一种开源多物理场耦合软件EasyFSI

张兵[1]，周帆[1]，员海玮[2]

1. 合肥工业大学

2. 南京航空航天大学

汇报人：张兵

2023. 08. 14 成都

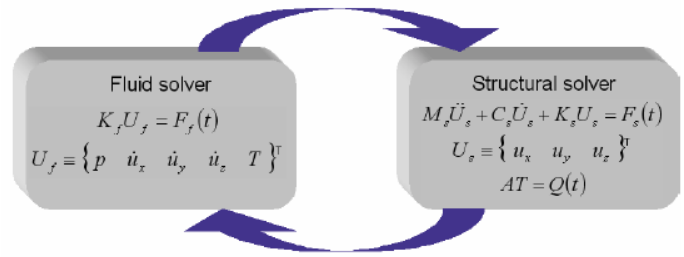
汇报内容

- 多物理场耦合软件技术简介
- 本文主要工作
  - 程序架构
  - 数据通信技术
  - 插值算法
  - 求解器集成
  - 测试算例
- 代码托管

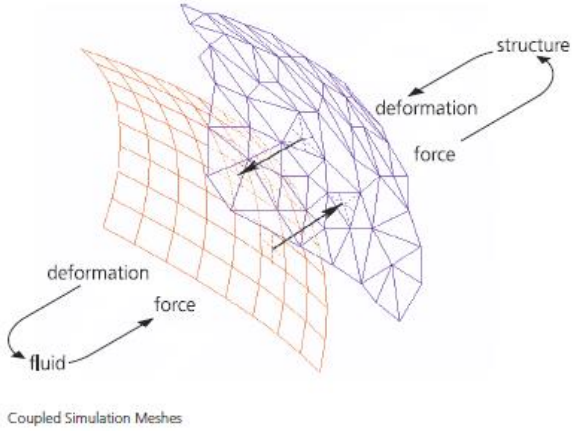
一、多物理场耦合软件技术简介

- 多物理场耦合计算概念
  - 以气动弹性为例

数学

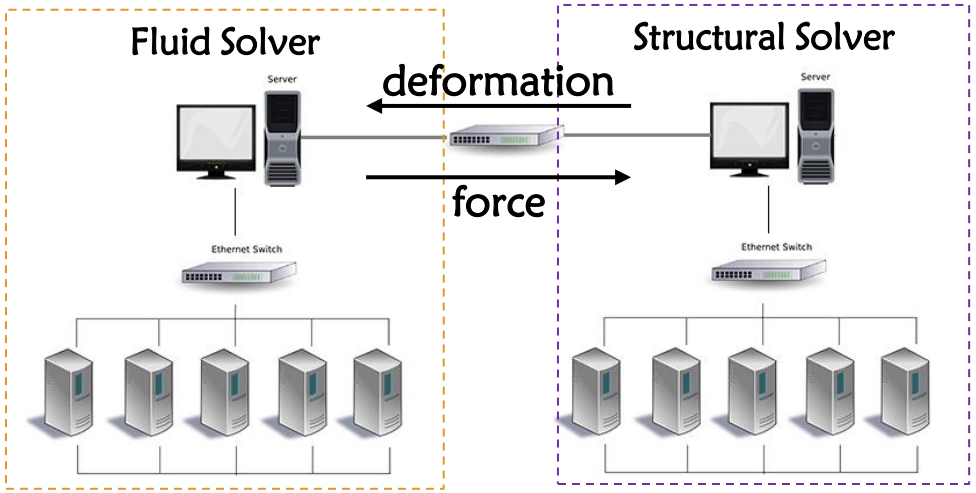


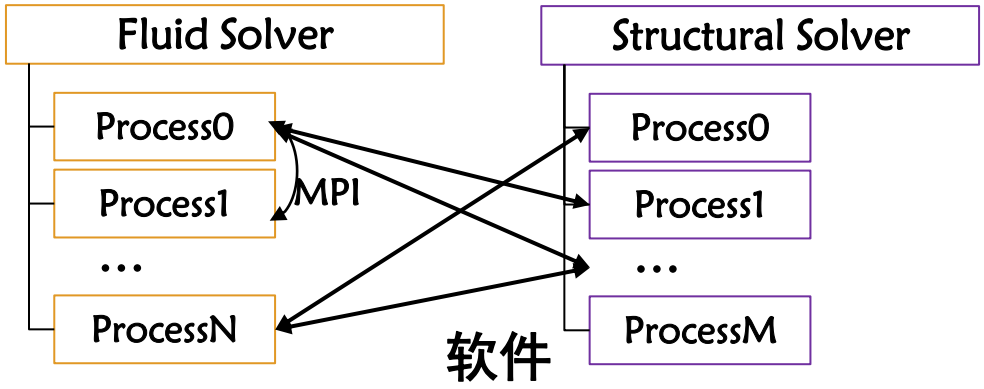
物理



Coupled Simulation Meshes

硬件



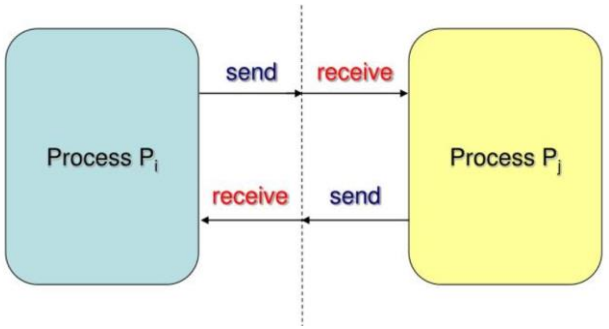


软件

多物理场耦合计算技术

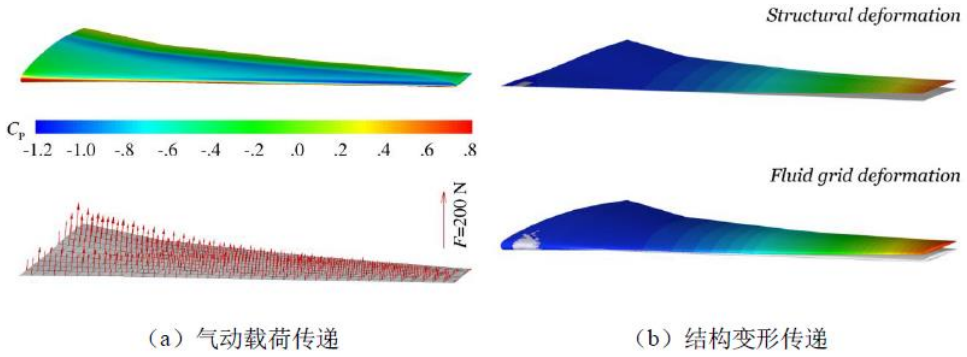
（进程间）数据通信技术

- FILE，RPC，SOCKET，SHM，RDMA



物理量插值算法

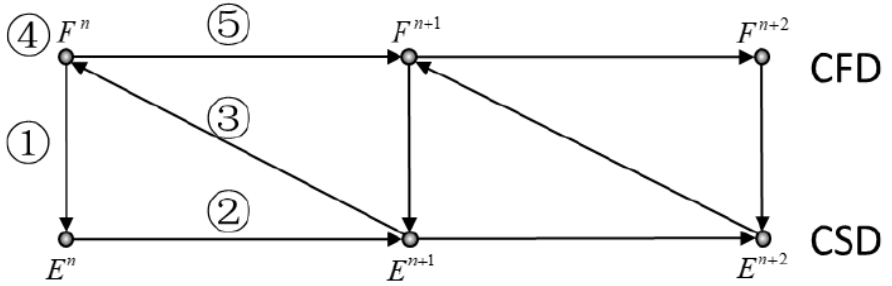
- IPS/TPS，RBF，Mapping



（a）气动载荷传递　　（b）结构变形传递

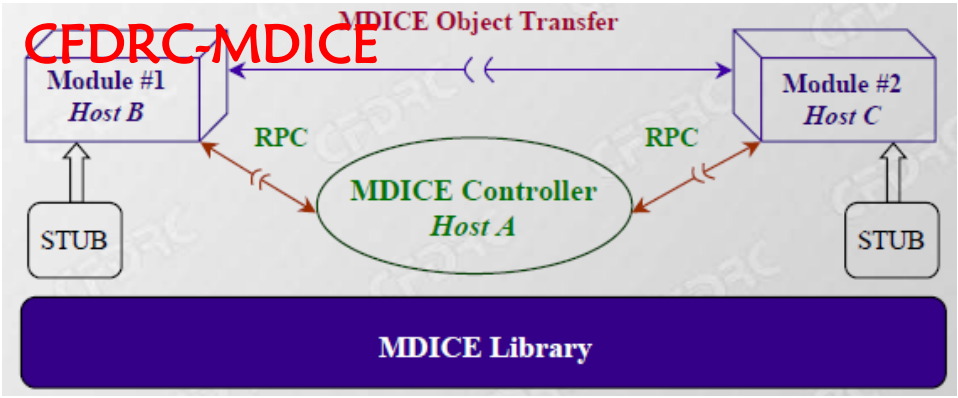
耦合迭代策略

- 松耦合、紧耦合、…

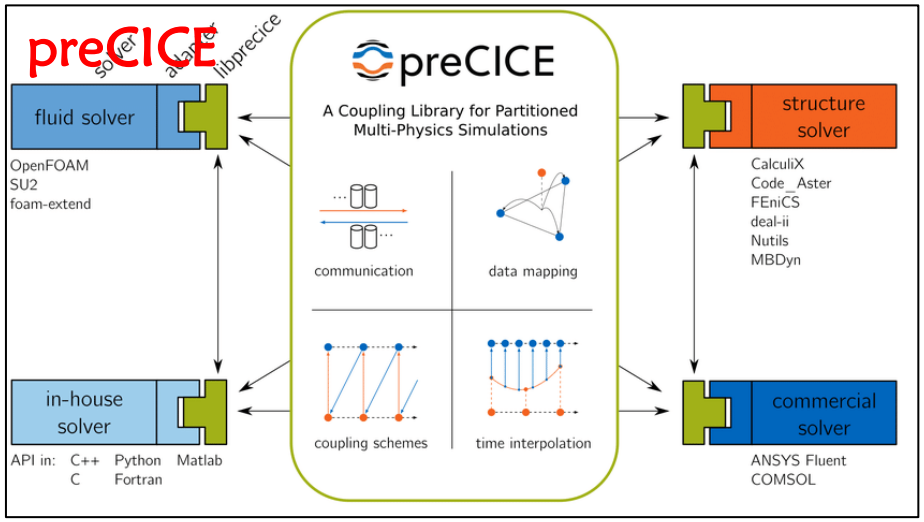


典型多物理场软件架构对比

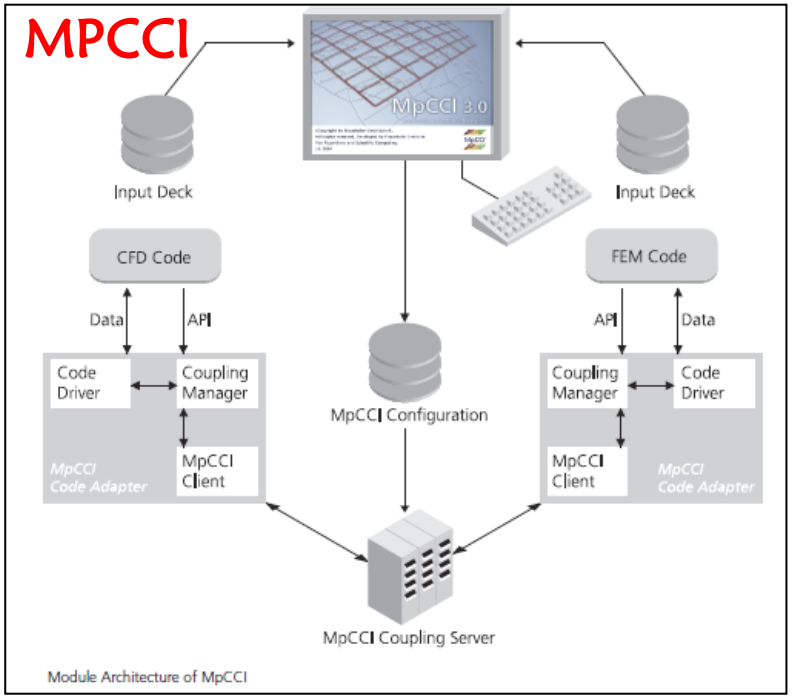
CFDRC-MDICE



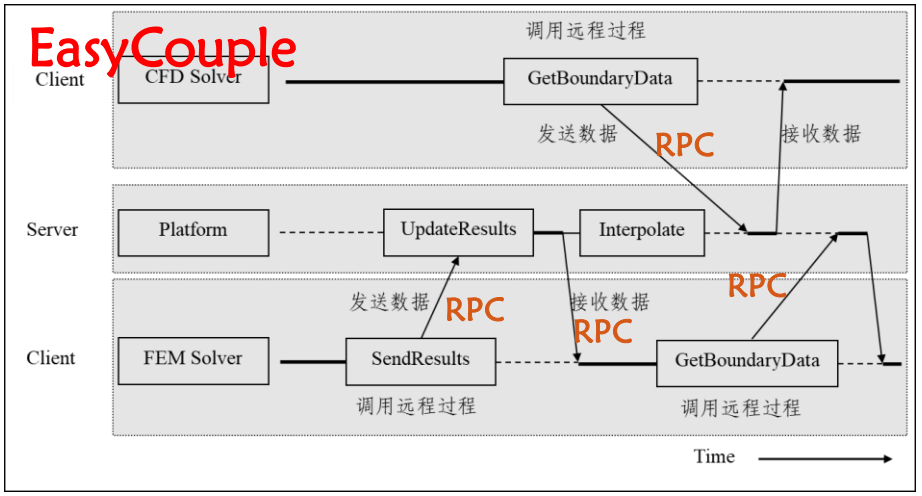
preCICE



MPCCI



EasyCouple

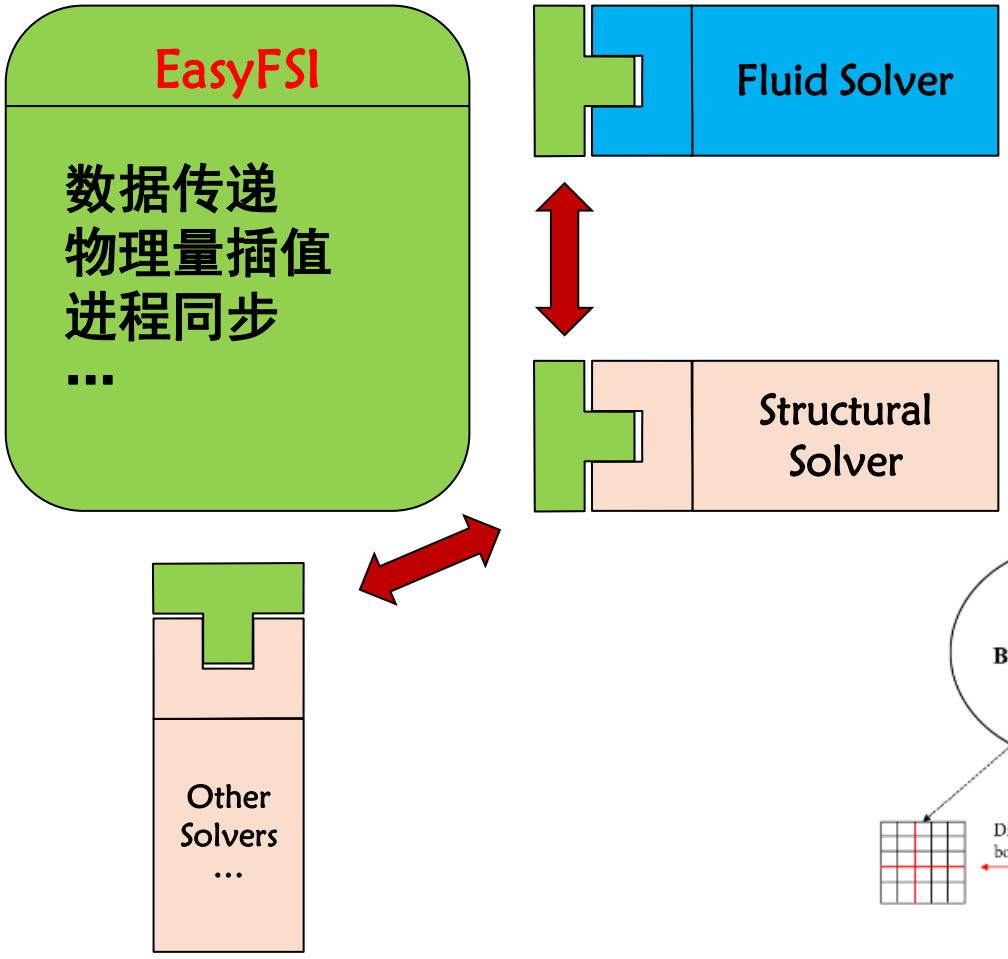


二、本文主要工作

- 开发了一种多物理场耦合软件EasyFSI
  - 基于C++语言开发
  - 可扩展的多种进程通信技术
    - SOCKET
    - SharedMemory
    - MPI
  - 多种物理量插值方法
    - Global/Local IPS/TPS/RBF
    - Mapping
    - Projection
  - 灵活的API接口：C/C++/Fortran/Python/MATLAB

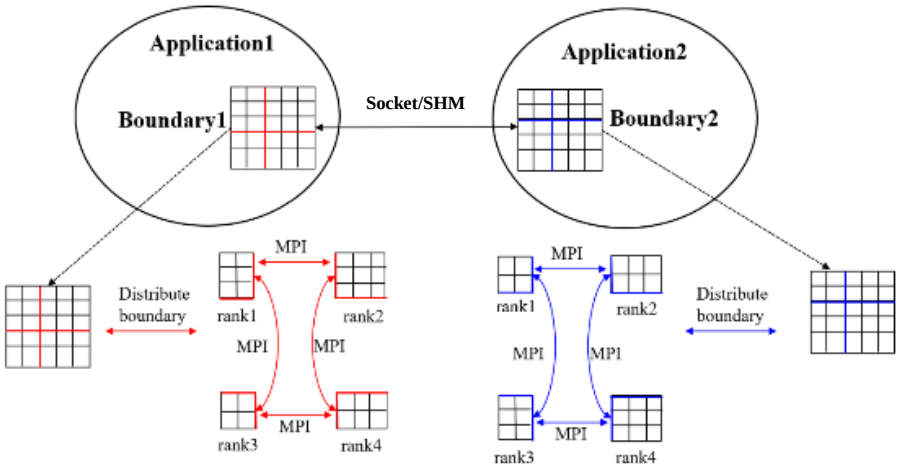
程序架构

- 非中心型程序库



主要程序对象

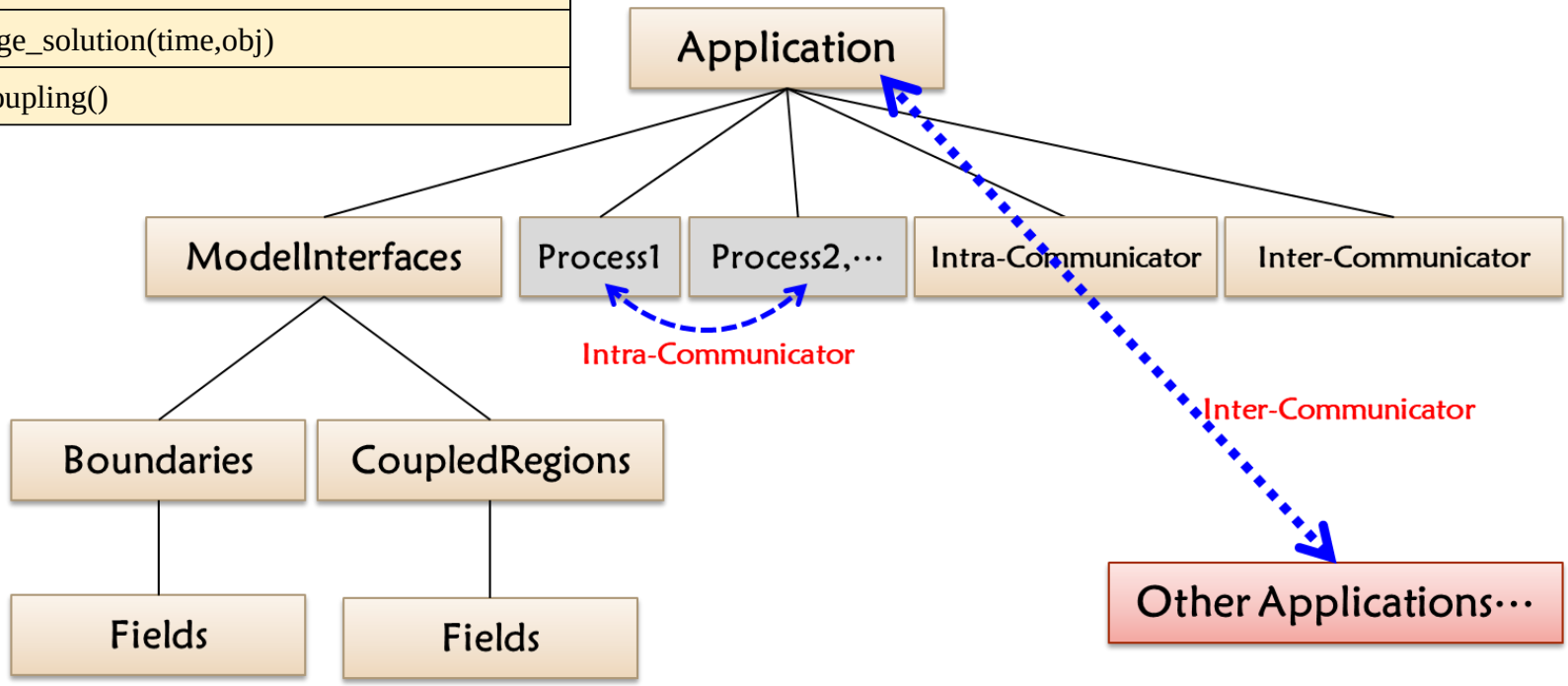
- Application
  - 求解器：CFD,FE,…
- Communicator
  - 一种进程间通信技术
- ModelInterface
  - 一个网格耦合接口
- Interpolator
  - 一种物理量插值算法



应用程序对象:Application

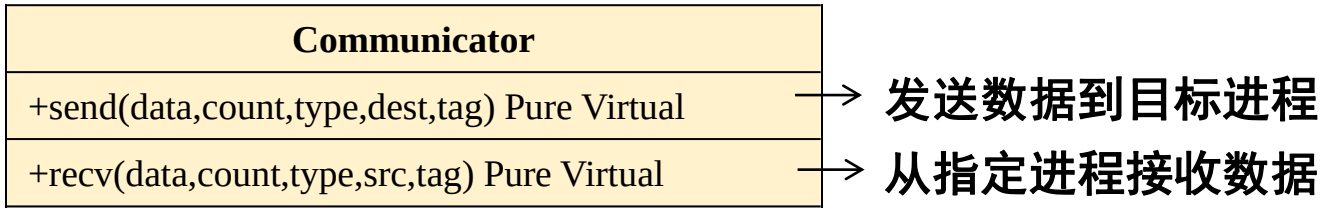
UML图

Application
+Application(name,intra_comm,root)
+register_field(name,ncomp,loc,io,units)
+set_field_function(getter,setter)
+add_interface()
+start_coupling(inter_comm)
+exchange_solution(time,obj)
+stop_coupling()



进程间通信子：Communicator

UML图



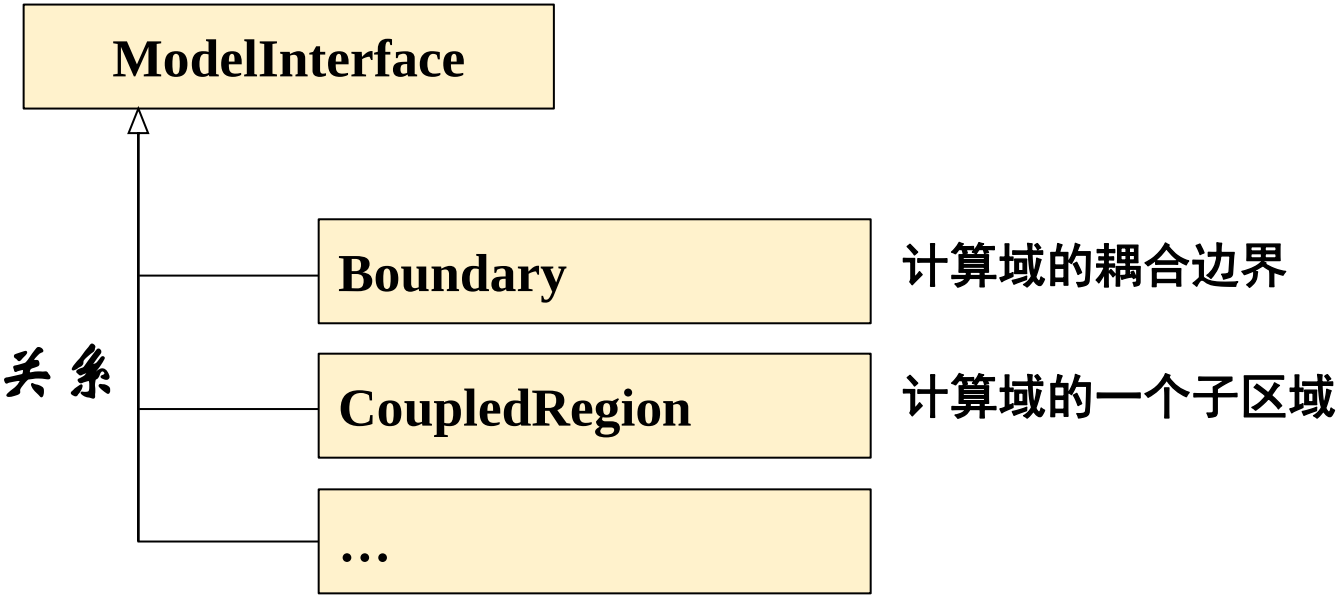
类继承关系

模型接口：ModelInterface

UML图

ModelInterface	
+nnode() Pure Virtual	获取节点个数
+nelem() Pure Virtual	获取单元个数
+register_field(info) Pure Virtual	注册一个场变量
+nfield()	
+get_field(field_name)	

类继承关系



插值算法：Interpolator

UML图

Interpolator	
+add_source_int(model_interface)	添加一个源边界
+add_target_int(model_interface)	添加一个目标边界
+interp_dofs_s2t(ndof, dof_s, dof_t)	插值自由度：源边界→目标边界
+interp_loads_t2s(nload, load_t, load_s)	插值载荷：目标边界→源边界
+compute_coeff(method)	计算插值系数

已实现的插值算法

- method = GlobalXPS：全局IPS/TPS插值，无需单元信息
- method = LocalXPS：局部IPS/TPS插值，无需单元信息
- method = Projection：单元几何投影方法，需要单元信息
- method = Mapping：几何求交算法，用于局部守恒型载荷插值

求解器集成

C/C++、Fortran求解器（demo.cpp）

```cpp
#include "EasyFsi.h"

static Application*     app       = nullptr;
static Communicator*    inter_comm = nullptr;
static MPICommunicator* intra_comm = nullptr;
static Boundary*        bd0       = nullptr;

//-----------------------------------------------------------
// define helper functions for field reading and writing
//-----------------------------------------------------------
//! @brief function used to reading outgoing fields invoked by Application
void get_boundary_field(const Application* app, const Boundary* bd, const char* name, int
ncomp, FieldLocation loc, double* data, void* user_data)
{

//! @brief function used to writing incoming fields invoked by Application
void set_boundary_field(const Application* app, const Boundary* bd, const char* name, int
ncomp, FieldLocation loc, const double* data, void* user_data)
{
```

①包含头文件

②定义各个对象

③定义读取和设置耦合边界场变量的回调函数

```
66  //-------------------------------------------------------
67  // preprocessing
68  //-------------------------------------------------------
69  ④预处理：创建边界、定义耦合变量等
70  void init()
71  {
72      // create communicator used to transfer data between different application.
73      inter_comm = cm_socket_new(true,2,"127.0.0.1",1234);定义用于不同求解器间通信的通信子
74
75      // create MPI communicator used to transfer data between process of this application
76      //     arg0   The MPI communicator
77      //     arg1   Rank of this process in mpi_comm
78      //     arg2   Total process number of mpi_comm
79      intra_comm = cm_mpi_new(mpi_comm, rank, size);定义用于同一求解器不同进程间通信的通信子
80
81      // create application
82      app = app_new("cfd",intra_comm,0);创建应用程序
83
84      // define coupled boundary
85      bd0 = app_add_boundary(app);定义耦合边界
86      // 1) create boundary manually:
87      // bd_add_node(bd0,x1,y1,z1,id1);
88      // ...
89      // bd_add_face(bd,ft1,nn1,nodes1);
90      // ...
91      //
92      // 2) create boundary from Gmsh file:
93      // bd_read_gmsh(bd, "???.msh");
94
95      bd_compute_metrics(bd0, 5.0);
96
97      // define coupled fields:
98      app_register_field(app,"displacement",3,NodeCentered,OutgoingDofs ,"m");
99      app_register_field(app,"force"       ,3,NodeCentered,IncomingLoads,"N");定义耦合物理量
100
101     // set field reading/writing functions
102     app_set_field_func(app,get_boundary_field,set_boundary_field);设置读取/更新边界物理量的回调函数
103
104     // start coupling: get solver infomation
105     app_start_coupling(app,inter_comm);开始耦合：获取各求解器信息
106 }
```

```
107
108     //-----------------------------------------------------------
109     // solving
110     //-----------------------------------------------------------
111     ⑤耦合求解：插值和交换物理量
112     void solve()
113     {
114         // solve this physics one step
115         // ...
116
117         // interpolate and exchange results between applications
118         app_exchange_solu(app,time,nullptr); 插值和交换物理量
119
120         // other operations
121         // ...
122     }
123
124     //-----------------------------------------------------------
125     // postprocessing
126     //-----------------------------------------------------------
127     ⑥后处理：停止耦合并删除各对象
128     void post()
129     {
130         app_stop_coupling(app);
131         app_delete(&app);
132         bd_delete(&bd0);
133         cm_delete(&intra_comm);
134         cm_delete(&inter_comm);
135
136         // other operations
137     }
```

◆ Python（demo.py）

```python
import EasyFsi
from EasyFsi import*

#-----------------------------------------------------------
# define helper functions for field reading and writing
#-----------------------------------------------------------
# function used to read outgoing field of boundary.
#    app: application object
#    bd: boundary object
#    fieldname: name of the field
#    location: location of the field, see FieldLocation
#    values: field data, type=MatView object
#    user_obj: object passed from app.exchange_solution
def get_bound_field(app,bd,fieldname,ncomp,location,values,user_obj):
    # TODO: update values
    for i in range(bd.nnode):
        values[i,0]=???; # update component-0
        values[i,1]=???; # update component-1
        # ...

# function used to write incoming field of boundary.
#    app: application object
#    bd: boundary object
#    fieldname: name of the field
#    location: location of the field, see FieldLocation
#    values: field data, type=MatView object
#    user_obj: object passed from app.exchange_solution
def set_bound_field(app,bd,fieldname,ncomp,location,values,user_obj):
    # TODO: update values
    for i in range(bd.nnode):
        # ... = values[i,0]; # update component-0
        # ... = values[i,1]; # update component-1
        # ...

```

①引入EasyFSI扩展

②定义读取和设置耦合边界场变量的回调函数

③预处理：创建边界、定义耦合变量等

```
#-----------------------------------------------------------
# preprocessing
#-----------------------------------------------------------
# define application
app = Application("PythonSolver");

# define boundary
bd0 = app.add_coupled_boundary()
bd0.name = "bd0"
# create boundary manually:
#bd0.reserve(200,100,400)
#bd0.add_node(x,y,z,unique_id) # define node
# ...
#bd0.add_face(type,nodes)  # define face
# ...
# create boundary from file:
#bd0.load("???.msh")  # read Gmsh file
#bd0.compute_metics(5.0)

# define field
#    arg0  The name of this field
#    arg1  The component number of this field, ≥1
#    arg2  Location of field, NodeCentered or FaceCentered
#    arg3  Input/Output type, see FieldIO
#    arg4  Units of this field
app.register_field("displacement",3,FieldLocation.NodeCentered,FieldIO.OutgoingDofs,"m")
app.register_field("force",3,FieldLocation.NodeCentered,FieldIO.IncomingLoads,"N")
app.set_field_function(get_bound_field,set_bound_field)

# define communicator between applications
#    arg0  True/False, this application is master?
#    arg1  Number of applications for this coupling problem, ≥2
#    arg2  IP address of machine that master application is running.
#    arg3  Port number
#    arg4  Timeout value in second.
inter_comm = SocketCommunicator(False,2,"127.0.0.1",1234,60)

```

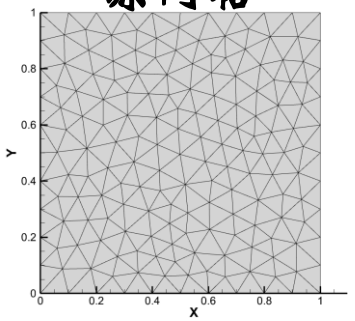
```
76
77  #-----------------------------------------------------
78  # solving
79  #-----------------------------------------------------
80  ④迭代求解
81  # solving
82  dt = 0.001  # timestep size
83  nt = 1000   # timestep number
84  for it in range(nt)
85      # TODO: solve one step
86      # ...
87
88      # exchange solution
89      app.exchange_solution(dt*(it+1), None)插值和交换物理量
90
91      # other post operations
92      #  ...
93
94  # stop coupling when finished
95  app.stop_coupling()
96
97  #-----------------------------------------------------
98  # postprocessing
99  #-----------------------------------------------------
100 ⑤后处理
101 # save results
102 app.save_tecplot("pysolver.res.plt")
```

测试算例1：插值算法（test_it.py）

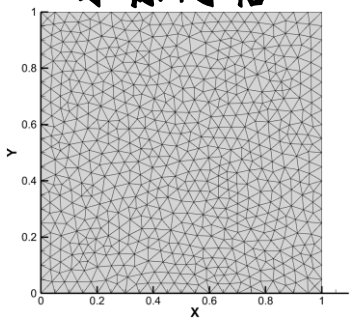
```python
import math
import EasyFsi
from EasyFsi import*

# create FieldInfo
disp_s = FieldInfo("displacement","m",3,FieldLocation.NodeCentered,FieldIO.OutgoingDofs)
disp_t = FieldInfo("displacement","m",3,FieldLocation.NodeCentered,FieldIO.IncomingDofs)
force_s = FieldInfo("force","N",3,FieldLocation.NodeCentered,FieldIO.IncomingLoads)
force_t = FieldInfo("force","N",3,FieldLocation.NodeCentered,FieldIO.OutgoingLoads)

# create boundary
bound_s = Boundary()
bound_t = Boundary()

bound_s.load("fe.msh")
bound_t.load("fv.msh")
bound_s.register_field(disp_s)
bound_s.register_field(force_s)
bound_t.register_field(disp_t)
bound_t.register_field(force_t)

# create interpolator
interp = Interpolator()
interp.add_source_boundary(bound_s)
interp.add_target_boundary(bound_t)
interp.compute_interp_coeff(InterpolationMethod.LocalXPS,20)
interp.save_coefficients("coeff.txt")

# setup field value
disp = bound_s.get_field("displacement")
for i in range(0,bound_s.nnode):
    coord=bound_s.node_coords(i)
    disp[i,2]=math.sin(coord.x*math.pi)*math.sin(coord.y*math.pi)

# do interpolating
interp.interp_dofs_s2t("displacement")
interp.interp_load_t2s("force")

# save results
bound_s.save("fe.plt")
bound_t.save("fv.plt")
```

局部XPS方法

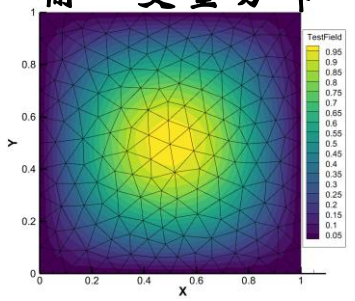
源网格



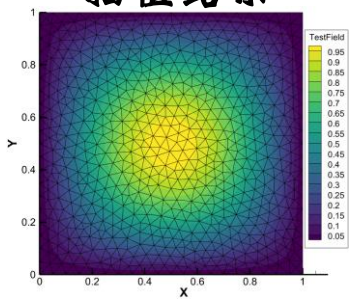
目标网格



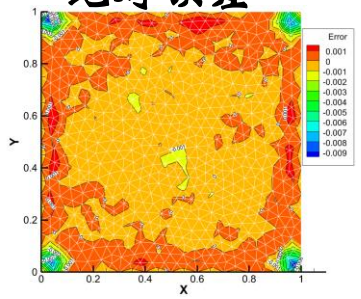
输入变量分布



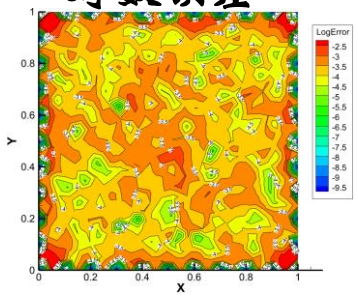
插值结果



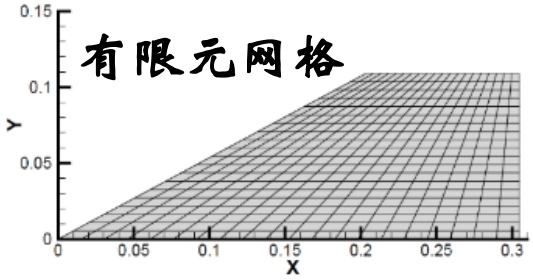
绝对误差

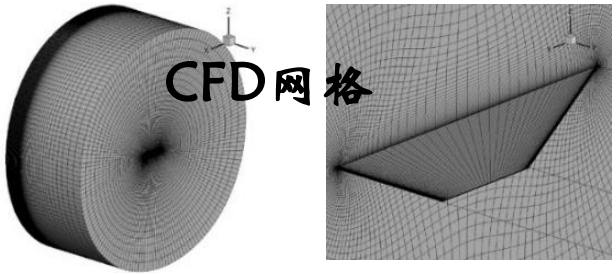


对数误差



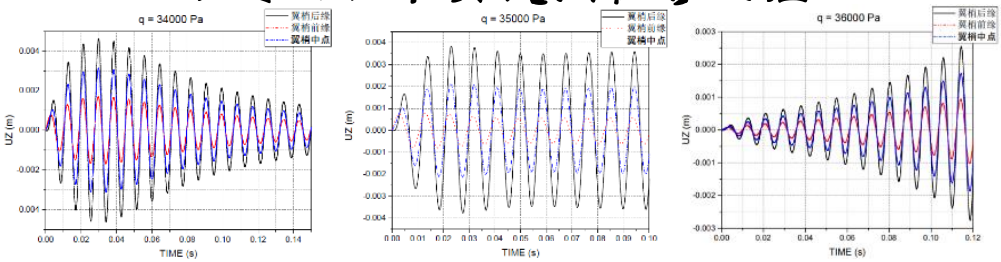
测试算例2：切剪三角翼超声速颤振





模型数据来源	NASA-TN-D-2038, Model-2B
CFD求解器	EasyCFD
CSD求解器	EasyMSP
CFD网格	82万，无黏
计算硬件	2个节点：2*Intel Xeon E52650 网络：56GB/s Infiniband

不同动压下关键点位移响应

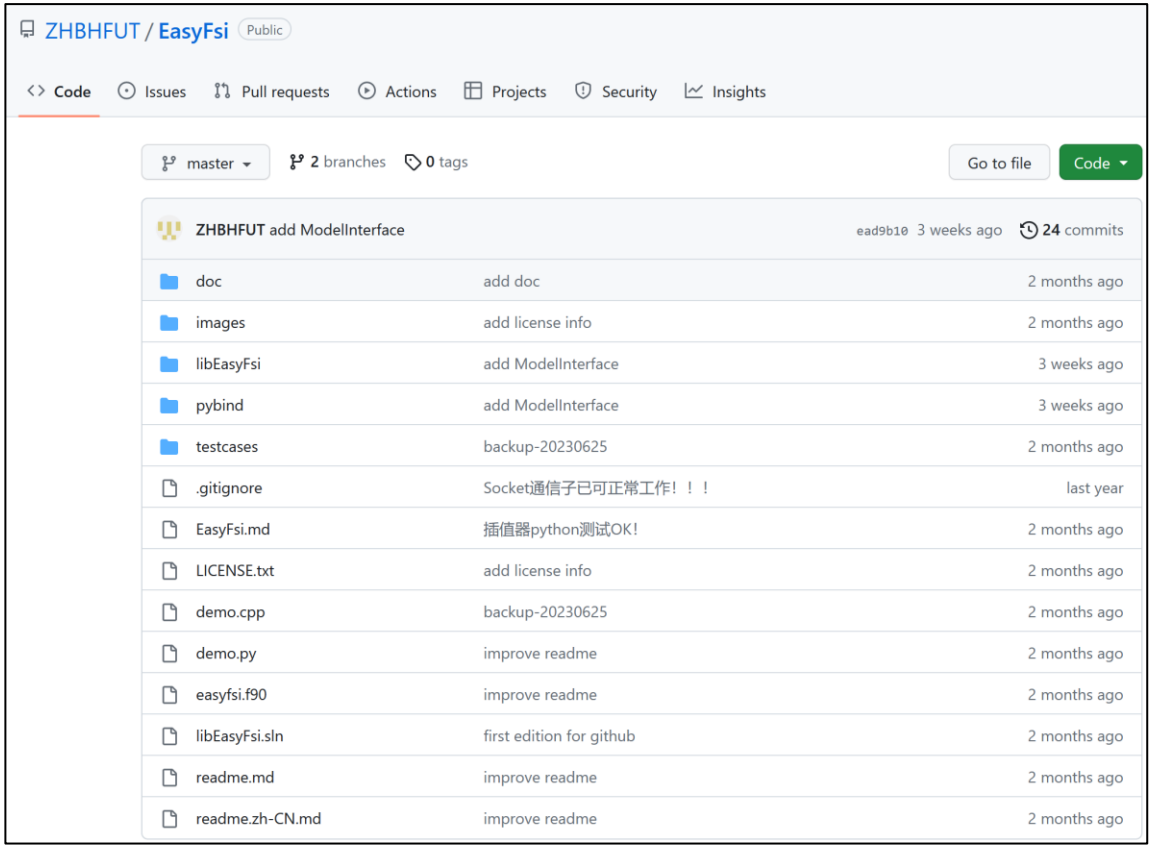


实验颤振动压：33995Pa

计算效率对比

求解器	时间/s	参数说明
EasyCFD	682	82万网格，32核MPI并行，9万迭代步
EasyMSP	1.6	5阶模态叠加，单核，3000步
EasyCouple	1.5	约1.2万次RPC调用
EasyFSI	0.6	约0.6万次Socket调用

代码托管：https://github.com/ZHBHFUT/EasyFsi



欢迎克隆使用、提出宝贵意见，盼共同开发

谢谢！请各位专家批评指正！