International Journal of Hybrid Information Technology
Vol.9, No.1 (2016), pp. 301-318
http://dx.doi.org/10.14257/ijhit.2016.9.1.26

HACP2: The Pre-processing Software Tool for the Hybrid Atomistic-Continuum Coupling Simulation

Qian Wang[1, 2], Wenjing Yang[1, 2,1], Fu Li[1, 2], Xiaoguang Ren[1, 2] and Yuhua Tang[1, 2]

College of Computer
National University of Defense Technology, Changsha, China
State key Laboratory of High Performance Computing
National University of Defense Technology, Changsha, China
{wangqian, wenjing.yang, renxiaoguang}@nudt.edu.cn
{lifunudt, yhtang62}@163.com

Abstract

Fluid simulation is an important application of High Performance Computing. The hybrid atomistic-continuum coupling fluid simulation can effectively handle the contradictory of the reduction of the simulation scale and the increase of the computation load when trying to improve the accuracy. The pre-processing procedure of the coupling simulation is quite important in the whole simulation process and requires the support of an efficient, user-friendly interface. Under the demand of efficient coupling fluid simulation, in this paper we design and implement a visualized pre-processing framework HACP2 based on SALOME, for the unified molecular dynamic-computational fluid dynamics modeling and apply it into the coupling simulation process. The experimental verification indicates that our HACP2 framework can offer efficient, easy-to-use pre-processing for the hybrid atomistic-continuum coupling simulation, and effectively improve the efficiency of the coupling simulation.

Keywords: *HPC, fluid simulation, hybrid atomistic-continuum, pre-processing*

1. Introduction

High Performance Computing (HPC) technology has made computing been the third scientific method, following theory and experiment. With scientific research and industrial fabrication demand, the simulation of fluids become an important application of HPC. Generally, the accuracy of the fluid simulation is in inverse proportion of the scale of the model abstraction [1]. When the scale of simulation decreases to certain degree, the computing load would be getting so heavy that sometimes even Milky-way 2 cannot meet the computational demand when the characteristic scale approaches to infinitesimal. How to get satisfied results with current computational power becomes a challenge [2]. In order to simulate physical problems with a large length scale as well as to capture the microscopic physical phenomena, the so-called hybrid atomistic-continuum (HAC) multi-scale simulation method is proposed [3-6].

The key idea of the hybrid atomistic-continuum simulation method is the domain decomposition of the simulation area, thus the pre-processing, which generates input data for the equation solver, turns out to be an important component of the whole simulation software. In macroscopic computational fluid dynamics, the mesh generation, parallelism partition and boundary definition of simulation domain are done in pre-processing; in

[*] Corresponding author.
E-mail address: wenjing.yang@nudt.edu.cn

molecular dynamics, the pre-processing finishes the configuration of physical parameters and the simulation box.

The pre-processing tool is a necessity in the coupling of the atomistic simulation and the continuum simulation. Users of this tool are usually researchers in specific fields where computer science is not included. The absence of a user-friendly pre-processing tool would cause low efficiency, while a visualized tool of good availability, low threshold of relevant technique background can contribute to the improvement of efficiency and the reduce of work and cost. The currently available pre-processing tools, which always in single scale, cannot meet the demand of the hybrid atomistic-continuum coupling simulation.

In this paper we present a hybrid atomistic-continuum coupling simulation oriented pre-processing tool based on SALOME. We analyze the basic framework, execution mechanism and development method of SALOME software. Based on the analysis, we design and implement a user-friendly and effective pre-processing tool. The contributions of this paper are as follows:

1. Associated with the demand of the hybrid atomistic-continuum coupling simulation and the analysis of SALOME, we design the main framework of the user-friendly pre-processing tool HACP2.

2. We design and implement the Hybrid Atomistic-Continuum Pre-Processing tool. We implement the user interface and the core functions of the HACP2 tool, which could simplified the operations and improve the efficiency of the pre-processing.

3. We verify the HACP2 tool through experiment. The result indicates that our pre-processing tool is able to offer efficient and user-friendly pre-processing for users.

2. Background

In this section, we present the general pre-processing procedure for the hybrid atomistic-continuum coupling simulation and review the current research status of the pre-processing tools for the HAC coupling simulation.

2.1. Pre-Processing of the Hybrid Atomistic-Continuum Coupling Simulation

The hybrid atomistic-continuum coupling simulation process can be divided into three basic stages as *pre-processing*, *simulating* and *post-processing*. Pre-processing is separated from the core simulating process and post-processing in the time-domain as shown in Figure 1.

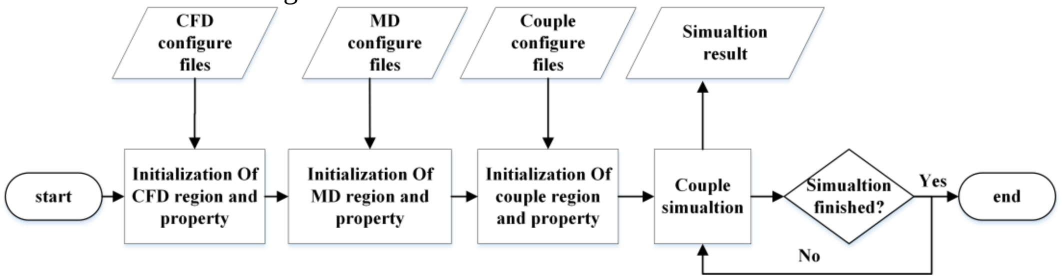


Figure 1. Flowchart of the Hybrid Atomistic-Continuum Coupling Solver

We present a brief introduction to the pre-processing process taking the simulation of the Couette flow as an example. The Couette flow refers to the viscous fluid flow between two mutually moving plates in parallel as shown in Figure 2, and includes the following steps:

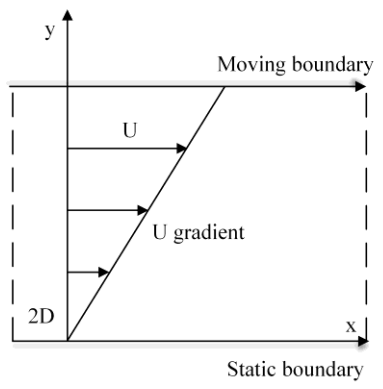


Figure 2. Illustration of the Couette Flow

Firstly, the users has to define the geometric region, namely to determine which part of the region should be calculated using macro CFD methods (CFD region) and which part need to be calculated with micro MD methods (MD region). In the case of the Couette flow, we can simply divide the whole space into two subdomains, as shown in Figure 3.

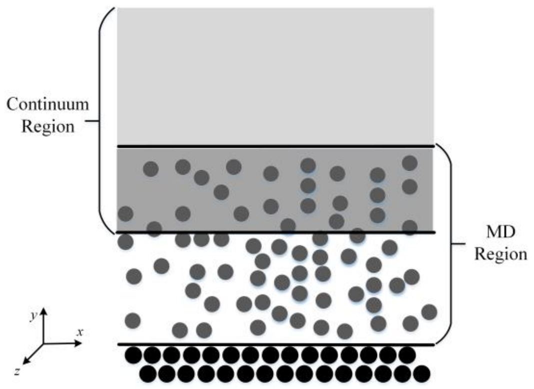


Figure 3. Partitions of the Overall Space into Continuum and MD Regions

Secondly, the users has to further process the CFD region. Considering using OpenFOAM as the CFD computing platform, this region must be organized into an OpenFOAM-supported format and the corresponding boundary conditions must be specified. The mesh can be generated with either the self-designed tool of OpenFOAM or others.

Thirdly, the users has to further process the MD region. Considering using LAMMPS to conduct the MD simulation, the information of the MD region is required. Since there are no suitable visualized configuration tool for the MD simulation, experts in this field are demanded to extract the complex configuration of the MD region into a simple configuration profile.

2.2. Research Status of pre-Processing Tools for the HAC Coupling Simulation

There are many sophisticated pre-processing software, but most of them are of single-scale rather than multi-scale. For the CFD pre-processing, ICEM CFD, PointWise and Gridpro are commonly used for mesh generation [7-9], either for the solid or the liquid simulation region. Some software, like Gridpro, can only generate structured mesh, while TGrid is professional for unstructured mesh generation, which is independent of the geometry complexity and size constraints. For MD, there are also some mature pre-processing software, *i.e.*, OVITO [10], Moltemplate and VMD TopoTools, *etc*. These pre-processing tools help users to automatically build molecular topologies, i.e., lists of bonds, angles, dihedrals, etc. So far as we know, there is no available unified pre-processing tool for the hybrid atomistic-continuum coupling simulation.

3. Framework of the Pre-Processing Software Tool HACP2

In this section, we firstly introduce the core modules of SALOME, and then present the detailed design of our HACP2 software tool.

3.1. Introduction of SALOME Software

SALOME is an open-source software that provides a generic pre-processing platform for the numerical simulation [11]. It is based on an open and flexible architecture made of reusable components. SALOME applied the CORBA technology and modeling methods of the distributed system in the software architecture [12].

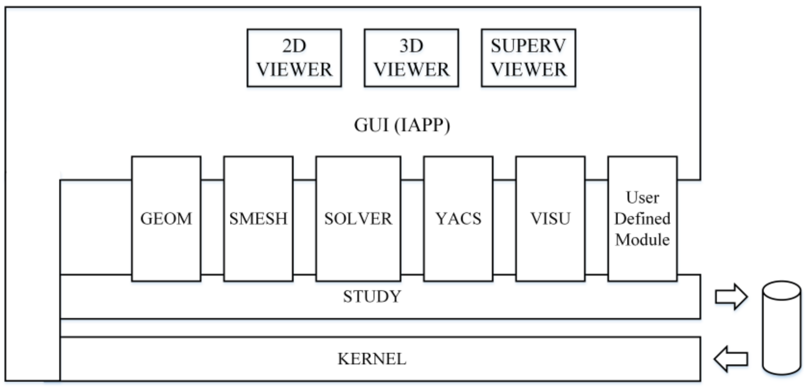


Figure 4. The Software Framework of SALOME

As shown in Figure 4, SALOME integrates lots of pre-processing modules of CFD [13-14]. The KERNEL module provides an available interface for integration of other modules. The GUI module offers a visible interface of data operations for users. Geometric models are produced in the GEOM module. The SMESH module generates meshes for geometric objects with various meshing algorithms. Visualize operations of data are performed in the VISU module [15]. All these modules are independent completely and located in the root folder of SALOME in the form of software packages. Modules perform information communication and data exchange with others through the core module using the CORBA protocol as shown in Figure 5.

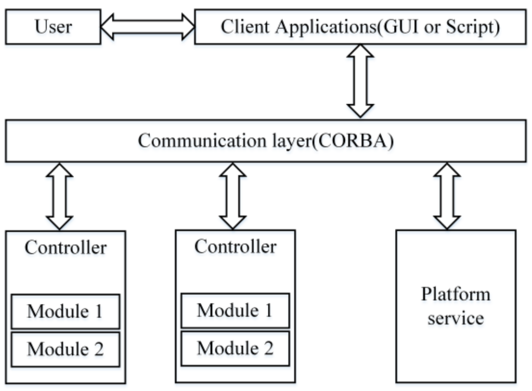


Figure 5. Communication between Modules

3.2. Design of the HACP2 Architecture

Considering the business processes and requirements, the architecture of the HACP2 (Hybrid Atomistic-Continuum Pre-Processing) tool based on SALOME platform is proposed in this paper. It consists of the GEOM module and the SMESH module integrated in SALOME. The HACP2 tool utilizes the modules in SALOME without any changes and adds some new pre-processing modules. The HACP2 separates tasks reasonably and coordinates those existing modules to perform the hybrid atomistic-continuum coupling simulation in pre-processing procedure, which greatly improves the efficiency of simulation. Figure 6 shows the architecture of the hybrid atomistic-continuum pre-processing tool.

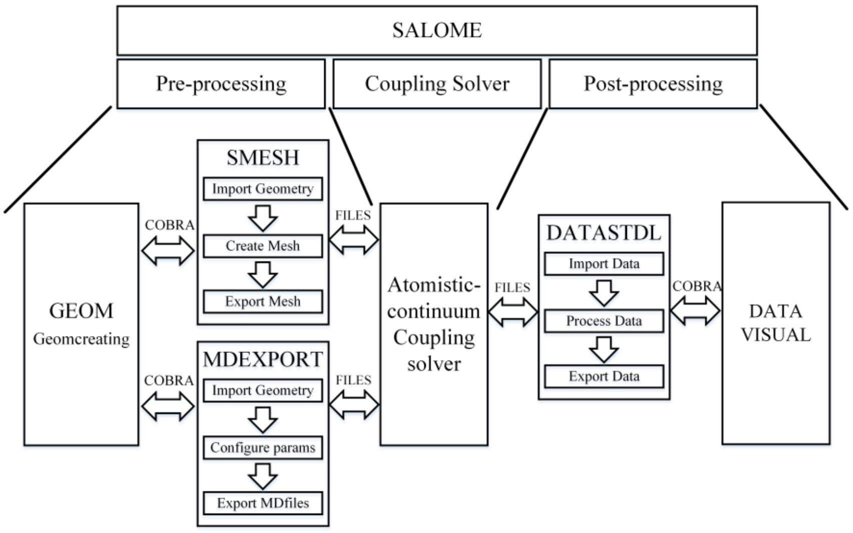


Figure 6. Framework of the HACP2 Tool

The pre-processing mainly includes the following procedures as shown in Figure 7.

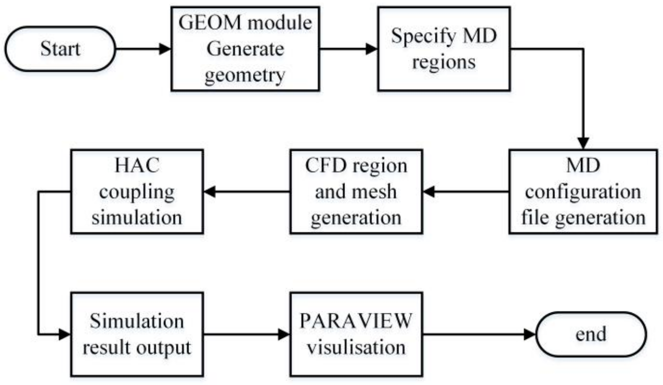


Figure 7. Hybrid Atomistic-Continuum Coupling Simulation Flow Chart

Firstly, definition of the geometry region for simulation utilizing the GEOM module. A complex two-dimensional and three-dimensional modeling could be performed in the GEOM module. The GEOM module also supports the import of geometry models even derived from other advanced geometric modeling tools.

Secondly, designation of the computation region. The MD computation region must be rectangle in this work. The designation of computational region is also performed in the GEOM module. The MD region is generated using simple graphics generation tools. The CFD and MD computation regions are separated by means of difference methods.

Thirdly, generation of configuration files for the MD region. The main processes is: the transmission the designated MD computation region from the GEOM module as the first step, the following saving the geometric graphics as data structure of other HACP2 modules and then the configuration of the coupling width of the MD region and other information and the last dealing with the imported configuration information and geometric region, finally exporting the MD computational configuration files.

Fourthly, mesh generation of the CFD computational domain. This procedure is conducted on the mesh generation module named SMESH. Due to the extraordinary complexity of the mesh generation algorithm, we utilize the SMESH generation module, which is a component of SALOME platform. Firstly, geometrical model of the CFD region is transferred from the GEOM module to the SMESH module. Then, particular mesh generation algorithm is applied to the geometrical model corresponding to the CFD simulation region with the help of the SMESH module. Lastly, the mesh information is exported into standard format.

Finally, solving with the hybrid atomistic-continuum coupling solver. The solver will generate CFD and MD simulation results separately. As this part beyond the scope of this work, the usages or operations with the simulation results will not be discussed.

3.3. Design of the HACP2 Time Sequence

In this section, we design the time sequence of our HACP2 tool. The time sequence of the pre-processing includes two kernel procedures, *i.e.*, data acquisition from the GEOM module and the MD configuration file output [16-17]. Focusing on these two kernel procedures, we create the sequence diagram in Figure 8.

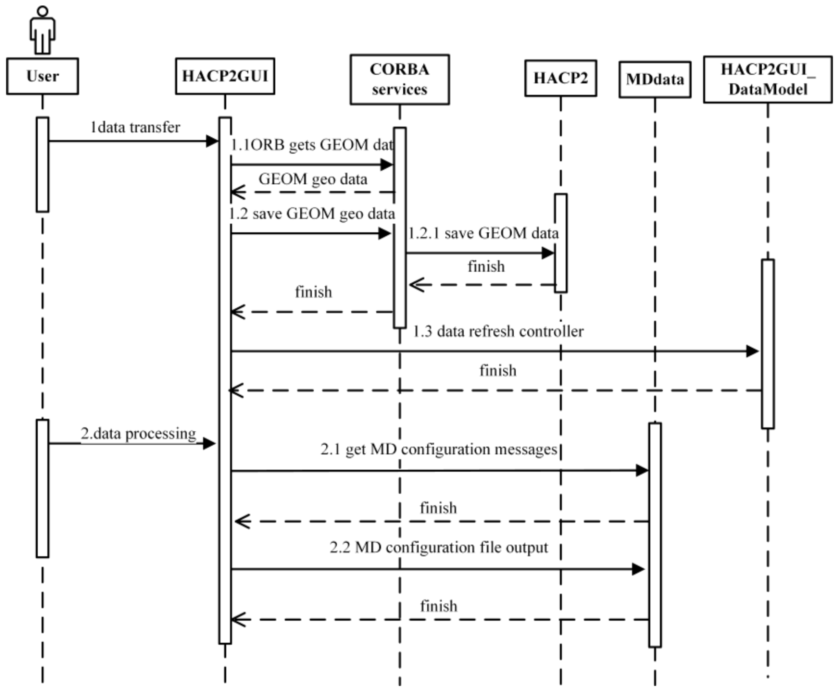


Figure 8. HACP2 Sequence Diagram

3.4. Design of the HACP2 Classes

Base on the sequence of our HACP2 tool mentioned in the previous section, we present the associated classes of the HACP2 tools in this section, which include the classes of the key sequences as shown in Figure 9 and some assistant classes for proper functioning.

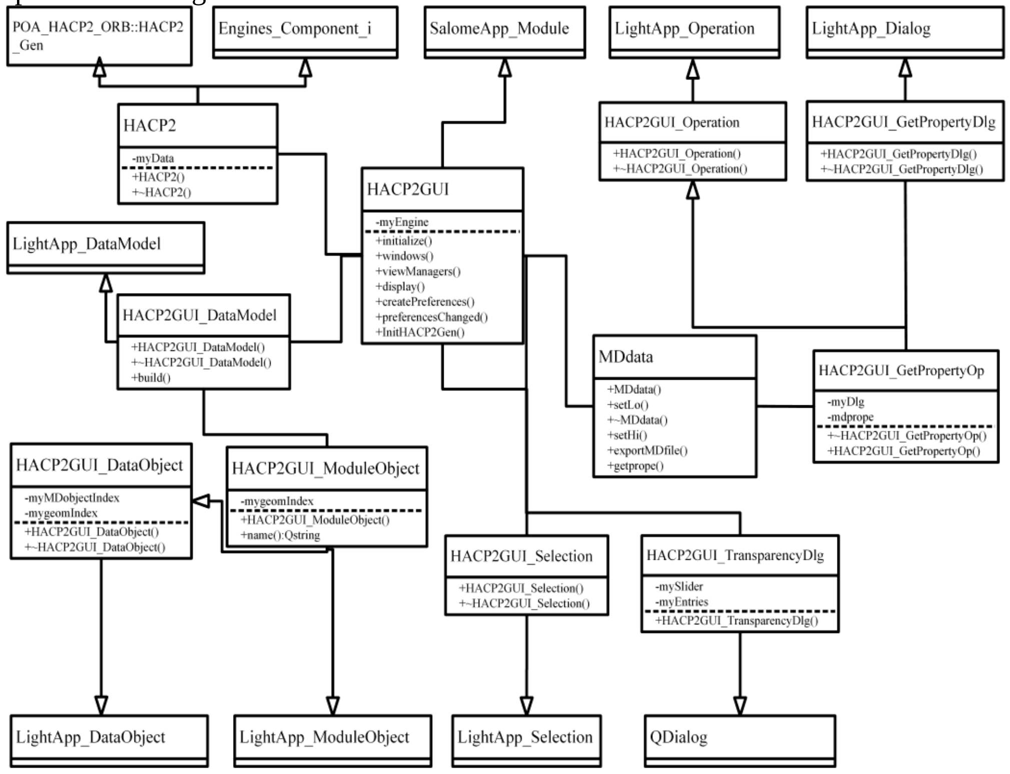


Figure 9. HACP2 Class Chart

The kernel class is the *HACP2GUI* class which inherits from the *SalomeApp_Module* class. It communicates with user for the configuration of the coupling simulation. The main configuration process is described as follows:

Data acquisition from GEOM module: user calls the *OnRetrieve()* function in the *HACP2GUI* class. Firstly, the *OnRetrieve()* function calls the CORBA service of getting data from the GEOM module, this service will return the GEOM object of the MD region; Seconly, the *OnRetrieve()* function gets the GEOM objet through the *setData()* interface provide by CORBA and save it to the core data structures; Finally it calls the *build()* function in the *DataModel* class to refresh the module data in the browser.

MD configuration file output: user calls the *OnProcessData()* function of the *HACP2GUI* class. Firstly, in the *OnProcessData()* function, it instantiates the *MDdata* class and calls the *getprope()* function of the *MDdata* class to get the configuration message of the MD region; Secondly, the *getprope()* function instantiates the *HACP2_GetPropertyOp* class and calls the *getprope()* function of the *HACP2_GetPropertyOp* class to popup dialog which gets the configuration messages from users; Finally, the *OnProcessData()* function calls the *exportMDfile()* function of the *MDdata* class to output the configuration file of the MD region.

4. Implementation of the pre-Processing Software Tool HACP2

In this section, we present the detailed implementation of our HACP2 software tool, including the implement of CORBA engine, graphic user interface and data structures.

4.1. Definition and Implement of the CORBA Engine

The definition of the CORBA engine is implemented in HACP2.idl, which is in the *idl* folder. HACP2.idl just defines but not implements the interface. In Table 1 gives the definition of the interface of HACP2.idl [18].

Table 1. The Pseudo-Code of *HACP2_ORB*

HACP2_ORB

```
1:  Define Property structure
2:  Define the core data structure MDobject
3:  Define MDobject queue,  MDobjectList
4:  Define interface HACP2_Gen
5:  Define interface function setData()
6:  Define interface function getData()
7:  Define interface function processData()
```

The *Property* structure contains the configuration information of the MD region, such as the fluid density, the solid density, the width of the coupling, *etc*. The *MDobject* structure of constitutes the core data structures of the HACP2 tool, which contains the geometric object *GEOM::GEOM_Object* defined by the *Property* structure and the GEOM module. The main function of the *MDobject* is to store the MD region geometric object transmitted by the GEOM module and the configuration information of the object.

The *HACP2_Gen* inherits from *Engines::EngineComponent* that defined by the KERNEL module which is the engine of the HACP2. The functions defined in *HACP2_Gen* constitutes the core service and interface of the HACP2 tool. In other part of the HACP2 tool, such as GUI, the core data structures can be operated by instantiating this engine. The *HACP2_Gen* defines three interface functions: the *setData()* interface writes data to the data index, the *getData()* interface gets data from the data index and the *processData()* interface processes these data.

In Figure 9, it defines the classes associated in the HACP2 tool. Focusing on the engine implementation, we present the central part classes in Figure 10.

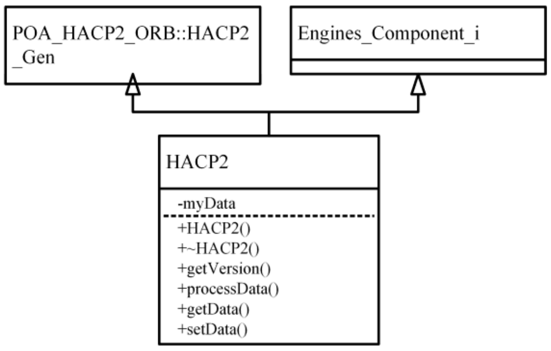


Figure 10. Implementation of the HACP2 Engine

The engine includes the *myData* property, which the typical *map* type in the standard *std* functional library, builds a *long* type to index the *HACP2_ORB::MDobjectList** type, which can dynamical expend its size, and each instantiation of object gets the unique *studyID* from *myData* corresponding to one object in *HACP2_ORB::MDobjectList**. The *myData* property is the central data structure of the engine to save the associated data. The engine provides interfaces for other modules or the *HACP2GUI* class to call the functions of accessing and manipulating of data in *myData*.

As we have mentioned, The *HACP2_Gen* defines three interface functions. We describe the implementation of these three functions separately.

The *setData()* function uses the *studyID* which is the index of the central data structure *myData* to put the input *theData* parameter to the end of the index queue. The pseudo code of the *setData()* function is shown in Table 2.

Table 2. The Pseudo-Code of *Setdata()*

setData(CORBA::Long studyID, const HACP2_ORB::MDobjectList& theData)

```
1: Declare the cur pointer to std::map
2: Using studyID in myData, assign cur
3: Judge whether exists studyID in myData, if not, create studyID
4: Add theData to the item indexed by studeID in myData
```

The *getData()* function uses the *studyID* to index the *theData* in the core data structure and transfers the indexed data to the *outData* parameter. The pseudo code of The *getData()* function is presented in Table 3.

Table 3. The Pseudo-Code Of *Getdata()*

getData(CORBA::Long studyID, HACP2_ORB::MDobjectList_out outData)

```
1: Judge whether exists studyID in myData
2: If so, transfer the indexed data to outData
3: If not, return error
```

The *processData()* function uses the indexed data from the *outData* parameter, through the OCCT technique, assesses the GEOM object and gets the configuration of the rectangle region. The pseudo code of The *processData()* function is shown in Table 4.

Table 4. The Pseudo-Code of *processData()*

processData(GEOM::GEOM_Gen_ptr aGeomEngine, CORBA::Long studyID)

```
1: Judge whether exists studyID in myData
2: If so, transfer the indexed data to temp variable data
3: Traverse all items in data queue and execute 4-7
4: Set ismd to 1
5: Declare a GEOM_Client object, aGEOMClient
6: Get the TopoDS_Shape of geomobject through aGEOMClient
7: Through TopExp_Explore, traverse the sub shape messages of TopoDS_Shape
8: If not, return error
```

4.2. Implementation of the Graphical User Interface

The *HACP2GUI* class contains not only the functions used to create, active and deactivate modules in the module lifecycle, but also the interface functions used to interactive with users. The core functions of the HACP2 tool are implemented in the related function of the *HACP2GUI* class. The graphical module includes the initialization function *Initialize()*, the module activation function *activateModule()*, the module deactivation function *DeactivateModule()*, the data transmitting function *OnRetrieveData()* and the data processing function *OnProcessData()*.

When the HACP2 tool is activated (after choosing *module* in the pull-down list in the main window of SALOME or after pressing the start button of module), the compiled link libraries of module are loaded to the main memory; The *createModule()* function is executed to active the object related to GUI on the first step; Once the GUI object is created, it takes control of the program and creates menu, object browser, visualization window, *etc.*; Then the module loading process completes and waits for user operation.

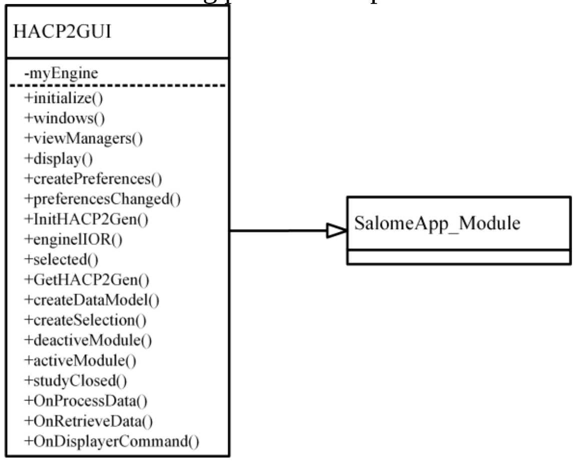


Figure 11. HACP2 GUI Class Chart

The *HACP2GUI* class, as shown is Figure 11, inherits from the *SalomeApp_Module* class of the KERNEL module. We presents the detailed instantiation and destroy functions of the *HACP2GUI* class.

Initialize(): when the HACP2 tool is activated, the *Initialize()* function is called. The object of the *HACP2GUI* class only calls it when the object is created. In the *Initialize()* function, it completes the creation of *Action*, *Menus*, *Toolbars* and *Right Click Menu*. The creation of these tools and the generation of graphical interface are implemented by QT visualized programming technique.

activateModule(): it is called when any module is activated. Its main function is to visualize the menus and toolbars that created by the *Initialize()* function.

DeactivateModule(): contrary to the *activateModule()* function, it is called when any module is destroyed. Its main function to make the menus and toolbars invisible.

GetHACP2Gen(): it returns a static variable so that other GUI classes can access the services in the COBRA engine. This static variable can directly use the instantiation of *Salome_Application*, invoke the module loading function *FindOrLoad_Component()* which is existed in *LifeCycleCORBA* interface and complete the registration of modules.

The *HACP2GUI* class interacts with users and completes the central functions of the HACP2 tool which are getting data from the GEOM module and outputting the configuration file of the MD regions. Through two functions, the *OnRetrieveData()* function and the *OnProcessData()* function, the *HACP2GUI* class deals with data processing. We presents the detailed implementations of these two functions.

The *OnRetrieveData()* function is getting the geometry of the MD regions from the GEOM module through the interface defined by the CORBA protocol. It has the data structure of *GEOM::GEOM_Object*, saves the obtained object of *GEOM::GEOM_Object* into its own data structure and then completes the data retrieving. This function is implemented by the CORBA programming technique and the OCCT technique. The pseudo code of the *OnRetrieveData()* function is shown in Table 5.

Table 5. The pseudo-code of *OnRetrieveData()*

OnRetrieveData()

```
1:  Declare the HACP2 engine and point to current pointer
2: Declare the SalomeApp_Application object, app, point to the current application
3:  Get the pointer comp of the GEOM module
4:  Declare and get the engine of the GEOM module
5: Declare the SalomeApp_Study object appStudy, point to the current application
6:  Declare studyID and set current module id to it
7:  If not GEOM, load the GEOM module
8:  Declare the MDobjectList_var object outData
9:  Declare the Study object study, point to the GEOM module
10:  Declare the ListIO object, selectedgeom
11:  Declare the selection controller aSel,  set the object chosen by app
12:  Set the object chosen by the GEOM module to selectedgeom
13:  Judge whether selectedgeom is empty
14:  If not, declare the anIO handler
15:  Traverse the GEOM objects in selectedgeom
16:  Add the GEOM object to the module through setData()
17:  Refresh the browser
```

The *OnProcessData()* function is providing the visualization configuration interface to users to get the configuration of the MD region, getting the configuration messages, dealing with the geometry object obtained from the GEOM module and outputting the configuration file ***coupledProperties***. Similar to the *OnRetrieveData()* function, the implementation of the *OnProcessData()* function is achieved by the service interface *processData()* and other methods packed in the *MDdata* class. The pseudo code of the *OnProcessData()* function is shown in Table 6.

The *OnProcessData()* function instantiates the *MDdata* class, and calls the associated methods of the *MDdata* class. The relation between the *MDdata* class and other classes is shown in Figure 12.

Table 6. The Pseudo-Code of *OnProcessData()*

OnProcessData()

```
1:  Declare the HACP2 engine and point to current pointer
2:  Declare the SalomeApp_Application object, app, point to the current application
3:  Declare studyID and set current module id to it
4:  Call the processData() dealing with data
```

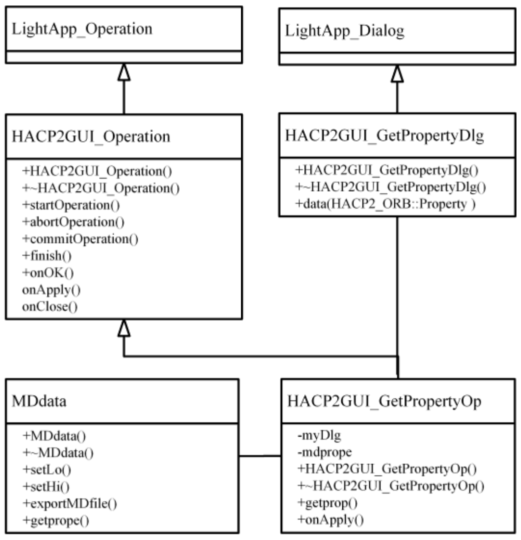


Figure 12. Class Chart of Getting MD Configuration Messages

The getprope() function gets the configuration message of the MD region and saves in the private variable *mdproperty*. The implementation of this function is instantiating the *HACP2GUI_GetPropertyOp* class which can obtain the popup dialog and interacting with users to get the configuration messages of the MD region.

The *HACP2GUI_GetPropertyOp* class is inherited from the *HACP2GUI_Operation* class and the *HACP2GUI_Operation* inherits the basic functions in the *LightApp_Operation* class. The *LightApp_Operation* class is the basic class defined in the KERNEL module of SALOME, which includes many operation functions, *i.e., startOperation(), abortOperation(), commitOperation(), etc.* The *HACP2GUI_GetPropertyOp* class owns the instantiation of the private variable *HACP2GUI_GetPropertyDlg, myDlg*. It is a dialog object that defines the interaction dialog with users, including the content and layout of the dialog. Combining the dialog with some associated operations, it completes the configuration messages retrieving. The *exporttoMDfile()* function combines the configuration of the MD region and the geometry of rectangular into the output file *coupledProperties*.

4.3. Organization and Implementation of the Data Structures

Data can be organized in different forms, and in the HACP2, we use index and queue to organization data. The *DataModel* class uses tree structure to organize data, and we can get the root data through the *root()* function. Root data is the data that has the highest level of abstraction. Generally, the root data of a module is the module itself. Here we need to introduce the object returned by the *DataModel* class, which is call *Data* Object. The function of *Data* Object is to provide a common interface that can be accessed by other objects. This interface actually shields the difference of data operations, and only the *DataModel* class knows its true data structure and operation function implementation. This shield enhances the independence between objects and increases the scalability and robustness of modules.

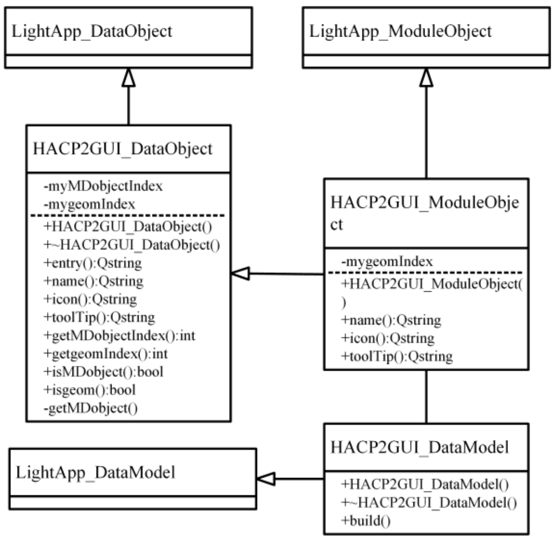


Figure 13. Class Chart of Data Model

The class chart of classes associated with the *DataModel* class is shown in Figure 13. The kernel functions of the *DataModel* class are the *build()* function and the *getMDobject()* function of the *DataObject* class.

The *build()* function of the *DataModel* class organizes the data structure as the tree structure and overloads the *build()* function of the KERNEL module in SALOME. The pseudo code of the *build()* function creation is shown in Table 7.

Table 7. The pseudo-code of *build()*

build()
1: Declare the *ModuleObject* object, *modelroot*, point to the tree root
2: If not, create it and assign to *modelroot*
3: Declare HACP2 engine, point to the current module
4: Create *studyID* to get the id of the current module
5: *getData()* to get the indexed data using *studyID*
6: Traverse object, add to the tree structure

The *DataObject* class uses the *getMDobject()* function to get the engine data of the HACP2 tool and provides data support for other functions in the *DataObject* class. The pseudo code of the *build()* function creation is shown in Table 8.

Table 8. The Pseudo-Code of *getMDobject()*

getMDobject()
1: Declare HACP2 engine, point to the current module
2: Create *studyID* to get the id of the current module
3: *getData()* to get the indexed data using *studyID*
4: Return the obtained MD data

5. Experimental Verification

This section is mainly about the results of the HACP2 tool through snapshots of the software. The example we take is the computation of the Couette benchmark flow mentioned above [19-20]. The verification of our HACP2 tool including the activating of the GEOM module, the initializing of the whole simulation domain, the specification of spatial decomposition and the exportation of the configuration files.

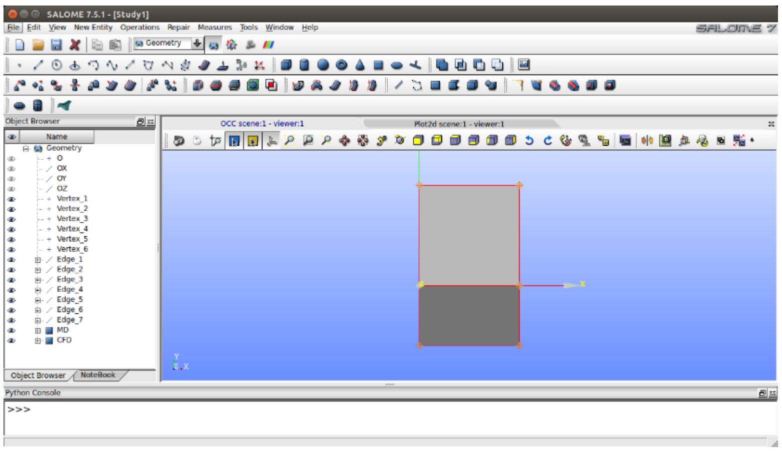


Figure 14. Modeling with the GEOM Module

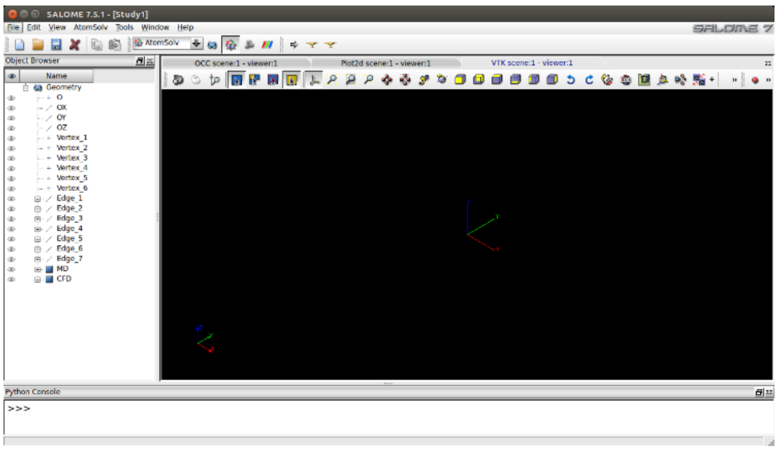


Figure 15. Activation of the HACP2 Tool

Figure 14 illustrates the modeling of the computing domain with the GEOM module of the HACP2 tool. We apply CFD on the upper half of the area while MD used on the bottom half.

Figure 15 shows the process of activating the HACP2 tool by click the icon of the HACP2 module in SALOME. The module is then initialized but no data structure is built at that time. Three functions could be seen: *data transferring*, *information configuration* and *output configuration data*.

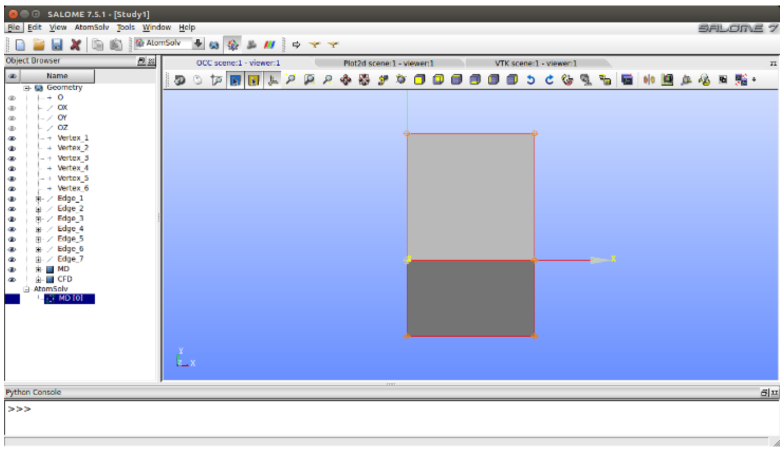
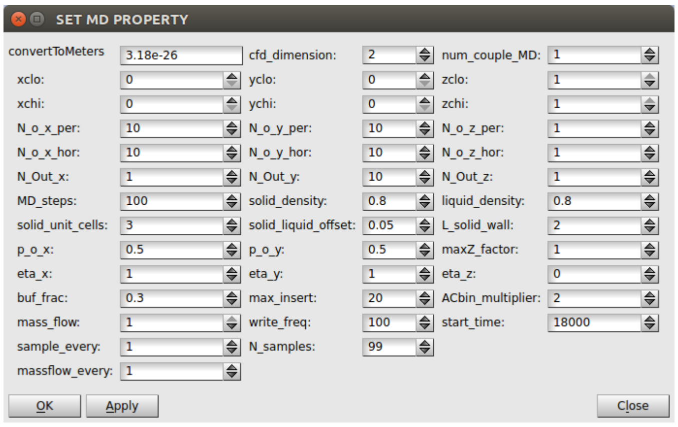


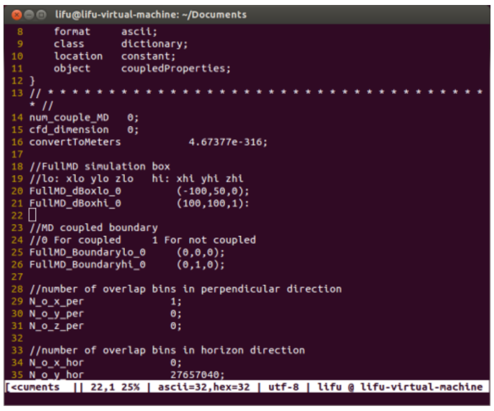
Figure 16. Transferring Data from GEOM Module

Figure 16 is the process that transferring the data of the MD region generated in the GEOM module within the HACP2 module, saving it in the form of its own data structure, and printing it in the data browser.


SET MD PROPERTY

convertToMeters	3.18e-26	cfd_dimension:	2	num_couple_MD:	1
xclo:	0	yclo:	0	zclo:	1
xchi:	0	ychi:	0	zchi:	1
N_o_x_per:	10	N_o_y_per:	10	N_o_z_per:	1
N_o_x_hor:	10	N_o_y_hor:	10	N_o_z_hor:	1
N_Out_x:	1	N_Out_y:	10	N_Out_z:	1
MD_steps:	100	solid_density:	0.8	liquid_density:	0.8
solid_unit_cells:	3	solid_liquid_offset:	0.05	L_solid_wall:	2
p_o_x:	0.5	p_o_y:	0.5	maxZ_factor:	1
eta_x:	1	eta_y:	1	eta_z:	0
buf_frac:	0.3	max_insert:	20	ACbin_multiplier:	2
mass_flow:	1	write_freq:	100	start_time:	18000
sample_every:	1	N_samples:	99		
massflow_every:	1				

Figure 17. Dialog for Configuring the MD Field Information


lifu@lifu-virtual-machine: ~/Documents

```
    format     ascii;
    class      dictionary;
    location   constant;
    object     coupledProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* //
num_couple_MD   0;
cfd_dimension   0;
convertToMeters           4.67377e-316;

//FullMD simulation box
//lo: xlo ylo zlo   hi: xhi yhi zhi
FullMD_dBoxlo_0         (-100,50,0);
FullMD_dBoxhi_0         (100,100,1):

//MD coupled boundary
//0 For coupled     1 For not coupled
FullMD_Boundarylo_0     (0,0,0);
FullMD_Boundaryhi_0     (0,1,0);

//number of overlap bins in perpendicular direction
N_o_x_per              1;
N_o_y_per              0;
N_o_z_per              0;

//number of overlap bins in horizon direction
N_o_x_hor              0;
N_o_y_hor              27657040;
[<cuments  || 22,1 25% | ascii=32,hex=32 | utf-8 | lifu @ lifu-virtual-machine
```

Figure 18. CoupledProperties File

Figure 17 is the dialog that conFigured the key words of the MD region cofiguration information in the HACP2 module, including *CFD_dimension*, *solid_density*, *etc*. This dialog is popped up when clicking on the button of "*information configuration and output of the MD field*". The default values are the values of the field configuration information in this example.

After closing the dialog, the HACP2 module will combine the field information from the GEOM module and the MD region configuration information Configured by users, and generate the *coupleProperties* file, which has the content showed in Figure 18, and then finish the pre-processing configuration procedure.

6. Conclusion

This paper designs an efficient pre-processing tool that based on SALOME for the coupling simulation of computational fluid dynamics and molecular dynamics simulation. Associated with the pre-processing of the coupling simulation of the continuum and atomistic scale, we design and implement the HACP2 tool based on SALOME. The experimental verification indicates our HACP2 tool is able to significantly improve the efficiency of the pre-processing of the hybrid atomistic-continuum coupling simulation.

Acknowledgments

This paper is a revised and expanded version of a paper entitled “M2P2: the pre-processing software tool for Micro-Macro coupling fluid simulation” presented at the 4th International Conference on Cloud-Computing and Super-Computing (CCSC 2015) which is held on December 16-19, 2015 at Jeju National University International Center, Jeju Island, Korea [21].

The authors would like to thank the National Natural Science Foundation of China (Grant no. 61303071) and the Open fund from State Key Laboratory of High Performance Computing (no. 201503-01 and 201503-02) for funding.

References

[1] G. Xu, “The hierarchy and scale of the material structure”, Science & Technology Review, vol. 20, no. 1, **(2002)**, pp. 3-6.
[2] B. Zhan, “CFD simualtion of flow and mass transfer in nanofluids”. Tianjing University, **(2012)**.
[3] S. T. O'Connell and P. A. Thompson, “Molecular dynamics-continuum hybrid computations: a tool for studying complex fluid flows,” Physical Review E, vol. 52, no. 6, **(1995)**, pp. R5792-R5795.
[4] X. Nie, S. Chen and M. O. Robbins, “Hybrid continuum-atomistic simulation of singular corner flow,” Physics of Fluids (1994-present), vol. 16, no. 10, **(2004)**, pp. 3579-3591.
[5] R. Kamali and A. Kharazmi, “Investigation of multiscale fluid flow characteristics based on a hybrid atomistic-continuum method,” Computer Physics Communications, vol. 184, no. 10, **(2013)**, pp. 2316-2320.
[6] N. G. Hadjiconstantinou, A. L. Garcia, M. Z. Bazantc and Gang He, “Statistical error in particle simulations of hydrodynamic phenomena”, Journal of Computational Physics, vol. 187, no. 1, **(2003)**, pp. 274–297.
[7] H. M. Senn and W. Thiel, “QM/MM Methods for Biomolecular Systems”, Angewandte Chemie International Edition, vol. 48, no. 7, **(2009)**, pp. 1198–1229.
[8] C. G. Bhattacharya, “A Simple Method of Resolution of a Distribution into Gaussian Components”, Biometrics, vol. 23, no. 1, **(1967)**, pp. 15-35.
[9] Y. Dong and Z. He, “CAD/CAM and mesh generation”, Applied Science and Technology, vol. 34, no. 6, **(2007)**, pp. 43-45.
[10] “OVITO, The Open Visualization Tool”, http://www.ovito.org/.
[11] “Salome: The Open Source Integration Platform for Numerical Simulation”, http://www.salome-platform.org.
[12] M. Henning and S. Vinoski, “Advanced CORBAr Programming with C++”, Addison-Wesley Longman Publishing, Boston, MA, USA, **(1999)**.
[13] M. Valiev, E. J. Bylaska, N. Govind, K. Kowalski, T. P. Straatsma, H. J J. Van Dam, D. Wang, J. Nieplocha, E. Apra, T. L. Windus and W. A. deJong, “NWChem: A comprehensive and scalable open-source solution for large scale molecular simulations”, Computer Physics Communications, vol. 181, no. 9, **(2010)**, pp. 1477-1489.
[14] H. Zhang, “The design of finite element analysis process automation software framework based on Salome platform”, Wuhan University of Technology, **(2013)**.

[15] A. Hébert, "Integration of the DRAGON5/DONJON5 codes in the SALOME platform for performing multi-physics calculations in nuclear engineering", Joint International Conference on Supercomputing in Nuclear Applications + Monte Carlo, EDP Sciences, no. 04307, **(2014)**.
[16] A. Ribes and A. Bruneton, "Visualizing results in the SALOME platform for large numerical simulations: an integration of ParaView", IEEE Symposium on Large Data Analysis & Visualization, **(2014)**, pp, 119-120.
[17] F. Nunio and P. Manil, "SALOME as a Platform for Magneto-Mechanical Simulation". IEEE Transactions on Applied Superconductivity, vol. 22, no. 3. **(2012)**, pp. 108-110.
[18] A. Elzamly, B. Hussin, Samy S. Abu Naser and M. Doheir, "Classification of Software Risks with Discriminant Analysis Techniques in Software planning Development Process", International Journal of Advanced Science and Technology, vol. 81, **(2015)**, pp. 35-48.
[19] S. Zou, X. Yuan, X. Yang, W. Yi and X. Xu, "An integrated lattice Boltzmann and finite volume method for the simulation of viscoelastic fluid flows". Journal of Non-Newtonian Fluid Mechanics, vol. 211, **(2014)**, pp. 99-113.
[20] X. Xu, X. Guo, Y. Cao, X. Ren, J. Chen and X. Yang, "Multi-scale simulation of non-equilibrium phase transitions under shear flow in dilute polymer solutions", RSC Adv., vol. 5, **(2015)**, pp. 54649-54657.
[21] Q. Wang, W. Yang, F. Li, X. Ren and Y. Tang, "M2P2: the pre-processing software tool for Micro-Macro coupling fluid simulation", Advanced Science and Technology Letters, vol.120, **(2015)**, pp. 727-734.

Authors

Qian Wang, received the BS degree in the School of Computer at National University of Defense Technology, China in 2011 and now is a PhD student at National University of Defense Technology. Her research interests focus on the large scale parallel numerical simulation and parallel software frameworks.

Wenjing Yang, received the PhD degree in the School of Chemical Engineering and Analysis Science at University of Manchester in 2014 and now is a lecturer at National University of Defense Technology. Her research interests focus on computational Rheology and large scale parallel numerical simulation.

Fu Li, received the BS degree in the School of Computer at National University of Defense Technology, China in 2015 and now is a master student at National University of Defense Technology. His research interests focus on robotic operating system.

Xiaoguang Ren, received the BS degree in the School of Computer at National University of Defense Technology, China in 2008 , Master degree in the School of Computer at National University of Defense Technology, China in 2012, and Ph.D degree in the School of Computer at National University of Defense Technology, China in 2014, and now is a lecturer at National University of Defense Technology. His research interests focus on fault tolerant, heigh performance computing, and the large scale parallel numerical simulation and parallel software frameworks.

Yuhua Tang, received her BS and MS degree in the Department of Computer Science from National University of Defense Technology, China in 1983 and 1986, respectively. She is now a professor in the State Key Laboratory of High Performance Computing at National University of Defense Technology. Her research interests include supercomputer architecture and core router's design.