Описание вычислительного комплекса Лаб. 4.3.1 ОИВТ РАН

Специфика архитектуры вычислительного комплекса

Программный комплекс состоит из трех основных частей: библиотеки (*package_library*), вычислительного модуля (*computing_module*) и интерфейса (*package_interface*). Выделение этих трех частей преследует две цели. Первая – разделение программ по частоте изменения. Так, библиотека содержит наименее изменяемые программы, которые представляют ядро программного комплекса. Ядро не привязано к конкретной задаче, и изменять его рядовому пользователю, как правило, не нужно. Вычислительный модуль содержит различные вычислительные алгоритмы и физико-математические модели, которые при работе используют библиотеку. Программы вычислительного модуля обособлены друг от друга и за редким исключением - взаимно заменяемы. При необходимости, пользователь может внедрить свои алгоритмы и модели в вычислительный модуль, используя типы данных и процедуры из библиотеки. Наконец, наиболее часто изменяемая часть программного модуля – интерфейс. Интерфейс состоит из одной программы, с помощью которой происходит постановка каждой конкретной вычислительной задачи. Интерфейс также использует процедуры и типы данных из библиотеки, что обеспечивает корректное взаимодействие между ним и расчетным модулем. Подробнее об этих трех частях будет сказано ниже.

Программный комплекс разработан с использованием возможностей для объектно-ориентированного программирования, добавленных в Fortran стандартом 2008 года. В основном этот подход выражается в том, что все части комплекса состоят из отдельных программных единиц – модулей. В качестве примера, рассмотрим небольшую программу **computational_mesh.f90** которая является частью библиотеки *package_library*. Первым идет название модуля *computational_mesh_class*:

```
module computational_mesh_class
```

Модули Fortran 2008 представляют собой аналог классов в C++, с этим связано то, что названия модулей, как правило, оканчиваются на class.

Далее с помощью оператора *use* загружаются другие модули библиотеки, необходимые для корректной работы *computational_mesh_class*:

```
use kind_parameters
use global_data
use computational_domain_class
```

Затем следует оператор implicit none

```
implicit none
```

Данный оператор предотвращает автоматическое объявление переменных. Каждая отдельная переменная внутри модуля теперь должна быть объявлена явно. Это позволяет значительно снизить риск появления ошибок вызванных опечатками и автоматическим заданием типов переменных. Одним из требований к стилю разработки в рамках данного проекта является использование оператора *implicit none* каждом модуле программы.

Далее идут операторы

```
private
public :: computational_mesh, computational_mesh_pointer, computational_mesh_c
```

Одной из основ парадигмы объектно-ориентированного программирования является инкапсуляция, написание программных обособленных программных единиц, которые содержат как данные (атрибуты), так и методы оперирования с этими данными. Одним из преимуществ такого подхода является взаимозаменяемость модулей. Для того чтобы модуль был максимально обособлен, требуется скрыть все внутренние процедуры и данные, чтобы они были недоступны из других программных модулей. Для этого используется оператор private. Однако что-то должно быть доступно извне, в данном случае само название класса *computational_mesh*, указатель на объект класса *computational_mesh_pointer* и конструктор для создания объектов класса *computational_mesh_c*. Класс в объектно-ориентированном программировании это нечто вроде шаблона, по которому могут быть созданы объекты. В простейшем случае можно провести аналогию с типом данных. К примеру, переменная *I* типа *integer* представляет собой объект, а сам тип *integer* – класс. Разница между типом данных и классом заключается в том, что в класс может содержать еще и процедуры, так называемые методы.

Далее идет объявление класса указателя

```
type computational_mesh_pointer
      type(computational_mesh)   ,pointer     :: mesh_ptr
end type computational_mesh_pointer
```

Особенностью модуля *computational_mesh_class* является то что в нем объявляется по сути два класса: указатель *computational_mesh_pointer* и сам *computational_mesh* (см.ниже). Это не совсем стандартное решение, просто писать отдельный модуль для класса указателя я посчитал излишним. Такие

указатели встречаются еще в нескольких модулях, а вот в *field_pointers.f90* они вынесены отдельно. Работать с объектом-указателем бывает удобнее и эффективнее, чем с самим объектом класса.

Далее идет объявление самого класса computational_mesh

```
type computational_mesh
	private
	real(dkind)   ,dimension(3)       :: cell_edges_length
	real(dkind)                       :: cell_volume
	real(dkind)   ,dimension(3)       :: cell_surface_area

	real(rkind)   ,dimension(:,:,:,:) ,allocatable  ,public      :: mesh
contains

	procedure     ,private     :: generate_uniform_mesh

	! Getters
	procedure     :: get_cell_edges_length
	procedure     :: get_cell_volume
	procedure     :: get_cell_surface_area
end type computational_mesh
```

В классе заданы как данные (переменные *cell_edges_length, cell_volume, cell_surface_area, mesh*) так и методы (*generate_uniform_mesh, get_cell_edges_length, get_cell_volume, get_cell_surface_area*). Все данные скрыты оператором private. Для того чтобы получить значение переменных извне необходимо использовать специальные процедуры *getter*'ы (*get_cell_edges_length, get_cell_volume, get_cell_surface_area*). Метод *generate_uniform_mesh* также скрыт.

После идет интерфейс для процедуры – конструктора объекта класса computational_mesh:

```
interface computational_mesh_c
	module procedure constructor
end interface
```

Это вообще говоря не очень красивое решение, но оно работает при использовании старых версий компилятора Fortran, в котором не до конца реализованы возможности стандарта 2008. По идее процедура-конструктор должна иметь то же название, что и класс. В данном программном комплексе все конструкторы называются с добавлением "_c" к названию класса.

Объявление класса закончено, остается заполнить его "начинкой", реализовать процедуру-конструктор и методы класса. Процедура-конструктор реализуется следующим образом:

```
type(computational_mesh)   function constructor(domain)
	type(computational_domain) ,intent(in)   :: domain
	integer                                  :: dimensions
	integer       ,dimension(3,2)      :: allocation_bounds
```

```
        integer       ,dimension(3) :: cells_number, global_cells_number
        real(dkind)   ,dimension(:,:)     ,allocatable  :: domain_lengths

        integer       :: dim

        dimensions                          = domain%get_domain_dimensions()
        allocation_bounds          = domain%get_local_utter_cells_bounds()
        global_cells_number        = domain%get_global_cells_number()
        cells_number               = domain%get_local_cells_number()

        global_cells_number(1:dimensions) = global_cells_number(1:dimensions) - 2

        allocate(domain_lengths,source = domain%get_domain_lengths())

        allocate(constructor%mesh( dimensions                  , &
                             allocation_bounds(1,1):allocation_bounds(1,2)   , &
                             allocation_bounds(2,1):allocation_bounds(2,2)   , &
                             allocation_bounds(3,1):allocation_bounds(3,2)))

        constructor%cell_volume    = 1.0_dkind
        do dim = 1,dimensions
               constructor%cell_edges_length(dim)= (domain_lengths(dim,2) -
domain_lengths(dim,1)) / global_cells_number(dim)
               constructor%cell_volume =
constructor%cell_volume*constructor%cell_edges_length(dim)
        end do

        do dim = 1,dimensions
               constructor%cell_surface_area(dim)       = constructor%cell_volume /
constructor%cell_edges_length(dim)
        end do

        call constructor%generate_uniform_mesh(domain)
end function
```

Класс *computational_mesh* представляет собой расчетную сетку. Для её создания нужно знать параметры расчетной области, которые хранятся в классе *computational_domain*. Поэтому конструктор в качестве входного параметра принимает объект класса *computational_domain* с названием *domain*. Далее идет объявление вспомогательных переменных dimensions, *allocation_bounds*, *cells_number*, *global_cells_number*, *domain_lengths* и *dim*. Пока эти переменные не имеют значений. Значения им присваиваются с помощью вызова *getter*'ов класса computational_domain. Так переменной *dimensions* присваивается значение выдаваемое процедурой *domain%get_domain_dimensions()*. Здесь нужно отметить синтаксис обращения к методу класса. Объект класса – *domain*, вызываемый метод – *get_domain_dimensions()*. Они разделены символом %.

После присваивания значений переменным из объекта *domain* идут некоторые операции с памятью и вычислительные операции. Нужно отметить, что *constructor*, по сути, является функцией, которая как результат выдает объект класса *computational_mesh*. Чтобы выдать объект необходимо инициализировать все необходимые данные объекта. В данном случае нужно

задать переменные *cell_edges_length, cell_volume, cell_surface_area, mesh*. Для обращения к этим переменным используется такой же синтаксис, как и для обращения к методам класса *computational_domain*: доступ к переменной *cell_edges_length* осуществляется оператором *constructor%cell_edges_length*. Вычисления значений величин *cell_edges_length, cell_volume, cell_surface_area* довольно прямолинейны: *cell_edges_length* – размер расчетной ячейки, массив из трех элементов, отношение размера расчетной области по направлению к числу расчетных ячеек по тому же направлению, *cell_volume* – объем расчетной ячейки, число, произведение *cell_edges_length* во всех направлениях, *cell_surface_area* – площадь граней расчетной ячейки, массив из трех элементов, попарное произведение размеров расчетной ячейки. *mesh* – четырехмерный массив координат центров расчетных ячеек. *mesh(1,i,j,k)* определяет координату *x* центра ячейки с индексами *i,j,k*, *mesh(2,i,j,k)* - координату *y*, *mesh(3,i,j,k)* – координату *z*. Для формирования этого массива, сначала для него необходимо выделить память.

```
allocate(constructor%mesh( dimensions                  , &
                    allocation_bounds(1,1):allocation_bounds(1,2)   , &
                    allocation_bounds(2,1):allocation_bounds(2,2)   , &
                    allocation_bounds(3,1):allocation_bounds(3,2)))
```

Первый индекс – по количеству измерений dimensions, второй, третий и четвертый – по количеству ячеек в расчетной области (см. ранее присваивание *allocation_bounds*. *local_utter_cells_bounds* – число всех ячеек в расчетной области, вместе с дополнительным рядом теневых ячеек по бокам расчетной области):

```
allocation_bounds          = domain%get_local_utter_cells_bounds()
```

После выделения памяти, для генерации координат центров ячеек вызывается процедура *generate_uniform_mesh*:

```
subroutine generate_uniform_mesh(this,domain)

            ! Generation of the uniform structured computational grid in computational
domain "domain"

            class(computational_mesh)  ,intent(inout)      :: this
            type(computational_domain) ,intent(in)         :: domain

            real(dkind)   :: dimless_length
            integer             :: i, j, k, dim

            integer                                        :: dimensions
            integer       ,dimension(3,2)                  :: loop_bounds
            integer ,dimension(3)                          :: decomposition_offset
            real(dkind)   ,dimension(:,:)     ,allocatable :: domain_lengths
```

```
            dimensions                          = domain%get_domain_dimensions()
            decomposition_offset = domain%get_global_offset()

            loop_bounds                         = domain%get_local_utter_cells_bounds()

            allocate(domain_lengths,source = domain%get_domain_lengths())

            do dim = 1,dimensions
                  do k = loop_bounds(3,1),loop_bounds(3,2)
                  do j = loop_bounds(2,1),loop_bounds(2,2)
                  do i = loop_bounds(1,1),loop_bounds(1,2)
                        dimless_length = real((i-1)*I_m(dim,1)+(j-1)*I_m(dim,2)+(k-
1)*I_m(dim,3),dkind)

                        this%mesh(dim,i,j,k) = domain_lengths(dim,1) +
(decomposition_offset(dim) + 0.5_dkind + dimless_length)*this%cell_edges_length(dim)
                  end do
                  end do
                  end do
            end do

            end subroutine
```

Процедура равномерно разбивает расчетную область на ячейки.

Процедуры *getter*'ы имеют простую форму, их смысл в возвращении данных объекта computational_mesh.

Структура остальных модулей программного комплекса в общих чертах идентична разобранному модулю computational_mesh.

Сборка программного комплекса в Visual Studio

Так как программный комплекс логически состоит из трех частей, необходимо создать одно решение и три отдельных проекта в нем *package_library*, *computing_module* и *package_interface*. Типы проектов:

Проект *computing_module*: console application.

Проект *package_library*: static library application

Проект *package_interface* с одним файлом: console application

Общие настройки проектов. Программный комплекс является кросс-платформенным и работает как в Windows так и в Unix системах. Чтобы сделать сборку комплекса для Windows, во всех трех проектах необходимо установить следующие настройки препроцессора:

Fortran -> General -> Preprocessor -> Preprocess Source File -> Yes (/fpp)

Fortran -> General -> Preprocessor -> Preprocessor Definitions -> Вписать «WIN».

Влияние препроцессора можно проследить в исходном коде программы. Директивы препроцессора начинаются на #. См. например global_data.f90:

```
#ifdef WIN
	character(len=100)  ,parameter    :: fold_sep                                  =
'\'
#else
	character(len=100)  ,parameter    :: fold_sep                                  =
'/'
#endif
```

В зависимости от того, задана ли переменная WIN используются различные варианты разделителя папок файловой системы (В windows используется “\” для разделения папок, в то время как в Unix – “/”, по умолчанию используется разделитель для Unix)

Библиотека *package_library* используется остальными проектами. При использовании одного решения для трех проектов, зависимости *computing_module* и *package_interface* от package_library учитываются автоматически, для этого необходимо вызвать меню решения, щелкнув на нем правой кнопкой мыши и открыть меню Project Dependencies, где проставить галочки напротив *package_library* для проектов *computing_module* и *package_interface*, тем самым указав среде необходимый порядок сборки.

Далее необходимо скомпилировать (Build -> Build solution) решение, при этом среда разработки должна провести последовательную компиляцию библиотеки *package_library* и зависящих от неё проектов *computing_module* и *package_interface*. Если компиляция проходит успешно, то можно приступить к постановке задачи в *package_interface, main.f90*.

Постановка задачи

Рассмотрим тестовую постановку задачи.

Загрузка библиотечных модулей и создание объектов основных классов:

```
use kind_parameters
      use global_data
      use computational_domain_class
      use chemical_properties_class
      use thermophysical_properties_class
      use solver_options_class
      use computational_mesh_class
      use mpi_communications_class
      use data_manager_class
      use boundary_conditions_class
      use field_scalar_class
      use field_vector_class
      use data_save_class
      use data_io_class
      use post_processor_manager_class

      type(computational_domain)                          :: problem_domain
      type(data_manager)                                  :: problem_data_manager
      type(mpi_communications)                            :: problem_mpi_support

      type(chemical_properties)              ,target      :: problem_chemistry
      type(thermophysical_properties)        ,target      :: problem_thermophysics

      type(solver_options)                                :: problem_solver_options

      type(computational_mesh)          ,target           :: problem_mesh
      type(boundary_conditions)         ,target           :: problem_boundaries
      type(field_scalar_cons)           ,target           :: p, T, rho
      type(field_vector_cons)           ,target           :: v, Y

      type(post_processor_manager)                        :: problem_post_proc_manager

      type(data_io)                                       :: problem_data_io
      type(data_save)                                     :: problem_data_save
```

Инициализация служебных и вспомогательных переменных для удобства постановки задачи:

```
real(dkind)   ,dimension(3)       :: cell_size
integer       ,dimension(3,2)     :: utter_loop, observation_slice
integer                           :: log_unit
logical       :: stop_flag
integer       :: front_lp
```

Создание папки с постановкой и файла-журнала постановки задачи:

```
call system('mkdir '// task_setup_folder)
open(newunit = log_unit, file = problem_setup_log_file, status = 'replace', form =
'formatted')
```

Создание объекта расчетной области:

```
problem_domain      = computational_domain_c(  dimensions    =      1,     &
      cells_number         =      (/5000,1,1/),       &
      coordinate_system    =      'cartesian'   ,     &
```

```
        lengths             =       reshape((/    0.0_dkind,0.0_dkind,0.0_dkind,    &
                                            0.05_dkind,0.005_dkind,0.005_dkind/),(/3,2/)), &
        axis_names          =       (/'x','y','z'/))
```

Dimensions – число измерений (1,2,3), cells_number – число ячеек вдоль каждого измерения (количество ячеек по незаданным измерениям не учитывается), *coordinate_system* – название системы координат (на данный момент поддерживается только прямоугольная декартова система координат ‘*cartesian*’), *lengths* - размеры расчетной области (функцией *reshape* формируется массив 3x2 где первое измерении отвечает за координату, второе – за минимум (:,1) или максимум (:,2) соответствующей координаты (в примере задается расчетная область x:0.0-0.05, координаты y, z не учитываются так как dimensions = 1), axis_names – названия осей (сейчас ни на что не влияет).

Создание объекта параметров химической кинетики:

```
problem_chemistry           = chemical_properties_c(
        chemical_mechanism_file_name      = 'WARNATZ.txt'      ,      &
        default_enhanced_efficiencies     = 0.0_dkind          ,      &
        E_act_units                       = 'kJ.mol')
```

chemical_mechanism_file_name – название файла с механизмом химической кинетики, default_enhanced_efficiencies – см. CHEMKIN, E_act_units – единицы измерения энергии активации.

Создание объекта параметров переноса и теплофизики:

```
problem_thermophysics       = thermophysical_properties_c(
              chemistry                 = problem_chemistry                    ,      &
              thermo_data_file_name     = 'WARNATZ_THERMO.txt'     ,      &
              transport_data_file_name  = 'WARNATZ_TRANSDATA.txt'  ,      &
              molar_masses_data_file_name= 'molar_masses.dat')
```

Chemistry – объект параметров химической кинетики, созданный ранее, thermo_data_file_name – название данных с теплофизическими константами, transport_data_file_name – название файла с коэффициентами переноса, molar_masses_data_file_name – название файла с молярными массами.

Файлы химической кинетики, данных переноса и теплофизики должны заранее присутствовать в папке ‘task_setup’.

Создание объекта менеджера данных (стандартно для всех задач):

```
problem_data_manager=
data_manager_c(problem_domain,problem_mpi_support,problem_chemistry,
problem_thermophysics)
```

Создание граничных условий:

```
call problem_data_manager%create_boundary_conditions( problem_boundaries          , &
                                                      number_of_boundary_types    = 1    , &
```

```
                                                default_boundary            = 1)
```

problem_boundaries - создаваемый объект параметров граничных условий, number_of_boundary_types – количество граничных условий, default_boundary – номер условия, выставляемого по умолчанию на границах области.

Создание сетки и физических полей, необходимых для постановки задачи:

```
call problem_data_manager%create_computational_mesh(problem_mesh)

call problem_data_manager%create_scalar_field(p ,'pressure'          , 'p')
call problem_data_manager%create_scalar_field(T ,'temperature'       , 'T')
call problem_data_manager%create_scalar_field(rho,'density'          , 'rho')

call problem_data_manager%create_vector_field(v ,'velocity'          ,'v'
      ,'spatial')
call problem_data_manager%create_vector_field(Y ,'specie_molar_concentration','Y'
      ,'chemical')
```

Задание вспомогательных параметров

```
cell_size           = problem_mesh%get_cell_edges_length()
utter_loop          = problem_domain%get_global_utter_cells_bounds()

front_lp                    = 1

observation_slice           = utter_loop
observation_slice(2,:)      = front_lp
```

Создание менеджера постпроцессоров:

```
problem_post_proc_manager         =         post_processor_manager_c(problem_data_manager,
number_post_processors = 1)
```

number_post_processors – число постпроцессоров в менеджере.

Создание первого постпроцессора

```
call problem_post_proc_manager%create_post_processor(problem_data_manager  ,      &
                                      post_processor_name = "proc1"     ,      &
                                      operations_number   = 1           ,      &
                                      save_time           = 500.0_dkind ,      &
                                      save_time_units     = 'nanoseconds')
```

problem_data_manager – экземпляр созданного ранее менеджера данных, post_processor_name – название постпроцессора, в соответствии с которым будет создан файл с данными, operations_number – количество выполняемых операций, save_time – временной интервал между одновременных выполнением всех операций постпроцессора, save_time_units – единицы измерения временного интервала (nanoseconds, microseconds, milliseconds).

Создание первой операции первого постпроцессора.

```
call
problem_post_proc_manager%create_post_processor_operation(problem_data_manager,1,'tempera
```

```
ture','min_grad',observation_slice,(/1,front_lp,1/)    ,1)
```

Последовательно задаются параметры: *problem_data_manager* – экземпляр созданного ранее менеджера данных, 1 – номер постпроцессора, “*temperature*” полное название обрабатываемого поля, “*min_grad*” - название операции (доступны на данный момент: *max*, *min*, *transducer*, *max_grad*, *min_grad*, *sum*, *mean*), *observation_slice* – задается область, в которой выполняется операция, в данном случае поиска минимального градиента, (/1,front_lp,1/) – т.н. anchor point (первая операция в постпроцессоре всегда считается ведущей, поэтому для первой операции задание *anchor_point* произвольно, для остальных операций *anchor_point* служит для согласования областей действия операции по следующему принципу: первая операция определяет некоторую точку в которой выполняется условие min_grad, далее определяется сдвиг этой точки относительно anchor_point других операций, на значение найденного сдвига преобразуются области операций), 1 – номер проекции градиента (для других типов операций произвольный).

Создание объекта вывода данных.

```
problem_data_save    = data_save_c(      problem_data_manager,      &
                           visible_fields_names= (/'pressure'       ,&
                                  'temperature'                     ,&
                                  'density'                         ,&
                                  'velocity'                        ,&
                                  'specie_molar_concentration'      ,&
                                  'specie_production_chemistry'     ,&
                                  'energy_production_chemistry'     ,&
                                  'velocity_of_sound'               ,&
                                  'velocity_production_viscosity'   ,&
                                  'mixture_molar_concentration'     ,&
                                  'full_energy'/) ,    &
              save_time                   = 1.0_dkind          ,      &
              save_time_units             = 'microseconds'     ,      &
              save_format                 = 'tecplot'          ,      &
              data_save_folder            = 'data_save'        ,      &
              debug_flag                  = .false.)
```

problem_data_manager – экземпляр созданного ранее менеджера данных, *visible_fields_names* – полные названия выводимых полей, *save_time* – время сохранения, *save_time_units* – единицы измерения времени сохранения, *save_format* – формат данных, *data_save_folder* – папка с данными, *debug_flag* – логический флаг режима отладки, в режиме отладки игнорируется параметр *visible_fields_names* и выводятся все задействованные в расчете поля.

Создание объекта сохранения/загрузки данных.

```
problem_data_io             = data_io_c(problem_data_manager         ,      &
                            check_time          = 5000.0_dkind       ,      &
                            check_time_units    = 'nanoseconds'      ,      &
```

```
                              output_time          = 1400.0_dkind      ,      &
                              data_output_folder   = 'data_output')
```

problem_data_manager – экземпляр созданного ранее менеджера данных, check_time – интервал расчетного времени между проверками реального времени, check_time_units – единицы измерения интервала расчетного времени, output_time – реаьлно время через которое осуществляется полное сохранение данных, data_output_folder – директория для хранения и загрузки данных расчета.

Блок с постановкой начальных условий. Ставится задача в ударной трубе, левая часть расчетной области (до 2000 ячеек) заполнена гелием при высоком давлении, правая часть области – стехиометрической водород-воздушной смесью при атмосферном давлении.

```
!****************************** Setting initial conditions ****************
p%cells(:,:,:)                    = 1.0_dkind*101325.0_dkind
p%cells(:2000,:,:)                = 2.385490_dkind*101325.0_dkind

T%cells(:,:,:)                    = 300.0_dkind

Y%pr(1)%cells(:2000,:,:)          = 1.0_dkind          ! Hydrogen
Y%pr(2)%cells(:2000,:,:)          = 0.5_dkind          ! Oxygen
Y%pr(3)%cells(:2000,:,:)          = 0.5*3.762_dkind    ! N2

Y%pr(4)%cells(2001:,:,:)          = 1.0_dkind          ! He
      !*********************************************************************
```

Создание типов граничных условий. Номера создаваемых типов выставляются последовательно, в соответствии с их созданием:

```
call problem_boundaries%create_boundary_type (  type_name     = 'wall'      ,      &
                                                slip          = .false.     ,      &
                                                conductive    = .false.     ,      &
                                                wall_temperature     = 0.0_dkind   ,      &
                                                wall_conductivity_ratio= 0.0_dkind,      &
                                                farfield_pressure    = 0.0_dkind   ,      &
                                                farfield_temperature= 0.0_dkind ,&
                                                priority                          = 1)
```

type_name – тип условия (доступные опции *wall, inlet, outlet, symmetry_plane*), *slip* – условие проскальзывания, conductive – условие теплоотдачи, *wall_temperature* – температура стенки, *wall_conductivity_ratio* – отношение теплопроводности стенки к теплопроводности смеси в приграничной ячейке, *farfield_pressure* – давление на бесконечности, *farfield_temperature* – температура на бесконечности, priority – приоритет условия в спорных ячейках.

Параметры расчетной методики:

```
problem_solver_options = solver_options_c(      solver_name   = 'cpm'       , &
                                    hydrodynamics_flag   = .true.          , &
                                    heat_transfer_flag   = .false.         , &
                                    molecular_diffusion_flag   = .true.    , &
                                    viscosity_flag             = .false.   , &
                                    chemical_reaction_flag     = .false.   , &
                                    CFL_flag                   = .true.    , &
                                    CFL_coefficient            = 0.1_dkind , &
                                    initial_time_step          = 1e-07_dkind)
```

solver_name – название метода решения (солвера). В данном случае используется метод крупных частиц, далее идут логические флаги включения выключения учета отдельных процессов: hydrodynamics_flag – гидродинамика (пока отключается только в солвере по методу крупных частиц), diffusion_flag – диффузия, viscosity_flag – вязкость, heat_trans_flag – теплопроводность, reactive_flag - химическая кинетика, courant_fraction – доля числа Куранта для расчета шага по времени.

Краткое описание задачи в журнале:

```
      !****************** Writing problem short description ******************
      write(log_unit,'(A)') 'General description: subsonic one-dimensional shock tube
test for heavy driver.'
      write(log_unit,'(A)') 'Main aim: testing.'
      write(log_unit,'(A)') 'Problem setup: O2 driver, H2 driven, subsonic pressure drop
5atm - 1atm, 300K.'
      write(log_unit,'(A)') 'Solver setup: CPM solver, hydrodynamics only, varying time
step, CFL = 0.1, initial time step 1e-07s.'
      write(log_unit,'(A)') 'Validation and comparison: analytical solution'
      write(log_unit,'(A)') '------------------------------------------------------------
--------------'
      !********************************************************************************
```

Сохранение постановки для последующей загрузки и визуализации.

```
      call problem_data_io%output_all_data(0.0_dkind,stop_flag,make_output = .true.)
      call problem_data_save%save_all_data(0.0_dkind,stop_flag,.true.)
```

После установки всех параметров, программу-интерфейс необходимо запустить для генерации постановки задачи. Чтобы она сработала корректно, в папке с программой уже должна находиться папка task_setup, содержащая папки chemical_mechanisms и thermophysical_data. Успешно сгенерированная задача состоит из трех папок: data_save, data_output, task_setup и одного файла problem_setup.log. После запуска внутри папки task_setup помимо папок с химической кинетикой, параметрами переноса и теплофизикой появятся также файлы с расширением *.inf, которые являются параметрами постановки задачи. Большинство из этих параметров можно менять без

использования интерфейса. Неизменными должны оставаться параметры расчетной области domain_data.inf, так как они необходимы для корректной загрузки данных. Остальные параметры можно менять непосредственно перед расчетом.

Расчет

Для расчета необходимо создать отдельную папку, в которую помещаются папки data_save, data_output и task_setup, а также problem_setup.log и скомпилированный computing_module.exe.