Школа-семинар

«Расширенные возможности пакета OpenFOAM»

ОБЗОР ПРИЛОЖЕНИЯ OPENFOAM

М.В. Крапошин (НИЦ Курчатовский Институт)
О.И. Самоваров (Институт Системного Программирования РАН)
С.В. Стрижак (ГОУ ВПО МГТУ им. Баумана)

СОДЕРЖАНИЕ

- Исходный код приложения OpenFOAM (пре-процессинг, решатель, пост-процессинг)
- Как обычно строится приложение OpenFOAM
- Порядок интегрирования уравнений в OpenFOAM
- Добавление расширений к приложению (решателю)

Files/day1_SolverBasics

ПРИЛОЖЕНИЯ OPENFOAM

ПРЕПРОЦЕССИНГ	РЕШЕНИЕ	ПОСТПРОЦЕССИНГ
Инициализация начальных полей — например, поля скоростей или давлений, конвертация сеток	IcoFoam или, например interFoam	Вычисление вспомогательных полей или величин: Динамическое давление, сила, действующая на поверхность

Современная тенденция OpenFOAM — в объединении всех трех этапов в единый процесс за счет использования технологии наследования C++ и динамических библиотек — libfoamCalcFunctions, libutilityFunctionObjects. И т.д.

СТРУКТУРА ПРИЛОЖЕНИЯ OPENFOAM

Основные части — исходные файлы, файлы конфигурации сборки (список компилируемых файлов и опции компиляции)

- Пример — emptyFoamApp
- Исходные файлы: createFields.H, emptyFoamApp.C
- Файлы конфигурации: Make/files, Make/options

emptyFoamApp.C

EXE = $(FOAM_USER_APPBIN)/emptyFoamApp

EXE_INC = -I$(LIB_SRC)/finiteVolume/lnInclude

EXE_LIBS = -lfiniteVolume

НАСТРОЙКА ЗАВИСИМОСТЕЙ ПРИ КОМПИЛЯЦИИ

- Make/files — список файлов, исходный код которых должен быть оттранслирован в объектный, имя исполняемого файла
- Make/options — опции компиляции и сборки (нестандартные ключи, настройка окружения поиска заголовочных файлов и библиотек, подключаемые динамические библиотеки)

```
emptyFoamApp.C

EXE = $(FOAM_USER_APPBIN)/emptyFoamApp
```

Исходный код в emptyFoamApp.C

Исполняемый файл разместить в $FOAM_USER_APPBIN, присвоить имя emptyFoamApp

```
EXE_INC = \
 -I$(LIB_SRC)/finiteVolume/lnInclude

EXE_LIBS = -lfiniteVolume
```

Где ещё искать заголовочные файлы

Подключить библиотеку libfiniteVolume

СТРУКТУРА ИСХОДНОГО КОДА ПРИЛОЖЕНИЯ OPENFOAM

Подключение заголовочных файлов

```
#   include "fvCFD.H"
```

Инициализация исходных данных

```
#   include "setRootCase.H"
#   include "createTime.H"
#   include "createMesh.H"
#   include "createFields.H"
```

Цикл итераций по времени

```
    while (runTime.loop())
    {
        Info<< "Time = " << runTime.timeName()
        << nl << endl;
    }
```

НЕМНОГО FORTRAN В C++

В OpenFOAM используется Fortran-образный синтаксис при написании конечных приложений.

Переменные объявляются не внутри приложения, а в заголовочных файлах. Пример — createMesh.H

createMesh.H

```
    Foam::Info << "Create mesh for time = "
        << runTime.timeName() << Foam::nl << Foam::endl;

    Foam::fvMesh mesh
    (
        Foam::IOobject
        (
            Foam::fvMesh::defaultRegion,
            runTime.timeName(),
            runTime,
            Foam::IOobject::MUST_READ
        )
    );
```

ИНИЦИАЛИЗАЦИЯ ПОЛЕЙ — createFields.H

- Также, как и создание сетки, так и создание полей обычно выносится в отдельный файл — createFields.H

```
    Info<< "Reading scalar field psi\n" << endl;

    volScalarField psi
    (
        IOobject
        (
            "psi",                      //имя поля
            runTime.timeName(),         //какой срез времени?
            mesh,                       //на какой сетке?
            Ioobject::MUST_READ,        //обязательно считывать
            Ioobject::AUTO_WRITE        //автоматически записывать
        ),
        mesh                            //на какой сетке
    );
```

ЦИКЛ ИНТЕГРИРОВАНИЯ ПО ВРЕМЕНИ

Цикл интегрирования по времени осуществляется с использованием стандартных средств C++, для управления циклом используются средства класса Time

```
Info<< "\n Starting time loop\n" << endl;

  while (runTime.loop()) //автоматически переходит на новый шаг
  {
      Info<< "Time = " << runTime.timeName() << nl << endl;

      Info<< "max(psi)=" << max(psi).value() << endl
          << "min(psi)=" << min(psi).value() << endl;

      runTime.write();

      Info<< "ExecutionTime = " << runTime.elapsedCpuTime() << " s"
          << "  ClockTime = " << runTime.elapsedClockTime() << " s"
          << nl << endl;
  }
```

ПРОСТРАНСТВА ИМЕН fvc и fvm

- Внутри цикла по времени можно осуществлять операции интегрирования и алгебраических операций над полями
- Операторы fvm::ddt, fvm::d2dt2, fvm::div, fvm::laplacian, fvm::Sp, fvm::SuSp — возвращают коэффициенты матрицы, соответствующие дискретизации данного слагаемого
- Операторы fvc::ddt, fvc::d2dt2, fvc::div, fvc::grad, fvc::laplacian, fvc::Sp, fvc::grad — возвращают «правую часть», соответствующую данному выражению

```
fvm::ddt(psi) + fvm::div(phi,psi)+
fvm::SuSp(kappa,psi)
```

РЕШАТЕЛЬ myIcoFoam

$$\nabla \cdot \boldsymbol{U} = 0$$

$$\frac{\partial \boldsymbol{U}}{\partial t} + \nabla \cdot (\boldsymbol{U}\boldsymbol{U}) = \nu \nabla^2 \boldsymbol{U} - \nabla p$$

PISO включает в себя серию шагов:
1) Прогноз скорости
2) Вычисление поля давления
3) Коррекция скорости

Методом разделения операторов

$$\frac{\partial \rho \boldsymbol{U}}{\partial t} + \nabla \cdot (\rho \boldsymbol{U}\boldsymbol{U}) = \nabla \cdot \boldsymbol{R}^{Eff} - \nabla p + \boldsymbol{S}_M \rightarrow$$
$$A U - H(U) = S_p \rightarrow$$
$$U = \underbrace{\frac{H(U)}{A}}_{\text{прогноз}} + \underbrace{\frac{S_p}{A}}_{\text{коррекция}} = \frac{H(U)}{A} - \frac{1}{A}\nabla p$$

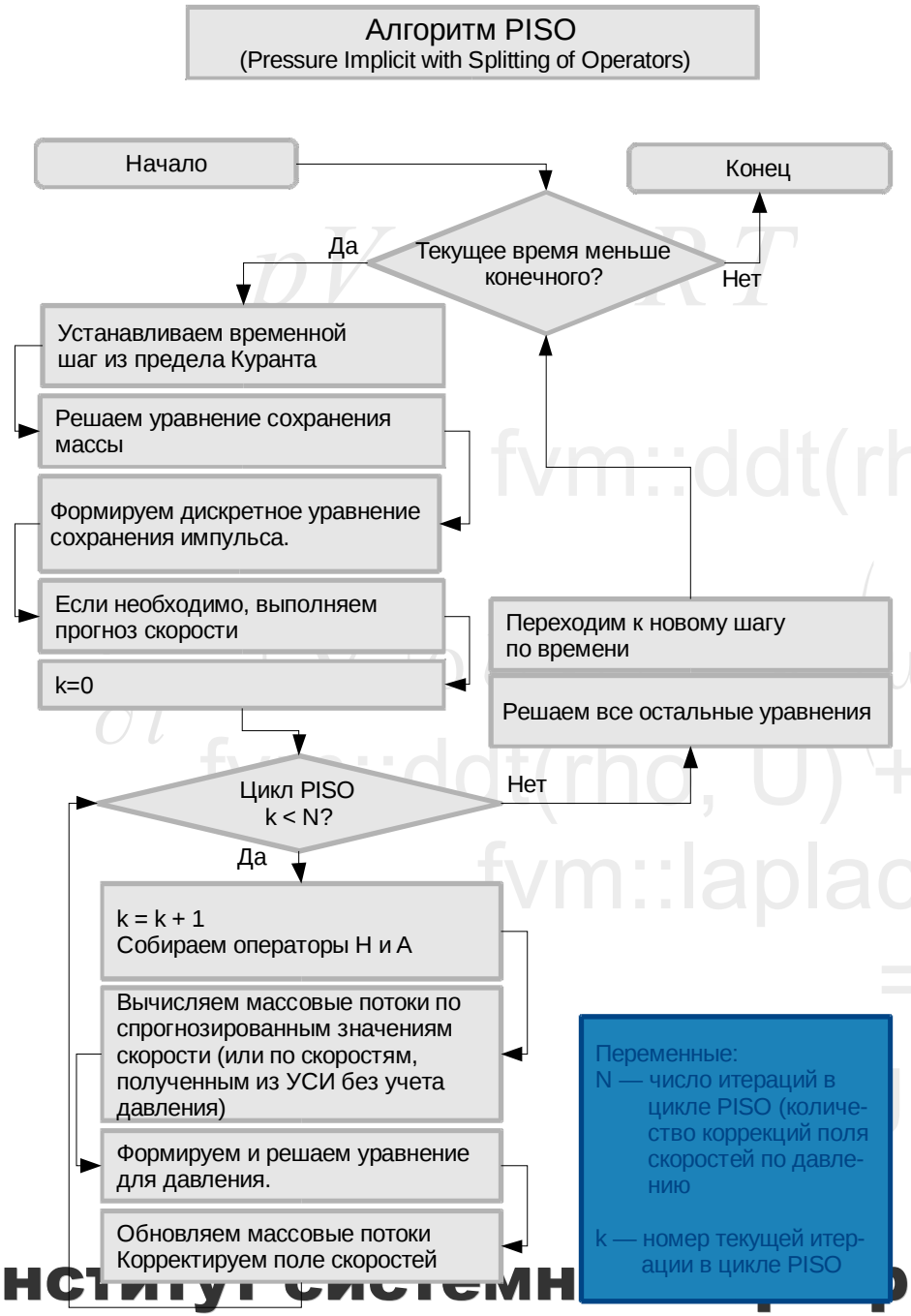
Представляем поле скоростей в виде спрогнозированного значения и коррекции

$$\frac{\partial \rho}{\partial p} = 0 \quad \frac{\partial \rho}{\partial t} = 0$$
$$\Phi = (\rho \boldsymbol{U} \cdot \boldsymbol{S}_f)_f = \rho_f (\hat{\boldsymbol{U}} + \boldsymbol{U}')_f \cdot \boldsymbol{S}_f$$
$$\nabla \cdot (\rho \boldsymbol{U}) = \nabla \cdot (\hat{\boldsymbol{U}} + \boldsymbol{U}') =$$
$$\nabla \cdot (\rho \frac{H}{A}) - \nabla \cdot (\frac{\rho}{A} \nabla p) = 0$$

Массовые потоки, которые должны удовлетворять уравнению неразрывности также раскладываются на спрогнозированную и корректирующую часть. Применение оператора дивергенции дает уравнение для давления

После решения уравнения для давления корректируем массовые потоки и скорости в ячейках в соответствии с новым значением давления. Подобная методика может использоваться как для сжимаемых, так и несжимаемых жидкостей.

АЛГОРИТМ PISO



Коррекция скорости вычисляется по процедуре, сходной с Ри-Чоу — консервативные потоки вычисляются в центрах граней. Вычисление потоков производится интерполяцией поля скорости, хранящегося в центрах ячеек

НАСТРОЙКА СБОРКИ РЕШАТЕЛЯ

Создадим необходимую структуру решателя:

```
mkdir myIcoFoam; cd myIcoFoam
```

```
mkdir Make; touch Make/files; touch Make/options
```

```
touch createFields.H; touch myIcoFoam.C
```

```
myIcoFoam.C

EXE = $(FOAM_USER_APPBIN)/myIcoFoam
```

Исходный код в myIcoFoam.C

Исполняемый файл разместить в $FOAM_USER_APPBIN, присвоить имя emptyFoamApp

```
EXE_INC = \
 -I$(LIB_SRC)/finiteVolume/lnInclude

EXE_LIBS = -lfiniteVolume
```

Где ещё искать заголовочные файлы

Подключить библиотеку libfiniteVolume

ИНИЦИАЛИЗАЦИЯ ПОЛЕЙ — createFields.H

```
    Info<< "Reading
transportProperties\n" << endl;

    IOdictionary transportProperties
    (
        IOobject
        (
            "transportProperties",
            runTime.constant(),
            mesh,
            IOobject::MUST_READ,
            IOobject::NO_WRITE
        )
    );

    dimensionedScalar nu
    (
     transportProperties.lookup("nu")
    );
```

Помимо полей используются и другие величины — например кинематическая вязкость.

В данном случае открывается текстовый файл, в котором хранится значение вязкости.

Затем значение вязкости nu считывается из поля «nu»

ИНИЦИАЛИЗАЦИЯ ПОЛЕЙ — createFields.H

Инициализация поля давления

```
    Info<<
    "Reading field p\n" << endl;
    volScalarField p
    (
        IOobject
        (
            "p",
            runTime.timeName(),
            mesh,
            IOobject::MUST_READ,
            IOobject::AUTO_WRITE
        ),
        mesh
    );
```

Инициализация поля скорости

```
    Info<<
    "Reading field U\n" << endl;
     volVectorField U
     (
         IOobject
         (
             "U",
             runTime.timeName(),
             mesh,
             IOobject::MUST_READ,
             IOobject::AUTO_WRITE
         ),
         mesh
     );
```

Инициализация поля объемных потоков через грани ячеек - phi

```
#    include "createPhi.H"
    label pRefCell = 0;
    scalar pRefValue = 0.0;
    setRefCell(p, mesh.solutionDict().subDict("PISO"), pRefCell,
pRefValue);
```

ИНИЦИАЛИЗАЦИЯ И ТЕЛО ЦИКЛА

```
#include "fvCFD.H" //необходимые определения OpenFOAM

int main(int argc, char *argv[])
{
#   include "setRootCase.H" //инициализация структуры каталогов

#   include "createTime.H" //создание времени (регистр объектов(
#   include "createMesh.H" //создание сетки
#   include "createFields.H" //инициал-ия полей
#   include "initContinuityErrs.H" //инициал-ия небаланса масс

    Info<< "\nStarting time loop\n" << endl;

    while (runTime.loop()) //переход к новому шагу если T<Tk
    {
        Info<< "Time = " << runTime.timeName() << nl << endl;

#       include "readPISOControls.H" //считать параметры PISO
#       include "CourantNo.H"        //вычислить число Co
```

ПРОГНОЗ ПОЛЯ СКОРОСТИ

$$\frac{\partial \boldsymbol{U}}{\partial t} + \nabla \cdot \left(\boldsymbol{U}\boldsymbol{U} \right) - \nu \nabla^2 \boldsymbol{U} = -\nabla p$$

```
fvVectorMatrix UEqn
(
    fvm::ddt(U)
  + fvm::div(phi, U)
  - fvm::laplacian(nu, U)
);

solve(UEqn == -fvc::grad(p));
```

Согласно процедуре PISO, формируются три матрицы (для Ux, Uy, Uz) уравнения сохранения импульса. Системы решаются последовательно с использованием давления с предыдущего шага (fvc::grad)

ТЕЛО ЦИКЛА PISO

```
// --- PISO loop

for (int corr=0; corr<nCorr; corr++)
{
    volScalarField rUA = 1.0/UEqn.A();

    U = rUA*UEqn.H();
    phi = (fvc::interpolate(U) & mesh.Sf())
        + fvc::ddtPhiCorr(rUA, U, phi);

    adjustPhi(phi, U, p);

    {
      // Уравнение для давления — см. след. слайд
    }

#   include "continuityErrs.H"

    U -= rUA*fvc::grad(p);
    U.correctBoundaryConditions();
}
```

УРАВНЕНИЕ ДЛЯ ДАВЛЕНИЯ

Сетка может быть неортогональной, каждая итерация цикла — поправка на неортогональность. По окончательному давлению делается поправка объемных потоков через грани ячеек

```
for (int nonOrth=0; nonOrth<=nNonOrthCorr; nonOrth++)
{
    fvScalarMatrix pEqn
    (
        fvm::laplacian(rUA, p) == fvc::div(phi)
    );

    pEqn.setReference(pRefCell, pRefValue);
    pEqn.solve();

    if (nonOrth == nNonOrthCorr)
    {
        phi -= pEqn.flux();
    }
}
```

$$U = \tilde{U} - \frac{1}{A_p}\nabla p$$

ДОБАВЛЕНИЕ УРАВНЕНИЯ ТРАНСПОРТА

При добавлении новой искомой величины нужно: определить её инициализацию и уравнение, описывающее её динамику

```
    volScalarField psi
    (
        IOobject
        (
            "psi",
            runTime.timeName(),
            mesh,
            IOobject::MUST_READ,
            IOobject::AUTO_WRITE
        ),
        mesh
    );
```

```
    fvScalarMatrix psiEqn
    (
        fvm::ddt(psi)
      + fvm::div(phi, psi)
      + fvm::laplacian(Dpsi,psi)
    );
```

УЧЕТ ТУРБУЛЕНТНОСТИ (1)

Для учета турбулентности в OpenFOAM введен специальный класс, предназначенный для дискретизации дивергенции тензора напряжений. Вместе с ним вводится класс, определяющий зависимость тензора напряжений от деформаций:

- Необходимо использовать класс incompressible::turbulenceModel
- Необходимо изменить файл createFields.H — инициализация новых полей
- Диффузионное слагаемое теперь должно содержать вклад от турбулентных пульсаций

```
EXE_LIBS = \
    -lincompressibleTurbulenceModel \
    -lincompressibleRASModels \
    -lincompressibleLESModels \
    -lincompressibleTransportModels \
    -lfiniteVolume \
    -lmeshTools
```

```
EXE_INC = \
    -I$(LIB_SRC)/turbulenceModels/incompre
    -I$(LIB_SRC)/transportModels \
    -I$
(LIB_SRC)/transportModels/incompressible/singlePhaseTransportModel \
    -I$(LIB_SRC)/finiteVolume/lnInclude
```

УЧЕТ ТУРБУЛЕНТНОСТИ (2)

Необходимо подключить новые заголовочные файлы

```
#include "singlePhaseTransportModel.H"
#include "turbulenceModel.H"
```

Инициализировать соответствующие объекта класса

```
singlePhaseTransportModel laminarTransport(U, phi);

autoPtr<incompressible::turbulenceModel> turbulence
(
    incompressible::turbulenceModel::New(U, phi, laminarTransport)
);
```

```
volScalarField DpsiEff = Dpsi +
turbulence().nut();
DpsiEff.rename("DpsiEff");
fvScalarMatrix psiEqn
(
    fvm::ddt(psi)
  + fvm::div(phi,psi)
  - fvm::laplacian(DpsiEff,psi)
);
```

```
fvVectorMatrix UEqn
(
    fvm::ddt(U)
  + fvm::div(phi, U)
  - turbulence().divDevReff(U)
);
```

Переписать уравнения

СПАСБО ЗА ВНИМАНИЕ!

$$pV = \nu RT$$

$$\frac{\partial \rho}{\partial t} + \nabla \cdot \rho \boldsymbol{U} = 0$$

fvm::ddt(rho) + fvc::div(phi)=0

$$\frac{\partial \rho \boldsymbol{U}}{\partial t} + \nabla \cdot (\rho \boldsymbol{U}\boldsymbol{U}) - \nabla \cdot \left(\mu \frac{1}{2} \left(\nabla \boldsymbol{U} + (\nabla \boldsymbol{U})^T \right) \right) = -\nabla p$$

fvm::ddt(rho, U) + fvm::div(phi,U) -
fvm::laplacian(mu,U)
=
-fvc::grad(p)