $(CUR)/
```

`$> make clean`	Clean files
`$> make uninstall`	Delete all installed files and directories

3. ppOpen-APPL/FVM

3.1 Structure

Figure 10-12 show the structure of ppOpen-APPL/FVM.

Fig.10 Structure of ppOpen-APPL/FVM (1)

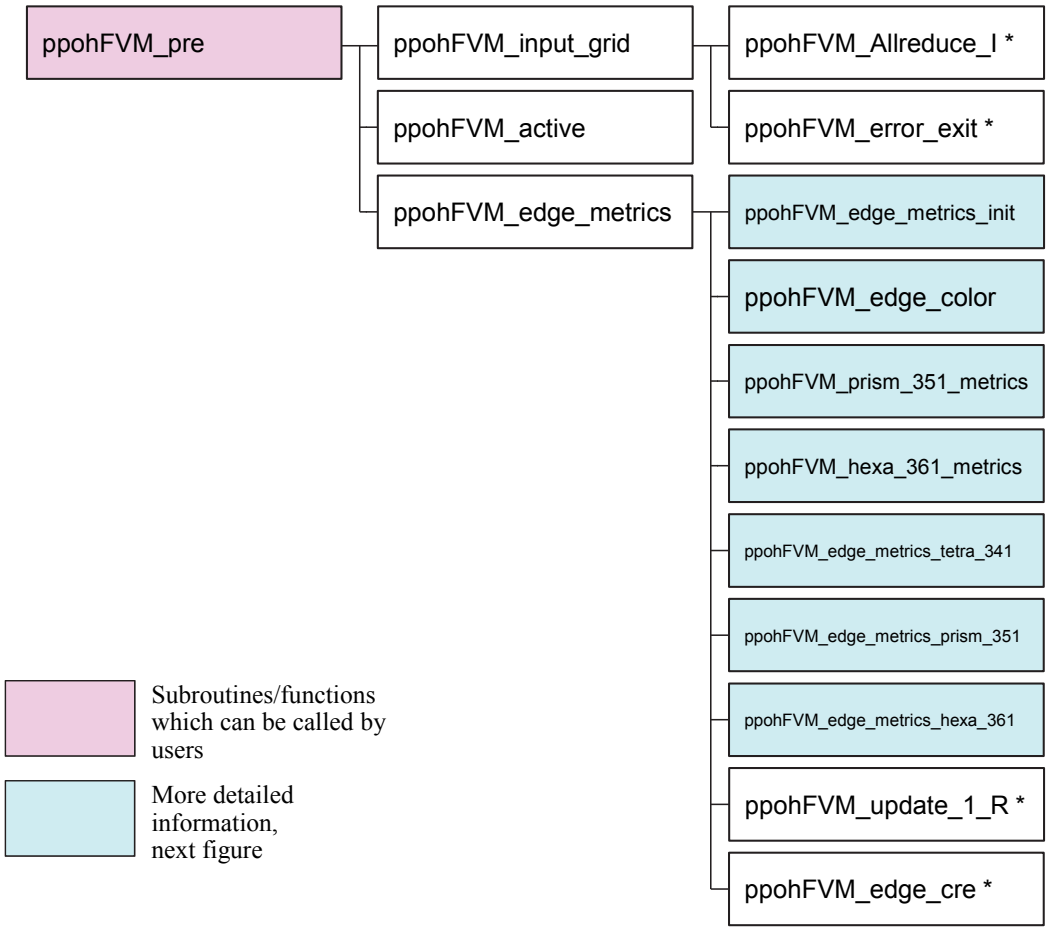


Fig.11 Structure of ppOpen-APPL/FVM (2): Subroutines called from ppohFVM_pre

Fig.12 Structure of ppOpen-APPL/FVM (3)

3.2 Modules

m_ppohFVM_util

This module contains information on variables for meshes, groups, communications, and edges.

Contains the following subroutines/functions

(none)

ppohFVM_precision.inc

```
integer, parameter   ::  ppohFVM_kint  = 4
integer, parameter   ::  ppohFVM_kreal = 8
integer, parameter   ::  ppohFVM_name_len = 80
```

ppohFVM_util.f90

```
module m_ppohFVM_util
  implicit none
  public
  include 'mpif.h'
  include 'ppohFVM_precision.inc'

  real(kind=ppohFVM_kreal):: ppohFVM_03rd, ppohFVM_06th, ppohFVM_08th

  integer(kind=ppohFVM_kint),parameter :: ppohFVM_sum              = 46801
  integer(kind=ppohFVM_kint),parameter :: ppohFVM_prod             = 46802
  integer(kind=ppohFVM_kint),parameter :: ppohFVM_max              = 46803
  integer(kind=ppohFVM_kint),parameter :: ppohFVM_min              = 46804
  integer(kind=ppohFVM_kint),parameter :: ppohFVM_integer          = 53951
  integer(kind=ppohFVM_kint),parameter :: ppohFVM_single_precision = 53952
  integer(kind=ppohFVM_kint),parameter :: ppohFVM_double_precision = 53953
  integer(kind=ppohFVM_kint),parameter :: ppohFVM_character        = 53954

type st_ppohFVM_file_info
  character(len=ppohFVM_name_len):: header(100)
  character(len=ppohFVM_name_len):: file  (100)
  logical :: mesh_asci
end type st_ppohFVM_file_info

type st_ppohFVM_local_mesh
  integer n_node_global
  integer n_node, n_internal
  real(kind=ppohFVM_kreal),pointer:: node   (:,:)
  integer,pointer::          node_id(:,:)
  integer n_elem, ne_internal, n_material
  integer,pointer:: elem_type(:)
  integer,pointer:: index_elem(:), ptr_elem(:)
  integer,pointer:: mat_id (:)
  integer,pointer:: elem   (:,:)
  integer,pointer:: elem_id(:,:)
  integer,pointer:: ne_internal_list(:)

  real(kind=ppohFVM_kreal),pointer:: material(:,:)
  real(kind=ppohFVM_kreal),pointer:: voln(:), volc(:)
  real(kind=ppohFVM_kreal),pointer:: sarea(:,:)

  integer :: CoarseGridLevels, HOWmanyADAPTATIONs
  integer,pointer:: WhenIwasRefined_node(:)
  integer,pointer:: WhenIwasRefined_elem(:)
  integer,pointer:: adaptation_parent_type (:)
  integer,pointer:: adaptation_type (:)
  integer,pointer:: adaptation_level(:)
  integer,pointer:: adaptation_parent  (:,:)
  integer,pointer:: adaptation_children(:,:)
  integer,pointer:: index_children(:)

  integer:: n_tetra_341, n_prism_351, n_hexa_361
  integer:: n_tetra_342, n_prism_352, n_hexa_362
  integer:: n_ACTtetra_341, n_ACTprism_351, n_ACThexa_361
  integer:: n_ACTtetra_342, n_ACTprism_352, n_ACThexa_362
```

```
(cont.)
        integer,pointer:: tetra_341_id(:), ACTtetra_341_id(:), tetra_342_id(:), ACTtetra_342_id(:)
        integer,pointer:: prism_351_id(:), ACTprism_351_id(:), prism_352_id(:), ACTprism_352_id(:)
        integer,pointer:: hexa_361_id(:), ACThexa_361_id(:), hexa_362_id(:), ACThexa_362_id(:)
        integer,pointer:: ne_internal_flag(:)
      end type st_ppohFVM_local_mesh

      type st_ppohFVM_ne_grp
        integer n_enum_grp
        character(len=ppohFVM_name_len),pointer:: enum_grp_name (:)
        integer,pointer::                        enum_grp_index(:)
        integer,pointer::                        enum_grp_node (:)
      end type st_ppohFVM_ne_grp

      type st_ppohFVM_s_grp
        integer n_surf_grp
        character(len=ppohFVM_name_len),pointer:: surf_grp_name(:)
        integer,pointer:: surf_grp_index(:)
        integer,pointer:: surf_grp_node (:,:)
      end type st_ppohFVM_s_grp

      type st_ppohFVM_grp_data
        type(ppohFVM_ne_grp) node_grp
        type(ppohFVM_ne_grp) elem_grp
        type(ppohFVM_s_grp)  surf_grp
      end type st_ppohFVM_grp_data

      type st_ppohFVM_comm_info
        integer my_rank, PEsmpTOT, PETOT, COMM
        integer n_neighbor_pe
        integer,pointer:: neighbor_pe(:)
        integer,pointer:: import_index(:)
        integer,pointer:: import_item(:)
        integer,pointer:: export_index(:)
        integer,pointer:: export_item(:)
        integer,pointer:: global_node_id(:)
        integer,pointer:: global_elem_id(:)
        real(kind=ppohFVM_kreal),pointer:: WS (:), WR (:)
        real(kind=ppohFVM_kreal),pointer:: WS2(:), WR2(:)
        real(kind=ppohFVM_kreal),pointer:: WS5(:), WR5(:)
      end type st_ppohFVM_comm_info

      type st_ppohFVM_edge_info
        logical :: use_edges
        integer(kind=ppohFVM_kint) :: n_edge, n_ACTedge, n_edge_color
        real(kind=ppohFVM_kreal), pointer :: area(:,:)
        real(kind=ppohFVM_kreal), pointer :: vol (:)

        integer(kind=ppohFVM_kint), pointer :: edgnod (:,:)
        integer(kind=ppohFVM_kint), pointer :: OtoN(:), NtoO(:)
        integer(kind=ppohFVM_kint), pointer :: color_index(:)
        integer(kind=ppohFVM_kint), pointer :: color_item (:)
        integer(kind=ppohFVM_kint), pointer :: ACTedge(:)

      end type st_ppohFVM_edge_info

      end module m_ppohFVM_util
```

Parameters

ppohFVM_kint I =4

ppohFVM_kreal I =8

ppohFVM_name_len I =80

ppohFVM_O3rd R =1/3

ppohFVM_O6th R =1/6

ppohFVM_O8th R =1/8

ppohFVM_sum I ID for MPI_SUM (=46801)

ppohFVM_prod	I	ID for MPI_PROD (=46802)
ppohFVM_max	I	ID for MPI_MAX (=46803)
ppohFVM_min	I	ID for MPI_MIN (=46804)
ppohFVM_integer	I	=53951
ppohFVM_single_precision	I	=53952
ppohFVM_double_precision	I	=53953
ppohFVM_character	I	=53954

st_ppohFVM_file_info Derived Type

header	C	Header of distributed files [ppohFVM_name_len][100]
file	C	Name of distributed files [ppohFVM_name_len][100]
mesh_asci	L	T: mesh files in ASCII format (default), F: mesh files in binary format,

st_ppohFVM_local_mesh Derived Type

n_node_global	I	Total number of nodes (global)
n_node	I	Total number of nodes (local)
n_internal	I	Total number of internal nodes (local)
node	RA	Coordinates of each node [3,n_node]
node_id	IA	Node ID based on the "double-numbering" rule [2,n_node] node_id(1,i): Local node ID at the "home" partition of the node node_id(2,i): Rank ID of the "home" partition of the node
n_elem	I	Total number of local elements
ne_internal	I	Total number of internal elements
n_material	I	Total number of material components (If n_material is not 0, material components are defined at each element)
elem_type	IA	Type of element, 341: tetrahedron, 351: prism, 361: hexahedron [n_elem]
index_elem	IA	Index of number of nodes for each element [0:n_elem]
ptr_elem	IA	List of local node ID for each element [index_elem(n_elem)]
elem_type	IA	Material ID of each element [n_elem]
elem	IA	(not used)
elem_id	IA	Element ID based on the "double-numbering" rule [n_elem,2] elem_id(i,1): Local element ID at the "home" partition of the element elem_id(i,2): Rank ID of the "home" partition of the element

ne_internal_list

IA List of internal elements [ne_internal]

material RA Material property of each element [n_elem,n_material]

(If n_material is equal to 0, this array is not defined)

voln RA Volume of each dual-cell [n_node]

volc RA Volume of each element [n_elem]

sarea RA Surface area of each dual-cell in X-Y-Z directions [3,n_elem]

CoarseGridLevels, HOWmanyADAPTATIONs

I (not used)

WhenIwasRefined_node, WhenIwasRefined_elem, adaptation_parent_type, adaptation_level, adaptation_parent, adaptation_children, index_children

IA (not used)

adaptation_type

IA Adaptation type of each element [n_elem]

n_tetra_341 I Total number of tetrahedra (linear element, type: 341)

n_prism_351 I Total number of prisms (linear element, type: 351)

n_hexa_361 I Total number of hexahedra (linear element, type: 361)

n_tetra_342 I Total number of tetrahedra (2^{nd}-order element, type: 342)

n_prism_352 I Total number of prisms (2^{nd}-order element, type: 352)

n_hexa_362 I Total number of hexahedra (2^{nd}-order element, type: 362)

n_ACTtetra_341 I Total number of active tetrahedra (linear element, type: 341)

n_ACTprism_351 I Total number of active prisms (linear element, type: 351)

n_ACThexa_361 I Total number of active prisms (linear element, type: 361)

n_ACTtetra_342 I Total number of active tetrahedra (2^{nd}-order element, type: 342)

n_ACTprism_352 I Total number of active prisms (2^{nd}-order element, type: 352)

n_ACThexa_362 I Total number of active prisms (2^{nd}-order element, type: 362)

tetra_341_id IA Element ID of tetrahedral meshes [n_tetra_341]

prism_351_id IA Element ID of prismatic meshes [n_prism_351]

hexa_361_id IA Element ID of hexahedral meshes [n_hexa_361]

tetra_342_id IA Element ID of tetrahedral meshes [n_tetra_342]

prism_352_id IA Element ID of prismatic meshes [n_prism_352]

hexa_362_id IA Element ID of hexahedral meshes [n_hexa_362]

ACTtetra_341_id IA Element ID of the active tetrahedral meshes [n_ACTtetra_341]

ACTprism_351_id IA Element ID of the active prismatic meshes [n_ACTprism_351]

ACThexa_361_id IA Element ID of the active hexahedral meshes [n_ACThexa_361]

ACTtetra_342_id IA Element ID of the active tetrahedral meshes [n_ACTtetra_342]
ACTprism_352_id IA Element ID of the active prismatic meshes [n_ACTprism_352]
ACThexa_362_id IA Element ID of the active hexahedral meshes [n_ACThexa_362]
ne_internal_flag
IA Flag for internal elements [n_elem]

st_ppohFVM_ne_grp Derived Type
n_enum_grp I Number of groups
enum_grp_name CA Name of each group [ppohFVM_name_len][n_enum_grp]
enum_grp_index
IA Index of the number of components for each group [0:n_enum_grp]
enum_grp_node IA List of components for each group
[enum_grp_index(n_enum_grp)]

st_ppohFVM_s_grp Derived Type
n_surf_grp I Number of surface groups
surf_grp_name CA Name of each group [ppohFVM_name_len][n_surf_grp]
surf_grp_index
IA Index of the number of components for each group [0:n_surf_grp]
surf_grp_node IA List of components (element ID and local surface ID) for each group
[enum_grp_index(n_enum_grp),2]

st_ppohFVM_grp_data Derived Type
node_grp Derived Type: ppohFVM_ne_grp
elem_grp Derived Type: ppohFVM_ne_grp
surf_grp Derived Type: ppohFVM_s_grp

st_ppohFVM_comm_info Derived Type
my_rank I Rank ID of the MPI process
PEsmpTOT I Number of threads for each MPI process
PETOT I Number of MPI processes
COMM I Communicator of MPI
n_neighbor_pe I Number of neighboring MPI processes
neighbor_pe IA Rank ID of neighboring MPI processes [n_neighbor_pe]
import_index IA Index of the number of “external” nodes imported from each
neighboring MPI process [0:n_neighbor_pe]

import_item	IA	List of “external” nodes imported from each neighboring MPI process [import_index(n_neighbor_pe)]
export_index	IA	Index of the number of “boundary” nodes imported from each neighboring MPI process [0:n_neighbor_pe]
export_item	IA	List of “boundary” nodes imported from each neighboring MPI process [export_index(n_neighbor_pe)]
WS , WR	RA	Send/receive buffers [n_node]
WS2, WR2	RA	Send/receive buffers [2*n_node]
WS5, WR5	RA	Send/receive buffers [5*n_node]
global_node_id, global_elem	IA	(not used)

st_ppohFVM_edge_info Derived Type

use_edges	L	T: information of edges used (default), F: information of edges NOT used
n_edge	I	Total number of edges
n_ACTedge	I	(not used)
n_edge_color	I	Total number of colors of edges
area	RA	Surface area of dual-mesh for each edge in the X-Y-Z directions (Fig.5) [4,n_edge]
Vol	RA	Volume of edge-based dual-mesh [n_edge]
edgnod	IA	Node IDs of two vertices of each edge [2,n_edge]
OtoN, NtoO	IA	(not used)
color_index	IA	Index of number of edges for each color and thread [0:n_edge_color*ppohFVM_comm_info%PEsmpTOT]
color_item	IA	(not used)
ACTedge	IA	(not used)

Uses the following modules

(none)

Used by the following subroutines/functions in ppOpen-APPL/FVM

All

m_ppohFVM_SR_r1

This module contains the subroutine used for updating information on a vector at the domain boundaries using MPI (1 DOF/node)

Contains the following subroutines/functions

`ppohFVM_SEND_RECV_r1`

Parameters

`N`	`I`	in	Total number of nodes
`N0`	`I`	in	Total number of internal nodes
`NEIBPETOT`	`I`	in	Number of neighboring MPI processes
`NEIBPE`	`IA`	in	Rank ID of neighboring MPI processes `[NEIBPETOT]`
`IMPORTindex`	`IA`	in	Index of number of “external” nodes imported from each neighboring MPI process `[0: NEIBPETOT]`
`IMPORTitem`	`IA`	in	List of “external” nodes imported from each neighboring MPI process `[IMPORTindex(NEIBPETOT)]`
`EXPORTindex`	`IA`	in	Index of the number of “boundary” nodes imported from each neighboring MPI process `[0: NEIBPETOT]`
`EXPORTitem`	`IA`	in	List of “boundary” nodes imported from each neighboring MPI process `[EXPORTindex(NEIBPETOT)]`
`WS, WR`	`RA`	in /out	Send/receive buffer `[N]`
`X`	`RA`	in /out	Target arrays to be updated `[N]`
`my_rank`	`I`	in	Rank ID of the MPI process
`COMM_FVM`	`I`	in	Communicator of MPI

Uses the following modules

`st_ppohFVM_util`

Used by the following subroutines/functions in ppOpen-APPL/FVM

`ppohFVM_update_1_R`

Calls the following subroutines/functions

`MPI_Isend, MPI_Irecv, MPI_Waitall`

```
      module m_ppohFVM_SR_r1
      use m_ppohFVM_util
      contains

      subroutine  ppohFVM_SEND_RECV_r1                                    &
     &                ( N, NO, NEIBPETOT,NEIBPE, IMPORTindex, IMPORTitem, &
     &                                           EXPORTindex, EXPORTitem, &
     &                  WS, WR, X, my_rank, COMM_FVM)

      implicit REAL*8 (A-H,O-Z)

      integer(kind=ppohFVM_kint )                , intent(in) :: N, NO
      integer(kind=ppohFVM_kint )                , intent(in) :: NEIBPETOT
      integer(kind=ppohFVM_kint ), pointer       , intent(in) :: NEIBPE      (:)
      integer(kind=ppohFVM_kint ), pointer       , intent(in) :: IMPORTindex(:), IMPORTitem(:)
      integer(kind=ppohFVM_kint ), pointer       , intent(in) :: EXPORTindex(:), EXPORTitem(:)
      real   (kind=ppohFVM_kreal), dimension(N)  , intent(inout):: WS
      real   (kind=ppohFVM_kreal), dimension(N)  , intent(inout):: WR
      real   (kind=ppohFVM_kreal), dimension(N)  , intent(inout):: X
      integer                              , intent(in)   :: my_rank, COMM_FVM

      integer(kind=ppohFVM_kint ), dimension(:,:), save, allocatable :: sta1
      integer(kind=ppohFVM_kint ), dimension(:,:), save, allocatable :: sta2
      integer(kind=ppohFVM_kint ), dimension(:  ), save, allocatable :: req1
      integer(kind=ppohFVM_kint ), dimension(:  ), save, allocatable :: req2

      integer(kind=ppohFVM_kint ), save :: NFLAG
      data NFLAG/0/

      if (NFLAG.eq.0) then
        allocate (sta1(MPI_STATUS_SIZE,NEIBPETOT))
        allocate (sta2(MPI_STATUS_SIZE,NEIBPETOT))
        allocate (req1(NEIBPETOT))
        allocate (req2(NEIBPETOT))
        NFLAG= 1
      endif

      do neib= 1, NEIBPETOT
        istart= EXPORTindex(neib-1)
        inum  = EXPORTindex(neib  ) - istart
!$omp parallel do private (k)
        do k= istart+1, istart+inum
           WS(k)= X(EXPORTitem(k))
        enddo
        call MPI_Isend (WS(istart+1), inum, MPI_DOUBLE_PRECISION,        &
     &                  NEIBPE(neib), 0, COMM_FVM,                       &
     &                  req1(neib), ierr)
      enddo

      do neib= 1, NEIBPETOT
        istart= NO + IMPORTindex(neib-1)
        inum  = IMPORTindex(neib) - IMPORTindex(neib-1)
        call MPI_Irecv (X(istart+1), inum, MPI_DOUBLE_PRECISION,         &
     &                  NEIBPE(neib), 0, COMM_FVM,                       &
     &                  req1(neib+NEIBPETOT), ierr)
      enddo

      call MPI_Waitall (NEIBPETOT, req1, sta1, ierr)

      end subroutine   ppohFVM_SEND_RECV_r1
      end module     m_ppohFVM_SR_r1
```

m_ppohFVM_SR_r2

This module contains the subroutine used for updating information on a vector at the domain boundaries using MPI (2 DOF/node)

Contains the following subroutines/functions

`ppohFVM_SEND_RECV_r2`

Parameters

`N`	`I`	in	Total number of nodes
`N0`	`I`	in	Total number of internal nodes
`NEIBPETOT`	`I`	in	Number of neighboring MPI processes
`NEIBPE`	`IA`	in	Rank ID of neighboring MPI processes `[NEIBPETOT]`
`IMPORTindex`	`IA`	in	Index of the number of “external” nodes imported from each neighboring MPI process `[0: NEIBPETOT]`
`IMPORTitem`	`IA`	in	List of “external” nodes imported from each neighboring MPI process `[IMPORTindex(NEIBPETOT)]`
`EXPORTindex`	`IA`	in	Index of the number of “boundary” nodes imported from each neighboring MPI process `[0: NEIBPETOT]`
`EXPORTitem`	`IA`	in	List of “boundary” nodes imported from each neighboring MPI process `[EXPORTindex(NEIBPETOT)]`
`WS, WR`	`RA`	in /out	Send/receive buffer `[2*N]`
`X`	`RA`	in /out	Target arrays to be updated `[N,2]`
`my_rank`	`I`	in	Rank ID of the MPI process
`COMM_FVM`	`I`	in	Communicator of MPI

Uses the following modules

`m_ppohFVM_util`

Used by the following subroutines/functions in ppOpen-APPL/FVM

`ppohFVM_update_2_R`

Calls the following subroutines/functions

`MPI_Isend, MPI_Irecv, MPI_Waitall`

```
      module m_ppohFVM_SR_r2
      use m_ppohFVM_util
      contains

      subroutine  ppohFVM_SEND_RECV_r2                                    &
     &                ( N, NO, NEIBPETOT,NEIBPE, IMPORTindex, IMPORTitem, &
     &                                           EXPORTindex, EXPORTitem, &
     &                  WS, WR, X, my_rank, COMM_FVM)

      implicit REAL*8 (A-H,O-Z)

      integer(kind=ppohFVM_kint )                , intent(in) :: N, NO
      integer(kind=ppohFVM_kint )                , intent(in) :: NEIBPETOT
      integer(kind=ppohFVM_kint ), pointer       , intent(in) :: NEIBPE      (:)
      integer(kind=ppohFVM_kint ), pointer       , intent(in) :: IMPORTindex(:), IMPORTitem(:)
      integer(kind=ppohFVM_kint ), pointer       , intent(in) :: EXPORTindex(:), EXPORTitem(:)
      real   (kind=ppohFVM_kreal), dimension(2*N), intent(inout):: WS
      real   (kind=ppohFVM_kreal), dimension(2*N), intent(inout):: WR
      real   (kind=ppohFVM_kreal), dimension(N,2), intent(inout):: X
      integer                              , intent(in)   :: my_rank, COMM_FVM

      integer(kind=ppohFVM_kint ), dimension(:,:), save, allocatable :: sta1
      integer(kind=ppohFVM_kint ), dimension(:,:), save, allocatable :: sta2
      integer(kind=ppohFVM_kint ), dimension(:  ), save, allocatable :: req1
      integer(kind=ppohFVM_kint ), dimension(:  ), save, allocatable :: req2

      integer(kind=ppohFVM_kint ), save :: NFLAG
      data NFLAG/0/

      if (NFLAG.eq.0) then
        allocate (sta1(MPI_STATUS_SIZE,NEIBPETOT))
        allocate (sta2(MPI_STATUS_SIZE,NEIBPETOT))
        allocate (req1(NEIBPETOT))
        allocate (req2(NEIBPETOT))
        NFLAG= 1
      endif

      do neib= 1, NEIBPETOT
        istart= EXPORTindex(neib-1)
        inum  = EXPORTindex(neib  ) - istart
!$omp parallel do private (k)
        do k= istart+1, istart+inum
           i= EXPORTitem(k)
           WS(2*k-1)= X(i,1)
           WS(2*k  )= X(i,2)
        enddo
        call MPI_Isend (WS(2*istart+1), 2*inum, MPI_DOUBLE_PRECISION,   &
     &                  NEIBPE(neib), 0, COMM_FVM,                      &
     &                  req1(neib), ierr)
      enddo

      do neib= 1, NEIBPETOT
        istart= NO + IMPORTindex(neib-1)
        inum  = IMPORTindex(neib) - IMPORTindex(neib-1)
        call MPI_Irecv (X(2*istart+1), 2*inum, MPI_DOUBLE_PRECISION,    &
     &                  NEIBPE(neib), 0, COMM_FVM,                      &
     &                  req1(neib+NEIBPETOT), ierr)
      enddo

      call MPI_Waitall (NEIBPETOT, req1, sta1, ierr)

      end subroutine   ppohFVM_SEND_RECV_r2
      end module     m_ppohFVM_SR_r2
```

m_ppohFVM_SR_r5

This module contains the subroutine used for updating information on a vector at the domain boundaries using MPI (5 DOF/node)

Contains the following subroutines/functions

`ppohFVM_SEND_RECV_r5`

Parameters

`N`	`I`	in	Total number of nodes
`N0`	`I`	in	Total number of internal nodes
`NEIBPETOT`	`I`	in	Number of neighboring MPI processes
`NEIBPE`	`IA`	in	Rank ID of neighboring MPI processes `[NEIBPETOT]`
`IMPORTindex`	`IA`	in	Index of the number of “external” nodes imported from each neighboring MPI process `[0: NEIBPETOT]`
`IMPORTitem`	`IA`	in	List of “external” nodes imported from each neighboring MPI process `[IMPORTindex(NEIBPETOT)]`
`EXPORTindex`	`IA`	in	Index of the number of “boundary” nodes imported from each neighboring MPI process `[0: NEIBPETOT]`
`EXPORTitem`	`IA`	in	List of “boundary” nodes imported from each neighboring MPI process `[EXPORTindex(NEIBPETOT)]`
`WS, WR`	`RA`	in /out	Send/receive buffer `[5*N]`
`X`	`RA`	in /out	Target arrays to be updated `[N,5]`
`my_rank`	`I`	in	Rank ID of the MPI process
`COMM_FVM`	`I`	in	Communicator of MPI

Uses the following modules

`m_ppohFVM_util`

Used by the following subroutines/functions in ppOpen-APPL/FVM

`ppohFVM_update_5_R`

Calls the following subroutines/functions

`MPI_Isend, MPI_Irecv, MPI_Waitall`

```
      module m_ppohFVM_SR_r5
      use m_ppohFVM_util
      contains

      subroutine  ppohFVM_SEND_RECV_r5                                    &
     &                ( N, NO, NEIBPETOT,NEIBPE, IMPORTindex, IMPORTitem, &
     &                                           EXPORTindex, EXPORTitem, &
     &                  WS, WR, X, my_rank, COMM_FVM)

      implicit REAL*8 (A-H,O-Z)

      integer(kind=ppohFVM_kint )                , intent(in) :: N
      integer(kind=ppohFVM_kint )                , intent(in) :: NEIBPETOT
      integer(kind=ppohFVM_kint ), pointer       , intent(in) :: NEIBPE      (:)
      integer(kind=ppohFVM_kint ), pointer       , intent(in) :: IMPORTindex(:), IMPORTitem(:)
      integer(kind=ppohFVM_kint ), pointer       , intent(in) :: EXPORTindex(:), EXPORTitem(:)
      real   (kind=ppohFVM_kreal), dimension(5*N)  , intent(inout):: WS
      real   (kind=ppohFVM_kreal), dimension(5*N)  , intent(inout):: WR
      real   (kind=ppohFVM_kreal), dimension(N,5)  , intent(inout):: X
      integer                              , intent(in)   :: my_rank, COMM_FVM

      integer(kind=ppohFVM_kint ), dimension(:,:), save, allocatable :: sta1
      integer(kind=ppohFVM_kint ), dimension(:,:), save, allocatable :: sta2
      integer(kind=ppohFVM_kint ), dimension(:  ), save, allocatable :: req1
      integer(kind=ppohFVM_kint ), dimension(:  ), save, allocatable :: req2

      integer(kind=ppohFVM_kint ), save :: NFLAG
      data NFLAG/0/

      if (NFLAG.eq.0) then
        allocate (sta1(MPI_STATUS_SIZE,NEIBPETOT))
        allocate (sta2(MPI_STATUS_SIZE,NEIBPETOT))
        allocate (req1(NEIBPETOT))
        allocate (req2(NEIBPETOT))
        NFLAG= 1
      endif

      do neib= 1, NEIBPETOT
        istart= EXPORTindex(neib-1)
        inum  = EXPORTindex(neib  ) - istart
!$omp parallel do private (k)
        do k= istart+1, istart+inum
           i= EXPORTitem(k)
           WS(5*k-4)= X(i,1); WS(5*k-3)= X(i,2); WS(5*k-2)= X(i,3); WS(5*k-1)= X(i,4);
           WS(5*k  )= X(i,5)
        enddo
        call MPI_Isend (WS(5*istart+1), 5*inum, MPI_DOUBLE_PRECISION,   &
     &                  NEIBPE(neib), 0, COMM_FVM,                      &
     &                  req1(neib), ierr)
      enddo

      do neib= 1, NEIBPETOT
        istart= NO + IMPORTindex(neib-1)
        inum  = IMPORTindex(neib) - IMPORTindex(neib-1)
        call MPI_Irecv (X(5*istart+1), 5*inum, MPI_DOUBLE_PRECISION,    &
     &                  NEIBPE(neib), 0, COMM_FVM,                      &
     &                  req1(neib+NEIBPETOT), ierr)
      enddo

      call MPI_Waitall (NEIBPETOT, req1, sta1, ierr)

      end subroutine   ppohFVM_SEND_RECV_r5
      end module     m_ppohFVM_SR_r5
```

m_ppohFVM_SR_rv2

This module contains the subroutine used for updating information on two vectors at the domain boundaries using MPI (1 DOF/node)

Contains the following subroutines/functions

`ppohFVM_SEND_RECV_rv2`

Parameters

`N`	`I`	in	Total number of nodes
`N0`	`I`	in	Total number of internal nodes
`NEIBPETOT`	`I`	in	Number of neighboring MPI processes
`NEIBPE`	`IA`	in	Rank ID of neighboring MPI processes `[NEIBPETOT]`
`IMPORTindex`	`IA`	in	Index of the number of “external” nodes imported from each neighboring MPI process `[0: NEIBPETOT]`
`IMPORTitem`	`IA`	in	List of “external” nodes imported from each neighboring MPI process `[IMPORTindex(NEIBPETOT)]`
`EXPORTindex`	`IA`	in	Index of the number of “boundary” nodes imported from each neighboring MPI process `[0: NEIBPETOT]`
`EXPORTitem`	`IA`	in	List of “boundary” nodes imported from each neighboring MPI process `[EXPORTindex(NEIBPETOT)]`
`WS, WR`	`RA`	in /out	Send/receive buffer `[2*N]`
`X, Y`	`RA`	in /out	Target arrays to be updated `[N]`
`my_rank`	`I`	in	Rank ID of the MPI process
`COMM_FVM`	`I`	in	Communicator of MPI

Uses the following modules

`m_ppohFVM_util`

Used by the following subroutines/functions in ppOpen-APPL/FVM

`ppohFVM_update_2_RV`

Calls the following subroutines/functions

`MPI_Isend, MPI_Irecv, MPI_Waitall`

```
      module m_ppohFVM_SR_rv2
      use m_ppohFVM_util
      contains

      subroutine  ppohFVM_SEND_RECV_rv2                                  &
     &                ( N, NO, NEIBPETOT,NEIBPE, IMPORTindex, IMPORTitem, &
     &                                           EXPORTindex, EXPORTitem, &
     &                  WS, WR, X, Y, my_rank, COMM_FVM)

      implicit REAL*8 (A-H,O-Z)

      integer(kind=ppohFVM_kint )                , intent(in) :: N
      integer(kind=ppohFVM_kint )                , intent(in) :: NEIBPETOT
      integer(kind=ppohFVM_kint ), pointer       , intent(in) :: NEIBPE      (:)
      integer(kind=ppohFVM_kint ), pointer       , intent(in) :: IMPORTindex(:), IMPORTitem(:)
      integer(kind=ppohFVM_kint ), pointer       , intent(in) :: EXPORTindex(:), EXPORTitem(:)
      real   (kind=ppohFVM_kreal), dimension(2*N)  , intent(inout):: WS, WR
      real   (kind=ppohFVM_kreal), dimension(  N)  , intent(inout):: X, Y
      integer                                    , intent(in)   :: my_rank

      integer(kind=ppohFVM_kint ), dimension(:,:), save, allocatable :: sta1
      integer(kind=ppohFVM_kint ), dimension(:,:), save, allocatable :: sta2
      integer(kind=ppohFVM_kint ), dimension(:  ), save, allocatable :: req1
      integer(kind=ppohFVM_kint ), dimension(:  ), save, allocatable :: req2

      integer(kind=ppohFVM_kint ), save :: NFLAG
      data NFLAG/0/

      if (NFLAG.eq.0) then
        allocate (sta1(MPI_STATUS_SIZE,NEIBPETOT))
        allocate (sta2(MPI_STATUS_SIZE,NEIBPETOT))
        allocate (req1(NEIBPETOT))
        allocate (req2(NEIBPETOT))
        NFLAG= 1
      endif

      do neib= 1, NEIBPETOT
        istart= EXPORTindex(neib-1)
        inum  = EXPORTindex(neib  ) - istart
!$omp parallel do private (k)
        do k= istart+1, istart+inum
           i= EXPORTitem(k)
           WS(2*k-1)= X(i)
           WS(2*k  )= Y(i)
        enddo
        call MPI_Isend (WS(2*istart+1), 2*inum, MPI_DOUBLE_PRECISION,   &
     &                  NEIBPE(neib), 0,  COMM_FVM,                     &
     &                  req1(neib), ierr)
      enddo

      do neib= 1, NEIBPETOT
        istart= IMPORTindex(neib-1)
        inum  = IMPORTindex(neib  ) - istart
        call MPI_Irecv (WR(2*istart+1), 2*inum, MPI_DOUBLE_PRECISION,   &
     &                  NEIBPE(neib), 0, COMM_FVM,                      &
     &                  req2(neib), ierr)
      enddo

      call MPI_Waitall (NEIBPETOT, req2, sta2, ierr)

      do neib= 1, NEIBPETOT
        istart= IMPORTindex(neib-1)
        inum  = IMPORTindex(neib  ) - istart
      do k= istart+1, istart+inum
        i= IMPORTitem(k)
        X(i)= WR(2*k-1)
        Y(i)= WR(2*k  )
      enddo
      enddo

      call MPI_Waitall (NEIBPETOT, req1, sta1, ierr)

      end subroutine   ppohFVM_SEND_RECV_rv2
      end module     m_ppohFVM_SR_r2v
```

3.3 Subroutines

ppohFVM_Init

This subroutine initializes MPI processes.

```
subroutine ppohFVM_Init (st_comm_info)
use m_ppohFVM_util
type (st_ppohFVM_comm_info) :: st_comm_info
```

Parameters

`st_comm_info Derived Type: st_ppohFVM_comm_info`

`st_comm_info%COMM`

`I` out Communicator of MPI (= `MPI_COMM_WORLD`)

`st_file_info%mesh_asci`

`L` out default (`T`: ASCII format) is set

`st_edge_info%use_edges`

`L` out default (`T`: edge information used) is set

Uses the following Modules

`m_ppohFVM_util`

Called by the following subroutines/functions in ppOpen-APPL/FVM

`(none)`

Calls the following subroutines/functions

`MPI_Init, MPI_Comm_size, MPI_Comm_rank, MPI_Comm_dup`

ppohFVM_Finalize

This subroutine terminates MPI processes.

```
subroutine ppohFVM_Finalize (st_comm_info)
use m_ppohFVM_util
type (st_ppohFVM_comm_info) :: st_comm_info
```

Parameters

st_comm_info Derived Type: st_ppohFVM_comm_info

Uses the following Modules

m_ppohFVM_util

Called by the following subroutines/functions in ppOpen-APPL/FVM

(none)

Calls the following subroutines/functions

MPI_Finalize

ppohFVM_Barrier

This subroutine synchronizes MPI processes.

```
subroutine ppohFVM_Barrier
use m_ppohFVM_util
```

Parameters

(none)

Uses the following Modules

m_ppohFVM_util

Called by the following subroutines/functions in ppOpen-APPL/FVM

(none)

Calls the following subroutines/functions

MPI_Barrier

ppohFVM_update_1_R

This subroutine updates the information on a vector at the domain boundaries using MPI (1 DOF/node).

```
subroutine ppohFVM_update_1_R (st_comm_info, VAL, n, n0)

use m_ppohFVM_SR_r1

integer :: n, ierr
real (kind=ppohFVM_kreal), dimension(n) :: VAL
type (st_ppohFVM_comm_info)             :: st_comm_info
```

Parameters

`st_comm_info` Derived Type: `st_ppohFVM_comm_info`

`n`	I	in	Size of array, total number of nodes (internal + external)
`n0`	I	in	Total number of internal nodes
`VAL`	RA	in/out	Array to be updated `[n]`

Uses the following Modules

`m_ppohFVM_SR_r1`

Called by the following subroutines/functions in ppOpen-APPL/FVM

`ppohFVM_edge_metrics`

Calls the following subroutines/functions

`ppohFVM_SEND_RECV_r1`

ppohFVM_update_2_R

This subroutine updates information on a vector at the domain boundaries using MPI (2 DOF/node).

```
subroutine ppohFVM_update_2_R (st_comm_info, VAL, n, n0)

use m_ppohFVM_SR_r2

integer :: n, ierr
real (kind=ppohFVM_kreal), dimension(n,2) :: VAL
type (st_ppohFVM_comm_info)               :: st_comm_info
```

Parameters

st_comm_info	Derived Type: st_ppohFVM_comm_info		
n	I	in	Size of array, total number of nodes (internal + external)
n0	I	in	Total number of internal nodes
VAL	RA	in/out	Array to be updated [n,2]

Uses the following Modules

m_ppohFVM_SR_r2

Called by the following subroutines/functions in ppOpen-APPL/FVM

ppohFVM_edge_metrics

Calls the following subroutines/functions

ppohFVM_SEND_RECV_r2

ppohFVM_update_5_R

This subroutine updates the information on a vector at the domain boundaries using MPI (5 DOF/node).

```
subroutine ppohFVM_update_5_R (st_comm_info, VAL, n, n0)

use m_ppohFVM_SR_r5

integer :: n, ierr
real (kind=ppohFVM_kreal), dimension(n,5) :: VAL
type (st_ppohFVM_comm_info)               :: st_comm_info
```

Parameters

st_comm_info	Derived Type: st_ppohFVM_comm_info		
n	I	in	Size of array, total number of nodes (internal + external)
n0	I	in	Total number of internal nodes
VAL	RA	in/out	Array to be updated [n,5]

Uses the following Modules

m_ppohFVM_SR_r5

Called by the following subroutines/functions in ppOpen-APPL/FVM

(none)

Calls the following subroutines/functions

ppohFVM_SEND_RECV_r5

ppohFVM_update_2_RV

This subroutine updates information on two vectors at the domain boundaries using MPI (1 DOF/node)

```
subroutine ppohFVM_update_2_RV (st_comm_info, VAL1, VAL2, n, n0)

use m_ppohFVM_SR_rv2

integer :: n, ierr
real (kind=ppohFVM_kreal), dimension(n) :: VAL1, VAL2
type (st_ppohFVM_comm_info)             :: st_comm_info
```

Parameters

st_comm_info	Derived Type: st_ppohFVM_comm_info		
n	I	in	Size of array, total number of nodes (internal + external)
n0	I	in	Total number of internal nodes
VAL1, VAL2	RA	in/out	Arrays to be updated [n]

Uses the following Modules

m_ppohFVM_SR_rv2

Called by the following subroutines/functions in ppOpen-APPL/FVM

(none)

Calls the following subroutines/functions

ppohFVM_SEND_RECV_rv2

ppohFVM_Allreduce_R

MPI_Allreduce for a real scalar parameter

```
      subroutine ppohFVM_Allreduce_R ( st_comm_info, VAL, ntag )

      use m_ppohFVM_util

      integer :: ntag, ierr
      real (kind=ppohFVM_kreal)   :: VAL, VALM
      type (st_ppohFVM_comm_info) :: st_comm_info
```

Parameters

st_comm_info	Derived Type: st_ppohFVM_comm_info		
VAL	R	in/out	Scalar parameter to be reduced
ntag	I	in	Reduce operation: sum, max, min

Uses the following Modules

m_ppohFVM_util

Called by the following subroutines/functions in ppOpen-APPL/FVM

(none)

Calls the following subroutines/functions

MPI_Allreduce

ppohFVM_Allreduce_I

MPI_Allreduce for an integer scalar parameter

```
subroutine ppohFVM_Allreduce_I ( st_comm_info, VAL, ntag )

use m_ppohFVM_util

integer :: ntag, ierr
integer :: VAL, VALM
type (st_ppohFVM_comm_info) :: st_comm_info
```

Parameters

st_comm_info	Derived Type: st_ppohFVM_comm_info		
VAL	I	in/out	Scalar parameter to be reduced
ntag	I	in	Reduce operation: sum, max, min

Uses the following Modules

m_ppohFVM_util

Called by the following subroutines/functions in ppOpen-APPL/FVM

(none)

Calls the following subroutines/functions

MPI_Allreduce

ppohFVM_Allreduce_RV

MPI_Allreduce for a real array parameter

```
      subroutine ppohFVM_Allreduce_R ( st_comm_info, VAL, n, ntag )

      use m_ppohFVM_util

      integer :: ntag, ierr
      real (kind=ppohFVM_kreal)    :: VAL, VALM
      type (st_ppohFVM_comm_info) :: st_comm_info
```

Parameters

st_comm_info Derived Type: st_ppohFVM_comm_info

VAL	R	in/out	Array to be reduced [n]
n	I	in	Size of array
ntag	I	in	Reduce operation: sum, max, min

Uses the following Modules

m_ppohFVM_util

Called by the following subroutines/functions in ppOpen-APPL/FVM

(none)

Calls the following subroutines/functions

MPI_Allreduce

ppohFVM_Bcast_R

MPI_Bcast for a real scalar parameter

```
subroutine ppohFVM_Bcast_R ( st_comm_info, VAL, nbase )

use m_ppohFVM_util

integer :: nbase
real (kind=ppohFVM_kreal)   :: VAL
type (st_ppohFVM_comm_info) :: st_comm_info
```

Parameters

st_comm_info	Derived Type: st_ppohFVM_comm_info		
VAL	R	in	Scalar parameter to be sent
nbase	I	in	Rank of root process

Uses the following Modules

m_ppohFVM_util

Called by the following subroutines/functions in ppOpen-APPL/FVM

(none)

Calls the following subroutines/functions

MPI_Bcast

ppohFVM_Bcast_I

MPI_Bcast for an integer scalar parameter

```
subroutine ppohFVM_Bcast_I ( st_comm_info, VAL, nbase )

use m_ppohFVM_util

integer :: nbase
integer :: VAL
type (st_ppohFVM_comm_info) :: st_comm_info
```

Parameters

st_comm_info Derived Type: st_ppohFVM_comm_info

VAL	I	in	Scalar parameter to be sent
nbase	I	in	Rank of root process

Uses the following Modules

m_ppohFVM_util

Called by the following subroutines/functions in ppOpen-APPL/FVM

(none)

Calls the following subroutines/functions

MPI_Bcast

ppohFVM_Bcast_L

MPI_Bcast for a logical scalar parameter

```
subroutine ppohFVM_Bcast_L ( st_comm_info, VAL, nbase )

use m_ppohFVM_util

integer :: nbase
logical :: VAL
type (st_ppohFVM_comm_info) :: st_comm_info
```

Parameters

st_comm_info Derived Type: st_ppohFVM_comm_info

VAL	L	in	Scalar parameter to be sent
nbase	I	in	Rank of root process

Uses the following Modules

m_ppohFVM_util

Called by the following subroutines/functions in ppOpen-APPL/FVM

(none)

Calls the following subroutines/functions

MPI_Bcast

ppohFVM_Bcast_C

MPI_Bcast for a character scalar parameter

```
subroutine ppohFVM_Bcast_C ( st_comm_info, VAL, nn, nbase )

use m_ppohFVM_util

integer :: nbase, nn
character(len=nn) :: VAL
type (st_ppohFVM_comm_info) :: st_comm_info
```

Parameters

st_comm_info	Derived Type: st_ppohFVM_comm_info		
VAL	C	in	Scalar parameter to be sent [length=nn]
nn	I	in	Length of VAL
nbase	I	in	Rank of root process

Uses the following Modules

m_ppohFVM_util

Called by the following subroutines/functions in ppOpen-APPL/FVM

ppohFVM_dist_file

Calls the following subroutines/functions

MPI_Bcast

ppohFVM_dist_file

This subroutine provides the names of distributed files.

```
subroutine ppohFVM_dist_file (file_info, st_comm_info, HEADER, len, root, my_rank, id)

use m_ppohFVM_util

integer :: my_rank, len, root, id
character (len=80) ::  HEADER

type (ppohFVM_file_info)  ::  file_info
type (st_ppohFVM_comm_info) :: st_comm_info
```

Parameters

`st_file_info` Derived Type: `st_ppohFVM_file_info`

`st_comm_info` Derived Type: `st_ppohFVM_comm_info`

`HEADER`	C	in	Header of file name `[length=80]`
`len`	I	in	Length of `HEADER`
`root`	I	in	Rank of root process
`my_rank`	I	in	MPI rank of the process
`id`	I	in	Address of the file in `file_info%file` `[length=80]` `file_info%file(id)` is defined in this subroutine

Uses the following Modules

`m_ppohFVM_util`

Called by the following subroutines/functions in ppOpen-APPL/FVM

`(none)`

Calls the following subroutines/functions

`ppohFVM_Bcast_C, ppohFVM_file_name`

ppohFVM_error_exit

This subroutine terminates the application if error conditions are detected during computation.

```
subroutine ppohFVM_error_exit (MODE)
use m_ppohFVM_util
```

Parameters

MODE	I	in	Error ID

Uses the following Modules

m_ppohFVM_util

Called by the following subroutines/functions in ppOpen-APPL/FVM

ppohFVM_input_grid, ppoh_FVM_prism_metrics, ppoh_FVM_QUAD_SURFACE

Calls the following subroutines/functions

MPI_Abort, MPI_Finalize

ppohFVM_pre

This subroutine calls three main subroutines for reading files, searching active meshes, and calculating edge metrics.

```
subroutine ppohFVM_pre (st_file_info, st_local_mesh, st_grp_data, st_comm_info, st_edge_info)

use  m_ppohFVM_util

type (st_ppohFVM_file_info)  :: st_file_info
type (st_ppohFVM_local_mesh) :: st_local_mesh
type (st_ppohFVM_grp_data)   :: st_grp_data
type (st_ppohFVM_comm_info)  :: st_comm_info
type (st_ppohFVM_edge_info)  :: st_edge_info
```

Parameters

st_file_info	Derived Type: st_ppohFVM_file_info
st_local_mesh	Derived Type: st_ppohFVM_local_mesh
st_grp_data	Derived Type: st_ppohFVM_grp_data
st_comm_info	Derived Type: st_ppohFVM_comm_info
st_edge_info	Derived Type: st_ppohFVM_edge_info

Uses the following Modules

m_ppohFVM_util

Called by the following subroutines/functions in ppOpen-APPL/FVM

(none)

Calls the following subroutines/functions

ppohFVM_input_grid, ppohFVM_active, ppohFVM_edge_metric, ppohFVM_input_grod_b

ppohFVM_input_grid

This subroutine reads distributed mesh files containing communication information (ASCII).

```
      subroutine ppohFVM_INPUT_GRID (st_local_mesh, st_grp_data, st_comm_info, st_file_info)

      use m_ppohFVM_util

      type (st_ppohFVM_st_local_mesh) :: st_local_mesh
      type (st_ppohFVM_st_grp_data)   :: st_grp_data
      type (st_ppohFVM_st_comm_info)  :: st_comm_info
      type (st_ppohFVM_st_file_info)  :: st_file_info

      character*1 LINE(132)

      IUNIT= 11
      open (IUNIT,file= st_st_file_info%file(1), status='unknown', form='formatted')
!C
!C-- Parallel info.

        read (IUNIT,*) st_local_mesh%CoarseGridLevels
        read (IUNIT,*) st_local_mesh%HOWmanyADAPTATIONs
        read (IUNIT,*) myID
        read (IUNIT,*) st_comm_info%n_neighbor_pe

     allocate (st_comm_info%neighbor_pe                                &
    &                    (st_comm_info%n_neighbor_pe))
        read (IUNIT,*) (st_comm_info%neighbor_pe(k),                   &
    &               k= 1, st_comm_info%n_neighbor_pe)

        read (IUNIT,*) st_local_mesh%n_material

      NEIBPETOT= st_comm_info%n_neighbor_pe

!C
!C-- NODE info.
        read (IUNIT,*) st_local_mesh%n_node, st_local_mesh%n_internal

        NODarray= st_local_mesh%n_node

        allocate (st_local_mesh%node(3,NODarray))
        allocate (st_local_mesh%WhenIwasRefined_node(NODarray))
        allocate (st_local_mesh%node_id(2,NODarray))

        do i= 1, st_local_mesh%n_node
          read (IUNIT,*)                                               &
    &          st_local_mesh%node_id(1,i),st_local_mesh%node_id(2,i), &
    &          st_local_mesh%WhenIwasRefined_node(i),                 &
    &         (st_local_mesh%node(k,i),k =1,3)
        enddo

!C
!C-- ELEMENT Info.
        read (IUNIT,*) st_local_mesh%n_elem, st_local_mesh%ne_internal

        CELarray= st_local_mesh%n_elem

        allocate (st_local_mesh%elem_id  (CELarray,2))

        if (st_local_mesh%n_material.ne.0) then
          allocate (st_local_mesh%material(CELarray,st_local_mesh%n_material))
        endif
        allocate (st_local_mesh%mat_id   (CELarray  ))
        allocate (st_local_mesh%elem_type(CELarray  ))
        allocate (st_local_mesh%ptr_elem   (CELarray*8))
        allocate (st_local_mesh%index_elem    (0:CELarray))
        allocate (st_local_mesh%index_children(0:CELarray))

        allocate (st_local_mesh%WhenIwasRefined_elem(CELarray))
        allocate (st_local_mesh%ne_internal_list(CELarray  ))

        allocate (st_local_mesh%adaptation_parent_type (CELarray))

        allocate (st_local_mesh%adaptation_type (CELarray))
        allocate (st_local_mesh%adaptation_level(CELarray))
```

```
        allocate (st_local_mesh%adaptation_parent  (CELarray,2))
        allocate (st_local_mesh%adaptation_children(CELarray*8,2))

        read (IUNIT,'(10i10)') (st_local_mesh%elem_type(i),            &
     &                i= 1, st_local_mesh%n_elem)


        st_local_mesh%index_elem    = 0
        st_local_mesh%index_children= 0
        do icel= 1, st_local_mesh%n_elem
          ityp= st_local_mesh%elem_type(icel)

          if (ityp.eq.341) then
            icon= 4
            ichi= 8
          endif

          if (ityp.eq.351) then
            icon= 6
            ichi= 4
          endif

          if (ityp.eq.361) then
            icon= 8
            ichi= 8
          endif

          st_local_mesh%index_elem(icel)=                              &
     &          st_local_mesh%index_elem(icel-1) + icon

          st_local_mesh%index_children(icel)=                          &
     &          st_local_mesh%index_children(icel-1) + ichi
        enddo

        if (st_local_mesh%n_material.eq.0) then
          do icel= 1, st_local_mesh%n_elem
            iS  = st_local_mesh%index_elem(icel-1)
            icon= st_local_mesh%index_elem(icel) -                       &
     &            st_local_mesh%index_elem(icel-1)
            read (IUNIT,*)                                               &
     &            st_local_mesh%elem_id(icel,1),                         &
     &            st_local_mesh%elem_id(icel,2),                         &
     &            st_local_mesh%WhenIwasRefined_elem(icel),              &
     &            st_local_mesh%mat_id(icel),                            &
     &           (st_local_mesh%ptr_elem(k),k=iS+1, iS+icon)
            do kk= iS+1, iS+icon
              in= st_local_mesh%ptr_elem(kk)
              if (in.le.0) call ppohFVM_error_exit(1005)
            enddo
          enddo
         else
          do icel= 1, st_local_mesh%n_elem
            iS  = st_local_mesh%index_elem(icel-1)
            icon= st_local_mesh%index_elem(icel) -                       &
     &            st_local_mesh%index_elem(icel-1)
            read (IUNIT,*)                                               &
     &            st_local_mesh%elem_id(icel,1),                         &
     &            st_local_mesh%elem_id(icel,2),                         &
     &            st_local_mesh%WhenIwasRefined_elem(icel),              &
     &            st_local_mesh%mat_id(icel),                            &
     &           (st_local_mesh%ptr_elem(k),k=iS+1, iS+icon),            &
     &           (st_local_mesh%material(icel,kk),kk= 1, st_local_mesh%n_material)
```

* If st_local_mesh%n_material is not 0, material components are defined at each element.

```
            do kk= iS+1, iS+icon
              in= st_local_mesh%ptr_elem(kk)
              if (in.le.0) call ppohFVM_error_exit(1005)
            enddo
          enddo
        endif
!C
!C-- ADAPTATION info.
        do icel= 1, st_local_mesh%n_elem

          iS  = st_local_mesh%index_children(icel-1)
          icon= st_local_mesh%index_children(icel) -                     &
     &          st_local_mesh%index_children(icel-1)

          read (IUNIT,*)                                                 &
     &          st_local_mesh%adaptation_type  (icel),                   &
     &          st_local_mesh%adaptation_level (icel),                   &
```

```
        read (IUNIT,*)                                                  &
     &       (st_local_mesh%ne_internal_list(i),                        &
     &        i= 1,st_local_mesh%ne_internal)


!C
!C-- IMPORT/EXPORT
      allocate (comm_info%import_index(0:NEIBPETOT))
      allocate (comm_info%export_index(0:NEIBPETOT))

      comm_info%import_index(0)= 0
      comm_info%export_index(0)= 0

        read (IUNIT,*) (comm_info%import_index(k), k= 1, NEIBPETOT)
      nn= comm_info%import_index(NEIBPETOT)
      allocate (comm_info%import_item(nn))

      if (NEIBPETOT.ne.0) then
        read (IUNIT,*) (comm_info%import_item(i), i= 1, nn)
      endif

      read (IUNIT,*) (comm_info%export_index(k), k= 1, NEIBPETOT)
      nn= comm_info%export_index(NEIBPETOT)
      allocate (comm_info%export_item(nn))

      if (NEIBPETOT.ne.0) then
        read (IUNIT,*) (comm_info%export_item(i), i= 1, nn)
      endif

!C
!C-- BOUNDARY Info. : NODE group
        read  (IUNIT,*) st_grp_data%node_grp%n_enum_grp
          N1= st_grp_data%node_grp%n_enum_grp
          allocate (st_grp_data%node_grp%enum_grp_name(N1))
          allocate (st_grp_data%node_grp%enum_grp_index(0:N1))
          st_grp_data%node_grp%enum_grp_index(0)= 0

        if (st_grp_data%node_grp%n_enum_grp.ne.0) then
          read  (IUNIT,*)                                               &
     &          (st_grp_data%node_grp%enum_grp_index(ig),               &
     &           ig= 1,st_grp_data%node_grp%n_enum_grp)
          N2= st_grp_data%node_grp%enum_grp_index(N1)
          allocate (st_grp_data%node_grp%enum_grp_node(N2))


          do ig= 1, st_grp_data%node_grp%n_enum_grp
            read  (IUNIT,*)                                             &
     &             st_grp_data%node_grp%enum_grp_name(ig)

            nn= st_grp_data%node_grp%enum_grp_index(ig) -               &
     &          st_grp_data%node_grp%enum_grp_index(ig-1)
            if (nn.ne.0) then
            read (IUNIT,*)                                              &
     &            (st_grp_data%node_grp%enum_grp_node(is),              &
     &             is= st_grp_data%node_grp%enum_grp_index(ig-1)+1,     &
     &                 st_grp_data%node_grp%enum_grp_index(ig))
            endif
          enddo
        endif

!C
!C-- BOUNDARY Info. : ELEMENT group
        read  (IUNIT,*) st_grp_data%elem_grp%n_enum_grp
          N1= st_grp_data%elem_grp%n_enum_grp
          allocate (st_grp_data%elem_grp%enum_grp_name(N1))
          allocate (st_grp_data%elem_grp%enum_grp_index(0:N1))
          st_grp_data%elem_grp%enum_grp_index(0)= 0

        if (st_grp_data%elem_grp%n_enum_grp.ne.0) then
          read  (IUNIT,*)                                                 &
     &          (st_grp_data%elem_grp%enum_grp_index(ig),                 &
     &           ig= 1,st_grp_data%elem_grp%n_enum_grp)
```

```
          N2= st_grp_data%elem_grp%enum_grp_index(N1)
          allocate (st_grp_data%elem_grp%enum_grp_node(N2))

          do ig= 1, st_grp_data%elem_grp%n_enum_grp
            read  (IUNIT,*)                                              &
     &             st_grp_data%elem_grp%enum_grp_name(ig)

              nn= st_grp_data%elem_grp%enum_grp_index(ig) -              &
     &            st_grp_data%elem_grp%enum_grp_index(ig-1)
              if (nn.ne.0) then
              read (IUNIT,*)                                             &
     &             (st_grp_data%elem_grp%enum_grp_node(is),              &
     &              is= st_grp_data%elem_grp%enum_grp_index(ig-1)+1,     &
     &                  st_grp_data%elem_grp%enum_grp_index(ig))
              endif
          enddo
        endif
!C
!C-- BOUNDARY Info. : SURFACE group
        read  (IUNIT,*) st_grp_data%surf_grp%n_surf_grp
          N1= st_grp_data%surf_grp%n_surf_grp
          allocate (st_grp_data%surf_grp%surf_grp_name(N1))
          allocate (st_grp_data%surf_grp%surf_grp_index(0:N1))
          st_grp_data%surf_grp%surf_grp_index(0)= 0

        if (st_grp_data%surf_grp%n_surf_grp.ne.0) then
          read  (IUNIT,*)                                              &
     &          (st_grp_data%surf_grp%surf_grp_index(ig),              &
     &           ig= 1,st_grp_data%surf_grp%n_surf_grp)
          N2= st_grp_data%surf_grp%surf_grp_index(N1)
          allocate (st_grp_data%surf_grp%surf_grp_node(N2,2))

          do ig= 1, st_grp_data%surf_grp%n_surf_grp
            read  (IUNIT,*)                                              &
     &             st_grp_data%surf_grp%surf_grp_name(ig)
            nn= st_grp_data%surf_grp%surf_grp_index(ig) -              &
     &          st_grp_data%surf_grp%surf_grp_index(ig-1)
            if (nn.ne.0) then
            read (IUNIT,*)                                             &
     &           (st_grp_data%surf_grp%surf_grp_node(is,1),            &
     &            is= st_grp_data%surf_grp%surf_grp_index(ig-1)+1,     &
     &                st_grp_data%surf_grp%surf_grp_index(ig))
            read (IUNIT,*)                                             &
     &           (st_grp_data%surf_grp%surf_grp_node(is,2),            &
     &            is= st_grp_data%surf_grp%surf_grp_index(ig-1)+1,     &
     &                st_grp_data%surf_grp%surf_grp_index(ig))
            endif
          enddo
        endif
      close (IUNIT)
```

Parameters

st_file_info	Derived Type:	st_ppohFVM_file_info
st_local_mesh	Derived Type:	st_ppohFVM_local_mesh
st_grp_data	Derived Type:	st_ppohFVM_grp_data
st_comm_info	Derived Type:	st_ppohFVM_comm_info

Uses the following Modules

m_ppohFVM_util

Called by the following subroutines/functions in ppOpen-APPL/FVM

ppohFVM_pre

Calls the following subroutines/functions

ppohFVM_Allreduce_I, ppohFVM_error_exit

ppohFVM_input_grid_b

This subroutine reads distributed mesh files containing communication information (Binary)

```
      subroutine ppohFVM_INPUT_GRID_B (st_local_mesh, st_grp_data, st_comm_info, st_file_info)

      use m_ppohFVM_util

      type (st_ppohFVM_local_mesh) :: st_local_mesh
      type (st_ppohFVM_grp_data)   :: st_grp_data
      type (st_ppohFVM_comm_info)  :: st_comm_info
      type (st_ppohFVM_file_info)  :: st_file_info

      character*1 LINE(132)

      IUNIT= 11
      open (IUNIT,file= st_file_info%file(1), status='unknown', form='unformatted')
```

Parameters

st_file_info	Derived Type:	st_ppohFVM_file_info
st_local_mesh	Derived Type:	st_ppohFVM_local_mesh
st_grp_data	Derived Type:	st_ppohFVM_grp_data
st_comm_info	Derived Type:	st_ppohFVM_comm_info

Uses the following Modules

m_ppohFVM_util

Called by the following subroutines/functions in ppOpen-APPL/FVM

ppohFVM_pre

Calls the following subroutines/functions

ppohFVM_Allreduce_I, ppohFVM_error_exit

ppohFVM_active

This subroutine finds active prisms and tetrahedra.

```
subroutine ppohFVM_active (st_local_mesh)
use m_ppohFVM_util
type (st_ppohFVM_local_mesh) :: st_local_mesh
```

Parameters

`st_local_mesh` Derived Type: `st_ppohFVM_local_mesh`

Uses the following Modules

`m_ppohFVM_util`

Called by the following subroutines/functions in ppOpen-APPL/FVM

`ppohFVM_pre`

Calls the following subroutines/functions

`(none)`

ppohFVM_edge_metrics

This subroutine calculates edge metrics.

```
subroutine ppohFVM_edge_metrics (st_local_mesh, st_comm_info, st_edge_info)

use m_ppohFVM_util

type (st_ppohFVM_local_mesh) :: st_local_mesh
type (st_ppohFVM_comm_info)  :: st_comm_info
type (st_ppohFVM_edge_info)  :: st_edge_info
```

Parameters

st_local_mesh	Derived Type: st_ppohFVM_local_mesh		
st_comm_info	Derived Type: st_ppohFVM_comm_info		
st_edge_info	Derived Type: st_ppohFVM_edge_info		
st_local_mesh%voln	RA	out	Volume of dual-mesh [st_local_mesh%local_mesh%n_node]
st_local_mesh%sarea	RA	out	Surface area of dual-mesh in the X-Y-Z directions [st_local_mesh%local_mesh%n_node]

Uses the following Modules

m_ppohFVM_util

Called by the following subroutines/functions in ppOpen-APPL/FVM

ppohFVM_pre

Calls the following subroutines/functions

ppohFVM_edge_metrics_init, ppohFVM_edge_color,
ppohFVM_prism_351_metrics, ppohFVM_edge_metrics_prism_351,
ppohFVM_hexa_361_metrics, ppohFVM_edge_metrics_hexa_361,
ppohFVM_edge_metrics_tetra_341, ppohFVM_edge_cre

ppohFVM_edge_metrics_init

This subroutine creates information on all edges from element-node information.

```
subroutine ppohFVM_edge_metrics_init (st_local_mesh, st_comm_info, st_edge_info)

use m_ppohFVM_util

type (st_ppohFVM_local_mesh) :: st_local_mesh
type (st_ppohFVM_comm_info)  :: st_comm_info
type (st_ppohFVM_edge_info)  :: st_edge_info
```

Parameters

st_local_mesh	Derived Type: st_ppohFVM_local_mesh		
st_comm_info	Derived Type: st_ppohFVM_comm_info		
st_edge_info	Derived Type: st_ppohFVM_edge_info		
st_edge_info%n_edge	I	out	Total number of edges
st_dge_info%edgnod	IA	out	edge-node information [st_edge_info%n_edge,2]

Uses the following Modules

m_ppohFVM_util

Called by the following subroutines/functions in ppOpen-APPL/FVM

ppohFVM_edge_metrics

Calls the following subroutines/functions

ppohFVM_edge_cre

ppohFVM_edge_cre

This subroutine creates information on all edges from element-node information.

```
subroutine ppohFVM_edge_cre (st_edge_info,nod1,nod2,iedge,NFLAG,ICELTOT,NODTOT,IEDGTOT)
use m_ppohFVM_util
type (st_ppohFVM_edge_info)  ::  st_edge_info
```

Parameters

st_edge_info	Derived Type: st_ppohFVM_edge_info		
nod1, nod2	I	in	1st & 2nd node of the edge
iedge	I	out	ID of the edge
NFLAG	I	in	Flag for calling status =0: creating edges, =1: just checking info.
ICELTOT	I	in	Total number of meshes (prisms, tetrahedra)
NODTOT	I	in	Total number of nodes
IEDGTOT	I	in	Total number of edges
edge_info%edgnod	IA	out	edge-node information [2,edge_info%n_edge]

Uses the following Modules

m_ppohFVM_util

Called by the following subroutines/functions in ppOpen-APPL/FVM

ppohFVM_edge_metrics_init, ppohFVM_edge_color,
ppohFVM_edge_metrics_prism_351, ppohFVM_edge_metrics_tetra_341,
ppohFVM_edge_metrics_hexa_361

Calls the following subroutines/functions

(none)

ppohFVM_edge_color

This subroutine colors and reorders edges for parallel computation.

```
      subroutine ppohFVM_edge_color (st_local_mesh, st_comm_info, st_edge_info)

      use m_ppohFVM_util

      type (st_ppohFVM_local_mesh) :: st_local_mesh
      type (st_ppohFVM_comm_info)  :: st_comm_info
      type (st_ppohFVM_edge_info)  :: st_edge_info
```

Parameters

st_local_mesh	Derived Type: st_ppohFVM_local_mesh		
st_comm_info	Derived Type: st_ppohFVM_comm_info		
st_edge_info	Derived Type: st_ppohFVM_edge_info		
st_edge_info%n_edge_color	I	out	Total number of colors
st_edge_info%color_index	IA	out	Index of edges for each color and thread [0:st_edge_info%n_edge*st_comm_info%PEsmpTOT]

Uses the following Modules

m_ppohFVM_util

Called by the following subroutines/functions in ppOpen-APPL/FVM

ppohFVM_edge_metrics

Calls the following subroutines/functions

ppohFVM_edge_cre

ppohFVM_prism_351_metrics

This subroutine calculates the volume of prisms (type: 351)

```
subroutine ppohFVM_prism_351_metrics (st_local_mesh, st_edge_info)

use m_ppohFVM_util

type (st_ppohFVM_local_mesh) :: st_local_mesh
type (st_ppohFVM_edge_info)  :: st_edge_info
```

Parameters

st_local_mesh	Derived Type: st_ppohFVM_local_mesh		
st_edge_info	Derived Type: st_ppohFVM_edge_info		
st_local_mesh%volc	RA	out	Volume of prisms [st_local_mesh%n_elem]

Uses the following Modules

m_ppohFVM_util

Called by the following subroutines/functions in ppOpen-APPL/FVM

ppohFVM_edge_metrics

Calls the following subroutines/functions

ppohFVM_QUAD_SURFACE, ppohFVM_error_exit

ppohFVM_edge_metrics_prism_351

This subroutine calculates edge-based metrics related to prisms (type: 351)

```
subroutine ppohFVM_edge_metrics_prism_351 (st_local_mesh, st_comm_info, st_edge_info)

use m_ppohFVM_util

type (st_ppohFVM_local_mesh) :: st_local_mesh
type (st_ppohFVM_comm_info)  :: st_comm_info
type (st_ppohFVM_edge_info)  :: st_edge_info
```

Parameters

st_local_mesh	Derived Type: st_ppohFVM_local_mesh		
st_comm_info	Derived Type: st_ppohFVM_comm_info		
st_edge_info	Derived Type: st_ppohFVM_edge_info		
st_edge_info%area	RA	out	Surface area of the dual-mesh for each edge in the X-Y-Z directions [4,st_edge_info%n_edge]
st_edge_info%vol	RA	out	Volume of edge-based dual-mesh [st_edge_info%n_edge]

Uses the following Modules

m_ppohFVM_util

Called by the following subroutines/functions in ppOpen-APPL/FVM

ppohFVM_edge_metrics

Calls the following subroutines/functions

ppohFVM_edge_lat, ppohFVM_edge_fac,
ppohFVM_edge_cre, ppohFVM_error_exit

ppohFVM_hexa_361_metrics

This subroutine calculates the volume of hexahedra (type: 361)

```
subroutine ppohFVM_hexa_361_metrics (st_local_mesh, st_edge_info)

use m_ppohFVM_util

type (st_ppohFVM_local_mesh) :: st_local_mesh
type (st_ppohFVM_edge_info)  :: st_edge_info
```

Parameters

st_local_mesh	Derived Type: st_ppohFVM_local_mesh		
st_edge_info	Derived Type: st_ppohFVM_edge_info		
st_local_mesh%volc	RA	out	Volume of prisms [st_local_mesh%n_elem]

Uses the following Modules

m_ppohFVM_util

Called by the following subroutines/functions in ppOpen-APPL/FVM

ppohFVM_edge_metrics

Calls the following subroutines/functions

ppohFVM_QUAD_SURFACE, ppohFVM_error_exit

ppohFVM_edge_metrics_hexa_361

This subroutine calculates edge-based metrics related to hexahedra (type: 361)

```
subroutine ppohFVM_edge_metrics_hexa_361 (st_local_mesh, st_comm_info, st_edge_info)

use m_ppohFVM_util

type (st_ppohFVM_local_mesh) :: st_local_mesh
type (st_ppohFVM_comm_info)  :: st_comm_info
type (st_ppohFVM_edge_info)  :: st_edge_info
```

Parameters

st_local_mesh	Derived Type: st_ppohFVM_local_mesh		
st_comm_info	Derived Type: st_ppohFVM_comm_info		
st_edge_info	Derived Type: st_ppohFVM_edge_info		
st_edge_info%area	RA	out	Surface area of the dual-mesh for each edge in the X-Y-Z directions [4,st_edge_info%n_edge]
st_edge_info%vol	RA	out	Volume of edge-based dual-mesh [st_edge_info%n_edge]

Uses the following Modules

m_ppohFVM_util

Called by the following subroutines/functions in ppOpen-APPL/FVM

ppohFVM_edge_metrics

Calls the following subroutines/functions

ppohFVM_edge_lat4, ppohFVM_edge_fac4,
ppohFVM_edge_cre, ppohFVM_error_exit

ppohFVM_edge_metrics_tetra_341

This subroutine calculates edge-based metrics related to tetrahedra (type: 341)

```
subroutine ppohFVM_edge_metrics_tetra_341 (st_local_mesh, st_comm_info, st_edge_info)

use m_ppohFVM_util

type (st_ppohFVM_local_mesh) :: st_local_mesh
type (st_ppohFVM_comm_info)  :: st_comm_info
type (st_ppohFVM_edge_info)  :: st_edge_info
```

Parameters

st_local_mesh	Derived Type: ppohFVM_local_mesh		
st_comm_info	Derived Type: ppohFVM_comm_info		
st_edge_info	Derived Type: ppohFVM_edge_info		
st_local_mesh%volc	RA	out	Volume of tetrahedra [st_local_mesh%n_elem]
st_edge_info%area	RA	out	Surface area of the dual-mesh for each edge in the X-Y-Z directions [4,st_edge_info%n_edge]
edge_info%vol	RA	out	Volume of edge-based dual-mesh [edge_info%n_edge]

Uses the following Modules

m_ppohFVM_util

Called by the following subroutines/functions in ppOpen-APPL/FVM

ppohFVM_edge_metrics

Calls the following subroutines/functions

ppohFVM_edge_tet_341,
ppohFVM_edge_cre, ppohFVM_error_exit

4. hybNS

4.1 Overview

The hybNS code for a parallel 3D compressible Navier-Stokes simulation is developed on ppOpen-APPL/FVM. An edge-based finite-volume method with unstructured prismatic/tetrahedral hybrid meshes suitable for complicated geometry is applied. The solution is marched in time using a Taylor series expansion following the Lax-Wendroff approach.

The Navier-Stokes equations for viscous fluid flow are written in the differential form as follows:

$$\frac{\partial \mathbf{U}}{\partial t} + \nabla \cdot \mathcal{F} = \nabla \cdot \mathcal{R} \tag{1}$$

where **U** is the state vector; $\mathcal{F}$ comprises the convective flux vector components **F**, **G** and **H** in the x, y, z- directions respectively; $\mathcal{R}$ comprises the viscous flux vector components **R**, **S** and **T** in the x, y, z- directions respectively. The state vector and the convective and viscous flux vectors are defined in terms of primitive variables.

The solution at any node N, at time level n+1 can be expressed in terms of the solution at time level n using a Taylor series expansion:

$$\mathbf{U}_N^{(n+1)} = \mathbf{U}_N^{(n)} + \delta\mathbf{U}_N^{(n)}$$

$$\delta\mathbf{U}_N^{(n)} = \mathbf{U}_N^{(n+1)} - \mathbf{U}_N^{(n)} = \left(\frac{\partial \mathbf{U}}{\partial t}\right)_N^{(n)} \Delta t + \left(\frac{\partial^2 \mathbf{U}}{\partial t^2}\right)_N^{(n)} \Delta t^2 + O\left(\Delta t^3\right) \tag{2}$$

The temporal derivatives in the preceding expression are evaluated in terms of spatial derivatives using the governing equations according to the Lax-Wendroff approach. The finite-volume method evaluates the integral averages of the temporal derivative terms in equation (2) over the control volume Ω_N associated with node N.

The integral average of the first-order temporal derivatives associated with the node N is written in discrete form following the governing equation (1):

$$\left(\frac{\partial \mathbf{U}}{\partial t}\right)_N = -\frac{1}{\Omega_N} \sum_f \left(\mathcal{F} - \mathcal{R}\right)_f \cdot \hat{\mathbf{n}}_f S_f \tag{3}$$

where the summation f is over all of the discrete faces of the dual mesh that constitute $\partial\Omega_N$. It is shown in [1] that the summation in equation (3) can be alternatively computed on an edgewise basis as:

$$\left(\frac{\partial \mathbf{U}}{\partial t}\right)_N = -\frac{1}{\Omega_N}\sum_e (\mathcal{F}-\mathcal{R})_e \cdot \hat{\mathbf{n}}_e S_e \tag{4}$$

where the summation e is over all of the edges that share the node N. The term S_e represents the dual-face area associated with each edge, and $\hat{\mathbf{n}}_e$ is the unit normal vector of the dual-face area S_e. The S_e is computed using the dual mesh construction of Fig.4 and Fig.5 in Chapter 1, by accumulating the areas of each dual-mesh face that shares the edge. The finite volume scheme then proceeds by computing $\delta\mathbf{U}$s at the nodes by a global sweep over the edges and is thus transparent to whether a node lies in the tetrahedral region, in the prismatic region, or at the interfaces.

The second-order temporal derivatives are evaluated along similar lines. The expression for the second-order derivatives at node N is given from [1] as follows:

$$\left(\frac{\partial^2 \mathbf{U}}{\partial t^2}\right)_N = -\frac{1}{\Omega_N}\int_{\partial\Omega_N}\left(\tilde{\mathbf{A}}n_x + \tilde{\mathbf{B}}n_y + \tilde{\mathbf{C}}n_z\right)\frac{\partial \mathbf{U}}{\partial t}\,dS \tag{5}$$

where $\tilde{\mathbf{A}} = \frac{\partial \mathbf{E}}{\partial \mathbf{U}}$, $\tilde{\mathbf{B}} = \frac{\partial \mathbf{F}}{\partial \mathbf{U}}$, $\tilde{\mathbf{C}} = \frac{\partial \mathbf{G}}{\partial \mathbf{U}}$ are the Jacobians of convective flux vectors. The Jacobians of flux vectors need to be computed to evaluate the second-order derivatives. However, only the convective flux vectors are considered in this step as the Jacobians of viscous flux vectors are too expensive to compute. Therefore, discretization of the viscous terms is first-order accurate in time and second-order accurate in space.

The dissipation modeling in this work is formulated in such a manner as to simulate the implicit dissipation terms of the upwinding schemes without increasing the computation cost of the algorithm [1]. The numerical formula for the flux vector at any intermediate state I between two end states L and R can be expressed as:

$$\mathbf{F}_I = \frac{1}{2}(\mathbf{F}_L - \mathbf{F}_R) - \tilde{\mathbf{A}}_r(\mathbf{U}_R - \mathbf{U}_L) \tag{6}$$

where $\tilde{\mathbf{A}}_r$ is Roe's matrix [1]. The dissipation terms are modeled so as to be similar to the second term of the above equation as this corresponds to the implicit smoothing term of the upwinding scheme. A simplified form of Roe's matrix [1] is obtained by replacing $\tilde{\mathbf{A}}_r$ with $\rho(\tilde{\mathbf{A}}_r) = |u| + c$, the maximum eigenvalue of Roe's matrix. This ensures that the dissipation terms do not dwindle down to zero near the stagnation or the sonic points.

To extend this concept to 3D, *edge-based* operations are adopted for calculation of the artificial dissipation term. The contribution $\left(\delta \mathbf{U}_0^n\right)_{s2}$ of shock smoothing terms to the change $\delta \mathbf{U}_0^n$ at the node 0 is given as follows:

$$\left(\delta \mathbf{U}_0^n\right)_{s2} = \frac{\Delta t}{\Omega_0} \sum_{e=1}^{Ne} \left(f(\mathbf{u},S) + |c||S|\right)_e \left(\mathbf{U}_{N(e)} - \mathbf{U}_0\right)$$

$$f(\mathbf{u},S) = \left|u S_x + v S_y + w S_z\right|$$

$$|S|_e = \left|S_x{}^2 + S_y{}^2 + S_z{}^2\right|_e^{1/2} \tag{7}$$

where e denotes the connected edge. The shock smoothing term is evaluated similar to the viscous fluxes on an edge-wise basis. The fourth order smoothing contribution $\left(\delta \mathbf{U}_0^n\right)_{s4}$ is computed in a similar fashion. Instead of the first difference of state vectors as used in equation (7), a difference of the accumulated first difference over the edges sharing a node is used for background smoothing in the flow high Reynolds number.

The change $\left(\delta \mathbf{U}_0^n\right)_s$ at the node 0 due to second and fourth order smoothing is given by:

$$\left(\delta \mathbf{U}_0^n\right)_s = \sigma_2(\Delta P)\left(\delta \mathbf{U}_0^n\right)_{s2} + \sigma_4(1-\Delta P)\left(\delta \mathbf{U}_0^n\right)_{s4} \tag{8}$$

The pressure switch ΔP is used to turn the shock smoothing and the background smoothing on at the appropriate regions. The coefficients σ_2, σ_4 are empirical parameters that control the amount of shock and background smoothing. Their values are the smallest possible for which the method converges.

The solution at each node is advanced in time using local time steps. A combination of the CFL and diffusion stability limitations is employed. The viscous-like smoothing term can have appreciable magnitude at shock regions, and therefore it is included in the diffusion limitation. The time-step restriction for the 1-D wave equation is $\Delta t \le \Delta x/(|u|+c)$, while the restriction for the 1-D diffusion equation is $\Delta t \le (1/2)/(\Delta x^2/\nu)$, where, in this case, $\nu = \mu/\rho + \sigma_2 \Delta P$.

4.2 Structure

Figure 13-15 show the structure of hybNS.

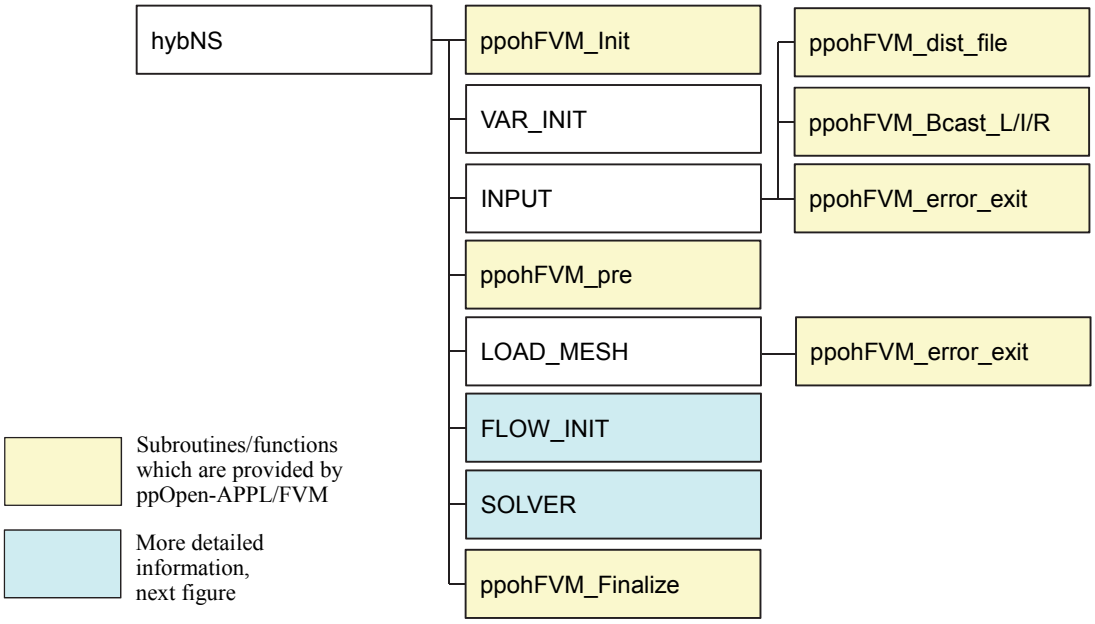


Fig.13 Structure of hybNS (1)

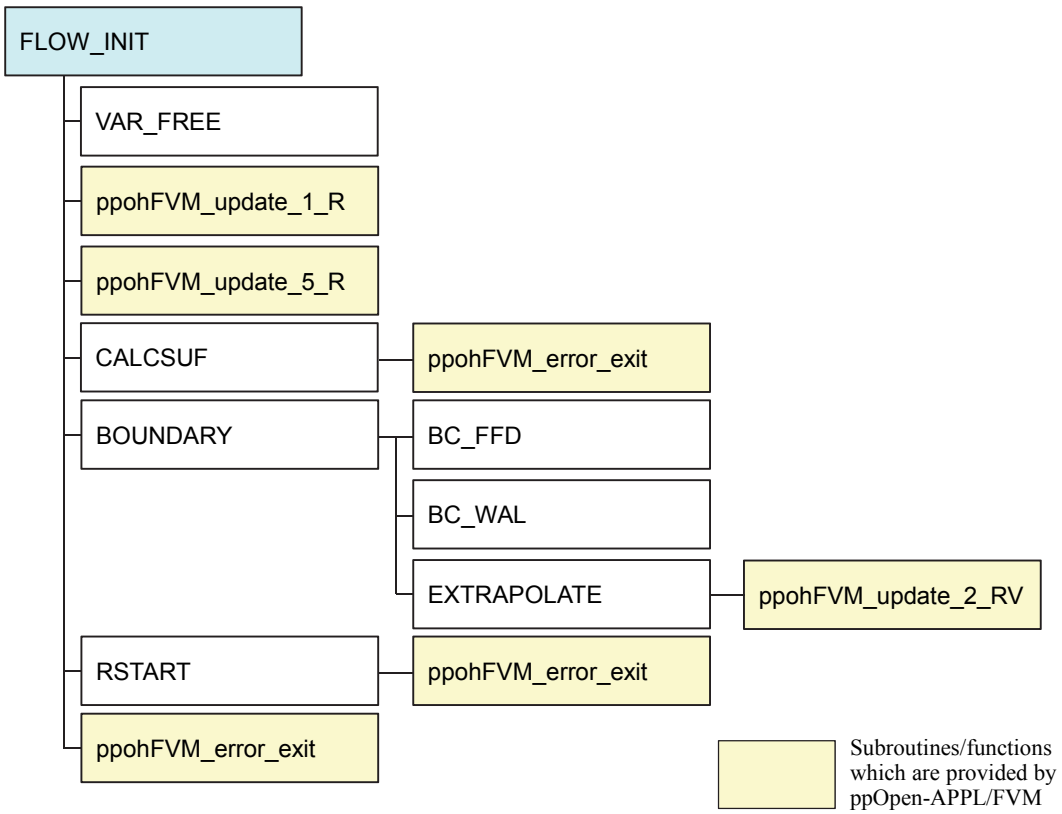


Fig.14 Structure of hybNS (2): subroutines called from `'FLOW_INIT'`

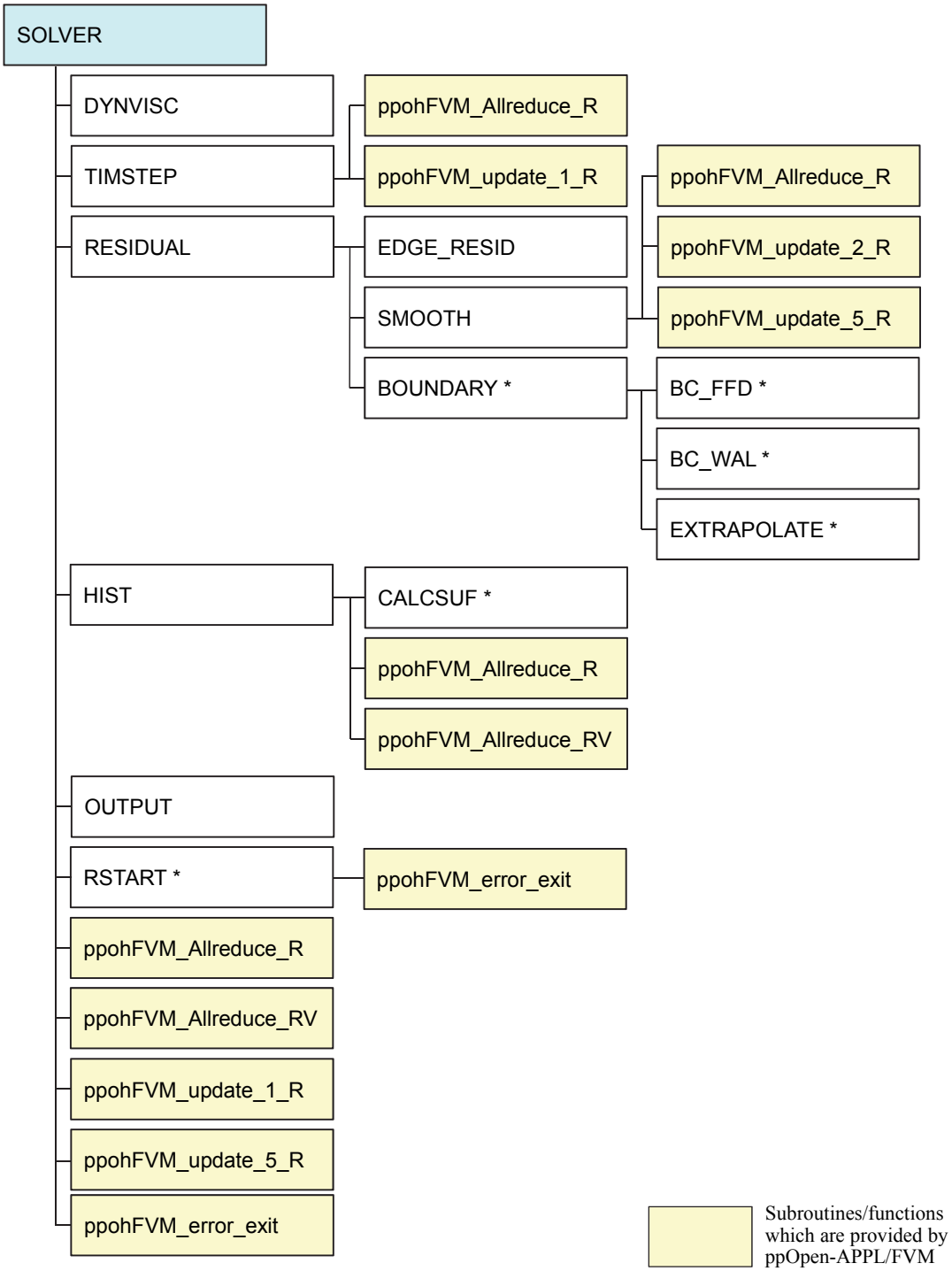


Fig.15 Structure of hybNS (3): subroutines called from `'SOLVER'`

4.3 Modules

HYBRID

This module contains information on variables for meshes, groups, communications, and edges.

Contains the following subroutines/functions

(none)

Parameters

GRIDFIL	C	Name of the distributed grid file [80]
RESTFIL	C	Name of the distributed restart file [80]
INPFIL	C	Name of the control file [80]
RESIDFIL	C	Name of the residual file [80]
RESTFIL	C	Name of the distributed UCD file [80]
RESTART	L	Restart-run (T), or initial-run (F)
STEADY	F	Steady-state (T), or transient (F)
PETOT	I	st_ppohFVM_comm_info%PETOT
my_rank	I	st_ppohFVM_comm_info%my_rank
PEsmpTOT	I	st_ppohFVM_comm_info%PEsmpTOT
STEADY	F	Steady-state (T), or transient (F)
OMEGA	R	Factor for time-step
TIMEmax	R	Maximum time for computation
OMEGA	R	Factor for time-step
DTMAX, DTMIN	R	Max/Min time-step
TIME	R	Current time for computation
iter	I	Current iteration (number of time-steps)
ITER_OLD	I	Final iteration number at the previous run
MAXITER	I	Maximum number of iterations
BCwalID	I	Flag for wall boundary conditions (=0: viscous, =1: inviscid)
NFREQ_HIS	I	Frequency for writing residual file
NFREQ_RES	I	Frequency for writing restart files
NODTOT	I	st_ppohFVM_local_mesh%n_node
NODTOTint	I	st_ppohFVM_local_mesh%n_internal
ICELTOT	I	st_ppohFVM_local_mesh%n_elem
ICELTOTint	I	st_ppohFVM_local_mesh%ne_internal
NODgrpTOT	I	st_grp_data%node_grp%n_enum_grp

```
CELgrpTOT     I  st_grp_data%elem_grp%n_enum_grp
SUFgrpTOT     I  st_grp_data%surf_grp%n_surf_grp

XYZ           RA st_ppohFVM_local_mesh%node
IDnode        IA st_ppohFVM_local_mesh%node_id
IDcell        IA st_ppohFVM_local_mesh%elem_id
ICELindex     IA st_ppohFVM_local_mesh%index_elem
ICELptr       IA st_ppohFVM_local_mesh%ptr_elem
ICELTYP       IA st_ppohFVM_local_mesh%elem_type
intCELlist    IA st_ppohFVM_local_mesh%ne_internal_list
ADAPT_TYP     IA st_ppohFVM_local_mesh%adaptation_type
NODgrpNAME    CA st_ppohFVM_grp_data%node_grp%enum_grp_name
CELgrpNAME    CA st_ppohFVM_grp_data%elem_grp%enum_grp_name
SUFgrpNAME    CA st_ppohFVM_grp_data%surf_grp%surf_grp_name
NODgrpITEM    IA st_ppohFVM_grp_data%node_grp%enum_grp_node
CELgrpITEM    IA st_ppohFVM_grp_data%elem_grp%enum_grp_node
SUFgrpITEM    IA st_ppohFVM_grp_data%surf_grp%surf_grp_node
NODgrpSTACK   IA st_ppohFVM_grp_data%node_grp%enum_grp_index
CELgrpSTACK   IA st_ppohFVM_grp_data%elem_grp%enum_grp_index
SUFgrpSTACK   IA st_ppohFVM_grp_data%surf_grp%surf_grp_index
ICELTOTtetra  I  st_ppohFVM_local_mesh%n_tetra
ICELTOTprism  I  st_ppohFVM_local_mesh%n_prism
ACTtetraTOT   I  st_ppohFVM_local_mesh%n_ACTtetra
ACTprismTOT   I  st_ppohFVM_local_mesh%n_ACTprism
IDtetra       IA st_ppohFVM_local_mesh%tetra_id
IDprism       IA st_ppohFVM_local_mesh%prism_id
ACTtetra      IA st_ppohFVM_local_mesh%ACTtetra_id
ACTprism      IA st_ppohFVM_local_mesh%ACTprism_id
intCELFLAG    IA st_ppohFVM_local_mesh%ne_internal_flag

IEDGTOT       I  st_ppohFVM_edge_info%n_edge
EAREA         RA st_ppohFVM_edge_info%area
VOLEDG        RA st_ppohFVM_edge_info%vol
IEDGNOD       IA st_ppohFVM_edge_info%edgnod
VOLCEL        RA st_ppohFVM_local_mesh%volc
VOLNOD        RA st_ppohFVM_local_mesh%voln
```

```
SAREA         RA st_ppohFVM_local_mesh%sarea
COLORedgeTOT  I  st_ppohFVM_edge_info%n_edge_color
COLORedgeINDEX
              I  st_ppohFVM_edge_info%color_index

DTNOD         RA Local time-step for each node [NODTOT]
U             RA Variables at each node [NODTOT,5]
P             RA Pressure at each node [NODTOT]
VISCL, VISCT  RA Laminar/turbulent viscous coef. at each node [NODTOT]
FCV,GCV,HCV   RA Variables at each node [NODTOT,5]
XCV, PU       RA Variables at each node [NODTOT,5]
```

Uses the following modules

```
m_ppohFVM_util
```

Used by the following subroutines/functions in ppOpen-APPL/FVM

```
All
```

4.4 Subroutines

hybNS

Main program of hybNS

```
!C
!C***************************************************************

      program hybNS
!C
!C
!C    solves :
!C        3-dimensional
!C        steady/unsteady COMPRESSIBLE Navier-Stokes equations
!C
!C    using  :
!C        edge-based finite-volume method
!C        1-step Lax-Wendroff explicit time-marching scheme
!C        embbedded tetrahedral/prismatic HYBRID grids
!C
!C***************************************************************
!C
      use  m_ppohFVM_util
      use  HYBRID

      implicit REAL*8 (A-H,O-Z)
      integer(kind=kint):: errno

      real(kind=4)       :: TARRAY(2), TT

      type (st_ppohFVM_file_info)  :: st_file_info
      type (st_ppohFVM_local_mesh) :: st_local_mesh
      type (st_ppohFVM_grp_data)   :: st_grp_data
      type (st_ppohFVM_comm_info)  :: st_comm_info
      type (st_ppohFVM_edge_info)  :: st_edge_info

!C
!C-- init. MPI
      call ppohFVM_Init (st_comm_info)

      PETOT  = st_comm_info%PETOT
      my_rank= st_comm_info%my_rank

!C
!C-- VARIABLE INITIALIZATION
      if (my_rank.eq.0) write (*,*) '*** VARIABLE INIT.'
      call VAR_INIT

!C
!C-- CONTROL DATA
      if (my_rank.eq.0) write (*,*) '*** CNTL. DATA INPUT'
      call INPUT (file_info, comm_info)

!C
!C-- MESH DATA
      if (my_rank.eq.0) write (*,*) '*** MESH DATA INPUT'

      call ppohFVM_pre (st_file_info, st_local_mesh, st_grp_data, st_comm_info, st_edge_info)
      call LOAD_MESH    (             st_local_mesh, st_grp_data, st_comm_info, st_edge_info)

!C
!C-- INITIAL FLOW FIELD
      if (my_rank.eq.0) write (*,*) '*** FLOW FIELD INIT.'
      call FLOW_INIT (st_comm_info)

!C
!C-- MAIN SOLVER
      STM= MPI_Wtime()
      if (my_rank.eq.0) write (*,*) '*** MAIN SOLVER'
      call SOLVER (st_comm_info)
      ETM= MPI_Wtime()

      call ppohFVM_Finalize (comm_info)

      stop
      end
```

Parameters

```
st_file_info       Derived Type: st_ppohFVM_file_info
st_local_mesh      Derived Type: st_ppohFVM_local_mesh
st_grp_data        Derived Type: st_ppohFVM_grp_data
st_comm_info       Derived Type: st_ppohFVM_comm_info
st_edge_info       Derived Type: st_ppohFVM_edge_info
```

Uses the following Modules

```
m_ppohFVM_util, HYBRID
```

Called by the following subroutines/functions in hybNS

```
(none)
```

Calls the following subroutines/functions

```
ppohFVM_Init, ppohFVM_pre, ppohFVM_Finalize
VAR_INIT, INPUT, LOAD_MESH
FLOW_INIT, SOLVER
```

VAR_INIT

This subroutine initializes variables.

```
      subroutine VAR_INIT

      use m_ppohFVM_util
      use HYBRID
```

Uses the following Modules

`m_ppohFVM_util, HYBRID`

Called by the following subroutines/functions in hybNS

`hybNS`

Calls the following subroutines/functions

`(none)`

INPUT

This subroutine reads the control file from #0 process, and sends information to other processes.

```
      subroutine INPUT (st_file_info, st_comm_info)

      use m_ppohFVM_util
      use HYBRID

      type (st_ppohFVM_file_info)  ::  st_file_info
      type (st_ppohFVM_comm_info)  ::  st_comm_info
```

Parameters

st_file_info	Derived Type: st_ppohFVM_file_info
st_comm_info	Derived Type: st_ppohFVM_comm_info

Uses the following Modules

m_ppohFVM_util, HYBRID

Called by the following subroutines/functions in hybNS

hybNS

Calls the following subroutines/functions

ppohFVM_Bcast_L, ppohFVM_Bcast_R, ppohFVM_Bcast_I

ppohFVM_dist_file, ppohFVM_error_exit

LOAD_MESH

This subroutine transfers the information on variables and arrays in ppOpen-APPL/FVM to those of hybNS

```
      subroutine LOAD_MESH (st_local_mesh, st_grp_data, st_comm_info, st_edge_info)

      use  m_ppohFVM_util
      use  HYBRID

      integer(kind=kint):: errno
      type (st_ppohFVM_local_mesh) :: st_local_mesh
      type (st_ppohFVM_grp_data)   :: st_grp_data
      type (st_ppohFVM_comm_info)  :: st_comm_info
      type (st_ppohFVM_edge_info)  :: st_edge_info
```

Parameters

st_local_mesh	Derived Type: st_ppohFVM_local_mesh
st_grp_data	Derived Type: st_ppohFVM_grp_data
st_comm_info	Derived Type: st_ppohFVM_comm_info
st_edge_info	Derived Type: st_ppohFVM_edge_info

Uses the following Modules

m_ppohFVM_util, HYBRID

Called by the following subroutines/functions in hybNS

hybNS

Calls the following subroutines/functions

ppohFVM_error_exit

FLOW_INIT

This subroutine initializes the flow-field.

```
      subroutine FLOW_INIT (st_comm_info)

      use m_ppohFVM_util
      use HYBRID

      type (st_ppohFVM_comm_info)  ::  st_comm_info
```

Parameters

st_comm_info　　Derived Type: st_ppohFVM_comm_info

Uses the following Modules

m_ppohFVM_util, HYBRID

Called by the following subroutines/functions in hybNS

hybNS

Calls the following subroutines/functions

ppohFVM_update_1_R, ppohFVM_update_5_R, ppohFVM_error_exit

VAR_FREE, CALCSUF, BOUNDARY, RSTART

VAR_FREE

This subroutine initializes free-stream variables.

```
      subroutine VAR_FREE

      use m_ppohFVM_util
      use HYBRID
```

Uses the following Modules

m_ppohFVM_util, HYBRID

Called by the following subroutines/functions in hybNS

FLOW_INIT

Calls the following subroutines/functions

(none)

CALC_SUF

This subroutine calculates the surface pressure coefficients (C_{px}, C_{py}, C_{pz})

```
      subroutine CALCSUF ( CP, MODE )

      use m_ppohFVM_util
      use HYBRID

      real   (kind=kreal), dimension (3) ::  CP
```

Parameters

CP	RA	out	Surface pressure coefficients (C_{px}, C_{py}, C_{pz})
MODE	I	in	=0: Initialization, =1: Calculation of C_p's

Uses the following Modules

m_ppohFVM_util, HYBRID

Called by the following subroutines/functions in hybNS

FLOW_INIT, HIST

Calls the following subroutines/functions

(none)

BOUNDARY

This subroutine controls the subroutines for boundary conditions.

```
      subroutine BOUNDARY (st_comm_info)

      use m_ppohFVM_util
      use HYBRID

      type (st_ppohFVM_comm_info) :: st_comm_info
```

Parameters

st_comm_info　　Derived Type: st_ppohFVM_comm_info

Uses the following Modules

m_ppohFVM_util, HYBRID

Called by the following subroutines/functions in hybNS

FLOW_INIT, RESIDUAL

Calls the following subroutines/functions

EXTRAPOLATE, BC_FFD, BC_WAL

EXTRAPOLATE

This subroutine extrapolates cell-center values to nodal values.

```
      subroutine EXTRAPOLATE (st_comm_info)

      use m_ppohFVM_util
      use HYBRID

      type (st_ppohFVM_comm_info) :: st_comm_info
```

Parameters

`st_comm_info` Derived Type: `st_ppohFVM_comm_info`

Uses the following Modules

`m_ppohFVM_util, HYBRID`

Called by the following subroutines/functions in hybNS

`BOUNDARY`

Calls the following subroutines/functions

`ppohFVM_update_2_RV`

BC_WAL

This subroutine calculates boundary conditions at wall surfaces.

```
      subroutine BC_WAL

      use m_ppohFVM_util
      use HYBRID
```

Uses the following Modules

m_ppohFVM_util, HYBRID

Called by the following subroutines/functions in hybNS

BOUNDARY

Calls the following subroutines/functions

(none)

BC_FFD

This subroutine calculates boundary conditions at far-field.

```
      subroutine BC_FFD

      use m_ppohFVM_util
      use HYBRID
```

Uses the following Modules

m_ppohFVM_util, HYBRID

Called by the following subroutines/functions in hybNS

BOUNDARY

Calls the following subroutines/functions

(none)

RSTART

This subroutine reads/writes distributed restart files.

```
      subroutine RSTART (MODE)

      use m_ppohFVM_util
      use HYBRID
```

Parameters

MODE	I	in	=0: Read, =1: Write

Uses the following Modules

m_ppohFVM_util, HYBRID

Called by the following subroutines/functions in hybNS

FLOW_INIT, SOLVER

Calls the following subroutines/functions

ppohFVM_error_exit

SOLVER

This subroutine controls subroutines for solving compressible 3D Navier-Stokes equations.

```
      subroutine SOLVER (st_comm_info)

      use m_ppohFVM_util
      use HYBRID

      type (st_ppohFVM_comm_info)  ::  st_comm_info
```

Parameters

st_comm_info　　Derived Type: st_ppohFVM_comm_info

Uses the following Modules

m_ppohFVM_util, HYBRID

Called by the following subroutines/functions in hybNS

hybNS

Calls the following subroutines/functions

ppohFVM_Allreduce_R, ppohFVM_Allreduce_RV

ppohFVM_update_1_R, ppohFVM_update_5_R, ppohFVM_error_exit

DYNVISC, TIMSTEP, RESIDUAL, HIST, OUTPUT, RSTART

DYNVISC

This subroutine calculates the dynamic viscosity at each node using Sutherland's law.

```
      subroutine DYNVISC

      use m_ppohFVM_util
      use HYBRID

!$omp parallel do private (in,TEMP)
      do in= 1, NODTOT
        TEMP     = GAM * P(in) / U(in,1)
        VISCL(in)= TEMP**1.5d0 * (1.d0+C2TREF) / (TEMP+C2TREF)
        VISCT(in)= 0.d0
      enddo

      return
      end
```

Parameters

GAM	R	in	Heat capacity ratio (=1.40)
C2TREF	R	in	Coefficient for Sutherland's law
VISCL	RA	out	Non-dimensional laminar viscosity at each node [NODTOT]
VISCT	RA	out	Non-dimensional turbulent viscosity at each node [NODTOT]

Uses the following Modules

m_ppohFVM_util, HYBRID

Called by the following subroutines/functions in hybNS

SOLVER

Calls the following subroutines/functions

(none)

TIMSTEP

This subroutine calculates the local time-step at each node according to stability conditions.

```
      subroutine TIMESTEP (st_comm_info)

      use m_ppohFVM_util
      use HYBRID

      type (st_ppohFVM_comm_info)  ::  st_comm_info
```

Parameters

st_comm_info			Derived Type: st_ppohFVM_comm_info
DTNOD	RA	out	Local time-step at each node [NODTOT]

Uses the following Modules

m_ppohFVM_util, HYBRID

Called by the following subroutines/functions in hybNS

SOLVER

Calls the following subroutines/functions

ppohFVM_Allreduce_R, ppohFVM_update_1_R

RESIDUAL

This subroutine calculates changes in the state-vectors over all edges.

```
      subroutine RESIDUAL (st_comm_info)

      use m_ppohFVM_util
      use HYBRID

      type (st_ppohFVM_comm_info)  ::  st_comm_info
```

Parameters

st_comm_info　　Derived Type: st_ppohFVM_comm_info

Uses the following Modules

m_ppohFVM_util, HYBRID

Called by the following subroutines/functions in hybNS

SOLVER

Calls the following subroutines/functions

EDGE_RESID, SMOOTH, BOUNDARY

EDGE_RESID

This subroutine calculates changes in the state-vectors over all edges.

```
      subroutine EDGE_RESID

      use m_ppohFVM_util
      use HYBRID

      type (st_ppohFVM_comm_info)  ::  st_comm_info
```

Parameters

`st_comm_info` Derived Type: `st_ppohFVM_comm_info`

Uses the following Modules

`m_ppohFVM_util, HYBRID`

Called by the following subroutines/functions in hybNS

`RESIDUAL`

Calls the following subroutines/functions

`(none)`

SMOOTH

This subroutine calculates the 2nd and 4th order smoothing terms.

```
      subroutine SMOOTH (st_comm_info)

      use m_ppohFVM_util
      use HYBRID

      type (st_ppohFVM_comm_info)  ::  st_comm_info
```

Parameters

`st_comm_info` Derived Type: `st_ppohFVM_comm_info`

Uses the following Modules

`m_ppohFVM_util, HYBRID`

Called by the following subroutines/functions in hybNS

`RESID`

Calls the following subroutines/functions

`ppohFVM_update_2_R, ppohFVM_update_5_R, ppohFVM_Allreduce_R`

HIST

This subroutine writes the residual history file.

```
      subroutine HIST (NCONV)

      use m_ppohFVM_util
      use HYBRID
```

Uses the following Modules

`m_ppohFVM_util, HYBRID`

Called by the following subroutines/functions in hybNS

`RESID`

Calls the following subroutines/functions

`ppohFVM_Allreduce_RV, CALC_SUF`

OUTPUT

This subroutine writes UCD files for visualization.

```
      subroutine OUTPUT

      use st_ppohFVM_util
      use HYBRID
```

Uses the following Modules

m_ppohFVM_util, HYBRID

Called by the following subroutines/functions in hybNS

RESID

Calls the following subroutines/functions

(none)

4.5 Procedures of hybNS

(1) Overview

The hybNS code solves 3D compressible Navier-Stokes equations for external flow problems. In the example, flow problems around a sphere surface are solved. `$(PREFIX)/bin/hybNS_mg` creates prismatic and tetrahedral meshes between two spherical surfaces.

Semi-unstructured prismatic grids generated from triangles on a spherical surface are used (Fig.16). Meshes are initiated from an icosahedron and are globally refined recursively as in Fig.17. The surface of the model is covered with triangles, which provide geometric flexibility, while the structure of the mesh in the direction normal to the surface provides thin prismatic elements suitable for the viscous region. Prismatic meshes away from the sphere surface are divided into three tetrahedra [1].

Figure 18 overviews procedures for parallel 3D Navier-Stokes simulations using hybNS. Three processes are needed as follows:

(1) Initial Mesh Generation using `$(PREFIX)/bin/hybNS_mg`

(2) Generation of Distributed Local Mesh Files using `$(PREFIX)/bin/ppohFVM_part`
Please do not use `ppohFVM_part` in ppohFVM_0.1.0. ppohFVM_0.2.0 does not accept distributed data sets generated by `ppohFVM_part` in ppohFVM_0.1.0.

(3) Running `$(PREFIX)/bin/hybNS`

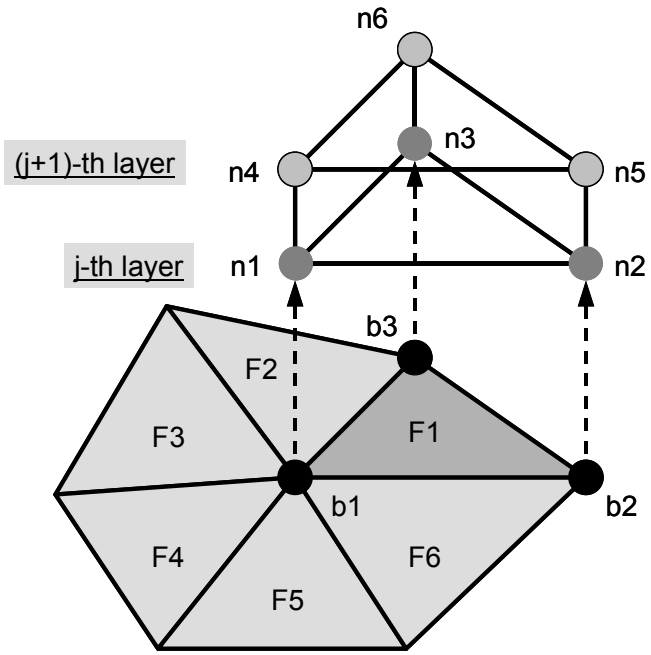


Fig.16 Prisms generated from triangular facets

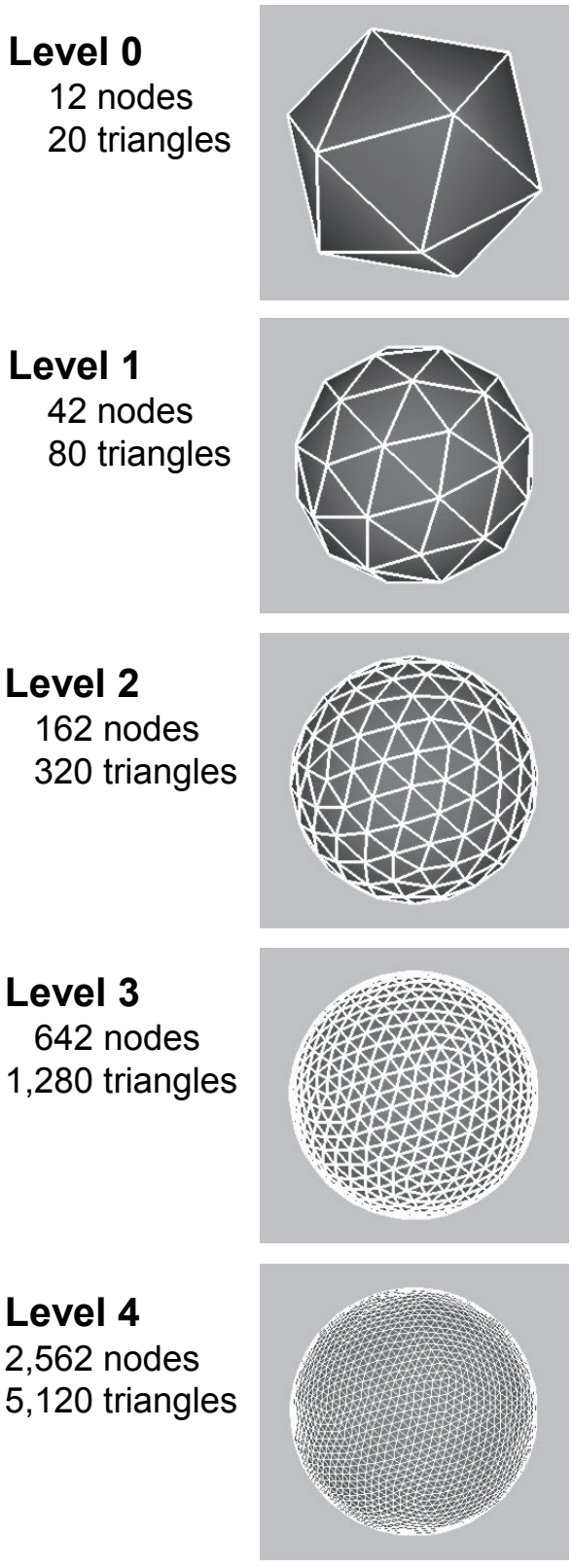


Fig.17 Surface triangle meshes generated from icosahedron

4 children generated from 1 parent triangle [1]

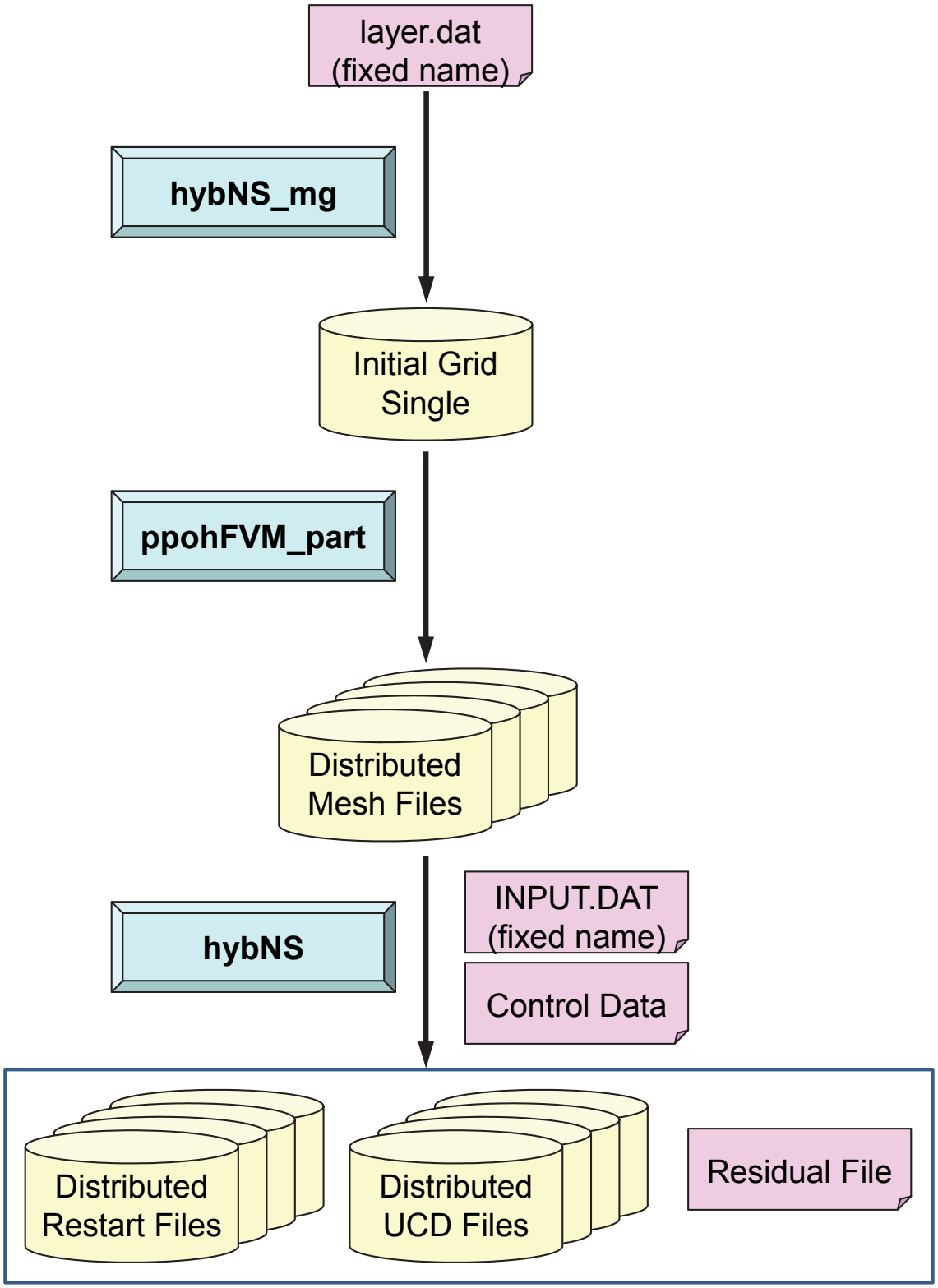


Fig.18 Procedures for parallel 3D Navier-Stokes simulations using hybNS

(2) Parallel Code using ppOpen-APPL/FVM

The main program of hybNS is written in the following manner, where statements written in red letters are related to ppOpen-APPL/FVM.

```
      program hybNS

      use  m_ppohFVM_util
      use  HYBRID

      implicit REAL*8 (A-H,O-Z)
      integer(kind=kint):: errno

      type (st_ppohFVM_file_info)  :: st_file_info
      type (st_ppohFVM_local_mesh) :: st_local_mesh
      type (st_ppohFVM_grp_data)   :: st_grp_data
      type (st_ppohFVM_comm_info)  :: st_comm_info
      type (st_ppohFVM_edge_info)  :: st_edge_info

      call ppohFVM_Init (st_comm_info)

      PETOT  = st_comm_info%PETOT
      my_rank= st_comm_info%my_rank

      call VAR_INIT
      call INPUT (st_file_info, st_comm_info)

      call ppohFVM_pre (st_file_info, st_local_mesh, st_grp_data, st_comm_info, st_edge_info)
      call LOAD_MESH   (              st_local_mesh, st_grp_data, st_comm_info, st_edge_info)

      call FLOW_INIT (st_comm_info)

      call SOLVER (st_comm_info)

      call ppohFVM_Finalize (st_comm_info)

      stop
      end
```

What users have to do are:

- Use 'm_ppohFVM_util'
- Introduce five derived types (st_ppohFVM_file_info, st_ppohFVM_local_mesh, st_ppohFVM_grp_data, st_ppohFVM_comm_info, st_ppohFVM_edge_info)
- Call 'ppohFVM_pre' for reading distributed mesh files, and for creating edge-based metrics automatically
- Call 'ppohFVM_Init' and 'ppohFVM_Finalize' instead of 'MPI_Init' and 'MPI_Finalize'
- Source files must be compiled in the following way:
  - Including module files in $(PREFIX)/include
  - Linking libraries in $(PREFIX)/lib

(3) Initial Mesh Generation

The initial mesh around a spherical surface is generated by `$(PREFIX)/bin/hybNS_mg`. Figure 19 describes procedures for compiling hybNS/hybNS_mg and initial mesh generation by hybNS_mg. As shown in Fig.18, `layer.dat` (fixed file name) should be prepared for initial mesh generation by hybNS_mg. This file provides information on the layer of prisms described in Fig.16. Fig.20 shows an example of `layer.dat`, with the total number of layers (`=ILAYTOT`), total number of prismatic layers (`=ILAYTOT_PRISM`), and the radius (height) of each layer (`DR(k), k=1, ILAYTOT+1`). In this case, meshes between the 4th and 12th layers are divided into tetrahedral meshes. `ILEVTOT` must be provided by users as shown in Fig.19. This value corresponds to the refinement level shown in Fig.18. If `ILEVTOT=3` is specified as shown in the example of Fig.19, triangle meshes at Level 3 of Fig.17 with 162 nodes and 1,280 triangles are generated. Figure 21 shows the format of the initial mesh file.

Maximum mesh size is defined by the two parameters (`NAFACES, NNODES`) in the following file:

```
$(CUR)/examples/hybNS/mg/HYBRID.inc
```

where `NFACES` is a parameter for number of nodes and faces on sphere surfaces, and `NNODES` is one for total number of nodes and elements. Currently, these parameters are specified as follows:

- `NFACES=     50,000`
- `NNODES= 10,000,000`

Users can change these numbers according to problem size and hardware specification. After changing `HYBRID.inc`, source files must be re-compiled in the following way:

Operations	Files created (libraries, module files, exec. files)
$> cd $(CUR)/** **$> make clean	
$> make hybNS	`$(CUR)/bin/hybNS` `$(CUR)/bin/hybNS_mg`
$> make bin_install	`$(PREFIX)/bin/hybNS` `$(PREFIX)/bin/hybNS_mg`

```
$> cd ppohFVM_0.1.0

$> make
$> make install
$> make hybNS
$> make hybNS_install

$> cd examples/mg

$> ./$(PREFIX)/bin/hybNS_mg
 ILEVTOT ?
3
           1          42         100         120
           2         162         420         480
           3         642        1700        1920
       1    5.000000E-01
       2    5.500000E-01
       3    6.000000E-01
       4    6.500000E-01
       5    7.000000E-01
       6    7.500000E-01
       7    8.000000E-01
       8    9.000000E-01
       9    1.000000E+00
      10    1.100000E+00
      11    1.200000E+00
      12    1.350000E+00
      13    1.500000E+00
        8346       38400
 GRID FILE NAME ?
XXX.grid
```

Fig.19 Procedures for compiling hybNS/hybNS_mg and initial mesh generation by hybNS_mg

```
      12              ILAYTOT:          Total number of layers
       3              ILAYTOT_PRISM:    Total number of prism layers
   5.000000E-01       DR(1):            Radius of sphere surface
   5.500000E-01
   6.000000e-01
   6.500000e-01
   7.000000E-01
   7.500000E-01
   8.000000E-01
   9.000000E-01
  10.000000E-01
  11.000000E-01
  12.000000E-01
  13.500000E-01
  15.000000E-01       DR(ILAYTOT+1):    Radius of outermost surface
```

Fig.20 Example of `layer.dat`

```
        read (IUNIT,*) npp                          =0
        read (IUNIT,*) myID                         =0
        read (IUNIT,*) neibNUM                      =0
        read (IUNIT,*) local_mesh%n_material        Number of material components
                                                    (defined at each element)
!C
!C-- NODE info.
        read (IUNIT,*) local_mesh%n_node, local_mesh%n_internal
                       local_mesh%n_node=local_mesh%n_internal (Total node number)
        do i= 1, local_mesh%n_node
          read (IUNIT,*) ii, (local_mesh%node(i,k),k =1,3)
        enddo
!C
!C-- ELEMENT Info.
        read (IUNIT,*) local_mesh%n_elem
        read (IUNIT,'(10i10)')                                                  &
     &       (local_mesh%elem_type(i),i=1,local_mesh%n_elem)
        do icel= 1, local_mesh%n_elem
          ityp= local_mesh%elem_type(icel)
          if (ityp.eq.341) IELMDMY= 4
          if (ityp.eq.351) IELMDMY= 6
          if (local_mesh%n_material.eq.0) then
            read (IUNIT,*) icc, local_mesh%mat_id(icel),                        &
     &           (local_mesh%elem(icel,k),k=1,IELMDMY)
           else
            read (IUNIT,*) icc, local_mesh%mat_id(icel),                        &
     &           (local_mesh%elem(icel,k),k=1,IELMDMY),                         &
     &           (local_mesh%material(icel,kk), kk=1, local_mesh%n_material)
           endif
        enddo
!C
!C-- BOUNDARY Info. : NODE group
          read  (IUNIT,*) grp_data%node_grp%n_enum_grp
          read  (IUNIT,*)                                                       &
     &          (grp_data%node_grp%enum_grp_index(ig),                          &
     &           ig= 1,grp_data%node_grp%n_enum_grp)
          do ig= 1, grp_data%node_grp%n_enum_grp
              read (IUNIT,'(a80)')                                              &
     &             grp_data%node_grp%enum_grp_name(ig)
              read (IUNIT,*)                                                    &
     &            (grp_data%node_grp%enum_grp_node(is),                         &
     &             is= grp_data%node_grp%enum_grp_index(ig-1)+1,                &
     &                 grp_data%node_grp%enum_grp_index(ig))
          enddo
!C
!C-- BOUNDARY Info. : ELEMENT group
          read  (IUNIT,*) grp_data%elem_grp%n_enum_grp
          read  (IUNIT,*)                                                       &
     &          (grp_data%elem_grp%enum_grp_index(ig),                          &
     &           ig= 1, grp_data%elem_grp%n_enum_grp)
          do ig= 1, grp_data%elem_grp%n_enum_grp
              read (IUNIT,'(a80)')                                              &
     &             grp_data%elem_grp%enum_grp_name(ig)
              read (IUNIT,*)                                                    &
     &            (grp_data%elem_grp%enum_grp_node(is),                         &
     &             is= grp_data%elem_grp%enum_grp_index(ig-1)+1,                &
     &                 grp_data%elem_grp%enum_grp_index(ig))
          enddo
!C
!C-- BOUNDARY Info. : ELEMENT-SURFACE group
        read (IUNIT,*) grp_data%surf_grp%n_surf_grp
          read  (IUNIT,*)                                                       &
     &          (grp_data%surf_grp%surf_grp_index(ig),                          &
     &           ig= 1, grp_data%surf_grp%n_surf_grp)
          do ig= 1, grp_data%surf_grp%n_surf_grp
              read (IUNIT,'(a80)')                                              &
     &             grp_data%surf_grp%surf_grp_name(ig)

              read (IUNIT,*)                                                    &
     &            (grp_data%surf_grp%surf_grp_node(is,1),                       &
     &             is= grp_data%surf_grp%surf_grp_index(ig-1)+1,                &
     &                 grp_data%surf_grp%surf_grp_index(ig))
              read (IUNIT,*)                                                    &
     &            (grp_data%surf_grp%surf_grp_node(is,2),                       &
     &             is= grp_data%surf_grp%surf_grp_index(ig-1)+1,                &
     &                 grp_data%surf_grp%surf_grp_index(ig))
          enddo
```

Fig.21 Format of the initial mesh file

(4) Generation of Distributed Local Mesh Files using ppohFVM_part

Distributed mesh files are generated by `$(PREFIX)/bin/ppohFVM_part`. An initial mesh file created by `$(PREFIX)/bin/hyb_NS` is needed. The following three partitioning strategies are available:

(1) RCB (Recursive Coordinate Bisection)
(2) k-**METIS**
(3) p-**METIS**

"RCB" is simple, but fast and robust. It is suitable for simple geometries but provides only 2^N domains. If you want to use "RCB", you should specify option (1) in the opening menu of the `ppohFVM_part`. **METIS** is widely used, is in the public domain, and can be obtained from the following web-site:

http://www-users.cs.umn.edu/~karypis/metis/

It's also fast and robust. Arbitrary numbers of domains can be obtained. k-**METIS** provides partitioning with least edge-cuts, while p-**METIS** generates load-balanced distributed mesh files. Figures 22 and 23 show procedures for generation of distributed local mesh files by ppohFVM_part, using RCB and k-**METIS**, respectively.

Please do not use `ppohFVM_part` in ppohFVM_0.1.0. ppohFVM_0.2.0 does not accept distributed data sets generated by `ppohFVM_part` in ppohFVM_0.1.0.

```
>$ $(PREFIX)/bin/ppohFVM_part
 Original GRID-FILE ?
a.0                          Initial mesh file by hybNS
 * INODTOT =    8346
 * GRID
 * IELMTOT =   38400
 * ELM
 * BOUNDARY : NODE group
 * BOUNDARY : ELEM group
 * BOUNDARY : SURF group
 * IEDGTOT =   49944   53769

# select PARTITIONING METHOD
  RCB                    (1)
  K-METIS                (2)
  P-METIS                (3)

 Please TYPE 1-3 !!

>>>
1                            RCB

*** RECURSIVE COORDINATE BISECTION (RCB)
 How many partitions (2**n)?

>>>
3                            2^3= 8 regions

***   8 REGIONS

# HEADER of the OUTPUT file ?

>>>
rcb                          HEADER of dist. files

#####  1-th BiSECTION #####

 in which direction ? X:1, Y:2, Z:3

>>>
1                            Direction of bisection
 X-direction

#####  2-th BiSECTION #####

 in which direction ? X:1, Y:2, Z:3

>>>
2
 Y-direction

#####  3-th BiSECTION #####

 in which direction ? X:1, Y:2, Z:3

>>>
3
 Z-direction
```

```
RECURSIVE COORDINATE BISECTION

*** GRID  file   a.0

    8 PEs

TOTAL EDGE     #      49944
TOTAL EDGE CUT #       6389

TOTAL NODE     #       8346
TOTAL CELL     #      38400

 PE    NODE#   CELL#
  1    1044    5932
  2    1044    5972
  3    1043    5910
  4    1043    5934
  5    1043    5958
  6    1043    5871
  7    1043    5946
  8    1043    5968

MAX.node/PE             1044
MIN.node/PE             1043
MAX.cell/PE             5972
MIN.cell/PE             5871

OVERLAPPED ELEMENTS    8635

 PE/NEIB-PE#    NEIB-PEs
  1   5        2   3   7   5   6
  2   6        1   4   3   6   8   5
  3   6        4   2   1   7   8   5
  4   5        2   3   8   6   7
  5   5        1   7   3   6   2
  6   6        2   8   4   5   1   7
  7   6        8   3   1   5   6   4
  8   5        6   4   7   3   2

PE:    1      1558       1044
PE:    2      1571       1044
PE:    3      1549       1043
PE:    4      1552       1043
PE:    5      1551       1043
PE:    6      1571       1043
PE:    7      1564       1043
PE:    8      1579       1043
 * normal termination

>$ ls rcb.*
rcb.0  rcb.1  rcb.2  rcb.3  rcb.4  rcb.5
rcb.6  rcb.7
```

Fig.22 Generation of Distributed Local Mesh Files by ppohFVM_part (RCB)

```
>$ $(PREFIX)/bin/ppohFVM_part
 Original GRID-FILE ?
a.0                    Initial mesh file by hybNS
 * INODTOT =    8346
 * GRID
 * IELMTOT =   38400
 * ELM
 * BOUNDARY : NODE group
 * BOUNDARY : ELEM group
 * BOUNDARY : SURF group
 * IEDGTOT =   49944    53769

# select PARTITIONING METHOD
  RCB                   (1)
  K-METIS               (2)
  P-METIS               (3)

 Please TYPE 1-3 !!

>>>
2                      K-METIS

*** K-METIS/P-METIS
 NUMBER of regions ?

>>>
8                      8 regions

***   8 REGIONS


# HEADER of the OUTPUT file ?

>>>
kmetis                 HEADER of dist. files

*** GRID  file   a.0

    8 PEs

TOTAL EDGE     #      49944
TOTAL EDGE CUT #       5488

TOTAL NODE     #       8346
TOTAL CELL     #      38400
```

```
 PE     NODE#    CELL#
  1     1033     5245
  2     1043     5625
  3     1016     5176
  4     1015     6480
  5     1071     6280
  6     1024     5438
  7     1070     5757
  8     1074     6342

MAX.node/PE             1074
MIN.node/PE             1015
MAX.cell/PE             6480
MIN.cell/PE             5176

OVERLAPPED ELEMENTS     7463

 PE/NEIB-PE#     NEIB-PEs
  1    6         5    6    2    3    4    7
  2    6         1    5    7    3    8    4
  3    6         1    2    6    7    8    4
  4    6         3    1    6    5    2    8
  5    6         1    2    7    6    8    4
  6    6         1    5    7    8    3    4
  7    6         5    2    6    8    3    1
  8    6         6    7    3    5    2    4

PE:     1       1495       1033
PE:     2       1504       1043
PE:     3       1474       1016
PE:     4       1435       1015
PE:     5       1608       1071
PE:     6       1535       1024
PE:     7       1555       1070
PE:     8       1539       1074
 * normal termination

>$ ls kmetis.*

kmetis.0  kmetis.1  kmetis.2  kmetis.3
kmetis.4  kmetis.5  kmetis.6  kmetis.7
```

Fig.23 Generation of Distributed Local Mesh Files by ppohFVM_part (k-**METIS**)

(5) Running hybNS

Running procedures with hybNS is shown in Fig.24.

The following files are required:

- ‘`INPUT.DAT` (fixed name)’, which specifies the name of the control file and must be in the current directory
- A control file, the name of which is specified in ‘`INPUT.DAT`’ (Fig.25)
- Distributed mesh files

The following files are generated after computation:

- Distributed restart files
- Distributed UCD files for visualization
- A residual history file
- Standard output (convergence history) , reference results can be found at `$(CUR)/examples/hybNS/run/results/results.lst`

```
$> cd ppohFVM_0.1.0/examples/hybNS/run

$> ls INPUT.dat
   INPUT.DAT

$> cat INPUT.DAT
   inp

$> ls inp
   inp

$> mpirun -np 8 $(PREFIX)/bin/hybNS
   (or corresponding operation)

$> ls bb2.*
bb2.0.inp  bb2.1.inp  bb2.2.inp  bb2.3.inp  bb2.4.inp
bb2.5.inp  bb2.6.inp  bb2.7.inp

$> ls bb2r.*
bb2r.0  bb2r.1  bb2r.2  bb2r.3  bb2r.4  bb2r.5  bb2r.6  bb2r.7
```

Fig.24 Running hybNS

```
resid1                          Name of residual history file
0                               FLAG for wall B.C. (=0: viscos, =1: inviscid)
1.40d00   0.00d00   0.00d-01    Non-dim. velocity (Mach number) (u,v,w) at far-field
1.00      1.0e-3                Non-dim. density and laminar viscosity
0.51d0                          Coefficient for Sutherland's law (C2TREF)
1.00e-01   1.00e-02             Coefficients for 2nd & 4th order artificial dissipation
0.72 0.90                       Laminar & turbulent Prandtl number (PRL, PRT)
0.10                            Coef. for time-step (OMEGA)
10                              Maximum number of itreations
T                               (=T): Steady State, (=F): Transient
F                               (=T): Restart run, (=F): Initial run
8000   8000                     Frequency of writing residual file and restart file
4                               Number of OpenMP threads (=0: Flat MPI)
../data/bbb                     HEADER for distributed mesh files
bb2r                            HEADER for distributed restart files
bb2                             HEADER for distributed UCD files
                                File name: '<HEADER>.<my_rank>.inp'
```

Fig.25 Example of a Control File

Laminar dynamic viscosity ν_L is defined as follows:

$$\nu_L = \frac{T^{3/2}(1+C)}{(T+C)}, \quad T = \frac{\gamma \cdot P}{\rho} \tag{9}$$

where γ: heat capacity ratio, =1.40 in hybNS, *T, P, ρ*: non-dimensionalized temperature, pressure and density, *C*: `C2TREF` in Fig.25.

Figure 25 shows an example of a control file. Restart files and UCD file are replaced with previous files, whenever they are written. Therefore, the most recent version of these files is kept.

Local time step for each edge is adopted in `STEADY` case, where time step `Δt` is calculated on each edge according to the method defined in 4.1. On the contrast, global time step is applied in `TRANSIENT` case, where minimum value of `Δt` is adopted for all edges [1].

Figure 26 shows results of supersonic simulations with 8,346 nodes and 38,400 meshes.

As is mentioned before, local data structure has been modified in ver.0.2.0, and performance of computation and communication of ver.0.2.0 has been much improved compared to that of ver.0.1.0. Performance of expensive subroutines has been improved as follows for sample problems with 40,992 nodes and 128,000 elements using 8 nodes of Fujitsu PRIMEHPC FX10 at the University of Tokyo (Oakleaf-FX), where each node has 16 threads.

- **`edge_rerid`**: 1.32 times faster than ver.0.1.0
- **`smooth`**: 1.89 times

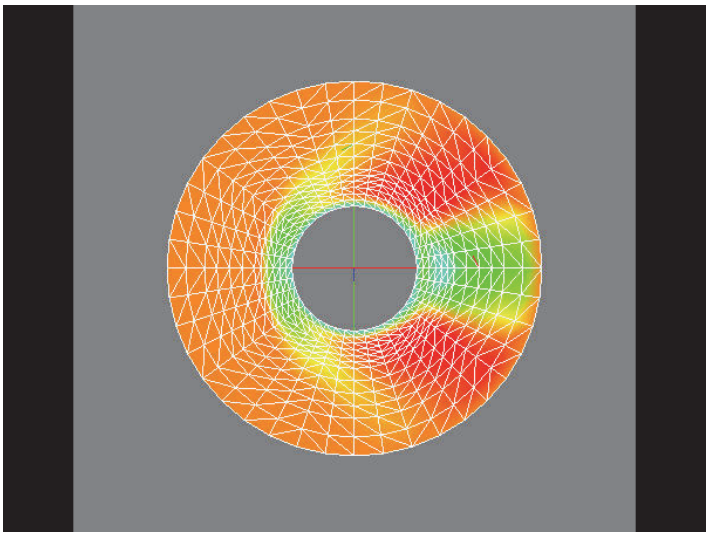


Fig.26 Results of a supersonic simulation with 8,346 nodes and 38,400 meshes, of the distribution of Mach number, M_{inf}= 1.40, Re=10^3

5. heat3D

5.1 Overview

The heat3D is a parallel 3D code for steady-state heat conduction code based on finite-element method (FEM). Because ppOpen-APPL/FVM provides parallel data structure for distributed unstructured meshes, users can utilize that as a platform for development of parallel FEM codes.

Target problem of heat3D is the following steady-state heat conduction equation for a simple cuboid shown in Fig.27:

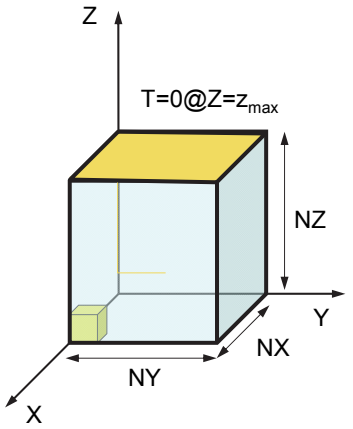


Fig.27 Target problem of heat3D

$$\frac{\partial}{\partial x}\left(\lambda\frac{\partial T}{\partial x}\right)+\frac{\partial}{\partial y}\left(\lambda\frac{\partial T}{\partial y}\right)+\frac{\partial}{\partial z}\left(\lambda\frac{\partial T}{\partial z}\right)+\dot{Q}(x,y,z)=0 \qquad (1)$$

Each mesh is 1×1×1 cubic tri-linear element, and each of *NX, NY*, and *NZ* is number of meshes in each of x-, y-, and z- direction. λ is thermal conductivity, and it is assumed to be uniform (=1) in this problem. Dirichlet boundary condition T=0 is applied at $Z=Z_{max}$ surface, and heat generation term is defined as follows:

$$\dot{Q}(x,y,z)=QVOL\left|x_C+y_C\right| \qquad (2)$$

where *QVOL* is a parameter defined by used, and x_c and y_c are x- and y-coordinates of the center of each mesh. heat3D is based on traditional Galerkin method, and derived linear equations is solved by parallel Conjugate Gradient (CG) solver with point Jacobi preconditioning method.

Figure 28 shows the structure of heat3D. heat3D includes some prototype functions of ppOpen-APPL/FVM. In future those functions starting from "ppohFVM_" will be included in ppOpen-APPL/FVM library.

In heat3D, we don't use procedures for edge-based computations of ppOpen-APPL/FVM, therefore, `st_ppohFVM_edge_info%used_edges` is set to "false". `ppohFVM_ucd_regular_hexa_1` is a special function for parallel visualization for cuboid geometries, as shown in Fig.27.

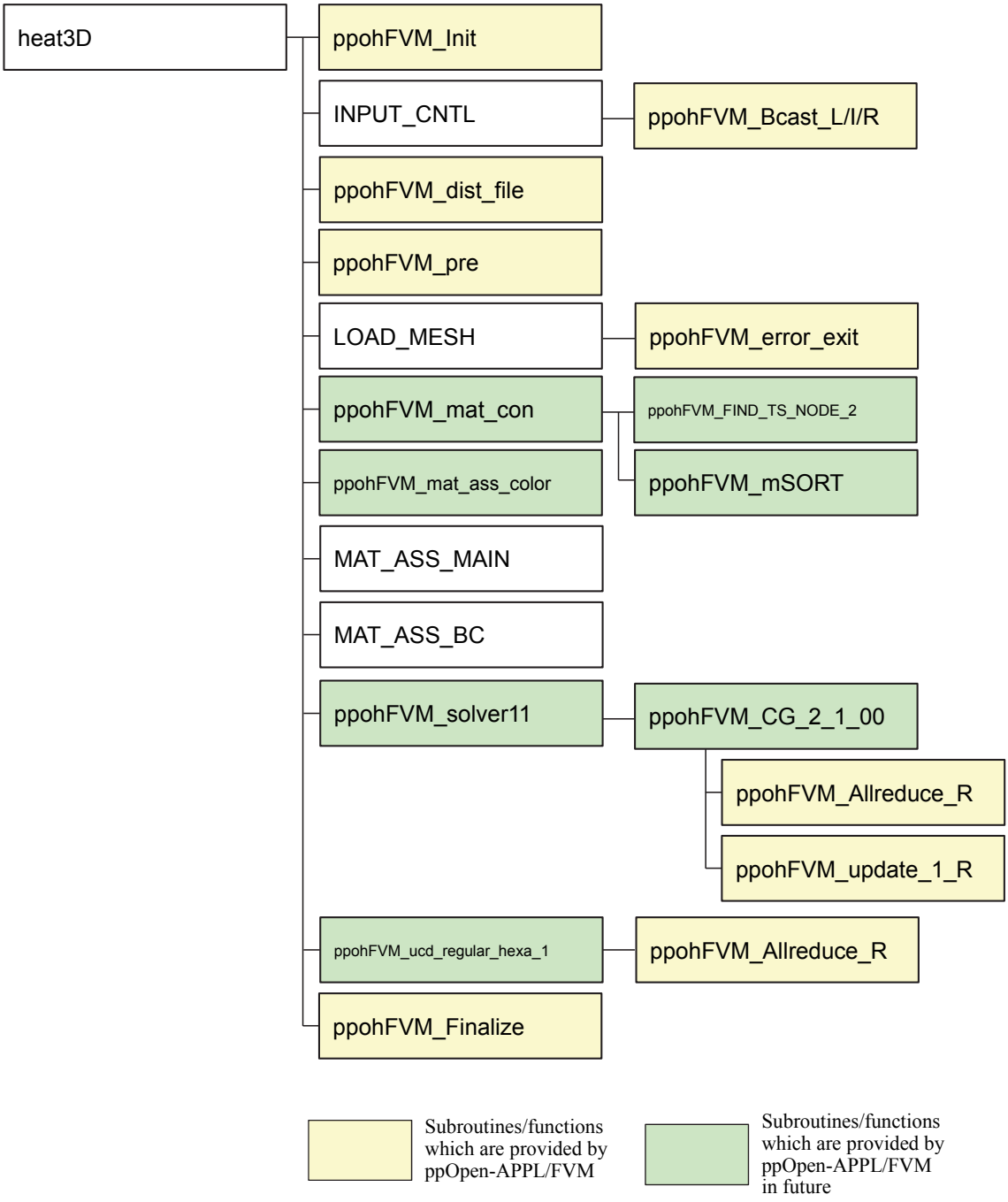


Fig.28 Structure of heat3D

m_ppohFVM_util_matrix

This module contains information on variables for sparse coefficient matrices of linear equations.

Contains the following subroutines/functions

(none)

ppohFVM_util_matrix.f90

```
module m_ppohFVM_util_matrix

use m_ppohFVM_util

implicit none
public

type st_ppohFVM_matrix_info
  integer TYPE, BLOCKsize, DomainDecomposition
  integer N, NP
  integer NL , NU , NLU
  integer NPL, NPU, NPLU
  integer,pointer:: INL(:)  , INU(:)  , INLU(:)
  integer,pointer:: IAL(:,:), IAU(:,:), IALU(:,:)

  integer, dimension(:), allocatable :: indexL, indexU, index
  integer, dimension(:), allocatable ::  itemL,  itemU, item

  real(kind=ppohFVM_kreal), dimension(:), allocatable :: AL, AU, D, RHS, X, AMAT

  integer hexa_361_color_tot
  integer, dimension(:), allocatable :: hexa_361_color_index, hexa_361_color_item
end type st_ppohFVM_matrix_info

type st_ppohFVM_solver_info
  integer METHOD, PRECOND, ITER, ITERactual, ERROR, ICFLAG
  real(kind=ppohFVM_kreal) :: RESID
  real(kind=ppohFVM_kreal) :: COMMtime, COMPtime
end type st_ppohFVM_solver_info

type st_ppohFVM_vis_info
  integer n_cell_ucd_reg_hexa_361_1
end type st_ppohFVM_vis_info

end module m_ppohFVM_util_matrix
```

Parameters

st_ppohFVM_matrix_info Derived Type

TYPE	I	Type of coefficient matrix = 1: [A] without [D] = 2: [A] with [D] = 3: [L], [U], and [D]
BLOCKsize	I	Block Size, Number of DOF on each vertex = 1: 1x1 block = 2: 2x2 block = 3: 3x3 block = 4: 4x4 block

DomainDecomposition		
	I	Type of domain decomposition = 0: LBJ (Localized Block Jacobi) = 1: LBJ with 1-layer overlapping extention = 2: LBJ with 2-layer overlapping extention = 3: LBJ with 3-layer overlapping extention =10: LBJ/RCM global =11: LBJ/RCM global with 1-layer overlapping extention =12: LBJ/RCM global with 2-layer overlapping extention =13: LBJ/RCM global with 3-layer overlapping extention =20: HID (Hierarchical Interface Decompotision) with 1-layer overlapping extention =21: HID with 1-layer overlapping extention =22: HID with 1-layer overlapping extention =23: HID with 1-layer overlapping extention
PRECOND	I	Preconditioning method = 0: NO preconditioning = 1: Point/Block Jacobi =10: ILU(0)/IC(0) =11: ILU(1)/IC(1) =12: ILU(2)/IC(2) =13: ILU(3)/IC(3)

Uses the following modules

m_ppohFVM_util

Used by the following subroutines/functions in ppOpen-APPL/FVM

(currently none)

5.3 Procedures of heat3D

(1) Overview

Figure 29 overviews procedures for parallel 3D steady-state simulations of heat conduction using heat3D. Two processes are needed as follows:

(1) Parallel mesh generation using `$(PREFIX)/bin/pmesh` (or `pmesh_bin`)

(2) Running `$(PREFIX)/bin/heat3D`

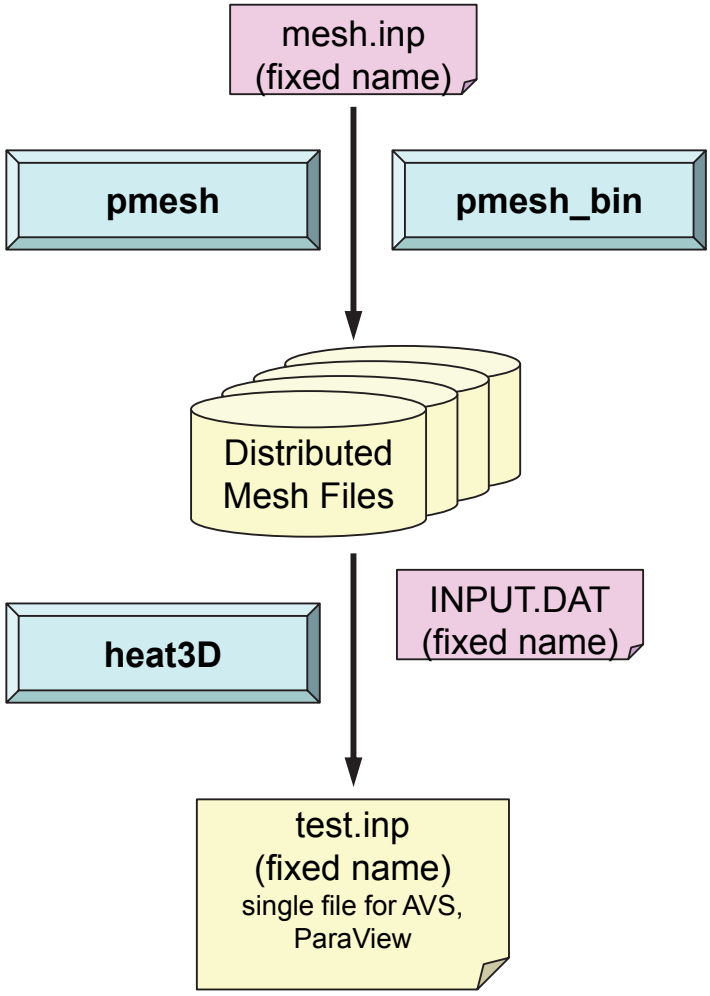


Fig.29 Procedures for parallel 3D steady-state simulations of heat conduction using hybNS

(2) Parallel Mesh Generation

Parallel mesh generation is done by `$(PREFIX)/bin/pmesh` or `pmesh_bin`, as shown in the previous section. `$(PREFIX)/bin/pmesh` generates files in ASCII format, while `$(PREFIX)/bin/pmesh_bin` generates binary files. In heat3D, ASCII format is used, but you can easily adopt binary-type by setting `st_ppohFVM_file_info%mesh_asci` as "false".

In order to conduct parallel mesh generation, control file (`mesh.inp` (fixed file name)) must be prepared at the current directory. This control file consists information, as shown in Fig.30.

```
(values)      (variables)        (descriptions)

32   32   32     npx,npy,npz      Total number of nodes in X-, Y-, and Z- direction
                                  (Nx+1, Ny+1, Nz+1 in Fig.27)
 2    2    2     ndx,ndy,ndz      Partition # in each direction (X,Y,Z)
pcube            HEADER           Header of distributed local file
```

Fig.30 Example of control file (`mesh.inp`) for parallel mesh generation

Each of (`npx, npy, npz`) is total number of nodes in x-, y-, and z- direction, and corresponds to N_x+1, N_y+1, and N_z+1 in Fig.27, respectively. Each of (`ndx, ndy, ndz`) is number of partition in each direction. Total number of distributed mesh files is equal to `ndx×ndy×ndz`, and number of nodes in each local file is `(npx/ndx)×(npy/ndy)×(npz/ndz)`. Therefore, each of of (`npx, npy, npz`) must be *divisible* by each of (`ndx, ndy, ndz`). In order to generate (`ndx×ndy×ndz`) files, you need to run (`ndx×ndy×ndz`) MPI processes. Each MPI process generates one local mesh file very efficiently in fully parallel manner. In the example of Fig.30, 8 (=2×2×2) files are generated using 8 MPI processes. `HEADER` is the header of distributed local mesh files. Each process creates a local mesh file, whose name is "`HEADER.my_rank`".

(3) Running heat3D

Parallel FEM procedure by heat3D (`$(PREFIX)/bin/heat3D`) requires distributed mesh files described in the previous section, and a control file (`INPUT.DAT` (fixed file name)). This control file consists information, as shown in Fig.31.

```
(values)      (variables)       (descriptions)

../pmesh/pcube  HEADER          Header of distributed local files
2000            ITER            Max. number of iterations for CG solver
1.0 1.0         COND, QVOL      Thermal conductivity, Heat generation rate in eq.(2)
1.0e-08         RESID           Convergence criteria for CG solver
1000            N_MESH_VIS      Approximate number of meshes for visualization
```

Fig.31 Example of control file (`INPUT.DAT`) for heat3D

Users have to run MPI processes, whose number corresponds to number of local mesh files created in parallel mesh generation. heat3D creates a result file (`test.inp` (fixed file name)), which can be browsed by AVS and ParaView. Number of meshes in the file is controlled by `N_MESH_VIS` parameter in the control file. Procedures of parallel visualization by `ppohFVM_ucd_regular_hexa_1` are based on the idea in ppOpen-MATH/VIS. Number of meshes are reduced using information of background voxels. Finally, results by large-scale meshes with more than 10^8 meshes can be shown on a lap-top by browsing files with only 10^3 meshes.

Figure 32 shows an example of parallel visualization in heat3D using 192 cores. Original mesh includes 1,179,648 nodes (=192×96×64), and 1,143,135 elements. But `test.inp` created by `ppohFVM_ucd_regular_hexa_1` has only 2,923 nodes, and 703 elements.

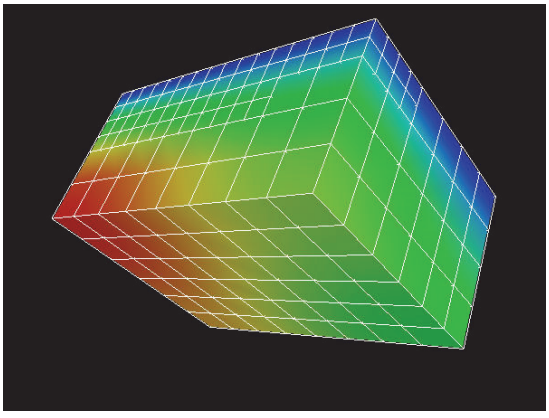


Fig.32 Example of parallel visualization in heat3D using 192 cores. Original mesh includes 1,179,648 nodes (=192×96×64), and 1,143,135 elements. But `test.inp` created by `ppohFVM_ucd_regular_hexa_1` has only 2,923 nodes, and 703 elements.

References

〔1〕 Nakajima, K., Fingberg, J. and Okuda, H., Parallel 3D Adaptive Compressible Navier-Stokes Solver in GeoFEM with Dynamic Load-Balancing by DRAMA Library, HPCN Europe 2001, Amsterdam, Netherlands, Lecture Notes in Computer Science 2110, p.183-193, Springer, 2001.

〔2〕 Nakajima, K., Parallel Iterative Solvers of GeoFEM with Selective Blocking Preconditioning for Nonlinear Contact Problems on the Earth Simulator, ACM/IEEE Proceedings of SC2003, 2003