Школа-семинар

«Расширенные возможности пакета OpenFOAM»

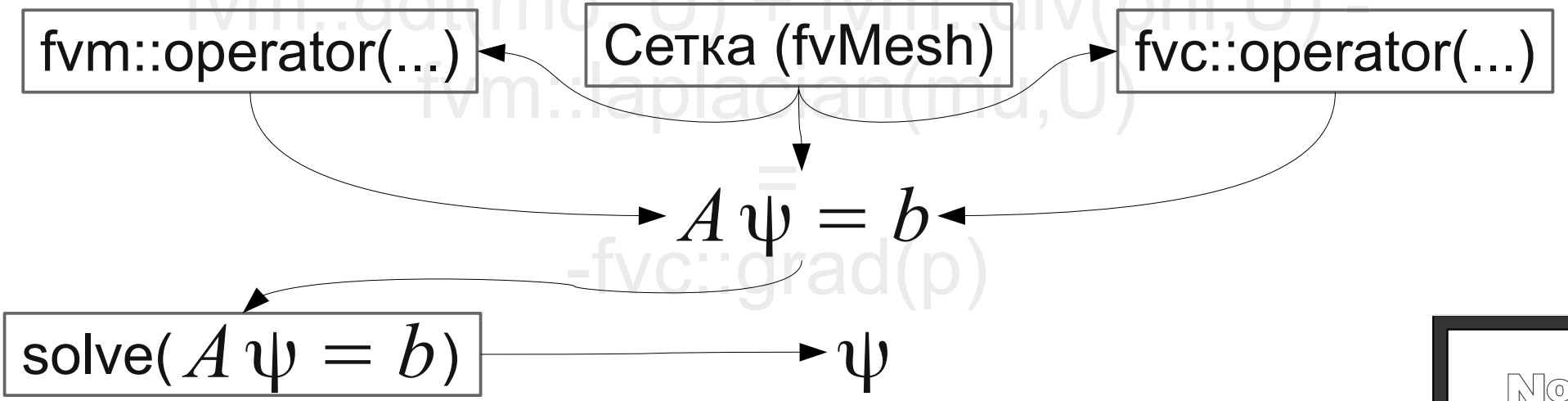
ОБЗОР РЕШАТЕЛЯ

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

ОБЩИЙ ПОРЯДОК ИНТЕГРИРОВАНИЯ УРАВНЕНИЙ

Решение задачи проводится на три этапа. Результат выполнения каждого из трех — **fvMatrix**. В случае неявной дискретизации (**fvm::**) - матрица коэффициентов в левой части, в случае явной (**fvc::**) — вектор правой части (поле, определенное в расчетных точках)

1) Неявная дискретизация слагаемых уравнений (**fvm::**)
2) Явная дискретизация слагаемых уравнений (**fvc::**)
3) Дискретизация по времени (**fvm::ddt,fvm::d2dt2** и **fvc::dtdt, fvc::d2tdt2**)
4) Обновление граничных условий (возможно на шагах 1 и 2)
5) Решение системы уравнений



ПРИЛОЖЕНИЕ OpenFOAM. ОСНОВНЫЕ ЭТАПЫ

1)Инициализация структуры каталогов рабочей задачи

2)Инициализация потоков выполнения (MPI)

3)Инициализация физического времени

4)Инициализация расчетной сетки

5)Инициализация искомых полей

6)Инициализация цикла интегрирования (время)

7)Интегрирование уравнений и запись результатов

8)Завершение работы

ПРИЛОЖЕНИЕ OpenFOAM. ОСНОВНЫЕ ЭТАПЫ. ПРИМЕР

laplacianFoam — решение уравнения Лапласа, например, тепловой диффузии в твердом теле

```
#include "fvCFD.H" //Основной заголовочный файл, содержащий
                   //практически все необходимы объявления

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

int main(int argc, char *argv[]) //Точка входа в программу
{

#   include "setRootCase.H" //Инициализация каталога задачи и
                            //потоков выполнения
#   include "createTime.H"  //Инициализация физического времени
#   include "createMesh.H"  //Инициализация расчетной сетки
#   include "createFields.H"//Инициализация полей
```

ПРИЛОЖЕНИЕ OpenFOAM. ЭТАПЫ 1 и 2.

setRootCase.H

```
    Foam::argList args(argc, argv);
    if (!args.checkRootCase())
    {
        Foam::FatalError.exit();
    }
```

Инициализация структуры каталога задачи производится в конструкторе класса argList, там же осуществляется и инициализация потоков выполнения MPI

```
    for (int argI = 0; argI < argc; argI++)
    {
        if (argv[argI][0] == '-')
        {
            const char *optionName = &argv[argI][1];

            if (validParOptions.found(optionName))
            {
                parRunControl_.runPar(argc, argv);
                Break;
}}}
```

ПРИЛОЖЕНИЕ OpenFOAM. ЭТАПЫ 3 и 4

Создание времени — файл createTime.H содержит обращение к конструктору класса Time

```
    Foam::Info<< "Create time\n" <<
Foam::endl;

    Foam::Time runTime
    (
        Foam::Time::controlDictName,
        args.rootPath(),
        args.caseName()
    );
```

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

Создание расчетной сетки — файл createMesh.H содержит обращение к конструктору класса fvMesh

ПРИЛОЖЕНИЕ OpenFOAM. ЭТАП 5

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

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

createMesh.H (представлен не полностью) — инициализация искомых полей, физических констант

ПРИЛОЖЕНИЕ OpenFOAM. ЭТАПЫ 6 и 7

В основном цикле (**while**) содержатся условие выхода из него (**runTime.loop()**) и процедура интегрирования уравнений (**solve(...)**). Запись результатов производится в подключаемом файле **"write.H"**

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

#       include "readSIMPLEControls.H"

        for (int nonOrth=0; nonOrth<=nNonOrthCorr; nonOrth++)
        {
            solve
            (
                fvm::ddt(T) - fvm::laplacian(DT, T)
            );
        }

#       include "write.H" //runTime.write()
```

ПРИЛОЖЕНИЕ OpenFOAM. ЭТАП 8

В конце каждого шага по времени выводится информация об общем затраченном времени на выполнение и использованном процессорном времени

```
#       include "write.H"

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

    Info<< "End\n" << endl;

    return 0;
}
```

По окончании расчета, вызов оператора **return 0** приводит к уничтожению всех переменных, в т.ч. и **args**, класса **argList**. В деструкторе последнего содержатся инструкции по корректному завершению программы

ПРОЦЕСС РЕШЕНИЯ ДУ В ЧП

$$\frac{\partial T}{\partial t} - \nabla \cdot \left(D_T \nabla T \right) = 0$$

```
solve
(
    fvm::ddt(T) - fvm::laplacian(DT, T)
);
```

1) Формируется матрица СЛАУ: операторы пространства имен **fvm::**

2) Формируется правая часть (поле): операторы пространства имен **fvc::**

3) Результаты суммируются для получения общей системы *Ax=b*

4) Вызывается процедура **solve (….)**, в которой:

   a) В которой выполняется предобуславливание полученной матрицы

   b) Производится поиск решения заданным пользователем методом

   c) Полученное решение возвращается в виде поля и присваивается искомой переменной (**T** в данном случае)

СПАСБО ЗА ВНИМАНИЕ!

$$pV = \nu R T$$

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

```
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)
```