Anes Team

ANES20XE: Код для численного моделирования процессов гидродинамики и тепломассообмена

Версия 2.24

Фортран-интерфейс пользователя

Москва 2019 г.

Оглавление.

Введение

Для решения простых задач пользователь может использовать подсистему myFORM для задания граничных условий, свойств, источников и для типичной обработки результатов расчета. При решении более сложных задач необходимо писать свои собственные подпрограммы на Фортране. В данном разделе описываются правила создания таких подпрограмм.

Все подпрограммы пользователя можно условно разделить на несколько групп:

1. *Виртуальные функции V-переменных*: эти функции всегда вызываются в различных циклах перебора контрольных объемов (ячеек) сетки. Для определения КО, для которого вызывается функция, используются понятие *текущего индекса* КО. Для структурной сетки текущий индекс – это тройка переменных (ixC, iyC, izC). Для неструктурной сетки – это одна переменная idCV.
2. *События пользователя*: эти функции не связаны с текущим КО, главная их особенность – они вызываются в определенных местах кода Решателя.
3. *Обработчики моделей* пористой среды и моделей турбулентности: Эти подпрограммы похожи на виртуальные функции. Однако в отличие от виртуальных функций эти подпрограммы выполняют сразу несколько вычислений.

В данном документе описаны сведения о программной архитектуре кода Решателя, необходимые для создания своего фортрановского интерфейса.

☞Замечание. При установке Anes создается подкаталог openfor, который содержит ряд подпрограмм из кода Решателя. Их изучение может существенно помочь при создании своих подпрограмм.

1. Виртуальные функции V-переменных

Если пользователь хочет определить свой собственный произвольный алгоритм расчета V - переменной, необходимо использовать оператор

```
Aname = Virtual(<Имя функции>)
```

и определить подпрограмму с указанным именем в *секции* файла проекта с именем [vf <Имя функции>] и следующим шаблоном:

```
SUBROUTINE <Имя функции> (RetVal)
real(4) retVal
!-----
…..
!-----
END SUBROUTINE
```

Содержимое этой подпрограммы зависит от типа AIL-оператора, использующего Variant-переменную. В частности, не всегда необходимо использовать единственный аргумент подпрограммы – RetVal (например, для Event-переменных пользователя), часть параметров, необходимых для вычислений, могут передаватся в виртуальную функцию через специальные COMMON-блоки (например, при расчете свойств материалов). Подробнее эти правила рассмотрены ниже при описании отдельных V-переменных.

Пользователь может поместить текст виртуальной функции в отдельный фортрановский файл, подключаемый к проекту на этапе сборки локального Решателя. В этом случае необходимо использовать другой вариант описания виртуальной функции

```
Aname = External(<Имя функции>)
```

Для описания свойств материалов в Базе Данных свойств используется еще одна форма для описания виртуальной функции

```
Aname = Internal(<Имя функции>)
```

В этом случае код подпрограммы *уже включен* в код Решателя. Для пользователя такой вариант описания используется при выборе модели FS взаимодействия или выборе пристенных функций модели турбулентности.

1.1 Общая форма вызова виртуальной функции

Этот вид виртуальной функции используется для (ниже перед именем AIL-переменной указывается имя секции проекта):

1) инициализации Ф-переменной : [Phi Variables].Init("Ф-имя");
2) расчета полей-переменных пользователя: [User Variables].Calc(“User-имя”);
3) расчета коэффициента турбулентной вязкости в алгебраической модели: [Turbulence].NuTVar;
4) расчета турбулентного числа Прандтля для Ф-переменной: [Turbulence]. SigmaTUR("Ф-имя") ;
5) Задания Coef и Val параметров источниковых членов: [Source]. COEF("ИмяПатча.Ф-имя") и [Source]. VAL("ИмяПатча.Ф-имя ");
6) Задания Coef и Val параметров граничных условий: [Boundary Conditions]. COEF("ИмяПатча.Ф-имя ") и [Boundary Conditions]. VAL("ИмяПатча.Ф-имя ").

В этом случае формат функции имеет самый «простой» вид:

```
SUBROUTINE <Имя функции> (RetVal)
real(4) retVal
!-----
<расчет значения и присвоение его переменной RetVal>
!-----
END SUBROUTINE
```

Внутри Решателя эта функция вызывается следующим образом:
для структурной сетки:

```
do izC = 1, NZ
  do iyC = 1, NY
    do ixC = 1, NX
      if < отбор нужных КО> then
          call <Имя функции> (RetVal)
         <Переменная Решателя>(ixC,iyC,izC) = RetVal
      else
         <Переменная Решателя>(ixC,iyC,izC) = 0
      endif
    end do
  end do
end do
```

для неструктурной сетки:

```
do idCV = 1, NoCells
      if < отбор нужных КО> then
          call <Имя функции> (RetVal)
         <Переменная Решателя >(idCV) = RetVal
      else
         <Переменная Решателя >(idCV) = 0
      endif
end do
```

В тексте виртуальной функции пользователю доступны *все* переменные Решателя, включая индексы

(ixC,iyC,izC) или idCV,

определяющие индекс *текущего* КО, для которого вызвана виртуальная функция.

Именно эти индексы и нужно использовать для реализации вычислений. Однако гораздо более удобным является подход, основанный на использовании пользовательских функций доступа:

1) эти функции одинаково работают как со структурной (индексы ixC,iyC,izC), так и неструктурной сеткой (индекс idCV);
2) пользователю не нужно знать архитектуру внутреннего представления полей Ф-переменных.

Текущий список функций доступа приведен в таблице 1.1.

Таблица 1.1

Функция	Назначение
subroutine userXYZ(XP,YP,ZP)	возвращает значение координат центра *текущего* КО,
real(8) function userPHI(iPHI)	возвращает значение Ф-переменной с индексом iPHI в центре КО. Для индекса iPHI можно использовать либо индексы предопределенных Ф-переменных: P_Pf, P_UGX, P_UGY, P_UGZ, & P_TG, P_TS, & P_kTur, P_epsTur, P_omTur & P_CONC(8) либо индекс Ф-переменной, переданный

	функцией GetPHIID("Ф-имя")
real(8) function userGrad(iDIR,iPHI)	возвращает значение производной Ф-переменной iPHI в направлении iDIR (1 - x, 2 - y, 3 - z)
subroutine userUG_XYZ(UX,UY,UZ)	подпрограмма возвращает значение трех компонент скорости G-фазы для центра КО; если какая-то компонента скорости отсутствует, то возвращается ноль.
subroutine userG_PROP(aDens, aVisc,aCP,aLamda)	подпрограмма возвращает свойства G-фазы для центра КО;
subroutine userS_PROP(aDens, aCP, aLamda)	подпрограмма возвращает свойства S-фазы для центра КО;

☞Замечание. Не используйте функцию GetPHIID("Ф-имя") внутри виртуальной функции! Эта функция находит индекс Ф-переменной путем перебора имен решаемых Ф-переменных, а это занимает время процессора. Правильнее всего определит этот индекс в подпрограмме Event-переменной onInit.

Приведем типичный пример: инициализация Ф-переменной UgZ для ламинарного течения в канале:

```
[Macro Variables]
  Macro(LY,U0)   ! это высота канала и средняя скорость
…LY = 0.2..      ! в м
…U0 = ..10.5     ! в  м/с
[User Data]
   RUDat(1) = LY
   RUDat(2) = U0
…..
[PHI Variables]
…Init(“UGZ”) = Virtual(myInitUGZ)
…..
[vf myInitUGZ]
   SUBROUTINE myInitUGZ (RetVal)
     real(4) retVal
!-----
     Call userXYZ(X00,Y00,Z00)     ! координаты текущего КО
      aCENTRE = RUDat(1)/2.0
     RetVal =  1.5*RUDat(2) *(1.0 – ((Y00-aCENTRE)/ aCENTRE)**2)
   END SUBROUTINE
```

☞Замечание. Макропеременные проекта можно использовать в любых AIL-операторах, но *нельзя* использовать в Фортрановских текстах! Для передачи значений макропеременных нужно использовать секцию [User Data].

Для расчета скалярных переменных также используются виртуальные функции общего типа:

1) для расчета скалярных переменных пользователя: [User Scalars].Calc(“User-имя”);
2) для расчета массового потока при стабилизированном течении: [Internal Source]. MassRateZ;

Главная особенность этих функций – они не вызываются в цикле по КО ! При вызове этих функций текущие индексы (ixC,iyC,izC) или idCV устанавливаются на самый первый КО в расчетной области.

1.2 Виртуальные функции для событий пользователя

Эти виртуальные функции определяются в секции [User Event] и *не* используются для вычисления значения переменной. Они служат для описания виртуальной функции, вызываемой в определенном месте алгоритма расчетного модуля (обработчики «события» пользователя).

Виртуальная функция в этом случае имеет вид:

```
SUBROUTINE <Имя функции>(RetVal)
real(4) retVal
  COMMON /CVF_ENTRY/ iEntry
!-----
<любые вычисления>
!-----
END SUBROUTINE
```

Главное отличие – появление Common-блока /CVF_ENTRY/. Смысл параметра iEntry определяется типом A-переменной:

AIL переменная	iEntry	Примечание
OnInput	1	Ввод исходных данных
OnInit	2	Инициализация
BeforeSweep	3	Перед SWEEP-итерацией
AfterSweep	4	После SWEEP-итерации
BeforeTime	5	Перед шагом по времени
AfterTime	6	После шага по времени
OnReport	7	Вывод пользователя
OnStop	8	Действия пользователя перед остановкой Решателя
OnLESolver	XXX	Реализация собственного линейного солвера:
	1	Инициализация собственных переменных
	2	Освобождение собственных переменных
	3	Проведение расчета

Естественно, что в данном случае переменная RetVal не используется Решателем. Порядок вызова событий пользователя определяется алгоритмом работы Решателя. Ниже приведены две блок-схемы работы для стационарных и нестационарных задач.

1.3 Блок-схема работы Решателя

При решении стационарных задач используется следующий алгоритм решения и порядок вызова событий пользователя:

```
<Setup>     ! Инициализация переменных по умолчанию,
<Input>     ! Чтение файла инициализации AID/AGR и выделение памяти под массивы
 *Вызов «события пользователя» OnInput
<Init>      ! Инициализация PHI-полей (по умолчанию и из файла результатов)
 *Вызов «события пользователя» OnInit

! ---------- Итерационный процесс ------------------
iSWEEP  = 0
Liter = .TRUE.
DO WHILE(LITER)
    iSWEEP = iSWEEP + 1
    *Вызов «события пользователя» BeforeSweep
    <Расчет одной итерации>                  : *Вызов события OnLESolver
    *Вызов «события пользователя» AfterSweep
ENDDO
<Отчет Решателя>
*Вызов «события пользователя» OnReport
<Создание файлов результатов >

 ……

STOP: *Вызов «события пользователя» OnStop
```

При решении нестационарных задач используется другой алгоритм решения:

```
<Setup>     ! Инициализация переменных по умолчанию,
<Input>     ! Чтение файлов исходных данных Решателя и выделение памяти под массивы полей
 *Вызов «события пользователя» OnInput
<Init>      ! Инициализация PHI-полей (по умолчанию и из файла результатов)
 *Вызов «события пользователя» OnInit
! ---------- Итерационный процесс ------------------
<BeforeSolve>   ! Расчет специальных Ф-переменных до основного решения
! ======== Шаги по времени ===============
iTIME = 0
Time = TimeBeg
DO WHILE(Time < TimeEnd)
   <Переприсвоение   Фk0 = Фk >
   Time = Time + dTime
   *Вызов «события пользователя» BeforeTime
! ---------- Итерационный процесс на шаге по времени ------------------
   iSWEEP  = 0
   Liter = .TRUE.
   DO WHILE(LITER)
      iSWEEP = iSWEEP + 1
      *Вызов «события пользователя» BeforeSweep
      <Расчет одной итерации>                  : *Вызов события OnLESolver
      *Вызов «события пользователя» AfterSweep
! --- Проверка сходимости ------
       IF(Все errФ(iPHI) < Eps)  Выход из SWEEP-цикла
   ENDDO
   *Вызов «события пользователя» AfterTime
   <Отчет на шаге по времени>
   *Вызов «события пользователя» OnReport
   <Создание файлов результатов на шаге по времени>
ENDDO
STOP: *Вызов «события пользователя» OnStop
```

1.4 Виртуальные функций для описания свойств материалов

Для задания расчета свойств используется либо секция [Properties] и операторы

```
PROP("Имя-материала.Имя-свойства") = Virtual(<ИмяФункции>)
PROP("Имя-материала.Имя-свойства ") = External(<ИмяФункции>)
```

либо база данных свойств. БД свойств будет описана ниже, а здесь рассмотрим особенности реализации виртуальных функций.

Если используется первый вариант описания (Virtual), то функция располагается в самом проекте. В этом случае для вычисления свойства используется стандартный подход, основанный на понятии текущего КО и функциях доступа (см. пункт 1.1).

При использовании второй формы (External) нужно добавить в вызов функции специальный COMMON-блок /VAR_PROP/:

```
SUBROUTINE <Имя функции> (RetVal)
real(4) retVal
  COMMON/VAR_PROP/ pT0, pP0, pT, pPF, pConc(8)
!-----
< вычисления RetVal>
!-----
END SUBROUTINE
```

Перед вызовом этой функции Решатель заполняет переменные этого блока текущими значениями:

pP0 - текущее значение переменной Решателя P0,

pT0 - текущее значение переменной Решателя T0,

pT - текущее значение температуры G- или S-фазы в текущем КО,

pPF - текущее значение давления PF G- фазы в текущем КО,
pCONC(1:8) - текущее значения концентраций G- фазы в текущем КО.

Эти переменные можно использовать для написания «независимых» подпрограмм для расчета свойств, когда подпрограмма никак не связана с Решателем (подпрограмма ничего «не знает» о полях Решателя и о текущих индексах КО)!

Anes допускает использовать еще одну форму виртуальной функции для описания свойств

```
PROP("Имя-материала.Имя-свойства ") = Internal(<ИмяФункции>)
```

Эта форма полностью идентична External-форме, но сама функция уже находится в составе ядра Решателя – в библиотеке Property.

Для добавления свойства в эту библиотеку (если у вас имеется код Решателя) необходимо выполнить следующие шаги:

1) в подпрограмму acSetInternalPROP необходимо добавить две строки:
```
external <ИмяФункции>
if(CompareStr(NameSubr,'<ИмяФункции>')) Addres = AddrFUN(<ИмяФункции>)
```
2) Добавить в библиотеку текст этой подпрограммы и перекомпилировать библиотеку.

1.5 Специальные виртуальные функций

Эти функции используются для описания:

1) пристенных функций моделей турбулентности: [Turbulence].TWallFun;
2) моделей межфазного пористого взаимодействия:
   [Porous Models]. InterFace(ИмяFS-зоны),
   [Porous Models].L2Model(ИмяFS-зоны).

Примеры этих виртуальных функций можно найти в каталоге openfor Anes.

2. Архитектура кода Решателя

2.1 Фортрановские переменные пользователя

В подпрограммах пользователь может использовать любые локальные переменные и все доступные переменные Решателя (главные из них будут описаны ниже). Для описания своих собственных глобальных переменных, доступных в любых подпрограммах, необходимо использовать секцию [User Fortan Variables].

А-компилятор проекта переводит содержимое этой секции в файл module Фортрана с именем UserData. Этот module доступен во всех виртуальных функциях и подпрограммах-событиях пользователя. Для описания переменных можно использовать синтаксис Фортран-95. В качестве примера ниже приведен пример содержимого секции [User Variables]:

```
!---------- Общие переменные----------------
 integer idRES
 real,pointer :: aTWALL(:), aTV(:), aAlfaV(:),aAreaG(:)
! -----
 real, parameter:: C_PI = 3.1416, GVAP = 2.e-03
 real, parameter:: D0   = 21.e-03, DelT = 2.e-3, CondT = 18.0
 real, parameter:: DensV = 0.06816, ViscV = 1.533E-4, CpV = 1901.0
 real, parameter:: CondV = 0.02
! -----
 real,parameter :: TV_IN = 81.7, TV_SAT = 45.0
! -----
 real vapCOEF
```

☞Замечание. Пользователь наряду с виртуальными функциями может создавать свои собственные подпрограммы и функции. Текст этих подпрограмм помещается в секцию [User Subroutines]. Эти подпрограммы при генерации локального Решателя рассматриваются как независимые подпрограммы. Поэтому для доступности в них переменных пользователя в каждую подпрограмму нужно добавить оператор

```
use UserData
```

Если пользователю необходимы массивы-поля, связанные с ячейками сетки, то можно использовать два подхода.

Первый вариант: можно использовать сеточные User-переменные пользователя, определяемые в секции [User Variables] как внутреннюю переменную (для такой переменной пользователь не задает AIL-оператор Calc).

Второй вариант: можно воспользоваться операторами секции [User Fortran Fields].

Как уже отмечалось, макропеременные файла проекта «действуют» только в пределах проекта и они недоступны в виртуальных функциях и подпрограммах-событиях.

Для передачи целых, действительных и символьных данных из файла проекта в Решатель используются массивы Решателя

IUDat(1 .. 64) - массив типа integer,
RUDat(1 .. 64) - массив типа real(4),
CUDat(1 .. 64) - массив типа character(32),

Для пересылки значений в эти массивы используются операторы секции [User Data] с «аналогичными» именами.

2.2 Структурные переменные Решателя

2.2.1 Переменные для описания сетки КО

Ячейки (КО) структурной сетки описываются тремя группами одномерных сеточных массивов. Число контрольных объемов по осям содержится в переменных NX, NY, NZ. Если в направлении оси X отсутствует размерность, то NX=1. Если NX > 1, то число КО равно NX-2. Два «лишних» КО – это нулевые КО на внешних границах расчетной области. Схема КО для случая NX > 1 показана на рисунке 2.1, схема КО для NX=1 показана на рисунке 2.2.

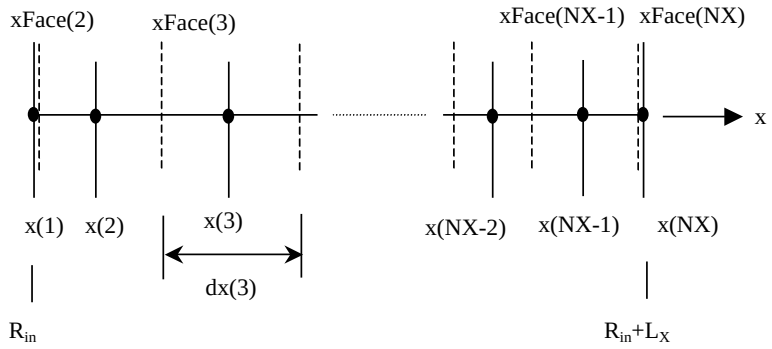

Рисунок 2.1 - Семейство сеток КО по оси x

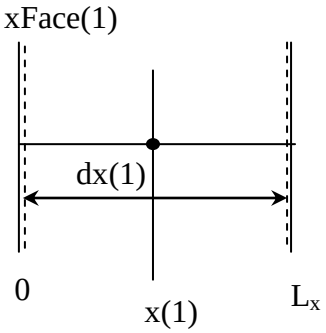

Рисунок 2.2 - Сетка с одним КО по координате x

Сетка по координате x описывается следующими массивами:
xFace(2:NX) – координата левой грани КО,
x(1:NX) – координата центра КО,
dX(1:NX) – ширина КО.
Характеристики КО с индексами (ix,iy,iz) следующие:

(x(ix),y(iy),z(iz))	- центр КО,
(xFace(ix+1),y(iy),z(iz))	- центр правой «e» грани КО,
(xFace(ix),y(iy),z(iz))	- центр левой «w» грани КО,
(x(ix),yFace(iy+1),z(iz))	- центр верхней «n» грани КО,
(x(ix),yFace(iy),z(iz))	- центр нижней «s» грани КО,
.(x(ix),y(iy),zFace(iz+1))	- центр дальней «h» грани КО,,
.(x(ix),y(iy),zFace(iz))	- центр ближней «l» грани КО.

Для описания материалов фаз и FS-моделей используются трехмерные массивы:

iFMAT = MatF(ix,iy,iz) - индекс материала G-фазы в зонах Flow или Porous,
iSMAT = MatS(ix,iy,iz) - индекс материала S-фазы в зонах Struct или Porous,
iFS = ModelFS(ix,iy,iz) - индекс модели межфазного G-S взаимодействия в зонах Porous.
Сами символьные имена материалов хранятся в символьных массивах
FlowMAT(iFMAT), StructMAT(iSMAT)
Для определения индекса материала по его имени можно использовать функции

```
iFMAT = GetFlowMatID(MatName)
iSMAT = GetStructMatID(MatName)
  character (*) MatName
```

Эти функции возвращают индекс материала или ноль, если такого материала нет.

Массивы индексов материалов удобно использовать для определения типа КО. Например, если
MatF(ix,iy,iz) = 0,

то КО не содержит G-фазы (он либо блокирован, либо содержит только S-фазу). Однако удобнее использовать специальные функции «раскраски» КО, которые используют индексы *текущего* КО (ixC,iyC,izC):

logical function IsBlockCV()
функция возвращает .TRUE., если КО блокирован,
logical function IsFlowCV()
функция возвращает .TRUE., если КО содержит G-фазу,
logical function IsStructCV()
функция возвращает .TRUE., если КО содержит S-фазу,

logical function IsOnlyFlowCV()
функция возвращает .TRUE., если КО содержит только G-фазу,
logical function IsOnlyStructCV()
функция возвращает .TRUE., если КО содержит только S-фазу,
logical function IsNoFlowCV()
функция возвращает .TRUE., если КО не содержит G-фазы,
logical function IsNoStructCV()
функция возвращает .TRUE., если КО не содержит S-фазы.

Геометрические размеры КО описываются 4 массивами:
Volume(ix,iy,iz) - объем КО,
AreaW(ix,iy,iz) - площадь левой «w» грани КО,
AreaS(ix,iy,iz) - площадь нижней «s» грани КО,
AreaL(ix,iy,iz) - площадь ближней «l» грани КО.

2.2.2 Ф-переменные

Все Ф-переменные хранятся в четырехмерном массиве
F8(1:NX, 1:NY, 1:NZ, 1:NoPHI)
где NoPHI – общее число решаемых Ф-переменных. Для получения индекса Ф-переменной по ее имени используется функция

```
integer function GetPHIID(PhiName)
  character(*) PhiName
```

которая по имени PhiName возвращает ее индекс. Если Ф-переменная не используется, то функция возвращает ноль. Для удобства для предопределенных Ф-переменных индексы Ф-переменных сохраняются в целых переменных Решателя

P_Pf, P_UGX, P_UGY, P_UGZ,
P_TG, P_TS,

```
P_kTur, P_epsTur,
P_CONC(8)
```

При решении нестационарной задачи дополнительно хранятся поля на предыдущем шаге по времени в массиве

```
F8OLD(1:NX, 1:NY, 1:NZ, 1:NoPHI)
```

При работе со структурной сеткой все Ф-переменные, за исключением компонент вектора скорости, привязаны к центру КО. Скорости задаются на гранях КО. Например:

```
F8(ix,iy,iz,P_PF)      - давление задано в точке [ x(ix),y(iy),Z(iz) ],
F8(ix,iy,iz,P_UGX)     - UgX задана на w-грани в точке [ xFace(ix),y(iy),Z(iz) ],
F8(ix,iy,iz,P_UGY)     - UgY задана на s-грани задана в точке [ x(ix),yFace(iy),Z(iz) ],
F8(ix,iy,iz,P_UGZ)     - UgZ задана на l-грани задана в точке [ x(ix),y(iy),ZFace(iz) ].
```

Если в направлении оси используется только один КО, то компонента скорости в этом направлении задается в центре КО.

Ф-переменные в блокированных КО обычно заполнены нулями и не используются в расчете. Однако приграничные блокированные КО используются для хранения граничных значений Ф-переменных. Нулевые КО на внешних границах базовой расчетной области (при ix = 1 и NX; iy = 1 и NY, iz = 1 и NZ) всегда помечаются как блокированные, что позволяет однотипным образом обрабатывать все границы РО. Например, пусть у КО с индексами (ixC, iyC, izC) правая грань «e» является граничной. Это значит, что КО (ixC+1,iyC,izC) будет блокированным и значение

```
F8(ix+1,iy,iz,iPHI)
```

будет содержать значение Ф-переменной с индексом iPHI *в центре правой грани* «e».

В процессе обработки часто возникает потребность переинтерполяции полей из центра КО на грани КО и обратно. Компоненты скорости в центре КО (ixC,iyC,izC) легко рассчитать по соотношениям:

```
UGX_P = 0.5*(F8(ixC,iyC,izC,P_UGX)+ F8(ixC+1,iyC,izC,P_UGX))
UGY_P = 0.5*(F8(ixC,iyC,izC,P_UGY)+ F8(ixC,iyC+1,izC,P_UGY))
UGZ_P = 0.5*(F8(ixC,iyC,izC,P_UGZ)+ F8(ixC,iyC,izC+1,P_UGZ))
```

Для расчета переменных, заданных в центре КО, на гранях КО необходимо использовать линейную интерполяцию и специальные массивы

```
cxP(1:NX),cxE(1:NX)     - для интерполяции на грань «e»,
cyP(1:NY),cyN(1:NY)     - для интерполяции на грань «n»,
czP(1:NZ),czH(1:NZ)     - для интерполяции на грань «h»
```

Например, для получения значения Ф-переменной на гранях «e» и «w» нужно использовать следующие операторы:

```
PHI_P = F8(ixC,iyC,izC,iPHI)
PHI_E = F8(ixC+1,iyC,izC,iPHI)
PHI_W = F8(ixC-1,iyC,izC,iPHI)
PHI_EE = PHI_P * cxP(ixC) + PHI_E * cxE(ixC)   ! значение на грани e
PHI_WW = PHI_W * cxP(ixC-1) + PHI_P * cxE(ixC-1)   ! значение на грани w
```

2.2.3 Дополнительные сеточные переменные Решателя

Наряду с Ф-переменными в Решателе используется дополнительные 3D массивы. Эти массивы имеют размерность (1:NX,1:NY,1:NZ) и привязываются к центрам КО:

```
RhoG(ix,iy,iz)      - плотность G-фазы,
NuG(ix,iy,iz)       - ламинарная кинематическая вязкость G-фазы,
LamdaG(ix,iy,iz)    - теплопроводность G-фазы,
```

Porous(ix,iy,iz) - пористость.
CpG(ix,iy,iz) - теплоемкость однокомпонентной G-фазы,
CoG(ix,iy,iz,iCONC) - теплоемкость многокомпонентной G-фазы (iCONC-порядковый номер Ф-переменной C_k),
NuTur(ix,iy,iz) - турбулентная кинематическая вязкость G-фазы.

При решении нестационарной задачи указанные выше массивы хранят поля на текущем шаге по времени. Значения на предыдущем шаге по времени хранятся в массивах

PorousOld(ix,iy,iz), RhoGOld(ix,iy,iz)

Коэффициент диффузионного переноса для текущей Ф-переменной (предопределенной или пользователя), связанной с G-фазой, в коде задается соотношениями вида (см. документ [1])

$$\Gamma_{\Phi} = \rho_g \left(\frac{\nu_g}{\sigma_{\Phi}} + \frac{\nu_t}{\sigma_{\Phi,t}} \right)$$

При решении дискретных уравнений для текущей Ф-переменной это коэффициент рассчитывается и хранится в массиве

GamG(ix,iy,iz)

При работе с виртуальными функциями для *задания источников и граничных условий* этот массив доступен пользователю. При переходе к решению другой Ф-переменной этот массив содержит уже другой диффузионный коэффициент. Поэтому в *подпрограммах-событиях* пользователя нужно перед использованием «обновить» этот массив. Для этого используется подпрограмма

subroutine userDiffCoef(iPHI)

здесь iPHI – индекс Ф-переменной.

Для Ф-переменных, связанных с S-фазой вместо массива GamG используется три массива

GamSX(ix,iy,iz), GamSY(ix,iy,iz), GamSZ(ix,iy,iz)

Если коэффициент диффузии изотропен, то все три массива содержат одно и то же значение. В подпрограммах-событиях надежнее всего обновить содержимое этих массивов с помощью подпрограммы userDiffCoef.

Для Ф-переменных связанных с Space-фазой, коэффициент диффузии хранится в массиве GamG и для его обновления также используется подпрограмма userDiffCoef.

При использовании моделей турбулентности с пристенными функциями при расчете потоков на гранях с признаком IsWALL коэффициенты диффузии подменяются на эффективные значения, рассчитанные с помощью пристенных функций. При этом эти значения нельзя сохранять в массиве GamG, поскольку в нем хранится значение для связи с другими КО. Для получения этого модифицированного значения нужно использовать функцию

real function staGamGWall(iPHI,ix,iy,iz)

Эта функция носит универсальный характер. Ее можно вызывать всегда. Если данный КО не содержит пристенных функций, то будет возвращено значение GamG.

Рассмотрим типичный пример.

Пусть у КО (ix,iy,iz) имеется граничная грань «e» с условием прилипания (это может быть граница типа BSWall или граница раздела G и S фаз). Для ламинарного течения полное трение на этой грани можно рассчитать с помощью операторов:

```
call userDiffCoef(P_UGY)
……..

U_P = 0.5*(F8(ix,iy,iz,P_UGY) + F8(ix,iy+1,iz,P_UGY))
TAU_E = GamG(ix,iy,iz)*U_P/(0.5*dX(ix))
```

Для расчета турбулентного трения лучше всего использовать другие операторы:

```
call userDiffCoef(P_UGY)
……..

U_P = 0.5*(F8(ix,iy,iz,P_UGY) + F8(ix,iy+1,iz,P_UGY))
TAU_E = staGamGWall(P_UGY,ix,iy,iz)*U_P/(0.5*dX(ix))
```

Эти операторы будут правильно работать для всех моделей турбулентности, включая ламинарные режим.

2.2.4 Поля пользователя

Пользователь может использовать при работе с виртуальными функциями и подпрограммами-событиями любое число своих трехмерных массивов. Для создания таких массивов лучше всего использовать внутренние сеточные User-переменные пользователя.

Внутренние User-переменные (для них не задается алгоритм расчета оператором Calc) и обычные User-переменные (для них задается алгоритм расчета с помощью оператора Calc) хранятся в массиве:

```
F4(1:NX,1:NY,1:NZ, 1:NoUserFields)
```

Последний индекс в этом массиве – это индекс сеточной User-переменной. Этот индекс можно определить по имени переменной с помощью функции

```
 integer function GetUserID(UserName)
integer function GetInternalUserID(UserName)
  character(*) UserName
```

Первая функция возвращает индекс для любой сеточной User-переменной, вторая – только для внутренней User-переменной (эта функция для не внутренней возвратит ноль).

Главное достоинство использования сеточных внутренних User-переменных заключается в том, что они автоматически сохраняются в файле результатов.

Пользователь может ввести свои трехмерные массивы в секции [User Fortran Fielfs]. В этом случае в коде для доступа к этим массивам нужно использовать трехмерный массив Фортрана

```
<Имя массива>(1:NX,1:NY,1:NZ)
```

Главный «минус» таких массивов - они не сохраняются в файле результатов.

2.2.5 Обработка ячеек расчетной области

Типичный алгоритм перебора ячеек – это трехмерный цикл. При этом рекомендуется в качестве переменных-индексов использовать глобальные переменные (ixC,iyC,izC), которые используются в большинстве функций Решателя, связанных с сеткой КО. Например:

```
do izC = IPZF,IPZL
  do iyC = IPYF,IPYL
    do ixC = IPXF,IPXL
```

```
        if(.NOT. IsFlowCV()) CYCLE     ! если КО блокированный, пропускаем
          <Обработка текущего КО >
      end do
    end do
  end do
```

Несколько комментариев:

1) Использование переменных IPXF, ... в качестве границ перебора КО гарантирует правильность перебора, как для трехмерных задач, так и для двумерных. Например, при NX > 1 переменные равны IPXF = 2, IPXL = NX-1. Если NX=1, то IPXF = IPXL = 1.
2) В этом цикле перебираются только внутренние КО, нулевые КО не рассматриваются.
3) Функция Решателя IsBlockCV() работает правильно только потому, что пользователь для организации цикла использует глобальные индексы текущего КО (ixC,iyC,izC).

2.2.6 Патчи

В Решателе патчи представляют собой либо список КО, либо список граней КО. При работе со структурной сеткой для описания патчей используется структура TPatch и массив Patches() (ниже перечислены только поля структуры, представляющие интерес для пользователя):

```
  Type TPATCH
……
    integer IXF,IXL,IYF,IYL,IZF,IZL
    logical  L_MAPCV
    real(4)    C_VOL,  S_VOL
    real(4)    C_AREA, S_AREA
    integer (2) DefMapCV
    integer (2), pointer :: MapCV(:,:,:)
  end type

  Type(Tpatch), pointer :: Patches(:)
```

Ячейки объемного патча и грани поверхностного патча описываются с помощью «параллепипеда» ячеек. Границы этого структурного бокса описываются переменными IXF,IXL,IYF,IYL,IZF,IZL. Не все КО этого бокса используются патчем. Для пометки используемых ячеек используется массив флагов MapCV(ix,iy,iz). Для проверки флагов используются следующие индексы битов этого массива:

CVvolume - КО принадлежит патчу,

EFace - левая «е» грань КО принадлежит поверхностному патчу,

WFace, NFace, SFace, HFace, LFace - соответствующие грани КО принадлежат поверхностному патчу.

Если ни один бит не установлен (MapCV() = 0), то данный КО не используется в патче. Для «упаковки» массива MapCV используется следующий подход. Если все КО бокса имеют один и тот же MapCV, то массив MapCV не используется. В этом случае используется скалярная переменная DefMapCV. Признаком использования массива MapCV является значение L_MAPCV = .TRUE.

Для получения индекса патча в массиве Patches() по имени нужно использовать функцию

```
integer function  GetindexPatch(PatchName)
  character(*) PatchName
```

После получения индекса для работы с конкретным патчем удобнее всего использовать ссылку Фортрана, например:

```
Type(TPatch), pointer:: CPatch
 ……
iPATCH = GetIndexPatch("CylWall")
if(iPHATCH.NE.0) Cpatch => Patches(iPATCH)
```

Дополнительные поля патча содержат его геометрические характеристики:
C_VOL - объем исходного патча Компилятора, для поверхностного патча = 0,
S_VOL - объем патча Решателя (сумма объемов всех используемых КО), именно это значение возвращает myFORM-функция VolPatch (см. пункт 2.4.2 документа [2]),
C_AREA - площадь исходного патча Компилятора, для объемного патча = 0,
S_AREA - площадь патча Решателя (сумма площадей всех используемых граней КО), именно это значение возвращает myFORM-функция AreaPatch.
Приведем пример типичной обработки как объемного, так и поверхностного патча:

```
Type(TPatch), pointer:: CPatch
 ……
iPATCH = GetIndexPatch("CylWall")
Cpatch => Patches(iPATCH)
!  --- Перебор всех КО патча ---
  DO IzC = CPatch%IZF,CPatch%IZL
    DO IyC = CPatch%IYF,CPatch%IYL
      DO IxC = CPatch%IXF,CPatch%IXL
!      --- биты для текущего КО патча ---
        if(CPatch%L_MAPCV) then
           iMap = CPatch%MapCV(IxC,IyC,IzC)
        else
           iMap = CPatch%DefMapCV
        endif
        if(iMap.EQ.0) CYCLE      ! КО не используется в патче
!       ------ Объемный КО  -------
        if(BTEST(iMap, CVvolume)) then
          <Обработка КО>
        endif
!       ------ грань E является гранью поверхностного патча? -------
        if(BTEST(cMap,EFace)) then
                 <Обработка КО>
        endif
     ……..
      ENDDO
   ENDDO
 ENDDO
```

2.3 НеСтруктурные переменные Решателя

2.3.1 Переменные для описания сетки КО

Неструктурная сетка в Решателе описывается тремя одномерными списками: ячеек, внутренних граней и граничных граней. Для ячеек используется одномерный массив объектов типа TCell:

```
type TCell
  integer (2) IX,IY,IZ       ! Fine индексы данного КО
  integer (1) level          ! уровень разбиения (счет с 0)
  real (4) Volume            ! объем ячейки
  real (4) Centre(3)         ! координаты центра ячейки
End type
Type (TCell)   Cells(1:NoCells)
```

Поле Volume содержит объем ячейки, Centre(3) – координаты (x,y,z) центра ячейки в используемой системе координат (декартовой или цилиндрической).

Неструктурные сетки Anes «наследуют» часть структурной сетки, которая использовалась для ее построения:

ix,iy,iz - структурные индексы самой мелкой структурной сетки (fine-индексы),

level - уровень дробления данной ячейки, 0 означает исходную грубую сетку, MaxLevel – уровень самой мелкой сетки.

Fine-индексы ячейки удобно использовать для структурной «навигации» по неструктурным ячейкам (см. раздел 1.5 документа [3]). Именно эти индексы используются в постпроцессоре Anes для анализа полей. Число Fine-ячеек по осям координат содержится в переменных NFX, NFY, NFZ.

Для описания внутренних граней используется массив объектов типа TFace:

```
Type TFace
  integer(4) posCellID, negCellID     ! индексы Соседних ячеек
  real (4)   Area                     ! площадь грани
  real (4)   Centre(3)                ! координаты центра тяжести грани
  real(4)   PN(3)                     ! единичная нормаль грани
End type

Type (TFace)   FaceXYZ(1:NoFaceXYZ)
```

Переменные posCellID, negCellID содержат индексы ячеек в массиве Cells(), прилегающих к грани. Положение грани в пространстве определяется ее нормалью PN(3). Нормаль граней, связывающих ячейки одного типа (Flow, Struct или Porous), всегда направлена в направлении к ячейке posCellID. Для граней, разделяющих Flow-Struct зоны и Porous-Flow, Porous-Struct зоны, нормаль направлена всегда в сторону Flow-ячейки.

Для описания *граничных* граней используется массив объектов типа TFaceBS

```
Type TFaceBS
  integer(4) CellID                   ! индексы КО объемов
  real(4)   Area                      ! площадь грани
  real(4)   Centre(3)                 ! координаты центра тяжести грани
  real(4)   PN(3)                     ! ВНУТРЕННЯЯ единичная нормаль, направлена внутрь РО
  real(4)   Dist                      ! Длина нормали из центра ячейки
End type

Type (TFaceBS)  FaceBS(1:NoFaceBS)
```

Отличие этих объектов от объектов внутренних граней заключается в следующем:

1) грань связывается только с одной ячейкой с индексом CellID,
2) нормаль грани PN(3) всегда направлена в расчетную область (в сторону ячейки CellID),
3) поле Dist содержит расстояние от центра ячейки CellID до плоскости грани.

Для описания материалов фаз и FS-моделей используются одномерные массивы, связанные с ячейками:

iFMAT = usMatF(idCV) - индекс материала G-фазы в зонах Flow или Porous,

iSMAT = usMatS(idCV) - индекс материала S-фазы в зонах Struct или Porous,

iFS = usModelFS(idCV) - индекс модели межфазного G-S взаимодействия в зонах Porous.

Эти массивы можно использовать для определения типа КО. Например, если

usMatF(idCV) = 0

то КО не содержит G-фазы (он содержит только S-фазу). Заметим, что в отличие от структурной сетки в неструктурной сетке нет блокированных ячеек (они были отброшены на этапе создания сетки).

При работе с неструктурными сетками используются понятие текущих индексов ячейки – idCV и грани idFACE, которые аналогичны текушим структурным индексам (ixC,iyC,izC).

2.3.2 Ф-переменные

Все Ф-переменные при работе с неструктурной сеткой хранятся в двумерном массиве

```
usF8(1:NoCells, 1:NoPHI)
```

где NoPHI – общее число Ф-переменных. Для получения индекса Ф-переменной по ее имени используется та же функция, что и для структурной сетки

```
integer function GetPHIID(PhiName)
 character(*) PhiName
```

которая по имени PhiName возвращает ее индекс. Если Ф-переменная не используется, то функция возвращает ноль. Для удобства для предопределенных Ф-переменных индексы Ф-переменных сохраняются в целых переменных Решателя

```
P_Pf, P_UGX, P_UGY, P_UGZ,
P_TG, P_TS,
P_kTur, P_epsTur,
P_CONC(8)
```

При решении нестационарной задачи дополнительно хранятся поля на предыдущем шаге по времени в массиве

```
usF8OLD(1:NoCells, 1:NoPHI)
```

При работе с неструктурной сеткой все Ф-переменные, включая компоненты вектора скорости, хранятся в центре ячейки Cells()%Centre(3). В структурной сетке граничные значения Ф-переменных хранятся в приграничных блокированных КО. В неструктурной сетке граничные значения Ф-переменных хранятся в двумерном массиве, связанном с граничными гранями (FaceBS()):

```
usFBV(1:NoFaceBS,1:NoPHI)
```

Для реализации дискретных уравнений и записи результатов расчета в неструктурных сетках Решатель рассчитывает градиент для каждой Ф-переменной. Градиент сохраняется в массиве

```
clGrad(1:3,1:NoCells, 1:NoPHI)
```

Первый индекс – это направление оси (1 - x, 2 - y, 3 - z), второй индекс – это индекс ячейки, третий индекс – индекс Ф-переменной. Градиент позволяет рассчитать со вторым порядком точности значение Ф-переменной в любой точке ячейки. Такой пример приведен ниже при описании патчей.

2.3.3 Дополнительные сеточные переменные Решателя

Наряду с Ф-переменными в Решателе используется дополнительные 3D массивы. Эти массивы имеют размерность (1:NoCells) и привязываются к центрам ячеек:

usRhoG(idCV)	- плотность G-фазы,
usNuG(idCV)	- ламинарная кинематическая вязкость G-фазы,
usLamdaG(idCV)	- теплопроводность G-фазы,
usPorous(idCV)	- пористость.
usCpG(idCV)	- теплоемкость однокомпонентной G-фазы,
usCoG(idCV,iCONC)	- теплоемкость многокомпонентной G-фазы (iCONC - порядковый номер Ф-переменной C_k),
usNuTur(ix,iy,iz)	- турбулентная кинематическая вязкость G-фазы.

При решении нестационарной задачи указанные выше массивы хранят поля на текущем шаге по времени. Значения на предыдущем шаге по времени хранятся в массивах

```
usPorousOld(idCV), usRhoGOld(idCV)
```

Коэффициент диффузионного переноса для текущей Ф-переменной (предопределенной или пользователя), связанной с G-фазой, в коде задается соотношениями вида (см. документ [1])

$$\Gamma_{\Phi} = \rho_g \left(\frac{\nu_g}{\sigma_{\Phi}} + \frac{\nu_t}{\sigma_{\Phi,t}} \right)$$

При решении дискретных уравнений для текущей Ф-переменной это коэффициент рассчитывается и хранится в массиве

clGamG(idCV)

При работе с виртуальными функциями для задания источников и граничных условий этот массив доступен пользователю. При переходе к решению другой Ф-переменной этот массив содержит уже другой диффузионный коэффициент. Поэтому в *подпрограммах-событиях* пользователя нужно перед использованием «обновить» этот массив. Для этого используется подпрограмма

```
subroutine userDiffCoef(iPHI)
```

здесь iPHI – индекс Ф-переменной.

Для Ф-переменных, связанных с S-фазой вместо массива GamG используется три массива

clGamSX(idCV), clGamSY(idCV), clGamSZ(idCV)

Если коэффициент диффузии изотропен, то все три массива содержат одно и то же значение. В подпрограммах-событиях надежнее всего обновить содержимое этих массивов с помощью подпрограммы userDiffCoef.

Для Ф-переменных связанных с Space-фазой, коэффициент диффузии хранится в массиве GamG и для его обновления используется userDiffCoef.

При использовании моделей турбулентности с пристенными функциями при расчете потоков на гранях с признаком IsWALL коэффициенты диффузии подменяются на эффективные значения, рассчитанные с помощью пристенных функций. При этом эти значения нельзя сохранять в массиве GamG, поскольку в нем хранится значение для связи с другими КО. Для получения этого модифицированного значения нужно использовать функцию

```
real function usaGamGWall(iPHI,idCV)
```

Эта функция носит универсальный характер. Ее можно вызывать всегда. Если данный КО не содержит пристенных функций, то будет возвращено значение clGamG.

2.3.4 Поля пользователя

Пользователь может использовать при работе с виртуальными функциями и подпрограммами-событиями любое число своих массивов, связанных с ячейками неструктурной сетки. Для создания таких массивов нужно использовать внутренние сеточные User-переменные пользователя (см. пункт 2.2.4).

Внутренние User-переменные (для них не задается алгоритм расчета оператором Calc) и обычные User-переменные (для них задается алгоритм расчета с помощью оператора Calc) хранятся в массиве:

```
usF4(1:NoCells, 1:NoUserFields)
```

Последний индекс в этом массиве – это индекс сеточной User-переменной. Этот индекс можно определить по имени переменной с помощью функции

```
 integer function GetUserID(UserName)
integer function GetInternalUserID(UserName)
```

Первая функция возвращает индекс для любой User-переменной, вторая – только для внутренней User-переменной (эта функция для не внутренней возвратит ноль).

Главное достоинство использования сеточных внутренних User-переменных заключается в том, что они автоматически сохраняются в файле результатов.

Пользователь может ввести свои трехмерные массивы в секции [User Fortran Fielfs]. В этом случае в коде для доступа к этим массивам нужно использовать трехмерный массив Фортрана

```
<Имя массива>(1:NoCells)
```

Главный «минус» таких массивов - они не сохраняются в файле результатов.

2.3.5 Обработка ячеек расчетной области

Типичный алгоритм перебора ячеек – это одномерный цикл, в котором для отбора ячеек можно использовать массивы материалов и моделей usMatF(:),usMatS(:),usModelFS(:). При этом рекомендуется в качестве переменной-индекса использовать глобальную переменную idCV, которая используется в большинстве функций Решателя, связанных с сеткой КО. Например:

```
do idCV = 1,NoCells
   if(usMatF(idCV).EQ.0) CYCLE          ! отбираем ячейки только с G-фазой
         <Обработка текущего КО >
end do
```

2.3.6 Патчи

Как и для структурной сетки, для неструктурной в Решателе патчи представляют собой либо список КО, либо список граней между КО. Для описания этих списков используются *другие поля* структуры TPatch:

```
 Type TPATCH
……
   real(4)   C_VOL,  S_VOL
   real(4)   C_AREA, S_AREA
   integer NoCells, NoFaces
   integer (4), pointer :: clList(1:NoCells)
   integer (4), pointer :: fcList(1:NoFaces)
end type

 Type(Tpatch), pointer :: Patches(:)
```

Если патч является объемным, то список индексов ячеек (индексы в массив Cells) хранится в массиве clList(1:NoCells) патча, а массив граней fcList не используется.

Для поверхностного патча используются массив fcList(1:NoFaces), который содержит индексы граней в:

FacesBS(:) – для граничных патчей типа BS и BSWall,

FaceXYZ(:) – для FS-патчей (границы раздела G и S фаз).

Для получения индекса патча в массиве Patches() по имени нужно использовать функцию

```
integer function GetindexPatch(PatchName)
  character(*) PatchName
```

После получения индекса патча для работы с ним удобнее всего использовать ссылку Фортрана, например:

```
Type(TPatch), pointer:: CPatch
 ……
iPATCH = GetIndexPatch(“CylWall”)
if(iPATCH.NE.0) Cpatch => Patches(iPATCH)
```

Дополнительные поля патча содержат геометрические характеристики:

C_VOL - объем исходного патча Компилятора, для поверхностного патча = 0,

S_VOL - объем патча Решателя (сумма объемов всех используемых КО), именно это значение возвращает myFORM-функция VolPatch,

C_AREA - площадь исходного патча Компилятора, для объемного патча = 0,

S_AREA - площадь патча Решателя (сумма площадей всех используемых граней КО), именно это значение возвращает myFORM-функция AreaPatch.

Приведем пример типичной обработки поверхностного граничного патча с именем "myWALL" с целью расчета составляющих сил, действующих на поверхность патча. Для анализа разделим эти силы на две составляющие: первая связана с полем давления (ForceP(3)), вторая – с трением потока на поверхности (ForceTAU(3)).

$$\mathbf{F}_p = -\oint p\mathbf{n}dA, \quad \mathbf{F}_\tau = \oint \boldsymbol{\tau}_w dA$$

здесь $\mathbf{n}$ – вектор нормали к поверхности патча, направленный в расчетную область, $\boldsymbol{\tau}_w$ – вектор касательного напряжения, направленный в направлении касательной составляющей скорости потока.

При использовании неструктурных сеток и модели дробных ячеек грань может быть наклонной. Ниже для расчета касательной составляющей скорости используются сервисные подпрограммы Решателя, предназначенные для работы с векторами. Подробнее эти подпрограммы описаны ниже. Итак, текст с комментариями:

```
! ---- Объявления локальных переменных ---
Type(TPatch), pointer:: CPatch
  real VEL(3),V_NORM(3),VEL_TAU(3),VEL_NORM(3)  ! Это вектора
  real P_CELL(3), P_FACE(3),pfGRAD(3)
  real ForceP(3), ForceTAU(3)
 ……
iPATCH = GetIndexPatch(“mylWall”)
Cpatch => Patches(iPATCH)

ForceP(1:3) = 0.0          ! Начальные значения векторов сил
ForceTAU(1:3)  = 0.0

   NFC = CPatch%NoFaces      ! Число граней в патче
!
!  --- на всякий случай обновляем диффузионный коэффициент для скоростей
!      в качестве пргумента можно указать индекс любой используемой компоненты
   CALL userDiffCoef(P_UGX)

!  ----Перебираем все грани патча --
   do iff =1, NFC
     iFACE = Cpatch%fcLIst(iff)       ! Это индекс грани в массиве FaceBS()
     iCELL  = FaceBS(iFACE)%CellID  ! Это индекс приграничной ячейки
     P_FACE = FaceBS(iFACE)%Centre   ! Это вектор центра грани
    Area   = FaceBS(iFACE)%Area         ! Площадь грани
    V_NORM = FaceBS(iFACE)%PN    ! Вектор нормали, направлена в сторону РО
     FDist  = FaceBS(iFACE)%Dist      ! Расстояние отцентра ячейки до грани
     P_CELL = Cells(iCELL)%Centre    ! Это вектор центра ячейки
!    --- P-составляющая  силы ----------
     PF_CELL0 = usF8(iCELL,P_PF)    !  Значение давления в ячейке
     pfGRAD = clGRAD(1:3,iCELL,P_PF)    ! Значение градиента в ячейке
!      --- Расчет давления в центре грани ---
     PF_FACE = PF_CELL0   + DotProduct(pfGRAD,P_FACE-P_CELL)
     ForceMOD = PF_FACE * Area            ! Модуль силы давления
     do iv=1,3         ! Компоненты силы направлены против нормали
       ForceP(iv) = ForceP(iv) - ForceMOD * V_NORM(iv)
     enddo
```

```
!     --- TAU-составляющая ----------
      VEL = 0.0     ! Заполняем составляющие вектора скорости в центре ячейки
      if(P_UGX.NE.0)  VEL(1) = usF8(iCELL,P_UGX)
      if(P_UGY.NE.0)  VEL(2) = usF8(iCELL,P_UGY)
      if(P_UGZ.NE.0)   VEL(3) = usF8(iCELL,P_UGZ)
!     --- Рассчитываем вектора скорости по нормали VEL_NORM и по касательной
!        VEL_TAU
      call usaTAUVelocity(VEL,V_NORM,VEL_TAU,VEL_NORM)
      VEL_TAU_MOD = VecMod(VEL_TAU)
!     ---  Значение эффективного коэффициента вязкости --
      aMuEFF = usaGamGWall(iVEL,iCELL)
!     --- касательная сила направлена в направлении VEL_TAU
      do iv=1,3
        ForceTAU(iv) = ForceTAU(iv) + aMuEFF*VEL_TAU(iv)/FDist * Area
      enddo
    enddo
```

В этом коде используется подпрограмма usaGamGWall и функция VecMod. Описание этих функций ниже.

Много аналогичных примеров можно найти в каталоге openfor Anes.

3. Универсальные подпрограммы пользователя

В этом разделе описаны сервисные подпрограммы Решателя, которые можно использовать как для структурных, так и для неструктурных сеток.

3.1 Функции доступа

Эти функции можно использовать внутри виртуальных функций, которые вызываются внутри циклов перебора ячеек (источники, свойства, граничные условия). Функции можно использовать и в подпрограммах-событиях. Важно, чтобы были определены текущие индексы КО: (ixC,iyC,izC) для структурной сетки и idCV для неструктурной:

subroutine userXYZ(XP,YP,ZP)

подпрограмма возвращает координаты центра текущего КО (xP,yP,zP).

real(8) function userPHI(iPHI)

функция возвращает значение любой Ф-переменной в центре текущего КО.

real(8) function userGrad(iDIR, iPHI)

функция возвращает значение компонента градиента Ф-переменной с индексом iPHI в текущем КО в направлении iDIR.

subroutine userUG_XYZ(UX,UY,UZ)

подпрограмма возвращает значение компонент вектора скорости G-фазы.

subroutine userG_PROP(aDens, aVisc,aCP,aLamda)

подпрограмма возвращает значение основных свойств G-фазы (плотности, кинематической вязкости, теплоемкости и теплопроводности) в текущем КО.

subroutine userS_PROP(aDens, aCP, aLamda)

подпрограмма возвращает значение основных свойств S-фазы (плотности, теплоемкости и теплопроводности) в текущем КО.

3.2 Функции для работы с векторами

Эти функции можно использовать для работы с векторами. Вектор в Решателе – это одномерный массив из трех элементов типа real(4).

```
real function DotProduct(x1,x2)
  real x1(3),x2(3)
```

функция возвращает скалярное произведение двух векторов.

```
subroutine CrossProduct(x1,x2,Cross)
 real x1(3),x2(3),Cross(3)
```

подпрограмма возвращает векторное произведение Cross двух векторов x1 и x2.

```
 real function VecMod(x)
  real x(3)
```

функция рассчитывает модуль вектора.

```
subroutine VecNorm(V)
 real V(3)
```

функция нормализует вектор (его модуль будет равен 1).

```
logical function VecEqual(V1,V2)
 real V1(3),V2(3)
```

функция выдает .TRUE. , если два вектора совпадают с машинной точностью.

```
subroutine usaTAUVelocity(VEL,V_NORM,VEL_TAU,VEL_NORM)
 real VEL_TAU(3), VEL_NORM(3), VEL(3)
 real V_NORM(3)
```

входными параметрами подпрограммы являются исходный вектор VEL и вектор нормали к плоскости V_NORN. Подпрограмма рассчитывает два вектора VEL_NORM и VEL_ TAU. Первый вектор направлен по нормали, а его длина равна проекции вектора VEL (это вектор-проекция на нормаль), второй вектор является проекцией вектора VEL на плоскость (касательная составляющая вектора).

3.3 Подпрограммы обработки. Расчет средних величин

Данная группа подпрограмм предназначена для расчета интегральных потоков на патчах:

3.3.1 Полный поток/источник Ф-переменной на патче

```
real function acTotFluxPHI(PhiName,PatchName)
 character(*)PhiName,PatchName
```

Это универсальная подпрограмма вычисляет *полный* источник для Ф-переменной с именем PhiName на патче PatchName. В текущей версии подпрограмма работает с патчами типа:

BS, BSWall - граничные поверхностные патчи,
FS_ - поверхностные патчи, описывающие G-S границу,
Volume, Point – объемный патч, на котором задается источник пользователя,
Surface – поверхностный патч пользователя.

На поверхностных патчах подпрограмма суммируем полные потоки Ф-переменной, умноженные на площадь грани. Для объемных патчей источник умножается на объем КО и суммируется. В обоих случаях знак потока (источника) считается положительным, если он «направлен» в расчетную область.

Для Ф-переменной «PF» в качестве потока/источника используется плотность массового потока. При расчете потоков для температуры TG подпрограмма использует в массовой составляющей *полные значения* энтальпий (см. подробнее [1]):

$$J_{Tg,k} = \left(\varphi \rho_g U_{g,k} H_g - \varphi \lambda_{eff} \frac{\partial T_g}{\partial x_k} \right), \qquad (3.1)$$

$$H_g = h_g + H^0$$

Для остальных Ф-переменных

$$J_{\Phi,k} = \varphi \left(\rho_g U_{gk} \Phi - \Gamma_\Phi \frac{\partial \Phi}{\partial x_k} \right), k = x,y,z \quad : \text{для } G - \text{фазы}, \qquad (3.2)$$

$$J_{\Phi,k} = -(1-\varphi)\Gamma_\Phi \frac{\partial \Phi}{\partial x_k}, k = x,y,z \quad : \text{для } S - \text{фазы},$$

Функция myFORM TotFlux использует эту функцию для вычисления полных потоков.

Заметим, что код этой функции (и всех других сервисных функций) можно посмотреть в каталоге openfor\source\usersub.

3.3.2 Силы, действующие на патч

```
subroutine acForceOnPatch(PatchName,ForceP,ForceTAU)
 character(*) PatchName
 real ForceP(3), ForceTAU(3)
```

Эта подпрограмма рассчитывает полные силы, действующие на поверхностном патче типа BS, BSWall или FS_ (см. geyrn 2.3.6). Вектор ForceP(3) содержит компоненты вектора силы, связанной с давлением, вектор ForceTAU(3) – с трением на поверхности тела.

3.3.3 Полный источник пористого взаимодействия Ф-переменной

Эта функция является аналогом функции acTotFlux, но она вычисляет интеграл от источника пористого взаимодействия (вид этого источника зависит от Ф-переменной – смотрите документ [1]) не на патче, а на пористой зоне с именем FS-зона. Напомним, что имя FS-зоны задается последним параметром пористого патча.

```
real function acTotPorousSource(PHIName, NameFS_zone)
 character(*)PHIName, NameFS_zone
```

Если для Ф-переменной в модели межфазного взаимодействия не предусмотрено межфазное взаимодействие, то функция возвращает ноль.

3.3.4 Интегральный нестационарный член

Для проверки балансов для произвольной Ф-переменной в нестационарных задачах необходимо определить интегральное значение нестационарного члена. Для этого можно воспользоваться функцией

```
real function acTotTimeSource(PHIName)
 character(*)PHIName
```

Эта функция рассчитывает интеграл по объему расчетной области от нестационарного члена. Вид нестационарного члена зависит от типа Ф-переменной. Для произвольной Ф-переменной, связанной с G-фазой

$$S_{time} = \int_{PO} \frac{\partial\left(\rho_g \varphi \Phi\right)}{\partial \tau} dV$$

Заметим, что если рассматривать нестационарный член как источник, то необходимо использовать значение с знаком минус:

$-S_{time}$

3.3.5 Интегральный поток массы на патче типа Surface

Для расчета потока массы (в кг/с) на плоском патче типа SURFACE можно воспользоваться функцией

```
real function acMassOnSurface (PatchName)
 character(*) PatchName
```

Эта функция рассчитывает интеграл по поверхности патча от величины:

$m_n = \rho \varphi U_n$

3.3.6 Интегральные значения Ф-переменной на патче

```
subroutine acAveragePHI(PHIName,PatchName,VALAVE,VALMIN,VALMAX)
 character(*)PHIName,PatchName
```

Эта подпрограмма рассчитывает среднее VALAVE, минимальное VALMIN и максимальное VALMAX значения Ф-переменной с именем PHIName на патче типа BS, BSWall или FS_ и с именем PatchName.

3.3.7 Среднее значение Ф-переменной в объеме

```
real function acAverageSect(PhiName,iDIR,sBEG,sEND)
```

Функция рассчитывает среднеобъемное значение Ф-переменной PhiName в части расчетной области, ограниченной двумя сечениями sBEG .. sEND в направлении оси iDIR (1 –x, 2 –y, 3-z).

3.4 Подпрограммы для обработки результатов

3.4.1 Интерполяция

Подпрограммы данной группы предназначены для интерполяции Ф-переменных и переменных пользователя вдоль заданной линии или в заданной точке.

```
subroutine userPHIonLINE(iPHI,iDIR,XP,YP,ZP,noV,vRET)
 real, pointer:: vRET(:,:)
```

Подпрограмма рассчитывает распределение Ф-переменной с индексом iPHI вдоль линии в направлении оси iDIR. Эта линия проходит через точку PO (xP,yP,zP). При построении линии используются только две компоненты этой точки, например, для оси x (iDIR = 1) координата xP не используется. Алгоритм работы программы следующий (как для структурной, так и для неструктурной сеток):

1) сначала отбираются все КО, через которые проходит линия,
2) для каждого КО опускается перпендикуляр из центра ячейки на линию и в этой точке пересечения с помощью интерполяции вычисляется значение Ф-переменной,
3) пара значений (координата на линии и значение Ф) помещается в массив vRET(1:2,noV),
4) элементы массива сортируются в порядке возрастания координаты vRET(1,noV).

При обращении к функции память массиву vRET должна быть выделена с помощью оператора allocatable. Для гарантирования нужного размера массива по второй координате нужно использовать размеры PO, хранящиеся в переменных NX,NY,NZ для структурной сетки и в массиве 2*NFZ, 2*NFY, 2*NFZ для неструктурной.

Индекс Ф-переменной можно получить по имени Ф-переменной с помощью функции

```
iPHI =  GetPHID(UserName)
```

```
subroutine userFUSERonLINE(iUSER,iDIR,XP,YP,ZP,noV,vPHI)
 real, pointer:: vPHI(:,:)
```

Это подпрограмма аналогична предыдущей. Главное отличие – она позволяет получить распределение по линии для User-пременной пользователя с индексом

iUSER. Напомним, что этот индекс можно получить по имени переменной с помощью функции

```
iUSER =  GetUserID(UserName)
```

```
real function userPHIonPOINT(iPHI,XP,YP,ZP)
```

Это функция возвращает значение Ф-переменной в произвольной точке (xP,yP,zP).

Пример использования данных функций буден приведен ниже при описании подпрограмм работы с файлами.

Приведем еще одну «необычную» функцию обработки, которая может быть удобна для обработки различных «предельных» распределений. Например, у нас есть распределение давления по каналу и на стабилизированном участке это распределение становится линейным. Для аппроксимации этого участка можно использовать функцию:

```
subroutine userLinearLSM(iFIRST,iLAST,F,Z,ACoef,BCoef)
 real, pointer:: F(:), Z(:)
```

Эта функция, используя часть элементов массивов Z(iFIRST,iLAST) и F(iFIRST,iLAST), рассчитывает методом наименьших квадратов аппроксимацию линейной зависимостью

```
   F = ACoef*Z +BCoef
```

Для интерполяции табличных данных используется функция

```
real function userLinInterTable(xCur,tabX,tabF,NoPoint)
  real:: tabX(NoPoint), tabF(NoPOint)
```

Функция использует линейную интерполяцию по точкам таблицы {tabX(i), tabF(i)} для расчета значения при x = xCur. Если аргумент выходит за границы изменения аргумента, то используются граничные значения функции tabF(1) и tabF(NoPoint).

3.4.2 Функции сортировки

При работе с неструктурными ячейками и гранями следует помнить, что и ячейки и грани в списках расположены в «произвольном» порядке. При обработке результатов возникает проблема упорядочивания распределений по ячейкам и по граням. Для этого можно использовать функции сортировки, описанные ниже.

```
subroutine userSortQUICK_ARG(AR2,NoItem,IRC)
  real, pointer:: AR2(:,:)
```

Эта подпрограмма сортирует в порядке возрастания элементы массива AR2(1:2,1:NoItem) по элементам AR2(1,:). В результате работы элементы массива AR2(1:2,:) в порядке возрастания первого столбца. Обычно массив AR2 представляет собой «неотсортированное» одномерное распределение переменной, при этом AR2(1,:) – это аргумент, AR2(2,:) – это значение переменной. Последний аргумент iRC возвращает код выполнения сортировки. Если iRC = 0, то сортировка прошла успешно.

```
subroutine userSortQUICK_ID(AR1,ID,NoItem,IRC)
  real, pointer:: AR1(:)
  integer, pointer :: ID(:)
```

Если необходимо отсортировать распределение нескольких переменных, то нужно использовать данную подпрограмму. Эта подпрограмма одновременно сортирует действительный массив AR1(1:NoItem) и целый массив ID(1:NoItem) в порядке воздастания элементов AR1.

Приведем пример использования этих подпрограмм. Пусть нужно рассчитать распределения параметров по поверхности цилиндра и построить зависимости этих параметров от угла атаки цилиндра (это модификация примера из пункта 10.3.6):

```
! ---- Объявления локальных переменных ---
Type(TPatch), pointer:: CPatch
  real VEL(3),V_NORM(3),VEL_TAU(3),VEL_NORM(3)  ! Это вектора
  real P_CELL(3), P_FACE(3),pfGRAD(3)
  real, pointer ForceP(:,:), ForceTAU(:.:)
real, pointer:: Angle(:)
integer, pointer:: oldID(:)

 ……
iPATCH = GetIndexPatch(“mylWall”)
Cpatch => Patches(iPATCH)

ForceP(1:3) = 0.0          ! Начальные значения векторов сил
ForceTAU(1:3)  = 0.0

   NFC = CPatch%NoFaces      ! Число граней в патче
! --- выделяем память ----
Allocate(ForceTAU(3,NFC), ForceP(3,NFC),Algle(NFC),oldID(NFC))

!  --- на всякий случай обновляем диффузионный коэффициент для скоростей
!       в качестве пргумента можно указать индекс любой используемой компоненты
   CALL userDiffCoef(P_UGX)

!  ----Перебираем все грани патча --
   do iff =1, NFC
      oldID(iff) = iff                    ! Запоминаем старый индекс
      iFACE = Cpatch%fcLIst(iff)       ! Это индекс грани в массиве FaceBS()
      iCELL  = FaceBS(iFACE)%CellID  ! Это индекс приграничной ячейки
      P_FACE = FaceBS(iFACE)%Centre   ! Это вектор центра грани
     Area   = FaceBS(iFACE)%Area         ! Площадь грани
     V_NORM = FaceBS(iFACE)%PN    ! Вектор нормали, направлена в сторону РО
      FDist  = FaceBS(iFACE)%Dist      ! Расстояние отцентра ячейки до грани
      P_CELL = Cells(iCELL)%Centre    ! Это вектор центра ячейки
!      ----- Определяем угол атаки -----
      xx = P_FACE(1) - X_CENTRE_CYL
      yy = P_FACE(2)
      aaCOS = -XX/SQRT(XX**2+YY**2)
      aANGLE = ACOS(aaCOS)*180.0/3.1416
      ANGLE(iff) = aANGLE

!     --- P-составляющая  силы -----------
      PF_CELL0 = usF8(iCELL,P_PF)    !  Значение давления в ячейке
      pfGRAD = clGRAD(1:3,iCELL,P_PF)    ! Значение градиента в ячейке
!       --- Расчет давления в центре грани ---
      PF_FACE = PF_CELL0   + DotProduct(pfGRAD,P_FACE-P_CELL)
      ForceMOD = PF_FACE * Area            ! Модуль силы давления
      do iv=1,3          ! Компоненты силы направлены против нормали
        ForceP(iv,iff) = - ForceMOD * V_NORM(iv)
      enddo
!     --- TAU-составляющая -----------
      VEL = 0.     ! Заполняем составляющие вектора скорости в центре ячейки
      if(P_UGX.NE.0)  VEL(1) = usF8(iCELL,P_UGX)
      if(P_UGY.NE.0)  VEL(2) = usF8(iCELL,P_UGY)
      if(P_UGZ.NE.0)   VEL(3) = usF8(iCELL,P_UGZ)
!     --- Рассчитываем вектора скорости по нормали VEL_NORM и по касательной
!         VEL_TAU
      call usaTAUVelocity(VEL,V_NORM,VEL_TAU,VEL_NORM)
      VEL_TAU_MOD = VecMod(VEL_TAU)
!     ---  Значение эффективного коэффициента вязкости --
      aMuEFF = usaGamGWall(iVEL,iCELL)
!     --- касательная сила направлена в направлении VEL_TAU
      do iv=1,3
```

```
      ForceTAU(iv,iff) = aMuEFF*VEL_TAU(iv)/FDist * Area
    enddo
  enddo

!    --- Сортируем угол и индексы ----------
  call userSortQUICK_ID(ANGLE,oldID,NFC,IRC)
!    --- Выводим в текстовый файл ----------
  LU_OUT = 77
  call userOpenOutFile(LU_OUT, TRIM(PrefixRes)//'_force.dat')
  write(LU_OUT,'(a)')  ' "Angle"    "ForceP_X" "ForceTau_X"'
  do iff =1, NFC
   iOLD = V_OLDID(iff)       ! Это старый индекс для неотсортированных массивов
   write(LU_OUT,'(3(1x,1Pg12.4))')  ANGLE(iff), ForceP(1,iOLD), ForceTAU(1,iOLD)
  enddo
  close(LU_OUT)

deAllocate(ForceTAU, ForceP,Algle,oldID)
```

3.4.3 Работа с файлами

Для работы с файлами при обработке результатов можно использовать следующие сервисные подпрограммы

logical function userFileExists(nameFile)

Функция проверяет существование файла. Она возвращает .TRUE., если файл существует.

subroutine userOpenOutFile(LU,nameFile)

Подпрограмма открывает текстовый файл для записи. Если файл существует, то старый файл переписвается. Целой переменной LU нужно присвоить дескриптор файла, который в дальнейшем будет использоваться в операторах write (это число должно быть больше 50). При обработке результатов имя файла лучше всего задавать в виде

```
TRIM(PrefixRes)//"пост-часть имени"
```

В этом случае путь к файлу будет содержать стандартный путь к файлам результата, к которому будет добавлено окончание, заданное пользователем, например:

```
TRIM(PrefixRes)//'_forcex.dat'
```

Для закрытия файла нужно использовать стандартный оператор Фортрана

```
close(LU)
```

subroutine userOpenOutBinaryFile(LU,nameFile)

Эта функция аналогична предыдущей, однако она открывает файл для двоичной бесформатной записи.

В качестве второго примера использования данных функции и функций интерполяции рассмотрим пример, в котором создаются файлы, содержащие профили скорости в заданных сечениях.

```
 real, pointer:: FFU(:,:)       ! Массив для функции userPHIonLINE
 real Sect(6)
 character(2) :: chSect(6) = (/ '_1','_2', '_3', '_4', '_5', '_6' /)
……
 LU_OUT = 66
 if(IsUSCartes) then   ! признак неструктурной сетки
   allocate(FFY(2,NFY))
 else
  allocate(FFY(2,NY))
 endif
! ----- Задаем координату X сечений   -------
  Sect(1) = 0.0
```

```
  Sect(2) = 3.06
  Sect(3) = 6.12
  Sect(4) = 9.74
  Sect(5) = 11.84
  NoSect = 5
 do i=1,Nosect
! ----- Получаем распределение UGX для текущего сечения по X  -------
   call userPHIonLINE(P_UGX,2, SEct(i) ,0.0,0.0,NoV,FFY)
! ----- Открываем файл с уникальным именем   -------
   call userOpenOutFile(LU_OUT,TRIM(PrefixRes)//'_Y'//chSect(i)//'.dat')
   write(LU_OUT,'(a)') '"Y,mm" "UGX,cm/s"'
   do iy=1,NoV
     write(LU_OUT,'(1x,2(g13.4))') FFY(1,iy)*1.e3,FFY(2,iy)*1.e2
   end do
   close(LU_OUT)
 enddo
 deallocate(FFY)
```

3.5 Подпрограммы для вычисления свойств

При проведении расчетов с переменной теплоемкостью возникает необходимость расчета свойств среды независимо от сетки КО. Для этого можно использовать функцию

```
real function userFlowPropByTG(curTG,iFMAT,NameProp)
  character(*) NameProp
```

эта функция возвращает значение свойства, заданного именем NameProp:
"Dens", "Visc", "Cond", "Cp", "BetaT"
по текущему значению температуры curTG. Свойство рассчитывается для материала с индексом iFMAT. Напомним, что это индекс имени материала в символьном массиве FlowMAT(iFMAT).

Если Flow-фаза состоит из несколько компонентов и свойства зависят от концентрций и задана внутренняя модель расчета свойств в секции [Special Data]
C(“PropConcModel”) = byCONC/ byWILKE,
то для расчета свойств нужно использовать другую подпрограмму

```
subroutine userFlowProps(curTG,curCONC,DensVal,ViscValCondVal)
 real curCONC(8)
```

Третья функция позволяет рассчитать температуру G-фазы по значению ее энтальпии curHG

```
real function userFlowTGByHG(curHG,iFMAT)
```

Важное замечание: эта функция корректно работает только для свойства «HG», заданного в виде таблицы (оператор myTABLE).

4. Параллельный режим работы

Работа Решателя в последовательном и параллельном режиме принципиально отличается. Главные отличия параллельного режима:

1) каждый процесс работает со своей частью РО (ниже эта часть будет называться субдоменом),
2) для обмена между процессами используются дополнительные слои ячеек, которые называются HALO-ячейками,
3) все операции ввода/вывода и работу с «целой» РО может делать только нулевой процесс со значением глобальной переменной myID = 0.

Часть функций, описанных выше, уже настроены для работы в параллельном режиме:

1) все функции доступа,
2) подпрограммы acTotFluxPHI, acForceOnPatch, acAveragePHI, acAverageSect.

Остальные подпрограммы никак не учитывают параллельный режим работы.

Самый лучший способ обработки результатов расчета – это делать обработку только в последовательном режиме. Для проверки режима работы можно использовать глобальную переменную IsParall. Если значение переменной .TRUE., то выполняется параллельный расчет.

Однако в ряде задач (например, выполняемых на кластере) удобнее обработку делать при решении задачи. В данном разделе описываются некоторые подробности работы параллельного Решателя и описываются несколько полезных параллельных функций.

Более подробную информацию можно найти в открытых кодах Anes, расположенных в каталоге openfor.

4.1 Нюансы параллельного Решателя

При параллельном решении запускается несколько копий Решателя, каждый из которых работает с частью ячеек РО. Для различения копий в них переменной myID передается номер копии, начиная с 0. Нулевой процесс считается привилегированным, только он может осуществлять операции ввода-вывода. При старте каждый процесс получает часть ячеек РО. Ниже такая часть будет называться субдоменом РО. Принцип разбиения РО на субдомены зависит от типа сетки.

4.2 Организация параллельной структурной сетки

Каждый субдомен представляет собой параллелепипед ячеек в пространстве индексов (ix,iy,iz), при этом используется сплошная нумерация КО в субдоменах. Переменные NX,NY,NZ как и в последовательном режиме содержат общее число КО вдоль осей. Области изменения индексов в субдомене описываются следующими переменными:

```
integer (4)  IDXF,IDXL, IDYF,IDYL,IDZF,IDZL ! Все КО субдомена
integer (4)  IPXF,IPXL, IPYF,IPYL,IPZF,IPZL  ! КО для расчета свойств (без нулевых)
integer (4)  ISXF,ISXL, ISYF,ISYL,ISZF,ISZL  ! КО для Решения (без нулевых и halo)
```

На границе субдомена может располагаться либо нулевой КО на внешней границе, либо слой Halo-ячеек для обмена с соседним субдоменом. Если нужно перебрать только внутренние КО субдомена, то нужно использовать следующий цикл

```
do izC = ISZF,ISZL
  do iyC= ISYF,ISYL
    do ixC= ISXF,ISXL
        <Обработка текущего КО >
    end do
  end do
end do
```

В патчах граничные индексы бокса патча

```
Type TPATCH
……
  integer IXF,IXL,IYF,IYL,IZF,IZL
….logical IsUSED
End type
```

уже подправлены на КО своего субдомена. Если в субдомене нет ячеек патча, то флаг IsUSED устанавливается в .FALSE.

4.3 Организация параллельной неструктурной сетки

При работе с неструктурными сетками одномерный список ячеек также разбивается на субдомены, нумерация которых совпадает с исходной нумерацией. В конец ячеек субдомена добавляются HALO-ячейки для обмена с другими субдоменами. Нумерация HALO-ячеек не соответствует их истинным копиям в соседних субдоменах.

Все необходимые индексы хранятся в переменных:

```
integer (4)  ISCVF,ISCVL   ! КО СубДомена ( где проводится решение)
integer (4)  IPCVF,IPCVL   ! КО Субдомена + HALO КО
```

Для перебора только внутренних ячеек субдомена нужно использовать следующий цикл

```
do idCV = ISCVF,ISCVL
        <Обработка текущей ячейки >
end do
```

В отличие от ячеек для граней используется локальная нумерация (начинается с 1), как внутренних граней, так и для граничных граней. Внутренние грани, как и ячейки, разбиваются на две группы: внутренние ячейки, связывающие родные ячейки и грани, связывающие родную и HALO-ячейку. Для описания границ граней используются следующие переменные

```
integer (4)  IFXYZ_L, &              ! Конец внутренних XYZ-граней
        IFXYZH_F,IFXYZH_L, &  ! Начало и конец HALO XYZ-граней
integer (4)  IFBS_L                  ! Конец BS-граней
```

Как и в структурной сетке, все списки граней и ячеек в патчах корректируются так, чтобы они содержали только свои ячейки и грани. Если патч отсутствует в субдомене, то флаг IsUSED патча устанавливается в .FALSE.

4.4 Полезные параллельные подпрограммы

Если нужно рассчитать какие-то скалярные величины, а потом просуммировать их по всем субдоменам, то нужно использовать подпрограммы

```
SUBROUTINE mpSUM1(Var)
 real Var

SUBROUTINE mpSUM2(VAR1,VAR2)
 real VAR1,VAR2
```

```
SUBROUTINE mpSUM3(VAR1,VAR2,VAR3)
 real VAR1,VAR2,VAR3

SUBROUTINE mpSUM1_8(Var)
 real(8) Var

SUBROUTINE mpSUM2_8(VAR1,VAR2)
 REAL(8) VAR1,VAR2
```

Перед входом в подпрограммы их аргументы содержат локальные значения для каждого субдомена, после выхода из подпрограммы во всех субдоменах эти переменные содержат просуммированные значения.

При обработке ячеек и граней патчей можно использовать следующий алгоритм:

1) обрабатываем независимо свои грани патчей в каждом субдомене и записываем информацию в свои неотсортированные массивы,
2) собираем эти массивы в нулевом процессе,
3) сортируем массив в нулевом процессе и выводим.

Для «сборки» массивов можно использовать две следующие подпрограммы.

```
subroutine userReduceADD(A,NoLoc,NoSum)
   real,pointer:: A(:)
```

Подпрограмма собирает массив A со всех субдоменов и складывает их *друг за другом* в массив A(:) *нулевого* процесса. Размерность массива A в нулевом процессе должна быть равна сумме размеров всех массивов A в других субдоменах NoLOC. Т.е. при вызове подпрограммы каждый процесс передает в нее массив A(1:NoLoc). После выхода из подпрограммы в нулевом процессе в массиве A(1:NoSum) будут просуммированные значения. В остальных процессах переменная NoSum будет равна NoLoc.

```
subroutine userReduceSUM(A,NoA)
  real,pointer:: A(:)
```

Подпрограмма собирает массив A(:) со всех субдоменов и суммирует их элементы. На выходе из подпрограммы в массиве A(:) во всех процессах будут просуммированные значения. Размерность массивов A должны быть одинаковы во всех субдоменах и равна NoA.

Литература

1. Код Anes20xe. «Описание математических моделей кода», версия 2.24, 2019.
2. Код Anes20xe. «Описание численных алгоритмов кода», версия 2.24, 2019.
3. Код Anes20xe. «Визуализация результатов расчетов. Программы - постпроцессоры», версия 2.24, 2019.