Министерство образования и науки Российской федерации
Санкт-Петербургский политехнический университет Петра Великого

А.А. Смирновский, Е.М. Смирнов

Высокопроизводительные вычисления с использованием пакета OpenFOAM

Санкт-Петербург
2019

Высокопроизводительные вычисления с использованием пакета OpenFOAM.

Предназначено для студентов, обучающихся по направлению «Прикладные математика и физика».

Оглавление

Введение

Здесь может быть будет введение

1. Решение задач механики сплошных сред по методу конечных объёмов

Метод конечных объёмов (МКО; Finite Volume Method, FVM) в настоящее время является одним из наиболее используемых численных методов, причём не только для задач механики сплошных сред, но и для других областей науки, оперирующих с понятиями «векторное поле», «тензорное поле» и т.п. В разделе 1.1 даются основные определения и общие математические формулировки из векторной и тензорной алгебры, на основе которых в разделе 1.2 подробно описывается сам метод конечных объёмов. Некоторые особенности использования МКО применительно к задачам гидроаэродинамики и теплообмена рассмотрены более подробно в разделе 1.3.

1.1. Предварительные замечания

1.1.1. Скалярные, векторные и тензорные поля. Уравнения физики для полевых величин

В механике сплошных сред (далее МСС), как и во многих других областях физики, используется аппарат векторного и тензорного исчислений. Под полем какой-либо физической величины (например, скорости, плотности, температуры) понимается совокупность её значений в пространстве[1]. Физическая величина может представлять собой скаляр (температура, давление), вектор (скорость, сила) или тензор (тензор напряжения, деформации). Здесь и далее под «пространством» всегда понимается физическое трёхмерное конфигурационное пространство. Таким образом, поле какой-либо скалярной величины φ представляется как функция пространственного радиус-вектора, а также времени t:

$$\varphi = \varphi(\vec{r}; t) = \varphi(x, y, z; t),$$

[1] Здесь и далее в этом разделе обозначения и формулировки основных понятий даны в соответствии с [1]

где радиус-вектор $\vec{r}$ имеет координаты (x, y, z). Аналогично вводится поле векторной величины $\vec{a}$:

$$\vec{a} = \vec{a}(\vec{r}; t) = \vec{a}(x, y, z; t),$$

которая также может быть записана покомпонентно в виде:

$$a_x = a_x(x, y, z; t), \quad a_y = a_y(x, y, z; t), \quad a_z = a_z(x, y, z; t).$$

Для сокращения записи иногда будет использоваться числовая индексация координат, т.е.

$$x = x_1, \quad y = x_2 \quad , z = x_3; \quad a_x = a_1, \quad a_y = a_2, \quad a_z = a_3.$$

Помимо скаляров и векторов, в МСС также часто встречается понятие тензора второго ранга (или для краткости просто «тензор»). В соответствии с определением из линейной алгебры, линейным преобразование вектора $\vec{a}$ в вектор $\vec{b}$ называется преобразование вида:

$$b_k = \sum_{l=1}^{3} T_{kl} a_l, \quad k = 1, 2, 3,$$

где T_{kl} — компоненты тензора T. Чтобы тензор T был физически объективной величиной, т.е. не зависел бы от системы координат, необходимо, чтобы такое преобразование было инвариантно относительно системы координат (т.е. компоненты тензора должны преобразовываться при переходе от одной системы координат к другой, что справедливо и для физически объективных векторов). Таким образом, в координатном представлении тензор характеризуется девятью своими компонентами, которые можно записать в виде матрицы:

$$T = \begin{pmatrix} T_{xx} & T_{xy} & T_{xz} \\ T_{yx} & T_{yy} & T_{yz} \\ T_{zx} & T_{zy} & T_{zz} \end{pmatrix} = \begin{pmatrix} T_{11} & T_{12} & T_{13} \\ T_{21} & T_{22} & T_{23} \\ T_{31} & T_{32} & T_{33} \end{pmatrix}.$$

Скалярное произведение двух векторов будем обозначать $\vec{a} \cdot \vec{b}$ (результатом этой операции является скаляр), векторное произведение — $\vec{a} \times \vec{b}$ (вектор), диадное произведение — $\vec{a}\vec{b}$. Результатом диадного произведения является тензор D с компонентами:

$$D_{kl} = a_k b_l.$$

Скалярное произведение вектора $\vec{a}$ на тензор T обозначается как $\vec{a} \cdot T$ и даёт в результате вектор. Сопряжённый к T тензор обозначается как T^T; его компоненты равны $T^T_{ij} = T_{ji}$, т.е. матрица сопряжённого тензора является транспонированной к матрице тензора T; кроме того, справедливо равенство

$$\vec{a} \cdot T = T^T \cdot \vec{a}.$$

Любой тензор T может быть разложен на сумму симметричного тензора S (для которого справедливо соотношение $S_{ij} = S_{ji}$, т.е. $S^T = S$) и антисимметричного тензора A (который удовлетворяет равенству $A_{ij} = -A_{ji}$):

$$T = S + A, \quad S = \frac{1}{2}(T + T^T), \quad A = \frac{1}{2}(T - T^T).$$

Инвариантами тензора являются: след тензора $tr\, T$, квадрат тензора T^2 и определитель его матрицы $det\, T$:

$$tr\, T = \sum_{i=1}^{3} T_{ii}, \quad T^2 = \sum_{i,j=1}^{3} T_{ij}^2, \quad det\, T = \begin{vmatrix} T_{11} & T_{12} & T_{13} \\ T_{21} & T_{22} & T_{23} \\ T_{31} & T_{32} & T_{33} \end{vmatrix}.$$

Если скаляр, вектор или тензор являются полевыми величинами, т.е. зависят от радиус-вектора и, возможно, времени, то для них справедливы понятия пространственной производной. В частности, для любой полевой величины Ψ (которая является либо скаляром, либо вектором, либо тензором) можно ввести определения производной по направлению как:

$$\frac{d\Psi}{dl} = \lim_{MM' \to 0} \frac{\Psi(M') - \Psi(M)}{MM'} = \lim_{\Delta l \to 0} \frac{\Delta \Psi}{\Delta l},$$

где M и M' — две близкие точки в пространстве, лежащие на линии l.

Если ввести стандартное понятие вектора-оператора «набла» ∇ через орты осей координат $\vec{i}_k$ как[1]

$$\nabla = \sum_{k=1}^{3} \vec{i}_k \frac{\partial}{\partial x_k},$$

[1] Такая форма записи справедлива только для декартовой системы координат

то производная по направлению вдоль оси $\vec{l}$ записывается в виде:

$$\frac{d\Psi}{dl} = \left(\vec{l} \cdot \nabla\right) \Psi.$$

Для скалярной функции φ произведение оператора «набла» на неё даёт вектор, называемый градиентом этой величины и обозначаемый как

$$\operatorname{grad} \varphi = \nabla\varphi.$$

Градиент скалярной величины $\operatorname{grad} \varphi$ представляет собой вектор, равный по величине производной по нормали к поверхности уровня этой величины и направленный по нормали к этой поверхности в сторону наибольшего изменения φ (поверхностью уровня называют поверхность, определяемую уравнением $\varphi(x_1, x_2, x_3) = C = const$).

Для вектора $\vec{a}$ и тензора T можно ввести следующие дифференциальные операторы (в скобках указан тип результата операции):

дивергенция вектора (скаляр):	$\operatorname{div} \vec{a} = \nabla \cdot \vec{a}$
ротор вектора (вектор):	$\operatorname{rot} \vec{a} = \nabla \times \vec{a}$
градиент вектора (тензор):	$\operatorname{Grad} \vec{a} = \nabla \vec{a}$
дивергенция тензора (вектор):	$\operatorname{Div} T = \nabla \cdot T$
ротор тензора (тензор):	$\operatorname{Rot} T = \nabla \times T$

Более подробно о дифференциальных операциях и их вычислении в разных системах координат можно найти в [1]. Важно подчеркнуть, что результат какой-либо дифференциальной операции не зависит от выбора системы координат, поэтому такие операции могут отражать физическую объективность, а следовательно, могут входить в математическую формулировку физических законов.

Таким образом, зачастую если какой-либо физический закон записывается для полевых величин в конфигурационном пространстве, то в уравнениях для этой величины будут входить только дифференциальные пространственные операторы (такие как градиент, ротор, дивергенция и др.) и, возможно, производные по времени. В качестве примера можно привести

закон индукции Фарадея (изменение магнитной индукции $\vec{B}$ во времени порождает электрическое поле $\vec{E}$):

$$\frac{\partial \vec{B}}{\partial t} + \nabla \times \vec{E} = 0 \tag{1.1}$$

или уравнение неразрывности (например, имеющее место в гидродинамике для массовой плотности или в электродинамике для объёмной плотности заряда):

$$\frac{\partial \rho}{\partial t} + \nabla \cdot \vec{j} = 0 \tag{1.2}$$

1.1.2. Формулировка дифференциальных операторных уравнений в интегральной форме, обобщённая теорема Гаусса-Остроградского

Покажем, как уравнения для полевых величин могут быть записаны в интегральной форме для произвольного пространственного объёма. Рассмотрим, например, уравнения переноса примеси:

$$\frac{\partial c}{\partial t} + \nabla \cdot (c\vec{V} + \vec{q}) = 0, \tag{1.3}$$

где c — концентрация примеси, $\vec{V}$ — скорость потока, $\vec{q}$ — вектор диффузионного потока, рассчитываемый, например, при помощи закона Фика:

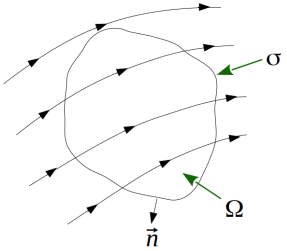

Рис. 1.1. Иллюстрация течения (векторные линии скорости) и задание произвольного объёма Ω

$$\vec{q} = -D\nabla c. \tag{1.4}$$

Рассмотрим произвольный неподвижный объём Ω, ограниченный замкнутой поверхностью σ с внешней нормалью $\vec{n}$ (см. рисунок 1.1). Проинтегрируем уравнение переноса по этому объёму, тогда получим следующее равенство:

$$\frac{\partial}{\partial t}\int\limits_{\Omega} c d\Omega + \int\limits_{\Omega} \nabla \cdot (c\vec{V} + \vec{q}) d\Omega = 0.$$

Далее, для второго слагаемого, воспользовавшись известной теоремой Гаусса-Остроградского, перейдём от интегрирования по объёму к интегри-

рованию по поверхности, тогда получим искомое уравнение переноса в интегральной форме:

$$\frac{\partial}{\partial t}\int\limits_{\Omega} c d\Omega + \int\limits_{\sigma} \vec{n}\cdot(c\vec{V}+\vec{q})d\sigma = 0. \tag{1.5}$$

В общем случае оказывается, что для дифференциальных операторов справедлива так называемая обобщённая теорема Гаусса-Остроградского (включающая также теорему Стокса), которую в условной форме можно записать следующим образом:

$$\int\limits_{\Omega} \nabla(...)d\Omega = \int\limits_{\sigma} \vec{n}(...)d\sigma,$$

где под (...) понимается любое математически корректное выражение, например:

$$(...) = \begin{cases} c \\ \cdot\vec{V} \\ \times\vec{V} \\ \cdot T \\ \times T \end{cases},$$

где T — тензор.

Отсюда, в частности, следует интегральное определение дифференциальных операций над полями:

$$\nabla(...) = \lim_{\Omega\to 0}\frac{1}{\Omega}\int\limits_{\sigma}\vec{n}(...)d\sigma. \tag{1.6}$$

Такое определение отражает суть оператора «набла» — это в некотором предельном смысле отношение «потока» через поверхность, ограничивающую некоторый объём, к этому объёму; под «потоком» здесь понимается выражение $\int_\sigma \vec{n}(...)d\sigma$. В частности, для вектора скорости $\vec{V}$ величина $\int_\sigma \vec{n}\cdot\vec{V}d\sigma$ является не чем иным, как объёмным расходом через поверхность σ (а $\vec{n}\cdot\vec{V}d\sigma$ — объёмным расходом через элемент площади $d\sigma$, соответственно). В уравнении неразрывности (1.2) величина $\vec{j}$ является плотностью потока; в случае уравнения неразрывности для массовой плотности ρ эта

величина равна $\vec{j} = \rho\vec{V}$ и представляет собой плотность массового потока, а $\int_\sigma \vec{n} \cdot \rho\vec{V} d\sigma$ является массовым расходом через поверхность σ.

Отметим, что в настоящем учебном пособии структура изложения построена исходя из дифференциальных формулировок, и далее осуществлён переход к интегральной формулировке, хотя в науке зачастую при математической записи каких-либо физических законов обычно следуют в обратном направлении — от интегральной формулировки к дифференциальной. Сделано это в связи с направленностью данного пособия на описание работы в пакете OpenFOAM, в котором основными «примитивами» являются как раз скаляры, векторы, тензоры и дифференциальные пространственные операторы (которые численно аппроксимируются при помощи своих интегральных формулировок в соответствии с (1.6)). Таким образом, если какой-либо физический закон может быть выражен через эти «примитивы», то есть все основания полагать, что для численного моделирования на основе этого закона можно эффективно использовать пакет OpenFOAM (путём расширения его функциональности, о чём подробнее будет сказано в разделе 2.1 и главе 6).

1.1.3. Введение в численное моделирование

Численное моделирование какой-либо физической проблемы включает в себя несколько этапов. Очевидным и необходимым предварительным этапом является формулировка физической модели, т.е. собственно описание проблемы и тех физических явлений, которые надо учитывать для корректного её решения. Несмотря на то, что формально этот этап не связан непосредственно с численным моделированием, тем не менее, выбор той или иной физической модели может в свою очередь влиять на выбор численного метода для решения задачи и наоборот — имеющиеся средства для численного моделирования могут влиять на то, какие аспекты физической модели будут включены в рассмотрение. В качестве несколько искусственного примера можно привести задачу моделирования движения корабля в прибрежных водах с учётом прилива или отлива. Если рассматривать корабль как материальную точку, то для него пишутся обычные законы сохранения механики, и, соответственно, численный метод будет заключаться в дискретизации обыкновенных дифференциальных уравне-

ний. Естественно учитывать прилив (или отлив) путём задания его параметров как известных функций времени, однако, можно пойти дальше и попытаться рассчитать задачу гравитационного взаимодействия Луны и воды, что явно потребует более сложного математического аппарата и, следовательно, более «изощрённого» численного моделирования.

Очевидно, что неразрывно с этапом формулировки физической модели идёт следующий этап — формулировка математической модели. По сути, численное моделирование означает численное решение математических уравнений. В принципе, эти уравнения могут и не описывать никакое физическое явление. Поэтому с точки зрения численного моделирования первым этапом является как раз формулировка математической модели, т.е. написание уравнений, граничных условий к ним и замыкающих соотношений, а также выбор значений входящих в уравнения коэффициентов (в том числе этим численное решение отличается от аналитического — невозможностью получить решение при любых значениях коэффициентов).

Второй этап численного моделирования заключается в выборе численного метода для решения математических уравнений. Этот этап обычно определяется используемым программным средством для численного моделирования. В частности, при решении дифференциальных уравнений в частных производных можно использовать следующие методы: метод конечных разностей (МКР), метод конечных элементов (МКЭ), метод конечных объёмов (МКО), метод граничных элементов, бессеточные (спектральные) методы и т.д. В настоящем пособии мы рассматриваемом только метод конечных объёмов, однако, некоторые понятия, которые будут использоваться нами, неразрывно связаны с методом конечных разностей. За деталями об этом методе можно обратится к [2]. Метод конечных элементов наиболее хорошо зарекомендовал себя в применении к задачам механики твёрдого деформируемого тела, подробнее о нём см. [3]. Среди перечисленных методов именно первые три — МКР, МКЭ и МКО — являются в настоящее время наиболее универсальными и распространёнными. Метод граничных элементов (см., например, [4]) заключается в преобразовании дифференциального уравнения в объёме к интегральному уравнению на границе и не обладает универсальностью, присущей, в частности, МКО. Спектральные методы представляют собой разложения решения в ряд по

базисным функциям и также не являются универсальными. По сравнению с МКО и МКЭ, метод конечных разностей более простой в реализации, однако, плохо применим для областей сложной геометрии. Преимущество МКО по отношению к МКЭ заключается в возможности создания полностью консервативных схем, о чём более подробно будет говориться далее. Недостатком МКО является чрезмерная сложность построения схемы с порядком дискретизации по пространству выше второго. Далее эти аспекты МКО будут рассматриваться более детально.

Следующий — третий — этап в случае выбора сеточных методов (в частности, МКР, МКЭ, МКО) заключается в построении расчётной сетки, т.е. дискретизации пространства. В настоящее время этот этап в основном выполняется «вручную», поскольку правильно построенная сетка помогает существенно сократить время самого расчёта и увеличить его точность. Подробнее о сетках будут рассказано в разделе 1.2.1.

Далее требуется задать начальные и граничные условия, доопределить параметры решаемой задачи и т.п., одним словом — подготовить расчёт для запуска.

Следующий этап является основным и может для сложных задач занимать довольно значительный (вплоть до нескольких месяцев) промежуток времени — собственно, численный расчёт (или иными словами, запуск программы-решателя[1]). Этот этап выполняется уже только компьютером, т.е. в основном без участия пользователя.

Заключительный этап — обработка и анализ результатов расчёта. Понятно, что после этого этапа возможно появится необходимость подкорректировать какие-либо предыдущие этапы, а затем повторить расчёта, так что этот «заключительный» этап может оказать лишь промежуточным в цепочке итераций по решению поставленной задачи.

Перечисленные выше этапы можно свести в три основных блока, как представлено на рисунке 1.2. Первый блок, условно называемый «постановкой задачи», включает в себя её формулировку, выбор численного метода, построение сетки, задания различных необходимых параметров. По-английски этот блок носит название pre-processing (предвари-

[1]Калька с английского solver — компьютерная программа для решения математической задачи тем или иным численным методом

тельная обработка), а компьютерная программа (одна или несколько) для выполнения необходимых подготовительных действий — pre-processor. Второй блок — собственно проведение расчёта, solving, выполняется программой-решателем[1] (solver). И третий блок — обработка (в т.ч. визуализация) результатов, post-processing. Программа, в которых выполняется эта обработка, называются post-processor.

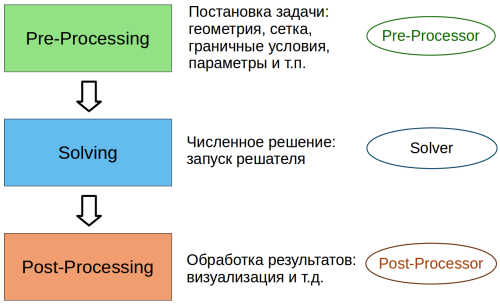

Рис. 1.2. Основные этапы численного моделирования

Таким образом, в дальнейшем мы будем ориентироваться на представленную выше схему (рисунок 1.2). Отметим сразу, что в основном программы из пакета OpenFOAM представляют собой как раз решатели (о чём подробно будет рассказано в следующей главе); что касается программ для подготовки расчёта и обработки результатов, то здесь возможности пакета OpenFOAM в некоторой степени ограничены.

1.2. Формулировка метода конечных объёмов (МКО)

Как следует из названия, метод конечных объёмов оперирует с некоторыми (замкнутыми) объёмами пространства. Для этих объёмов записываются уравнения физики в интегральной векторно-операторной форме. Возможность свести любое операторное уравнение для полевых величин к интегральной форме обусловлена выполнением обобщённой теоремы Гаусса-Остроградского, описанной в разделе 1.1.2. Если же уравнение записано в форме, исключающей переход к интегральной, то для него метод конечных объёмов неприменим[2].

[1] В дальнейшем просто «решатель»

[2] Этим, в частности, МКО отличается от метода конечных разностей: в последнем дискретизируются производные, а в МКО — дифференциальные операторы «как целое»

Таким образом, для применения МКО необходимо некоторым образом разбить расчётную область на объёмы, т.е. провести дискретизацию расчётной области или, другими словами, построить расчётную сетку. Здесь мы сразу разделяем два понятия: расчётная область и расчётная сетка. Расчётная область представляет собой область, в которой необходимо найти решение заданных уравнений для полевых величин; эта область должна быть ограничена некоторыми поверхностями, на которых необходимо задать граничные условия. Важно отметить, что с т.з. математики решение уравнения может существовать и в частично или полностью неограниченной области, но в численном расчёте по МКО невозможно построить решение в неограниченном пространстве[1]. Поэтому при численном моделировании по МКО всегда имеют дело с ограниченными в пространстве расчётными областями. Для некоторых задач такое условие зачастую может требовать проведения дополнительных исследований на предмет степени влияния ограниченности расчётной области на получаемое решение.

1.2.1. Дискретизация расчётной области

Дискретизация расчётной области для МКО[2] представляет собой разбиение её на набор связанных друг с другом объёмов (ячеек). В качестве иллюстрации на рисунке 1.3 представлена расчётная область (двумерная задача о течении потока в трубе с присоединённой небольшой дополнительной трубкой) и результат разбиения её на объёмы (расчётная сетка).

По способу разбиения расчётной области на объёмы (т.е. ячейки расчётной сетки) можно выделить следующие типы расчётных сеток:

- структурированные — область разбивается на ячейки таким образом, что каждая ячейка может быть однозначно идентифицирована двумя (для двумерных задач) или тремя (для трёхмерных) индексами, т.е. сетка может быть отображена на двумерную или трехмерную матрицу так, что индексы соседних ячеек отличаются на единицу по одному из индексов; элементом структурированной сетки является че-

[1] Это касается и некоторых других методов, в частности, МКР, МКЭ, МГЭ. Иногда проблему неограниченности можно обойти путём подходящей замены переменных

[2] И для других сеточных методов

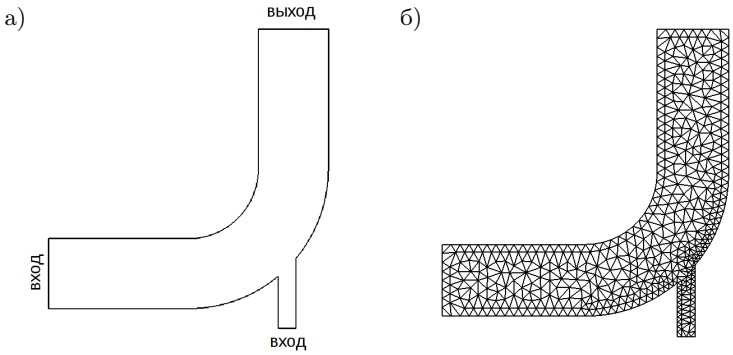

Рис. 1.3. Расчётная область (а) и расчётная сетка (б) для двумерной задачи о течении потока

тырёхугольник (двумерная задача) или шестигранник (трёхмерная); пример структурированной сетки приведён на рисунке 1.4а;

- блочно-структурированные — область может быть разбита на несколько связанных подобластей, в каждой из которых строится структурированная сетка (см. рисунок 1.4б);

- неструктурированные — ячейки произвольного типа могут быть расположены неупорядоченным образом; в качестве примера на рисунке 1.4в приведена расчётная сетка, состоящая из трегольных и четырёхугольных ячеек.

Таким образом, одним из признаков структурированной сетки является то, что все элементы сетки должны быть четырёхугольниками или шестигранниками (в зависимости от размерности задачи). Однако, определяющим признаком является возможность ввести индексацию ячеек (как указано выше), например, путём задания целочисленных индексов i и j как на рисунке. «Соседями» ячейки с координатами (i, j) будут ячейки с координатами $(i+1, j)$, $(i-1, j)$, $(i, j+1)$, $(i, j-1)$ (для двумерной сетки).

Следует разделить понятия типа сетки и формата её хранения. Понятно, что структурированные и блочно-структурированные сетки можно хранить в виде набора таблиц (двумерных или трёхмерных массивов); для

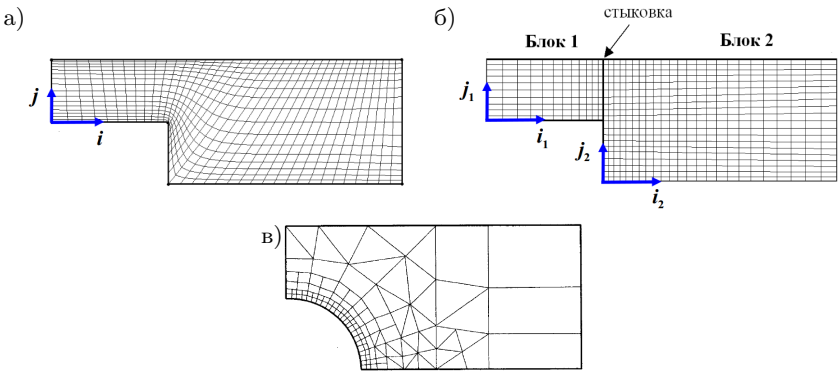

Рис. 1.4. Типы сеток: а) структурированная; б) блочно-структурированная; в) неструктурированная

неструктурированной же сетки необходимо использовать более сложный формат хранения. Под форматом хранения мы здесь понимаем описание некоторых характеристик ячеек и связей между ними, которые однозначно позволяют восстановить структуру сетки и её геометрическое положение в пространстве (такой информацией может служить, например, тип ячейки, координаты её вершин и список её соседей). Также очевидно, что структурированные сетки можно хранить в формате неструктурированных, а вот обратное неверно.

Таким образом, если программа-решатель использует структурированный подход для хранения сетки, то очевидно, что она может работать только со структурированными сетками; в этом случае решатель называется структурированным. Если же формат хранения сетки (а также связанных с ней полей переменных) позволяет использовать разные типы сеток, то такой решатель можно называть неструктурированным. Понятно, что использование произвольных неструктурированных сеток открывает больше возможностей при их построении (и зачастую сильно облегчает этот процесс), чем использование структурированных. Однако, достоинством структурированных решателей является возможность в некоторых случаях использовать схемы дискретизации повышенного порядка точности (о чём более подробно будет говориться далее). Поэтому зачастую для об-

ластей сложной геометрии используют различные подходы, расширяющие область применения структурированных сеток, в частности, за счёт использования уже упомянутых блочно-структурированных сеток, либо других подходов, одним из которых является использование многоблочных структурированных сетках с перекрывающимися блоками типа Chimera, когда расчётная область разбивается на несколько пересекающихся блоков, в каждом из которых строится своя (обычно структурированная) сетка, в области пересечения блоков данные переинтерполируются с одной сетки на другую (подробнее см. [5], главу 11 и другие главы).

1.2.2. Ячейка расчётной сетки и дискретизация уравнения в интегральной форме по МКО

После того, как построена расчётная сетка, для применения метода конечных объёмов необходимо выделить собственно эти конечные объёмы, для которых будут записаны балансовые соотношения в интегральной форме.

Здесь существует два подхода:

- конечный объём является ячейкой сетки (рисунок 1.5а);
- конечный объём строится вокруг узла расчётной сетки (рисунок 1.5б).

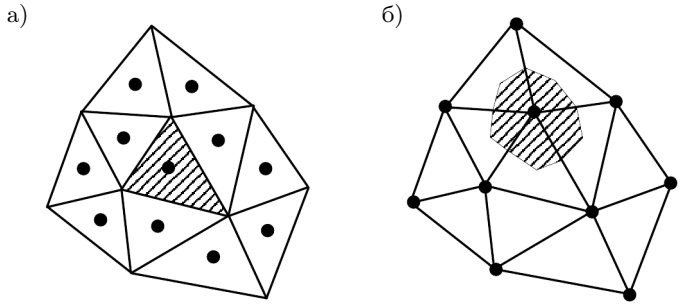

Рис. 1.5. Два подхода к выделению конечного объёма: а) конечный объём — ячейка сетки; б) конечный объём строится вокруг узла сетки

Первый предполагает, что в качестве конечного объёма выступает ячейка сетки. Этот подход является наиболее распространённым по нескольким причинам, одна из которых — простота выделения объёма, геометрические параметры которого по сути уже определены на этапе построения сетки. Второй подход требует наличия в решателе некоторой процедуры создания конечного объёма вокруг узла, причём соседние объёмы должны иметь общие грани, т.е. не должно быть «просветов» между объёмами. При таком подходе могут возникнуть сложности аппроксимации интегральных соотношений для конечных объёмов, построенных вблизи границ расчётной области. Преимуществом такого подхода является возможность контролировать качество построенного объёма. Поскольку в пакете OpenFOAM используется первый подход, далее будем рассматривать только его.

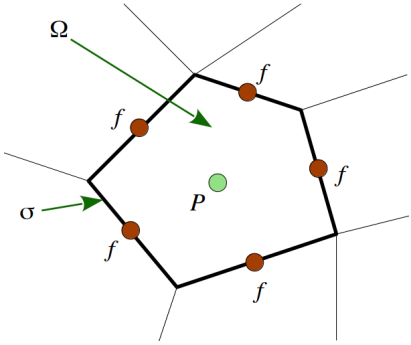

Рис. 1.6. Ячейка расчётной сетки с центром объёма P и центрами граней f

Используя приведённый выше пример для уравнения переноса примеси потоком (см. раздел 1.1.2) покажем, как производится дискретизация по методу конечных объёмов. Поскольку интегральная запись (1.5) этого уравнения справедлива для произвольного объёма, то она верна и для ячейки расчётной сетки. Рассмотрим некоторую (произвольную) ячейку расчётной сетки и введём следующие обозначения (см. рисунок 1.6): центр расчётной ячейки обозначим буквой P, центр каждой грани ячейки будем обозначать одной и той же буквой f (для уменьшения количества вводимых обозначений; центр грани с номером i можно также обозначить как f_i). Объём ячейки как и ранее обозначен буквой Ω, ограничивающая его поверхность — σ. Площадь каждой грани обозначается S_f, так что можно написать

$$\sigma = \sum_f S_f,$$

т.е. суммарная площадь поверхности ячейки равна сумме площадей всех граней. Нормаль для каждой грани обозначается как $\vec{n}_f$.

Уравнение переноса концентрации в интегральной форме (1.5) включает в себя слагаемое с интегрированием по объёму и слагаемое с интегрированием по поверхности. Рассмотрим сначала второе слагаемое и перепишем его в виде суммы интегралов по граням ячейки:

$$\int\limits_{\sigma} \vec{n} \cdot (c\vec{V} + \vec{q}) d\sigma = \sum_f \int_{S_f} \vec{n}_f \cdot (c\vec{V} + \vec{q}) d\sigma.$$

Ключевой шаг для метода конечных объёмов — дискретизация каждого интеграла в сумме с использованием квадратурной формулы. Строго говоря, можно воспользоваться теоремой о среднем и заменить интеграл на произведение значения подынтегральной функции в некоторой точке грани и площади грани, однако, такая замена будет точной только для одной заранее неизвестной точки грани. Однако, приближённо можно взять эту точку в центре грани, и тогда, как известно, получаемая квадратурная формула имеет порядок точности $o(S_f \Delta^2)$, где Δ — линейный размер ячейки. Таким образом, дискретизация интеграла осуществляется по формуле:

$$\int_{S_f} \vec{n}_f \cdot (c\vec{V} + \vec{q}) d\sigma \approx \vec{n}_f \cdot (c\vec{V} + \vec{q})_f S_f,$$

где величина $(c\vec{V} + \vec{q})_f$ приписывается центру грани. Отсюда:

$$\int\limits_{\sigma} \vec{n} \cdot (c\vec{V} + \vec{q}) d\sigma \approx \sum_f \vec{n}_f \cdot (c\vec{V} + \vec{q})_f S_f.$$

Аналогичным образом поступают с интегралом по объёму, приписывая значение подынтегральной величины центру объёма (данная квадратурная формула имеет порядок точности $o(\Omega\Delta^2)$):

$$\int\limits_{\Omega} c d\Omega \approx c_P \Omega.$$

Вопрос о том, что подразумевается под «значением величины на грани», мы рассмотрим позже. Пока будем считать, что это значение не влияет на порядок дискретизации, хотя (и это важно запомнить!) это совсем не так, и рассмотрим чуть подробнее вопрос о порядке точности полученных аппроксимаций. Оставляя в стороне вопрос о дискретизации временной

производной (он будет рассмотрен позже), запишем балансовое соотношение (1.5) в дискретизированном виде (с заменой $c\vec{V} + \vec{q}$ на так называемый вектор потока $\vec{Q}$):

$$\frac{\partial}{\partial t}(c_P \Omega) + \sum_f \vec{n}_f \cdot \vec{Q}_f S_f = 0. \tag{1.7}$$

Порядок точности полученного дискретного уравнения определяется тем, насколько уменьшается погрешность аппроксимации при уменьшении линейного размера конечного объёма. Для простоты в качестве конечного объёма рассмотрим куб со стороной a. В этом случае, деля выражение (1.7) на $\Omega = a^3$ и учитывая направления нормалей, получим:

$$\frac{\partial c_P}{\partial t} + \frac{Q_{x2} - Q_{x1}}{a} + \frac{Q_{y2} - Q_{y1}}{a} + \frac{Q_{z2} - Q_{z1}}{a} = 0, \tag{1.8}$$

где Q_{x2}, Q_{x1} и т.п. — значение проекции величины $\vec{Q}$ на соответствующую грань. Как видно, выражения, содержащие разности Q, представляют не что иное, как аппроксимацию первой производной по соответствующей координате от проекции $\vec{Q}$ на грань, т.е. выражение (1.8) является конечно-разностной аппроксимацией дифференциального уравнения

$$\frac{\partial c}{\partial t} + \nabla \cdot \vec{Q} = 0.$$

Поскольку величина c_P берётся в центре куба, то аппроксимация (1.8) является аппроксимацией второго порядка точности $o(\Delta^2)$, как следует из анализа конечно-разностных схем[1]. С учётом того, что схема вычисления $\vec{Q}_f$ на грани влияет на суммарный порядок аппроксимации, можно сделать следующий вывод: порядок дискретизации по представленной выше схеме аппроксимации по методу конечного объёма не может быть выше второго. Для повышения порядка при дискретизации интегралов необходимо использовать квадратурные формулы более высоких порядков (что, однако, не гарантирует повышение порядка точности в целом), однако, такой подход весьма сложен в реализации, поэтому практически не применяется.

Итак, при дискретизации уравнения в интегральной форме по методу конечных объёмов мы получаем для каждой ячейки уравнение типа (1.7). Возникает вопрос: как решать полученную систему? Считать, что неиз-

[1] Подробнее о порядке дискретизации для метода конечных разностей см., например, [6]

вестными являются и значения величины в центре ячейки, и значения её в центре грани, нельзя, так как получается недоопределённая система (количество уравнений, равное количеству ячеек сетки, будет в этом случае заметно меньше, чем количество неизвестных). Поэтому обычно искомыми величинами являются значения в центрах ячеек, а неизвестные величины на грани необходимо каким-то образом рассчитать исходя из значений в центрах ячеек. Такой способ определения искомых величин (т.е. когда все они находятся в центрах ячеек) называется «collocated grid», т.е. «совмещённые сетки»; он является наиболее универсальным (особенно для неструктурированных сеток) и используется в том числе и в пакете OpenFOAM. Существуют также схемы, когда часть переменных определяется в центрах ячеек, а другая часть — в центрах граней (т.н. шахматные сетки или staggered grid, т.е. разнесённые сетки), однако, в данном пособии они не рассматривается.

Как уже отмечалось выше, схема вычисления величины в центре грани влияет на порядок дискретизации всего уравнения. Кроме того, от этого также может зависеть устойчивость численной схемы. Подробнее об этом будет рассказано в разделе 1.3.

1.2.3. Некоторые замечания о вычислении геометрических параметров ячейки

При разбиении расчётной области на ячейки произвольного типа могут возникать некоторые неоднозначности. Например, даже при разбиении на шестигранники может возникнуть ситуация, когда четыре ребра одной грани не лежат в одной плоскости; по сути это означает, что «шестигранник» на самом деле является фигурой с бо́льшим числом граней, однако, обычно никакой информации, каким образом данная «грань» должна быть разбита на «подграни» (т.е. треугольники), не имеется. Поэтому решатель должен рассчитать геометрические характеристики ячейки таким образом, чтобы удовлетворить некоторым тождествам.

Под геометрическими параметрами ячейки будем понимать следующее: объём ячейки Ω, центр объёма P, площадь каждой грани S_f, нормаль к каждой грани $\vec{n}_f$, центр грани f. В настоящем пособии мы не будем по-

дробно останавливаться на непосредственных способах вычисления всех этих характеристик, однако, сделаем несколько важных замечаний.

Введём понятие вектора грани ячейки $\vec{S}_f$, которое в дальнейшем будем всюду использовать, как

$$\vec{S}_f \equiv S_f \vec{n}_f. \tag{1.9}$$

Для того, чтобы вычисления по МКО не приводили к появлению «паразитных» источников, необходимо, чтобы способ вычисления $\vec{S}_f$ приводил к выполнению следующего тождества (желательно, с машинной точностью):

$$\sum_f \vec{S}_f \equiv 0, \tag{1.10}$$

т.е. сумма векторов граней по всем граням ячейки равна нулю.

В принципе, для этого достаточно вычислять вектор грани как векторную сумму произведения площади каждого треугольника, на которые разбивается данная грань, и нормали к этому треугольнику. Можно показать, что в этом случае способ триангуляции грани не влияет на результирующее значение вектора грани. Заметим, что необходимость выполнения (1.10) прямо следует из конечно-объёмной аппроксимации очевидного равенства

$$\operatorname{grad} 1 = 0.$$

В пакете OpenFOAM для вычисления вектора грани сначала вычисляются координаты центральной точки как среднее арифметическое радиус-векторов узлов грани, а затем строятся треугольники, состоящие из этого центра и двух последовательных узлов грани.

Для вычисления объёма ячейки Ω её также разбивают на тетраэдры и суммируют их объёмы. Здесь желательно при разбиении соседних ячеек не допускать появления «зазоров» между ними, обеспечивая таким образом выполнения тождества

$$\sum \Omega \equiv \Omega_\Sigma,$$

где суммирование производится по всем ячейкам сетки, а за Ω_Σ обозначен объём расчётной области.

Что же касается процедуры вычисления центров объёма и грани, то здесь имеется некоторая неоднозначность, поскольку есть несколько примерно равноценных способов. Одним из наиболее употребительных способов (который также используется и в пакете OpenFOAM) является вычисление центра объёма или грани как центра тяжести соответствующей фигуры. Другим способом является вычисление этих величин как центра тяжести «каркаса», где в качестве «каркаса» выступают: для грани — рёбра, а для объёма — либо грани, либо также рёбра. Отметим, что существенная разница в расчётах при использовании разных способов вычисления центров наблюдается только для «плохих» ячеек, т.е. ячеек, либо неортогональных, либо сильно вытянутых по одному направлению и неоднородных по другим, либо имеющих сложную топологию. В остальных случаях все эти способы дают практически одинаковое положение центра и не влияют на результат расчёта. Отметим однако, что если в качестве центра грани использовать её центр тяжести, то вычисленный по МКО дифференциальный оператор от линейного поля будет давать точное решение, в том числе для произвольных ячеек с плоскими гранями.

Качество расчётной сетки определяется качеством составляющих её ячеек. Как уже упоминалось, есть несколько важных параметров ячеек, влияющих на расчёт, и критериев оценки качества сетки. Одним из таких параметров является скошенность ячейки (skewness). Под этим термином в OpenFOAM понимается то, насколько сильно точка пересечения линии, соединяющей центры соседних ячеек, отходит от центра соответствующей грани; для нескошенных ячеек эта точка совпадает с центром грани. Другой параметр — ортогональность, т.е. угол (или синус угла) между нормалью к грани ячейки и линией, соединяющей центр ячейки и центр грани; сетка полностью ортогональна, если этот угол равен нулю.

Ещё один важный параметр сетки — это её «вытянутость» или aspect ratio (соотношение размеров ячейки в разных направлениях). Например, для двумерных сеток из прямоугольных ячеек это величина равна отношению длинной стороны ячейки к короткой. В некоторых случаях эта величина может достигать значений порядка 1000 и более, что может иметь место, например, в задаче с сильной неоднородностью разрешаемых особенностей поля (градиентов) в разных направлениях (почти однородное

поле в одном направлении и сильно меняющееся в поперечном); очевидно, что для такой задачи не имеет смысла сильно мельчить сетку в продольном направлении. Для задач гидроаэродинамики такая ситуация типична при разрешении пограничных слоёв вблизи стенки.

Помимо перечисленных параметров, которые могут влиять на качество дискретизации и в конечном итоге на качество решения в целом, также стоит отметить степень сгущения ячеек, т.е. отношение линейного размера одной ячейки к размеру следующей ячейки вдоль одного направления. Как будет показано далее, этот параметр влияет на порядок дискретизации при вычислении величин в центре грани ячейки.

Наилучшим с точки зрения точности расчёта является использование полностью ортогональных ячеек с параметром «aspect ratio» близким к 1. В этой связи лучше всего использовать сетки, состоящие из кубических либо полиэдральных элементов. Тетраэдральные сетки также могут иметь хорошее качество, однако, они несколько хуже с точки зрения численного расчёта из-за того, что каждая ячейка «контактирует» только с четырьмя соседями (тогда как для сеток из шестигранников число соседей равно шести), поэтому «связность» ячеек друг с другом весьма «слабая». Несколько подробнее об этом будет говориться далее.

1.2.4. Вычисление дифференциальных операторов по МКО

Выше при формулировке метода конечных объёмов мы использовали интегральную запись дифференциальных уравнений. Данное пособие ориентировано на работу с пакетом OpenFOAM, в котором как бы условно предполагается, что метод конечных объёмов применяется для дискретизации дифференциальных операторов, т.е. замене какого-либо дифференциального оператора на его аппроксимацию по методу конечного объёма. Например, дискретизация дивергенции вектора потока $\vec{Q}$ по МКО запишется в виде:

$$\nabla \cdot \vec{Q} \approx \frac{1}{\Omega} \sum_f \vec{Q}_f \cdot \vec{S}_f,$$

что совпадает с дискретизацией соответствующего слагаемого в уравнении (1.7) с точностью до деления на объём ячейки и по сути следует из определения дифференциальных операторов по формуле (1.6). Таким образом,

можно дискретизацию по МКО делать исходя сразу из дифференциальной формулировки как бы минуя переход к интегральной форме; отличие будет только в множителе Ω. То есть, например, дискретизируя уравнение переноса в дифференциальной форме (1.3) по МКО получим такое уравнение:

$$\frac{\partial c_P}{\partial t} + \frac{1}{\Omega}\sum_f \left(c\vec{V} + \vec{q}\right)_f \cdot \vec{S}_f = 0,$$

которое полностью совпадает с (1.7)[1].

Запишем конечно-объёмные аппроксимации различных дифференциальных операторов (см. раздел 1.1.1), неявно подразумевая переход от дифференциальной записи к интегральной при помощи формулы Гаусса-Остроградского (1.6) ($\vec{a}$ — вектор, T — тензор; отметим, что для некоторых операций необходимо соблюдать правильный порядок следования сомножителей):

дивергенция вектора:	$\nabla \cdot \vec{a} \approx \frac{1}{\Omega}\sum_f \vec{S}_f \cdot \vec{a}_f$
ротор вектора:	$\nabla \times \vec{a} \approx \frac{1}{\Omega}\sum_f \vec{S}_f \times \vec{a}_f$
градиент вектора:	$\nabla \vec{a} \approx \frac{1}{\Omega}\sum_f \vec{S}_f \vec{a}_f$
дивергенция тензора:	$\nabla \cdot T \approx \frac{1}{\Omega}\sum_f \vec{S}_f \cdot T_f$
ротор тензора:	$\nabla \times T \approx \frac{1}{\Omega}\sum_f \vec{S}_f \times T_f$

Такая дискретизация дифференциальных операторов будет тем точнее, чем меньше рассматриваемый объём Ω. Отсюда, очевидно, следует, что решение дискретизированного по МКО уравнения должно стремится к решению соответствующего дифференциального уравнения при измельчении сетки и в этом смысле разница между интегральной формулировкой и дифференциальной практически несущественна. Тем не менее, такой подход сильно отличается от того же метода конечных разностей: в МКР дискретизируются непосредственно сами производные. На равномерных ортогональных сетках зачастую оказывается, что МКО и МКР дают идентичные

[1] Такой переход справедлив для постоянного Ω, т.е. неподвижных сеток. Случай движущихся сеток необходимо рассматривать отдельно, но в данном пособии этот вопрос не затрагивается

конечно-разностные формулы. Различия начинаются при использовании неравномерных сеток и/или сеток с ячейками, отличными от шестигранников (отметим, что на практике не встречается использование МКР в применении к сеткам, отличным от шестигранных). Кроме того, разрывные решения строго говоря не могут быть решениями дифференциальных уравнений, но вполне допустимы в качестве решения интегральных уравнений, именно поэтому интегральную формулировку использовать предпочтительней.

1.3. Решение уравнений гидроаэродинамики по МКО

В данном разделе подробно рассматривается метод конечных объёмов применительно к задачам гидроаэродинамики и теплообмена. Вначале (разделы 1.3.1, 1.3.2) рассматривается дискретизация уравнения конвективно-диффузионного переноса, которое описывает процессы переноса примеси, тепла и др. в движущейся (несущей) среде. Затем (раздел 1.3.3) формулируется проблема численного моделирования уравнений динамики несжимаемой жидкости и даётся краткий обзор основных подходов к её решению (раздел 1.3.4).

1.3.1. Дискретизация уравнения конвективно-диффузионного переноса

Рассмотрим дискретизацию уравнения переноса в случае использования совмещённых сеток (т.е. когда все искомые величины задаются в центрах ячеек). Как и ранее, в качестве примера будет выступать уравнение переноса примеси в движущейся среде (раздел 1.1.2), дискретизированное по МКО, см. раздел 1.2.2, формула (1.7). Перепишем это уравнение с учётом закона Фика для диффузионного потока $\vec{q}$ (1.4) и определения вектора грани ячейки (1.9):

$$\frac{\partial}{\partial t}\left(c_P \Omega\right) + \sum_f \left(c\vec{V}\right)_f \cdot \vec{S}_f - \sum_f \left(D\nabla c\right)_f \cdot \vec{S}_f = 0. \tag{1.11}$$

Выражение $\left(c\vec{V}\right)_f \cdot \vec{S}_f$ называется *конвективным* потоком через грань ячейки, оно определяет перенос примеси за счёт движения несущей среды. Выражение $\vec{q} \cdot \vec{S}_f = -\left(D\nabla c\right)_f \cdot \vec{S}_f$ называется *диффузионным* потоком и

характеризует перенос примеси через грань ячейки за счёт диффузионных процессов. Поэтому уравнение (1.11) называется уравнением конвективно-диффузионного переноса. Подобное уравнение может быть записано, например, для переноса тепла в потоке; также в правой части могут быть различные источниковые члены (объёмное тепловыделение, источник примеси за счёт химических реакций и т.п.).

Вычисление диффузионного слагаемого

Рассмотрим сначала в (1.11) слагаемое с диффузионным потоком. Для его вычисления необходимо рассчитать значение $D\nabla c$ в центре каждой грани ячейки. В случае постоянного коэффициента диффузии[1] D можно записать:

$$D\left(\nabla c\right)_f \cdot \vec{S_f} = D\left(\nabla c\right)_f \cdot \vec{n}_f S_f = D\left.\frac{\partial c}{\partial n}\right|_f S_f.$$

Для дискретизации производной по нормали на грани ячейки можно использовать различные аппроксимации. Наиболее распространённым является гибридный способ.

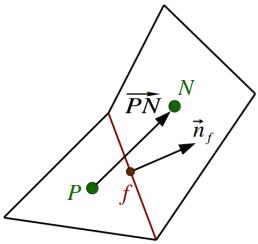

Рис. 1.7. Грань f и две примыкающие к ней ячейки

Рассмотрим грань f с внешней нормалью $\vec{n}_f$, разделяющую две ячейки с центрами P и N (см. рисунок 1.7)[2]. Вектор $\overrightarrow{PN}$ — это вектор, соединяющий центры ячеек. Запишем очевидное равенство:

$$\left.\frac{\partial c}{\partial n}\right|_f = \left(\nabla c\right)_f \cdot \frac{\overrightarrow{PN}}{|\overrightarrow{PN}|} + \left(\nabla c\right)_f \cdot \left(\vec{n}_f - \frac{\overrightarrow{PN}}{|\overrightarrow{PN}|}\right). \tag{1.12}$$

Первое слагаемое в правой части этого равенства представляет собой производную по направлению $\overrightarrow{PN}$. В случае, когда векторы $\vec{n}_f$ и $\overrightarrow{PN}$ практически коллинеарны, вклад второго слагаемого близок к нулю; для «хороших» сеток такая ситуация вполне типична. Аппроксимация первого слагаемого может быть выполнена по формуле:

[1] Для переменного по пространству D в большинстве случаев можно считать $\left(D\nabla c\right)_f \approx D_f\left(\nabla c\right)_f$, D_f находится путём интерполяции на грань из центров ячеек

[2] Здесь и далее подразумевается, что конечно-объёмные аппроксимации пишутся для объёма с центром P, окружённого ячейками с центрами N

$$(\nabla c)_f \cdot \frac{\overrightarrow{PN}}{|\overrightarrow{PN}|} \approx \frac{c_N - c_P}{|\overrightarrow{PN}|},$$

которая обеспечивает второй порядок точности в случае, если центр грани f делит вектор $\overrightarrow{PN}$ ровно пополам, что имеет место в случае отсутствия сгущения ячеек вдоль этого направления (отсюда следует, что слишком сильное изменение размера ячеек при переходе от одной к другой приводит к уменьшению точности дискретизации по гибридной схеме, формально до первого порядка; практика расчётов показывает, что эта ошибка мала, если коэффициент сгущения не выходит за диапазон [0.9;1.1]). Второй порядок дискретизации здесь обеспечивает также второй порядок дискретизации по МКО в целом для диффузионных слагаемых. Следует отметить, что вопрос вычисления диффузионных слагаемых для граничных граней требует отдельного обсуждения, но в данном пособии не рассматривается.

Что касается аппроксимации второго, «корректирующего» слагаемого, то здесь необходимо знать значение градиента величины на грани. Для этого можно использовать интерполяцию градиентов из центров ячеек, например, линейную интерполяцию по формуле:

$$(\nabla c)_f = \beta\,(\nabla c)_N + (1-\beta)\,(\nabla c)_P\,, \quad \beta = \frac{Pf}{Pf + Nf},$$

где через Pf и Nf обозначены расстояния между центрами ячеек и центром грани.

Описанный выше гибридный метод даёт следующие преимущества. Во-первых, он относительно легко реализуется. Во-вторых, поскольку вклад второго слагаемого на хороших сетках обычно значительно меньше, чем вклад первого слагаемого, то в итерационном процессе, который неизбежно возникает при решении сложных задач, поправка на неортогональность может учитываться явным образом, т.е. значение градиента величины берётся с предыдущей итерации. Именно так сделано в OpenFOAM; имеется даже такое понятие, как итерации по неортогональности (параметр nNonOrthogonalCorrectors; подробнее см. раздел 3.1.7). Другие схемы вычисления диффузионного потока на грани либо обладают худшими свойствами, либо существенно более труднореализуемые. В частности, схема вычисления диффузионного слагаемого с использованием только градиен-

та величины, интерполированного из центров ячеек, может давать осциллирующее на сетке решение, поэтому практически не применяется.

Вычисление градиента в центре ячейки может осуществляться различными способами. Здесь мы рассмотрим два возможных варианта: формулу Грина-Гаусса и метод наименьших квадратов.

Формула Грина-Гаусса представляет собой не что иное, как вычисление градиента по МКО:

$$(\nabla c)_P = \frac{1}{\Omega} \sum_f c_f \vec{S}_f,$$

где c_f — значение величины на грани, которое получается путём интерполяции из центров ячеек, например, также при помощи линейной интерполяции: $c_f = \beta c_N + (1 - \beta) c_P$.

Метод наименьших квадратов исходит из задачи подбора такого значения $(\nabla c)_P$, которое минимизирует сумму квадратов:

$$\sum_N w_N^2 \left[c_N - \left(c_P + \overrightarrow{PN} \cdot (\nabla c)_P \right) \right]^2 \to \min,$$

где суммирование идёт по всем ячейкам N, являющимися по отношению к ячейке P соседними[1], а w_N — весовые функции, в качестве которых лучше всего выбирать $1/|\overrightarrow{PN}|$. Отметим, что метод наименьших квадратов даёт точное значение градиента для квадратичной зависимости c от координаты, тогда как метод Грина-Гаусса точен только для линейных функций. Кроме того, метод Грина-Гаусса даёт большие погрешности в случае сильно скошенных неортогональных ячеек, приводя иногда даже к расходимости расчёта. В целом же, способ вычисления градиентов в центре ячеек (если он используется только для корректирующего слагаемого при вычислении диффузионного потока на грани) не сильно сказывается на общей точности решения в случае сеток приемлемого качества.

Вычисление конвективного слагаемого

Обратимся теперь к вопросу вычисления конвективного слагаемого в (1.11). Сразу подчеркнём, что этот вопрос является одним из самых «трудных» в численном моделировании (в отличие от аппроксимации диффузи-

[1] Например, имеющие общую с ячейкой P грань

онных слагаемых). Связано это с тем, что задачи с чисто конвективный переносом допускают разрывные решения. Простейший пример — стационарный перенос примеси в однородном потоке ($\vec{V} = const$), описываемый уравнением:

$$\vec{V} \cdot \nabla c = 0.$$

Нетрудно убедиться, что функция c, которая является постоянной вдоль линий тока и произвольной (в т.ч. и разрывной) в поперечном к ним направлении, удовлетворяет этому уравнению[1]. Численное решение такого, казалось бы, элементарного уравнения приносит немало сюрпризов.

Во многих случаях конвективный поток на грани можно «факторизовать», т.е. записать в виде[2]:

$$\left(c\vec{V}\right)_f \cdot \vec{S}_f \approx c_f \vec{V}_f \cdot \vec{S}_f.$$

Величина $\vec{V}_f \cdot \vec{S}_f$ представляет собой (объёмный) расход через грань. Вопрос о её вычислении будет далее рассматриваться несколько более подробно. В целом, можно использовать простую линейную интерполяцию: $\vec{V}_f = \beta\vec{V}_N + (1-\beta)\vec{V}_P$.

Что касается вычисления c_f, то здесь ситуация более сложная. Связано это с тем, что использование простой линейной интерполяции на грань для определения c_f приводит к центрально-разностной схеме, которая является абсолютно неустойчивой. Проиллюстрируем это на простейшем примере. Рассмотрим одномерную задачу стационарного переноса примеси потоком с постоянной скоростью $u = const$:

$$u\frac{\partial c}{\partial x} = 0. \tag{1.13}$$

Очевидно, аналитическое решение этого уравнения $c = const$. Для дискретизации его по МКО введём равномерную сетку с постоянным шагом Δx, центры «объёмов» будем обозначать номером i, соседей слева и справа — $i-1$ и $i+1$, грани — $i-\frac{1}{2}$ и $i+\frac{1}{2}$. Дискретизация по МКО даёт:

[1] Здесь не учитываются граничные условия, которые и определяют в итоге вид функции c

[2] Такая факторизация может давать неустойчивое численное решение в задачах, когда переносимая величина и скорость потока жёстко связаны друг с другом, например, при высокоскоростном движении сжимаемого газа. В этом случае необходимо применять другие схемы вычисления конвективного потока на грани

$$u\frac{c_{i+1/2}-c_{i-1/2}}{\Delta x}=0.$$

Используя линейную интерполяцию для вычисления значений на грани

$$c_{i+1/2}=\left(c_{i+1}+c_{i-1}\right)/2,$$

можно записать результирующее уравнение:

$$u\frac{c_{i+1}-c_{i-1}}{2\Delta x}=0. \tag{1.14}$$

Отметим, что данное выражение представляет собой не что иное, как конечно-разностную аппроксимацию рассматриваемого дифференциального уравнения по центрально-разностной схеме (т.е. аппроксимацию производной от c в точке i по центральной схеме второго порядка точности). Видно, что такому конечно-разностному уравнению удовлетворяет как решение $c_i = const$, $i = 1, 2, 3, ...$, так и «осциллирующее» решение $c_{2n} = const_1$, $c_{2n+1} = const_2$, $n = 1, 2, 3, ...$, $const_1 \neq const_2$ (см. рисунок 1.8).

Таким образом, представленный пример иллюстрирует возможную неустойчивость центрально-разностной схемы. Как показывает практика расчётов, и в случае произвольных трёхмерных сеток простая линейная интерполяция на грань переносимой величины даёт подобное осциллирующее решение, которое в результате накопления ошибок приводит к расходимости.

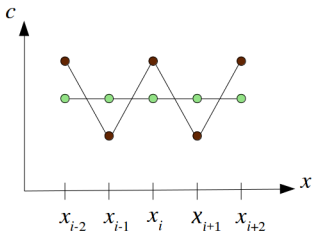

Рис. 1.8. Возможные решения уравнения (1.14) при использовании центрально-разностной схемы

Ситуация улучшается, если в уравнении наряду с конвективным слагаемым присутствует также и диффузионное (что в принципе типично для реальных физических задач), которое в свою очередь подавляет (сглаживает) возникающие неоднородности. Однако, центрально-разностная численная схема при наличии диффузионных слагаемых устойчива только, если сеточное число Пекле достаточно мало: $Pe = VL/D < 2$ (V — скорость в ячейке, L — размер ячейки, D — коэффициент диффузии). На практике зачастую коэффициент диффузии очень мал, поэтому для выполнения данного условия необ-

ходимо сильно мельчить сетку, что приводит к неоправданно большому числу ячеек и, соответственно, длительному времени счёта.

Другим способом обеспечить устойчивость численной схемы и избежать появления нефизических осцилляций является использование так называемых противопоточных схем. В основе этих схем лежит такое физическое соображение, что при переносе какой-либо величины через грань расчётной ячейки естественно брать значение этой величины выше по потоку (поскольку она переносится от ячейки выше по потоку к ячейке ниже по потоку от грани; можно показать, что численная схема, в которой значение на грани берётся *ниже* по потоку, также приводит к некорректным решениям).

Рассмотрим грань между двумя ячейками (см. рисунок 1.9). Центр ячейки, из которой направлен вектор скорости на грани $\vec{V}_f$, обозначим через U, а центр другой ячейки — D (от слов upwind и downwind, т.е. выше по потоку и ниже). Наиболее простая противопоточная схема записывает так:

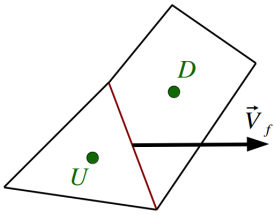

Рис. 1.9. Схема обозначений ячеек выше (U) и ниже (D) по потоку

$$c_f = c_U. \tag{1.15}$$

Это противопоточная схема первого порядка точности. Утверждение о первом порядке точности можно проиллюстрировать на том же примере (1.13). Если $u > 0$, то такая противопоточная схема приводит к следующей аппроксимации уравнения (1.13) в ячейке i:

$$u\frac{c_i - c_{i-1}}{\Delta x}, \tag{1.16}$$

что соответствует конечно-разностной аппроксимации производной в точке i с первым порядком точности (так называемая аппроксимация «разностью назад»).

Важно подчеркнуть, что применение такой схемы для вычисления значения переносимой величины на грани приводит к понижению порядка точности МКО до первого. С точки зрения численного решения оказывается, что противопоточная схема первого порядка привносит дополнительную

численную диффузию (или иначе численную диссипацию) таким образом, как будто бы в уравнение добавлено диффузионное слагаемое первого порядка точности. Эта численная диффузия ведёт к сильному сглаживанию неоднородностей в решении (в том числе и физически обоснованных), так что полученное численное решение может значительно расходиться с аналитическим (особенно хорошо это заметно при расчёте задачи с разрывами). Поэтому использование противопоточных схем первого порядка в целом не рекомендуется (хотя в ряде случае, например, в задачах газовой динамики, их необходимо применять локально, в частности, в области сильных разрывов). Отметим, что поскольку центрально-разностная схема, наоборот, приводит к возникновению осцилляций в решении, то её иногда ещё называют бездиссипативной.

Простейшая противопоточная схема второго порядка точности получается в случае, если экстраполировать значение переносимой величины из центра ячейки на грань по формуле:

$$c_f = c_U + (\nabla c)_U \cdot \overrightarrow{Uf}, \tag{1.17}$$

где $\overrightarrow{Uf}$ — вектор от центра ячейки к центру грани (очевидно, что приведённая формула является ничем иным, как разложением в ряд Тейлора относительно точки U с отбрасыванием малых второго порядка). Эта схема обеспечивает одновременно и второй порядок точности МКО, и добавку численной диффузии, однако, в ряде случаев (в области немонотонного изменения c) может давать значение c_f на грани, выходящее за пределы интервала $[c_U; c_D]$, что в свою очередь может приводить к появлению нефизических осцилляций и расходимости решения.

Таким образом, в общем случае вопрос о схеме интерполяции на грань при вычислении конвективных потоков остаётся открытым: для каждой задачи необходимо подбирать свою схему, обеспечивающую наибольшую точность решения при сохранении устойчивости численного метода. При этом каждая противопоточная схема вносит свою численную диффузию. В ряде задач (например, моделирование турбулентных течений на основе вихреразрешающих подходов) величина этой численной диффузии (и её порядок точности) играет существенную роль для получения качественного решения. В частности, несмотря на то, что в целом порядок точно-

сти МКО не может превышать второго, порядок точности именно интерполяционной схемы стараются выбрать как можно больший, поскольку, как показывает практика расчётов, схемы повышенного порядка точности обеспечивают лучшее разрешение особенностей для решаемой задачи (это может касаться также и порядка точности центральной бездиссипативной схемы: в ряде случаев имеет смысл использовать не простую линейную интерполяцию, дающую второй порядок, а более сложную, обеспечивающую порядок точности интерполяционной схемы третий или выше). Более подробно о схемах интерполяции на грань при вычислении конвективных потоков будет рассказано в разделе 3.1.

Отметим важное свойство, характерное для МКО. Очевидно, что значение конвективного или диффузионного потока на грани вычисляется по одной и той же формуле при дискретизации интегрального уравнения как для одной ячейки, так и для другой, для которых эта грань является смежной. Поэтому для обеих этих соседних ячеек выполняется свойство сохранения потока: сколько «вытекает» из одной ячейки, столько же и «втекает» в другую. Если просуммировать потоки по всем ячейкам, то из-за взаимной компенсации потоков на внутренних гранях останется только сумма потоков по внешней границе расчётной области. Таким образом, автоматически выполняется интегральный баланс и для всей расчётной области. В частности, если решение стационарное, это означает, что втекающий в расчётную область поток должен равняться вытекающему с точностью решения стационарной задачи. Можно сказать, что МКО обладает свойством *консервативности*. Подчеркнём, что в отличие от МКО, методы конечных разностей и конечных элементов не обеспечивают консервативность на любых расчётных сетках. Именно поэтому в задачах, где важную роль играет интегральное сохранение потоков (гидроаэродинамика, конвективный теплообмен и др.), наиболее часто применяется МКО. Следует отличать понятие консервативности численного метода от понятия консервативной записи уравнений сохранения (хотя в некоторой степени они связаны); в разделе 1.3.3 об этом будет сказано более подробно.

1.3.2. Аппроксимация временной производной

В уравнение (1.11) входит нестационарное слагаемое, которое для численного решения также необходимо дискретизировать. Поскольку МКО является методом дискретизации только пространственных операторов, то схемы аппроксимации временной производной здесь по сути никак с МКО не связаны и скорее следуют из метода конечных разностей.

Для дискретизации по времени необходимо ввести так называемую временную сетку, т.е. выделить дискретные моменты времени, в которые собственно и будет находиться численное решение. Запишем уравнение переноса (1.11), поделив его на объём Ω, перенеся все пространственные операторы в правую часть и обозначив их через R (также опустим индекс ячейки P):

$$\frac{\partial c}{\partial t} = -R \tag{1.18}$$

Введём дискретные моменты времени t_n, значения всех величин при $t = t_n$ будем обозначать верхним индексом n. Поскольку задача нестационарная, то для неё должно быть обязательно задано начальное условие, т.е. решается задача Коши. Простейшая явная схема Эйлера аппроксимации уравнения (1.18) записывает как:

$$\frac{c^{n+1} - c^n}{\Delta t} = -R^n, \tag{1.19}$$

где $\Delta t = t_{n+1} - t_n$, а правая часть уравнения вычисляется в момент времени t_n. Таким образом, алгоритм решения нестационарной задачи по явной схеме Эйлера весьма прост: в момент $t = t_0$ рассчитывается правая часть R^0 по МКО на основе заданных начальных полей, затем происходит продвижение на один шаг по времени до $t = t_1$ по приведённой формуле, затем снова происходит расчёт правой части уже при $t = t_1$ и так далее до тех пор, пока не будет достигнут заданный момент времени. Очевидно, что левая часть (1.19) представляет собой аппроксимацию производной с первым порядком точности (поскольку правая часть берётся при $t = t_n$), а порядок аппроксимации временной производной никак не связан с порядком аппроксимации пространственных операторов.

Прежде чем рассмотреть более сложные схемы аппроксимации, введём понятие числа Куранта[1]. Для этого запишем явную схему Эйлера для нестационарного одномерного уравнения переноса примеси с постоянной скоростью, где конвективное слагаемое аппроксимировано «разностью назад» (1.16):

$$\frac{c_i^{n+1} - c_i^n}{\Delta t} + u\frac{c_i^n - c_{i-1}^n}{\Delta x} = 0.$$

Показано (см., например, [6]), что такая явная схема устойчива, только если выполняется условие:

$$\mathrm{CFL} = u\frac{\Delta t}{\Delta x} < 1,$$

где CFL — безразмерный параметр, называемый числом Куранта-Фридрихса-Леви[2]. Таким образом, шаг по времени для явной схемы должен быть связан с размером ячейки сетки для обеспечения устойчивости численной схемы.

Для того, чтобы повысить устойчивость схемы, можно использовать неявную схему Эйлера, где правая часть должна вычисляться на «новом» шаге по времени t_{n+1}, то есть

$$\frac{c^{n+1} - c^n}{\Delta t} = -R^{n+1}.$$

В отличие от явной схемы Эйлера, здесь уже возникает система уравнений на временном слое t_{n+1}, поскольку в правой части уравнения для каждой ячейки входят не только величины в текущей ячейке на слое $n+1$, но и величины на этом слое в соседних ячейках. Размер этой системы уравнений равен количеству ячеек, и для реальных задач может быть очень большим, поэтому для нахождения неизвестных c^{n+1} обычно используется какой-либо итерационный метод решения систем линейных алгебраических уравнений (СЛАУ)[3]. Подробнее о методах решения СЛАУ будет сказано далее.

[1] Также общепринято название «число Куранта-Фридрихса-Леви»

[2] Для задачи чисто диффузионного нестационарного переноса аналогом числа Куранта является число фон Неймана (называемое также диффузионным числом Куранта) $\mathrm{VNN} = D\Delta t/\Delta x^2$

[3] Строго говоря, система может быть нелинейной, поэтому для её решения необходимо предварительно провести процедуру линеаризации

Доказано, что для нестационарного одномерного уравнения переноса примеси с постоянной скоростью неявная схема Эйлера устойчива при любом числе Куранта. На практике же из-за наличия нелинейностей различной природы, в т.ч. из-за особенностей аппроксимации конвективных слагаемых, а также по другим причинам, оказывается, что даже неявная схема может терять устойчивость при слишком большом значении числа Куранта. Кроме того, для хорошей точности разрешения по времени необходимо обеспечивать довольно малый шаг по времени, либо использовать схемы аппроксимации временной производно повышенного порядка точности. Одной из таких схем является схема Кранка-Николсон, дающая второй порядок точности по времени:

$$\frac{c^{n+1} - c^n}{\Delta t} = -\frac{1}{2}\left(R^{n+1} + R^n\right).$$

Особенностью этой схемы является то, что она нейтрально устойчива, т.е. сохраняет колебания во времени, если они имеют место. В некоторых задачах, где интересуют именно колебательные процессы, такая схема довольно привлекательна, однако, может в нелинейных задачах также приводить к расходимости. Поэтому для устойчивости вводят параметр α:

$$\frac{c^{n+1} - c^n}{\Delta t} = -\alpha R^{n+1} - (1-\alpha)R^n,$$

где при $\alpha = 0.5$ получаем схему Кранка-Николсон, при $\alpha = 0$ или 1 — явную или неявную схему Эйлера, а для лучшей устойчивости схемы Кранка-Николсон выбирают значение α чуть больше 0.5.

Более устойчивой, чем схема Кранка-Николсон, и имеющей второй порядок точности по времени является трёхслойная схема «разностью назад», которая для постоянного Δt записывается в виде:

$$\frac{3c^{n+1} - 4c^n + c^{n-1}}{2\Delta t} = -R^{n+1}.$$

Левая часть представляет собой аппроксимацию временной производной со вторым порядком точности в точке $n+1$. Видно, что для этой схемы требуется два предыдущих временных шага, поэтому для перехода от t_0 к t_1 требуется использовать другую схему, либо какое-нибудь допущение (например, $c^{-1} = c^0$).

1.3.3. Дискретизация уравнений Навье-Стокса

Рассмотрим теперь вопрос о дискретизации стационарных и нестационарных уравнений Навье-Стокса, которые описывают движение несжимаемой вязкой жидкости. В декартовой системе координат дифференциальная форма записи нестационарных уравнений Навье-Стокса выглядит следующим образом [1]:

$$\nabla \cdot \vec{V} = 0, \tag{1.20}$$

$$\frac{\partial \vec{V}}{\partial t} + \nabla \cdot \left(\vec{V}\vec{V}\right) = -\frac{1}{\rho}\nabla p + \nu \nabla^2 \vec{V}. \tag{1.21}$$

Здесь ρ — плотность среды, ν — кинематический коэффициент вязкости ($\nu = \mu/\rho$, где μ — динамический коэффициент вязкости), p — давление. Свойства среды ρ и ν являются заданными и постоянными, скорость $\vec{V}$ и давление p являются неизвестными полями. Первое уравнение называется уравнением неразрывности для несжимаемой среды (следует из (1.2) при $\rho = const$), оно представляет собой закон сохранения массы. Второе уравнение — уравнение баланса импульса; слагаемое с ν представляет собой действие вязких сил. Уравнение баланса импульса содержит нелинейность в конвективном слагаемом $\nabla \cdot \left(\vec{V}\vec{V}\right)$, где $\vec{V}\vec{V}$ является диадным произведением векторов скорости, т.е. тензором.

Помимо нелинейности, уравнения Навье-Стокса содержат другую важную особенность, которую можно проиллюстрировать следующим образом. Допустим, мы дискретизируем все пространственные операторы по МКО, а для временной производной от скорости запишем явную схему Эйлера. Но тогда мы получим систему, в которой давление входит только на старом временном слое, при этом уравнение неразрывности вообще ни для чего не пригодно! В этом и заключается важная особенность уравнений Навье-Стокса: сами по себе они являются «вырожденными» относительно давления p, поскольку напрямую его невозможно найти из данной системы. Иными словами, для давления отсутствует уравнение, которое бы содержало временную производную от p. Связано это с тем, что формально уравнение баланса импульса (1.21) может быть решено при любом распределении p, а поле давления должно быть подобрано таким образом, чтобы удовлетворялось уравнение неразрывности (1.20).

Одним из методов, позволяющим «обойти» указанную особенность, является метод искусственной сжимаемости, который сформулирован для решения стационарных[1] задач методом установления и заключается в добавлении искусственной временной производной от давления в уравнение неразрывности:

$$\frac{1}{\beta}\frac{\partial p}{\partial t} + \nabla \cdot \vec{V} = 0.$$

Здесь β — параметр метода. Данный метод отличается простотой и надёжностью с точки зрения численного моделирования, однако, в задачах с протоком жидкости по длинным каналам он обладает очень медленной сходимостью, что связано с распространением искусственных волн давления, движущихся с некоторой скоростью, близкой к скорости потока, по расчётной области.

Другой метод, широко используемый в настоящее время в той или иной форме, заключается в построении в некотором роде искусственного уравнения для давления типа уравнения Пуассона, о чём подробнее будет говориться в следующем разделе. Отметим, что для решения нестационарных задач широко используются так называемые проекционные методы (или, другими словами, методы дробных шагов; см. [6]), в которых также формулируется уравнение Пуассона для давления. Вообще, тот факт, что многие методы решения уравнений Навье-Стокса приводят к формулировке уравнения для давления типа Пуассона, имеет под собой некоторую физическую и математическую основу: это связано с тем, что формально скорость распространения возмущений в несжимаемой жидкости стремится к бесконечности, а это в свою очередь означает мгновенное распространение информации об изменении в любую точку пространства, что как раз отражается уравнением Пуассона. Подобным свойством обладают, например, и уравнения электростатики, в которых предполагается, что электрическое поле от зарядов распространяется мгновенно, поэтому для потенциала электрического поля также записывается уравнение Пуассона.

Уравнение баланса импульса (1.21) записано здесь в консервативной форме. Для перехода к неконсервативной форме перепишем конвективное слагаемое с учётом уравнения неразрывности (1.20):

[1] Метод искусственной сжимаемости без труда обобщается и на нестационарные задачи, подробней см. [7]

$$\nabla \cdot \left(\vec{V}\vec{V}\right) = \vec{V} \cdot \nabla \vec{V} + \vec{V}\cancelto{0}{\nabla \cdot \vec{V}} = \vec{V} \cdot \nabla \vec{V}.$$

Отметим, что такой переход можно сделать всегда, если выполняется уравнение неразрывности (в общем случае и для $\rho \neq const$). Кроме того, неконсервативная форма записи уравнения переноса (в т.ч. и уравнения баланса импульса) является «первоочередной», поскольку представляет собой конвективную производную при раскрытии полной (лагранжевой) производной какой-либо величины по времени (подробности см. в [1]):

$$\frac{d}{dt} = \frac{\partial}{\partial t} + \vec{V} \cdot \nabla.$$

В принципе, и уравнение в неконсервативной форме можно некоторым образом дискретизировать по МКО, однако, в этом случае не будет выполняться свойство сохранения потока величины через грань. Поэтому для применения МКО всегда уравнения переписывают (по возможности) в консервативную форму; одним из признаков консервативной формы записи уравнения является то, что все дифференциальные операторы стоят как бы «снаружи».

Выполним теперь дискретизацию уравнений Навье-Стокса по МКО (без аппроксимации временной производной). Для каждой ячейки P расчётной сетки получим[1]:

$$\sum_f \vec{V}_f \cdot \vec{S}_f = 0,$$

$$\frac{\partial \vec{V}_P}{\partial t} + \frac{1}{\Omega} \sum_f \vec{S}_f \cdot \left(\vec{V}\vec{V}\right)_f = -\frac{1}{\rho\Omega} \sum_f p_f \vec{S}_f + \frac{\nu}{\Omega} \sum_f \vec{S}_f \cdot \left(\nabla \vec{V}\right)_f.$$

В задачах динамики несжимаемой жидкости общепринята замена: $\left(\vec{V}\vec{V}\right)_f \approx \vec{V}_f \vec{V}_f$. Тогда конвективное слагаемое перепишется в виде:

$$\frac{1}{\Omega} \sum_f \left(\vec{V}_f \cdot \vec{S}_f\right) \vec{V}_f.$$

[1] Здесь дискретизация слагаемого с градиентом давления выполнена по МКО, т.е. по методу Грина-Гаусса, хотя, в принципе, можно использовать и другие методы дискретизации градиента, например, метод наименьших квадратов

Как уже упоминалось, величина $\vec{V}_f \cdot \vec{S}_f$ представляет собой объёмный расход жидкости через грань f. Обозначив её как $\varphi_f \equiv \vec{V}_f \cdot \vec{S}_f$, перепишем уравнения Навье-Стокса в дискретном виде:

$$\sum_f \varphi_f = 0, \tag{1.22}$$

$$\frac{\partial \vec{V}_P}{\partial t} + \frac{1}{\Omega} \sum_f \varphi_f \vec{V}_f = -\frac{1}{\rho\Omega} \sum_f p_f \vec{S}_f + \frac{\nu}{\Omega} \sum_f \vec{S}_f \cdot \left(\nabla \vec{V}\right)_f. \tag{1.23}$$

Дискретизация диффузионного слагаемого с ν здесь может быть выполнена точно также, как в разделе 1.3.1, если расписать его для каждой компоненты вектора $\vec{V}$. Что касается конвективного слагаемого, то поскольку оно нелинейно, необходимо произвести линеаризацию (за исключением случая использования явной схемы по времени для нестационарной задачи; в этом случае и φ_f, и $\vec{V}_f$ берутся с предыдущего временного слоя).

Если рассматривать только стационарные задачи, т.е. положить

$$\frac{\partial \vec{V}_P}{\partial t} \equiv 0,$$

то в этом случае стандартный путь линеаризации заключается в введении так называемых итераций по нелинейности: поток φ_f берут с предыдущей итерации, а $\vec{V}_f$ — с текущей.

Для нестационарных задач проблема нелинейности также может быть решена введением дополнительных итераций на каждом шаге по времени, либо путём использования каких-либо безытерационных подходов (например, упоминавшегося ранее метода дробных шагов). Практика расчётов показывает, что число итераций по нелинейности сильно возрастает при увеличении числа Куранта от единицы; в случае же малых значений числа Куранта зачастую можно их не делать.

При вычислении диффузионных слагаемых по гибридной схеме требуется знать значение градиента величины в центрах ячеек. При использовании неявных схем довольно проблематично расписать значение градиента величины на новом слое, поэтому зачастую поправочные добавки берут с предыдущей итерации/слоя по времени. Однако, в случае «плохих» ячеек эти поправки могут вносить довольно ощутимую коррекцию, из-за чего

иногда возникает необходимость вводить так называемые дополнительные итерации по неортогональности/скошенности.

Для уравнений Навье-Стокса (так же, как и для рассмотренного в разделе 1.3.1 уравнения переноса примеси) при вычислении конвективных слагаемых в большинстве случаев следует использовать противопоточные схемы для определения $\vec{V}_f$.

1.3.4. SIMPLE-подобные методы для решения задач динамики несжимаемой жидкости

Как отмечалась, одна из основных проблем численного решения системы Навье-Стокса для несжимаемой жидкости заключается в «подборе» такого поля давления, чтобы найденная из уравнения баланса импульса скорость $\vec{V}$ удовлетворяла уравнению неразрывности. В настоящее время существует группа методов, в которых для выполнения уравнение неразрывности специальным образом конструируется уравнение Пуассона для давления [8]. Исторически первым таким методом можно считать метод SIMPLE (Semi-Implicit Method for Pressure-Linked Equations), предложенный в [9]. Среди наиболее известных модификаций — методы SIMPLER (SIMPLE-Revised), SIMPLEC (SIMPLE-Consistent), PISO (Pressure Implicit Splitting of Operators). Методы этой группы используются в различных CFD-пакетах (в т.ч. и в OpenFOAM) и в настоящее время обладают в целом наилучшими свойствами по скорости и устойчивости сходимости итерационного процесса при решении уравнений Навье-Стокса. Ниже приведена классическая формулировка метода SIMPLE, используемая, в частности, в пакете OpenFOAM.

Будем рассматривать стационарную систему Навье-Стокса. Введём итерации по нелинейности (обозначая их при помощи верхнего индекса k) в уравнение баланса импульса (1.23) (здесь и далее под p подразумевается p/ρ):

$$\sum_f \varphi_f^k \vec{V}_f^{k+1} = -\sum_f p_f^{k+1} \vec{S}_f + \nu \sum_f \vec{S}_f \cdot \left(\nabla \vec{V}^{k+1}\right)_f . \qquad (1.24)$$

Здесь скорость $\vec{V}^{k+1}$ и модифицированное давление p^{k+1} являются неизвестными величинами на новой итерации $k+1$, φ^k — расход через грань

на старой итерации k. Выражение (1.24) записано для ячейки P. Значения скорости на грани находятся путём интерполяции из центров ячеек при помощи какой-либо противопоточной схемы. Диффузионное слагаемое может быть дискретизировано по гибридной схеме, см. (1.12), в которой поправка на неортогональность рассчитывается по значениям скорости с предыдущей итерации. Слагаемое с градиентом давления пока мы не будем расписывать, а наоборот, для простоты изложения заменим его конечно-объёмную аппроксимацию на условное обозначение:

$$\frac{1}{\Omega}\sum_f p_f^{k+1}\vec{S}_f \equiv \left(\nabla p^{k+1}\right)_P$$

Тогда расписав все аппроксимации, можно составить систему линейных алгебраических уравнений вида:

$$A_P\vec{V}_P^{k+1} + \sum_N A_N\vec{V}_N^{k+1} = \vec{R}_P^k - \left(\nabla p^{k+1}\right)_P, \tag{1.25}$$

где $\vec{R}_p^k$ обозначает все слагаемые, рассчитываемые по значениям с предыдущей итерации, A_P, A_N — коэффициенты при неизвестных величинах для ячейки P (иными словами, A_P — диагональный коэффициент матрицы СЛАУ, A_N — внедиагональные коэффициенты; выражение (1.25) записывается для каждой ячейки, для приграничных ячеек нужно также учесть соответствующие граничные условия).

Метод SIMPLE разбивается на два шага: предикторный и корректорный. Для этого неизвестные $\vec{V}^{k+1}$ и p^{k+1} ищем в виде суммы:

$$\vec{V}^{k+1} = \vec{V}^{*} + \vec{V}^{**}, \tag{1.26}$$

$$p^{k+1} = p^{k} + p^{*}.$$

Здесь $\vec{V}^{*}$ — предикторное значение скорости, $\vec{V}^{**}$ — корректорное (поправка скорости), p^k — значение давление с предыдущей итерации (предикторное), p^* — корректорное значение давления (поправка давления). Расщепим уравнение (1.25) на два:

$$A_P\vec{V}_P^{*} + \sum_N A_N\vec{V}_N^{*} = \vec{R}_P^k - \left(\nabla p^{k}\right)_P, \tag{1.27}$$

$$A_P \vec{V}_P^{**} + \sum_N A_N \vec{V}_N^{**} = -\left(\nabla p^*\right)_P . \tag{1.28}$$

Очевидно, что сумма этих уравнений даёт исходное уравнение (1.25). Здесь первое уравнение решается для нахождения предикторной скорости по известному с предыдущей итерации полю давления. Найденное из (1.27) поле скорости в общем случае не удовлетворяет уравнению неразрывности, т.е.

$$\sum_f \vec{V}_f^* \cdot \vec{S}_f \neq 0,$$

тогда как искомое поле на новой итерации $\vec{V}^{k+1}$ должно ему удовлетворять:

$$\sum_f \vec{V}_f^{k+1} \cdot \vec{S}_f = 0. \tag{1.29}$$

Подставим разложение (1.26) в (1.29), получим:

$$\sum_f \vec{V}_f^{**} \cdot \vec{S}_f = -\sum_f \vec{V}_f^* \cdot \vec{S}_f. \tag{1.30}$$

Эти два уравнения — уравнение для поправки скорости (1.28) и уравнение неразрывности (1.30) — представляют собой систему для отыскания неизвестных $\vec{V}^{**}$ и p^*, но решать их напрямую — это такая же задача, как и отыскание решения напрямую системы Навье-Стокса (1.25), (1.29), поскольку поправка давления не входит в (1.30). Для обхода этого затруднения в методе SIMPLE предлагается следующее: будем предполагать, что в итерационном процессе поправки скорости и давления на корректорном шаге довольно малы[1], поэтому в выражении (1.28) пренебрежём внедиагональными слагаемыми для $\vec{V}^{**}$, т.е. вместо (1.28) запишем:

$$A_P \vec{V}_P^{**} = -\left(\nabla p^*\right)_P . \tag{1.31}$$

Таким образом, появилась прямая связь между поправкой скорости в ячейке P и градиентом давления в этой ячейке. Подставляя это выражение в (1.30), получим:

[1] В общем случае это утверждение неверно, что приводит к необходимости вводить релаксационные параметры, см. далее

$$\sum_f \left(\frac{1}{A_P}\nabla p^*\right)_f \cdot \vec{S}_f = \sum_f \vec{V}_f^* \cdot \vec{S}_f, \tag{1.32}$$

т.е. дискретный аналог некоторого уравнения Пуассона для поправки давления. Подчеркнём, что это уравнение «искусственное», т.е. не соответствует никакому дифференциальному уравнению, поскольку коэффициент A_P (являющийся здесь как бы коэффициентом диффузии) определяется в т.ч. и параметрами сетки.

После решения (1.32), т.е. нахождения поправки давления, значение скорости в центре ячейке на новой итерации вычисляется по формуле:

$$\vec{V}_P^{k+1} = \vec{V}_P^* - \frac{1}{A_P}\left(\nabla p^*\right)_P. \tag{1.33}$$

Помимо скорости в центре, необходимо также для дальнейшей итерации знать расход на грани φ_f^{k+1}. И здесь важно обеспечить выполнение уравнения неразрывности для φ_f^{k+1}, поскольку именно эта величина входит в конвективную производную. Для этого вне зависимости от того, как вычисляется $\vec{V}_P^{k+1}$, следует вычислять расход как

$$\varphi_f^{k+1} = \vec{V}_f^* \cdot \vec{S}_f - \left(\frac{1}{A_P}\nabla p^*\right)_f \cdot \vec{S}_f, \tag{1.34}$$

что автоматически удовлетворяет уравнению неразрывности (1.22) с точностью решения уравнения (1.32). Это замечание связано с тем, что вычисление градиента от поправки давления в центре ячейки и в (1.32) зачастую производится по несогласованным формулам. В частности, в формуле (1.32) необходимо проводить аппроксимацию левой части по гибридной схеме с учётом поправки на неортогональность (в противном случае решение для давления на плохих сетках будет неверное и, возможно, приведёт к расходимости решения), тогда как для вычисления градиента давления в центре ячейки этого обычно не требуется.

Необходимо отметить, что представленный здесь алгоритм метода SIMPLE не пригоден для применения «в лоб» по двум причинам:

- матрица системы уравнений для предикторной скорости (1.27) не обладает свойством диагонального преобладания, т.е. $|A_P|$ не обязательно больше, чем $\sum_N |A_N|$; это приводит к проблемам при чис-

ленном решении СЛАУ, поскольку не гарантируется положительная определённость матрицы;

- поправка давления p^* не обязательно мала; это может приводить к слишком сильному изменению p^{k+1} от итерации к итерации, что в свою очередь ведёт к «разбалтыванию» поля $\vec{V}$ и расходимости итерационного процесса.

Для устранения указанных проблем вводятся параметры релаксации. Для обеспечения диагонального преобладания в матрице системы (1.27) необходимо увеличить диагональный коэффициент путём деления его на параметр релаксации $\alpha_V < 1$:

$$A_P^{'} = \frac{A_P}{\alpha_V} \tag{1.35}$$

и во всех формулах вместо A_p использовать модифицированный коэффициент $A_P^{'}$. При этом необходимо добавить слагаемое $(1-\alpha_V)A_p\vec{V}^k/\alpha_V$ для скорости, взятой с предыдущей итерации, в правую часть СЛАУ, чтобы обеспечить стремление решения модифицированной СЛАУ к решению исходной при $k \to \infty$.

Для того, чтобы нивелировать возможное сильное возмущение, вносимое поправкой давления, давление на новой итерации рассчитывается с релаксацией:

$$p^{k+1} = p_k + \alpha_p p^*.$$

Подчеркнём, что массовый поток необходимо рассчитывать без какой-либо релаксации по формуле (1.34), чтобы точно обеспечить выполнение уравнение неразрывности на новой итерации. Что же касается скорости в центре ячейке на новой итерации, то её также следует вычислять с релаксацией p^* (именно так реализовано в пакете OpenFOAM), однако, возможно построение алгоритма с вычислением $\vec{V}_P^{k+1}$ по формуле (1.33), где p^* берётся без релаксации.

Значения релаксационных параметров α_V, α_p подбираются для каждой задачи, но как показала многолетняя практика использования метода SIMPLE, во многих задачах гидродинамики (без массовых сил) наилучшую скорость сходимости обеспечивают следующие значения этих параметров:

$$\alpha_V = 0.7, \quad \alpha_p = 0.3.$$

Необходимость такой относительно сильной релаксации давления возникает из-за предположения о несущественном вкладе внедиагональных слагаемых в (1.28) и последующем их отбрасывании. Более разумным здесь выглядит замена в (1.28) внедиагональных $\vec{V}_N^{**}$ на $\vec{V}_P^{**}$ в предположении близости значений поправки скорости в текущей и соседних ячейках. Такой подход составляет основу метода SIMPLEC (SIMPLE-Consistent) и приводит с учётом релаксации вместо формулы (1.31) к формуле:

$$\left(A_P^{'} + \sum_N A_N \right) \vec{V}_P^{**} = - \left(\nabla p^* \right)_P . \tag{1.36}$$

Поскольку при дискретизации стационарных уравнений Навье-Стокса оказывается, что $A_P = -\sum_N A_N$, то с учётом (1.35) из (1.36) получаем следующее уравнение для поправки давления:

$$\frac{1-\alpha_V}{\alpha_V} A_P \vec{V}_P^{**} = - \left(\nabla p^* \right)_P ,$$

Более согласованная запись уравнений в методе SIMPLEC позволяет не вводить явную релаксацию давления, т.е. положить[1] $\alpha_p = 1$.

Поскольку в методе SIMPLE делается замена (1.28) на (1.31), то получаемое решение строго говоря не является решением (1.25). Понятно, что вводимая ошибка не существенна для стационарных задач (поскольку всё равно требуется делать внешние итерации по нелинейности), однако, делает метод SIMPLE неприменимым впрямую для нестационарных задач. В этом случае уточнение метода SIMPLE может быть осуществлено за счёт введения дополнительных итераций при определении поправки скорости и давления. Такой подход используется в методе PISO: после того, как найдена поправка давления и скорости, происходит уточнение формулы (1.28) путём вычисления следующей поправки к полю скорости $\vec{V}^{***}$ и давления p^{**} исходя из выражения:

$$A_P \vec{V}_P^{***} = - \left(\nabla p^{**} \right)_P - \sum_N A_N \vec{V}_N^{**} .$$

[1] При использовании метода SIMPLEC в ряде задач для улучшения сходимости также рекомендуется задавать $\alpha_p < 1$, например, $\alpha_p = 0.9$

Методы типа PISO используются в основном только для нестационарных задач, либо при использовании метода установления по времени (т.е. когда задача решается как нестационарная, но интересует только выход на установившееся стационарное решение), позволяя делать относительно большие временные шаги; подробнее см. в [8].

В качестве дополнения можно сделать замечание насчёт связи параметра релаксации α_V и числа Куранта. В «классическом» методе SIMPLE, описанном здесь, для улучшения диагонального преобладания используется релаксационный параметр. Иногда вместо этого параметра вводят так называемый шаг по псевдовремени $\Delta\tau$, и в уравнение баланса импульса добавляют производную от скорости по псевдовремени $\Delta\vec{V}/\Delta\tau$. Значение шага по псевдовремени выбирается в каждой ячейке своё исходя из условия на локальное число Куранта CFL и фон Неймана VNN. Введение производной по псведовремени по сути просто увеличивает диагональный коэффициент следующим образом:

$$A_P^{'} = A_P + \frac{1}{\Delta\tau}.$$

При этом оказывается, что в случае CFL=VNN и при определённом способе вычисления этих чисел справедливо равенство

$$\mathrm{CFL} = \frac{\alpha_V}{1-\alpha_V},$$

т.е. использование параметра релаксации эквивалентно введению производной по псевдовремени.

Необходимо сделать ещё одно важное замечание. Дело в том, что зачастую численный алгоритм решения уравнения Навье-Стокса на совмещённых сетках приводит к появлению так называемых чётно-нечётных осцилляций давления (на рисунках распределения давления это выглядит как зигзагообразные линии уровня). Здесь мы не будем сильно углубляться в детали природы возникновения этих осцилляций, дадим лишь общее представление.

В уравнении баланса импульса градиент давления берётся из центра ячейки, что, например, для равномерной двумерной прямоугольной сетки даёт центрально-разностную аппроксимацию производной, аналогичную приведённой в формуле (1.14). Поэтому и монотонная функция для

давления, и осциллирующая через точку оказывают одинаковое влияние на скорость. Для подавления возникновения этих нефизических осцилляций давления обычно используют поправку Рхи-Чоу [10], которая вводится явным или неявным (как сделано в OpenFOAM) образом при помощи добавления к расходу на грани умноженной на некий коэффициент разности двух градиентов давления: один считается по разности значений давления слева и справа от грани, а второй — как линейная интерполяция градиентов давления, посчитанных в центрах смежных ячеек. Вводимая таким образом добавка имеет второй порядок точности и пропорциональна третьей производной от давления по координате, поэтому в случае относительно гладких распределений давления практически не влияет на решение, но активно подавляет возникающие в ходе решения чётно-нечётные осцилляции (подробнее см. в [8, 13]).

2. Общее описание пакета OpenFOAM в применении к задачам механики сплошных сред

2.1. Введение

OpenFOAM — это открытая интегрируемая платформа для численного решения уравнений в частных производных методом конечного объема. Изначально OpenFOAM создавался как инструмент для решения задач вычислительной гидродинамики (Computational Fluid Dynamics, CFD), однако с течением времени благодаря гибкости используемой платформы и открытости исходного кода в нём появилась возможность решать множество других физических задач, которые могут быть описаны в рамках скалярных, векторных и тензорных полей и операций над ними, или (в некоторых случаях) обыкновенными дифференциальными уравнениями.

Основное преимущество OpenFOAM перед коммерческими программными продуктами, предназначенными для решения задач CFD (такими, как ANSYS Fluent, CFX, STAR-CCM+ и др.), является его открытость и доступность: пакет OpenFOAM распространяется под свободной лицензией GNU GPLv3[1] вместе с исходными кодами, что даёт возможность модифицировать код, если это необходимо для тех или иных задач. Существенным недостатком пакета, препятствующим его широкому распространению, является отсутствие пользовательской графической оболочки (GUI — Graphical User Interface), вся работа с пакетом осуществляется при помощи редактирования управляющих файлов, а также при помощи сторонних программ (например, для визуализации результатов расчёта используется пакет ParaView); имеются коммерческие графические оболочки для взаимодействия с OpenFOAM.

[1] GNU GPL — универсальная общественная лицензия Фонда свободного программного обеспечения, подробнее см. http://www.gnu.org/licenses/gpl.html

В настоящее время пакет OpenFOAM поддерживается, развивается и распространяется некоммерческой организацией The OpenFOAM Foundation[1]. Несмотря на то, что пакет OpenFOAM ориентирован в первую очередь на решение научных и исследовательских задач, он также используется для инженерных расчётов различными промышленными организациями. Пакет OpenFOAM активно развивается; ежегодно для всех желающих проводится симпозиум, где проводятся учебные курсы по OpenFOAM, представляются новые возможности и происходит обмен опытом между исследователями, использующими этот пакет для своих задач[2].

Структурно пакет OpenFOAM представляет из себя набор библиотек, позволяющих написать собственные программы для решения уравнений в частных производных (на языке C++), и множество готовых решателей и утилит. В основном представленные в OpenFOAM решатели предназначены для численного моделирования различных задач механики сплошных сред, однако, благодаря модульной структуре пакета, возможно использовать его и для других, не связанных с механикой сплошных сред, задач (в качестве примера здесь можно упомянуть входящий в состав пакета решатель для финансовых расчётов по модели ценообразования опционов Блэка-Шоулза).

Стоит также отметить, что поскольку пакет OpenFOAM распространяется совершенно свободно, существует несколько так называемых «форков»[3], поддерживаемых сообществом заинтересованных людей/организаций, одним из которых является проект The OpenFOAM® Extend Project[4], аккумулирующий сторонние свободно-распространяемые разработки на основе кодовой базы OpenFOAM, по тем или иным причинам не вошедшие в «официальную» версию. Список известных «форков» и вариантов OpenFOAM можно также найти на сайте https://openfoamwiki.net/index.php/Forks_and_Variants.

Что касается названия пакета, то FOAM является аббревиатурой от Field Operation And Manipulation (операции и манипуляции с полями) и одновременно представляет собой английское слово foam (пена, море). Таким

[1] Подробнее см. http://www.openfoam.org
[2] См. http://www.openfoamworkshop.org
[3] От англ. fork (ответвление) — программа, созданная на основе некоторой другой
[4] См. http://www.extend-project.de

образом, название FOAM отражает основную суть пакета — реализация численной аппроксимации дифференциальных и алгебраических операций над скалярными, векторными и тензорными полями. Приставка Open означает, что пакет распространяется свободно, его исходные коды открыты для чтения и модификации.[1]

2.2. Установка и структура OpenFOAM

Установка пакета OpenFOAM в настоящее время не представляет особых сложностей, а детальные инструкции приведены на сайте http://openfoam.org/download. Однако, есть важное замечание: официальный дистрибутив поставляется только для операционных систем из семейства GNU/Linux.[2] Данное учебное пособие ориентировано на работу в ОС GNU/Linux, поскольку именно такие ОС установлены на подавляющем большинстве суперкомпьютеров/кластеров, и в среде высокопроизводительных вычислений являются стандартом де-факто. Однако, для учебных целей одним из решений проблемы установки дистрибутива OpenFOAM может являться использование виртуальных машин, таких, как VirtualBox.[3] В этом случае необходимо установить виртуальную машину, затем в ней установить любой предпочитаемый дистрибутив GNU/Linux[4], и далее переходить к следующему разделу данного пособия.

2.2.1. Установка OpenFOAM в ОС на базе ядра Linux

Пакет OpenFOAM может быть установлен несколькими способами. Для ОС семейства Ubuntu предоставляется уже скомпилированные бинарные пакеты, поэтому процесс установки здесь совсем несложный и довольно быстрый.[5] Однако, поскольку в настоящем пособии будут рассматривать-

[1] Изначально был разработан закрытый коммерческий пакет Foam, на основе которого в 2004 году его разработчиками был выпущен уже открытый пакет OpenFOAM

[2] Имеются неофициальные сборки для других ОС, но в данном руководстве они не рассматриваются

[3] Распространяется под свободной лицензией GNU GPL v2, см. https://www.virtualbox.org

[4] Авторы рекомендуют использовать дистрибутив Ubuntu (или его вариации) как наиболее дружественный к неискушённому пользователю, см. http://ubuntu.ru или http://www.ubuntu.com

[5] Установка версии 7 для Ubuntu описана здесь: https://openfoam.org/download/7-ubuntu/

ся вопросы использования библиотек OpenFOAM и создания собственных решателей, целесообразней использовать установку из исходников, включающую в себя компиляцию всех библиотек OpenFOAM, что занимает довольно продолжительное время (кроме того, установка из исходников не требует наличия прав root, о чём подробнее будет сказано далее).

Для установки из исходников предлагается два варианта: установка последнего выпуска OpenFOAM[1] путём скачивания исходников с сайта, либо установка из Git-репозитория. Последний вариант предпочтителен в том случае, если предполагается активная работа с пакетом OpenFOAM и постоянное обновление. Принципиально эти два способа не отличаются, поэтому далее процесс установки будет описан одновременно для них обоих.

Прежде чем перейти к собственно установке, необходимо сделать несколько замечаний об организации работы в ОС на базе GNU/Linux. ОС этого семейства являются многопользовательскими, поэтому каждый пользователь должен иметь свой логин (например, user) и домашний каталог (располагающийся на диске по адресу `/home/user`). Однако, существует один суперпользователь с именем root, который имеет доступ к любому компоненту и каталогу ОС[2]. При установке бинарного дистрибутива OpenFOAM необходимо иметь права суперпользователя root, а сам дистрибутив обычно устанавливается в каталог `/opt`, и таким образом, становится доступным для всех других пользователей. Установку OpenFOAM из исходников также можно осуществлять как для всех пользователей, так и локально в свой домашний каталог. Первый способ требует получения прав root и может быть потенциально опасным для системы, если осуществляется неопытным пользователем (ну и кроме того, для этого требуется получить root-права, что не всегда возможно, если речь идёт, например, о стороннем кластере). Поэтому в данном пособии будет описан способ установки в домашний каталог пользователя, который (для определённости) будет иметь логин user.

[1] На момент написания данного пособия доступна последняя версия OpenFOAM 7 от 08.07.2019

[2] Во многих ОС, например Ubuntu, учётная запись пользователя root по умолчанию отключена, а права root могут быть получены при помощи команды sudo

Сначала необходимо создать рабочий каталог для установки OpenFOAM, для этого необходимо открыть терминал и выполнить команду:

```
mkdir $HOME/OpenFOAM
```

В приведённой выше команде использована переменная окружения $HOME, в которой хранится путь к домашнему каталогу пользователя, и данная команда полностью аналогична следующей для пользователя user:

```
mkdir /home/user/OpenFOAM
```

Далее необходимо перейти в созданный каталог и скачать исходники дистрибутива OpenFOAM. Переход в каталог:

```
cd $HOME/OpenFOAM
```

Установка OpenFOAM версии 7 из исходников подробно описана на странице https://openfoam.org/download/7-source/ (или на странице http://openfoam.org/download/source для установки из git-репозитория). Необходимо сразу дать некоторые пояснения: дистрибутив OpenFOAM распространяется в виде двух пакетов: собственно OpenFOAM и так-называемые сторонние приложения ThirdParty. В набор сторонних приложений входят исходные коды различных вспомогательных свободных программ, в частности, библиотеки Scotch (для параллельных вычислений) и пакета ParaView. ParaView представляет собой мощный инструмент для визуализации и проведения различных операций над данными (подробнее о ней будет рассказано далее). По умолчанию она не компилируется, а процесс её сборки занимает довольно значительное время. Тем не менее, ParaView следует установить, либо путём компиляции из пакета ThirdParty, либо из бинарных пакетов, доступных по адресу http://www.paraview.org/download.

Для скачивания и распаковки файлов можно воспользоваться командой wget, для OpenFOAM 7:

```
wget -O - http://download.openfoam.org/source/7 | tar xvz
```

Для ThirdParty:

```
wget -O - http://download.openfoam.org/third-party/7 | tar xvz
```

После этого следует переименовать каталоги:

```
mv OpenFOAM-7-version-7 OpenFOAM-7
mv ThirdParty-7-version-7 ThirdParty-7
```

Из git-репозитория можно установить как мажорную версию с исправлениями (например, для версии 7 соответствующий репозиторий будет называться OpenFOAM-7), так и текущую разрабатываемую версию OpenFOAM-dev[1]. Для установки из git-репозитория необходимо выполнить следующую команду (программа git должна быть установлена в используемой ОС):

```
git clone git://github.com/OpenFOAM/OpenFOAM-7.git
```

При установке из git-репозитория необходимо также скачать текущую версию пакета ThirdParty:

```
git clone git://github.com/OpenFOAM/ThirdParty-7.git
```

Для того, чтобы скомпилировать библиотеки и программы OpenFOAM (как, впрочем, и другие программы), может потребоваться доустановить в ОС некоторые системные пакеты (компиляторы и утилиты). Для ОС на базе Ubuntu минимально необходимым набором является пакет build-essential, установить который можно командой:

```
sudo apt-get install build-essential
```

Также может потребоваться установка других программ и библиотек, например, flex, bison, boost, CGAL и некоторых других[2]. Кроме того, для компиляции и работы OpenFOAM необходимо наличие библиотеки MPI (Message Passing Interface — библиотека для обмена сообщений между параллельными процессами). В большинстве дистрибутивов ОС на базе Linux имеется библиотека openmpi, установить которую в Ubuntu можно выполнив команду

```
sudo apt-get install libopenmpi-dev
```

[1] Подробности см. http://openfoam.org/download/source

[2] Конкретный набор требуемых пакетов может зависеть от ОС, версии OpenFOAM и др. Подробнее см. http://openfoam.org/download/source/software-for-compilation

Перед тем, как начать собственно процесс сборки OpenFOAM, необходимо выставить так называемые переменные окружения (переменные среды) — текстовые переменные ОС, хранящие какую-либо информацию (например, данные о настройках системы). Для этого в текущей терминальной сессии достаточно выполнить команду

```
source $HOME/OpenFOAM/OpenFOAM-7/etc/bashrc
```

Чтобы переменные окружения, требующиеся для работы OpenFOAM, устанавливались автоматически при запуске ОС и терминала, необходимо прописать указанную выше строчку в файл `$HOME/.bashrc` (отметим, что этот файл, как и другие с именами, начинающимися на «.», является скрытым, поэтому для его отображения в списке файлов при использовании файлового менеджера необходимо включить опцию «Показать скрытые файлы»).

Переменные окружения необходимы, в частности, для того, чтобы скомпилировать библиотеки и программы OpenFOAM, а также для осуществления запуска этих программ. Одна из основных переменных — `WM_PROJECT_DIR` — содержит путь к каталогу OpenFOAM-7[1], переменная `WM_PROJECT_USER_DIR` содержит путь, где предполагается компилировать пользовательские программы и библиотеки, запускать расчёты. По умолчанию `WM_PROJECT_DIR` устанавливается равной `/home/user/OpenFOAM/OpenFOAM-7`, а `WM_PROJECT_USER_DIR` — `/home/user/OpenFOAM/user-7`. Если необходимо переопределить стандартные пути, следует соответствующим образом отредактировать файл `OpenFOAM-7/etc/bashrc`.

После того, как переменные окружения установлены, можно переходить к процессу компиляции OpenFOAM. Для этого достаточно перейти в каталог `$WM_PROJECT_DIR` и запустить команду Allwmake[2]:

```
cd $WM_PROJECT_DIR
./Allwmake
```

Стоит отметить, что OpenFOAM использует собственную систему сборки wmake, предназначенной для автоматической генерации файлов сборки

[1] Число после дефиса означает номер версии. Отметим, что одновременно может быть установлено несколько разных версий OpenFOAM

[2] Для ускорения процесса компиляции можно запускать команду Allwmake в параллельном режиме при помощи ключа -jN, где N — количество процессов

Makefile стандартной утилиты make. Процесс сборки OpenFOAM полностью автоматизирован, поэтому участие пользователя в нём не предполагается (если нет ошибок, которые в основном связаны с отсутствием тех или иных системных программ и библиотек); сам процесс занимает довольно длительное время (несколько часов).

Для компиляции пакета ParaView используются следующие команды (возможно, потребуется доустановить в систему некоторые библиотеки, например, Qt):

```
cd $WM_THIRD_PARTY_DIR
./makeParaView
```

2.2.2. Структура пакета: основные каталоги

После компиляции (или установки другим способом) пакет OpenFOAM готов к использованию. Поскольку графическая оболочка отсутствует, вся структура пакета по сути представлена в виде каталогов и файлов. Поэтому необходимо более детально ознакомиться с основными каталогами, располагающимися по адресу `$WM_PROJECT_DIR`. Их список представлен на рисунке 2.1. В корневом каталоге `OpenFOAM-7` находятся файлы COPYING и README, представляющие собой текст лицензии и краткое описание пакета, а также файл `Allwmake` — файл сценария для сборки всего пакета. В остальных каталогах находятся исходные тексты программ и библиотек, исполняемые файлы, примеры и многое другое.

Исходные коды всех библиотек OpenFOAM располагаются в каталоге `src`[1]. В каталоге `applications` находятся исходные коды всех решателей, утилит и некоторых тестовых программ. Если не предполагается что-либо менять в коде OpenFOAM, то пользователю нет необходимости детально изучать содержимое этих каталогов. Однако, в некоторых случаях (когда документация и/или другие источники информации не даёт ответа на какой-либо вопрос), имеет смысл изучать исходники, поскольку в них зачастую присутствуют полезные комментарии о тонкостях реализации физической и математической моделей.

[1] Здесь и далее, если не указан полный путь, подразумевается путь от корневого каталога `$WM_PROJECT_DIR`

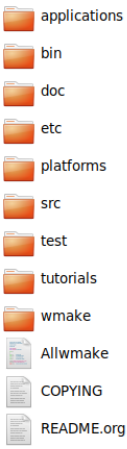

Рис. 2.1. Содержимое корневого каталога OpenFOAM

Все бинарные файлы — исполняемые файлы и разделяемые библиотеки, скомпилированные в процессе сборки, — располагаются в каталоге `platforms`. Также внутри этого каталога располагаются объектные файлы, создаваемые в процессе компиляции. Отметим, что по умолчанию OpenFOAM компилируется в режиме оптимизации и двойной точности, поэтому в каталоге `platforms` присутствует только каталог `linux64GccDPInt32Opt` (при компиляции на 64-битных ОС). Если указать другие опции сборки, то её результат будет помещён уже в каталог с соответствующим названием (например, при компиляции с одинарной точностью создаётся каталог `linux64GccSPInt32Opt` и т.п.).

В каталоге `doc` располагается документация, в частности, `OpenFOAMUserGuide` (руководство пользователя), отмеченное в списке литературы как [14]. Кроме того, здесь находятся управляющие файлы для системы документирования исходного кода Doxygen.

В каталоге `tutorials` собраны все примеры расчётов, которые могут служить для знакомства и освоения OpenFOAM, а также для создания собственных постановок задач на основе какого-либо близкого по постановке примера. В основном, представленные примеры довольно синтетические и носят чисто демонстрационный характер, хотя есть и расчёты, основанные на эксперименте и/или реально используемой геометрии. Каталог `test` содержит некоторые примеры для тестирования кода OpenFOAM.

Каталог `bin` содержит различные вспомогательные файлы сценариев для автоматизации некоторых часто используемых действий. Например, все исполняемые файлы для подготовки и запуска расчёта в примерах (tutorials) написаны с использованием этих вспомогательных сценариев.

Настройка параметров компиляции, работы собственно самого пакета OpenFOAM и другие файлы настроек содержатся в каталоге `etc`. Переменные окружения и параметры компиляции устанавливаются в файле `etc/`

bashrc; параметры для работы OpenFOAM — в файле etc/controlDict. Кроме того, здесь находятся шаблоны различных файлов, используемых при проведении расчётов.

Последний каталог — wmake — содержит файлы для системы сборки и компиляции wmake. Здесь же (в каталоге wmake/rules) можно найти и отредактировать параметры компиляции. Например, для указания дополнительных флагов процессора при компиляции оптимизированного кода следует соответствующим образом отредактировать файл wmake/rules/linux64Gcc/c++Opt.

2.2.3. Обзор основных решателей OpenFOAM

Как отмечалось ранее, OpenFOAM — это в первую очередь набор библиотек для численного моделирования задач механики сплошной среды. Однако, он также представляет большой набор решателей, предназначенных для широкого круга задач. Все решатели[1] находятся в каталоге applications/solvers и разделены по категориям. Список всех категорий и их краткое описание приведено в таблице 2.1.

Стоит добавить, что некоторые решатели имеют две версии: обычную и версию для расчёта на движущихся сетках. В последнем случае в имени решателя появляется часть DyM (Dynamic Mesh). Например, один из решателей для задач ламинарного или турбулентного движения несжимаемой жидкости называется pimpleFoam, а версия этого же решателя с учётом движения сетки — pimpleDyMFoam. Кроме того, благодаря использованию библиотек многие решатели дают возможность выбора той или иной физической модели. Например, упомянутый решатель pimpleFoam позволяет рассчитывать ламинарное движение жидкости, а также турбулентное при помощи какой-либо модели турбулентности, которая задаётся пользователем при постановке расчёта; библиотека OpenFOAM содержит большое число полуэмпирических и гибридных моделей турбулентности, о чём далее будет говориться более подробно.

[1] Точнее, их исходные коды

Таблица 2.1. Категории стандартных решателей OpenFOAM

Категория (подкаталог в applications/solvers)	Описание
DNS	Специальный решатель для прямого численного моделирования изотропной турбулентности
basic	Решатели для простейших уравнений (уравнение Лапласа, перенос скаляра в заданном поле скорости и др.)
combustion	Набор решателей для задач химической кинетики (в т.ч. горения) и расчёта распространения пламени
compressible	Решатели для сжимаемых сред (дозвуковое и трансзвуковое течение газа)
discreteMethods	Решатели, оперирующие с отдельными частицами (молекулярная динамика, метод Монте-Карло расчёта течений разреженных газов)
electromagnetics	Решатели для электростатики и магнитной гидродинамики
financial	Решатель для финансовых расчётов
heatTransfer	Решатели для задач теплообмена, включая свободную конвекцию в поле силы тяжести, сопряжённый теплообмен
incompressible	Решатели для задач ламинарного и турбулентного движения вязкой жидкости
lagrangian	Решатели для задач течения жидкости и газа при наличии дискретной фазы (лагранжево-эйлеровское описание движения многофазных сред)
multiphase	Решатели для движения многофазных сред (метод объёма жидкости Volume of Fluid и двухжидкостное приближение)
stressAnalysis	Решатели для задач механики деформируемого твёрдого тела

2.3. Пример постановки расчёта в OpenFOAM (задача нестационарной теплопроводности)

Для того, чтобы познакомиться с процессом постановки и запуска расчёта в среде OpenFOAM (а также с визуализацией результатов), рассмотрим одну из наиболее простых задач о нестационарной теплопроводности в твёрдом теле. Данная задача входит в стандартный набор примеров OpenFOAM для одного из простейших решателей laplacianFoam (см. `$WM_PROJECT_DIR/tutorials/basic/laplacianFoam/flange`; код решателя laplacianFoam можно найти в каталоге `$WM_PROJECT_DIR/applications/solvers/basic/laplacianFoam`).

Хорошей практикой работы в OpenFOAM является использование стандартного каталога пользователя, определяемого переменной окружения `$FOAM_RUN`. Для создания этого каталога достаточно выполнить команду

```
mkdir -p $FOAM_RUN
```

Для удобства пользователя, в OpenFOAM определены некоторые команды для навигации по каталогам (список которых можно посмотреть командой alias). Для перехода в созданный пользовательский каталог можно выполнить команду **run** или

```
cd $FOAM_RUN
```

Для работы с примером следует скопировать его в пользовательский каталог:

```
cp -r $WM_PROJECT_DIR/tutorials/basic/laplacianFoam/flange $FOAM_RUN
```

2.3.1. Описание задачи

Данная задача носит чисто демонстрационный характер и представляет собой расчёт поля температуры в однородном фланце методом установления во времени. Геометрия и сетка на поверхности представлены на рисунке 2.2. Решается уравнение нестационарной теплопроводности с постоянными физическими свойствами:

$$\frac{\partial T}{\partial t} - D_T \nabla^2 T = 0,$$

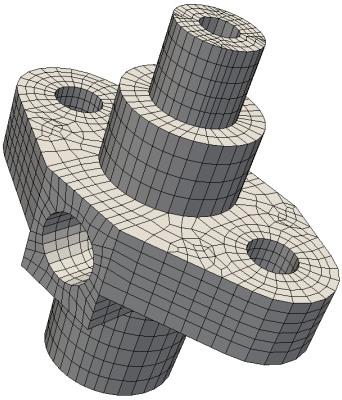

Рис. 2.2. Геометрия и расчётная сетка на поверхности фланца

где под D_T понимается коэффициент температуропроводности, вычисляемый через коэффициент теплопроводности λ, теплоёмкость c и плотность ρ по формуле $D_T = \lambda/(\rho c)$.

В начальный момент времени во всей расчётной области задаётся температура 273K, часть поверхности находится при постоянной температуре 273K, другая часть — при температуре 573K, оставшаяся часть — адиабатическая (какие именно части поверхности находятся при тех или иных условиях, в данном примере несущественно, о визуализации этих границ будет рассказано ниже).

Для любого расчёта в OpenFOAM все данные для него должны быть записаны в специальные файлы, которые в свою очередь располагаются в следующих каталогах:

- Каталог `0` (или другое число) — файлы с полями всех переменных, участвующих в расчёте, в начальный момент времени.
- Каталог `constant` — файлы с расчётной сеткой и физическими параметрами задачи.
- Каталог `system` — файлы с параметрами численных схем и параметрами для управления расчётом.

В рассматриваемом примере рабочий каталог задачи имеет имя `flange` и содержит перечисленные выше каталоги, а также некоторые вспомогательные файлы: файл `flange.ans` с геометрией и сеткой в формате ANSYS; скрипты для запуска расчёта `Allrun` и очистки каталогов `Allclean`. Для выполнения расчёта достаточно запустить соответствующий скрипт следующей командой[1]

[1] В ОС на базе Linux запуск команды из текущего каталога необходимо предварять символами «./», поскольку в противном случае файл будет искаться только в каталогах их переменной окружения `$PATH`, в которой обычно текущий каталог не прописан

```
./Allrun
```

Этот скрипт производит конвертацию файла с сеткой в формат OpenFOAM, запускает расчёт и производит некоторые дополнительные действия после его окончания. Отметим, что в процессе расчёта создаются каталоги с именами в виде чисел (0.1, 0.2 и т.д.) — эти каталоги содержат файлы с полями в соответствующий момент времени.

2.3.2. Задание начальных и граничных условий

В каталоге 0 содержатся файлы с полями и граничными условиями в начальный момент времени. В рассматриваемом примере здесь находится только один файл с именем «T». Это имя указывает, что содержащиеся в нём данные относятся к полю температуры T (для которого и решается указанное выше уравнение теплопроводности). Содержимое данного файла представлено ниже.

```
Файл $WM_PROJECT_DIR/tutorials/basic/laplacianFoam/flange/0/T:
/*--------------------------------*- C++ -*----------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     | Website:  https://openfoam.org
    \\  /    A nd           | Version:  7
     \\/     M anipulation  |
\*---------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    object      T;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
dimensions      [0 0 0 1 0 0 0];
internalField   uniform 273;
boundaryField
{
    patch1
    {
        type            zeroGradient;
```

```
    }
    patch2
    {
        type            fixedValue;
        value           uniform 273;
    }
    patch3
    {
        type            zeroGradient;
    }
    patch4
    {
        type            fixedValue;
        value           uniform 573;
    }
}
// ************************************************************************* //
```

Как видно из представленного листинга, формат записи файла соответствует стандарту записи файлов с исходными текстами в языке программирования C++. Строки, начинающиеся с символов «//», являются комментариями и могут быть опущены. Многострочные комментарии разделяются признаком начала «/*» и признаком окончания «*/» (могут быть опущены). Смысловые строки также оформлены с использованием синтаксиса C++: в конце выражения должен стоять символ «;», блоки выделены символами «{» и «}». Рассмотрим подробнее синтаксис данного файла (отметим, что все файлы с данными для расчёта в OpenFOAM имеют подобный синтаксис).

Таким образом, первая важная часть, которая должна присутствовать во всех файлах данных OpenFOAM, начинается с ключевого слова FoamFile — в этом блоке описывается формат файла и объект, для которого этот файл используется. Детальное описание этого блока пока опускаем.

Следующая важная часть — ключевое слово dimensions. В квадратных скобках далее идёт указание размерности полевой величины, описываемой в рассматриваемом файле `T`. Каждое число в квадратных скобках является степенью соответствующей по порядку единицы измерения из списка: масса (кг), длина (м), время (с), температура (К), количество вещества (моль), сила тока (А), интенсивность освещения (Кд), т.е. `[kg m s K mol A Kd]`; в

данном примере размерность величины T — Кельвины (более подробно см. раздел 6.1.3).

Ключевое слово internalField указывает на блок с данными для всех внутренних ячеек: «uniform 273» означает, что во всех внутренних ячейках расчётной области задаётся постоянное значение $T = 273$.

Далее идёт блок описания граничных условий boundaryField. Каждый следующий под-блок начинается с имени граничной зоны (все зоны должны быть определены на этапе подготовки сетки, каждая зона представляет собой какую-либо часть поверхности расчётной сетки). Для каждой такой зоны необходимо задать тип граничного условия (ключевое слово type) и некоторые другие параметры в зависимости от этого типа. В рассматриваемом примере зоны patch1 и patch3 имеют тип zeroGradient, что означает условие нулевой производной температуры на этих поверхностях (условие адиабатичности). Зоны patch2 и patch4 имеют тип fixedValue, что означает фиксированное (во времени) значение температуры (условие первого рода). Ключевое слово value указывает на значение температуры: для patch2 это 273, а для patch4 — 573 (слово uniform говорит о том, что на всех поверхностных ячейках зоны задаётся одно и тоже значение).

2.3.3. Задание управляющих параметров расчёта

Для задания физических свойств используются управляющие файлы в каталоге `costant`. В рассматриваемом примере необходимо задать значение коэффициента температуропроводности D_T в файле `constant/transportProperties`. Содержимое этого файла представлено ниже.

```
Файл flange/constant/transportProperties:
/*--------------------------------*- C++ -*----------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     | Website:  https://openfoam.org
    \\  /    A nd           | Version:  7
     \\/     M anipulation  |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
```

```
    class       dictionary;
    location    "constant";
    object      transportProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
DT              DT [0 2 -1 0 0 0 0] 4e-05;
// ************************************************************************* //
```

Как видно, «шапка» этого файла такая же, как и для файла граничных и начальных условий. Различия начинаются в блоке FoamFile: класс описываемого в данном файле «объёкта» — это dictionary (словарь). В разделе 6.2 работа с такими объектами будет описана более подробно, пока отметим, что все файлы с параметрами имеют класс «словарь».

Собственно коэффициент температуропроводности определяется в данном файле в строке, начинающейся с «DT» (последующее повторение «DT» в данном случае может быть опущено). DT является именем используемого внутри кода параметра (может быть и другим — это зависит от используемого решателя; в рассматриваемом примере используется решатель laplacianFoam, для которого определён один физический параметр D_T). Далее в строке идёт определение размерности параметра DT (в данном случае это $м^2/с$) и его значение $(4 \cdot 10^{-5})$.

Параметры решателя (численные схемы и управляющие параметры) задаются в файлах, расположенных в каталоге `system`. Здесь мы не будем подробно останавливаться на разборе каждого файла (за некоторым исключением), отметим лишь предназначение каждого из них.

- `system/fvSchemes` — здесь описываются параметры численной схемы, а именно: схемы дискретизации по пространству и времени, вычисления градиентов и т.п.
- `system/fvSolution` — здесь задаются некоторые опции используемых численных методов, в частности, можно выбрать конкретный линейный решатель (т.е. метод решения системы линейных алгебраических уравнений — СЛАУ), критерии сходимости и т.п.
- `system/controlDict` — файл управляющих параметров расчёта.

Немного подробнее рассмотрим некоторые параметры в файле `system/controlDict`. Содержимое этого файла приведено ниже.

Файл flange/system/controlDict:

```
/*--------------------------------*- C++ -*----------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     | Website:  https://openfoam.org
    \\  /    A nd           | Version:  7
     \\/     M anipulation  |
\*---------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      controlDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
application     laplacianFoam;
startFrom       latestTime;
startTime       0;
stopAt          endTime;
endTime         3;
deltaT          0.005;
writeControl    runTime;
writeInterval   0.1;
purgeWrite      0;
writeFormat     ascii;
writePrecision  6;
writeCompression off;
timeFormat      general;
timePrecision   6;
runTimeModifiable true;
// ********************************************************************* //
```

Одним из основных интересующих в данный момент параметров является startFrom. Этот параметр указывает, с какого момента времени следует начинать расчёт. Значение latestTime говорит о том, что надо начинать с последнего имеющегося в данном каталоге момента времени. В исходном варианте это каталог «0» — с него и будет начинаться расчёт. Другое возможное значение параметра startFrom — startTime. Тогда расчёт будет вестись от момента времени, указанного в параметре startTime, находя-

щемся на следующей строке файла. Окончание расчёта происходит при достижении момента времени endTime (если значение параметра stopAt установлено как endTime; можно использовать и другие опции остановки расчёта, о чём более подробно будет рассказано далее).

Другим важным параметром является deltaT — это значение шага по времени. Отметим, что в рассматриваемом примере шаг по времени постоянен (и равен 0.005), однако, можно указать также и переменный шаг по времени, который зависит от числа Куранта.

Вывод полей осуществляется через интервал времени (физического, а не реального), указанного в writeInterval. Можно также указать, чтобы вывод полей производился другим образом — за это отвечает параметр writeControl (значение runTime указывает на использование значения writeInterval). Отметим, что расчёт можно начинать с любого момента времени — формат выводных файлов идентичен описанному выше в разделе 2.3.2.

2.3.4. Запуск расчёта и мониторинг хода решения

В рассматриваемом примере расчётная сетка импортируется из файла `flange.ans` при помощи утилиты OpenFOAM ansysToFoam путём выполнения команды (опция scale служит для масштабирования значений координат узлов сетки):

```
ansysToFoam flange.ans -scale 0.001
```

Расчётная сетка в формате OpenFOAM записывается в каталоге `constant/polyMesh`. Более подробно о расчётных сетках будет рассказываться в соответствующем разделе. Запуск расчёта осуществляется вызовом соответствующего решателя. В данном случае это

```
laplacianFoam
```

После запуска решателя на экран терминала выводится информация о запуске и ходе решения, примерно в таком виде, как представлено ниже.

```
/*---------------------------------------------------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
```

```
   \\    /   O peration     | Website:  https://openfoam.org
    \\  /    A nd           | Version:  7
     \\/     M anipulation  |
\*---------------------------------------------------------------------------*/
Build  : 7-6ef561967074
Exec   : laplacianFoam
Date   : Sep 04 2019
Time   : 09:40:49
Host   : "baby11"
PID    : 2296
I/O    : uncollated
Case   : /home/aero/OpenFOAM/aero-7/run/flange
nProcs : 1
sigFpe : Enabling floating point exception trapping (FOAM_SIGFPE).
fileModificationChecking : Monitoring run-time modified files using
    ↪ timeStampMaster (fileModificationSkew 10)
allowSystemOperations : Allowing user-supplied system call operations
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
    ↪  * //
Create time
Create mesh for time = 0
SIMPLE: No convergence criteria found
Reading field T
Reading transportProperties
Reading diffusivity DT
No finite volume options present
Calculating temperature distribution

Time = 0.005

DICPCG:  Solving for T, Initial residual = 1, Final residual = 1.65094e
    ↪ -07, No Iterations 5
DICPCG:  Solving for T, Initial residual = 0.00446913, Final residual =
    ↪ 2.58563e-07, No Iterations 3
DICPCG:  Solving for T, Initial residual = 0.000148071, Final residual =
    ↪ 1.6611e-07, No Iterations 2
ExecutionTime = 0.02 s  ClockTime = 0 s

Time = 0.01

DICPCG:  Solving for T, Initial residual = 0.203755, Final residual =
    ↪ 3.79207e-07, No Iterations 4
```

```
DICPCG:  Solving for T, Initial residual = 0.00184335, Final residual =
    ↪ 1.15759e-07, No Iterations 3
DICPCG:  Solving for T, Initial residual = 5.70567e-05, Final residual =
    ↪ 6.6188e-08, No Iterations 2
ExecutionTime = 0.02 s  ClockTime = 0 s
```

После стандартной «шапки» идёт служебная информация о запуске: версия OpenFOAM (Build), решатель (Exec), время запуска (Date и Time), на каком компьютере (Host; выводится имя компьютера), идентификатор процесса (PID), каталог, откуда произведён запуск (Case), количество процессов (nProcs; в данном случае программа запущена в однопроцессном режиме) и др. Начиная со строки «Create time» выводятся собственно результаты работы решателя (для каждого конкретного решателя вывод может немного отличаться). После этапа подготовки расчёта (который заканчивается строкой «Calculating temperature distribution»), начинается основный цикл по времени. Каждый шаг по времени начинается с строки «Time =», где указывается текущее (физическое) время задачи. На каждом шаге по времени выполняется решение дискретизированного уравнения нестационарной теплопроводности. Каждая строка, начинающаяся с «DICPCG:», соответствует решению СЛАУ для температуры T на данном шаге по времени. Количество этих строк связано с параметром nNonOrthogonalCorrectors в файле `system/fvSolution` (значение которого равно 2) и отражает необходимость нескольких дополнительных итераций для учёта поправки на скошенность ячеек (то есть первая итерация выполняется с учётом поправки на скошенность, где градиенты величины T берутся с предыдущего шага по времени, а в оставшихся двух градиент считается от T с предыдущей итерации).

Подробнее рассмотрим строки, начинающиеся с «DICPCG:». Аббревиатура DICPCG означает, что решается СЛАУ по методу предобусловленных сопряжённых градиентов (PCG — Preconditioned Conjugate Gradient), с предобуславливателем Diagonal-based Incomplete Cholesky (DIC, неполное разложение Холецкого). Далее идёт информация, что решается уравнение для величины T (Solving for T), и информация о начальной (Initial), конечной (после решения СЛАУ, Final) невязки (residual), причём значение невязки отнормировано на начальное значение (поэтому в первой строке Initial residual равно 1), и количестве совершённых итераций линейного ре-

шателя (No Iterations). Поскольку в рассматриваемом примере требуется решение стационарной задачи (т.е. при бесконечном времени), то достаточно добиться уменьшения начальной невязки первой итерации на каждом шаге по времени до требуемой точности (при достижении времени значения 3 начальная невязка на первой итерации составляет величину около 0.0001, что означает падение невязки на 4 порядка; такая сходимость вполне достаточна для рассматриваемой задачи). Подробнее о методах решения СЛАУ будет говориться в разделе 3.1.6.

В конце каждого шага по времени выводится информация о затраченном реальном времени вплоть до данного шага по времени. ExecutionTime содержит информацию о затраченном времени процессора на расчёт, а ClockTime — реальное прошедшее время начиная с запуска решателя (в секундах; последняя величина может сильно превосходить первую в случае, если процессор был загружен ещё какими-либо процессами, либо выполнение программы было приостановлено).

Зачастую информацию о ходе сходимости задачи бывает полезно представить в виде графиков так называемой истории сходимости. Под этим подразумевается зависимость начальной невязки на каждом шаге по времени от времени (или от номера итерации в случае решения стационарной задачи каким-либо итерационным методом). Для удобства пользователя из вывода решателя OpenFOAM можно извлечь информацию о сходимости при помощи утилиты foamLog. Для этого необходимо перенаправить вывод решателя в файл следующим образом:

```
laplacianFoam > log-file &
```

где `log-file` — имя файла. куда будет писаться вывод, а знак & необходим для запуска программы в фоновом режиме. После окончания работы решателя можно запустить программу для извлечения истории сходимости

```
foamLog log-file
```

Эта команда создаёт каталог `logs`, куда помещает файлы с различными данными (извлечёнными из файла `log-file`), в частности, файлы `T_0`, `T_1`, `T_2`, состоящие из двух столбцов: первый столбец — это момент времени (физического), второй — значение начальной невязки (Initial residual); 0, 1 или 2 соответствует номеру итерации по ско-

шенности. Наиболее интересной в данном примере является зависимость невязки на первой итерации от времени, т.е. содержимое файла `T_0`. В качестве иллюстрации на рисунке 2.3 приведён график зависимости начальной невязки от времени из файла `T_0` для рассматриваемого примера.

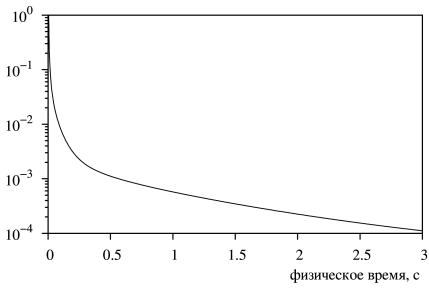

Рис. 2.3. Пример истории сходимости для задачи теплопроводности во фланце (начальная невязка уравнения для T в зависимости от времени)

Отметим быстрое падение невязки в начале процесса и далее постепенное её уменьшение. Такое поведение связано с тем, что начальное распределение температуры является «разрывным»: во всей области задаётся значение 273 K, а на некоторых границах — 573 K.

Помимо информации о невязках также может представлять интерес информация о количестве итераций линейного решателя при решении СЛАУ. В частности, в данной задаче итерации по скошенности требуются на начальном этапе расчёта, далее изменение решения во времени идёт настолько медленно, что итерации по скошенности уже не требуются (невязка СЛАУ после первой итерации становится достаточно малой, так что на следующих итерациях СЛАУ уже не решается и количество итераций линейного решателя равно нулю).

2.3.5. Визуализация решения при помощи пакета ParaView

Для того, чтобы посмотреть результаты расчёта, необходимо воспользоваться какой-либо программой визуализации. В дистрибутив OpenFOAM (точнее, в части ThirdParty сторонних приложений) входит с свободнораспространяемый пакет ParaView, о котором более подробно будет говориться чуть ниже. Однако, помимо ParaView можно воспользоваться и другими программами, в частности, которые поддерживают формат

VTK[1]. Отметим среди таких свободно-распространяемых программ пакет VisIt[2], одним из преимуществом которого по сравнению с ParaView является более гибкая настройка внешнего вида графиков и рисунков, что позволяет создавать иллюстрации, в большей степени пригодные для научных статей и отчётов. Для визуализации 2D графиков[3] (и не только) можно использовать также следующие свободно-распространяемые программы: gnuplot (http://www.gnuplot.info), LabPlot (https://www.kde.org/applications/education/labplot), SciDAVis (http://scidavis.sourceforge.net) и др.

Краткое описание возможностей пакета ParaView

ParaView — это открытый графический кросс-платформенный пакет для интерактивной визуализации, разрабатываемый Национальной Лабораторией Сандиа, компанией Kitware и Национальной Лабораторией Лос-Аламоса[4]. Распространяется под свободной лицензией BSD. Пакет предназначен для визуализации 3D-данных, представленных в виде «расчётная сетка + поля данных». Основными особенностями пакета являются:

- Использование библиотеки VTK[5].
- Возможность работы с большим объёмом данных и в параллельном режиме.
- Наличие множества операций над данными (т.н. «фильтры»).
- Возможность использования скриптов на языке Python и работы в полностью консольном режиме.

Отметим, что все операции над данными (фильтры) в ParaView являются *немодифицирующими*, то есть при выполнении какой-либо операции создаётся новый объект. Например, при визуализации трёхмерного объекта

[1] В OpenFOAM есть утилиты, которые позволяют экспортировать данные в различные форматы, см. applications/utilities/postProcessing/dataConversion

[2] Подробнее см. https://wci.llnl.gov/simulation/computer-codes/visit

[3] То есть зависимостей вида $y(x)$

[4] См. https://ru.wikipedia.org/wiki/ParaView, http://www.paraview.org

[5] Visualization Toolkit, http://www.vtk.org

зачастую необходимо делать его срез (slice) плоскостью. При этом из исходных трёхмерных данных «вырезаются» данные для среза без изменения исходных данных, то есть создаётся новый объект «slice».

Визуализация решения OpenFOAM

Полученные в расчёте OpenFOAM данные можно визуализировать при помощи ParaView двумя способами. В случае, если ParaView собран из исходных кодов (в каталоге `ThirdParty`), OpenFOAM предлагает также специальную утилиту paraFoam, которую необходимо запускать из рабочего каталога задачи (в рассматриваемом примере это каталог `$FOAM_RUN/flange`); эта утилита подготавливает данные расчёта OpenFOAM для чтения их ParaView и запускает саму программу ParaView. Однако, в пакете ParaView также есть встроенный модуль для чтения данных OpenFOAM. Для того, чтобы подгрузить данные встроенным модулем, необходимо в рабочем каталоге задачи создать пустой файл с именем, соответствующим названию каталога, и расширением `.foam`. В рассматриваемом примере это выполняется командой

```
touch flange.foam
```

После этого загрузить ParaView с данными расчёта для рассматриваемого примера можно при помощи команды

```
paraview flange.foam
```

Рабочее окно программы Paraview с загруженными данными для рассматриваемого примера приведено на рисунке 2.4. Как видно, окно разделено на несколько различных областей. Рассмотрим подробнее каждую из этих областей.

Большую часть занимает область, где собственно и выводится изображение (в данном примере это геометрия и сетка на поверхности фланца); она располагается в нижней правой части основного окна (с именем RenderView1 на рисунке 2.4). Управление изображением осуществляется мышкой (поворот, приближение-удаление и перемещение), а также кнопками на панели инструментов (слева от надписи RenderView1).

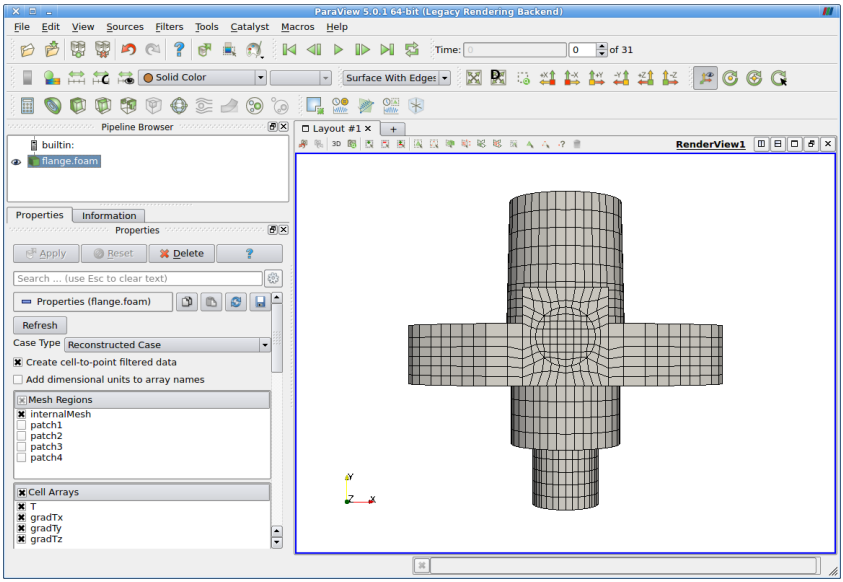

Рис. 2.4. Окно программы ParaView с загруженными данными из примера flange

Вверху основного окна располагается меню и панели инструментов. Первый ряд панели инструментов связан с общими действиями с данными, в том числе и номер отображаемого временного слоя (подпись «Time:» вверху в правой части); таким образом, при изменении номера временного слоя отображаются данные, выведенные в соответствующий момент времени (при загрузке ParaView показываются данные в начальный момент времени). В следующем ряду находятся кнопки для управлением изображением; следует отметить выпадающее меню («Solid Color» на рисунке), где выбирается, каким образом закрашивать поверхность изображаемого объекта, а также выпадающее меню с надписью «Surface With Edges» — что рисовать на поверхности объекта (в данном случае это залитая однотонным цветом поверхность с нанесёнными сеточными линиями). Последний ряд кнопок служит для быстрого вызова различных операций с данными (т.н. фильтры; все фильтры можно найти в разделе меню «Filters»).

Слева от окна с изображением находятся две небольшие области. Это «Pipeline Browser» (окно со списком доступных объектов) и «Properties» (свойства текущего выбранного объекта, в качестве которого в данном примере выступает flange.foam, т.е. данные расчёта OpenFOAM). В области «Properties» присутствуют несколько разделов. Наиболее важные – это раздел «Mesh Regions», где можно указать, какие области сетки необходимо загрузить, и «Cell Arrays», где указываются загружаемые поля величин. В данном примере в качестве области сетки выбрана только «InternalMesh» (все внутренние ячейки сетки) и загружены следующие поля: T, gradTx, gradTy, gradTz (поле температуры, а также посчитанные самим решателем laplacianFoam компоненты градиента температуры). Если требуется визуализировать данные только на одной из граничных зон (в данном примере граничные зоны имеют имена patch1, patch2 и т.д.), необходимо выбрать требуемую зону и нажать «Apply». Кнопка «Refresh» может быть полезна в случае, если после запуска ParaView в рабочем каталоге задачи появились выводные файлы на новых временных слоях.

Простейшие операции в ParaView

Визуализация решения может потребовать предварительной обработки загруженных данных (например, сделать «срез» расчётной области), что в терминах ParaView называется «фильтром». В ParaView имеется множество различных методов обработки данных, полный список можно посмотреть в пункте меню Filters → Alphabetical. Один из наиболее часто используемых фильтров — Slice (сечение плоскостью). На рисунке 2.5 приведён пример применения этого фильтра (сечение плоскостью, ортогональной оси z, и проходящей через центр координат). Полученное сечение окрашено по значениям температуры. Отметим следующие изменения по сравнению с предыдущим рисунком: выбран последний момент времени (равный 3), т.е. данная картина соответствует установившемуся решению, и в области «Pipeline Browser» появился новый объект с именем «Slice1» (также изменены некоторые параметры отображения).

Перечислим некоторые другие наиболее используемые фильтры с кратким их описанием.

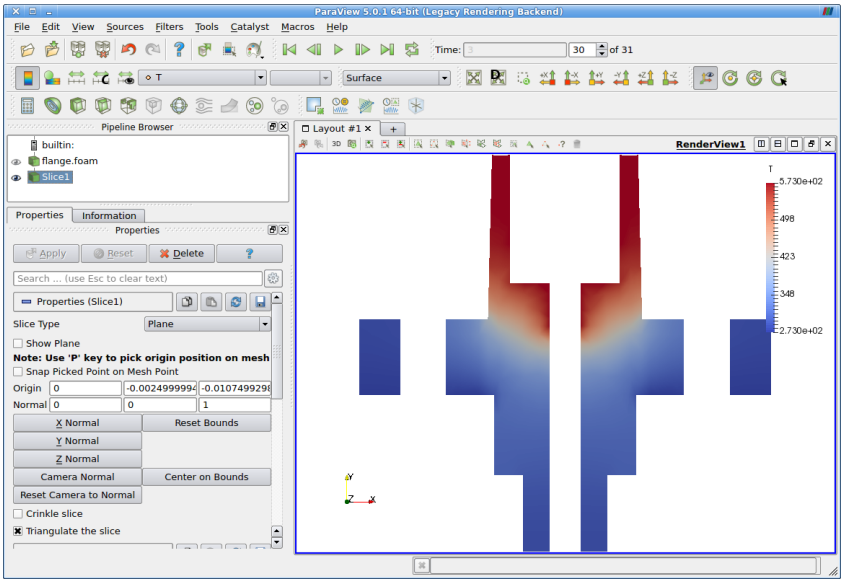

Рис. 2.5. Результат применения фильтра «Slice» в ParaView

Clip — примерно аналогичен Slice, но при работе последнего создаётся поверхностная сетка на секущей плоскости, тогда как Crop обрезает по выбору левую или правую область от секущей плоскости, оставляя соответствующую объёмную сетку.

Contour — рисует изоповерхности какого-либо скаляра по заданным пользователем значениям.

Threshold — оставляет только те ячейки, которые удовлетворяют заданному критерию (значение какого-либо скаляра в них лежит в указанном диапазоне).

Glyph — рисует векторное поле (в виде, задаваемом пользователем; в частности, объёмными или плоскими стрелками).

Plot over line — извлекает (и отображает в виде 2D графика) данные вдоль указанной прямой линии.

Calculator — создаёт новое поле при помощи различных математических операций с имеющимися полями.

Отметим, что при помощи ParaView можно излекать данные различным образом, однако, не всегда качество визуализации этих данных самим ParaView бывает приемлемым. В последнем случае можно сохранить «извлечённые» данные в файл, воспользовавшись пунктом меню File → Save Data..., чтобы затем визуализировать их в какой-либо другой программе (наиболее «человеко-читаемый» формат — это csv, Comma Separated Values; он поддерживается многими приложениями).

Необходимо сделать небольшое замечание, касающееся типов данных, поддерживаемых ParaView. Расчётная сетка, загружаемая в ParaView, определяется своими узлами и ячейками. Данные (поля) могут быть определены либо в центрах ячеек, либо в узлах. В первом случае тип данных в ParaView называется «Cell Array», а во втором — «Point Array». При загрузке полей из OpenFOAM (которые определены в центрах ячеек) автоматически происходит интерполяция из центров ячеек в узлы с созданием новых полей. Именно поэтому в результате в ParaView возникает как бы «двойные» поля T и других величин, как можно видеть на рисунке 2.6, где представлено выпадающее меню для выбора метода «раскраски» поверхности. Здесь элементы списка с символом в виде жирной точки означают те самые интерполированные в узлы сетки поля, а с символом кубика — исходные поля, определённые в центрах ячеек. Отметим, что некоторые фильтры могут работать только с типом «Point Array».

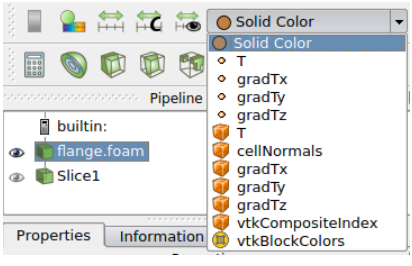

Рис. 2.6. Список полей, загружённых (и интерполированных) из OpenFOAM

2.4. Расчётная сетка в OpenFOAM

OpenFOAM использует собственный формат хранения расчётных сеток и поддерживает любые типы ячеек (тетраэдры, шестигранники, призмы и другие произвольные многогранники; все типы ячеек в дальнейшем для краткости будем называть полиэдральными ячейками), но работает только с трёхмерными сетками (позволяя при этом считать на сетках с одной

ячейкой в каком-либо направлении, что применяется, в частности, для расчётов двумерных и осесимметричных задач).

Необходимо подчеркнуть, что в OpenFOAM имеется некоторое количество собственных генераторов сеток (в т.ч. автоматических), однако, их использование сопряженно с некоторыми техническими сложностями. Например, распространённая для создания простейших сеток утилита blockMesh работает только с блочно-структурированными сетками (где используются шестигранные ячейки), а задание параметров для неё осуществляется через текстовый файл (подробнее об этой утилите будет рассказано далее). Аналогичным образом (через текстовый файл) задаются и параметры для других генераторов сеток. В этой связи одним из способов создание сетки является импортирование её из каких-либо сторонних форматов. Среди свободно-распространяемых программ для генерации сетки можно отметить программный пакет SALOME[1] (в котором генерация сетки является только одним из модулей) и программу gmsh[2] (предназначенную в основном для создания тетраэдральных сеток). В отличие от утилит OpenFOAM, эти программы имеют графический интерфейс GUI (Graphic User Interface), что в некоторой степени облегчает процесс создания сетки.

Как и ранее, здесь необходимо разделить два понятия: геометрия задачи и расчётная сетка. Геометрия представляет собой расчётную область конкретной задачи; она может быть задана в различной форме, в том числе и в виде модели CAD[3] (т.е. компьютерного «чертежа» интересующего объекта. Например, в задаче внешнего обтекания автомобиля воздушным потоком с точки зрения расчёта интересует только его корпус; задание поверхности корпуса в виде компьютерного «чертежа» и будет являться CAD-моделью). В этой связи отметим возможность использования в OpenFOAM геометрии, заданной в виде CAD-модели (в некотором формате), для построения сетки. С технической точки зрения CAD-модель представляет собой поверхностную сетку (или другими словами поверхность, заданную при помощи дискретного набора точек и/или двумерных элементов типа треугольников, квадратов и т.п.). Естественно, геометрия являются частью постановки задачи (разная геометрия будет соответствовать

[1] См. http://www.salome-platform.org
[2] См. http://gmsh.info
[3] Computer-aided drafting или система автоматического проектирования (САПР)

разным задачам). Расчётная сетка же не является частью постановки задачи, а представляет собой один из «параметров» решения; сетка строится в объёме заданной геометрии и представляет собой по сути дискретизацию пространства.

Поскольку для численного расчёта требуется только расчётная сетка (которая в свою очередь определяет геометрию задачи через свои границы), в дальнейшем (если не оговаривается особо) под расчётной сеткой будем понимать как собственно сетку, так и определяемую с её помощью геометрию задачи.

2.4.1. Формат сетки в OpenFOAM, типы границ

Как отмечалось ранее, расчётная сетка в формате OpenFOAM хранится в каталоге задачи `constant/polyMesh`. Рассмотрим немного более подробно формат её хранения (детали можно найти, например, в [12]). Для этого снова обратимся к рассмотренному выше примеру задачи теплопроводности в фланце (см. раздел 2.3). После того, как сетка была экспортирована из формата ANSYS (файл `flange.ans`), в каталоге `constant/polyMesh` вся информация о сетке хранится в следующих файлах[1]: `boundary`, `faces`, `neighbour`, `owner`, `points`. Каждый из этих файлов записан в уже упоминавшемся стандартном формате с использованием синтаксиса C++. Рассмотрим предназначения каждого файла немного подробнее.

В файле `points` содержится список координат всех узлов сетки, записанном в формате, представленном ниже (здесь и далее стандартную «шапку» файлов будем опускать).

```
Файл flange/constant/polyMesh/points:
FoamFile
{
    version     2.0;
    format      ascii;
    class       vectorField;
    location    "constant/polyMesh";
```

[1] Также в рассматриваемом примере в каталоге constant/polyMesh присутствует файл `faceZones` — он создаётся при импортировании сетки из формата ANSYS и содержит некоторую дополнительную информацию, которая не требуется для чтения сетки в OpenFOAM, но может быть использована другими утилитами OpenFOAM при работе с сеткой

```
    object      points;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
7189
(
(0.0023537 -1.74e-18 -0.01375)
(0.00170133 -3.04e-18 -0.01375)
(4.8963e-05 0.00230474 -0.01375)
...
```

Сначала идёт целое число — количество всех узлов сетки (в примере их 7189 штук). Затем последовательно перечисляются координаты каждого узла в формате (x, y, z). Отметим, что класс объекта points объявлен как vectorField — то есть как векторное поле радиус-векторов узлов.

Файл faces содержит список всех граней всех ячеек в виде, представленном ниже.

```
Файл flange/constant/polyMesh/faces:
FoamFile
{
    version     2.0;
    format      ascii;
    class       faceList;
    location    "constant/polyMesh";
    object      faces;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
18584
(
4(19 21 43 35)
4(10 35 43 14)
4(3 14 43 21)
...
```

Здесь каждый элемент списка представлен в виде «количество узлов (список индексов узлов из файла `points`)». Поскольку грани в принципе могут быть произвольными многоугольниками, то «количество узлов» варьируется от 3 (треугольник) и выше. В данном примере представленные на листинге грани представляют собой четырёхугольники. «Индекс узла» представляет собой номер в списке узлов из файла `points` (нумерация идёт

от 0); последовательность узлов для каждой грани определяет направление вектора нормали к этой грани.

Соответствие граней и ячеек устанавливается при помощи файлов `owner` и `neighbour`. Поскольку каждая грань может принадлежать только либо одной ячейке (граничные грани), либо двум ячейкам (внутренние грани), то для неё можно выделить её «владельца» (owner) и «соседа» (neighbour) при помощи следующего правила: нормаль к данной грани должна быть направлена от ячейки «владельца» к ячейке «соседа». Таким образом, в файле `owner` для каждой грани определяется номер ячейки «владельца», а в файле `neighbour` — номер ячейки «соседа» (причём очевидно, что количество «соседей» меньше, чем число «владельцев», поскольку у граничных граней есть только один «владелец»; отсюда также следует, что нормаль к граничной грани всегда направлена из расчётной области). Более подробную информацию о формате хранения сетки в OpenFOAM можно найти в упоминавшейся книге [12] и в руководстве пользователя OpenFOAM [14].

Наиболее важным с точки зрения пользователя является файл `boundary`, в котором описываются граничные зоны расчётной сетки. Его содержимое для рассматриваемого примера приведено в листинге.

```
Файл flange/constant/polyMesh/boundary:
FoamFile
{
    version     2.0;
    format      ascii;
    class       polyBoundaryMesh;
    location    "constant/polyMesh";
    object      boundary;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
4
(
    patch1
    {
        type            patch;
        nFaces          2440;
        startFace       15316;
    }
    patch2
```

```
    {
        type            patch;
        nFaces          348;
        startFace       17756;
    }
    patch3
    {
        type            patch;
        nFaces          96;
        startFace       18104;
    }
    patch4
    {
        type            patch;
        nFaces          384;
        startFace       18200;
    }
)
// ************************************************************************* //
```

Как видно, файл boundary представляет собой список (в данном случае — из четырёх записей) граничных зон. Каждая зона начинается с имени зоны (например, patch1; имя может быть любым), далее следует описание зоны. Зона имеет определённый тип (type), количество граней в зоне определяется значением nFaces. Величина startFace представляет собой индекс первой грани из файла faces (таким образом, в файле faces все граничные грани должны быть соответствующим образом упорядочены). Отметим, что файл boundary может быть отредактирован пользователем (например, для изменения названия зоны или её типа), но также имеются и специальные утилиты для работы с сеткой, позволяющие менять граничные зоны (разбивать или соединять зоны вместе и др.), в частности, для работы с граничными зонами служит утилита createPatch. Вопрос о работе с граничными зонами сетки выходит за рамки данного пособия; интересующиеся могут найти достаточно много информации по этому вопросу на форуме http://www.cfd-online.com/Forums/openfoam.

Наиболее важным для пользователя в файле boundary является информация о типе граничной зоны. От этого зависит, какое граничное условие

может быть поставлено для данной зоны. Возможными вариантами являются:

- patch — наиболее общий тип границы; можно задавать произвольные граничные условия первого, второго или третьего рода;
- wall — условие «стенка», обеспечивает корректную работу решателя в случае использования моделей турбулентности (в частности, требуется для пристенных функций);
- symmetryPlane — плоскость симметрии (вся зона должна быть плоской; на ней может задаваться только граничное условие symmetryPlane);
- symmetry — общее условие симметрии (в т.ч. и для неплоских поверхностей);
- empty — используется в случае 2D-постановки (а также 1D-постановки) для тех поверхностей, которые лежат «в плоскости» рассматриваемой задачи (для данной граничной зоны можно использовать только граничное условие empty);
- wedge — аналогично empty, но для осесимметричной постановки (с одной ячейкой в окружном направлении);
- cyclic и cyclicAMI — для периодических (в т.ч. вращательной периодичности) граничных условий; первое используется при стыковке двух периодических поверхностей «грань-в-грань», а второе — для стыковки двух поверхностей с нестыкуемыми поверхностными сетками (AMI — Arbitrary Mesh Interface, произвольная стыковка сеток);
- processor — специальный тип, который указывает, что данная зона является стыковочной при декомпозиции расчётной области для запуска в параллельном режиме на нескольких процессах.

В рассматриваемом примере все граничные зоны имеют тип patch; как показано в разделе 2.3.2, на них задаются граничные условия первого (фиксированная температура, fixedValue) или второго (нулевая производная, zeroGradient) рода.

2.4.2. Создание сетки при помощи утилиты blockMesh

Рассмотрим простейшую утилиту OpenFOAM для создания блочно-структурированных сеток blockMesh. Поскольку все параметры геометрии и сетки задаются в специальном файле `blockMeshDict` в текстовой форме, эта утилита зачастую не подходит для создания хороших сеток в сложных геометриях. Тем не менее, для простых задач её функционала бывает достаточно. Кроме того, blockMesh может служить для построения простой начальной сетки, которая может использоваться далее для автоматической генерации сетки в сложной геометрии (например, при помощи утилиты snappyHexMesh).

Рассмотрим простейший пример построения сетки для задачи 2D течения в квадратной каверне (см. рисунок 2.7). Постановку этой задачи можно найти в примерах OpenFOAM в каталоге `$WM_PROJECT_DIR/tutorials/incompressible/icoFoam/cavity/cavity`.

Файл с параметрами геометрии и сетки для утилиты blockMesh называется `blockMeshDict` и находится в каталоге `system`. Поскольку blockMesh создаёт блочно-структурированные сетки, основной элемент при построении сетки — так называемый топологический блок, который характеризуется восьмью вершинами. Таким образом, для рассматриваемого примера необходимо задать координаты восьми вершин топологического блока, причём поскольку задача двумерная, четыре вершины лежат в плоскости xy, а другие четыре вершины — в плоскости, сдвинутой по z относительно xy на произвольное расстояние; эта геометрия представлена на рисунке 2.8 (нумерация вершин начинается от нуля). В случае более сложной геометрии необходимо разбить расчётную область на блоки таким образом, чтобы «грани» этих блоков соприкасались «один в один» (в общем случае такое разбиение довольно нетривиально).

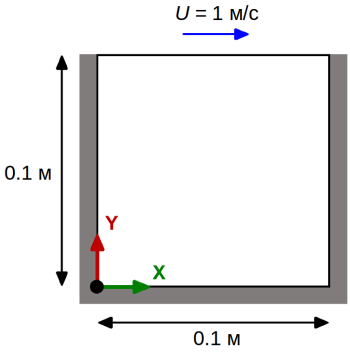

Рис. 2.7. Геометрия для задачи течения жидкости в квадратной каверне

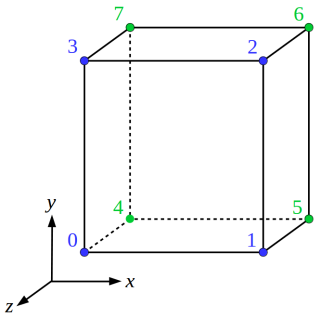

Рис. 2.8. Топологический блок

Координаты вершин топологического блока задаются в файле blockMeshDict в виде списка vertices, как показано ниже.

```
convertToMeters 0.1;
vertices
(
    (0 0 0)     // Вершина 0
    (1 0 0)     // Вершина 1
    (1 1 0)     // Вершина 2
    (0 1 0)     // Вершина 3
    (0 0 0.1)   // ...
    (1 0 0.1)
    (1 1 0.1)
    (0 1 0.1)
);
```

Ключевое слово converToMeters означает масштаб, в котором записываются координаты вершин (для удобства пользователя; в данном примере при построении сетки координаты точек будут умножаться на множитель 0.1). Синтаксис для задания блока представлен ниже.

```
blocks
(
    hex (0 1 2 3 4 5 6 7) (20 20 1) simpleGrading (1 1 1)
);
```

Описание каждого топологического блока в списке blocks начинается с ключевого слова hex. За ним идёт список вершин, составляющих этот блок, причём каждое (целое) число здесь соответствует порядковому номеру вершины из списка vertices. Вершины должны перечисляться в определённом порядке, а именно: первые четыре вершины перечисляются в порядке их обхода против часовой стрелки относительно внешней нормали к образуемой ими грани, а последующие четыре вершины должны перечисляются в том же порядке (другими словами, соответствующие вершины из каждой «четвёрки» должны попарно лежать на одном ребре). Таким образом, корректным также будет, например, такой порядок перечисления вершин: (1 2 3 0 5 6 7 4).

За списком вершин блока идёт список (20 20 1). Каждое их этих трёх чисел представляет собой количество разбиений соответствующего ребра топологического блока для последующего создания сетки (подчеркнём, что для структурированной сетки количество разбиений на соответствующих параллельных рёбрах должно быть одинаковым). При этом здесь вводится собственная «блочная система координат». Если блок определён условно следующим образом: hex (N0 N1 N2 N3 N4 N5 N6 N7), где N0 и т.д. — номера вершин, то первая ось блочной системы координат определяется по ребру (N0 N1), вторая — по ребру (N0 N3), третья — по ребру (N0 N4). Таким образом, в рассматриваемом примере (см. рисунок 2.8) первое ребро (0 1) и параллельные ему рёбра (3 2), (4 5), (7 6) будут разбиты на 20 промежутков, второе ребро (0 3) и параллельные ему рёбра (1 2), (4 7), (5 6) — также на 20 промежутков, а третье ребро (0 4) и параллельные ему рёбра (1 5), (3 7), (2 6) — на 1 промежуток (т.е., одна ячейка в третьем направлении, что необходимо для 2D задач).

Ключевое слово simpleGrading означает степень сгущения (задаётся в виде списка из трёх вещественных чисел) в каждом из трёх направлений блочной системы координат. Степень сгущения определяется как отношение размера последнего промежутка к первому (в рассматриваемом примере степень сгущения во всех направлениях равна 1, что означает равномерную сетку). Помимо ключевого слова simpleGrading можно также использовать edgeGrading — задание степени сгущения отдельно для каждого из 12 рёбер.

При использовании simpleGrading возможно также задавать разные степени сгущения на отдельных частях ребра. Для этого вместо указания степени сгущения по одному (или нескольким) из направлений задаётся список вида:

```
simpleGrading ( ((0.5 0.5 10) (0.5 0.5 0.1)) 1 1)
```

Между первой открывающейся и последней закрывающейся скобками внутри списка simpleGrading идёт произвольное количество записей вида «(число1 число2 число3)». Первое число «число1» определяет долю длины ребра, на которой будет задаваться сгущение. Второй число «число2» определяет долю ячеек, приходящихся на выбранную часть ребра. Третье «число3» — это и есть коэффициент сгущения на данной части ребра.

В указанном примере рёбра, лежащие вдоль первой оси блочной системы координат, будут поделены пополам, на каждую половину будет приходится количество ячеек, равное 0.5 от указанного ранее (до ключевого слова simpleGrading) количества разбиений, для первой части будет использоваться коэфициент сгущения 10, для второй — 0.1. Более подробно о рассматриваемой опции можно найти в [14], раздел 5.3.1.4.

По умолчанию все рёбра строящейся сетки являются прямыми. Однако утилита blockMesh позволяет задавать также и криволинейные рёбра. Для этого в `blockMeshDict` можно использовать ключевое слово edges, например, как представлено в листинге.

```
edges ( arc 1 5 (1.1 0.0 0.5) );
```

В данном случае вершины 1 и 5 буду соединены не прямым отрезком, а дугой окружности с некоторыми параметрами. Помимо дуги окружности можно задавать и другие криволинейные отрезки. Подробнее об этом можно посмотреть в [14], раздел 5.3.1.2.

Следующий важный шаг при построении сетки — задание граничных зон. В `blockMeshDict` для этого используется ключевое слово boundary как показано в листинге на примере задачи течения в каверне.

```
boundary
(
    movingWall
    {
        type wall;
        faces
        (
            (3 7 6 2)
        );
    }
    fixedWalls
    {
        type wall;
        faces
        (
            (0 4 7 3)
            (2 6 5 1)
            (1 5 4 0)
```

```
        );
    }
    frontAndBack
    {
        type empty;
        faces
        (
            (0 3 2 1)
            (4 5 6 7)
        );
    }
);
```

Здесь представлено объявление трёх граничных зон с именами: movingWall, fixedWalls и frontAndBack. Зона movingWall является поверхностью «движущейся стенки», fixedWalls — три других неподвижные стенки (нижняя и боковые, см. рисунок 2.7), а frontAndBack представляет собой две поверхности (передняя и задняя) в плоскости рисунка. При объявлении каждой из граничных зон необходимо указать её тип type (должен соответствовать типам из раздела 2.4.1) и грани faces топологического блока, составляющие описываемую границу. Как видно из листинга, каждая грань блока в faces задаётся четырьмя вершинами (их порядковыми номерами); порядок их перечисления не существенен. В случае, если какие-то грани топологического блока не описаны в разделе boundary, они все включаются в одну граничную зону с именем defaultFaces.

Для генерации сетки необходимо запустить утилиту blockMesh в корневом каталоге задачи. Вывод в процессе работы этой утилиты для примера построения сетки в задаче о течении в каверне представлен ниже.

```
Часть вывода в процессе работы утилиты blockMesh:
Create time
Creating block mesh from
    "/home/aero/OpenFOAM/aero-7/run/cavity/system/blockMeshDict"
Creating block edges
No non-planar block faces defined
Creating topology blocks
Creating topology patches
Creating block mesh topology
Check topology
```

```
        Basic statistics
                Number of internal faces : 0
                Number of boundary faces : 6
                Number of defined boundary faces : 6
                Number of undefined boundary faces : 0
        Checking patch -> block consistency
Creating block offsets
Creating merge list .
Creating polyMesh from blockMesh
Creating patches
Creating cells
Creating points with scale 0.1
    Block 0 cell size :
        i : 0.005 .. 0.005
        j : 0.005 .. 0.005
        k : 0.01

Writing polyMesh
----------------
Mesh Information
----------------
  boundingBox: (0 0 0) (0.1 0.1 0.01)
  nPoints: 882
  nCells: 400
  nFaces: 1640
  nInternalFaces: 760
----------------
Patches
----------------
  patch 0 (start: 760 size: 20) name: movingWall
  patch 1 (start: 780 size: 60) name: fixedWalls
  patch 2 (start: 840 size: 800) name: frontAndBack
End
```

Наиболее интересна здесь информация о размере сетки (Mesh Information) и созданных граничных зонах (Patches). Как следует из вывода, сетка содержит 882 узлов, 400 ячеек и 1640 граней, из них 760 внутренних. Создано три граничные зоны с соответствующими именами.

Для проверки корректности построенной сетки следует воспользоваться утилитой checkMesh. Отметим, что этой утилитой следует пользоваться всегда в процессе постановки расчёта, в т.ч. и при импорте сеток из других

форматов. Результат работы checkMesh на только что построенной сетке представлен ниже.

```
Часть вывода в процессе работы утилиты checkMesh:
Mesh stats
    points:           882
    internal points:  0
    faces:            1640
    internal faces:   760
    cells:            400
    faces per cell:   6
    boundary patches: 3
    point zones:      0
    face zones:       0
    cell zones:       0
Overall number of cells of each type:
    hexahedra:     400
    prisms:        0
    wedges:        0
    pyramids:      0
    tet wedges:    0
    tetrahedra:    0
    polyhedra:     0
Checking topology...
    Boundary definition OK.
    Cell to face addressing OK.
    Point usage OK.
    Upper triangular ordering OK.
    Face vertices OK.
    Number of regions: 1 (OK).
Checking patch topology for multiply connected surfaces...
    Patch               Faces    Points   Surface topology
    movingWall          20       42       ok (non-closed singly connected)
    fixedWalls          60       122      ok (non-closed singly connected)
    frontAndBack        800      882      ok (non-closed singly connected)
Checking geometry...
    Overall domain bounding box (0 0 0) (0.1 0.1 0.01)
    Mesh has 2 geometric (non-empty/wedge) directions (1 1 0)
    Mesh has 2 solution (non-empty) directions (1 1 0)
    All edges aligned with or perpendicular to non-empty directions.
    Boundary openness (8.47033e-18 -8.47033e-18 -4.51751e-17) OK.
    Max cell openness = 1.35525e-16 OK.
```

```
    Max aspect ratio = 1 OK.
    Minimum face area = 2.5e-05. Maximum face area = 5e-05. Face area
    ↪ magnitudes OK.
    Min volume = 2.5e-07. Max volume = 2.5e-07.  Total volume = 0.0001.
    ↪ Cell volumes OK.
    Mesh non-orthogonality Max: 0 average: 0
    Non-orthogonality check OK.
    Face pyramids OK.
    Max skewness = 1.66533e-14 OK.
    Coupled point location match (average 0) OK.
Mesh OK.
```

В данном случае checkMesh выводит «Mesh OK», что означает успешную выполнение проверки. В случае, если при проверке возникла ошибка, будет выведено сообщение об этом. Как следует из вывода, checkMesh осуществляет простейшие проверки корректности геометрических характеристик сетки «Checking geometry», в частности, рассчитывает объёмы ячеек, площади их граней и т.п. Кроме итого, из вывода можно найти информацию о типе ячеек «Overall number of cells of each type». В данном случае все 400 ячеек являются шестигранниками.

2.4.3. Импорт из разных форматов и другие утилиты для работы с сеткой

Поскольку создание сложных сеток средствами OpenFOAM — задача довольно нетривиальная, рассмотрим более подробно вопрос импортирования сеток, созданных при помощи других инструментов.

Полный список утилит, предназначенных для конвертации сеток как из формата так и в формат OpenFOAM, можно найти в каталоге `$WM_PROJECT_DIR/applications/utilities/mesh/conversion`. В частности, для импортирования сетки, созданной при помощи свободнораспространяемого генератора gmsh, используется утилита gmshToFoam. В случае, если сетка была создана в пакете SALOME, то для возможности чтения её OpenFOAM необходимо в самом SALOME записать её в формате I-DEAS UNV; тогда импорт её в OpenFOAM осуществляется при помощи утилиты ideasUnvToFoam. Для чтения сетки, записанной в формате ANSYS Fluent (файл с расширением `.msh` или `.cas`), используются утилиты

fluent3DMeshToFoam (для трёхмерных сеток) или fluentMeshToFoam (для двумерных и трёхмерный сеток). Записать сетку из OpenFOAM в формат ANSYS Fluent можно при помощи утилиты foamMeshToFluent. Отметим, что некоторые утилиты позволяют при импорте масштабировать сетку при помощи опции -scale. Посмотреть список опций какой-либо утилиты можно выполнив команду с опцией -help (это замечание касается всех программ из пакета OpenFOAM), например:

```
fluent3DMeshToFoam -help
```

Отметим также возможность импортирования сетки из формата VTK при помощи утилиты vtkUnstructuredToFoam (это может быть полезно, в частности, при записи сетки из ParaView, который имеет возможность создавать простейшие сетки для некоторых геометрических фигур).

OpenFOAM предлагает множество утилит для различных операций с сеткой (все они находятся в каталоге `$WM_PROJECT_DIR/applications/utilities/mesh/manipulation`). В частности, утилита transformPoints позволяет совершать различные геометрические операции над координатами узлов сетки в зависимости от указанной опции: rotate — поворот сетки от вектора A к вектору B, scale — масштабирование (по каждой из осей в отдельности), translate — параллельный сдвиг вдоль вектора.

3. Решение задач гидроаэродинамики и теплообмена в OpenFOAM (основные положения)

В данной главе будут более подробно рассмотрены решатели, численные схемы и параметры запуска OpenFOAM применительо к задачам гидроаэродинамики и теплообмена. Подробное описание имеющихся в OpenFOAM численных схем, а также различных параметров численного решения, представлено в разделе 3.1. В разделе 3.2 сделан обзор реализованных в OpenFOAM граничных условий и подробно рассмотрены наиболее часто используемые.

3.1. Численные схемы в OpenFOAM

3.1.1. Общие сведения

Рассмотрим теперь простейший пример решения задачи динамики вязкой несжимаемой жидкости на задаче течения в каверне, уже рассмотренной ранее при изучении утилиты для построения сетки blockMesh (см. раздел 2.4.2; постановка задачи находится в каталоге `$WM_PROJECT_DIR/tutorials/incompressible/icoFoam/cavity/cavity`). Как следует из пути, по которому находится каталог с постановкой, расчёт проводится с использованием решателя icoFoam из класса решателей incompressible — простейшего решателя для задач ламинарного движения несжимаемой вязкой жидкости с использованием метода PISO (кратко описанному в разделе 1.3.4). Исходные коды этого решателя находятся в каталоге `$WM_PROJECT_DIR/applications/solvers/incompressible/icoFoam`.

Прежде чем перейти к описанию численных схем, необходимо сделать некоторые пояснения насчёт используемых в OpenFOAM обозначений и подхода к созданию решателей (подробней об этом будет говориться в главе 6). В пакете реализован высокоуровневый подход для описания решаемых уравнений при помощи собственного предметно-ориентированного

языка (Domain Specific Language, DSL). Рассмотрим запись выражений в OpenFOAM на примере уравнения баланса импульса (1.21):

$$\frac{\partial \vec{V}}{\partial t} + \nabla \cdot \left(\vec{V}\vec{V}\right) - \nu\nabla^2\vec{V} = -\frac{1}{\rho}\nabla p.$$

Например, в решателе icoFoam левая часть этого уравнения записывается в виде следующего программного кода (см. файл applications/solvers/incompressible/icoFoam/icoFoam.C):

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

Этот код предназначен для «создания» СЛАУ (системы линейных алгебраических уравнений) для скорости путём аппроксимации уравнения баланса импульса (без слагаемого с градиентом давления) по МКО примерно так, как описано в разделах 1.2, 1.3. Каждое слагаемое, начинающееся с «fvm::», дискретизируется по МКО (см. 1.2.4), при этом конкретный метод дискретизации определяется путём считывания информации из файла system/fvSchemes после запуска программы icoFoam. Этот файл создаётся пользователем и имеет стандартную для управляющих файлов OpenFOAM структуру (как описано в разделе 2.3).

Решение созданной СЛАУ на языке OpenFOAM описывается следующим кодом:

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

Такая конструкция означает, что будет решена созданная под именем «UEqn» СЛАУ, к правой части которой добавлена аппроксимация по МКО слагаемого с градиентом давления[1].

Параметры решателя СЛАУ задаются в файле system/fvSolution, который также имеет стандартную структуру. Кроме параметров линейных

[1] В OpenFOAM во всех решателях из класса incompressible под «p» подразумевается отношение «настоящего» давления к плотности

решателей СЛАУ, в этом файле также задаются параметры используемого метода и релаксационные параметры.

В приведённых выше частях кода используется два пространства имён: fvm и fvc. Первый содержит функции для создания матрицы СЛАУ, а второй — функции для вычисления какого-либо дифференциального оператора от имеющегося поля. Подробнее о них рассказывается в главе 6.

Для рассматриваемого примера (течение в каверне) содержимое файла system/fvSchemes представлено ниже.

```
Файл system/fvSchemes:
ddtSchemes
{
    default         Euler;
}
gradSchemes
{
    default         Gauss linear;
    grad(p)         Gauss linear;
}
divSchemes
{
    default         none;
    div(phi,U)      Gauss linear;
}
laplacianSchemes
{
    default         Gauss linear orthogonal;
}
interpolationSchemes
{
    default         linear;
}
snGradSchemes
{
    default         orthogonal;
}
```

Файл состоит из нескольких секций, предназначение которых следует из названия: ddtSchemes — схемы аппроксимации временной производной, gradSchemes, divSchemes, laplacianSchemes — схемы дискретизации гради-

ента, дивергенции и оператора Лапласа, interpolationSchemes — схемы интерполяции на грань, snGradSchemes — схемы вычисления производной по нормали на грани.

По сути, при запуске решателя icoFoam происходит следующее: решатель последовательно «создаёт» СЛАУ (на основе вышеприведённого кода) и для каждого слагаемого (начиная с fvm::ddt(U)) ищет запись о способе аппроксимации в файле fvSchemes. Здесь для рассматриваемого примера в секции ddtSchemes в качестве схемы по умолчанию (default) задаётся (неявная) схема Эйлера. Для слагаемого с дивергенцией fvm::div(phi,U) в файле fvSchemes есть отдельная строка (помимо default). Это означает, что именно для этого слагаемого будет использоваться указанная схема. Значение «none» в строке default означает, что по умолчанию никакая схема для дивергенции не используется, поэтому в случае, если в коде решателя есть какое-либо слагаемое с div, но оно не указано в fvSchemes, то при запуске он остановится с ошибкой «не указана схема вычисления дивергенции». Например, если убрать (закомментировать) строку с «div(phi,U)» в файле fvSchemes и запустить решатель icoFoam, то получим примерно следующее сообщение:

```
--> FOAM FATAL IO ERROR:
keyword div(phi,U) is undefined in dictionary ".../fvSchemes.divSchemes"
```

В рассматриваемом примере обратимся к описанию секции divSchemes. В строке, начинающейся с «div(phi,U)», далее следует ключевое слово «Gauss» — это означает, что дискретизация соответствующего оператора производится по МКО (т.е. с привлечением теоремы Гаусса-Остроградского). За этим словом должна указываться используемая схема интерполяции величины (в данном случае U) на грань ячейки; «linear» указывает, что используется обычная линейная интерполяции (которая является абсолютно неустойчивой, а, значит, должна «демпфироваться» вязкими слагаемыми, как описано в разделе 1.3.1).

Что касается секции laplacianSchemes, то здесь описывается дискретизация диффузионных слагаемых. Слово «Gauss» снова указывает на использование МКО, далее следует схема интерполяции коэффициента диффузии на грань (в рассматриваемом примере — это коэффициент вязкости nu; используется линейная интерполяция linear) и далее — схема вычис-

ления производной по нормали (здесь стоит слово orthogonal, что означает простейшую схему без учёта поправки на неортогональность). Секция gradSchemes описывает схему вычисления градиента в центре ячейки: используется формула Грина-Гаусса (Gauss) с линейной интерполяцией давления на грань (linear). Секции interpolationSchemes и snGradSchemes необходимы в том случае, если в коде используется интерполяция или вычисление производной по нормали где-нибудь ещё помимо дифференциальных операторов.

Далее приводятся описания различных численных схем. Некоторые детали, которые здесь опущены, можно найти в руководстве пользователя OpenFOAM [14].

3.1.2. Схемы вычисления диффузионных слагаемых

Рассмотрим более подробно возможные варианты задания схемы для вычисления диффузионных слагаемых laplacianSchemes в файле `fvSchemes`. Синтаксис указания опций (с примером значений параметров из задачи о движении в каверне) выглядит следующим образом:

```
laplacianSchemes
{
  <слагаемое>     <метод> <интерполяция> <вычисление snGrad>;
  laplacian(nu,U) Gauss   linear         orthogonal;
}
```

Здесь предполагается, что дискретизация проводится для слагаемого вида $\nabla \cdot \left(\nu \nabla \vec{V}\right)$, т.е. коэффициент диффузии (здесь кинематический коэффициент вязкости ν) может быть переменным, поэтому необходимо указывать схему его интерполяции на грань.

В первую очередь отметим один простой приём для определения доступных в OpenFOAM опций для того или иного параметра, который применим не только для объёктов в файле `fvSchemes`, но и для большинства других параметров, используемых в OpenFOAM. Это так называемый метод «абракадабры», суть которого заключается в том, что в каком-либо поле вместо валидного значение ввести заведомо бессмысленный набор букв и/или цифр, а затем запустить решатель/утилиту. При чтении файла с параметрами OpenFOAM определяет, является то или иное значение допустимым,

и в противном случае выводит список всех возможных вариантов. Значение опции «метод» в секции laplacianSchemes может быть только «Gauss», поскольку МКО является единственным способом вычисления диффузионных слагаемых в OpenFOAM.

Методы интерполяции коэффициента диффузии на грань могут быть такими же, как и в секциях interpolationSchemes и divSchemes, и буду описаны далее. Что касается метода вычисления производной по нормали к грани snGrad, то здесь доступно несколько опций, а именно:

```
corrected
faceCorrected
limited
linearFit
orthogonal
quadraticFit
uncorrected
```

Опция orthogonal означает вычисление без какой-либо коррекции на скошенность/неортогональность, а опция corrected — вычисление по гибридной схеме, описанной в разделе 1.3.1, с добавлением к главному слагаемому релаксационного множителя $1/\cos\alpha$ (с аналогичным видоизменением поправочного слагаемого; см. формулу (1.12), где вместо $|\overrightarrow{PN}|$ надо подставить $|\overrightarrow{PN}|\cos\alpha$, α — угол между нормалью и линией, соединяющей центры ячеек). Опция uncorrected эквивалентна orthogonal за исключением добавления релаксирующего множителя $1/\cos\alpha$.

Поправка на неортогональность может быть для плохих ячеек очень большой и приводить к расходимости. Для предотвращения этого используется опция limited с идущим после неё числовым значением ψ в диапазоне $[0;1]$, которое указывает, насколько надо применять поправку ($\psi = 0$ даёт uncorrected, $\psi = 1$ — полностью с поправкой, $\psi = 0.5$ — поправка ограничивается таким образом, чтобы она не превышала основное слагаемое). Опция faceCorrected означает вычисление поправки на неортогональность при помощи градиента по касательной к грани, вычисленного по интерполированным в узлы расчётной сетки значениям поля. Схема quadraticFit позволяет сохранить второй порядок точности аппроксимации производной по нормали к грани на неструктурированных сетках за счёт расширения шаблона (при вычислении задействуются не только смежные к грани

ячейки, но и более дальние), однако, обладает меньшей вычислительной надёжностью и может приводить к расходимости на плохих сетках. Схема linearFit также использует механизм, аналогичный quadraticFit, но по сути работает как схема corrected. Отметим, что такие же схемы вычисления производной по нормали доступны и для задания в секции snGradSchemes.

3.1.3. Схемы интерполяции на грань

Схемы интерполяции на грань используются: непосредственно в секции interpolationSchemes; для задания метода интерполяции коэффициента диффузии в секции laplacianSchemes; при задании схемы интерполяции величины для вычисления дивергенции (секция divSchemes). Рассмотрим чуть подробнее синтаксис в секции divSchemes:

```
divSchemes
{
    <слагаемое>     <метод> <интерполяция>;
    div(phi,U)      Gauss   linear;
}
```

Запись оператора дивергенции здесь полностью аналогична формуле (1.23) для конвективного слагаемого: под «phi» здесь понимается расход (в данном случае объёмный) через грань (о задании и вычислении этой величины будет подробно рассказано в главе 6; здесь можно провести корреляцию с SIMPLE-подобными алгоритмами, в котором расход на грань необходимо вычислять в некотором смысле «особым» образом, см. раздел 1.3.4).

В качестве метода дискретизации здесь можно указать либо «Gauss» (что соответствует МКО для вычисления дивергенции), либо «bounded Gauss». Последнее требуется только в случае решения стационарных задач, когда вычисление конвективных слагаемых по консервативной схеме, т.е. по формуле $\nabla \cdot \left(\vec{V}\vec{V}\right)$, даёт сильный вклад из-за того, что в процессе получения стационарного решения $\nabla \cdot \vec{V} \neq 0$, т.е. суммарный расход через ячейку не всегда равен нулю. В этом случае с точки зрения численного расчёта лучше вычислять конвективные слагаемые в неконсервативной формулировке. Однако, как отмечалось в первой главе, расчёт «в лоб» неконсервативной записи, т.е. записи вида $\vec{V} \cdot \nabla \vec{V}$, нарушает свойство

МКО сохранения потоков через грань. Решением здесь является аппроксимация выражения $\nabla\cdot\left(\vec{V}\vec{V}\right)-\vec{V}\left(\nabla\cdot\vec{V}\right)$ вместо исходного, что и происходит при выборе схемы «bounded Gauss». Понятно, что в сошедшемся решении $\nabla\cdot\vec{V}=0$, поэтому такая замена полностью оправдана.

Схема интерполяции на грань величины является наиболее важной именно при вычислении конвективных слагаемых; как отмечалось в разделе 1.3.1, от этого во многом зависит точность решения, особенно в случае, когда вклад диффузионных слагаемых мал. В задаче о течении в каверне используется схема linear, т.е. линейная интерполяция, поскольку тут коэффициент вязкости достаточно велик (т.е. число Рейнольдса $Re=UL/\nu$ мало и течение ламинарное).

OpenFOAM предлагает множество вариантов интерполяционных схем (полный список здесь не приводится). Среди этого множества отметим важный класс схем с ограничителями. Необходимость введения ограничителей связана с тем, что в задачах, где допускается существование разрывных решений (это в первую очередь задачи газовой динамики и задачи, где диффузионные слагаемые отсутствуют либо очень малы), в области этих разрывов обычные линейные схемы повышенного порядка (выше первого) становятся в целом неустойчивы (точнее, теряют свойство монотонности[1], что утверждает известная теорема Годунова), и для предотвращения возникновения осцилляций необходимо в этих областях использовать схемы первого порядка. Для «переключения» порядка схемы как раз используются так называемые ограничители. Стандартный подход к введению ограничителя для какого-либо поля c можно условно записать следующим образом:

$$c_f=c_{1\mathrm{ord}}+\psi(r)\left(c_{\mathrm{HRS}}-c_{1\mathrm{ord}}\right), \tag{3.1}$$

где $c_{1\mathrm{ord}}$ — значение величины на грани, вычисленной по схеме первого порядка (first order), c_{HRS} — значение, полученное по схеме высокого порядка (High Resolution Scheme), $\psi(r)$ — функция ограничителя, которая зависит от аргумента r, представляющего собой некоторое отношение градиентов величины «до» и «после» грани. Таким образом, если $\psi=1$, то ограни-

[1] Свойство монотонности численной схемы означает невозможность самопроизвольного возникновения экстремумов

читель отсутствует, работает схема высокого порядка, в противном случае происходит «переключение» на схему пониженного порядка точности.

Существует большое разнообразие в методе построения таких «переключающихся» схем. Одним из классов является так называемые схемы TVD (Total Variation Diminishing, уменьшение полной вариации), которые локально могут не обеспечивать монотонность решения, однако, сохраняют её в некотором «интегральном» смысле (подробнее см. [11, 36, 13]). Здесь ограничитель вводится следующим образом:

$$c_f = c_U + \psi(r)\left(c_{\mathrm{CD}} - c_U\right), \tag{3.2}$$

где в качестве схемы повышенного порядка используется простая линейная интерполяция, дающая центрально-разностную аппроксимацию (CD — Central Difference), а функция $\psi(r)$ должна удовлетворять некоторым условиям. Распространённые функции ограничителя — minmod и ограничитель ван Лира (van Leer), для которых:

$$\psi_{\mathrm{minmod}} = \max(\min(r, 1), 0), \quad \psi_{\mathrm{vanLeer}} = \frac{r + |r|}{1 + |r|}.$$

Поскольку все TVD схемы обычно исходят из конечно-разностной дискретизации для структурированных сеток, обобщение их на неструктурированные сетки (а именно вычисление величины r) может осуществляться несколько по-разному. В OpenFOAM используется следующий способ:

$$r = 2\frac{(\nabla c)_U \cdot \overrightarrow{UD}}{c_D - c_U} - 1,$$

где обозначения введены в соответствии с разделом 1.3.1 (U — центр ячейки вверх по потоку, D — центр ячейки вниз по потоку). Отметим, что $\psi(r) = 0$ соответствует противопоточной схеме первого порядка, $\psi(r) = 1$ — центральной схеме, $\psi(r) = 2$ — схеме downwind (т.е. значение на грани берётся из центра ячейки ниже по потоку), $\psi(r) = r$ соответствует экстраполяции со вторым порядком точности по двум значениям выше по потоку в центр грани, а $\psi(r) = 2r$ — той же экстраполяции со вторым порядком точности по двум значениям выше по потоку, но в центр ячейки ниже по потоку. Одним из свойств, которому должны[1] удовлетворять ограничители, является симметричность, определяемая как:

[1] Не обязательно, но весьма желательно

$$r\psi\left(\frac{1}{r}\right) = \psi(r).$$

В таблице 3.1 приведены некоторые схемы, имеющиеся в OpenFOAM.

Таблица 3.1. Некоторые схемы интерполяции на грань в OpenFOAM

Именование и опции	Описание
`upwind`	Противопоточная схема первого порядка (1.15)
`linear`	Линейная интерполяция на грань
`midPoint`	Линейная интерполяция с весом 0.5: $(c_U + c_D)/2$
`linearUpwind grad(U)`	Противопоточная схема второго порядка, формула (1.17); опция grad(U) указывает, градиент какой величины надо использовать
`minmod`	TVD схема с ограничителем minmod
`vanLeer`	TVD схема с ограничителем ван Лира
`QUICK`	Схема Леонарда, формально обеспечивающая квадратичную интерполяцию на грань (т.е. схема третьего порядка; в OpenFOAM значение на грани лимитируется интервалом $[c_U; c_D]$)
`LUST grad(U)`	Взвешенная схема: 0.75 от linear и 0.25 от linearUpwind. Применяется для LES-расчётов турбулентных течений (Large Eddy Simulation, моделирование крупных вихрей)
`cubic`	Центральная схема четвёртого порядка
`harmonic`	Гармоническая интерполяция: $\frac{1}{c_f} = \frac{\beta}{c_P} + \frac{1-\beta}{c_N}$, где $\beta = \frac{Pf}{Pf+Nf}$, т.е. интерполяция идёт с обратным весом (больший вклад даёт значение из той ячейки, центр которой дальше от грани)
`reverseLinear`	Линейная интерполяция с обратным весом
`localMin/localMax`	Выбирается минимальное по модулю/максимальное значение из c_U и c_D
`downwind`	Берётся значение вниз по потоку c_D
`pointLinear`	Линейная интерполяция из значений в узлах грани, полученных путём интерполяции из центров ячеек

Таблица 3.1. (Продолжение)

Именование и опции	Описание
`outletStabilised scheme`	Везде используется схема scheme, за исключением выходных границ, на которых применяется upwind (для лучшей сходимости расчёта)
`limitWith scheme limiter`	Ограниченная схема по формуле (3.1): для вычисления c_{RHS} используется указанная пользователем схема scheme; ограничитель $\psi(r)$ указывается опцией limiter
`blended <psi>`	Взвешенная между linear и upwind схема с коэффициентом psi (т.е. по формуле (3.2) с постоянным $\psi(r)$, задаваемым пользователем)
`fixedBlended <psi> S1 S2`	Взвешенная между схемами S1 и S2 с постоянным коэффициентом psi
`cellCoBlended <C1> S1 <C2> S2`	Выбор из двух схем в соответствии с числом Куранта CFL: если CFL $<$ C1, то работает схема S1, если CFL $>$ C2, то S2, в промежутке используется взвешенная между S1 и S2 схема
`limiterBlended limiter S1 S2`	Взвешенная между схемами S1 и S2 с коэффициентом, рассчитываемым по формуле ограничителя limiter
`skewCorrected scheme`	К схеме scheme добавляется поправка на скошенность[1]

Отметим, что часть из представленных в таблице 3.1 схем имеют специальную версию для интерполяции векторного поля, в названии которой добавляется буква «V» в конце, например, vanLeerV. Такие схемы отличаются от обычных тем, что здесь ко всем компонентам вектора применяется один ограничитель, а не отдельные для каждой компоненты, при этом этот ограничитель рассчитывается исходя из направления наибольшего градиента. Такой способ даёт более устойчивую численную схему, но в ряде случаев менее точную.

Помимо этого, для скалярных величин существуют специальные схемы, используемые в случае, если значение скаляра не должно выходить за пределы интервала $[0; 1]$; такие схемы имеют в конце наименования «01»,

[1] В OpenFOAM под скошенностью понимается несовпадение центра грани и точки пересечения линии, соединяющей центры ячеек, с гранью

например, «vanLeer01», и если значение на грани, полученное по схеме vanLeer, выходит за указанный интервал, то применяется схема upwind. Если необходимо указать собственный интервал, за который не должно выходить значение скалярной величины, то для некоторых схем есть вариант с префиксом «limited»; например, «limitedVanLeer 0.1 0.2» означает, что если значение на грани, полученное по схеме vanLeer, выходит за интервал $[0.1; 0.2]$, то будет использоваться схема upwind.

3.1.4. Схемы вычисления градиента

Описание используемых схем для вычисления градиента величины осуществляется следующим образом:

```
gradSchemes
{
    <слагаемое> <метод> <интерполяция>;
    grad(p)     Gauss   linear;
}
```

В качестве метода здесь может выступать стандартная формула Грина-Гаусса (опция Gauss), для которого необходимо задать схему интерполяции на грань (выбирается из описанных в предыдущем разделе). Здесь также можно использовать противопоточные схемы, однако, в отличие от секции «divSchemes», необходимо указать переменную, в которой содержится значение расхода на грани (обычно она носит название «phi»). Пример использования противопоточной схемы linearUpwind при вычислении градиента давления представлен ниже.

```
    grad(p)     Gauss   linearUpwind phi grad(U);
```

Другие возможные варианты вычисления градиента перечислены ниже.

- leastSquares — метод наименьших квадратов (см. раздел 1.3.1);
- fourth — метод, при котором для получения итогового значения градиента используется коррекция при помощи полученного по методу наименьших квадратов значения (формально обеспечивает четвёртый порядок точности);

- pointCellsLeastSquares, edgeCellsLeastSquares — метод наименьших квадратов, где в качестве шаблона используются либо все центры ячеек, граничащие с текущей по узлам (pointCells), либо по рёбрам (edgeCells); отметим, что шаблон для стандартного метода наименьших квадратов включает центры только тех ячеек, которые имеют общую грань с текущей;

- faceLimited scheme <k> — версия схемы вычисления градиента по указанным выше схемам scheme с ограничителем: градиент корректируется таким образом, чтобы экстраполированное с его помощью на грань значение величины не выходило за границы максимального/минимального значений из центров прилегающих к грани ячеек; коэффициент k определяет допустимый интервал выхода за указанные пределы (k = 0 — коррекция вообще не применяется, k = 1 — коррекция при малейшем выходе за пределы);

- faceMDLimited scheme <k> — аналогично faceLimited, но для каждой компоненты градиента рассчитывается свой ограничитель;

- cellLimited, cellMDLimited — аналогично faceLimited и faceMDLimited, но максимальное/минимальное допустимое значение экстраполированной на грань величины ищется среди значений в центрах текущей ячейки и всех её соседей.

Более подробную информацию о схемах вычисления градиента можно найти в [12].

3.1.5. Аппроксимация временной производной

Схемы аппроксимации временной производной задаются в соответствующей секции как указано ниже.

```
ddtSchemes
{
    <слагаемое> <метод>;
    ddt(U)      Euler;
}
```

Все методы аппроксимации временной производной являются неявными[1]. OpenFOAM предлагает следующие схемы (см. также раздел 1.3.2):

- Euler — схема Эйлера (первый порядок точности);
- backward — трёхслойная схема «разностью назад» (второй порядок точности);
- CrankNicolson <psi> — схема Кранка-Николсон с параметром α (см. описание схемы в разделе 1.3.2), вычисляемым через параметр psi по формуле: $\alpha = 1/(1 + \psi)$; таким образом, psi = 1 соответствует схеме Кранка-Николсон второго порядка с $\alpha = 0.5$, psi = 0 — неявная схема Эйлера (Euler);
- bounded scheme — используется «ограниченная» версия схемы scheme: по аналогии со схемой bounded Gauss, описанной в разделе 3.1.3, вместо слагаемого $\partial(\rho c)/\partial t$ производится дискретизация следующего выражения, дающего неконсервативную форму временной производной: $\partial(\rho c)/\partial t - c\partial\rho/\partial t$; такая схема более устойчива в случае решения нестационарной задачи с переменной ρ, когда уравнение неразрывности в виде (1.2) не полностью сведено;
- steadyState — данная схема на самом деле означает решение стационарной задачи, т.е. вместо аппроксимации временной производной в уравнение записывается просто ноль;
- localEuler, CoEuler, SLTS — схемы для решения стационарной задачи путём введения производной по псевдовремени; шаг по псевдовремени выбирается в каждой ячейке исходя из: числа Куранта ячейки (CoEuler), параметра релаксации диагонального коэффициента alpha (SLTS) или определённого в коде программы поля rDeltaT (localEuler); для схем CoEuler и SLTS необходимо также в параметрах указать имена переменных с расходом через грань и плотности; например, для схемы CoEuler, где шаг по псевдовремени выбирается исходя из локального числа Куранта 0.3, запись будет выглядеть следующим образом:

[1] Точнее, явность или неявность метода решения конкретного уравнения определяется дискретизацией пространственных операторов: fvm (неявная) или fvc (явная)

```
ddt(U) CoEuler phi rho 0.3
```

Отметим, что в OpenFOAM также есть возможность решать уравнение со второй производной по времени (которая встречается, например, в волновом уравнении). Единственная доступная схема для дискретизации второй производной — схема Эйлера, в которой аппроксимация второй производной происходит по центральной конечно-разностной схеме, используя текущий и два предыдущих временных слоя, при этом остальные слагаемые уравнения берутся с текущего временного слоя. Поэтому такая аппроксимация даёт первый порядок точности по времени.

3.1.6. Параметры решателей СЛАУ

Перейдём теперь к рассмотрению параметров алгоритмов, задаваемых в файле `system/fvSolution`. Как следует из названия файла, в нём содержатся все опции для собственно решения имеющихся уравнений. Однако, прежде необходимо сделать несколько предварительных замечаний.

При дискретизации уравнений (по МКО или другими методами, в дифференциальной или интегральной форме) и, если необходимо, линеаризации, получается дискретная система уравнений относительно неизвестных, задаваемых (для сеточных методов) в некотором дискретном наборе точек пространства (определяемом вычислительной сеткой). Для рассматриваемого метода конечных объёмов на совмещённых сетках это система линейных алгебраических уравнений (СЛАУ) относительно неизвестных в центрах ячеек. Для замыкания получаемой при дискретизации СЛАУ необходимо добавить также граничные условия в дискретизированном виде (о чём более подробно будет говорить в следующем разделе), и в результате получаем СЛАУ вида:

$$Ax = b,$$

где A — матрица системы, b — правая часть, x — вектор неизвестных. Размерность данной СЛАУ (т.е. длина векторов x и b) обычно равна количеству ячеек сетки, и для реальных задач может составлять от нескольких десятков тысяч до десятков и даже сотен миллионов. Особенность данной СЛАУ состоит в том, что её матрица A (которая, очевидно, является квад-

ратной) является сильно разреженной: при дискретизации уравнения переноса (например, для скорости, концентрации, или «искусственного» уравнения для давления, как описано в разделе 1.3.4) входящие в него дифференциальные операторы аппроксимируются по относительно компактному шаблону: задействуются только значение в текущей ячейке и в нескольких её соседях. Таким образом, в каждой строке СЛАУ ненулевыми являются очень малое количество элементов. Например, при дискретизации уравнения Пуассона для двумерной задачи на равномерной прямоугольной структурированной сетке значение искомого поля в текущей ячейке «перевязано» только с четырьмя её соседями (как следует из конечно-объёмной или, что в данном случае то же самое, конечно-разностной аппроксимации оператора Лапласа), т.е. в каждой строке матрицы СЛАУ будет всего пять ненулевых элементов. Такая сильная разреженность матрицы при очень большой её размерности приводит к тому, что наиболее эффективными[1] методами решения СЛАУ становятся итерационные. Отметим, что часть программного кода, ответственная за решение СЛАУ, обычно носит название «линейный решатель» (linear solver).

Существует множество различных итерационных методов решения СЛАУ, и не все они одинаково «хороши» для той или иной задачи, поэтому зачастую (и в OpenFOAM также) имеется возможность выбора из нескольких. В данном пособии мы не будем подробно останавливаться на описании каждого метода из тех, что доступны в OpenFOAM; некоторые детали можно найти в [2, 13, 15]. Остановимся лишь на важных определениях и обозначениях. Но в качестве примера простейшего итерационного алгоритма приведём алгоритм метода Якоби, записываемый следующим образом:

$$Dx^{(k+1)} = b - (L + U)x^{(k)}.$$

Здесь $x^{(k+1)}$ — искомое решение на новой итерации, $x^{(k)}$ — известное решение на старой; за D обозначена матрица, представляющая собой диагональ матрицы A; L и U — соответственно, нижняя (или левая, left) и верхняя (upper, или правая) по отношению к диагонали части матрицы A, так что справедливо равенство:

[1] По сути, единственно возможными на практике

$$A = L + D + U.$$

Отметим, что процедуру вычисления нескольких итераций подобного итерационного алгоритма ещё также называют *сглаживанием*, поскольку она может применяться в некоторых случаях для сглаживания сильно осциллирующего в пространстве поля.

Невязкой СЛАУ называется норма следующего выражения:

$$\left\| Ax^{(k)} - b \right\|.$$

Для сходимости итераций необходимо, чтобы невязка системы стремилась к нулю при $k \to \infty$. Понятно, что практически невозможно проводить итерации «до бесконечности», поэтому их останавливают при достижении невязки некоторого заранее заданного малого значения.

Приведённое выше определение относится к понятию абсолютной невязки. Также существует понятие относительной невязки, рассчитываемой как:

$$\frac{\left\| Ax^{(k)} - b \right\|}{\|Ax^{(0)} - b\|}.$$

Скорость сходимости решения какого-либо итерационного метода зависит от нескольких факторов, одним из которых является число обусловленности матрицы, которое обычно определяется как модуль отношения максимального собственного числа к минимальному (по модулю) и меняется от 1 до ∞. Число обусловленности при численном решении СЛАУ определяет, насколько быстро сходятся итерации: чем выше это число, тем «хуже» сходятся итерации; в этом случае СЛАУ называют «плохо обусловленной». Отметим, что число обусловленности матрицы СЛАУ, полученной при дискретизации какого-либо уравнения переноса, обычно растёт с ростом размерности матрицы (т.е. при увеличении числа ячеек), и время решения СЛАУ итерационным методом также увеличивается. Для улучшения обусловленности СЛАУ используется так называемое *предобуславливание*, заключающееся в том, что СЛАУ домножают на некоторую матрицу-предобуславливатель M^{-1}, например, следующим образом[1]:

[1] В данном примере используется левое предобуславливание; возможно также применять и правое по формуле $AM^{-1}Mx = b$

$$M^{-1}Ax = M^{-1}b.$$

Здесь показатель степени -1 означает операцию получения обратной матрицы, т.е. $M^{-1}M = E$, где E — единичная матрица.

Чем «ближе» матрица M^{-1} к матрице A^{-1}, тем более предобусловленной становится СЛАУ и тем быстрее сводятся итерации. Простейшим примером является предобуславливатель Якоби $M^{-1} = D^{-1}$, где как и ранее D — матрица с диагональными элементами A.

Матрица системы может быть симметричной, т.е. $A^T = A$, где символ «T» означает транспонирование матрицы. Такая матрица, например, получается в случае дискретизации уравнения Пуассона для давления. Симметричность матрицы даёт некоторые преимущества при решении СЛАУ итерационным методом; в частности, число операций, необходимых для обращения симметричной матрицы (т.е. получения решения СЛАУ), может быть в два раза меньше, чем для несимметричной.

Ещё одним важным свойством матрицы является положительная (отрицательная) определённость. По определению, матрица является положительно (отрицательно) определённой, если все её собственные числа положительные (отрицательные)[1]. Отметим, что в основном итерационные методы решения СЛАУ предполагают положительную определённость матрицы и расходятся в случае, если это не так. Матрица называется *вырожденной*, если хотя бы одно её собственное число равно нулю; это также означает, что в СЛАУ имеются два (или более) линейно-зависимых уравнения и решение такой системы не является единственным. В частности, уравнение Пуассона для давления с условиями второго рода (нулевая производная по нормали на границе расчётной области) даёт как раз вырожденную матрицу: решение системы определено с точностью до постоянной. Для снятия вырождения необходимо добавить ещё одно линейно-независимое уравнение; обычно это осуществляется путём определения («привязки») давления в одной точке к какому-либо значению. Отметим, что условие диагонального преобладания в СЛАУ (т.е. для каждого уравнения в СЛАУ модуль

[1] Поскольку отрицательно определённая матрица превращается в положительно определённую путём умножения на -1, дальше будем говорить только о положительной определённости

диагонального элемента больше суммы модулей внедиагональных) обеспечивает положительную определённость матрицы.

Рассмотрим теперь содержимое файла system/fvSolution на примере задачи о движении жидкости в каверне $WM_PROJECT_DIR/tutorials/incompressible/icoFoam/cavity/cavity (как и ранее, стандартную «шапку» файла опускаем).

```
Файл cavity/system/fvSolution:
solvers
{
    p
    {
        solver          PCG;
        preconditioner  DIC;
        tolerance       1e-06;
        relTol          0.05;
    }
    pFinal
    {
        $p;
        relTol          0;
    }
    U
    {
        solver          smoothSolver;
        smoother        symGaussSeidel;
        tolerance       1e-05;
        relTol          0;
    }
}
PISO
{
    nCorrectors     2;
    nNonOrthogonalCorrectors 0;
    pRefCell        0;
    pRefValue       0;
}
```

Рассмотрим сначала секцию solvers. Здесь перечисляются все переменные, для которых решаются уравнения. Кроме того, поскольку используется метод PISO (см. раздел 1.3.4), уравнение Пуассона для давления мо-

жет решаться несколько раз; последний раз является «особым» (поскольку необходимо обеспечить хорошее выполнение уравнения неразрывности к следующему временному шагу), поэтому для него вводится отдельная подсекция, озаглавленная «pFinal». В этой подсекции есть строка с «$p;». Это означает, что используются значения всех параметров из подсекции «p», а последующая строка «relTol 0;» указывает, что значение параметра relTol необходимо переопределить нулём.

Параметры в секции «solvers» следующие:

- solver — используемый линейный решатель для СЛАУ; может принимать значения: для несимметричных матриц — smoothSolver (простые итерационные методы путём «сглаживания» решения), GAMG (алгебраический многосеточный метод с геометрической агломерацией), PBiCG (предобусловленный метод бисопряжённых градиентов), для симметричных — smoothSolver, GAMG, PCG (предобусловленный метод сопряжённых градиентов);

- smoother — метод «сглаживания» решения: GaussSeidel (метод Гаусса-Зейделя), nonBlockingGaussSeidel (специальная «неблокирующая» версия GaussSeidel при вычислении в параллельном режиме), symGaussSeidel (симметричный метод Гаусса-Зейделя), DILU (неполное LU-разложение с диагональным предобуславливанием; только для несимметричных матриц), DILUGaussSeidel (попеременное применение DILU и GaussSeidel для лучшего сглаживания DILU), DIC (неполное разложение Холецкого с диагональным предобуславливанием; только для симметричных матриц), DICGaussSeidel (аналогично DILUGaussSeidel, но для симметричных матриц), FDIC (более быстрая версия DIC);

- preconditioner — предобуславливатель; может принимать значения diagonal (предобуславливатель Якоби), DIC или FDIC (для симметричных матриц), GAMG, DILU (для несимметричных);

- tolerance — значение абсолютной невязки, при достижении которого итерации линейного решателя прекращаются;

- relTol — значение относительной невязки для завершения итераций; если установлено значение 0, то выход из итераций происходит только по абсолютной невязке; если установлено и relTol, и tolerance, то выход происходит тогда, когда будет достигнуто одно из двух значений.

Помимо указанных опций, существует также много других, причём наличие или отсутствие той или иной опции зависит также и от выбранного метода решения СЛАУ. В частности, метод GAMG является одним из самых эффективных для решения уравнений типа Пуассона; суть этого метода состоит в том, чтобы на основе исходной сетки создать ряд последовательно огрубляемых сеток (вплоть до сетки из нескольких ячеек) и далее некоторым образом «огрублять» систему до самого грубого уровня, затем последовательно решать СЛАУ на грубом уровне и далее «вверх» до исходной сетки, передавая решение с грубых уровней на «верхние» путём некой интерполяции. Опции, которые доступны при выборе метода GAMG, следующие: smoother (метод сглаживания при переходе от одного уровня к другому), nCellsInCoarsestLevel (примерное количество ячеек на самом грубом уровне) и другие. (подробнее см. [14]).

Общая стратегия выбора того или иного метода решения СЛАУ зависит, конечно, от задачи, однако, можно сделать несколько общих рекомендаций. Для решения уравнений Пуассона (в частности, для давления) в случае относительно простых задач с небольшой сеткой достаточно использовать метод PCG с предобуславливателем Холецкого (DIC). Для больших сеток лучшим выбором будет GAMG, при этом, как показывает практика, настройки по умолчанию и выбор GaussSeidel в качестве «сглаживателя» (smoother) в большинстве случаев оказываются достаточными. При решения уравнений переноса в сложных случаях достаточно использовать PBiCG, метод же GAMG здесь зачастую оказывается более «затратным», поскольку уравнения переноса обычно содержат производную по времени/псевдовремени (или, что аналогично, релаксацию), поэтому СЛАУ для них является хорошо обусловленной. Что касается использования smoothSolver, то этот метод применим только для небольших по размеру сеток и простых задач. Отметим, что выбор того или иного мето-

да решения СЛАУ сказывается в основном на скорости счёта, а не на его устойчивости в целом.

Более важным в ряде случаев оказывается задание точности решения: tolerance и relTol. Для стационарных задач, решаемых методом установления, очевидно, не имеет смысл слишком «хорошо» решать СЛАУ на каждой итерации/шаге по времени, поэтому обычно можно указать значение relTol равное 0.1. Для нестационарных же задач необходимо подбирать такие значения этих параметров, чтобы обеспечивать требуемую точность получаемого нестационарного решения; эта задача в целом является нетривиальной, поскольку влияние оказывает множество факторов (схема дискретизации по времени, шаг по времени, количество внутренних итерации и т.д.).

3.1.7. Параметры алгоритмов

Перейдём теперь к рассмотрению секции «PISO». Здесь указываются параметры метода решения уравнений Навье-Стокса для несжимаемой среды[1].

В OpenFOAM имеется три SIMPLE-подобных метода[2]: собственно SIMPLE (для стационарных задач), PISO (для нестационарных задач) и PIMPLE. Алгоритм PIMPLE является гибридным PISO/SIMPLE методом для нестационарных задач и позволяет держать шаг по времени больше, чем в методе PISO за счёт введения дополнительных внешних итераций по нелинейности. Параметры этих трёх методов описываются в файле `fvSolution` в секции с соответствующим названием (т.е. «PISO», «SIMPLE» или «PIMPLE»), и часть из них является общими для всех трёх методов. Рассмотрим некоторые из них:

- nNonOrthogonalCorrectors — количество итераций для поправки на неортогональность сетки при расчёте диффузионных слагаемых (см. раздел 1.3.1);
- momentumPredictor — может принимать значение on (включён) или off (выключен) и контролирует выполнение предикторного шага для

[1] Аналогичные методы (с небольшими модификациями) применяются в OpenFOAM и для расчёта низкоскоростного движения сжимаемых сред

[2] Подробнее о них рассказывается в разделе 1.3.4

получения предикторной скорости (как описано в разделе 1.3.4); в случае off уравнение для скорости не решается, а используется в некотором роде «явная» схема для получения предикторной скорости, что, естественно, уменьшает устойчивость метода; опция «off» используется только для расчёта течений с низким числом Рейнольдса и в некоторых случаях для многофазных течений по методу VOF (Volume of Fluid, объём жидкости);

- nCorrectors (для PISO и PIMPLE) — количество PISO-итераций (см. раздел 1.3.4); обычно задаётся равным 2 или 3;

- nOuterCcorrectors (для PIMPLE) — количество внешних итераций; каждая внешняя итерация включает в себя весь цикл решения по методу PISO (включая предикторное уравнение для скорости), поэтому время решения увеличивается пропорционально количеству этих внешних итераций и несмотря на то, что PIMPLE-алгоритм позволяет вести расчёт с большими шагами по времени, с т.з. общего времени расчёта бывает выгодней уменьшать шаг по времени и, соответственно, количество PIMPLE-итераций, иногда вплоть до 1; при этом метод PIMPLE становится эквивалентным методу PISO;

- consistent (для SIMPLE) — установка опции «consistent yes» переключает SIMPLE в режим SIMPLEC (см. раздел 1.3.4).

Отметим, что при решении методом SIMPLE в соответствующей секции с параметрами метода можно задать условия остановки расчёта по достижению заданного пользователем уровня, как показано в примере:

```
SIMPLE
{
    residualControl
    {
        p        1e-4;
        U        1e-4;
    }
}
```

Таким образом, расчёт будет идти до тех пор, пока на текущей итерации начальная невязка уравнений для скорости и давления не станет меньше значения 10^{-4}.

Аналогичные опции можно указать и для метода PIMPLE. В этом случае внешние итерации на каждом шаге по времени остановятся при достижении невязок указанных пользователем значений, как показано в примере:

```
PIMPLE
{
   residualControl
    {
        "(U|p)"
        {
            relTol          0;
            tolerance       0.0001;
        }
    }
}
```

Здесь используется возможность задавать шаблон в кавычках: строка «(U|p)» означает, что и для U, и для p используются одинаковые параметры. Кроме того, здесь можно указывать как значение относительной невязки relTol, так и абсолютной tolerance, при достижении которого произойдёт выход из внешних итераций метода PIMPLE.

В случае, если решается задача без выходных границ (на которых обычно указывается фиксированное давление), в соответствующей секции помимо опций метода необходимо также «назначить» точку привязки давления и значение давления в ней[1]. За это отвечают опции: pRefValue (значение давления «p» в точке «привязки»), pRefCell (номер ячейки, в которой фиксируется давление; под номером ячейки понимается её индекс в массиве ячеек, который строится на основании данных из `constant/polyMesh`; подробнее о структуре сетки в OpenFOAM см. 2.4.1), pRefPoint (вместо pRefCell; указываются непосредственно координаты точки «привязки» давления — в этом случае OpenFOAM сам определяет ячейку с фиксированным давлением).

[1]Это связано с тем, что уравнение Пуассона для давления в случае отсутствия выходных границ становится вырожденным, и его решение определено с точностью до аддитивной постоянной

Необходимо сделать несколько замечаний касательно «давления» в OpenFOAM. В случае решения задачи движения несжимаемой жидкости в уравнение входит только градиент давления, т.е. абсолютное значение давления не важно. Поэтому обычно давление отсчитывается от значения 0. Также, все решатели из класса incompressible (т.е. предназначенные для решения задач движения несжимаемой вязкой жидкости) оперирует на самом деле не с настоящим давлением, а с давлением, отнесённым к плотности, т.е. под «p» понимается p/ρ. В случае решения задач естественной конвекции (т.е. при наличии поля силы тяжести) абсолютное давление раскладывается на гидростатическое (вычисляемое по формуле $\rho g h$, где g — ускорение свободного падения, h — высота относительно некоторого уровня) и редуцированное, обозначаемое в OpenFOAM как «p_rgh»; уравнение Пуассона решается относительно редуцированного давления p_rgh, для которого и задаются параметры в соответствующих файлах[1].

Последнее, но немаловажное замечание касается задания релаксационных параметров, которые зачастую необходимы при решении задач SIMPLE-подобными алгоритмами; эти параметры указываются в секции relaxationFactors, как представлено в примере:

```
relaxationFactors
{
    fields
    {
        p           0.3;
    }
    equations
    {
        U           0.7;
    }
}
```

Здесь имеются две подсекции. Подсекция «fields» означает, что релаксация применяется непосредственно к переменной при её обновлении после решения уравнения (как указано в разделе 1.3.4 для давления), т.е. в данном случае новое значение давления будет рассчитываться как $0.3p^{new} + (1 - 0.3)p^{old}$, где p^{new} — значение, полученное после решения

[1] Подробнее о редуцированном давлении см. раздел 4.1.3

СЛАУ, а p^{old} — значение давления до решения СЛАУ (с предыдущей итерации).

В подсекции «equations» указываются параметры релаксации для уравнения, т.е. релаксация диагонального коэффициента (как указано в разделе 1.3.4 для скорости); в примере до решения СЛАУ диагональный коэффициент матрицы для U будет поделен на 0.7. Отметим. что существуют следующие варианты для значение релаксационного параметра α_{eqn} в подсекции «equations»:

- $0 < \alpha_{eqn} < 1$ — релаксация применятся обычным образом (т.е. так, как описано выше);
- $\alpha_{eqn} = 0$ — релаксация полностью отсутствует;
- $\alpha_{eqn} = 1$ — в случае, если отсутствует диагональное преобладание в матрице, диагональный коэффициент делается равным сумме модулей внедиагональных (такая специфическая релаксация требуется иногда для сложных уравнений с источниками, которые учитываются неявно).

В заключении данного раздела отметим, что здесь были указаны не все возможные опции, задаваемые в файле `fvSolution`; их набор определяется также и используемым решателем. Для получения представления о других возможных вариантах можно просмотреть примеры, поставляемые в дистрибутиве OpenFOAM. Также, в случае, если какой-то требуемый параметр не указан, решатель при запуске выдаёт сообщение с ошибкой и названием отсутствующей опции, по имени которой можно найти её описание (например, в руководстве пользователя или в Интернете).

3.2. Граничные условия в OpenFOAM

3.2.1. Предварительные замечания

До сих пор вопросу о граничных условиях уделялось довольно мало внимания, хотя их корректная постановка определяет успешность проведения расчётов. В данном разделе мы постараемся исправить сие досадное упущение. Но по традиции первноначально сделаем несколько важных замечаний.

Граничные условия вкупе с уравнениями составляют математическую постановку задачи. И именно граничные условия определяют решение. Тут следует сразу сделать разделение понятий: граничные и начальные условия. Граничные условия — это условия на переменные, для которых имеются уравнения, назначаемые на границе расчётной области, т.е. в пространстве. Начальные условия — это распределение полей переменных в начальный момент времени, т.е. постановка задачи Коши при решении нестационарных уравнений. Кроме того, для решения стационарных задач итерационными методами необходимо задавать начальные приближения (или другими словами начальные распределения). Понятно, что решение нестационарной задачи зависит от начальных условий, тогда как для стационарной задачи такая зависимость обычно отсутствует (если это не так, то решение стационарной задачи не является единственным, и такие случаи надо рассматривать особо). В данном разделе мы будем говорить только о граничных условиях в пространстве; что касается начальных условий (или в дальнейшем, инициализация/задание начального распределения), то здесь ситуация более простая; об этом будет немного подробнее говориться далее.

Граничные условия, накладываемые на переменные, обычно делятся на три типа: условие Дирихле (или условие первого рода — задано значение величины на границе), условие Неймана (или условие второго рода — задано значение производной величины на границе), условие третьего рода (линейная комбинация первых двух). Помимо этих относительно простых типов, естественно, существуют также и более сложные, например, нелинейные, интегральные или связывающие несколько неизвестных величин. Но с точки зрения численного расчёта в OpenFOAM все граничные условия так или иначе сводятся к первым трём (точнее даже, к условию третьего рода как к общему для всех трёх типов).

Понятно, что при численном расчёте необходимо также дискретизировать и граничные условия. Поскольку в OpenFOAM (как и в большинстве CFD кодов) конечный объём является ячейкой сетки, то граница расчётной области должна проходить по граням расчётных ячеек. Таким образом, граница расчётной области дискретизируется путём разбиения её поверхности на грани,

и граничное условие при дискретизации должно записываться для каждой граничной грани приграничной расчётной ячейки. Например, при решении уравнения переноса концентрации по МКО с условиями первого рода граничное значение концентрации должно задаваться в центрах граничных граней.

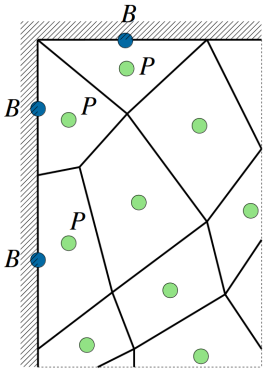

Рис. 3.1. Фрагмент расчётной сетки вблизи границы расчётной области

На рисунке 3.1 представлен фрагмент расчётной сетки вблизи границы расчётной области; буквами P обозначены центры приграничных конечных объёмов (в которых необходимо найти искомые величины из уравнений), а буквами B — центры граничных граней, на которых должно быть задано граничное условие. Подчеркнём, что для МКО необходимо задавать граничное условие именно на всей граничной грани каждого приграничного конечного объёма. В частности, при дискретизации диффузионного слагаемого для приграничной ячейки в результирующую формулу входит диффузионный поток на граничной грани, см. формулу (1.11). Для конвективного потока в предположении, что скорость несущей среды на границе задана, необходимо задавать значение концентрации на граничной грани, т.е. ставить условие первого рода.

Однако, задавать сразу два граничных условия (на диффузионный поток и на конвективный) нельзя, поскольку значение концентрации и градиента от неё связаны друг с другом. В этом случае ставится то граничное условие, которое отвечает физике процесса, а искомое значение (величины или её градиента) «восстанавливается». И здесь необходимо представлять, насколько точной является схема «восстановления» в используемом численном решателе. В частности, в случае задания условия первого рода необходимо «восстановить» градиент (а точнее, производную по нормали) от величины на граничной грани. Простейшая схема вычисления производной через разность $c_B - c_P$ очевидно даёт только первый порядок точности на любых сетках в случае нелинейного изменения величины при подходе к границе. К сожалению, построить схему повышенного порядка точности

для ячеек произвольного типа представляется весьма трудновыполнимой задачей (например, можно попытаться проэкстраполировать градиент из центра ячейки, но для этого требуется знать градиент градиента величины в центре ячейки, получение которого для приграничных ячеек нетривиально). Именно поэтому в OpenFOAM расчёт диффузионного слагаемого в приграничной ячейке осуществляется с первым порядком точности, что нужно иметь в виду при построении расчётной сетки. В принципе, в большинстве задач достаточно просто сильно измельчить сетку к границе для получения приемлемой точности, тем более, что при приближении к границе закон изменения величины обычно приближается к линейному.

Таким образом, в задачах, где диффузионные процессы вблизи границы играют важную роль, необходимо тщательно разрешать эти приграничные области. В задачах динамики вязкой жидкости на твёрдых границах расчётной области (стенках) физически оправданным является задание условия прилипания, т.е. нулевой скорости. Поскольку на некотором расстоянии от границы скорость отлична от нуля, вблизи стенки возникает область резкого падения скорости, где вязкость играет определяющую роль. Эта область носит название *пограничного слоя*. В большей части этого слоя изменение скорости нелинейно, именно поэтому для хорошего разрешения необходимо мельчить сетку вплоть до границы, где закон падения скорости до нуля близок к линейному. Отметим, что чем больше число Рейнольдса течения, тем тоньше пограничный слой и тем сильнее необходимо измельчать сетку. При этом изменение скорости (и других величин) в направлении поперёк стенки гораздо более сильное, чем вдоль. Именно поэтому в области пограничного слоя ячейки обычно являются сильно вытянутыми вдоль, соотношение размеров может достигать значений 1000 и более. Отметим, что помимо пограничного слоя для скорости (обычно называемого динамическим), в случае переноса других величин (температуры, концентрации и др.) возникают температурные, концентрационные и др. пограничные слои, причём их толщина может отличаться от толщины динамического пограничного слоя[1], что необходимо также учитывать при построении расчётной сетки.

[1]Это определяется числом Прандтля Pr для температурного и числом Шмидта Sc для концентрационного слоя: если Pr или Sc больше 1, то тоньше

Рассмотрим теперь вопрос корректной постановки граничных условий для задач динамики несжимаемой вязкой жидкости. Вообще, не для всякого уравнения необходимо задавать граничные условия на всей границе. В частности, если взять уравнение чисто конвективного переноса (без диффузии), то математически корректно задавать граничное условие только там, где происходит втекание в расчётную область. В этом случае, если использовать чисто противопоточную схему первого порядка при дискретизации уравнения, то значения величины на выходных границах действительно не нужны, однако, при использовании более сложных схем для «однородности» алгоритма обычно «доопределяют» значения на границах путём постановки так называемых «мягких» граничных условий, а именно, первая или вторая производная от величины полагается равной нулю (чем выше порядок производной, тем «мягче» условия, но сложнее реализация). Отметим, что не во всех задачах такие «мягкие» условия практически не оказывают влияния на решение; в частности, это относится к задачам газовой динамики, где на выходных границах следует ставить специальные «неотражающие» граничные условия. Вообще, в задачах с входными и выходными границами (или в задачах внешнего обтекания, когда все границы открыты для протока) зачастую необходимо исследовать вопрос их влияния на решение, например, задавая их на разном удалении от объекта исследований. Отметим, что в случае решения задач с диффузионными слагаемыми математически корректно задавать граничные условия на всех границах.

Важным является постановка граничных условий для давления при решении уравнений Навье-Стокса. Строго говоря, для давления *не требуется* задавать какие-либо граничные условия. Это связано с его ролью в уравнении баланса импульса (в которое входит только градиент давления): по сути, градиент давления играет роль «силы», которая подбирается во всех внутренних ячейках так, чтобы выполнить уравнение неразрывности. При этом, если граничные условия для скорости заданы так, что изначально интегральный расход через всю границу не равен нулю, то задача уже поставлена некорректно, и никакое поле давления «не исправит» ситуацию. Тем не менее, для численного решения задачи всё-таки необходимо зада-

вать какие-то граничные условия для давления[1]. Кроме того, есть задачи, где наоборот, задан перепад давления, а скорость «подбирается» исходя из этого перепада.

Резюмируя вышесказанное, отметим, что общей практикой является следующая постановка граничных условий для давления и скорости:

- На выходных границах задаётся постоянное давление, на скорость ставятся «мягкие» граничные условия. В ряде случаев необходимо задавать не постоянное давление на всей выходной границе, а среднее давление, при этом локально давление экстраполируется изнутри расчётной области, а затем «сдвигается» так, чтобы выполнить условие на среднее давление.

- На входных границах задаётся либо скорость, тогда на давление ставится «мягкие» граничные условия, либо давление, тогда скорость экстраполируется изнутри расчётной области. Иногда вместо давления (также называемого статическим) ставят условие на полное давление, которое для несжимаемой жидкости рассчитывается по формуле: $p_0 = p + \rho V^2/2$.

- На стенках для скорости ставится условие прилипания, а для давления — нулевая производная. Такое условие на давление следует из уравнений Навье-Стокса, а точнее, их упрощения для пограничного слоя, называемого уравнениями Прандтля: поперёк пограничного слоя давления практически постоянно.

3.2.2. Простейшие граничные условия в OpenFOAM

В OpenFOAM идеология задания граничных условий идёт от указанных выше трёх типов (1-го, 2-го и 3-го родов). При этом для каждой переменной, для которой решается уравнение, необходимо задавать граничные условия на всех границах. Синтаксис файла с граничными (и начальными) условиями, который является отдельным для каждой переменной, описывался в разделе 2.3.2; все граничные условия задаются в блоке описания граничных условий с именем boundaryField.

[1] Существуют подходы, в которых можно не задавать граничные условия для давления, например, при расчётах на MAC-сетках

Для задания условия первого рода используется синтаксис, указанный ниже:

```
boundaryField
{
    patchName
    {
        type            fixedValue;
        value           uniform 0;
    }
}
```

Опция type здесь имеет значение «fixedValue», т.е. «фиксированное значение». Само значение задаётся опцией «value». Отметим важный момент: опция «value» в принципе не связана с типом граничного условия; это просто ключевое слово, которое служит для хранения значений величины на данной границе patchName; для некоторых граничных условий его наличие обязательно, а для других, где значение на границе «восстанавливается» каким-либо образом, его можно опускать. Для типа «fixedValue» его наличие обязательно, значения на границе берутся из «value» и никогда не перезаписываются.

Синтаксис опции «value» следующий: ключевое слово «uniform» означает, что на всех гранях граничной зоны задаётся одно значение, которое идёт следом за словом «uniform» (в примере — 0). Ключевое слово «nonuniform» означает, что далее должен идти список отдельных значений для каждой грани; при этом порядковый номер значения в списке соответствует порядковому номеру грани в граничной зоне, который определяется форматом хранения расчётной сетки. Пример задания неоднородного распределения величины на границе представлен ниже:

```
    patchName
    {
        type            fixedValue;
        value nonuniform List<scalar>
        10(0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9);
    }
```

Здесь ключевое слово List означает, что далее идёт список значений типа «scalar», потом указана размерность списка (10), и затем в скобках —

значение для каждой грани. Отметим, что, конечно, вручную сформировать такой список довольно проблематично; если необходимо задать неоднородные значения на границе, есть другие способы это сделать (о чём чуть подробнее будет говориться в разделе 3.3.3).

Здесь представлены примеры задания скалярного значения. Для задания векторного значения надо вместо числа писать три компоненты вектора в скобках (а для тензора, соответственно, девять компонент), т.е. например для вектора

```
        value           uniform (1 0 0);
```

Для задания граничного условия второго рода применяются ключевые слова «zeroGradinet» (нулевой градиент величины по нормали к границе) и «fixedGradient» (заданный градиент). Для граничного условия «zeroGradient» более не требуется указывать никаких других опций, а для «fixedGradient» необходимо указать значение градиента при помощи ключевого слова «gradient», как показано на примере ниже:

```
        type            fixedGradient;
        gradient        uniform 0.1;
```

Очевидно, что fixedGradient со значением «gradient uniform 0» эквивалентно условию zeroGradient.

Смешанное условие третьего рода называется в OpenFOAM «mixed». В практических задачах именно это условие редко используется, однако, оно является «базовым» для многих других реально используемых граничных условий в OpenFOAM, поскольку представляет собой граничное условие общего вида в форме:

$$c_B = \alpha \bar{c}_B + (1-\alpha)\left(c_P + \overline{(\nabla c \cdot \vec{n})}_B |\overrightarrow{BP}|\right),$$

где величины с чертой сверху и параметр α задаются, c_B — значение на границе, c_P — значение в центре приграничной ячейки. Таким образом, при $\alpha = 1$ мы имеем граничное условие первого рода (fixedValue), а при $\alpha = 0$ — граничное условие второго рода (fixedGradient). В OpenFOAM данное граничное условие имеет следующий синтаксис:

```
    patchName
    {
```

```
    type            mixed;
    valueFraction   uniform 0.5;
    refValue        uniform 0;
    refGradient     uniform 0.1;
}
```

Здесь valueFraction — α, refValue — $\bar{c}_B$, refGradient — $\overline{(\nabla c \cdot \vec{n})}_B$.

Другими «базовыми» граничными условиями в OpenFOAM являются[1] (вид граничного условия должен соответствовать типу граничной зоны, см. раздел 2.4.1):

- empty — применяется в двумерных и одномерных задачах и означает, что данная граница лежит «в плоскости» рассматриваемой задачи;
- symmetryPlane — плоскость симметрии (только для плоских границ);
- symmetry — общее условие симметрии (в т.ч. и для неплоских поверхностей);
- wedge — аналогично empty, но для осесимметричной постановки (с одной ячейкой в окружном направлении);
- cyclic и cyclicAMI — для периодических (в т.ч. вращательной периодичности) граничных условий; первое используется при стыковке двух периодических поверхностей «грань-в-грань», а второе — для стыковки двух поверхностей с нестыкуемыми поверхностными сетками;
- calculated — означает, что значения на границе вычисляются в процессе расчёта самим решателем; обычно используется для тех переменных, для которых не решается никакое уравнение (но требуется определить граничное условие на некоторых границах; это необходимо, в частности, при использовании пристенных функций, см. раздел 4.2.3).

3.2.3. Специальные граничные условия

С точки зрения организации программного кода, все остальные, более специфические, граничные условия в OpenFOAM основываются на

[1] Здесь перечисляется только часть базовых граничных условий

простейших граничных условиях, перечисленных выше. Их можно найти в каталоге `$WM_PROJECT_DIR/src/finiteVolume/fields/fvPatchFields/derived`. Также полный список возможных граничных условий для данной переменной можно получить, задав для неё в поле «type» заведомо некорректное значение и запустив решатель. Отметим, что некоторые граничные условия применимы только для векторных или только для скалярных величин. Кроме того, в OpenFOAM имеется утилита foamList, при помощи которой можно получить весь список граничных условий для скаляров (запустив «foamList -scalarBCs») или векторов (командой «foamList -vectorBCs»).

Здесь мы не будем приводить полный список всех возможных граничных условий, а ограничимся описанием лишь некоторых из них.

- `inletOutlet` — переключающееся граничное условие: если поток втекает в расчётную область, то задаётся постоянное значение величины, определяемое параметром inletValue. В противном случае ставится условие zeroGradient.

- `outletInlet` — противоположное условию inletOutlet: для вытекающего потока задаётся постоянное значение величины, равное outletValue, для втекающего — zeroGradient.

- `totalPressure` — условие полного давления на входе. Используется совместно с pressureInletOutletVelocity для скорости. Если поток втекает в область, то давление рассчитывается как $p = p_0 - \rho V^2/2$ (для несжимаемой жидкости); в случае вытекания $p = p_0$. Значение полного давления p_0 задаётся параметром «p0».

- `pressureInletOutletVelocity` — для втекающего потока значение скорости на границе равно нормальной (к грани) компоненте скорости из центра прилежащей ячейки (опционально можно задать значение тангенциальной компоненты скорости на входе опцией tangentialVelocity). В случае вытекания ставится условие zeroGradient. Используется в задачах, где движения вызывается приложенным перепадом давления (например, совместно с условием totalPressure).

- `fixedFluxPressure` — необходимо задавать на стенках для давления в задачах, где присутствует объёмная сила (в частности, сила тяжести; соответствует zeroGradient в задачах без объёмных сил). Связано это с уравнением Пуассона для давления, в котором градиент давления по нормали к границе должен подбираться так, чтобы расход через грань соответствовал заданной на границе скорости.

- `uniformFixedValue` — нестационарное граничное условие: значение на границе в каждый момент времени задаётся исходя из значения параметра uniformValue. В частности, «uniformValue constant 0» даёт постоянное во времен значение 0. Другие возможные опции для uniformValue (подробное описание всех опций можно найти в руководстве пользователя [14]):
  - table ((0 1) (10 2)): задание кусочно-линейного изменения в виде таблицы прямо в файле с граничными условиями (в данном примере от момента времени 0 до 10 значение величины будет меняться линейно от 1 до 2);
  - tableFile: как table, но таблица считывается из внешнего файла;
  - csvFile: задание в виде таблицы из файла в формате csv (comma separated values);
  - sine: синусоидальное изменение величины;
  - square: задание меандра (периодического сигнала прямоугольной формы);
  - polynomial: задание изменения в виде полинома произвольной степени.

- `freestream` — условие «свободного потока» в задачах обтекания тел. Необходимо задать значение величины (например, скорости) freestreamValue. В случае втекания берётся значение из freestreamValue, при вытекании — zeroGradient.

- `freestreamPressure` — условие «свободного потока» для давления (используется совместно с freestream для скорости). Ставится zeroGradient с учётом расхода через границу (по аналогии с fixedFluxPressure).

- `waveTransmissive` — неотражающее граничное условие на выходе (для задач течения сжимаемого газа).

- `slip/noSlip` — условие проскальзывания/прилипания для скорости. Условие slip по сути эквивалентно симметрии (нормальная компонента нулевая, тангенциальная сносится из прилежащей ячейки), а noSlip — заданию нулевой скорости на стенке.

- `fixedProfile` — задание одномерного профиля величины на входе. Параметры профиля определяются примерно так же, как описано для uniformFixedValue (используется так называемая функциональность Function1).

- `fixedMean` — значения на границе экстраполируется изнутри расчётной области и сдвигаются так, чтобы осреднённое по зоне значение равнялось meanValue (meanValue может быть также функцией времени).

3.2.4. Примеры постановки граничных условий

Приведём несколько примеров постановок граничных условий для некоторых задач гидродинамики и теплообмена. Эти примеры взяты из постановок задач, имеющихся в дистрибутиве OpenFOAM и находящихся по адресу `$WM_PROJECT_DIR/tutorials`. Отметим, что возможность использования того или иного граничного условия может зависеть от используемого решателя.

Простейшая задача нестационарной теплопроводности в фланце была подробно рассмотрена в разделе 2.3. Для этой задачи использовались только два типа граничных условий: 1-го и 2-го рода (fixedValue и zeroGradient).

Аналогичные условия используются и в задаче движения несжимаемой вязкой жидкости в каверне, упоминавшейся в разделе 2.4.2; для этой задачи граничные условия на скорость U и давление p приведены в листинге.

```
Файл tutorials/incompressible/icoFoam/cavity/cavity/0/U:
boundaryField
{
    movingWall
    {
```

```
        type            fixedValue;
        value           uniform (1 0 0);
    }
    fixedWalls
    {
        type            noSlip;
    }
    frontAndBack
    {
        type            empty;
    }
}


Файл tutorials/incompressible/icoFoam/cavity/cavity/0/p:
boundaryField
{
    movingWall
    {
        type            zeroGradient;
    }
    fixedWalls
    {
        type            zeroGradient;
    }
    frontAndBack
    {
        type            empty;
    }
}
```

Граничная зона movingWall является движущейся стенкой, на ней задаётся постоянное значение скорости, направленной вдоль стенки. Зона fixedWalls представляет собой все остальные стенки; на них ставится условие прилипания. Задача двумерная, поэтому на границах, лежаших в плоскости задачи frontAndBack ставится условие empty. На всех стенках для давления ставится условие второго рода, поэтому при расчёте необходимо задавать точку привязки давления pRefCell и значение давления в ней pRefValue в файле system/fvSolution (подробнее об этом говорилось в разделе 3.1.7).

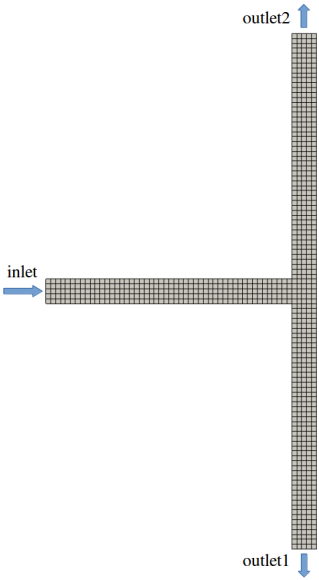

Рис. 3.2. К задаче о течении в Т-образном тройнике

В каталоге tutorials/incompressible/pimpleFoam/TJunction содержится постановка задачи о трёхмерном течении в Т-образном тройнике, расчётная сетка и геометрия для которого представлены на рисунке 3.2. Жидкость втекает слева через граничную зону «inlet», растекается и вытекает через зоны «outlet1» и «outlet2», причём течение происходит под действием нестационарного перепада давления: на входе задан кусочно-линейный профиль изменения полного давления при помощи граничного условия uniformTotalPressure (по аналогии с условием uniformFixedValue, описанным в предыдущем разделе). Сначала давление линейно меняется от 10 до 40 единиц[1] до момента времени 1, а затем держится постоянным и равным 40. Все остальные поверхности тройника являются стенками, на которых задаётся условие прилипания. Граничные условия для U и p приведены ниже.

```
Файл tutorials/incompressible/pimpleFoam/TJunction/0/U:
boundaryField
{
    inlet
    {
        type            pressureInletOutletVelocity;
        value           uniform (0 0 0);
    }
    outlet1
    {
        type            inletOutlet;
        inletValue      uniform (0 0 0);
```

[1] Напоминаем, что здесь под «давлением» понимается отношение p/ρ

```
        value           uniform (0 0 0);
    }
    outlet2
    {
        type            inletOutlet;
        inletValue      uniform (0 0 0);
        value           uniform (0 0 0);
    }
    defaultFaces
    {
        type            noSlip;
    }
}
```

Файл tutorials/incompressible/pimpleFoam/TJunction/0/p:

```
boundaryField
{
    inlet
    {
        type            uniformTotalPressure;
        p0              table
        (
            (0 10)
            (1 40)
        );
    }
    outlet1
    {
        type            fixedValue;
        value           uniform 10;
    }
    outlet2
    {
        type            fixedValue;
        value           uniform 0;
    }
    defaultFaces
    {
        type            zeroGradient;
    }
}
```

В каталоге tutorials/incompressible/simpleFoam/airFoil2D содержится постановка двумерной задачи внешнего обтекания крылового профиля, сетка и расчётная область для которой приведены на рисунке 3.3.

Внешние границы здесь имеют имена «inlet» и «outlet», стенка профиля обозначена как «walls». Граничные условия для скорости и давления приведены ниже. Как из них следует, внешний поток направлен слева под небольшим углом к оси x. Отметим, что в этой связи заранее сказать, где поток втекает в расчётную область, а откуда вытекает, проблематично, поэтому названия «inlet» и «outlet» довольно условны, и здесь используется граничное условие freestream, что для класса задач внешнего обтекания является общепринятым.

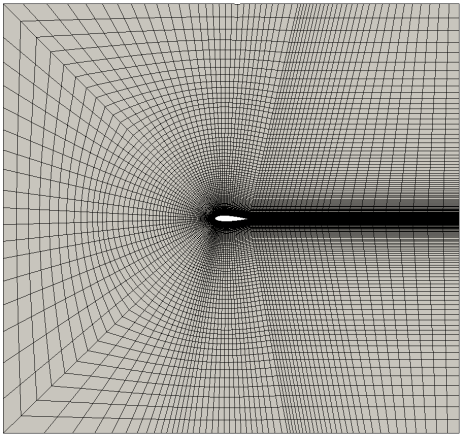

Рис. 3.3. Расчётная сетка для задачи обтекания профиля

```
Файл tutorials/incompressible/simpleFoam/airFoil2D/0/U:
boundaryField
{
    inlet
    {
        type            freestream;
        freestreamValue uniform (25.75 3.62 0);
    }
    outlet
```

```
    {
        type            freestream;
        freestreamValue uniform (25.75 3.62 0);
    }
    walls
    {
        type            noSlip;
    }
    frontAndBack
    {
        type            empty;
    }
}

Файл tutorials/incompressible/simpleFoam/airFoil2D/0/p:
boundaryField
{
    inlet
    {
        type            freestreamPressure;
    }
    outlet
    {
        type            freestreamPressure;
    }
    walls
    {
        type            zeroGradient;
    }
    frontAndBack
    {
        type            empty;
    }
}
```

Пример постановки задачи о естественной конвекции в поле силы тяжести в замкнутой области можно найти в каталоге tutorials/heatTransfer/buoyantSimpleFoam/hotRadiationRoom (см. рисунок 3.4). Конвекция в области возникает из-за наличия

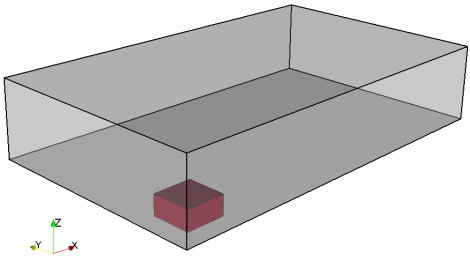

Рис. 3.4. Задача о естественной конвекции в помещении

обогреваемого «ящика» на полу помещения (обозначен красным цветом на рисунке; поверхность «ящика» имеет название «box»). Пол (floor) и потолок (ceiling) помещения поддерживаются при постоянной температуре 300 K, боковые стенки (fixedWalls) являются адиабатическими. Граничные условия для температуры T и редуцированного давления p_rgh представлены ниже (для скорости U на всех границах задаётся условие noSlip).

```
Файл tutorials/heatTransfer/buoyantSimpleFoam/hotRadiationRoom/0/T:
boundaryField
{
    floor
    {
        type            fixedValue;
        value           uniform 300.0;
    }
    ceiling
    {
        type            fixedValue;
        value           uniform 300.0;
    }
    fixedWalls
    {
        type            zeroGradient;
    }
    box
    {
        type            fixedValue;
        value           uniform 500.0;
    }
}

Файл tutorials/heatTransfer/buoyantSimpleFoam/hotRadiationRoom/0/p_rgh:
boundaryField
```

```
{
    floor
    {
        type            fixedFluxPressure;
        value           uniform 100000;
    }
    ceiling
    {
        type            fixedFluxPressure;
        value           uniform 100000;
    }
    fixedWalls
    {
        type            fixedFluxPressure;
        value           uniform 100000;
    }
    box
    {
        type            fixedFluxPressure;
        value           uniform 100000;
    }
}
```

3.3. Дополнительные возможности

3.3.1. Инициализация и пост-обработка данных средствами OpenFOAM

Начальное распределение какой-либо величины задаётся в том же файле, где и граничные условия (см. раздел 2.3.2), в секции «internalField». Синтаксис этой секции такой же, как и при задании, например, «value» в граничном условии: можно задать равномерное распределение, например:

```
internalField   uniform 300;
```

либо неравномерное распределение, тогда в этой секции записывается список значений для каждой ячейки расчётной сетки, например:

```
internalField   nonuniform List<vector>
4096
(
```

```
(-0.0376011 0.020584 -0.0051027)
(-0.0262359 0.0149309 -0.0048244)
(-0.0141003 0.00810973 -0.00427023)
...
```

Отметим важную особенность OpenFOAM: при выводе полей в процессе расчёта (в заданные пользователем моменты времени, см. раздел 2.3.3, файл `controlDict`), они сохраняются точно в таком же формате, как и при задании начальных и граничных условий. Поэтому расчёт можно без проблем запускать начиная с любого последующего момента времени, а также редактировать «выводные» файлы, например, в случае, если требуется изменить какие-то граничные условия и/или распределения величин в расчётной области в процессе проведения расчёта.

Как отмечалось, редактирования файлов полей «вручную» для задания неоднородных распределений весьма проблематично (а для реальных задач и вовсе невозможно на практике). OpenFOAM предлагает множество возможностей для задания этих неоднородных распределений, в частности, для инициализации начальных полей, о чём далее и будет идти речь.

Утилита, позволяющая задавать неоднородные распределения, называется setFields. Она считывает параметры из файла `system/setFieldsDict`. Пример этого файла приведён ниже.

```
Файл tutorials/compressible/rhoCentralFoam/shockTube/system/setFieldsDict:
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      setFieldsDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
defaultFieldValues
(
    volVectorFieldValue U (0 0 0)
    volScalarFieldValue T 348.432
    volScalarFieldValue p 100000
);
regions
```

```
(
    boxToCell
    {
        box (0 -1 -1) (5 1 1);
        fieldValues
        (
            volScalarFieldValue T 278.746
            volScalarFieldValue p 10000
        );
    }
);
```

Данный файл служит для задания начальных распределений в задаче об ударной трубе. Ударная труба представляется собой замкнутую прямолинейную трубу, разделённой перегородкой (диафрагмой) на камеры низкого и высокого давления. В начальный момент диафрагма разрывается и возникает ударная волна, движущаяся от области высокого давления в область низкого давления. Для численного решения такой задачи необходимо в начальный момент в одной части расчётной области (соответствующей камере высокого давления) задать высокое давление, а в другой — низкое. Именно это и выполняет утилита setFields с параметрами, приведёнными выше: секция «defaultFieldValues» служит для указания значений во всей области «по умолчанию». Это: нулевая скорость U, значения температуры T 348.432 K и давления p 10^5 Па. В секции «regions» перечисляются области, в которых необходимо проинициализировать поля другими значениями. В примере этой области соответствует камера низкого давления со значениями T 278.746 K и p 10^4 Па.

Ключевое слово «boxToCell» означает, что указанными далее значениями будут проинициализированы только те ячейки, центры которых попадают в прямоугольный параллелепипед с диагональю от точки с координатами $(0; -1; -1)$ до точки $(5; 1; 1)$. Помимо прямоугольного параллелепипеда можно задавать: цилиндр (cylinderToCell), кольцевой зазор (cylinderAnnulus), сферу (sphereToCell) и др. Кроме того, в OpenFOAM есть мощное средство для работы с сеткой — набор утилит topoSet, setSet и др., которые позволяет манипулировать (создавать, изменять, преобразовывать и т.п.) так называемые «множества ячеек» и «множества граней» (cellSet и faceSet), а также «зоны ячеек» и «зоны граней» (cellZone и

faceZone). Первые представляют собой просто именованные списки индексов ячеек/граней; вторые могут использоваться в солвере для каких-либо целей (например, для задания вращающейся части расчётной области или пористой области). «Зоны ячеек» могут быть, в частности, созданы в процессе работы утилиты blockMesh, см. пример tutorials/incompressible/porousSimpleFoam/angledDuctImplicit.

В OpenFOAM имеются инструменты для обработки данных, полученных в результате проведения расчёта; их исходные коды располагаются в каталоге $WM_PROJECT_DIR/applications/utilities/postProcessing. Все эти утилиты работают с полями, выведенными в процессе расчёта в каталоги с именем, соответствующим значению физического времени. По умолчанию, при запуске утилит они выполняют действия для всех таких каталогов. Можно указать, с каким моментом времени работать, при помощи опции «time»; кроме того, существует опция «-latestTime», которая означает, что будет использоваться данные для последнего момента времени, в который произошёл вывод полей.

Одна из полезных утилит — postProcess[1]. Она позволяет проводить различные операции над полем; формат запуска её следующий:

```
postProcess -func "<операция>"
```

В качестве «операции» может служить, например: div(U) — вычисление дивергенции скорости U и запись результата в файл; mag(U) — вычисление абсолютного значения (длины вектора) поля и т.п. Кроме того, можно запускать решатель с опцией «-postProcess»; в этом случае все требуемые данные автоматически подгружаются, если это требуется для проведения операции.

Другие операции утилиты postProcess — patchAverage и patchIntegrate. Первая возвращает среднее значение величины на выбранной граничной зоне, а вторая — интеграл от этой величины по поверхности зоны. Операция «probes» позволяет вычислять значение величины в указанной пользователем точке (путём интерполяции в неё из центров ячеек). Для вычисления напряжения трения на стенке служит операция «wallShearStress», а для вычисления теплового потока — утилита wallHeatFlux (в будущих вы-

[1] Заменяет многие отдельные утилиты из предыдущих версий OpenFOAM, в частности, foamCalc

пусках OpenFOAM также войдёт в состав postProcess). Чтобы получить весь список доступных операций утилиты postProcess, достаточно выполнить команду:

```
postProcess -list
```

Помимо указанных операций, postProcess также позволяет делать так называемые «выборки» данных (samples) при помощи операции «sample». Подробнее об этом будет сказано в следующем разделе.

3.3.2. Обработка данных в ходе расчёта

Перейдём теперь к вопросу о выводе различных данных в процессе работы решателя. При проведении нестационарного расчёта зачастую требуется выводить информацию о значениях некоторых величин в конкретных точках (т.н. точках мониторинга). Эта информация, в частности, позволяет делать вывод о получении установившегося нестационарного режима, о том, является ли этот режим нестационарным, и если да, то каковы характерные частоты колебаний. Последнее может быть использовано для определения необходимого для хорошего разрешения колебаний шага по времени (эмпирически установлено, что для схем второго порядка точности по времени приемлемую точность расчёта обеспечивает около 50 точек на период колебаний).

Задание точек мониторнига в OpenFOAM осуществляется в файле system/controlDict в подсекции probes секции functions, как показано в примере (взят из упоминавшейся ранее задачи о тройнике, см. tutorials/incompressible/pimpleFoam/TJunction/system/controlDict).

```
functions
{
    probes
    {
        // Where to load it from
        libs ( "libsampling.so" );

        type            probes;

        // Name of the directory for probe data
        name            probes;
```

```
        // Write at same frequency as fields
        writeControl    writeTime;
        writeInterval   1;

        // Fields to be probed
        fields ( p U );
        probeLocations
        (
            (1e-06 0 0.01 )       // at inlet
            (0.21 -0.20999 0.01)  // at outlet1
            (0.21 0.20999 0.01)   // at outlet2
            (0.21 0 0.01)         // at central block
        );
    }
}
```

Местоположения точек указывается в «probeLocations», а величины, которые должны в этих точках выводится — в «fields». Здесь параметр writeControl задан как «writeTime», что означает вывод точек мониторинга в те же моменты, что и вывод всех полей (параметр writeInterval задаёт частоту вывода, 1 — каждый раз, 2 — через раз и т.д.). Другой вариант — timeStep — означает вывод на каждом шаге по времени (с частотой writeInterval).

OpenFOAM позволяет также делать «выборки» данных (samples) в процессе расчёта. Под «выборкой» здесь понимается некоторый «срез» расчётной области поверхностью, на которую интерполируются значения полевых величин. Для создания среза в файле system/controlDict в секции functions надо указать подсекцию с произвольным именем (например, «cuttingPlane», «surfaceSample» и др.) и указать следующий тип: «type surfaces». Примеры таких срезов указаны в листинге (взяты из tutorials/incompressible/pisoFoam/les/motorBike/motorBike/system/controlDict и tutorials/incompressible/pisoFoam/les/pitzDaily/system/controlDict).

```
functions
{
    surfaceSampling
    {
```

```
    // Sample near-wall velocity
    type surfaces;

    // Where to load it from (if not already in solver)
    libs            ("libsampling.so");
    writeControl    writeTime;

    interpolationScheme cellPoint;
    surfaceFormat vtk;

    // Fields to be sampled
    fields ( U );

    surfaces
    (
        nearWall
        {
            type            patchInternalField;
            patches         ( lowerWall );
            distance        1E-6;
            interpolate     true;
            triangulate     false;
        }
    );
}

cuttingPlane
{
    type            surfaces;
    libs ( "libsampling.so" );
    writeControl    writeTime;
    surfaceFormat   vtk;
    fields          ( p U );
    interpolationScheme cellPoint;
    surfaces
    (
        yNormal
        {
            type cuttingPlane;
            planeType pointAndNormal;
            pointAndNormalDict
            {
```

```
                basePoint (0 0 0);
                normalVector (0 1 0);
            }
            interpolate true;
        }
    );
}
}
```

Здесь в подсекции с именем «surfaceSampling» создаётся срез поверхностью с именем «nearWall», образованной путём отступа от граничной зоны «lowerWall» внутрь расчётной области на расстояние distance, равное 10^{-6}. В подсекции «cuttingPlane» создаётся срез плоскостью. Плоскость имеет имя «yNormal» и определяется своей нормалью (параметр normalVector) и точкой, лежащей в плоскости (параметр basePoint). В обоих примерах вывод полей на указанных поверхностях осуществляется каждый раз при выводе всех основных полей (параметр writeControl установлен в writeTime), формат вывода (surfaceFormat) — VTK («родной» формат ParaView). Поля, которые необходимо выводить, указываются параметром fields.

В OpenFOAM имеется также возможность вычислять и выводить осреднённые во времени поля величин, как указано в примере (tutorials/incompressible/pimpleFoam/channel395/system/controlDict):

```
functions
{
    fieldAverage1
    {
        type            fieldAverage;
        libs            ("libfieldFunctionObjects.so");
        writeControl    writeTime;

        fields
        (
            U
            {
                mean        on;
                prime2Mean  on;
                base        time;
            }
            p
```

```
            {
                mean        on;
                prime2Mean  on;
                base        time;
            }
        );
    }
}
```

Здесь «fieldAverage1» — имя, задаваемое пользователем; «type fieldAverage» указывает на то, что данная секция отвечает за осреднение. Осреднённые величины выводятся в те же каталоги, что и основные поля (т.е. с именем, представляющим собой значение физического времени), но с суффиксом «Mean», например «pMean», «UMean» и т.д. Также могут выводится среднеквадратичные отклонения величин (суффикс «prime2Mean»). Что выводить, а что нет, регламентируется параметрами «mean» и «prime2Mean» (on — выводить, «off» — нет). Можно осреднять по времени (параметр «base» со значениям «time»), либо по итерациям (значение «iteration»).

В заключение отметим наличие опции для вывода силовых характеристик, действующих на выбранные пользователем граничные зоны. Для этого создаётся секция с типом «type forceCoeffs» и далее указывается, что именно рассчитывать (силы и их моменты). Примеры задания подобной секции можно найти в tutorials (например, tutorials/incompressible/pisoFoam/les/motorBike/motorBike/system/controlDict) и здесь не приводятся.

3.3.3. Библиотека расширений swak4foam

Пакет OpenFOAM — это мощный инструмент, позволяющий при помощи программирования собственных модулей создавать совершенно различные инструменты для широкого класса задач вычислительной физики. Однако, зачастую необходимо иметь возможность добавлять некоторую функциональность для решения той или иной задачи без того, чтобы редактировать код (тем более, что не всегда это достаточно просто для неискушённого в программировании пользователя). Именно для этого и существуют сторонние библиотеки, расширяющие функционал OpenFOAM и обеспечивающие

достаточно простой для пользователя «интерфейс». Наиболее распространённой и «мощной» библиотекой является swak4foam, она сочетает в себе функциональность ранее развивавшихся независимо библиотек groovyBC и funkySetFields.

Библиотека swak4foam позволяет пользователю создавать собственные, зачастую сложные граничные условия (groovyBC) и инициализировать поля величин (funkySetFields) при помощи специально разработанного «языка», на котором записываются различные выражения, содержащие арифметические и другие операции с полями. Кроме того, библиотека добавляет множество так называемых функциональных объектов — функций, которые выполняются в процессе работы решателя, аналогично, например, выводу точек мониторинга и осреднению полей из предыдущего раздела.

Недостатком сторонних библиотек (и swak4foam тоже) является то, что зачастую при выходе новой (мажорной) версии OpenFOAM последние версии библиотек перестают с ней работать, поэтому приходится либо использовать старые версии, либо ждать, пока разработчики сторонних библиотек выпустят обновление. В частности, в настоящий момент библиотека swak4foam (версии 0.4.0) работает только с OpenFOAM до версии 3.0 включительно, ведётся активная работа по обеспечению поддержки 4-ой версии OpenFOAM. Более подробно о версиях и совместимости можно найти на официальной странице проекта https://openfoamwiki.net/index.php/Contrib/swak4Foam.

Здесь мы не будем подробно останавливаться на описании библиотеки swak4foam, приведём лишь несколько примеров. На листинге ниже представлен пример задания в качестве граничного условия на входной границе inlet осциллирующего параболического профиля скорости.

```
inlet
{
    type            groovyBC;
    variables (
       "yp=pts().y;minY=min(yp);maxY=max(yp);"
       "v=(pos().y-maxY)*(pos().y-minY)/(0.25*pow(maxY-minY,2))*normal();"
    );
    valueExpression "(1+0.5*sin(500*time()))*v";
}
```

Здесь в качестве типа граничного условия необходимо указать «groovyBC». В секции variables можно создать любые (корректные) переменные на основе определённых в OpenFOAM и groovyBC полей. Например, «yp» — это переменная, задаваемая как набор y-координат узлов граничной зоны при помощи операции «yp=pts().y»; «minY» и «maxY» — минимальное и максимальное значения y-координаты. Параболический профиль задаётся при помощи определения переменной «v» при помощи умножения вектора нормали (на каждой грани граничной зоны) normal() на амплитуду, вычисляемую по формуле «v=(pos().y-maxY)*(pos().y-minY)/(0.25*pow(maxY-minY,2))*normal()», где pos().y — y-координата центра каждой грани граничной зоны inlet (в формуле учтено, что нормаль на границе всегда направлена из расчётной области). Непосредсвенное задание величины на границе происходит при помощи определения в valueExpression. Здесь определённый ранее профиль скорости «v» домножается на временной множитель с синусом.

Для работы утилиты funkySetFileds, предназначенной для инициализации полей, требуется создать файл system/funkySetFieldsDict (можно обойтись и без него, указав необходимые параметры прямо при запуске утилиты, см. примеры https://openfoamwiki.net/index.php/Contrib/funkySetFields). Пример файла funkySetFieldsDict приведён ниже.

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      setFieldsDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
expressions
(
    circleVel
    {
        field U;
        expression "(grad(rdist(vector(0.05,0.05,0)))^vector(0,0,-1))*
                            rdist(vector(0.05,0.05,0))/0.05";
    }
```

```
    pressure1
    {
        field p;
        expression "10.*(0.1-pos().y)";
    }
);
```

Первая подсекция «circelVel» служит для задания поля скорости, направленного по кругу от центра, задаваемого вектором «vector(0.05,0.05,0)», с линейным ростом модуля скорости при удалении от этого центра (символ «^» является обозначением векторного произведения). Подсекция «pressure1» служит для задания линейного по y распределения поля давления «p».

Множество других примеров применения groovyBC, funkySetFields и пр. можно найти в подкаталоге `Examples` основного каталога библиотеки swak4foam.

4. Практикум по решению задач гидродинамики и теплообмена в OpenFOAM

В данном разделе подробно рассматривается процесс поэтапной постановки и запуска некоторых задач гидродинамики и теплообмена, как ламинарных, так и турбулентных.

4.1. Ламинарные течения

4.1.1. Конвективный перенос скаляра однородным потоком

Данная задача является одной из «канонических» для тестирования численных схем вычисления конвективных потоков, которые в рамках метода конечных объёмов в конечном итоге сводятся к заданию схемы интерполирования величины на грань расчётной ячейки. Как отмечалось в разделах 1.3.1, 3.1.3, схема интерполяции на грань при вычислении конвективных потоков является главным фактором, влияющим на точность и корректность получаемого решения в случае, когда вклад диффузионных слагаемых достаточно мал.

Математическая постановка задачи предельно проста: рассматривается уравнение чисто конвективного переноса величины T в виде:

$$\frac{\partial T}{\partial t} + \nabla \cdot \left(\vec{V} T\right) = 0,$$

где поле скорости полагается заданным и постоянным: $\vec{V} = const$. Аналитическое решение данного уравнения очевидно: оно описывает распространение T вдоль линий тока $\vec{V}$ от заданного в начальный момент времени распределения с учётом граничных условий.

В численной постановке задачи будем рассматривать область в виде квадрата 1×1 м^2, см. рисунок 4.1, скорость направлена вдоль его диагонали. На части левой и нижней границы (длиной 0.2 м) задаётся постоянное значение c равное 1, на остальной части — значение 0; верхняя и

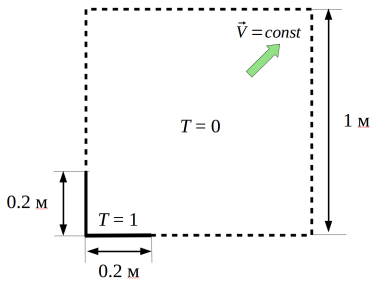

Рис. 4.1. Расчётная область для задачи конвективного переноса

правая границы являются выходными, на них можно задавать условие нулевой производной. В начальный момент времени внутри области задаётся $T = 0$.

Для решения данной задачи в OpenFOAM можно использовать решатель scalarTransportFoam, предназначенный для решения нестационарного уравнения конвективно-диффузионного переноса скаляра в заданном поле скорости, «занулив» коэффициент диффузии. Для подготовки постановки задачи следует перейти в пользовательский каталог OpenFOAM и создать каталог задачи, например, путём клонирования примера для scalarTransportFoam при помощи утилиты foamCloneCase:

```
run; foamCloneCase $FOAM_TUTORIALS/basic/scalarTransportFoam/pitzDaily
    ↪ convectiveTransport
```

Данная команда создаст каталог `$FOAM_RUN/convectiveTransport` и скопирует туда все файлы из соответствующего примера; далее будут описаны необходимые модификации этих файлов для получения постановки решаемой задачи.

Для построения равномерной структурированной сетки используем утилиту blockMesh; файл `system/blockMeshDict` с параметрами сетки представлен ниже.

```
Файл system/blockMeshDict:
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      blockMeshDict;
}
convertToMeters 1;
vertices
(
```

```
    (0 0 0)
    (1 0 0)
    (1 1 0)
    (0 1 0)
    (0 0 0.1)
    (1 0 0.1)
    (1 1 0.1)
    (0 1 0.1)
);
blocks ( hex (0 1 2 3 4 5 6 7) (50 50 1) simpleGrading (1 1 1) );
boundary
(
    boundTopRight
    {
        type patch;
        faces ( (3 7 6 2) (2 6 5 1) );
    }
    boundBottomLeft
    {
        type patch;
        faces ( (1 5 4 0) (0 4 7 3) );
    }
    frontAndBack
    {
        type empty;
        faces ( (0 3 2 1) (4 5 6 7) );
    }
);
```

Сетка имеет размер 50×50 ячеек; заданы три граничные зоны: boundTopRight содержит верхнюю и правую границы расчётной области, boundBottomLeft — нижнюю и левую; зона frontAndBack как и всегда при решении двумерных задач имеет тип empty.

Для решения задачи чисто конвективного переноса скаляра необходимо занулить коэффициент диффузии в файле `constant/transportProperties` как показано ниже.

```
Файл constant/transportProperties:
FoamFile
{
    version     2.0;
```

```
    format      ascii;
    class       dictionary;
    location    "constant";
    object      transportProperties;
}
DT              DT [0 2 -1 0 0 0 0] 0.0;
```

Отметим, что в решателе scalarTransportFoam в качестве скаляра выступает температура T, именно поэтому коэффициент диффузии обозначен здесь как «DT» (коэффициент температуропроводности).

Следующий шаг — задание поля скорости $\vec{V}$ и граничных и начальных условий для скаляра T. Поле скорости задаётся в файле `0/U` следующим образом:

```
Файл 0/U:
FoamFile
{
    version     2.0;
    format      ascii;
    class       volVectorField;
    object      U;
}
dimensions      [0 1 -1 0 0 0 0];
internalField   uniform (1 1 0);
boundaryField
{
    "bound.*"
    {
        type            fixedValue;
        value           $internalField;
    }
    frontAndBack { type empty; }
}
```

Здесь для задания граничных значений использовано так называемое регулярное выражение: вместо перечисления двух границ boundTopRight и boundBottomLeft задаётся одно граничное условие для всех имён, совпадающих с шаблоном «bound.*», где «.*» означает любые символы. В качестве значения value задаётся «$internalField» — это означает, что берётся значение, определяемое в секции internalField, т.е. «uniform (1 1 0)».

Для задания граничных условий для поля T необходимо воспользоваться утилитой setFields, поскольку на части границы boundBottomLeft необходимо задать одно значение (например, 1), а на оставшейся — 0. Для этого необходимо создать «промежуточный» файл `0/T` со следующим содержимым:

```
Промежуточный файл 0/T:
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    location    "0";
    object      T;
}
dimensions      [0 0 0 1 0 0 0];
internalField   uniform 0;
boundaryField
{
    boundTopRight
    {
        type            zeroGradient;
    }
    boundBottomLeft
    {
        type            fixedValue;
        value           uniform 0;
    }
    frontAndBack { type empty; }
}
```

Здесь внутренние ячейки проинициализированы значением 0, на верхней и правой границах расчётной области поставлено условие нулевой производной, а на нижней и левой границах — фиксированное значение 0. Для того, чтобы на части границы boundBottomLeft задать ненулевое значение, необходимо создать файл `system/setFieldsDict` со следующим содержимым:

```
Файл system/setFieldsDict:
FoamFile
{
```

```
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      setFieldsDict;
}
regions
(
    // Set patch values (using ==)
    boxToFace
    {
        box (-0.1 -0.1 -1) (0.2 0.2 1);
        fieldValues ( volScalarFieldValue T 1 );
    }
);
```

Секция boxToFace указывает на то, что на всех гранях ячеек, попавших в прямоугольный параллелепипед с координатами двух точек на диагонали (опция box), необходимо установить значение скаляра «T» равным 1. При запуске утилиты setFields будет выдано сообщение, что на граничных гранях значение установлено, а на внутренних гранях значение игнорируется (поскольку у переменной класса volScalarField нет понятия «значение на внутренней грани ячейки»). Примерный вывод в процессе работы утилиты представлен ниже.

```
Часть вывода в процессе работы setFields:
Create time
Create mesh for time = 0
Reading setFieldsDict
Setting field region values
    Adding faces with centre within boxes 1((-0.1 -0.1 -1) (0.2 0.2 1))
    Setting patchField values of volScalarField T
--> FOAM Warning :
    From function bool setFaceFieldType(const Foam::word&, const Foam::
    ↪ fvMesh&, const labelList&, Foam::Istream&) [with Type = double;
    ↪ Foam::labelList = Foam::List<int>]
    in file setFields.C at line 263
    Ignoring internal face 0. Suppressing further warnings.
    On patch boundBottomLeft set 20 values
    On patch frontAndBack set 200 values
End
```

Отметим, что на зонах с граничным условием «empty» или «zeroGradient» значение также не устанавливается, несмотря на то, что в процессе работы утилиты она выводит сообщение «On patch frontAndBack set 200 values».

Несмотря на то, что задача решается как нестационарная, в данном случае нас будет интересовать не переходный процесс, а установившееся решение. Поэтому для аппроксимации временной производной используется неявная схема Эйлера, а шаг по времени и число шагов выбирается достаточно произвольно. Параметры численной схемы, задаваемые в файле system/fvSolution, можно оставить такие же, как в примере, с которого скопирована постановка задачи. Содержимое файлов system/controlDict и system/fvSolution приведено ниже. Указанное значение deltaT обеспечивает значение числа Куранта 0.5.

```
Файл system/controlDict:
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      controlDict;
}
application     scalarTransportFoam;
startFrom       startTime;
startTime       0;
stopAt          endTime;
endTime         1;
deltaT          0.005;
writeControl    runTime;
writeInterval   1;

Файл system/fvSolution:
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      fvSolution;
```

```
}
solvers
{
    T
    {
        solver          PBiCG;
        preconditioner  DILU;
        tolerance       1e-06;
        relTol          0;
    }
}
SIMPLE { nNonOrthogonalCorrectors 0; }
```

Поскольку данная задача предназначена для тестирования влияния схемы вычисления конвективного слагаемого, будем рассматривать несколько разных схем, задаваемых в секции divSchemes файла system/fvSchemes. Рассмотрим сначала результаты расчёта при использовании противопоточной схемы первого порядка upwind. Содержимой файла system/fvSchemes для этого случая приведено ниже.

```
Файл system/fvSchemes:
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      fvSchemes;
}
ddtSchemes { default Euler; }
gradSchemes { default Gauss linear; }
laplacianSchemes { laplacian(DT,T) Gauss linear orthogonal; }
divSchemes
{
    default         none;
    div(phi,T)      Gauss upwind;
}
```

Распределение поля T в момент времени 1, полученное в результате расчёта при помощи решателя scalarTransportFoam с указанными параметрами показано на рисунке 4.2. Из-за большой численной диффузии, присущей

схеме upwind, происходит «размывание» поля T в перпендикулярном потоку направлении, что, очевидно, противоречит аналитическому решению, в котором T должно сохраняться вдоль линий тока.

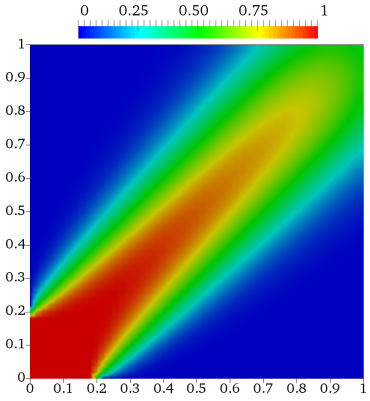

Рис. 4.2. Распределение T при использовании схемы upwind

Таким образом, в данной задаче для получения более корректного решения необходимо использовать другие схемы интерполяции на грань при дискретизации конвективного слагаемого. Для сравнения результатов были проведены также расчёты со следующими схемами из таблицы 3.1: linear, QUICK и vanLeer. Для сравнения результатов строились распределения T вдоль оси x при $y = 0.5$ (см. рисунок 4.3), которые были получены при помощи фильтра «Plot Over Line» в ParaView.

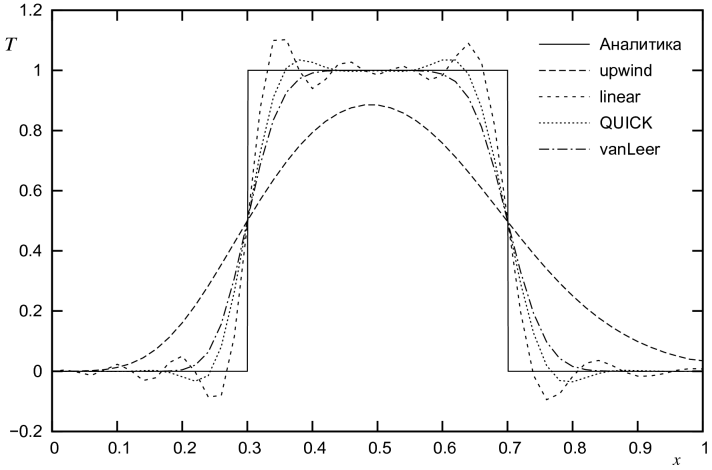

Рис. 4.3. Распределение T вдоль x при $y = 0.5$: влияние схемы интерполяции

Как и ожидалось, схема linear даёт осциллирующее решение (поскольку является абсолютно неустойчивой). Схема QUICK устойчива, но немонотонна — вблизи разрыва возникают нефизические осцилляции. Лучший результат даёт схема vanLeer: решение монотонное, а уровень численной диффузии заметно ниже, чем у схемы upwind, причём при использовании схемы upwind «размытие» за счёт численной диффузии усиливается вниз по потоку (как следует из рисунка 4.2), тогда как схема vanLeer не даёт такого эффекта. Однако, при увеличении числа Куранта (т.е. шага по времени) до 1 расчёт со схемой vanLeer перестаёт сходиться, тогда как схема upwind даёт устойчивый счёт при любом числе Куранта.

4.1.2. Течение в начальном участке плоского канала

Задача о развитом ламинарном течении в бесконечном плоском канале имеет аналитическое решение [1]; здесь мы рассмотрим задачу об установлении течения в канале конечной длины в случае задания на входе равномерного профиля скорости; на установившемся участке профиль скорости должен быть параболическим [1]. Постановка задачи весьма проста (рисунок 4.4): расчётная область представляет собой прямоугольник, сторона AB является входной, противоположная (CD) — выходной, нижняя и верхняя стороны (BC и AD) — стенки. На входе в канал задаётся скорость $U_{вх}$. В качестве критерия подобия здесь выступает число Рейнольдса $Re = U_{вх}H/\nu$, где H — высота канала. Чтобы течение было ламинарным и стационарным, необходимо выбирать значение числа Рейнольдса меньше примерно 1500, при этом чем меньше число Рейнольдса, тем быстрее устанавливается параболический профиль скорости; в данной постановке будем рассматривать течение при $Re = 400$.

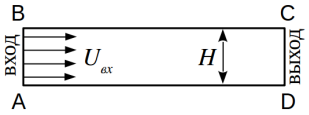

Рис. 4.4. Постановка задачи о течении в плоском канале

Для оценки длины переходного участка L можно использовать следующую эмпирическую зависимость:

$$\frac{L}{H} = 0.04Re. \tag{4.1}$$

Для $Re = 400$ длина переходного участка примерно равна 16; в расчёте длина канала (размер сторон AD и BC) берётся $40H$ для возможности более точного определения длины переходного участка и минимизации влияния выходной границы на решения.

Задача решается в безразмерной постановке, поэтому высоту канала берётся равной $H = 1$, скорость на входе $U_{вх} = 1$, тогда значение кинематического коэффициента вязкости следует положить равным $\nu = 0.0025$. Используется сгущённая к стенкам и ко входу в канал расчётная сетка, параметры которой приведены ниже.

```
Файл system/blockMeshDict:
convertToMeters 1;
vertices
(
    (0 0 0)
    (40 0 0)
    (40 1 0)
    (0 1 0)
    (0 0 1)
    (40 0 1)
    (40 1 1)
    (0 1 1)
);
blocks ( hex (0 1 2 3 4 5 6 7) (100 20 1) simpleGrading (10 ((0.5 0.5 4)
    ↪ (0.5 0.5 0.4)) 1) );
boundary
(
    inlet
    {
       type patch;
       faces ( (0 3 4 7) );
    }
    outlet
    {
       type patch;
       faces ( (1 2 5 6) );
    }
    walls
    {
       type wall;
```

```
        faces ( (0 1 5 4) (2 3 7 6) );
    }
    fb
    {
        type empty;
        faces ( (0 1 2 3) (4 5 6 7) );
    }
);
```

В качестве решателя используется простейший решатель icoFoam, предназначенный для решения нестационарных ламинарных задач движения несжимаемой жидкости. Таким образом, данную задачу будем решать методом установления по времени.

Значение кинематического коэффициента турбулентной вязкости задаётся в файле constant/transportProperties как показано ниже.

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "constant";
    object      transportProperties;
}
nu              [0 2 -1 0 0 0 0] 0.0025;
```

Параметры численных схем следует взять из примера tutorials/incompressible/icoFoam/cavity/cavity/, в котором рассматривается течение в каверне с движущейся крышкой. Обратим внимание, что при вычислении конвективных слагаемых используется линейная интерполяция (в данном случае это возможно, поскольку число Рейнольдса достаточно мало и используется относительно грубая сетка):

```
Часть файла system/fvSchemes:
divSchemes
{
    default         none;
    div(phi,U)      Gauss linear;
}
```

Шаг по времени следует выбирать исходя из значения числа Куранта (которое выводится в процессе расчёта), которое желательно должно быть меньше 1. В данной постановке шаг по времени равен 0.05. Для получения стационарного решения достаточно просчитать до момента времени 40; файл `system/controlDict` для данного случая приведён ниже.

```
Часть файла system/controlDict:
application     icoFoam;
startFrom       startTime;
startTime       0;
stopAt          endTime;
endTime         40;
deltaT          0.05;
writeControl    runTime;
writeInterval   10;
```

На рисунке 4.5 представлено распределение модуля скорости вблизи входа в канал. Видно, что от входа развиваются пограничные слои, что приводит к увеличению скорости в центре и перестроению профиля скорости.

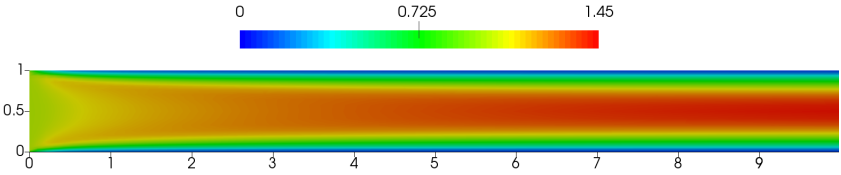

Рис. 4.5. Распределение модуля скорости вблизи входа

Рассчитаем длину начального участка по распределению скорости на оси вдоль x (рисунок 4.6). Видно, что скорость практически выходит на установившееся значение к концу канала, однако это значение отличается от аналитического 1.5 примерно на 1% (в расчёте получается значение 1.489). Если в качестве длины начального участка взять абсциссу, при которой скорость отличается от скорости на выходе на 1%, то получится значение длины переходного участка L равное примерно 18, что довольно близко к значению, даваемому формулой (4.1).

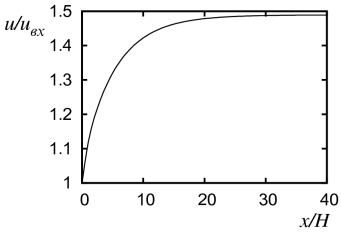

Рис. 4.6. Скорость на оси канала

Расхождение с аналитическим решением связано с тем, что в OpenFOAM вязкое трение на стенках рассчитывается с первым порядком точности; если бы трение рассчитывалось со вторым порядком точности, можно было бы ожидать получение профиля, в точности соответствующего аналитическому, т.е. параболе Пуазейля. Данное расхождение также иллюстрируется рисунком 4.7, где представлены установившийся профиль скорости и аналитическое решение:

$$\frac{u}{U_{вх}} = 6\frac{y}{H}\left(1 - \frac{y}{H}\right).$$

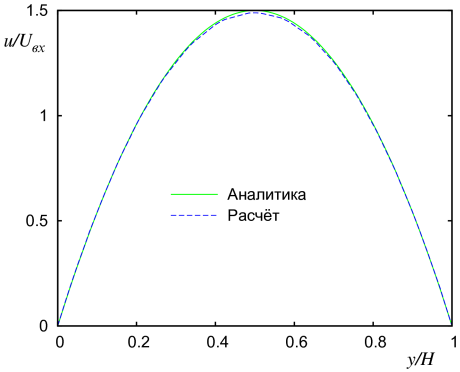

Рис. 4.7. Профиль скорости на участке установившегося течения

Аналогичное небольшое расхождение наблюдается и для градиента давления, который на установившемся участке должен быть постоянным и равным:

$$\frac{\Delta p}{\rho l} = -12\frac{\nu U_{вх}}{H^2},$$

т.е. для рассматриваемой задачи -0.3. В расчёте получается значение -0.298, что также примерно на 1% ниже аналитического значения.

4.1.3. Свободная конвекция в квадратной полости

Задача о свободной конвекции в квадратной полости с разнонагретыми вертикальными стенками является одним из канонических тестов для программных кодов, используемых при расчёте термоконвективных течений, и для неё в литературе имеется подробная информация о структуре и свойствах течения в широком диапазоне определяющих параметров (например, см. [16]).

В рассматриваемой задаче моделируется двумерное стационарное ламинарное течение в квадратной полости (рисунок 4.8), температура правой и левой вертикальных стенок соответственно равна T_h и T_c ($T_h > T_c$), горизонтальные стенки теплоизолированы. Ускорение силы тяжести $\vec{g}$ направлено в сторону, противоположную оси y. Задача решается в приближении Буссинеска, при котором эффект изменения плотности (за счёт изменения температуры среды) учитывается в уравнениях Навье-Стокса только в слагаемом с силой тяжести, что приводит к появлению силы плавучести (или иначе силы Архимеда), подробнее см., например, [17]. Таким образом, уравнения неизотермического нестационарного движения несжимаемой жидкости в поле силы тяжести в приближении Буссинеска записываются в следующем виде:

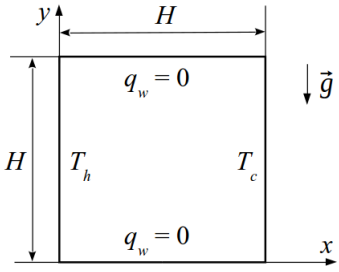

Рис. 4.8. Квадратная полость с разнонагретыми вертикальными стенками

$$\nabla \cdot \vec{V} = 0,$$

$$\frac{\partial \vec{V}}{\partial t} + \nabla \cdot \left(\vec{V}\vec{V}\right) = -\nabla p_{rgh} - \beta\left(T - T_{ref}\right)\vec{g} + \nu\nabla^2\vec{V},$$

$$\frac{\partial T}{\partial t} + \nabla \cdot \left(\vec{V}T\right) = D_T\nabla^2 T.$$

Здесь кинематический коэффициент вязкости ν, коэффициент температуропроводности D_T и коэффициент объёмного теплового расширения β являются постоянными, T_{ref} — некоторое характерное (reference) значение

температуры, $p_{rgh} = p/\rho - gh$ — редуцированное давление (отнесённое к плотности), h — расстояние в направлении $\vec{g}$ (в рассматриваемой задаче $h = -y$).

В OpenFOAM для численного решения представленной выше системы уравнений имеются два решателя: buoyantBoussinesqPimpleFoam (нестационарные задачи) и buoyantBoussinesqSimpleFoam (стационарные). Именно последний и будет использоваться для решения стационарной задачи свободной конвекции в квадратной полости.

Прежде чем приступить к описанию постановки представленной задачи в OpenFOAM, необходимо определить, какие физические параметры следует задать. Известно [17], что в задачах свободной конвекции критериями подобия являются число Грасгофа Gr (или число Рэлея Ra) и число Прандтля Pr, которые определяются следующим образом:

$$Gr = \frac{g\beta\Delta T H^3}{\nu^2}, \quad Pr = \frac{\nu}{D_T}, \quad Ra = Gr \cdot Pr,$$

где ΔT — характерный перепад температур, который в рассматриваемой задаче равен $\Delta T = T_h - T_c$. Таким образом, несмотря на то что в OpenFOAM все задачи решаются в размерной постановке, в данном случае можно выбирать физические параметры произвольным образом, обеспечивая заданные значения критериев подобия. Известно [16], что для воздуха ($Pr = 0.7$) вплоть до чисел $Ra = 10^6$ наблюдается устойчивый стационарный режим конвекции в квадратной полости.

Будем рассматривать постановку задачи для $Pr = 0.7$ и $Ra = 10^4$. Возьмём размер расчётной области H равным 0.1 м. Будем использовать равномерную расчётную сетку размером 40×40 ячеек; файл `system/blockMeshDict` для этого случая приведён ниже.

```
Часть файла system/blockMeshDict:
convertToMeters 0.1;
vertices
(
    (0 0 0)
    (1 0 0)
    (1 1 0)
    (0 1 0)
    (0 0 0.1)
```

```
    (1 0 0.1)
    (1 1 0.1)
    (0 1 0.1)
);
blocks ( hex (0 1 2 3 4 5 6 7) (40 40 1) simpleGrading (1 1 1) );
boundary (
    w_left
    {
        type wall;
        faces ( (0 3 7 4) );
    }
    w_right
    {
        type wall;
        faces ( (1 2 6 5) );
    }
    w_top_bottom
    {
        type wall;
        faces ( (3 7 6 2) (0 4 5 1) );
    }
    frontAndBack
    {
        type empty;
        faces ( (0 1 2 3) (4 5 6 7) );
    }
);
```

Физические параметры среды задаются в файле constant/transportProperties, а вектор ускорения свободного падения $\vec{g}$ — в файле constant/g. Ниже представлены примеры этих файлов, где параметры ν, β и $\vec{g}$ подобраны таким образом, чтобы обеспечить значение числа $Ra = 10^4$ при условии, что перепад температур задаётся равным $\Delta T = 1$. Значение T_{ref} может быть выбрано достаточно произвольным образом (его выбор влияет по сути только на скорость сходимости) и в данном случае берётся равным $0.5(T_c + T_h)$, где $T_c = 300\,\text{K}$, $T_h = 301\,\text{K}$.

```
Файл constant/transportProperties:
FoamFile
{
    version     2.0;
```

```
    format      ascii;
    class       dictionary;
    object      transportProperties;
}
transportModel Newtonian;
nu              [0 2 -1 0 0 0 0] 1e-05;    // Laminar viscosity
beta            [0 0 0 -1 0 0 0] 0.001428; // Thermal expansion coefficient
TRef            [0 0 0 1 0 0 0]  300.5;    // Reference temperature
Pr              [0 0 0 0 0 0 0]  0.7;      // Laminar Prandtl number
Prt             [0 0 0 0 0 0 0]  0.9;      // Turbulent Prandtl number

Файл constant/g:
FoamFile
{
    version     2.0;
    format      ascii;
    class       uniformDimensionedVectorField;
    location    "constant";
    object      g;
}
dimensions      [0 1 -2 0 0 0 0];
value           (0 -1 0);
```

Отметим, что решатель buoyantBoussinesqSimpleFoam предназначен как для решения ламинарных задач, так и турбулентных при помощи RANS-подхода, поэтому в файле `constant/transportProperties` также необходимо указывать значение турбулентного числа Прандтля Pr_t, которое в данной задаче не используется. Тип задачи (ламинарный или турбулентный) задаётся в файле `constant/turbulenceProperties`, который в рассматриваемом случае выглядит как представлено ниже.

```
Файл constant/turbulenceProperties:
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "constant";
    object      turbulenceProperties;
}
simulationType laminar;
```

В решателе buoyantBoussinesqSimpleFoam решаются уравнения для скорости $\vec{V}$, температуры T и редуцированного давления p_{rgh}, поэтому необходимо задать граничные условия для всех этих величин. Кроме того, решатель buoyantBoussinesqSimpleFoam требует наличия файла `alphat`, в котором должны содержаться граничные условия для турбулентного коэффициента теплоотдачи α_t, поскольку, как говорилось выше, этот решатель предназначен и для решения турбулентных задач. Подробнее о турбулентных характеристиках и граничных условиях для них будет рассказываться в разделе 4.2. В данном случае можно задать значение этого коэффициента равным нулю. Таким образом, граничные условия для всех требуемых полей задаются как представлено ниже.

```
Часть файла 0/T:
dimensions      [0 0 0 1 0 0 0];
internalField   uniform 300;
boundaryField {
    w_left
    {
        type            fixedValue;
        value           uniform 301;
    }
    w_right
    {
        type            fixedValue;
        value           uniform 300;
    }
    w_top_bottom
    {
        type            zeroGradient;
    }
    frontAndBack { type empty; }
}

Часть файла 0/U:
dimensions      [0 1 -1 0 0 0 0];
internalField   uniform (0 0 0);
boundaryField {
    "w_.*"
    {
        type            noSlip;
```

```
    }
    frontAndBack { type empty; }
}

Часть файла 0/p_rgh:
dimensions      [0 2 -2 0 0 0 0];
internalField   uniform 0;
boundaryField {
    "w_.*"
    {
        type            fixedFluxPressure;
        rho             rhok;
        value           uniform 0;
    }
    frontAndBack { type empty; }
}

Часть файла 0/alphat:
dimensions      [0 2 -1 0 0 0 0];
internalField   uniform 0;
boundaryField {
    "w_.*"
    {
        type            zeroGradient;
    }
    frontAndBack { type empty; }
}
```

Для задания параметров численных схем в файле fvSchemes за основу можно взять значения из соответствующего файла примера tutorials/heatTransfer/buoyantBoussinesqSimpleFoam/hotRoom. Ниже приведена секция divSchemes, которая была изменена для получения более точного решения за счёт использования схемы QUICK.

```
Часть файла system/fvSchemes:
divSchemes
{
    default         Gauss linear;
    div(phi,U)      bounded Gauss QUICKV;
    div(phi,T)      bounded Gauss QUICK;
}
```

В приведённом ниже файле system/fvSolution указаны оптимальные для быстрого получения решения параметры. Отметим, что в данной задаче основной интерес представляет теплоотдача от стенок (помимо получения самих полей переменных), т.е. значение теплового потока, и для получения установившегося значения теплового потока (которое, очевидно, должно быть одинаковым для левой и правой стенок в сошедшемся решении) необходимо сводить уравнения с хорошей точностью, поэтому абсолютную точность решения СЛАУ необходимо задавать достаточно малой (здесь задано значение 10^{-10}).

```
Часть файла system/fvSolution:
solvers
{
    p_rgh
    {
        solver          PCG;
        preconditioner  DIC;
        tolerance       1e-10;
        relTol          0.1;
    }
    "(U|T)"
    {
        solver          PBiCG;
        preconditioner  DILU;
        tolerance       1e-10;
        relTol          0.1;
    }
}
SIMPLE
{
    nNonOrthogonalCorrectors 0;
    pRefCell        0;
    pRefValue       0;
}
relaxationFactors
{
    fields { p_rgh 0.7; }
    equations
    {
        U               0.5;
```

```
        T               0.9;
    }
}
```

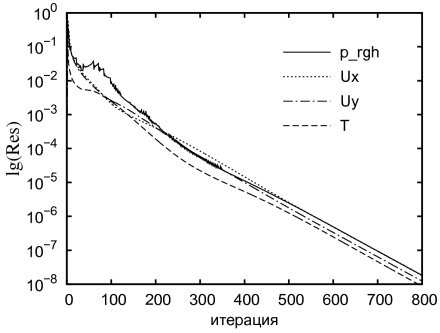

Рис. 4.9. История сходимости (логарифм начальной невязки уравнений в зависимости от номера итерации)

Для получения сошедшегося решения при указанных параметрах достаточно сделать 1000 итераций. История сходимости (полученная при помощи утилиты foamLog) приведена на рисунке 4.9.

После нескольких сотен итераций зависимость невязки от итерации становится практически линейной (в логарифмическом масштабе); при достаточном увеличении числа итераций невязка падает до значений, соответствующих машинной точности представления чисел с плавающей точкой при использовании двойной точности (порядка 10^{-14}).

В качестве иллюстрации рассматриваемого свободноконвективного течения в полости на рисунке 4.10 представлены распределение температуры и диний тока. В полости наблюдается единственный вихрь (конвективная ячейка), а также видно формирование вблизи вертикальных стенок температурных пограничных слоёв. Отметим, что при увеличении числа Рэлея толщина этих слоёв уменьшается, так что в центре полости устанавливается примерно одинаковая температура, равная полусумме температур стенок.

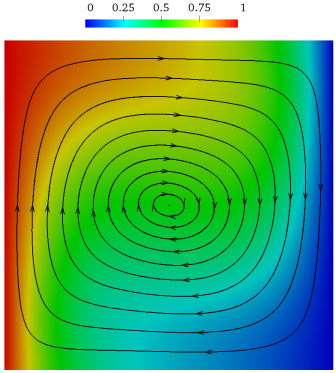

Рис. 4.10. Распределение $T - T_c$ и линии тока в полости

Один из наиболее важных вопросов (с практической точки зрения) — это насколько интенсифицируется теплоотдача за счёт возникающего движения или, другими словами, каково значение теплового потока на изотермических стенках. В качестве характеристики этого теплового потока выступает безразмерная величина — число Нуссельта, рассчитываемое через перепад температур ΔT, высоту области H, коэффициент теплопроводности и средний тепловой поток на левой или правой стенке q_w (измеряется в Вт/м^2) по формуле:

$$Nu = \frac{q_w H}{\lambda \Delta T}, \tag{4.2}$$

где q_w — средний тепловой поток на левой или правой стенке (измеряется в Вт/м^2) , λ — коэффициент теплопроводности.

Для вычисления числа Нуссельта в OpenFOAM можно воспользоваться утилитой wallHeatFlux, которая вычисляет мощность тепла (измеряемого в Вт), поступающего (или уходящего) с каждой граничной зоны в соответствии с определением:

$$Q_w = -\int \lambda \nabla T \cdot \vec{n} dS,$$

где для каждой граничной зоны интеграл берётся по всей её поверхности, $\vec{n}$ — внутренняя нормаль.

Для работы утилиты wallHeatFlux необходим файл `constant/thermophysicalProperties`, в котором задаются теплофизические свойства среды[1]. Отметим, что входящий в вышеприведённые формулы коэффициент теплопроводности λ после подстановки $q_w = Q_w/S$ (S — площадь поверхности) в формулу для Nu сокращается, т.е. при вычислении Q_w с помощью wallHeatFlux не обязательно задавать «правильное» значение λ. Поэтому файл `constant/thermophysicalProperties` может выглядеть следующим образом:

```
Файл constant/thermophysicalProperties:
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
```

[1] Этот файл также используется в ряде решателей OpenFOAM

```
    location    "constant";
    object      thermophysicalProperties;
}
thermoType
{
    type            heRhoThermo;
    mixture         pureMixture;
    transport       const;
    thermo          hConst;
    equationOfState rhoConst;
    specie          specie;
    energy          sensibleEnthalpy;
}
mixture
{
    equationOfState
    {
        rho             1;
    }
    specie
    {
        nMoles          1;
        molWeight       1;
    }
    thermodynamics
    {
        Cp              100;
        Hf              0;
    }
    transport
    {
        mu              1;
        Pr              1;
    }
}
```

В этом файле присутствуют некоторые параметры, которые не используются утилитой wallHeatFlux, но должны быть указаны, а именно: секция species, параметры rho и Hf. Остальные параметры, требуемые для вычисления λ по формуле $\lambda = \mu C_p/Pr$ при работе утилиты wallHeatFlux, подобраны таким образом, чтобы в результате сразу получалось число Нус-

сельта (с учётом того, что в расчёте $H = 0.1$, площадь боковой стороны каверны получается равной $0.1 \times 0.01 = 10^{-3}$, именно поэтому исходя из (4.2) значение C_p задано равным 100).

Запуск утилиты для вычисления числа Нуссельта после 1000 итераций решателя осуществляется следующим образом:

```
wallHeatFlux -time 1000
```

Вывод приведён ниже.

```
Часть вывода утилиты wallHeatFlux:
Create time
Create mesh for time = 1000
Time = 1000
Selecting thermodynamics package
{
    type            heRhoThermo;
    mixture         pureMixture;
    transport       const;
    thermo          hConst;
    equationOfState rhoConst;
    specie          specie;
    energy          sensibleEnthalpy;
}
Reading/calculating face flux field phi
Selecting turbulence model type laminar
Wall heat fluxes [W]
w_left
    convective: 2.25648
    radiative:  -0
    total:      2.25648
w_right
    convective: -2.25648
    radiative:  -0
    total:      -2.25648
w_top_bottom
    convective: 0
    radiative:  -0
    total:      0
End
```

Утилита wallHeatFlux разделяет суммарную тепловую мощность (total) на конвективную составляющую (convective; это тепло, поступающее за счёт теплопроводности и/или переноса средой) и радиационную (radiative), которая в данной задаче, естественно, равна нулю. Положительное значение Nu =2.25648 на левой границе w_left означает, что тепло поступает в полость, тогда как отрицательное на правой (w_right) означает отток тепла из полости в стенку. В сошедшемся решении эти два потока должны совпадать по модулю, что и наблюдается после расчёта 1000 итераций. Например, после 500 итераций числа Нуссельта на левой и правой границах отличаются друг от друга на ~0.03%. Отметим также, что в результате работы утилиты wallHeatFlux в каталоге с соответствующим номером итерации появляется файл с именем `wallHeatFlux`, содержащий распределения теплового потока на каждой граничной зоне.

При использовании равномерной сетки 40×40 ячеек отличие полученного числа Нуссельта $Nu = 2.25648$ от «эталонного» $Nu_{ref} = 2.245$ из [16] составляет 0.5%. Расчёт на измельчённой в два раза по каждому направлению сетке (т.е. 80×80 ячеек) даёт после 4000 итераций значение $Nu = 2.2471$, которое отличается от эталонного на 0.09%. Отметим, что при измельчении сетки в два раза отличие числа Нуссельта от «эталонного» уменьшилось более чем в 4 раза. Это говорит о том, что используемые в данном расчёте численные схемы обеспечивают порядок аппроксимации не ниже второго[1].

4.2. Моделирование турбулентных течений на основе RANS-подхода

Феномен турбулентного движения сплошной текучей среды неразрывно связан с понятием динамического хаоса в детерминированных системах [18, 19]. Принимая во внимание многогранность и сложность явления, в целом турбулентное движение среды можно определить как хаотическое во времени и пространстве движение вязкой текучей среды, характеризующееся существенной нестационарностью, трехмерностью и завихренностью.

[1] Важно подчеркнуть, что расчёт тепловых потоков на стенке в OpenFOAM осуществляется по схеме первого порядка, однако поскольку в рассматриваемой задаче при низком числе Ra в пристеночных слоях распределение температуры практически линейное, это не не влияет на общий порядок аппроксимации

Как отмечается в работе [20], особенность турбулентного движения состоит в том, что хотя в отличие от многих других сложных явлений турбулентные течения могут быть описаны в рамках классических уравнений Навье-Стокса (и поэтому в этом смысле проблема моделирования турбулентности может считаться решённой), численное решение этих уравнений при высоких числах Рейнольдса требует огромных вычислительных ресурсов, и несмотря на огромный прогресс вычислительной техники, достигнутый в последние десятилетия, даже по самым оптимистичным прогнозам задача прямого моделирования турбулентных течений на основе уравнений Навье-Стокса для практических приложений останется неразрешимой вплоть до конца XXI века. В этой связи ключевым вопросом описания турбулентности является поиск приемлемой модели, с одной стороны пригодной для адекватного описания турбулентного движения, а с другой — не слишком требовательной к вычислительным ресурсам. Выбор того или иного подхода к моделированию турбулентности определяется в первую очередь требованиями к точности получаемых решений и имеющимися в наличии вычислительными ресурсами.

Более подробно о различных подходах и их применимости рассказывается в разделе 4.2.1, затем в разделе 4.2.2 описываются имеющиеся в OpenFOAM модели турбулентности на основе RANS-подхода, и далее в разделах 4.2.3, 4.2.4 описываются некоторые канонические задачи, дающие приемлемые по точности результаты (с инженерной точки зрения) при использовании RANS-моделей.

4.2.1. Краткое введение в моделирование турбулентности

В соответствии с общепринятой в настоящее время классификацией (см., например, [20]), подходы к моделированию турбулентности делятся на три основные категории. К первой относится наиболее распространенный в настоящее время подход на основе осредненных по Рейнольдсу уравнений Навье-Стокса (Reynolds Averaged Navier-Stokes, RANS); поскольку в настоящем пособии рассматривается именно этот подход, далее он будет описан более подробно. Ко второй и третьей категориям относятся так называемые вихреразрешающие подходы, когда некоторая (или вся, как в методе DNS) часть турбулентных пульсаций разрешается непосредственно путём чис-

ленного решения соответствующих уравнений. В методе прямого численного моделирования турбулентности (Direct Numerical Simulation, DNS) осуществляется численное интегрирование трехмерных нестационарных уравнений Навье-Стокса с разрешением всех пространственно-временных масштабов турбулентности без использования какой-либо модели. В рамках метода крупных вихрей (Large Eddy Simulation, LES) применяется процедура пространственной фильтрации, в результате чего из рассмотрения исключается часть масштабов турбулентных пульсаций, позволяя значительно снизить требования к пространственно-временному разрешению (и следовательно, к вычислительным ресурсам). Для учета влияния отфильтрованных масштабов турбулентности привлекаются те или иные полуэмпирические модели (называемые также «подсеточными»). Оба метода (DNS и LES) относятся ко второй категории. К третьей категории относят так называемые «гибридные» методы, основанные на совместном использовании RANS и LES/DNS в различных областях потока.

В RANS-подходе уравнения Рейнольдса получаются из уравнений Навье-Стокса для несжимаемой жидкости путём осреднения скорости и давления актуального (турбулентного) движения по времени. Например, осреднение давления p записывается следующим образом:

$$\overline{p} = \frac{1}{T} \int\limits_{t-T/2}^{t+T/2} p(t)dt.$$

Здесь T — период осреднения, который предполагается достаточно большим по сравнению с временными масштабами всех турбулентных пульсаций, присутствующих в течении, и достаточно малым по сравнению с характерным временным масштабом осредненного течения. Строго говоря, корректное осреднение по времени предполагает переход к пределу $T \to \infty$, т.е. осреднение всех временных колебаний. Однако в ряде случаев в турбулентном течении присутствует ярко-выраженная нестационарность с «медленными» колебаниями, тогда как турбулентные пульсации имеют гораздо меньшие характерные временных масштабы. Расчёт такого рода задач на основе нестационарных уравнений Рейнольдса (с конечным T) называется URANS (Unsteady RANS), или также VLES (Very Large Eddy Simulation); последнее название подчёркивает то обстоятельство, что в по-

токе предполагается наличие крупных вихревых структур, «время жизни» которых на порядки превосходит характерные периоды турбулентных пульсаций; такие структуры также называют «когерентными». Отметим, что иногда подход URANS достаточно точно воспроизводит имеющуюся в потоке крупномасштабную нестационарность, хотя в каждом конкретном случае требуется тщательно анализировать возможность применения URANS и получаемые на его основе результаты[1].

Актуальные значения скорости и давления можно переписать через их осреднённые значения, вводя понятие пульсации соответствующей величины (обозначается штрихом), т.е.:

$$\vec{V} = \overline{\vec{V}} + \vec{V}', \quad p = \overline{p} + p'.$$

После осреднения уравнений Навье-Стокса по времени указанным выше способом получаются уравнения Рейнольдса для осреднённого движения, включающие дополнительное слагаемое, называемое тензором турбулентного напряжения $\tau_{turb} = -\rho\overline{\vec{V}'\vec{V}'}$ (или также рейнольдсовыми напряжениями):

$$\nabla \cdot \overline{\vec{V}} = 0,$$

$$\frac{\partial \rho \overline{\vec{V}}}{\partial t} + \nabla \cdot \left(\rho \overline{\vec{V}}\, \overline{\vec{V}} \right) = -\nabla \overline{p} + \nabla \cdot \left(2\mu \overline{\dot{S}} + \tau_{turb} \right).$$

Здесь $\overline{\dot{S}}$ — тензор скоростей деформации осреднённого движения, записываемый в декартовой системе координат как:

$$\overline{\dot{S}}_{ij} = \frac{1}{2} \left(\frac{\partial \overline{V}_j}{\partial x_i} + \frac{\partial \overline{V}_i}{\partial x_j} \right).$$

Преимуществом RANS-подхода является то, что уравнения Рейнольдса сформулированы непосредственно относительно осреднённых по времени характеристик течения, которые в основном и представляют интерес для практических приложений. Однако, уравнения Рейнольдса являются незамкнуты: тензор турбулентных напряжений не определён, поскольку рассчитывается на основе пульсационных характеристик, которые в свою

[1] Кроме того, важно подчеркнуть, что в URANS величина T не «назначается», а по сути неявным образом определяется используемой моделью турбулентности.

очередь всецело определяются самим турбулентным движением и никоим образом не связаны с физическими свойствами среды (иначе говоря, тензор турбулентных движений является характеристикой самого движения, а не среды). Именно поэтому, в отличие, например, от вязких напряжений, имеющих молекулярную природу, невозможно в принципе сформулировать какой-либо универсальный закон для τ_{turb}, хотя для некоторых специальных случаев (вроде однородной изотропной турбулентности) удаётся вычленить «универсальные» законы, присущие конкретному типу турбулентности. Таким образом, для использования уравнений Рейнольдса необходимо ввести дополнительные соотношения, связывающие τ_{turb} с характеристиками осредненного движения. И поскольку эти соотношения могут быть получены только с использованием той или иной эмпирической информации, их принято называть полуэмпирическими моделями турбулентности.

Один из подходов к замыканию уравнений Рейнольдса состоит в использовании уравнения переноса турбулентных напряжений, которые могут быть получены из уравнений Навье-Стокса с помощью процедуры осреднения по времени. Однако эти уравнения содержат моменты третьего порядка (например, $\overline{V_i' V_j' V_k'}$), которые также неопределены. Для моментов третьего порядка тоже можно вывести уравнения, но и они будут незамкнуты. Таким образом, необходимо для неопределённых слагаемых в получаемых уравнениях строить какие-либо модели; подобные модели носят названия моделей рейнольдсовых напряжений; подробней о них см. [20].

В настоящее время для замыкания уравнений Рейнольдса наиболее распространённым является подход, основанный на гипотезе Буссинеска о линейной связи τ_{turb} с тензором скоростей деформации осреднённого движения (аналогично обобщённому закону Ньютона для вязких напряжений):

$$\tau_{turb} = 2\mu_t \overline{\dot{S}} - \frac{2}{3}\rho k E, \tag{4.3}$$

где k — кинетическая энергия турбулентных пульсаций (или просто кинетическая энергия турбулентности), равная по определению:

$$k = \sum_{i=1}^{3} \frac{\overline{V_i'^2}}{2},$$

а μ_t — так называемый динамический коэффициент турбулентной вязкости. Подчеркнём, что как и τ_{turb}, турбулентная вязкость не является свойством среды, и даже более того, не является какой-либо физически обоснованной величиной, хотя наиболее близким к реальности следует считать представление о турбулентной вязкости как о механизме передачи импульса пульсационным движением, который непосредственно воздействует на осреднённое движение[1].

Другие подходы, связывающие τ_{turb} с какими-либо характеристиками осреднённого движения нелинейным образом, в данном пособии не рассматриваются; интересующиеся могут обратиться к монографии [20] за более подробной информацией.

Следуя [20], необходимо отметить, что в настоящее время идея создания универсальной RANS-модели, т.е. модели, пригодной для расчёта большинства турбулентных течений, признана полностью неосуществимой. Это в первую очередь связано с сущностью турбулентного движения: многочисленные экспериментальные и расчётные исследования убедительно показали, что локальные осреднённые характеристики турбулентных потоков сильно подвержены глобальному влиянию устойчивых, крупномасштабных, принципиально трёхмерных и нестационарных когерентных структур, причём характеристики этих структур зависят от конкретной геометрии рассматриваемого течения и граничных условий. Иными словами, «гипотеза локальности», на которой в неявной форме базируются RANS-модели турбулентности, не выполняется, что в принципе исключает возможность построения идеальной модели такого типа. Это относится как к моделям, базирующимся на гипотезе Буссинеска, так и к моделям переноса рейнольдсовых напряжений.

Тем не менее, существуют классы задач, где использование RANS-подхода вполне оправдано и даёт удовлетворительные по точности результаты. Информация о применимости той или иной модели в конкретной задаче является весьма важной: по-прежнему ведутся работы по накоплению тестовой базы для валидации различных моделей турбулентности. И несмотря на отмеченные серьёзные недостатки RANS-моделей, именно

[1] Здесь можно опять же провести аналогию с возникновением вязкости за счёт молекулярного движения, если вместо молекул рассматривать так называемые «моли», т.е. макроскопические объёмы жидкости, которые движутся хаотически в потоке и теряют «индивидуальность» только после передачи своего импульса осреднённому движению.

они ещё долго будут оставаться основным рабочим инструментом решения практических задач науки и техники.

4.2.2. RANS-модели в OpenFOAM

В OpenFOAM реализовано множество моделей турбулентности, как RANS, так и LES и некоторых гибридных. Далее подробнее остановимся на описании нескольких RANS-моделей, опирающихся на гипотезу Буссинеска (4.3), и некоторых связанных с моделированием турбулентности вычислительных технологий, таких как пристенные функции. Дополнительную информацию об имеющихся в OpenFOAM моделях можно получить в [14, 12]. Кроме того, полезную информацию о различных моделях, включая уравнения и значения калибровочных коэффициентов в них, можно найти на сайте `https://turbmodels.larc.nasa.gov`; там же приводится обширная тестовая база для валидации различных моделей.

Все модели турбулентности в OpenFOAM вынесены в отдельную библиотеку и доступны для многих решателей, при этом «активация» моделей турбулентности для всех решателей осуществляется единым образом при помощи задания соответствующих параметров в файле `constant/turbulenceProperties`. За выбор того или иного подхода к моделированию турбулентности отвечает параметр simulationType в этом файле. Возможными вариантами являются: laminar (расчёт без модели турбулентности или DNS), RAS (Reynolds-Averaged Simulation, т.е. использование RANS-подхода) и LES (собственно метод LES и гибридные RANS/LES методы).

При задании параметра «simulationType RAS;» необходимо указать RANS-модель в секции RAS при помощи параметра RASModel. Пример файла `constant/turbulenceProperties` приведён ниже.

```
Файл constant/turbulenceProperties:
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      turbulenceProperties;
}
simulationType RAS;
```

```
RAS
{
    RASModel        kOmegaSST;
    turbulence      on;
    printCoeffs     on;
}
```

Здесь в качестве модели турбулентности указана k-ω SST модель Ментера (kOmegaSST), о которой далее будет говориться более подробно. Параметр «turbulence», установленный в «off», означает «замораживание» турбулентности, т.е. не происходит решение уравнений для каких-либо турбулентных характеристик и коэффициент турбулентной вязкости не меняется со временем/итерацией. Параметр «printCoeffs» отвечает за вывод на экран справочной информации о значении коэффициентов выбранной модели турбулентности.

Отметим, что в OpenFOAM все модели турбулентности разбиты на две группы: для несжимаемых течений (incompressible) и для сжимаемых (compressible). Оставляя за рамками данного пособия вопрос о моделировании турбулентности в сжимаемых течениях[1], будем в дальнейшем рассматривать только модели для несжимаемых течений; при этом вместо динамического коэффициента турбулентной вязкости μ_t в OpenFOAM используется кинематический коэффициент $\nu_t = \mu_t/\rho$, и все уравнения записываются в соответствующей форме (т.е. поделены на плотность ρ).

Некоторые из имеющихся в OpenFOAM RANS-моделей турбулентности приведены в таблице 4.1 (с кратким описанием; термины «низкорейнольдсовый» и «высокорейнольдсовый» расшифровываются далее). Отметим, что опыт использования различных моделей свидетельствует о том, что к настоящему времени наиболее «популярными» среди них являются следующие: модель Спаларта-Аллмараса переноса турбулентной вязкости, модель переноса кинетической энергии турбулентности k и удельной скорости ее диссипации ω (k-ω Shear Stress Transport модель Ментера или SST модель) и некоторые модели из семейства k-ε (модели переноса кинетической энергии турбулентности k и её диссипации ε). Отметим, что первые две в основном используются для расчёта течений, где важны пристенные эф-

[1] Подробнее о моделировании турбулентности в сжимаемых потоках см. [20]

Таблица 4.1. Некоторые RANS-модели в OpenFOAM для несжимаемых течений

Именование в OpenFOAM	Общепринятое название	Описание
SpalartAllmaras	Модель Спаларта-Аллмараса	Низкорейнольдсовая модель с одним дифференциальным уравнением переноса модифицированной турбулентной вязкости [21]
LaunderSharmaKE	k-ε модель Лаундера-Шарма	Низкорейнольдсовая модель, предложенная в 1974 г. [22]
kEpsilon	Стандартная k-ε модель Лаундера-Сполдинга	Высокорейнольдсовая модель, предложенная в 1972 г. [23]
kOmega	k-ω модель Вилкокса	Вариант высокорейнольдсовой модели Вилкокса, предложенной в 1988 г., с небольшими модификациями [24]
kOmegaSST	k-ω SST модель Ментера	Низкорейнольдсовая модель, предложенная в 1994 г., с некоторыми модификациями [25]

фекты, тогда как область применения k-ε моделей обычно ограничивается свободными (струйными) турбулентными течениями.

Необходимо сделать несколько важных замечания относительно моделирования пристенных турбулентных течений. Как показывают эксперименты, в развитом турбулентном пограничном слое на плоской пластине (или в прямом канале) профили осреднённых характеристик (скоростей) носят универсальный характер и могут быть описаны в переменных стенки, которые являются безразмерными и вводятся следующим образом:

$$y^+ = \frac{y u_\tau}{\nu}, \quad u^+ = \frac{u}{u_\tau}, \quad u_\tau = \sqrt{\frac{\tau_w}{\rho}}.$$

Здесь y — координата по нормали к стенке, u — продольная компонента осреднённой по Рейнольдсу скорости, u_τ — динамическая скорость, τ_w — касательное напряжение трения на стенке. Построенный в этих перемен-

ных профиль осреднённой скорости в пограничном слое является универсальным и имеет несколько характерных областей: вязкий подслой (располагающийся при y^+ от 0 до примерно 3), где турбулентные пульсации практически отсутствуют и профиль скорости линейный; логарифмический участок, который начинается с $y^+ \approx 30$ и в котором турбулентные напряжения существенно превосходят вязкие, но при этом характерные масштабы турбулентных вихрей по-прежнему «чувствуют» влияние стенки; переходная область (между $y^+ \approx 3$ и 30). Протяжённость логарифмического участка зависит от числа Рейнольдса и других параметров внешнего потока; минимальным значением условной границы логарифмического участка можно считать $y^+ \approx 100$, при увеличении числа Рейнольдса граница доходит до $y^+ \sim 10^3$, 10^4 и выше. За логарифмическим участком идёт внешняя область пограничного слоя, где турбулентность близка к свободной сдвиговой турбулентности, а влияние стенки на масштабы вихрей отсутствует. Далее условно располагается область внешнего потока (в случае турбулентного пограничного слоя на пластине), который может быть как сильно турбулентным, так и практически ламинарным, что обычно характеризуется величиной степени турбулентности, определяемой формулой:

$$Tu = \frac{\sqrt{\overline{u'^2}}}{U_\infty} \cdot 100\%,$$

U_∞ — характерная скорость внешнего потока. В предположении об однородности и изотропности турбулентности внешнего потока степень турбулентности можно рассчитывать через кинетическую энергию турбулентных пульсаций как:

$$Tu = \frac{\sqrt{2k/3}}{U_\infty} \cdot 100\%. \tag{4.4}$$

Таким образом, хорошая модель турбулентности, претендующая на детальное описание пристенных турбулентных течений, должна воспроизводить описанные области турбулентного пограничного слоя. Отметим, что представленный выше универсальный закон стенки справедлив только для так называемого равновесного турбулентного пограничного слоя, т.е. в установившемся турбулентном течении на плоской пластине без каких либо локальных отрывных зон и/или сильного неоднородного градиента давления. В ряде задач, где может применяться RANS-подход и где имеются

пристенные области, но при этом они не являются предметом рассмотрения и/или их влияние на остальной поток достаточно мало, либо заранее известно, что пограничный слой является равновесным, модель турбулентности может и не описывать детально структуру пристенной турбулентности; от неё требуется только корректное «восстановление» значения трения на стенке. Оказывается, что для подобных задач достаточно использовать так называемые пристенный функции, построенные на основе универсального закона стенки и позволяющие использовать достаточно грубые сетки, при этом воспроизводя некоторые характерные особенности пристенного течения.

При использовании пристенных функций накладываются некоторые условия на размер первой пристенной ячейки сетки (в зависимости от вида этих функций). Изначально при создании моделей турбулентности исходили из условия, что центр первой пристенной ячейки должен находится в области логарифмического участка; это означает, что безразмерная координата центра первой пристенной ячейки, обозначаемая[1] y_1^+, должна лежать в области от 30 до примерно 100 или выше (в зависимости от условий). Используя универсальный закон стенки в этой области можно «восстановить» значение трения на стенке без прописывания всего профиля скорости в пограничном слое. Такие RANS-модели, которые описывают пограничный слой только до логарифмического участка и используют пристенные функции, называют *высокорейнольдсовыми*[2]. Если модель турбулентности предполагает аккуратное разрешения пограничного слоя вплоть до вязкого подслоя (где профиль скорости линейный), то такие модели называют *низкорейнольдсовыми*; условия их применения обычно предполагают $y_1^+ < 1$. Для подобных моделей не требуется использовать каких-либо специальных пристенных функций, а только корректные граничные условия на стенке для параметров турбулентности.

Понятно, что при моделировании течения с пристенной турбулентностью сложно заранее оценить значение y_1^+, поскольку величина трения на

[1] Зачастую, когда речь идёт о качестве сетки, индекс «1» в обозначении y_1^+ опускают, т.е. используют просто обозначение y^+, подразумевая под ним значение этой величины для первой пристенной ячейки

[2] Термин «высокорейнольдсовый» исходит из определения y^+, которое по форме представляет собой число Рейнольдса, построенное по текущей координате y и динамической скорости u_τ, а вовсе не из значения числа Рейнольдса для данного течения

стенке τ_w неизвестна. В таких случаях можно поступать следующим образом: на основании некоторых предварительных предположений о течении оценить характерную величину τ_w и построить сетку с размером первой пристенной ячейки исходя из оцененного значения, затем провести расчёт, и если величина y_1^+ оказывается не соответствующей используемой модели турбулентности, перестроить сетку. Очевидно, что такой подход применим в случае относительно простых течений, но становится труднореализуемым для более сложных. В этой связи разными группами исследователей были разработаны так называемые универсальные пристенные функции (или автоматические), которые используются совместно с низкорейнольдсовыми моделями турбулентности и «переключаются» в зависимости от значения y_1^+: если $y_1^+ \lesssim 3$, то пристенные функции дают значения турбулентных величин, справедливые для вязкого подслоя, если $y_1^+ > 30$ — работают пристенные функции для логарифмического участка. Основной вопрос касается расчёта турбулентных характеристик в переходной области при $3 \lesssim y_1^+ < 30$, и здесь есть несколько различных вариантов, например, [26, 27, 28]. В настоящее время в OpenFOAM в качестве «универсальной» пристенной функции для ν_t можно использовать аппроксимацию «закона» стенки, предложенную в [29]; в OpenFOAM такая пристенная функция называется nutUSpaldingWallFunction. Простая пристенная функция в OpenFOAM, которая «переключается» с логарифмической области на вязкий подслой при переходе через некоторое пороговое значение y_1^+, называется nutkWallFunction (при этом расчёт значения y_1^+ осуществляется на основе значения k в первой пристенной ячейке, поэтому данная функция применима только с теми моделями турбулентности, которые оперируют с k); её имеет смысл использовать в случае, когда в основном сетка низкорейнольдсовая или высокорейнольдсовая, но существуют области пристенного течения, где трудно контролировать значение y_1^+, а точность моделирования пристенных эффектов в этих областях не так важна.

4.2.3. Турбулентное течение в плоском канале

Одним из примеров задачи для валидации RANS-моделей является турбулентное течение в плоском (двумерном) канале. Известно, что в установившемся режиме профиль скорости является универсальным (т.е. при-

сутствуют все перечисленные в предыдущем разделе области: вязкий подслой, переходная область, логарифмический участок, внешняя область). Для получения такого установившегося профиля можно поставить задачу, аналогичную рассмотренной в разделе 4.1.2 для развития ламинарного течения в плоском канале, также в безразмерной постановке.

Будем рассматривать прямоугольную расчётную область, на входе в которую задаётся равномерный профиль скорости и заданные параметры турбулентности, боковые границы являются стенками, а на выходе ставятся «мягкие» условия и постоянное давление. В отличие от ламинарной задачи, выход на установившийся режим в RANS-расчёте занимает довольно продолжительный участок канала. Для рассматриваемого здесь значения числа Рейнольдса $Re = U_{\text{вх}}H/\nu = 10^5$ длину расчётной области возьмём равной $150H$, где H — высота канала. Отметим, что результаты этой задачи также будут использоваться в следующем разделе при исследовании течения за обратным уступом.

В отличие от ламинарной задачи течения в канале, здесь сетку следует делать более подробной; в качестве исходного варианта расчёта с использованием низкорейнольдсовых моделей турбулентности будем использовать сетку с y^+ в первой пристенной ячейке равным ≈ 0.4 на установившемся участке. Файл сетки для этого варианта приведён ниже (задача имеет симметрию относительно оси канала, так что можно рассматривать только половину канала, но здесь для простоты расчёт ведётся для полного канала).

```
convertToMeters 1;
vertices
(
    (0 0 0)
    (150 0 0)
    (150 1 0)
    (0 1 0)
    (0 0 1)
    (150 0 1)
    (150 1 1)
    (0 1 1)
);
```

```
blocks ( hex (0 1 2 3 4 5 6 7) (250 50 1) simpleGrading (1 ((0.1 0.28 200)
    ↪ (0.8 0.44 1)(0.1 0.28 0.005)) 1));
boundary
(
    inlet
    {
       type patch;
       faces ( (0 3 4 7) );
    }
    outlet
    {
       type patch;
       faces ( (1 2 5 6) );
    }
    fb
    {
       type empty;
       faces ( (0 1 2 3) (4 5 6 7) );
    }
    walls
    {
       type wall;
       faces ( (0 1 5 4) (2 3 7 6) );
    }
);
```

Для решения данной задачи используется стационарный решатель simpleFoam; кинематический коэффициент вязкости задаётся в файле `constant/transportProperties` равным 10^{-5} (поскольку задача решается в безразмерной постановке, его величина не принципиальна, важно обеспечить заданное значение числа Рейнольдса $Re = 10^5$). Используется k-ω SST модель турбулентности; файл `constant/turbulenceProperties` имеет вид как в листинге на стр. 184.

Граничные условия для скорости U и давления p задаются аналогично ламинарной задаче (раздел 4.1.2) и здесь не приводятся. Помимо них необходимо задать начальные и граничные условия для k и ω (в файлах `0/k` и `0/omega`), а также граничные условия для турбулентной вязкости ν_t в файле `0/nut`. Отметим, что значение ν_t вычисляется на основе k и ω, однако, в случае использования пристенных функций необходимо явно прописывать

их и для ν_t. Поэтому файл граничных условий для данной задачи может выглядеть следующим образом:

```
Файл 0/nut:
dimensions      [0 2 -1 0 0 0 0];
internalField   uniform 0;
boundaryField
{
    inlet
    {
        type            calculated;
        value           uniform 0;
    }
    outlet
    {
        type            calculated;
        value           uniform 0;
    }
    walls
    {
        type            nutUSpaldingWallFunction;
        value           uniform 0;
    }
    fb { type empty; }
}
```

Здесь на входной и выходной границах указан тип «calculated», т.е. величина ν_t вычисляется через k и ω. На стенках используется простая пристенная функция nutUSpaldingWallFunction, хотя на используемой низкорейнольдсовой сетке можно также использовать тип calculated или fixedValue со значением вязкости 0.

Для k и ω граничные условия представлены ниже.

```
Файл 0/k:
dimensions      [0 2 -2 0 0 0 0];
internalField   uniform 6e-6;
boundaryField
{
    inlet
    {
        type            fixedValue;
```

```
        value           uniform 6E-6;
    }
    outlet
    {
        type            zeroGradient;
    }
    walls
    {
        type            kqRWallFunction;
        value           $internalField;
    }
    fb { type empty; }
}

Файл 0/omega:
dimensions      [0 0 -1 0 0 0 0];
internalField   uniform 0.2;
boundaryField
{
    inlet
    {
        type            fixedValue;
        value           uniform 0.2;
    }
    outlet
    {
        type            zeroGradient;
    }
    walls
    {
        type            omegaWallFunction;
        value           $internalField;
    }
    fb { type empty; }
}
```

Входные значения k и ω берутся из следующих условий: $Tu = \sqrt{2k/3}/U_{\text{вх}} = 0.002$, $\nu_t/\nu = 3$ ($\nu_t = k/\omega$); в данной задаче входные условия на турбулентные характеристики не влияют на решение на установившемся участке (могут незначительно влиять на сам процесс установления); выбранные значения исходят из задачи, описанной в следующем разделе.

Пристенная функция для k, имеющая название kqRWallFunction, представляет собой просто условие нулевой производной для k. Более корректным для низкорейнольдсовой сетки является условие на стенке $k = 0$, но в целом результаты при разных условиях близки друг к другу. В случае же использования высокорейнольдсовой сетки только условие $\partial k/\partial y = 0$ даёт правильное решение. Для ω пристенная функция omegaWallFunction реализует граничное условие, предложенное в [30]: $\omega = \sqrt{\omega_{vis}^2 + \omega_{log}^2}$, где ω_{vis} — значение ω в случае использования низкорейнольдсовой сетки, ω_{log} — значение ω для логарифмического участка; $\omega_{vis} = 6\nu/\left(0.075\sqrt{y_1}\right)$ — условие для ω на стенке, также предложенное Ментером.

Таким образом, предложенная постановка граничных условий позволяет проводить расчёт как на низкорейнольдсовой сетке, так и на высокорейнольдсовой; далее будет проводиться сравнение результатов, полученных на разных сетках.

Параметры численных схем можно взять из примера `tutorials/incompressible/simpleFoam/motorBike` (для ускорения расчёта можно в файле `fvSolution` поставить для p `reltol 0.1;`, а также в секции relaxationFactors для k и omega указать параметр релаксации 0.9). Для получения решения, сошедшегося с хорошей точностью, достаточно 300 итераций.

На рисунке 4.11 представлено распределение V_x, k и ν_t на оси канала. Отметим наличие немонотонности в процессе установления: после небольшого входного участка скорость на оси канала растёт почти линейно до некоторого значения, а затем начинает уменьшаться и выходит на своё установившееся значение ≈ 1.086, причём её экстремум приходится на область резкого роста k и ν_t. Такое поведение связано с тем, что образующиеся пограничные слои развиваются от входа в канал и затем происходит их смыкание; до момента смыкания скорость попрёк канала вне пограничных слоёв практически постоянная, поэтому генерация турбулентности отсутствует (она возможна только в сдвиговом течении), кинетическая энергия турбулентности уменьшается. После смыкания начинается интенсивная генерация турбулентности и рост k и ν_t; профиль скорости перестраивается. В качестве иллюстрации, демонстрирующей перестроение профиля скорости, на рисунке 4.12а приведены профили $V_x(y)$ в трёх сечениях, соответ-

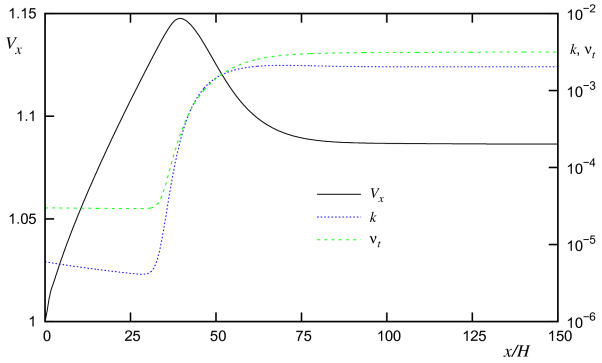

Рис. 4.11. Распределение продольной компоненты скорости и турбулентных характеристик вдоль оси канала

ствующих моменту до смыкания пограничных слоёв, области смыкания и

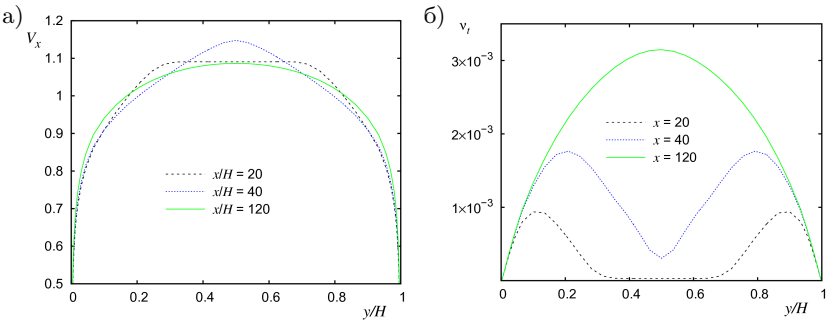

Рис. 4.12. Профили V_x (а) и ν_t (б) в разных сечениях

установившемуся режиму; также на рисунке 4.12б приведены профили ν_t в тех же сечениях. Условно можно считать окончанием переходной области такое значение x, при котором разность значений скорости на оси канала при данном x и в установившемся режиме отличается от максимального отклонения скорости от входной не более чем на 1%; в данном расчёте длина переходного участка составляет примерно $100H$.

Считая, что развивающийся от входа пограничный слой является турбулентным[1], можно оценить, при каком x/H произойдёт смыкание нарастающих пограничных слоёв. Известно [31], что толщина турбулентного пограничного слоя меняется по закону:

$$\delta(x) = 0.37x \left(\frac{U_{\text{вх}} x}{\nu} \right)^{-1/5},$$

откуда сечение x, при котором происходит смыкание, определяется полагая $\delta(x) = H/2$, т.е. по формуле:

$$\frac{x}{H} = 1.457 Re^{1/4}.$$

Для рассматриваемой задачи с $Re = 10^5$ получаем значение $x/H \approx 26$. В расчёте сечение, где происходит смыкание, можно определить по графику ν_t из рисунка 4.11: $x \approx 30$. Полученное (довольно неплохое) совпадение говорит только о том, что используемая RANS модель адекватно воспроизводит закономерности развития турбулентного пограничного слоя[2].

Наиболее важным с точки зрения практических приложений является способность моделей турбулентности предсказывать напряжение трения на стенке τ_w, или (для течений в трубе) коэффициент сопротивления λ, определяемый как:

$$\lambda = \frac{\Delta p}{L} \frac{2H}{\rho U_{\text{вх}}^2}.$$

Здесь $\Delta p/L$ — падение давления в канале на длине L. В установившемся режиме коэффициент сопротивления связан с τ_w по формуле:

$$\lambda = \frac{4\tau_w}{\rho U_{\text{вх}}^2}.$$

Распределение τ_w на стенках канала можно получить при помощи функции wallShearStress, выполнив команду:

```
simpleFoam -postProcess -func wallShearStress -latestTime
```

[1] Оставляя в стороне вопрос реализуемости подобного входного условия на практике, подчеркнём, что в реальности образующийся на стенках канала пограничный слой сначала развивается как ламинарный, а переход к турбулентности происходит позднее, при $Re_x = U_{\text{вх}} x/\nu \gtrsim 10^5$, т.е. через несколько калибров вниз по потоку

[2] Этого следует ожидать, поскольку задача о турбулентном пограничном слое является одной из канонических для валидации моделей турбулентности, особенно низкорейнольдсовых

Эта команда записывает распределение τ_w/ρ в файл[1] `200/wallShearStress`, который имеет такую же структуру, как и файлы других полевых величин (типа `p` и `U`). В описываемом расчёте в установившемся режиме получено значение $\tau_w/\rho = 0.00194$, откуда $\lambda = 7.77 \times 10^{-3}$. Для сравнения полученного значения можно воспользоваться экспериментальной зависимостью $\lambda_{exp}(Re)$ для плоского канала, у которого ширина (протяжённость в z-направлении) много больше его высоты, приведённой в [32]:

$$2\lambda_{exp}(Re) = \frac{1.1}{(1.8\lg(2Re) - 1.64)^2}.$$

Для числа $Re = 10^5$ указанная формула даёт значение $\lambda_{exp} = 8.81 \times 10^{-3}$, что на 13% выше, чем полученное в расчёте. Таким образом, данный тест даёт представление о возможной погрешности расчёта при использовании RANS-подхода[2].

Получить распределение y_1^+ после завершения расчёта можно при помощи команды:

```
simpleFoam -postProcess -func yPlus -latestTime
```

Эта команда записывает в каталог, соответствующий последнему моменту времени, файл `yPlus`, который содержит распределение y_1^+ на стенках, а также выводит информацию о максимальном, минимальном и среднем значениях, как показано ниже[3]:

```
yPlus yPlus write:
    writing object yPlus
    patch walls y+ : min = 0.354215, max = 0.510727, average = 0.366873
```

Для иллюстрации выполнения «закона стенки» в рассматриваемом течении на рисунке 4.13 приведено распределение $u^+(y^+)$, полученное после

[1] В файл выводится значение вектора напряжения трения на стенке, в данном случае нас интересует его x-компонента

[2] Формула для λ_{exp} является результатом обработки многих экспериментов при разных числах Рейнольдса, поэтому также содержит некоторые экспериментальные и регрессионные погрешности ($\sim 5\%$). Кроме того, в основном эксперименты выполняется для круглых труб, и настройка модели турбулентности осуществляется для течения в круглой трубе, что также следует учитывать при сравнении расчёта и эксперимента.

[3] При использовании nutkWallFunction в качестве граничного условия для nut рассчитываемые значения y_1^+ получаются некорректными (сильно заниженными).

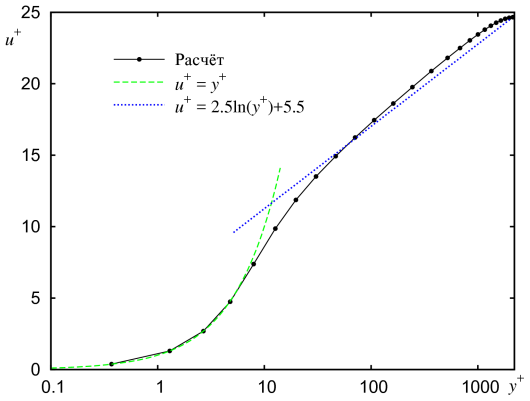

Рис. 4.13. Профиль скорости при $x = 120$ в «переменных стенки»

обезразмеривания профиля скорости в сечении $x = 120$ (точки на расчётной кривой соответствуют центрам ячеек расчётной сетки). На том же рисунке приведены также уравнения $u^+(y^+)$ для вязкого подслоя и логарифмического участка; выражение для последнего записывается в виде:

$$u^+ = \frac{1}{\varkappa} \ln\left(y^+\right) + B,$$

где константа Кармана $\varkappa$ и постоянная B определяются их эксперимента. Логарифмический профиль на рисунке построен для «канонических» значений $\varkappa = 0.4$ и $B = 5.5$ из [1]. Небольшое расхождение расчёта и «канонической» зависимости можно объяснить тем, что во-первых, эксперименты в основном проводятся для круглой трубы, а во-вторых, разброс значений константы Кармана, полученной в разных экспериментах и при разных условиях достигает $\pm 15\%$ [33]; в частности, для данного расчёта лучшее согласие можно получить, выбрав $\varkappa = 0.38$.

Рассмотрим теперь влияние сетки и модели турбулентности на получаемое в расчёте установившееся значение коэффициента сопротивления λ. Для построения высокорейнольдсовой сетки можно использовать следующие параметры в файле `system/blockMeshDict`:

```
blocks ( hex (0 1 2 3 4 5 6 7) (250 28 1) simpleGrading (1 ((0.1 0.14 2)
    ↪ (0.8 0.72 1)(0.1 0.14 0.5)) 1));
```

Результаты расчётов на разных сетках и при использовании разных моделей турбулентности сведены в таблице 4.2. Отметим, что модель k-ω SST даёт близкие результаты на высокорейнольдсовой и низкорейнольдсовой сетках, тогда как результаты, полученные при помощи k-ε модели Лаундера-Шарма на двух сетках сильно различаются, что означает сильную чувствительность данной модели к расчётной сетке и/или пристенным функциям. Также, сходимость итерационного процесса при использовании модели Лаундера-Шарма на низкорейнольдсовой сетке несколько хуже (требуется в два раза больше итераций), чем для модели k-ω SST, хотя значение, полученное с её помощью, ближе к экспериментальному. Что касается высокорейнольдсовых моделей (k-ω Вилкокса и k-ε Лаундера-Сполдинга), то их результаты довольно близки к результатам, полученным при помощи модели k-ω SST на высокорейнольдсовой сетке. В целом, наилучший результат на разных сетках даёт модель Спаларта-Аллмараса, хотя по сравнению с k-ω SST она сходится немного хуже.

Таблица 4.2. Коэффициент сопротивления λ при использовании разных сеток и моделей

Модель турбулентности	Среднее значение y_1^+	$\lambda \times 10^3$	Отклонение от $\lambda_{exp} = 8.81 \times 10^{-3}$
kOmegaSST	0.37	7.77	-12%
kOmegaSST	38.8	8.35	-5%
kOmega	39.7	8.56	-3%
LaunderSharmaKE	0.32	8.94	$+1\%$
LaunderSharmaKE	36.7	7.51	-15%
kEpsilon	38.8	8.22	-7%
SpalartAllmaras	0.38	8.53	-3%
SpalartAllmaras	39.5	8.63	-2%

4.2.4. Течение и теплообмен за обратным уступом

Данная задача также является канонической для тестирования применимости различных подходов к моделированию турбулентности, поскольку течение за обратным уступом включает отрыв турбулентного потока от уступа на нижней стенке, его присоединение на некотором расстоянии от уступа и релаксацию неравновесного турбулентного пограничного слоя вниз по потоку от точки присоединения — то есть многие сложные элементы реальных турбулентных течений.

Постановка задачи исходит из эксперимента [34, 35], в котором рассматривается турбулентное течение воздуха в плоском канале с внезапным односторонним расширением (обратная ступенька) с учётом теплопереноса от обогреваемой нижней стенки (см. рисунок 4.14). В моделируемом эксперименте отношение высот канала до и после ступеньки равно 4:5 (степень расширения 1.25), высота ступеньки $h = 3.81$ см. На входе в измерительную секцию ($3.8h$ вверх по потоку от уступа) скорость на оси канала составила $U_{ref} = 11.3$ м/с (число Рейнольдса $Re_h = U_{ref}h/\nu = 28000$), толщина пограничного слоя[1] $\delta_{99} = 4.05$ см, коэффициент трения на стенке $c_f = 2\tau_w/(\rho U_{ref}^2) = 0.00312$. Степень турбулентности в ядре потока $Tu \approx 0.2\%$; температура воздуха $T_0 = 300$ K. На нижней стенке канала после ступеньки подводится постоянный тепловой поток $q_w = 270$ Вт/м^2, остальные стенки канала теплоизолированные.

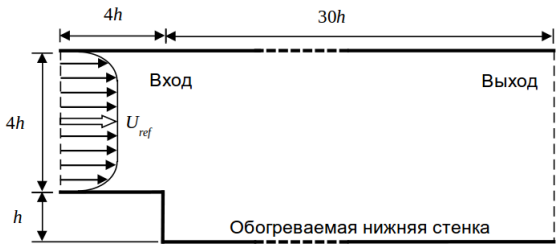

Рис. 4.14. Постановка задачи о течении за обратным уступом

[1] Имеется в виду процентная толщина δ_{99}, т.е. расстояние от стенки, где скорость равна 0.99 от скорости внешнего потока

Расчётная область включает участок длиной $4h$, расположенный вверх по потоку от уступа, и участок длиной $30h$ — вниз по потоку от уступа. Таким образом, расчётная область покрывает всю измерительную секцию экспериментальной установки и продолжается достаточно далеко вниз по потоку, чтобы исключить влияние выходных граничных условий на решение.

Поскольку течение является существенно дозвуковым, в расчёте можно использовать приближение несжимаемой среды и использовать решатель simpleFoam (рассматривается стационарное течение). Однако, в этом решателе (как и в других из класса incompressible) отсутствует уравнение переноса тепла. Это связано с тем, что в пренебрежении эффектами изменения плотности за счёт изменения температуры уравнение теплопереноса в несжимаемой жидкости представляет собой конвективно-диффузионное уравнение переноса пассивного скаляра (термин «пассивный» означает отсутствие обратного влияние скаляра на основное течение). В OpenFOAM имеется возможность добавлять уравнения переноса пассивного скаляра без модификации кода при помощи так называемых функциональных объектов (functionObject). Задача переноса температуры будет рассмотрена детально после исследования течения.

Численный расчёт будем проводить в безразмерной постановке, выбрав в качестве масштаба длины высоту ступеньки h, а в качестве масштаба скорости — U_{ref}. Для обеспечения числа Рейнольдса $Re_h = 28000$ необходимо задать кинематический коэффициент вязкости $\nu = 3.57 \times 10^{-5}$. Для проведения расчёта используется блочно-структурированная сетка со сгущением к стенкам, полученная при помощи утилиты blockMesh (файл `blockMeshDict` представлен ниже). Для моделирования турбулентности используется RANS SST k-ω модель Ментера (аналогично задаче из предыдущего раздела).

```
convertToMeters 1;
vertices
(
    (-4 1 0)                //0
    (0 1 0)                 //1
    (0 5 0)                 //2
    (-4 5 0)                //3
```

```
    (-4 1 1)                //4
    (0 1 1)                 //5
    (0 5 1)                 //6
    (-4 5 1)                //7
    (30 1 0)                //8
    (30 5 0)                //9
    (30 1 1)                //10
    (30 5 1)                //11
    (0  0 0)                //12
    (30 0 0)                //13
    (0  0 1)                //14
    (30 0 1)                //15
);
blocks
(
    hex (0 1 2 3 4 5 6 7)      (24  50 1) simpleGrading (((0.8 0.6  1)(0.2
    ↪  0.4 0.2)) ((0.1 0.18 10)(0.8 0.64 1)(0.1 0.18 0.1)) 1)
    hex (1 8 9 2 5 10 11 6)    (140 50 1) simpleGrading (((0.2 0.35 7)(0.8
    ↪  0.65  1)) ((0.1 0.18 10)(0.8 0.64 1)(0.1 0.18 0.1)) 1)
    hex (12 13 8 1 14 15 10 5) (140 40 1) simpleGrading (((0.2 0.35 7)(0.8
    ↪  0.65  1)) ((0.2 0.32 10)(0.55 0.36 1)(0.25 0.32 0.3)) 1)
);
boundary
(
    inlet
    {
       type patch;
       faces ( (0 3 4 7) );
    }
    outlet
    {
       type patch;
       faces ( (8 9 10 11) (8 10 13 15) );
    }
    wall_bottom
    {
       type wall;
       faces ( (12 13 14 15) );
    }
    walls_other
    {
       type wall;
```

```
      faces ( (0 1 5 4) (2 3 7 6) (1 12 14 5) (2 9 6 11) );
   }
   fb
   {
      type empty;
      faces ( (0 1 2 3) (4 5 6 7) (1 2 8 9) (5 6 10 11) (1 8 12 13) (5 10
   ↪  14 15) );
   }
);
```

В данной задаче решение определяется в том числе и корректно поставленными условиями на входной границе: следует задавать профиль скорости и турбулентных характеристик в соответствии с экспериментом. Для получения этих профилей в расчёте можно рассмотреть задачу развития турбулентного течения от однородного профиля[1] и воспользоваться постановкой задачи, описанной в предыдущем разделе, с соответствующими модификациями. Во первых, требуется изменить координаты y канала: для данной постановки y должна меняться от 1 до 5. Во-вторых, здесь не требуется рассчитывать течение до выхода на установившийся режим, достаточно просчитать участок нарастающих пограничных слоёв до момента, где $\delta_{99}/(4h) = 4.05/(4 \times 3.81) \approx 0.27$; как показывает предыдущий расчёт, близкое к 0.27 значение $\delta_{99}/(4h)$ наблюдается при $x/(4h) = 20$ (см. рисунок 4.12а), а смыкание пограничных слоёв наступает при $x/(4h) \approx 30$; это значение и можно взять в качестве длины расчётной области, т.е. $120h$. В-третьих, в качестве масштаба скорости необходимо взять максимальную скорость на оси канала в сечении, где $\delta_{99}/(4h) \approx 0.27$. Заранее точно сказать, какова должна быть скорость на входе чтобы получить значение 1 в искомом сечении нельзя, но можно примерно оценить на основе предыдущего расчёта, вычислив скорость на оси канала при $x/(4h) = 20$ (получается значение 1.09); тогда на входе в канал необходимо задать скорость, равную $1/1.09 = 0.917$. Кинематический коэффициент вязкости для этого вспомогательного расчёта задаётся такой же, как и при расчёте обратного уступа, т.е. $\nu = 3.57 \times 10^{-5}$, обеспечивая таким образом требуемое число Рейнольдса Re_h =28000. Важно подчеркнуть, что если в такой постановке

[1] В эксперименте перед измерительной секцией находилась область отсоса через пористые стенки для управления толщиной нарастающего на стенках пограничного слоя

посчитать число Рейнольдса по входной скорости, равной 0.917, указанному коэффициенту вязкости и высоте канала, равной $4h$, то получится значение $\approx 1.03 \times 10^5$, близкое к значению из предыдущего раздела. Таким образом, постановка задачи из предыдущего раздела «подобна» вспомогательному расчёту течения в канале; необходимость пересчёта связана только с изменением масштабных величин; в принципе, можно было бы и не пересчитывать течение в канале, но тогда потребовалось бы корректное перемасштабирование требуемых для постановки входных граничных условий данных, а именно: скорости, k и ω. Граничные условия для k и ω во вспомогательном расчёте следует брать из тех же соображений, что и для задачи из предыдущего раздела, т.е. $Tu = 0.2\%$, $\nu_t/\nu = 3$.

Вспомогательный расчёт показывает, что $\delta_{99}/(4h) \approx 0.27$ в сечении $x = 80h$. Для извлечения профилей в сечении можно воспользоваться ParaView (фильтр PlotOverLine) или средствами OpenFOAM. Рассмотрим второй вариант, реализуемый при помощи функции singleGraph (см. пример в [14]). Для использования этой функции необходимо создать файл `system/singleGraph` следующего содержания:

```
start   (80 1 0.5);
end     (80 5 0.5);
fields  (U k omega);
// Sampling and I/O settings
#includeEtc "caseDicts/postProcessing/graphs/sampleDict.cfg"
setFormat csv;
// Must be last entry
#includeEtc "caseDicts/postProcessing/graphs/graph.cfg"
```

и запустить команду

```
simpleFoam -postProcess -func singleGraph -time 200
```

Эта команда создаст файлы `line_U.csv` и `line_k_omega.csv` в каталоге `postProcessing/singleGraph/200`, в которых содержатся столбцы величин, разделённые запятой, в первом столбце расстояние от точки start вдоль отрезка (start; end), в остальных — значения соответствующих полей, интерполированные в узлы расчётной сетки.

Полученные профили можно использовать при задании граничного условия на входе в задаче обтекания обратного уступа с помощью гра-

ничного условия fixedProfile. Например, граничные условия для скорости будут выглядеть следующим образом:

```
Часть файла 0/U:
boundaryField
{
    inlet
    {
        type            fixedProfile;
        profile         csvFile;
        profileCoeffs
        {
            nHeaderLine         1;          // Number of header lines
            refColumn           0;          // Reference column index
            componentColumns    (1 2 3);    // Component column indices
            separator           ",";        // Optional (defaults to ",")
            mergeSeparators     yes;        // Merge multiple separators
            fileName            "line_U.csv";// Name of csv data file
        }
        direction        (0 1 0);
        origin           1;
    }
    outlet
    {
        type            zeroGradient;
    }
    "wall.*"
    {
        type            noSlip;
    }
    fb { type empty; }
}
```

Здесь refColumn — номер столбца с координатой (со значениями y; нумерация столбцов идёт от нуля), direction — направление изменения координаты (вдоль оси y), origin — смещение, componentColumns — номера столбцов с тремя компонентами скорости. Путь к файлу, который указывается опцией fileName, отсчитывается от каталога задачи, поэтому необходимо скопировать файлы `line_U.csv` и `line_k_omega.csv` в соответствующий каталог.

Для турбулентных характеристик k и ω ставятся следующие граничные условия:

```
Часть файла 0/k:
boundaryField
{
    inlet
    {
        type            fixedProfile;
        profile         csvFile;
        profileCoeffs
        {
            nHeaderLine         1;
            refColumn           0;
            componentColumns    (1);
            separator           ",";
            mergeSeparators     yes;
            fileName            "line_k_omega.csv";
        }
        direction        (0 1 0);
        origin           1;
    }
    outlet
    {
        type            zeroGradient;
    }
    "wall.*"
    {
        type            kqRWallFunction;
        value           $internalField;
    }
    fb { type empty; }
}

Часть файла 0/omega:
boundaryField
{
    inlet
    {
        type            fixedProfile;
        profile         csvFile;
        profileCoeffs
```

```
        {
            nHeaderLine         1;
            refColumn           0;
            componentColumns    (2);
            separator           ",";
            mergeSeparators     yes;
            fileName            "line_k_omega.csv";
        }
        direction        (0 1 0);
        origin           1;
    }
    outlet
    {
        type            zeroGradient;
    }
    "wall.*"
    {
        type            omegaWallFunction;
        value           $internalField;
    }
    fb { type empty; }
}
```

Для давления накладываются стандартные граничные условия: на всех границах, кроме выходной, ставится условие нулевой производной, на выходе — постоянное (нулевое) значение. Для турбулентной вязкости в файле `nut` используется пристенная функция nutkWallFunction.

Параметры численных схем и решателей СЛАУ можно задавать как в предыдущей задаче. Для получения сошедшегося решения достаточно 500 итераций.

На рисунке 4.15 представлена картина течения. Отрывная область за уступом локализована в области $x/h \lesssim 10$, а начиная с $x/h \approx 20$ течение выходит на практически установившийся режим.

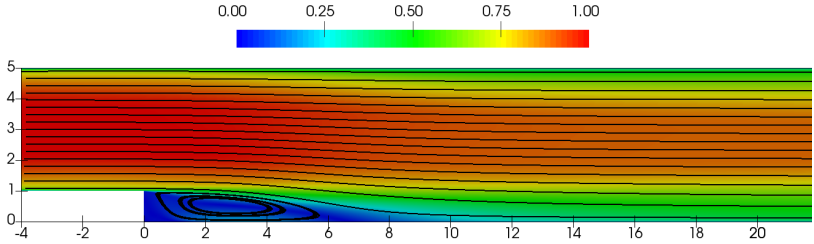

Рис. 4.15. Линии тока и распределение модуля скорости

Наибольший интерес в данной задаче представляет распределение коэффициента трения c_f на нижней стенке, поскольку для него имеются экспериментальные данные [35] (приведены в таблице 4.3; область отрицательных значений c_f соответствует обратному течению в рециркуляционной зоне). В этой связи важно иметь хорошее разрешение пограничного слоя вблизи нижней стенки. При помощи функции yPlus можно получить распределение y_1^+ на нижней стенке wall_bottom (аналогично процедуре, описанной для предыдущей задачи); в данном расчёте y_1^+ на нижней стенке не превосходит значения 0.25. Распределение коэффициента трения можно получить из напряжения трения на стенке, рассчитываемого при помощи функции wallShearStress (аналогично описанному в предыдущем разделе), по формуле:

$$c_f = \frac{2\tau_w}{\rho U_{ref}^2}.$$

Извлечь распределение τ_w вдоль нижней стенки можно при помощи ParaView или функции singleGraph утилиты postProcess (по аналогии с описанной процедурой извлечения данных в предыдущем разделе).

Расчётные и экспериментальные значения c_f представлены на рисунке 4.16. Там же для сравнения приведёны результаты расчёта, проведённого с использованием ANSYS Fluent 16.2. Расчётные данные хорошо совпадают в области рециркуляционной зоны и несколько расходятся вниз по потоку, но оба расчёта дают при $x/h > 15$ сильно заниженное (около 25%) значение c_f по сравнению с экспериментом. Примерно такое же сильное

Таблица 4.3. Распределение коэффициента трения вдоль нижней стенки из [35]

x/h	$c_f \times 10^3$	x/h	$c_f \times 10^3$
0.38	0.055	7.40	0.321
0.71	0.007	8.03	0.637
1.38	−0.053	8.76	0.831
2.00	−0.133	9.45	0.927
2.73	−0.460	10.76	1.200
3.99	−0.898	12.03	1.418
5.43	−0.634	13.40	1.582
6.05	−0.360	16.05	1.768
6.40	−0.163	18.80	1.992
6.71	0.032		

расхождение наблюдается и в области $x/h \approx 5$. Таким образом, данный пример демонстрирует «предсказательную силу» RANS-моделей: даже для относительно простых (по геометрии) течений возможно довольно сильное расхождение получаемого в расчётах решения с экспериментом.

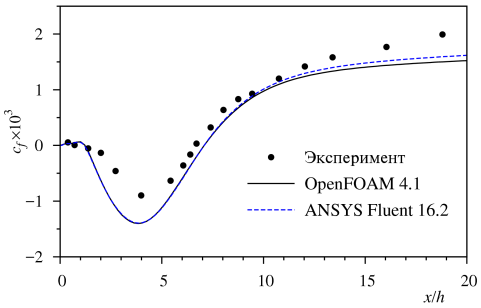

Рис. 4.16. Распределение коэффициента трения на нижней стенке

Рассмотрим теперь вопрос решения уравнения переноса тепла при помощи функционального объекта scalarTransport. Для добавления в процесс решения уравнения конвективно-диффузионного переноса пассивного скаляра необходимо добавить в файл `system/controlDict` дополнительную секцию functions следующим образом:

Часть файла system/controlDict:

```
functions
{
    temperature
    {
        type          scalarTransport;
        libs          ("libsolverFunctionObjects.so");
        field         T;
        alphaD        1.43;
        alphaDt       1.18;
    }
}
```

Здесь temperature — название секции (может быть любым), type — тип функционального объекта (в данном случае scalarTransport, т.е. решение уравнение переноса), field — название поля, для которого решается уравнение переноса, alphaD и alphaDt — значения молекулярного α_D и турбулентного α_{Dt} диффузионных коэффициентов переноса скаляра[1] для вычисления суммарного коэффициента диффузии по формуле:

$$D_\Sigma = D + D_t = \alpha_D \nu + \alpha_{Dt} \nu_t.$$

По сути $\alpha_D = 1/Sc$, т.е. величина, обратная числу Шмидта $Sc = \nu/D$. В задаче переноса тепла вместо числа Шмидта используется число Прандтля $Pr = \nu/D_T$, $D_T = \lambda/(\rho C)$ — коэффициент температуропроводности. Поскольку в эксперименте исследовалось течение воздуха, для которого $Pr = 0.7$, то $\alpha_D = 1/Pr \approx 1.43$. Что касается турбулентного числа Прандтля, то эта величина (так же, как и турбулентная вязкость ν_T) является «искусственной» и исходит из предположения о справедливости выполнения гипотезы Буссинеска также и для переноса тепла за счёт турбулентных пульсаций, и более того, предположения (не всегда справедливого) о подобности процессов турбулентного переноса импульса и тепла. Эти два предположения означают, что отношение ν_t и коэффициента турбулентной температуропроводности D_{Tt} является константой, называемой турбулентным числом Прандтля. Его значение определяется экспериментальным путём и может варьироваться в зависимости от задачи, но в основном выбирается

[1] В настоящее время возможность задания этих коэффициентов доступна только для текущей разрабатываемой версии OpenFOAM-dev; в OpenFOAM версий 4.1 и ниже можно задавать только постоянный коэффициент диффузии D, либо (в случае его отсутствия) неявно предполагается $\alpha_D = \alpha_{Dt} = 1$

порядка 0.9; в данном расчёте используется значение $Pr_t = 0.85$, поэтому $\alpha_{Dt} = 1/Pr_t \approx 1.18$.

После определения функционального объекта в файле system/contrlDict необходимо задать параметры численных схем для введённого уравнения и начальные и граничные условия для T. Параметры численных схем задаются аналогично другим уравнениям; для конвективного слагаемого «div(phi,T)» можно использовать схему linearUpwind, как в примере ниже.

```
Часть файла system/fvSchemes:
divSchemes
{
    default         none;
    div(phi,U)      bounded Gauss linearUpwindV grad(U);
    div(phi,k)      bounded Gauss upwind;
    div(phi,omega)  bounded Gauss upwind;
    div((nuEff*dev2(T(grad(U))))) Gauss linear;
    div(phi,T)      bounded Gauss linearUpwind grad(T);
}
```

Граничные условия на температуру необходимо задать в соответствии с экспериментом, где на входе подавался поток с постоянной температурой T_0 =300 K, все стенки, кроме нижней, были теплоизолированы, а на нижней задавался постоянный тепловой поток $q_w = 270$ Вт/м^2. Здесь, также как и для течения, можно использовать безразмерную постановку, поскольку экспериментальные результаты приведены в безразмерной форме в виде распределения числа Стэнтона St вдоль нижней стенки, определяемого через число Нуссельта Nu, число Рейнольдса и число Прандтля как:

$$St = \frac{Nu}{Re_h \cdot Pr} = \frac{q_w h}{\lambda \Delta T} \frac{1}{Re_h Pr},$$

где $\Delta T = T_w - T_0$, T_w — температура обогреваемой стенки.

В OpenFOAM нет условия теплового потока на стенке, но можно задать фиксированный градиент, т.е. по сути величину q_w/λ. Поэтому во-первых, задавать λ нигде не требуется, а во-вторых, значение q_w/λ можно брать произвольным, поскольку на число Стэнтона оно не влияет: получающийся перепад температур ΔT прямо пропорционален q_w/λ. Также значение

температуры на входе T_0 не влияет на решение уравнения теплопереноса, поэтому можно для простоты задавать на входе $T_0 = 0$, как бы неявно предполагая, что решается уравнение для приращения температуры по отношению ко входной.

С учётом вышесказанного граничные условия для T можно задавать как представлено ниже. Чтобы не пересчитывать задачу заново, можно стартовать с последней итерации (в настоящем расчёте это 500).

```
Часть файла 500/T:
dimensions      [0 0 0 1 0 0 0];
internalField   uniform 0;
boundaryField
{
    inlet
    {
        type            fixedValue;
        value           uniform 0;
    }
    outlet
    {
        type            zeroGradient;
    }
    wall_bottom
    {
        type            fixedGradient;
        gradient        100.0;
    }
    walls_other
    {
        type            zeroGradient;
    }
    fb { type empty; }
}
```

Уравнение для T сводится довольно быстро, достаточно несколько десятков итераций. Результаты расчёта приведены на рисунке 4.17 (совместно с результатом расчёта на Fluent; экспериментальные значения числа Стэнтона приведены в таблице 4.4). Отметим, что результаты расчётов, полученных при помощи Fluent и OpenFOAM, несколько расходятся, что связано с гораздо более сильной сеточной чувствительностью решения

для температуры по сравнению с расчётом коэффициента трения (см. рисунок 4.16). В частности, величина St чувствительна к значению температуры на обогреваемой стенке, которая должна «восстанавливаться» по значению градиента на ней. В пакете ANSYS Fluent температура на стенке берётся как температура в центре пристенной ячейки, тогда как в OpenFOAM значение температуры на стенке экстраполируется из центра по заданному градиенту. График «OpenFOAM, центр» построен как раз по значениям температур, взятых из центров пристенных ячеек, и он оказывается даже

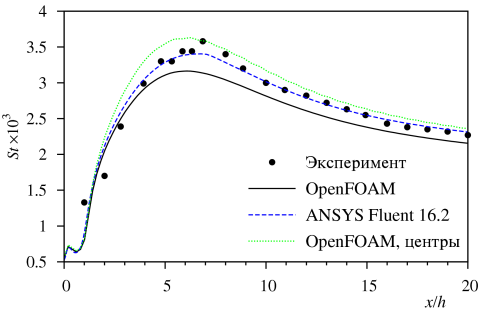

Рис. 4.17. Распределение числа Стэнтона на нижней стенке

Таблица 4.4. Распределение числа Стэнтона вдоль нижней стенки из [35]

x/h	$St\times10^3$	x/h	$St\times10^3$
1.00	1.33	12.00	2.82
2.00	1.70	13.00	2.72
2.80	2.39	14.00	2.63
3.93	2.99	14.93	2.55
4.80	3.30	16.00	2.43
5.33	3.30	17.00	2.38
5.87	3.44	18.00	2.35
6.33	3.44	19.00	2.32
6.87	3.58	20.00	2.27
8.00	3.40	21.00	2.25
8.87	3.20	22.00	2.12
10.00	3.00	23.13	2.09
10.93	2.90		

несколько выше, чем Fluent и эксперимент. При измельчении сетки результаты, полученных при помощи двух кодов, стремятся друг к другу. Таким образом, в данной задаче для получения сошедшихся по сетке результатов при моделировании теплообмена требуется гораздо более мелкая сетка, чем для разрешения течения.

В завершение отметим, что данное пособие не претендует на описание многочисленных аспектов моделирования турбулентности на основе RANS-подхода. Более детальную информацию о применимости тех или иных моделей турбулентности в разных течениях можно найти в [20].

5. Технология параллельных вычислений в OpenFOAM

5.1. Общие сведения о параллелизации

5.1.1. Предварительные замечания

Параллелизация вычислений — это динамично развивающееся и по сути единственное практическое средство ускорения численных расчётов прикладных задач. Параллелизация предполагает запуск задачи в параллельном режиме на каком-либо вычислительном комплексе, пригодном для этих целей. Наиболее распространенными среди подобных комплексов являются системы кластерного типа (суперкомпьютеры), которые состоят из отдельных узлов (представляющие собой многопроцессорные и многоядерные компьютеры), объединенных в единую высокоскоростную сеть, что позволяет достаточно легко наращивать вычислительную мощность системы.

В настоящее время существуют два основных подхода к распараллеливанию вычислений — параллелизм данных и параллелизм задач. Основная идея параллелизма данных заключается в том, что одна операция выполняется сразу над всеми элементами массива данных; при этом отдельные фрагменты такого массива могут обрабатываться на векторном процессоре и/или на разных процессорах параллельного узла суперкомпьютера. Распараллеливание в этом случае чаще всего выполняется уже на этапе компиляции; роль программиста обычно сводится к заданию директив параллельной компиляции или использованию специализированных языков (и соответствующих компиляторов) для параллельных вычислений.

Параллелизм задач подразумевает, что вычислительная задача разбивается на несколько относительно самостоятельных подзадач, каждая из которых загружается на отдельный процессор. Подзадачи периодически обмениваются результатами своей работы с помощью вызова специальных процедур. Программист должен «вручную» организовать разбиение зада-

чи на подзадачи, обеспечив при этом равномерность загрузки процессоров и эффективный обмен данными между ними, что требует определенных усилий на этапе разработки программы. Таким образом, по сравнению с подходом, основанным на параллелизме данных, данный подход является более трудоёмким, но обеспечивает большую гибкость в разработке программы и, как следствие, возможность достижения большего быстродействия.

В рамках концепции параллелизма задач различают два направления: MPMD (Multiple Program, Multiple Data) и SPMD (Single Program, Multiple Data), отличающиеся способом разбиения исходной задачи на подзадачи. В первом случае подразумевается наличие нескольких программ, выполняющих различные действия с разными объектами в составе общей задачи. В рамках более распространенной технологии SPMD на каждом процессоре параллельного узла суперкомпьютера запускается один и тот же программный код. В зависимости от номера запущенного процесса, код выполняет требуемые действия с данными, относящимися к этому процессу.

Распараллеливание с использованием параллелизма задач может осуществляться, например, при помощи библиотеки MPI (Message Passing Interface), которая является широко распространенной и хорошо документированной, а также имеет интерфейсы для различных языков программирования и доступна в свободных реализациях (например, OpenMPI или MPICH).

Разбиение общей задачи на отдельные подзадачи в вычислительной гидроаэродинамике обычно осуществляется путём пространственной декомпозиции расчётной сетки на блоки. При этом на каждом процессоре параллельного компьютера запускается один и тот же программный код, который выполняет вычисления только для блоков, относящихся к данному процессу. Обычно организация вычислений внутри блока не зависит от того, в каком процессе обрабатывается данный блок. Обмен данными между процессами осуществляется путём стыковки блоков (или другими словами, путём введения так называемых гало-ячеек — ячеек, общих для двух стыкуемых блоков). Помимо стыковки блоков, межпроцессорный обмен данными обычно требуется также для сбора информации по всем блокам сетки (например, при определении максимальных невязок), однако доля таких

операций в общем объеме межпроцессорного обмена обычно невелика. Поэтому при параллелизации кода особое внимание уделяют эффективности межпроцессорной передачи данных при стыковке блоков.

Разбиение задачи на блоки (т.е. декомпозиция расчётной сетки) может осуществляться как пользователем, так и в некотором роде автоматически. Понятно, что число запускаемых процессов ограничено числом блоков сетки. Кроме того, при неодинаковом размере блоков (числе ячеек) могут возникать сложности с обеспечением равномерного распределения вычислительной нагрузки между процессами. Поэтому при параллелизации важно учитывать такие моменты, как обеспечение равномерной загруженности процессов и минимизацию обменов между ними.

5.1.2. Количественные измерения качества паралеллизации

Очевидно, что расчёт какой-либо задачи в параллельном режиме должен занимать меньше времени, чем в последовательном, причём чем меньше это время, значит, тем эффективнее работает программа в параллельном режиме. Для количественно оценки степени эффективности параллелизации программы вводятся некоторые характеристики, которые должны отражать степень *масштабируемости* программного кода, т.е. зависимости ускорения расчёта от количества используемых процессоров/ядер (в дальнейшем вместо «процессора» или «ядра» будем использовать термин «процесс», подразумевая, что при MPI-параллелизации при запуске программы создаётся столько процессов, на сколько распараллелен расчёт). Наилучшая масштабируемость имеет место тогда, когда время расчёта уменьшается во столько раз, во сколько растёт число используемых процессоров/ядер, что, естественно, в реальности практически никогда не выполняется.

Таким образом, одной из основных количественных характеристик для оценки качества параллелизации является *ускорение*, т.е. величина

$$S_p = \frac{t_1}{t_p},$$

где t_1 — время расчёта задачи на одном процессе, а t_p — время расчёта той же задачи, но запущенной на p процессах. При хорошем качестве параллелизации ускорение S_p растёт пропорционально количеству процессов p, причём наилучший (идеальный) случай имеет место тогда, когда $S_p = p$,

т.е. запуск на p процессах даёт ускорение в p раз. Отметим, что в ряде тестов возможна даже ситуация, когда расчёт ускоряется больше, чем в p раз. Это связано с особенностями архитектуры компьютера (процессора и периферии). В частности, такой эффект наблюдается в случае, если в результате запуска на б*о*льшем числе процессов требуемые для расчёта данные для каждого процесса начинают помещаться в кэш процессора, что приводит к сильному ускорению расчёта, поскольку в настоящее время одним из основных ограничивающих быстродействие компьютера факторов является скорость доступа к основной оперативной памяти компьютера (именно поэтому производители процессоров стараются увеличить кэши разных уровней).

«Производной» величиной, связанной с параметром ускорения, является эффективность параллелизации, которая вычисляется по формуле:

$$E_p = \frac{t_1}{pt_p}.$$

Величина эффективности $E_p = 1$ соответствует наилучшей степени параллелизации. В реальности при росте числа процессов эта величина падает (в частности, из-за возрастающих накладных расходов на пересылку данных между процессами); на практике значение $E_p = 0.5 \div 0.8$ считается приемлемым, если при увеличении числа процессов её значение почти не меняется; это как раз говорит о хорошей масштабируемости по числу процессов. Хорошая параллелизация означает также, что эффективность параллелизации практически не меняется при увеличении размера расчётной сетки; это говорит о хорошей масштабируемости по размеру задачи.

Из-за сильного влияния особенностей архитектуры конкретного компьютера зачастую в разных ситуациях можно получить различные значения параметра ускорения S_p и эффективности E_p, поэтому вычисление этих параметров может представлять нетривиальную проблему. Существует несколько подходов к оценке эффективности параллелизации. Один из них, называемый также оценкой сильной масштабируемости, заключается в том, что берётся достаточно объёмная по вычислениям задача (с т.з. вычислительной гидродинамики это обычно означает, что размер сетки достаточно большой — от миллиона ячеек и выше), и затем запускается в однопроцессном режиме и многопроцессном на p процессах, где p посте-

пенно увеличивается. Рассчитывается время каждого запуска и вычисляются параметры S_p и E_p. Максимальное значение p, очевидно, равняется число процессоров/ядер, доступных для тестирования (при тестировании на кластерной системе, узлы которой представляют собой «обычные» компьютеры, соединённые высокоскоростной сетью, опять-таки из-за архитектурных особенностей возможно ситуация, когда расчёт на одном узле при использовании p процессов оказывается медленнее, чем расчёт на двух узлах, на каждом из которых используется по $p/2$ процессов, если p близко к максимально доступному числу ядер на узле).

Другой подход, называемый вычислением эффективности параллелизации при слабой масштабируемости, заключается в том, чтобы при запусках на различном количестве процессов обеспечивать одинаковый объём вычислений на каждом из них (для вычислительной гидродинамики это означает сохранение числя ячеек на процесс при увеличении p). Обозначая в этой ситуации время работы в однопроцессном режиме по-прежнему за t_1, а время работы каждой задачи, запущенной в режиме p процессов, через T_p, эффективность параллелизации можно рассчитать по формуле

$$E_{weak} = \frac{t_1}{T_p}.$$

По-прежнему, наилучшая эффективность соответствует $E_{weak} = 1$. Преимущество такого подхода, очевидно, заключается в том, что в связи с одинаковым объёмом вычислений на каждом процессе влияние особенностей архитектуры компьютера здесь мннимизировано. Недостатки также присутствуют, можно отметить следующие: сложность постановки задачи при запуске на разном количестве процессов (по сути, необходимо каждый раз перестраивать расчётную сетку); если используются, например, итерационные алгоритмы решения СЛАУ, то поскольку их эффективность напрямую зависит от размера матрицы системы (т.е. от общего числа ячеек сетки), необходимо аккуратно подходить к вопросу сравнения двух расчётов, полученных с использованием различных по размеру сеток. В частности, здесь необходимо измерять время расчёта одного и того же числа итераций при решении СЛАУ, а не сводить задачу до указанного уровня невязки СЛАУ.

Немаловажным аспектом является также количество оперативной памяти на узле вычислительного кластера: не всякую задачу можно запустить в однопроцессном режиме или даже только на одном узле. В этом случае оценку ускорения и эффективности параллелизации проводят путём сравнения с запуском на наименьшем количестве процессов (такой подход будет использоваться в разделе 5.2.3).

Более подробно о различных методах параллелизации путём декомпозиции, а также некоторые теоретические оценки эффективности параллелизации можно найти в книге [36].

5.2. Параллелизация в OpenFOAM

В OpenFOAM параллелизация основана на декомпозиции расчётной сетки на блоки; для обмена информацией между процессами используется библиотека MPI (как правило, её открытая реализация OpenMPI). Параллелизация внутри кода выполнена таким образом, что практически все программы из состава дистрибутива OpenFOAM позволяют осуществлять запуск в параллельном режиме, и в случае разработки собственного решателя программисту-пользователю обычно не требуется задумываться о параллелизации — она выполняется в некотором смысле «автоматически». Т.е. любой решатель может работать и в однопроцессном, и в параллельном режиме.

5.2.1. Декомпозиция расчётной области

Для декомпозиции расчётной сетки, т.е. разбиения её на блоки, в OpenFOAM сущсетвует специальная утилита decomposePar. Для её работы необходимо задать параметры разбиения сетки в файле `system/decomposeParDict`. В OpenFOAM имеется несколько методов декомпозиции, отметим следующие:

- simple (простой способ): сетка разбивается на заданное пользователем количество блоков в трёх координатных направлениях (x, y, и z); количество ячеек, которое должно получится в каждом блоке, подбирается автоматически на основании указанного пользователем допуска delta;

- hierarchical (иерархический): аналогично simple, но в отличие от него пользователь может указать, в каком порядке разбивать сетку (опция order задаёт порядок направления, например, «order xyz» означает, что сетка сначала бьётся вдоль x, затем, вдоль y и затем вдоль z, что соответствует методу simple);
- scotch: автоматическое разбиение при помощи библиотеки scotch;
- metis: автоматическое разбиение при помощи библиотеки metis, которую необходимо установить в систему самостоятельно;
- manual: соответствие «ячейка – процессор» задаётся пользователем вручную для каждой ячейки сетки.

Пример содержимого файла `system/decomposeParDict` приведён ниже.

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      decomposeParDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
numberOfSubdomains 4;
method             simple;
simpleCoeffs
{
    n              (2 2 1);
    delta          0.001;
}
```

Опция numberOfSubdomains указывает общее количество блоков, на которое должна быть разбита сетка. Здесь в качестве метода разбиения используется simple; в секции simpleCoeffs указываются параметры для него. Параметр «n» задаёт количество разбиений вдоль каждого координатного направления (в виде вектора); в примере сетка бьётся вдоль направления x на два блока, потом каждый из них бьётся вдоль направления y также на два блока (в сумме получается 4 блока, т.е. произведение компонент «n» должно равняться numberOfSubdomains).

После работы утилиты decomposePar в каталоге задачи создаётся подкаталоги с именами processor0, processor1 и т.д. Внутри каждого такого каталога повторяется файловая структура, как для исходного каталога задачи: имеется каталог constant, в котором находится часть сетки, принадлежащая соответствующему процессу, а также каталоги с меткой времени (например, 0), в которых лежат поля для соответствующей части сетки. Можно запустить утилиту paraFoam в каждом из созданных каталогов processor0, processor1 и т.д., и тогда будет отображаться только та часть сетки, которая принадлежит соответствующему процессу.

Если задача уже была разбита на блоки утилитой decomposePar, но требуется заново «разбить» поля величин, то для этих целей необходимо запустить утилиту со следующей опцией:

```
decomposePar -fields
```

При разбиении можно также указать конкретный момент времени, для которого «разбивать» поля, при помощи опции -time.

5.2.2. Запуск решателя и «сборка» решения

Запуск решателя (или утилиты) OpenFOAM в параллельном режиме осуществляется при помощи стандартной команды MPI mpiexec, за которой следует указать число процессов ключом -np N, далее название решателя OpenFOAM с обязательной опцией -parallel. Например, запуск решателя icoFoam для расчёта разбитой на 4 блока задачи осуществляется следующей командой:

```
mpiexec -np 4 icoFoam -parallel
```

При этом на экран выводится информация о запуске (аналогично описанной в разделе 2.3.4), содержащая в том числе сведения, связанные с параллельным выполнением, как показано ниже.

```
/*---------------------------------------------------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  4.x                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
```

```
\*---------------------------------------------------------------------------*/
Build  : 4.x-3d754f4a122d
Exec   : icoFoam -parallel
Date   : Mar 07 2017
Time   : 13:00:45
Host   : "n10"
PID    : 532622
Case   : /home/smirnovskiy/OpenFOAM/smirnovskiy-4.x/run/cavity
nProcs : 4
Slaves :
3
(
"n10.532623"
"n10.532624"
"n10.532625"
)
Pstream initialized with:
    floatTransfer      : 0
    nProcsSimpleSum    : 0
    commsType          : nonBlocking
    polling iterations : 0
sigFpe : Enabling floating point exception trapping (FOAM_SIGFPE).
fileModificationChecking : Monitoring run-time modified files using
    ↪ timeStampMaster
allowSystemOperations : Allowing user-supplied system call operations
```

В строке «Host» содержится имя узла кластера, на котором запущен головной процесс icoFoam; в «Slaves» указан список дочерних процессов в формате «<имя узла>.<идентификатор процесса>». В строке «nProcs» указывается общее число процессов.

В течение параллельного расчёта каждый процесс осуществляет вывод полей переменных в «свой» каталог `processorN`, в подкаталог с соответствующей временной меткой. Чтобы «собрать» решение в единый блок, необходимо воспользоваться утилитой reconstructPar; можно указать конкретные моменты времени, для которых производится «сборка» решения, при помощи опции «-time», либо «собрать» поля для наиболее позднего момента времени при помощи опции «-latestTime». Кроме того, можно указать список полей, которые необходимо «собрать», опцией «-fields». Напри-

мер, следующая команда произведёт «сборку» полей U и p для моментов времени 10 и 30:

```
reconstructPar -fields '(U p)' -time '10,30'
```

Если требуется переразбить уже разбитую при помощи decomposePar задачу на другое количество процессов, для этих целей в OpenFOAM имеется утилита redistributePar, которая работает только в параллельном режиме. Например, для того, чтобы переразбить на 8 процессов разбитую на 4 процесса задачу необходимо выставить соответствующие для 8 процессов параметры в `system/decomposeParDict` (в частности, выставить numberOfSubdomains 8) и запустить команду:

```
mpiexec -np 8 redistributePar -parallel
```

Также можно переразбить задачу с уменьшением числа процессов. Например, при уменьшении с 4 до 2 необходимо запустить команду

```
mpiexec -np 4 redistributePar -parallel
```

То есть число MPI-процессов при запуске redistriburePar должно равняться максимальному из старого и нового количества процессов.

5.2.3. Пример расчёта с использованием параллелизации

В настоящее время задачи вычислительной гидродинамики, которые требуют параллельных вычислений, можно условно разбить на два класса:

- задачи со сложной геометрией расчётной области и/или задачи мультидисциплинарной физики (например, течения с большим количеством химических реакций, излучением, магнитными полями и т.д.);
- моделирование турбулентных течений при помощи вихреразрешающих подходов.

Для первого класса задач размер расчётной сетки может быть не слишком большой, однако, на каждой итерации/временном шаге требуется производить много вычислений (например, решать системы обыкновенных дифференциальных уравнений в задачах с учётом химической кинетики). Задачи из второго класса обычно характеризуются не слишком сложной расчёт-

ной областью и относительно малым числом решаемых уравнений, однако, предполагают решение нестационарных уравнений с детальным пространственным разрешением, позволяющим аккуратно прописывать турбулентные структуры в потоке на интересующих масштабах (подробнее о моделировании турбулентности см. раздел 4.2.1). Чем детальнее требуется прописать турбулентные пульсации, тем мельче должна быть сетка и шаг по времени; в случае прямого численного моделирования турбулентности (DNS), когда разрешаются все масштабы турбулентных пульсаций, грубая оценка даёт значение требуемого числа ячеек сетки N порядка числа Рейнольдса в кубе: $N \sim Re^3$, а число шагов по времени, требуемое для получения репрезентативной выборки, составляет порядка Re. Таким образом, именно DNS при больших числах Рейнольдса в настоящее время требует наиболее внушительного объёма вычислений и соответствующих вычислительных ресурсов. Поэтому в качестве примера расчёта при помощи OpenFOAM с использованием параллелизации будет рассмотрена задача прямого численного моделирования, а именно, задача конвекции Рэлея-Бенара в кубической полости при высоком значении числа Рэлея.

Свободная конвекция в кубической полости является предметом детальных исследований во всём мире, как экспериментальных, так и численных (в качестве примера приведём работы [37, 38]). Отличие в постановке задачи от описанной в разделе 4.1.3 заключается в первую очередь в том, что нагрев происходит снизу, а охлаждение — сверху; кроме того, для моделирования турбулентной конвекции необходимо использовать трёхмерную постановку. В этом случае возникает неустойчивая стратификация температуры (и, соответственно, плотности), приводящая к возникновению подъёмных и нисходящих течений, которые при больших числах Рэлея носят случайных характер; при этом в полости может образовываться так называемая крупномасштабная конвективная ячейка, однако, в отличие от случая нагрева и охлаждения через боковые стенки, её положение недетерминировано. Кроме того, в случае нагрева сбоку вдоль боковых стенок развиваются тонкие свободно-конвективные пограничные слои, и точность разрешения течения во всей полости определяется в первую очередь качеством разрешения этих слоёв. В конвекции Рэлея-Бенара вблизи верхней и нижней стенок также существуют пограничные слои, но как показывает

практика расчётов, теплоперенос определяется в первую очередь крупномасштабными вихревыми структурами в потоке, а не детальным прописыванием пристенных областей. В этой связи, при сходных числах Рэлея для конвекции Рэлея-Бенара не требуется такого сильного сгущения сетки к нижней и верхней поверхностям, как к боковым стенкам при боковом нагреве.

Все расчёты проводились с использованием вычислительных ресурсов суперкомпьютерного центра (СКЦ) «Политехнический» Санкт-Петербургского политехнического университета Петра Великого (http://scc.spbstu.ru). СКЦ «Политехнический» располагает высокопроизводительными вычислительными системами разной архитектуры с общей пиковой производительностью более 1.2 Пфлопс[1]. В СКЦ представлены три суперкомпьютера, из которых для расчётов использовался кластер «Политехник – РСК Торнадо» с пиковой производительностью 938 Тфлопс[2]. Этот кластер содержит 668 узлов с двумя процессорами Intel Xeon E5 2697 v3 на узле, из них 56 узлов имеют по два ускорителя вычислений NVIDIA K40 на каждом узле. Один процессор Intel Xeon E5 2697 v3 имеет 14 ядер, так что на узле можно запускать до 28 процессов в параллельном режиме. Таким образом, максимальное число ядер составляет 18704. Узлы кластера объединены высокоскоростной сетью InfiniBand FDR с полосой пропускания до 56 Гбит/с; управление данными и их хранение обеспечивают распределенная файловая система Lustre и система хранения Xyratex ClusterStor 6000 (ёмкость 1.1 Пбайт).

Как было отмечено в разделе 5.1.2, для оценки эффективности распараллеливания могут применять две методики: оценка сильной и слабой масштабируемости. Рассмотрим сначала вариант слабой масштабируемости. Для этого необходимо сформулировать ряд постановок для запуска на разном числе процессов так, чтобы в каждой их них на один процесс приходился бы примерно одинаковый объём вычислений. С учётом того, что рассматривается задача прямого численного моделирования нестационарной свободной конвекции Рэлея-Бенара, измельчение сетки для одной

[1] Единица измерения «флопс» — калька с английского FLOPS, FLoating point Operations Per Second, т.е. число операций с плавающей точкой в секунду, подробнее см. https://en.wikipedia.org/wiki/FLOPS. Приставка П (Пета) означает 10^{15}

[2] Приставка Т (Тера) означает 10^{12}

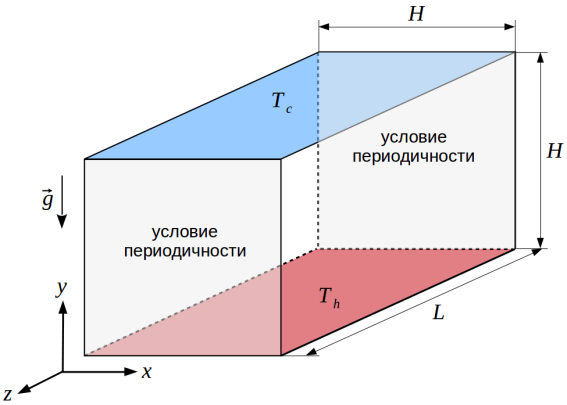

Рис. 5.1. Постановка задачи о конвекции Рэлея-Бенара в «бесконечном канале»

и той же расчётной области может приводить к изменению характера движения и, соответственно, объёму вычислений. Поэтому для оценки слабой масштабируемости используется несколько другая постановка, а именно: нестационарная конвекция Рэлея-Бенара в бесконечном канале квадратного сечения, где бесконечный канал заменяется на канал конечной длины с условиями периодичности на боковых поверхностях, см. рисунок 5.1; длина «канала» L варьируется в зависимости от разбиения на требуемое число процессов.

Для постановки с минимальным рассмотренным числом ячеек сетки использовалась длина «канала», равная его высоте и ширине ($L = H$), т.е. расчётная область представляла собой куб, на двух противоположных гранях вдоль оси z которого ставилось условие периодичности; использовалась сетка размером $64 \times 64 \times 70$ ячеек (70 — число ячеек вдоль «канала»), сгущённая к стенкам и равномерная вдоль «канала». Данная задача рассчитывалась на одном вычислительном узле кластера «Политехник – РСК Торнадо» с разбиением на 28 процессов, причём для разбиения использовался алгоритм hierarchical: 2 части вдоль x, затем 2 части вдоль y и 7 частей вдоль z (вдоль «канала»). Это обеспечивает одинаковое число ячеек на каждом ядре (по 10240 ячеек). При использовании двух узлов кластера длина «канала» бралась равной удвоенной высоте ($L = 2H$), размер рас-

чётной сетки составлял $64 \times 64 \times 140$ ячеек; сетка разбивалась уже на 14 частей вдоль z, и на каждое ядро опять приходилось одинаковое число ячеек, равное 10240. Аналогичная процедура удвоения длины «канала» (и соответствущего удвоения числа ячеек сетки) проводилась вплоть до 256 узлов, т.е. максимальное число ячеек сетки составило $64 \times 64 \times 17920$ (около 73 млн ячеек). Поскольку сеточное разрешение было довольно небольшое, расчёты проводились для умеренного значения числа Рэлея $Ra = 10^6$ при числе Прандтля $Pr = 7$; использовался решатель buoyantBoussinesqPimpleFoam.

Для замера времени счёта для всех вариантов длины «канала» использовалась нестационарная постановка при старте от начального однородного поля температуры (равного полусумме температур на горячей нижней и холодной верхней стенках) с отсутствием движения. Расчёт вёлся до безразмерного момента времени 2000 с шагом 0.2 (такой шаг обеспечивает максимальное значение числа Куранта в области не более 1). При этом выход на установившийся режим нестационарной конвекции происходил примерно за 200 единиц времени. Для конвективных слагаемых в уравнении движении использовалась схема QUICKV, в уравнении энергии — QUICK. Для решения уравнений баланса импульса и энергии использовался линейный решатель PBiCGStab с предобуславдивателем DILU (с критерием сходимости relTol 0.1), а для решения уравнения Пуассона для давления — GAMG со сглаживателем DIC, с двумя PISO-итерациями (relTol 0.05 на первой и relTol 0.02 на второй) и одной глобальной итерацией PIMPLE (т.е. использовался чистый метод PISO). В качестве примера вывода в процессе работы решателя ниже приводится вывод для последнего шага по времени для варианта запуска на 1 узле (сетка $64 \times 64 \times 70$ ячеек). Как видно, в этом случае общее время работы программы составило 918.87 с.

```
Courant Number mean: 0.125904 max: 0.599608
Time = 2000

PIMPLE: iteration 1
DILUPBiCGStab:  Solving for Ux, Initial residual = 0.00481362, Final
    ↪ residual = 2.72351e-05, No Iterations 1
DILUPBiCGStab:  Solving for Uy, Initial residual = 0.00527111, Final
    ↪ residual = 2.90788e-05, No Iterations 1
DILUPBiCGStab:  Solving for Uz, Initial residual = 0.00739047, Final
    ↪ residual = 4.04019e-05, No Iterations 1
```

```
DILUPBiCGStab:  Solving for T, Initial residual = 0.0109449, Final
    ↪ residual = 5.20137e-05, No Iterations 1
GAMG:  Solving for p_rgh, Initial residual = 0.0175522, Final residual =
    ↪ 0.000304063, No Iterations 1
time step continuity errors : sum local = 1.22828e-05, global = 9.1064e
    ↪ -19, cumulative = -6.49291e-14
GAMG:  Solving for p_rgh, Initial residual = 0.000317303, Final residual =
    ↪  2.63693e-06, No Iterations 3
time step continuity errors : sum local = 1.06737e-07, global = 9.00518e
    ↪ -19, cumulative = -6.49282e-14
ExecutionTime = 918.87 s  ClockTime = 924 s
```

В таблице 5.1 приведены данные, полученные в тестах по определению эффективности параллелизации при слабом масштабировании. Время расчёта бралось как разница между полным временем работы программы и временем работы до конца первого шага по времени, чтобы исключить из оценки промежуток времени, связанный с началом работы решателя (чтение сетки, подготовительная работа и т.п.). Эффективность E_{weak} определялась как отношение времени расчёта на 1 узле ко времени расчёта на p узлах. Вплоть до использования 32 узлов эффективность достаточно высокая и варьируется в диапазоне $0.82 \div 0.94$. При переходе к 64 узлам происходит уменьшение E_{weak} до значения ≈ 0.7, а вот расчёт на 256 узлах уже показывает сильный «провал» в эффективности до ≈ 0.5. Последнее обстоятельство может быть связано как с особенностями OpenFOAM, так и с архитектурой используемого кластера.

Для исследования влияния загруженности каждого узла для той же постановки были также приведены расчёты с 14 процессами на узел (т.е. на каждый процесс приходилось уже по 20480 ячеек). Полученные данные здесь не приводятся, а будут представлены на сводном графике далее. Отметим здесь только тот момент, что при неполной загрузке узла (14 процессов вместо 28) время расчёта замедляется не в два раза, а примерно в $1.5 \div 1.8$ раза. Это говорит о том, что полная загрузка узла приводит к некоторому уменьшению его производительности (этот факт также отмечался в разделе 5.1.2), что будет также видно при тестировании сильной масштабируемости.

[1] На каждом расчётном узле запускалось 28 процессов

Таблица 5.1. Результаты тестирования слабой масштабируемости

Число расчётных узлов[1]	Размер сетки (число ячеек)	Время расчёта $(10^4 - 1)$ шагов по времени, с	Эффективность E_{weak}
1	2.9×10^5	918.5	
2	5.7×10^5	1044.8	0.88
4	1.1×10^6	979.1	0.94
8	2.3×10^6	1112.2	0.82
16	4.6×10^6	1043.8	0.88
32	9.2×10^6	1119.8	0.82
64	1.8×10^7	1322.4	0.69
128	3.7×10^7	1332.9	0.69
256	7.3×10^7	1782.5	0.52

Обратимся теперь к вопросу тестирования эффективности параллелизации при сильной масштабируемости. Для этого рассмотрим прямое численное моделирования конвекции Рэлея-Бенара в кубической полости при относительно высоком числе Рэлея $Ra = 2 \times 10^9$ и числе Прандтля $Pr = 6.1$; данные параметры взяты для соответствия эксперименту из [37], в котором при помощи PIV-технологии исследовались течение и пульсационные характеристики турбулентной конвекции воды (на рисунке 5.2 из [37] приведена схема эксперимента).

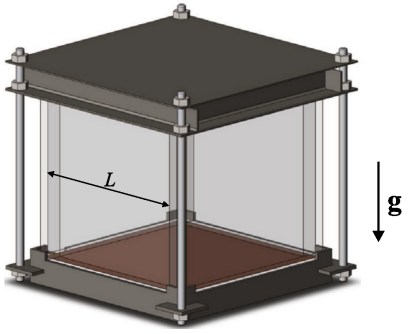

Рис. 5.2. Схема эксперимента для исследования конвекции Рэлея-Бенара (из [37])

Для численного моделирования строилась равномерная расчётная сетка размером $600 \times 600 \times 600$ ячеек (т.е. 216 млн ячеек). Как и ранее, использовался решатель buoyantBoussinesqPimpleFoam. Перед процедурой измерения

эффективности параллелизации проводился расчёт от начального однородного распределения до момента времени 700 для получения полностью развитой установившейся турбулентной конвекции. При этом для конвективных слагаемых в уравнениях баланса импульса и энергии использовалась схема четвёртого порядка cubic, для временной производной — схема backward. Шаг по времени выбирался исходя из условия максимального числа Куранта 0.5. Остальные параметры брались такими же, как и в предыдущем случае. В тестах для определения эффективности параллелизации определялось время расчёта от 700 до 705 безразмерных физических секунд (без первого шага по времени аналогично предыдущему случаю). В среднем число шагов по времени составляло порядка 500. Для декомпозиции сетки использовался метод hierarchical.

В качестве примера вывода в процессе работы решателя ниже приводится вывод для первого шага по времени (от момента времени 700) при запуске на 128 узлах кластера «Политехник – РСК Торнадо».

```
Courant Number mean: 0.129713 max: 0.491224
deltaT = 0.0101783
Time = 700.010178

PIMPLE: iteration 1
DILUPBiCGStab:  Solving for Ux, Initial residual = 0.000872203, Final
    ↪ residual = 5.7564e-06, No Iterations 1
DILUPBiCGStab:  Solving for Uy, Initial residual = 0.000874381, Final
    ↪ residual = 5.84679e-06, No Iterations 1
DILUPBiCGStab:  Solving for Uz, Initial residual = 0.00123408, Final
    ↪ residual = 8.1828e-06, No Iterations 1
DILUPBiCGStab:  Solving for T, Initial residual = 0.00327297, Final
    ↪ residual = 2.40434e-05, No Iterations 1
GAMG:  Solving for p_rgh, Initial residual = 0.0180908, Final residual =
    ↪ 0.000192636, No Iterations 2
time step continuity errors : sum local = 3.658e-07, global = 2.83703e-15,
    ↪  cumulative = 2.83703e-15
GAMG:  Solving for p_rgh, Initial residual = 0.000224758, Final residual =
    ↪  1.55334e-06, No Iterations 4
time step continuity errors : sum local = 2.94966e-09, global = 2.83705e
    ↪ -15, cumulative = 5.67408e-15
ExecutionTime = 6.89 s  ClockTime = 22 s
```

Таблица 5.2. Результаты тестирования сильной масштабируемости, сетка 216 млн ячеек

Число процессов	Время расчёта, с (28 процессов на узел)	Ускорение S_p	Эффективность E_p
56	37764.9		
224	11108.9	3.40	0.85
448	4983.0	7.58	0.95
896	2580.6	14.6	0.91
1792	1287.1	29.3	0.92
3584	627.1	60.2	0.94
Число процессов	Время расчёта, с (14 процессов на узел)	Ускорение S_p	Эффективность E_p
28	65997.4		
112	18218.8	3.62	0.91
224	8176.2	8.07	1.01
448	4078.5	16.2	1.01

В таблице 5.2 приведены сводные данные для разных запусков. Для оценки влияния загруженности узлов на скорость счёта применялось два способа распределения процессов по узлам: расчёт с 28 процессами на узел и с 14 процессами на узел. В связи с большим размером сетки для запуска на 56 и 28 процессах использовалось два узла с количеством оперативной памяти на каждом 256 Гбайт, тогда как остальные расчёты были выполнены на узлах с 64 Гб памяти. Отметим, что здесь для вычисления ускорения S_p и эффективности E_p использовалось время, полученное при запуске на минимальном числе процессов.

Полученные данные по эффективности параллелизации рассмотренных задач сведены в графике на рисунке 5.3. Отметим следующие характерные особенности. Во-первых, при распараллеливании до числа процессов около 1000 наблюдается сильный разброс в данных: значение эффективности E_p колеблется от 0.8 до 1. Во-вторых, для слабой масштабируемости наблюдается тенденция к понижению эффективности при увеличении числа процессов, особенно сильное падение имеет место при числе процессов

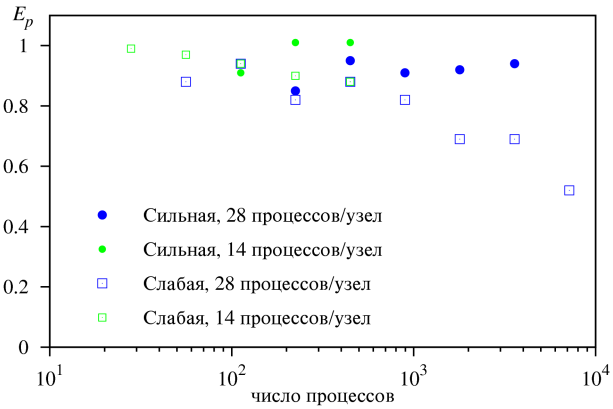

Рис. 5.3. Сводный график по результатам тестирования сильной и слабой масштабируемости

более 1000, тогда как для сильной масштабируемости, наоборот, с ростом числа процессов имеется тенденция к увеличению эффективности. Такое поведение скорее всего связано со значительным увеличением объёма пересылаемых при помощи MPI данных в случае слабой масштабируемости, поскольку число ячеек на процесс (а следовательно, и объём вычислений) фиксировано и достаточно мало. В случае же сильной масштабируемости объём вычислений, приходящийся на один процесс, уменьшается линейно с ростом числа процессов, что, видимо, способствует большему ускорению расчёта, чем увеличение задержек, связанных с увеличивающимся объём пересылок между процессами. В целом, даже при расчёте на 7168 процессах эффективность составляет 0.5, что является вполне неплохим результатом.

6. Программирование с использованием библиотек OpenFOAM. Создание решателей

Как отмечалось в предыдущих главах, пакет OpenFOAM является не столько набором готовых решателей, сколько мощным инструментом программиста, предназначенным для создания собственных решателей и библиотек на основе имеющихся в OpenFOAM. Причём благодаря иерархической организации кода OpenFOAM программисту не требуется досконально знать все аспекты программирования в OpenFOAM, а только тот «уровень», который достаточен для выполнения той или иной задачи. В этом смысле не обязательно быть «настоящим программистом», чтобы создавать простые решатели, хотя, безусловно, для более глубокого понимания внутреннего устройства библиотек OpenFOAM требуется обладать довольно хорошим уровнем программирования.

В данной главе будет представлены лишь базовые возможности программирования в среде OpenFOAM (как это можно условно называть), поэтому в целом данная глава не требует каких-либо особых программистских навыков, достаточно иметь общие представление о синтаксисе и возможностях языка программирования C++, на котором и написаны библиотеки OpenFOAM, а также понимать парадигму объектно-ориентированного программирования (ООП). В разделе 6.1 описаны общие сведения об организации кода, а в разделе 6.2 пошагово рассказывается о процессе создания собственного решателя для относительно простой физической задачи.

6.1. Общие сведения об организации кода OpenFOAM

Библиотеки OpenFOAM написаны на C++ с использованием объектно-ориентированного программирования (ООП). Такой подход позволяет создавать несколько уровней абстракций таким образом, что при работе на верхних уровнях программисту не требуется знать детали реализации на нижних уровнях.

Коды всех библиотек находятся в каталоге $WM_PROJECT_DIR/src; для быстрого доступа к этому каталогу имеется переменная окружения FOAM_SRC и команда src, выполняющая переход в этот каталог.

В OpenFOAM используется модульная организация кода. Это означает, что весь код разбит на совокупность относительно небольших условно независимых блоков, каждый из которых компилируется отдельно в библиотеку, причём библиотеки верхнего уровня могут использовать библиотеки нижнего, но не наоборот. Для сборки (компиляции) кода используется собственная система сборки wmake, основанная на GNU make. Несколько более подробно о том, как устроен процесс компиляции в OpenFOAM, будет говориться далее.

Исходные коды располагаются в файлах с расширениями .C и .H. Если первые содержат собственно компилируемый код программы (в частности, реализацию методов класса), то роль вторых в OpenFOAM несколько «размыта»: это и заголовочные файлы для описания классов (причём простые методы могут реализовываться непосредственно при объявлении класса), и файлы с кодами inline-методов, и также просто куски программы, вынесенные в отдельные файлы (особенно часто в этой роли выступают H-файлы в кодах решателей $WM_PROJECT_DIR/applications/solvers).

Для упрощения изучения структуры кода OpenFOAM и иерархии классов имеется документация программиста в формате HTML, построенная при помощи системы автоматического документирования Doxygen, которая генерирует документацию на основе набора исходных текстов. Можно самостоятельно создать эту документацию из дистрибутива OpenFOAM как описано в $WM_PROJECT_DIR/doc/Doxygen, либо использовать готовую по адресу http://cpp.openfoam.org.

6.1.1. Уровни абстракций и иерархия классов

Код OpenFOAM структурирован таким образом, чтобы обеспечивать различные уровни абстракции. Под «уровнем абстракций» понимается степень детализации объекта. Например, в качестве такого (высокоуровневого) объекта может выступать поле скалярной, векторной или тензорной величины. Тогда, абстрагируясь от деталей реализации, можно определить различные операции над полями, например, алгебраические (сумма полей,

скалярное и векторное произведение и т.п.). С точки зрения кода C++ каждое такое поля является экземпляром соответствующего класса, для которого определены некоторые операции.

В качестве примера другого уровня абстракции можно указать расчётную сетку, состоящую из ячеек с теми или иными топологическими и геометрическими свойствами, но для программиста на этом уровне абстракции сетка представляет собой единой целое, т.е. все данные и методы, необходимые для работы с сеткой (например, вычисление объёмов ячеек), *скрыты* (или иначе, *инкапсулированы*) от программиста на этом уровне абстракции. Отметим, что поля величин всегда задаются на расчётной сетке, т.е. с точки зрения программы класс, описывающий поля величин, содержит класс, описывающего расчётную сетку, т.е. с т.з. программиста расчётная сетка лежит на более низком уровне абстракции (инкапсулирована).

Подобная организация уровней абстракций позволяет программисту писать алгоритмы, которые используют, например, расчётную сетку как целое, т.е. как аргумент при вызове какой-либо функции, не отвлекаясь на детали того, что именно скрыто за объектом «сетка».

Уровни абстракции в OpenFOAM можно условно разделить на три группы:

1) Системный уровень. Наиболее низкий уровень. Включает в себя работу с файлами, работу с такими примитивами, как массивы, списки, указатели, контейнеры и т.п.

2) Уровень разработчика. В основном используется для написания высокоуровневых библиотек OpenFOAM. Включает в себя: работу с математическими примитивами (скаляр, вектор, тензор), расчётную сетку, поля величин, классы для дискретизации пространственных операторов, для создания матрицы СЛАУ, методы решения СЛАУ и т.п.

3) Уровень пользователя. Под «пользователем» здесь понимается программист, создающий собственный решатель без детального изучения/модификации библиотек OpenFOAM путём использования объектов, определённых на предыдущем уровне абстракции (в частности, сетки, полей величин и операций с ними). В целом, все исход-

ные коды решателей, располагающиеся в каталоге `$WM_PROJECT_DIR/applications/solvers` можно отнести к этому наиболее высокому уровню абстракции; именно на него и делается в основном акцент в данном пособии.

Для лучшего понимания представленной иерархии, можно выделить следующие классы, расположенные в порядке возрастания уровня абстракции:

- классы математических примитивов: **scalar**, **vector**, **tensor**;
- классы для описания сетки и физического времени: **fvMesh**, **polyMesh**, **time**;
- класс для граничных условий: **fvPatchField** и наследуемые от него;
- классы для описания полей на сетке: **Field**, **DimensionedField**, **GeometricField**;
- классы (шаблоны классов) для описания матрицы СЛАУ: **fvMatrix**, **lduMatrix**;
- функции для дискретизации дифференциальных операторов по МКО из пространства имён **fvm**, **fvc**.

Классы математических примитивов определяют данные и методы для различных алгебраических операций со скалярами, векторами и тензорами; подробнее о них будет рассказано в следующем разделе. Класс расчётной сетки содержит всю необходимую информацию о ячейках, границах и геометрических характеристиках сетки. Соединив, условно говоря, класс **scalar** с классом **field**, получим класс описания поля скалярной величины **scalarField**. Объединив этот класс с классом сетки **fvMesh** и классом граничных условий **fvPatchField**, получим класс описания поля скалярной величины на расчётной сетке **volScalarField**. Применив какую-либо операцию из пространства имён **fvm** к этому полю, получим объект класса из шаблона **fvMatrix**, содержащий СЛАУ. При этом множество всех необходимых операций, которые надо выполнить при использовании операции из **fvm**, «скрыто» от программиста и даёт возможность в принципе не задумываться о том, как именно реализована та или иная операция.

Стоит отметить, что в пакете OpenFOAM интенсивно используются шаблоны C++, метапрограммирование и пр. Это позволяет «на лету» создавать множество классов, реализующих сходный алгоритм, но работающих с объектами разного типа. Как следствие такого подхода, неискушённому программисту зачастую достаточно сложно разобраться в коде и добраться до сути какого-либо метода, т.е. непосредственно до алгоритма. Кроме того, при неправильном использовании каких-либо методов в разрабатываемой программе, получаемые при её компиляции сообщения об ошибках могут быть весьма «мудрёными». Таким образом, программисту, желающему «залезть в потроха» OpenFOAM, следует быть готовым к кропотливой работе с исходниками, которые зачастую бывают недостаточно хорошо документированы, что является одним из недостатков OpenFOAM.

6.1.2. Математические примитивы

Для создания объекта типа скаляр, тензор или вектор в OpenFOAM используется класс с соответствующим названием. Например, создание тензора T в программе выглядит следующим образом:

```
tensor T(1, 2, 3, 4, 5, 6, 7, 8, 9);
```

В скобках последовательно перечисляются компоненты тензора, так что созданный в приведённой строке объект класса «тензор» соответствует следующему математическому определению T:

$$T = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}.$$

Для доступа к каждой компоненте тензора можно использовать следующие методы:

```
T.xx(); T.xy(); T.xz(); ...
```

Таким образом, для вывода yz-компоненты тензора можно воспользоваться следующим кодом:

```
Info << "Tyz = " << T.yz() << endl;
```

Здесь используется стандартный потоковый вывод C++ в объект Info, который «транслирует» вывод на стандартный поток вывода (т.е. по умолчанию в консоль); endl является сокращением для перевода на новую строку и очистки буфера (т.е. незамедлительный вывод строки) из стандартной библиотеки C++. Как результат, данный код выведет следующую строку:

```
Tyz = 6
```

Для создания скалярного и векторного объектов используются соответствующие команды:

```
scalar s(0.1); vector V(1,2,3);
```

Доступ к компонентам вектора осуществляется следующими методами:

```
V.x(); V.y(); V.z();
```

Для математических примитивов определены различные операции, некоторые из них приведены в таблице 6.1.

Таблица 6.1. Некоторые операции с математическими примитивами в OpenFOAM

Операция	Описание
`=`	Покомпонентное присвоение
`+, -, *, /`	Покомпонентные арифметические операции
`&`	Скалярное произведение
`&&`	Двойное скалярное произведение
`^`	Векторное произведение

Операция	Описание
`mag()`	Модуль
`T.T()`	Транспонирование тензора
`tr(T)`	След тензора
`symm(T)`	Симметричная часть тензора
`dev(T)`	Антисимметричная часть тензора

Некоторые примеры работы с тензорами и векторами можно найти в тестовых программах, находящихся в каталогах `$WM_PROJECT_DIR/applications/test/vector` и `$WM_PROJECT_DIR/applications/test/tensor`. Исходные коды определения классов различных примитивов можно найти в каталоге `$WM_PROJECT_DIR/src/OpenFOAM/primitives`.

6.1.3. Работа с сеткой и полями физических величин

Расчётная сетка в OpenFOAM является объектом класса **fvMesh**; это один из основных объектов в методе конечных объёмов и, соответственно, пакете OpenFOAM: практически во всех решателях OpenFOAM она необходима.

Код для создания объекта сетки находится в файле `$WM_PROJECT_DIR/src/OpenFOAM/include/createMesh.H` и выглядит следующим образом:

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

Здесь **Foam** — так называемое пространство имён (namespace), которое является основным в OpenFOAM и содержит большинство используемых в библиотеках классов; класс **fvMesh** также принадлежит этому пространству имён. Создание объекта **mesh** происходит путём вызова конструктора класса **IOobject**, который предназначен для работы с файлами, в частности, чтения и записи полей, словарей и др. (чуть подробнее об этом см. далее). В данном случае вызов конструктора означает, что сетка должна быть считана из соответствующих файлов (её формат хранения описан в разделе 2.4.1).

С точки зрения пользователя-программиста (или в дальнейшем просто пользователя, под которым здесь понимается программист, работающий на уровне пользователя, см. 6.1.1) наиболее важным при работе с сеткой является получения тех или иных её геометрических параметров (объёмов ячеек, площадей граней и т.п.). Все эти параметры являются в свою очередь также полевыми величинами и имеют соответствующий тип. Таким образом, с т.з. OpenFOAM и физические поля (типа скорости, давления и т.п.), и поля параметров сетки имеют один и тот же класс.

Классы полевых величин, определённых на сетке, имеют следующие названия:

- **volScalarField**, **volVectorField**, **volTensorField** — классы для определённой на данной сетке скалярной, векторной и тензорной полевой величины, соответственно; в объектах этих классов содержатся значения величины в центрах ячеек и на границах;
- **surfaceScalarField**, **surfaceVectorField**, **surfaceTensorField** — аналогичные классы, но предназначенные для величин, определённых в центрах граней.

Таким образом, например, объёмы ячеек сетки условно «принадлежат» центрам ячеек и поэтому являются объектом типа **volScalarField**, а векторы граней, естественно, являются объектом типа **surfaceVectorField**. В таблице 6.2 представлены методы класса **fvMesh**, предназначенные для получения некоторых параметров расчётной сетки, с указанием типа возвращаемого объекта. Пример работы с сеткой и её геометрическими характеристиками можно найти в файле `$WM_PROJECT_DIR/applications/test/mesh`.

Таблица 6.2. Некоторые методы класса **fvMesh**

Метод	Тип возвращаемого объекта	Описание
`V()`	volScalarField	Объём ячейки
`C()`	volVectorField	Радиус-вектор центра объёма
`Sf()`	surfaceVectorField	Вектор грани
`magSf()`	surfaceScalarField	Площадь грани
`Cf()`	surfaceVectorField	Радиус-вектор центра грани

Как было отмечено, поля физических величин, определённых на сетке, также имеют один из перечисленных выше типов. Для создания поля какой-либо величины в OpenFOAM может использоваться следующий код:

```
volScalarField p
(
    IOobject
    (
```

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

В приведённом примере создаётся объект p типа **volScalarField** на сетке «mesh» путём чтения его из файла с именем «p». Чтение из файла здесь также реализовано путём вызова конструктора класса **IOobject**, однако, уже отличного от конструктора для чтения сетки. Рассмотрим несколько подробней класс **IOobject**.

При вызове конструктора **IOobject** здесь передаются пять параметров. Первый — это имя файла, из которого будут считываться данные. Второй — это имя, представляющее собой временную метку для текущего момента времени. Текущий момент времени обычно определяется заданием в файле `system/controlDict` параметра startTime; для расчёта с нулевого момента времени это обычно «0». Таким образом, данные для создаваемого объекта будут считываться, например, из файла `0/p`.

Третий параметр — это объект сетки (mesh), на которой определён создаваемый объект «p». Если файл «p» не соответствует сетке mesh, то программа выдаст сообщение о ошибке и остановится. Четвёртый и пятый параметры отвечают за чтение и запись поля. Константа `IOobject::MUST_READ` означает, что данные обязательно должны быть считаны из файла (в случае его отсутствия программа выдаст ошибку); другими возможными значениями здесь являются: `IOobject::READ_IF_PRESENT` — чтение из файла, если он присутствует (иначе инициализация значением, передаваемым в качестве аргумента конструктора класса **volScalarField**), `IOobject::NO_READ` — не читать из файла, даже если он присутствует.

Константа `IOobject::AUTO_WRITE` означает, что будет производится вывод в файл с указанным именем в те моменты времени, в которые происходит вывод всех полей. Другая возможная константа — `IOobject::NO_WRITE` — означает, что автоматически поле не выводится в файл (если в програм-

ме требуется вывести поле, это можно сделать явным образом, вызвав соответствующий метод).

Помимо использования указанной конструкции, поля в OpenFOAM можно создавать и другими способами. Например, следующий код создаст скалярное поле «x» со значениями, соответствующими x-компоненте радиус вектора центра ячейки:

```
volScalarField x( word("coord_x"), mesh.C().component(0) );
```

Здесь конструкция `word("coord_x")` предназначена для «именования» созданной переменной (например, так будет называться выводной файл для неё). Можно опустить указание имени (тогда переменная будет названа автоматически на основе некоторого правила):

```
volScalarField x( mesh.C().component(0) );
```

Обратите внимание, что здесь для доступа к компонентам вектора используется метод `component()`, поскольку каждая компонента векторного поля `mesh.C()` — это также полевая величина. Для доступа к x-компоненте векторного поля `V` можно также использовать конструкцию вида:

```
V.component(vector::X)
```

Поля величин на гранях ячейки расчётной сетки могут быть получены путём интерполяции из центров ячеек в центр грани следующим образом:

```
surfaceVectorField V_face( fvc::interpolate(Velocity) );
```

Здесь создаётся векторное поле типа **surfaceVectorField** в результате интерполяции скорости `Velocity` на грань; схема интерполяции выбирается при запуске решателя исходя из заданной пользователем в секции interpolationSchemes файла `system/fvSchemes` (см. раздел 3.1.3).

Поля указанных выше типов являются как раз теми объектами, для которые и записываются уравнения в OpenFOAM. Над полями можно совершать все действия, определённые для математических примитивов (алгебраические операции и др.), а также проводить вычисления различных дифференциальных операторов.

Вернёмся теперь к описанию важного класса OpenFOAM **dimensionSet**. Как ранее неоднократно упоминалось, практически все величины, пред-

ставляющие собой физические поля или свойства, должны иметь определённые размерности. Например, в файле начальных и граничных условий для любого поля обязательно присутствует строка вида (например, см. файл $WM_PROJECT_DIR/tutorials/basic/laplacianFoam/flange/0/T, описанный в разделе 2.3.2):

```
dimensions      [0 0 0 1 0 0 0];
```

Данная строка задаёт *размерность* поля. Каждое число в квадратных скобках является степенью соответствующей по порядку единицы измерения из списка: масса (кг), длина (м), время (с), температура (К), количество вещества (моль), сила тока (А), интенсивность освещения (Кд), т.е. [kg m s K mol A Kd]; в данном примере размерность величины — Кельвины[1].

При работе с размерными величинами OpenFOAM проверяет размерность каждого выражения, и если они не соответствуют друг другу (например, килограммы складываются с секундами), выдаёт ошибку. Можно для конкретной задачи отключить проверку на размерность (что может быть полезно при отладке), указав в файле controlDict в секции DebugSwitches[2] значение параметра dimensionSet 0;.

Отметим, что указанные в разделе 6.1.2 классы математических примитивов являются безразмерными. Их «размерные версии» определяются классами **dimensionedScalar**, **dimensionedVector** и пр. Например, орт оси x, имеющий размерность «метр», может быть определён как

```
dimensionedVector x-ort ( "x-ort", dimLength, vector(1,0,0) );
```

Здесь создаётся именованный размерный вектор x-ort, где в качестве единицы измерения указана константа dimLength типа **dimensionSet**. Эти константы определяются в файле $WM_PROJECT_DIR/src/OpenFOAM/dimensionSet/dimensionSets.C и приведены в листинге.

```
const dimensionSet dimless(0, 0, 0, 0, 0, 0, 0);
const dimensionSet dimMass(1, 0, 0, 0, 0, 0, 0);
const dimensionSet dimLength(0, 1, 0, 0, 0, 0, 0);
```

[1] По умолчанию в OpenFOAM используются единицы СИ, но есть и др. варианты

[2] Глобальные значения параметров OpenFOAM, в т.ч. и указанные в секции DebugSwitches, находятся в файле $WM_PROJECT_DIR/etc/controlDict

```
const dimensionSet dimTime(0, 0, 1, 0, 0, 0, 0);
const dimensionSet dimTemperature(0, 0, 0, 1, 0, 0, 0);
const dimensionSet dimMoles(0, 0, 0, 0, 1, 0, 0);
const dimensionSet dimCurrent(0, 0, 0, 0, 0, 1, 0);
const dimensionSet dimLuminousIntensity(0, 0, 0, 0, 0, 0, 1);
```

Любую составную размерность можно определить путём алгебраических операций с этими константами. Например, задать размерность для коэффициента динамической вязкости кг/(м·с) можно как

```
dimMass/(dimLength*dimTime)
```

При объявлении в коде какой-либо размерной величины в случае, если она не считывается из файла (в котором должна быть указана размерность) или не создана на основе других размерных величин, её значение необходимо инициализировать размерной величиной соответствующего типа. Например, при объявлении в коде вспомогательной векторной величины, имеющей размерность скорости и равной везде нулю, можно воспользоваться следующей конструкцией:

```
volVectorField auxVel
(
    IOobject
    (
        "auxVel",
        runTime.timeName(),
        mesh,
        IOobject::NO_READ,
        IOobject::NO_WRITE
    ),
    mesh,
    dimensionedVector("zero", dimLength/dimTime, Zero)
);
```

Отметим ещё один момент при работе с полями. Поскольку они привязаны к сетке, а сетка имеет как понятие ячеек и внутренних граней, так и понятие граничных зон (и, соответственно, их граней), то все поля типа **volScalarField**, **volVectorField** и т.п. на самом деле состоят из двух «полей»: значений в центрах ячеек, доступ к которым осуществляется при помощи метода `internalField()`, и значений на границе, доступных при

помощи метода `boundaryField()`. В некоторых случаях бывает необходимо раздельно обрабатывать внутренние и граничные значения.

6.1.4. Дискретизация пространственных операторов по явной и неявной схемам. Составление уравнений

К полевым величинам в OpenFOAM можно применять различные операции (div, grad и т.п.). При этом метод аппроксимации этих операций задаётся при постановке задачи в файле `system/fvSchemes`, как описано в разделе 3.1.

Если в программе требуется просто применить пространственный оператор к какой-либо величине, для этого используются функции из пространства имён **fvc** (от Finite Volume Calculus — вычисление по методу конечных объёмов), которые производят вычисления над заданным полем и возвращают в качестве результата поле соответствующего типа. Например, если задана полевая величина `scalar` типа **volScalarField**, то результат вычисления градиента от неё при использовании функции **grad()** из **fvc** вернёт поле типа **volVectorField**, как показано в примере:

```
volVectorField grad_scalar = fvc::grad(scalar);
```

Дискретизация пространственного оператора `fvc::grad()` выполняется в соответствии с заданием схемы в файле `fvSchemes`, и за редким исключением[1] используется метод конечных объёмов; именно поэтому **fvc** имеет такую аббревиатуру[2]. Таким образом, функции из пространства имён **fvc** можно применять к известным полевым величинам для получения других полевых величин.

Однако, основная задача использования OpenFOAM — численное решение какого-либо уравнения или их совокупности. То есть, необходимо путём дискретизации уравнения получить СЛАУ для определения неизвестных величин. Для этого в OpenFOAM служит пространство имён **fvm** (Finite Volume Matrix). Функции из **fvm** возвращают объект типа **fvScalarMatrix**, **fvVectorMatrix** и т.п., который содержит матрицу

[1] Для вычисления градиента может использоваться как МКО, так и другой метод, см. раздел 3.1.4

[2] Следует сделать оговорку: МКО можно понимать и в более широком смысле, чем в данном пособии, подразумевая под основным «объектом» в МКО именно ячейку, имеющую некоторую форму, объём и поверхность

СЛАУ, полученную в результате дискретизации указанного оператора, а также правую часть и ссылку на поле для решаемой переменной. Например, следующий код

```
fvScalarMatrix TEqn( fvm::laplacian(DT, T) );
```

создаёт СЛАУ (переменную TEqn) для стационарного уравнения теплопроводности с нулевой правой частью, где в качестве искомой величины выступает поле T, а DT является заданным коэффициентом температуропроводности (может быть скаляром или скалярным полем):

$$\nabla \cdot (D_T \nabla T) = 0.$$

Для решение созданной СЛАУ используется функция solve():

```
solve( Teqn );
```

Эта функция считывает из файла system/fvSolution информацию о методе решения СЛАУ для данной переменной (см. раздел 3.1.6) и вызывает соответствующий линейный решатель; результат его работы (решение СЛАУ) записывается в переменную T (исходное поле, или иначе, предыдущее, может быть получено путём вызова метода prevIter() объекта T, что необходимо, в частности, для применения релаксации).

Таким образом, можно условно считать **fvm** как *неявную* аппроксимацию слагаемых в уравнении, а **fvc** — *явную*. То есть решение уравнения для скорости из, например, решателя icoFoam может быть в принципе записано в виде (ср. с $WM_PROJECT_DIR/applications/solvers/incompressible/icoFoam/icoFoam.C):

```
solve (
    fvm::ddt(U) + fvm::div(phi, U) - fvm::laplacian(nu, U)
    == - fvc::grad(p)
);
```

Здесь производная по времени, конвективное и диффузионное слагаемое для U аппроксимируются неявным образом (создаётся СЛАУ), а правая часть с градиентом давления — явным. Оператор-функция «==» здесь служит для явного разбиения левой и правой части уравнения, но ничто не мешает перенести градиент давления в левую часть (с изменением знака и

откидыванием оператора «==»), или часть слагаемых из левой в правую, и это также будет корректная запись с т.з. OpenFOAM. Понятное ограничение при составлении СЛАУ — переменная, относительно которой будет решаться СЛАУ, должна быть одинакова во всех слагаемых. Также, не все пространственные операторы из **fvc** имеют «аналог» из **fvm**; это касается, в частности, оператора градиента.

Список некоторых пространственных операторов, производных по времени и их запись в OpenFOAM приведён в таблице 6.3 (в последнем столбце указывается, в каких пространствах имён определены данные операции).

Особо стоит отметить операцию дивергенции (div), математически записываемую как $\nabla \cdot (\vec{U}\psi)$, где ψ — единица или скалярное/векторное/тензорное поле. При записи оператора дивергенции в OpenFOAM в качестве первого аргумента должна выступать величина φ типа `surfaceScalarField`, представляющая собой скалярное произведение вектора $\vec{U}$ на грани на вектор этой грани $\vec{S}f$, т.е. $\varphi = \vec{U}_f \cdot \vec{S}_f$, т.е. расход на грани (объёмный или массовый в случае $\vec{U} = \rho\vec{V}$). Вычислять φ можно следующим образом (см. `$WM_PROJECT_DIR/src/finiteVolume/cfdTools/incompressible/createPhi.H`):

```
surfaceScalarField phi ( fvc::flux(U) )
```

Вызов `fvc::flux(U)` в некоторой степени эквивалентен следующей операции:

```
surfaceScalarField phi ( fvc::interpolate(U)&mesh.Sf() )
```

т.е. интерполяции `U` на грань и последующее скалярное произведение с вектором грани.

Таблица 6.3. Операции в OpenFOAM

Операция	Функция в OpenFOAM	Типы аргументов	**fvc**/**fvm**
$\frac{\partial\psi}{\partial t}$, $\frac{\partial}{\partial t}(\rho\psi)$	`ddt(psi)`, `ddt(rho,psi)`	`psi`: `vol<type>Field` `rho`: `volScalarField` или `dimensionedScalar`	fvc, fvm
$\frac{\partial^2\psi}{\partial t^2}$, $\frac{\partial^2}{\partial t^2}(\rho\psi)$	`d2dt2(psi)`, `d2dt2(rho,psi)`	`psi`: `vol<type>Field` `rho`: `volScalarField` или `dimensionedScalar`	fvc, fvm
$\nabla^2\psi$ $\nabla\cdot(\Gamma\nabla\psi)$	`laplacian(psi)` `laplacian(G,psi)`	`psi`: `vol<type>Field` `G`: `vol<type>Field`, `surface<type>Field` или `dimensioned<type>`	fvc, fvm
$\nabla\cdot\varphi$	`div(phi)`	`phi`: `surfaceScalarField`	fvc
$\nabla\cdot(\varphi\psi)$	`div(phi,psi)`	`psi`: `vol<type>Field` `phi`: `surfaceScalarField`	fvc, fvm
$\nabla\psi$	`grad(psi)`	`psi`: `vol<type>Field`	fvc
$\nabla\times\psi$	`curl(psi)`	`psi`: `vol<type>Field`	fvc
$\alpha\psi$	`Sp(alpha,psi)` `SuSp(alpha,psi)`	`psi`: `vol<type>Field` `alpha`: `volScalarField` или `dimensionedScalar`	fvc, fvm

Представленные в таблице 6.3 операции возвращают переменную типа `vol<type>Field`, где `<type>` является одним из `Scalar`, `Vector`, `Tensor` и выбирается исходя из типа аргумента `psi` и операции. Например, если вычисляется градиент скалярной величины (т.е. `psi` имеет тип `volScalarField`), то `fvc::grad(psi)` вернёт переменную типа `volVectorField`.

Отметим также операцию `fvm::SuSp(alpha,psi)` — произведение `alpha` на `psi`. В отличие от `fvm::Sp(alpha,psi)`, которая записывает `alpha` в диагональный коэффициент матрицы СЛАУ для `psi`, `fvm::SuSp()` записывает в матрицу только положительные `alpha`, а отрицательные идут в правую часть как источниковое слагаемое (т.е. произведение `alpha` на `psi`). Это обеспечивает сохранение диагонального преобладания в матрице и улучшает сходимость численного метода. `fvc::Sp()` и `fvc::SuSp()` по сути эквивалентны.

Более подробно об имеющихся операциях в OpenFOAM, а также о типах аргументах, возвращаемых значениях и др. можно найти в документации Doxygen или непосредственно в исходных кодах `$WM_PROJECT_DIR/src/finiteVolume/finiteVolume`. Некоторые примеры использования функций из **fvc** можно найти в `$WM_PROJECT_DIR/applications/test/fvc`.

6.2. Создание собственного решателя

В данном разделе поэтапно расписан процесс создания собственного решателя. Сразу оговоримся, что обычно новый решатель в OpenFOAM создаётся не «с нуля», а на базе уже имеющихся. Т.е. стратегия создания может быть такой: определяется набор решаемых уравнений; ищется решатель, наиболее подходящий под задачу (или наиболее простой, если подходящего нет); исходные файлы найденного решателя копируются в пользовательский каталог, изменяются под требования задачи; решатель компилируется под новым именем. Поэтому в данном разделе мы также будем исходить из имеющихся решателей и делать на них отсылки.

6.2.1. Постановка задачи

Прежде чем перейти к процессу создания решателя, необходимо поставить задачу. В качестве примера будем рассматривать так называемые *уравнения мелкой воды* (уравнения Сен-Венана, Saint-Venant), см. [39, 40, 41]. Эти уравнения описывают движение жидкости со свободной поверхностью при наличии силы тяжести в предположении, что высота жидкости над поверхностью много меньше характерных продольных размеров рассматриваемой задачи. Таким образом, движением жидкости в вертикальном направлении можно пренебречь, откуда из уравнений Эйлера движения несжимаемой жидкости после их упрощения получается система двумерных нестационарных уравнений следующего вида [40]:

$$\frac{\partial \vec{V}}{\partial t} + \vec{V} \cdot \nabla \vec{V} = -|\vec{g}| \nabla h, \tag{6.1}$$

$$\frac{\partial h}{\partial t} + \nabla \cdot \left(\vec{V} \left(h - h_B \right) \right) = 0. \tag{6.2}$$

Здесь $\vec{g}$ — ускорение свободного падения (направлено вдоль оси z), $\vec{V} = \vec{V}(x, y)$ — некоторая средняя по z скорость жидкости, h_B — заданная высота «неровностей» дна относительно горизонтального уровня (может быть функцией x и y), h — высота столба жидкости относительно того же горизонтального уровня. Отметим, что в представленную систему уравнений мелкой воды можно добавить и другие слагаемые, связанные, например, с вращением (сила Кориолиса; важно для геофизических задач), с трением о дно и др., см. [40, 41].

Будем рассматривать только случай «плоского» дна, т.е. $h_B = 0$, тогда h является глубиной жидкости (искомая функция от x и y). Кроме того, уравнения мелкой воды записаны в неконсервативной форме, для перехода к консервативной форме здесь мы внесём скорость в конвективном слагаемом под оператор «набла» с соответствующими изменениями. В результате, при создании собственного решателя мы будем ориентироваться на следующие уравнения:

$$\frac{\partial \vec{V}}{\partial t} + \nabla \cdot \left(\vec{V}\vec{V}\right) - \vec{V}\left(\nabla \cdot \vec{V}\right) = -|\vec{g}|\nabla h, \tag{6.3}$$

$$\frac{\partial h}{\partial t} + \nabla \cdot \left(\vec{V}h\right) = 0. \tag{6.4}$$

Отметим некоторые свойства данной системы. Во-первых, поскольку $\vec{V}$ — это некоторая средняя скорость, то в этой модели $\nabla \cdot \vec{V}$ не обязательно должно равняться нулю; законом сохранения массы здесь служит уравнение (6.4). Во-вторых, уравнения являются чисто конвективными (бездиссипативными); это означает, что при их решении необходимо контролировать численную диффузию, что является нетривиальной задачей (см. раздел 1.3.1). Данная система уравнений является нелинейной, поэтому необходимо проводить линеаризацию. Здесь мы будем применять наиболее простую итерационную схему в рамках одного шага по времени, а именно: поток на грани $\vec{S}_f \cdot \vec{V}_f$ берётся с предыдущей итерации, затем решаются последовательно уравнения (6.3) (с h со старой итерации) и (6.4), после чего поток на грани обновляется и происходит переход к следующей итерации. Отметим, что для уравнений мелкой воды в OpenFOAM также есть решатель shallowWaterFoam; в нём рассматривается общий случай произвольного направления силы тяжести (а также возможен учёт вращения),

и реализована более сложная численная схема; здесь мы его не рассматриваем.

6.2.2. Компиляция программы

Для начала рассмотрим процесс компиляции программы.

Создадим каталог, где будут находится исходники программы, например, `$FOAM_RUN/src/SaintVenantFoam`. Файл с исходниками, содержащий основную программу `main()`, по традиции, принятой в OpenFOAM, будет иметь название `SaintVenantFoam.C`. Помимо этого (и, возможно, других) файлов, для компиляции при помощи утилиты wmake необходимо иметь подкаталог `Make`. В этом подкаталоге должна содержаться вся информация об опциях компиляции решателя, представленная в двух файлах: `files` и `options`; также сюда пишется подкаталог с результатами работы утилиты wmake, в частности, с объектными файлами. На рисунке 6.1 привёден пример возможной структуры каталогов решателя после компиляции.

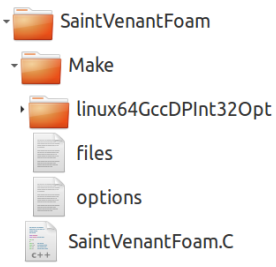

Рис. 6.1. Каталог с исходниками решателя SaintVenantFoam

Файл `Make/files` должен содержать список всех компилируемых файлов и информацию об имени программы и месте её расположения. Ниже приведён пример такого файла для рассматриваемого случая:

```
Файл $FOAM_RUN/src/SaintVenantFoam/Make/files:
SaintVenantFoam.C
EXE = $(FOAM_USER_APPBIN)/SaintVenantFoam
```

Переменная `EXE` содержит путь и имя исполняемой программы. Переменная окружения `FOAM_USER_APPBIN` указывает на каталог, где лежат все исполняемые файлы, создаваемые пользователем; этот каталог отличается от того, где лежат исполняемые файлы из дистрибутива OpenFOAM — они располагаются в каталоге `$FOAM_APPBIN`.

В рассматриваемом примере имеется только один компилируемый файл, что типично для решателей OpenFOAM, в котором принято следующая ор-

ганизация файлов с исходниками: имеется один файл с расширением .C, в котором содержится вызов main(), а некоторые логически отделённые куски программы выносятся в файлы с расширением .H, которые «подключаются» в файл .C при помощи директивы #include. В качестве примера можно привести содержимое каталога $WM_PROJECT_DIR/applications/solvers/basic/laplacianFoam с исходниками уже рассматриваемого в разделе 2.3 решателя laplacianFoam:

```
createFields.H
write.H
laplacianFoam.C
```

Здесь все файлы .H являются не заголовочными файлами, а просто логическими «кусками» программы laplacianFoam.C, вынесенными в отдельные файлы[1].

В файле Make/options содержатся опции для компиляции. Для нашей задачи содержимое этого файла следующее:

```
Файл $FOAM_RUN/src/SaintVenantFoam/Make/options:
EXE_INC = \
    -I$(LIB_SRC)/finiteVolume/lnInclude \
    -I$(LIB_SRC)/meshTools/lnInclude
EXE_LIBS = \
    -lfiniteVolume \
    -lmeshTools
```

В EXE_INC описываются все include-каталоги, т.е. те каталоги, откуда будут браться отсутствующие в текущем каталоге файлы .H. Указанные здесь каталоги содержат все основные требуемые при компиляции заголовочные (и не только) файлы.

В EXE_LIBS перечисляются все требуемые подключаемые библиотеки OpenFOAM при помощи опции -l. Сивол «\» означает перенос строки.

Процесс компиляции программы (и создание исполняемого файла) осуществляется при помощи команды wmake из каталога решателя, т.е.

```
cd $FOAM_RUN/src/SaintVenantFoam
wmake
```

[1] Отметим, что в C/C++ директива #include просто вставляет текст из указанного файла на место этой директивы

Для удаления всех созданных в процессе компиляции вспомогательных файлов служит команда wclean. В качестве дополнения отметим, что для компиляции библиотеки используется команда `wmake libso`.

6.2.3. Функция main()

Основной файл программы `SaintVenantFoam.C` должен содержать функцию main(). В этой функции происходит инициализация решателя, чтение параметров и организация основного цикла (по времени). Стандартный вид части основного файла решателя приведён в листинге.

```
Часть файла $FOAM_RUN/src/SaintVenantFoam/SaintVenantFoam.C:
#include "fvCFD.H"
#include "pimpleControl.H"

int main(int argc, char *argv[])
{
    #include "setRootCase.H"
    #include "createTime.H"
    #include "createMesh.H"
    #include "createFields.H"
```

Здесь мы опустили стандартную «шапку» исходников OpenFOAM, в которой содержится текст лицензии (License), название программы (Application) и её краткое описание (Description). Естественно, это всё можно исключить из текста программы, поскольку является комментарием. Единственный важный момент: лицензия GNU GPL является свободной и «копилефтной», поэтому при распространении исполняемых файлов, использующих библиотеки OpenFOAM, необходимо также и обеспечить доступ к исходным текстам, которые должны иметь ту же лицензию.

Заголовочный файл `fvCFD.H` является необходимым (подключает все требуемые заголовочные файлы для возможности использования функций OpenFOAM). Заголовочный файл `pimpleControl.H` подключает функции для PIMPLE-алгоритма, который здесь используется лишь для того, чтобы на каждом шаге по времени организовать внутренний цикл по итерациям (без него в принципе можно обойтись).

«Кусочек» программы, вставляемый директивой `#include "setRootCase.H"`, где файл `setRootCase.H` берётся из каталога

$WM_PROJECT_DIR/src/OpenFOAM/include, приведён ниже; он предназначен для определения и синтаксического разбора переданных при запуске решателя параметров (аргументов) командной строки.

```
Файл $WM_PROJECT_DIR/src/OpenFOAM/include/setRootCase.H:
Foam::argList args(argc, argv);
if (!args.checkRootCase())
{
    Foam::FatalError.exit();
}
```

В файле createTime.H (расположенном в каталоге #WM_PROJECT_DIR/src/OpenFOAM/include) инициализируется объект runTime, необходимый для всех действий, связанных с продвижением по физическому времени; содержимое файла приведено ниже.

```
Файл $WM_PROJECT_DIR/src/OpenFOAM/include/createTime.H:
Foam::Info<< "Create time\n" << Foam::endl;
Foam::Time runTime(Foam::Time::controlDictName, args);
```

В файле createMesh.H создаётся объект сетки mesh путём чтения её из соответствующих файлов; подробнее о коде, приведённом ниже, рассказывалось в начале раздела 6.1.3.

```
Файл $WM_PROJECT_DIR/src/OpenFOAM/include/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
    )
);
```

6.2.4. Инициализация переменных

Подключаемый в функции main() файл `createFields.H` обычно находится в том же пользовательском каталоге (`$FOAM_RUN/src/SaintVenantFoam/`) и содержит код для инициализации используемых в решателе «глобальных» переменных (полей), а также чтения физических параметров из т.н. «словарей» (о чём подробно будет говориться в следующем разделе). Часть содержимого этого файла для рассматриваемой задачи приведена ниже.

Часть файла $FOAM_RUN/src/SaintVenantFoam/createFields.H:

```
    Info<< "Reading fields h and U\n" << endl;
    volScalarField h
    (
        IOobject
        (
            "h",
            runTime.timeName(),
            mesh,
            IOobject::MUST_READ,
            IOobject::AUTO_WRITE
        ),
        mesh
    );
    volVectorField U
    (
        IOobject
        (
            "U",
            runTime.timeName(),
            mesh,
            IOobject::MUST_READ,
            IOobject::AUTO_WRITE
        ),
        mesh
    );
    surfaceScalarField phi
    (
        IOobject
        (
            "phi",
            runTime.timeName(),
```

```
            mesh,
            IOobject::READ_IF_PRESENT,
            IOobject::AUTO_WRITE
        ),
        fvc::flux(U)
    );
```

В приведённом коде создаются два поля (h и U), которые должны считываться из соответствующих файлов (в каталоге для постановки расчёта), а также инициализируется поле phi, представляющее собой объёмный расход через грани ячеек. Константа IOobject::READ_IF_PRESENT означает, что поле phi будет считано из файла с соответствующим именем, если таковой имеется.

6.2.5. Чтение параметров задачи из управляющих файлов

Для задания физических параметров и/или используемых моделей в OpenFOAM используются так называемые «словари» (dictionary), содержимое которых описывалось ранее вкратце в разделах 2.3.3, 3.1. В отличие от полей переменных, объекты в OpenFOAM, представляющие собой словари, имеют класс **dictionary** или наследуемый от него класс **IOdictionary** (предназначенный для считывания словаря из файла). Объявление переменной-словаря происходит следующим образом:

```
Часть файла $FOAM_RUN/src/SaintVenantFoam/createFields.H:
    Info<< "Reading gravitationalProperties\n" << endl;
    IOdictionary gravitationalProperties
    (
        IOobject
        (
            "gravitationalProperties",
            runTime.constant(),
            mesh,
            IOobject::MUST_READ_IF_MODIFIED,
            IOobject::NO_WRITE
        )
    );
```

Здесь представлена часть кода из createFields.H для рассматриваемой задачи, который создаёт объект gravitationalProperties типа

IOdictionary путём чтения из файла с соответствующим названием; в рассматриваемой задаче имеется только один параметр — модуль ускорения свободного падения $|\vec{g}|$, поэтому файл и словарь имеют такое название. Конструктор класса **IOobject** здесь имеет такой же вид, как и при инициализации полей, за исключением константы `IOobject::MUST_READ_IF_MODIFIED`, которая означает, что данный файл будет считываться заново в случае, если в процессе расчёта он поменялся. Такая возможность предназначена для возможности менять «на лету» параметры (в т.ч. и численные схемы) в процессе расчёта и управляется опцией runTimeModifiable (принимающего значения «true» или «false»), задаваемой в файле `system/controlDict`. Отметим, что если runTimeModifiable установлена в «true», то изменения файлов `fvSchemes`, `fvSolution` и `controlDict` из каталога `system` также отслеживаются.

После создания объекта словаря, доступ к различным параметрам (определённым в соответствующем файле в случае **IOdictionary**) осуществляется следующим образом:

```
Часть файла $FOAM_RUN/src/SaintVenantFoam/createFields.H:
    Info<< "Reading magnitude of g\n" << endl;
    dimensionedScalar g
    (
        gravitationalProperties.lookup("g")
    );
```

Если требуется использовать какой-либо параметр, имеющий некоторое значение по умолчанию, то используется шаблонный метод `lookupOrDefault<type>()` класса **dictionary**, где тип `type` задаётся в соответствии с типом параметра.

Отметим, что для чтения из словаря также может применяться следующая инструкция:

```
dimensionedScalar g( "g", dimAcceleration, 9.8 );
gravitationalProperties.lookup("g") >> g;
```

Такая инструкция применяется, в частности, для чтения безразмерных параметров (например, типа **label**, т.е. целых чисел).

6.2.6. Задание совокупности решаемых уравнений

Полный текст функции main(), частично приведённый в разделе 6.2.3, представлен ниже.

```
Часть файла $FOAM_RUN/src/SaintVenantFoam/SaintVenantFoam.C:
int main(int argc, char *argv[])
{
    #include "setRootCase.H"
    #include "createTime.H"
    #include "createMesh.H"
    #include "createFields.H"
    pimpleControl pimple(mesh);
    Info<< "\nCalculating ...\n" << endl;
    while (runTime.loop())
    {
        Info<< "Time = " << runTime.timeName() << nl << endl;
        while (pimple.loop())
        {
            solve
            (
                fvm::ddt(U) + fvm::div(phi, U) - fvm::Sp(fvc::div(phi),U)
                == - g*fvc::grad(h)
            );
            solve
            (
                fvm::ddt(h) + fvm::div(phi, h)
            );
            phi = fvc::flux(U);
        }
        runTime.write();
        Info<< "ExecutionTime = " << runTime.elapsedCpuTime() << " s"
            << "  ClockTime = " << runTime.elapsedClockTime() << " s"
            << nl << endl;
    }
    Info<< "End\n" << endl;
    return 0;
}
```

После инициализации полей (в файле `createFields.H`) следует создание объекта pimple класса **pimpleControl**. Как отмечалось ранее, в данном коде этот класс нужен только для организации итерационного цикла на

каждом шаге по времени (с возможностью использовать некоторые другие опции PIMPLE, как описано в разделе 3.1.7).

Строка «`while (runTime.loop())`» нужна для организации цикла по времени (который контролируется параметрами из `system/controlDict`), а строка «`while (pimple.loop())`» — для итерационного цикла. В теле последнего цикла как раз и происходит собственно решений уравнений (6.3), (6.4). Здесь для первого уравнения все слагаемые, аппроксимируемые неявным образом, находятся в левой части уравнения, а источниковое слагаемое «`-g*fvc::grad(h)`» — в правой. После последовательного решения этих двух уравнений идёт обновление потока на грани `phi`.

Метод `runTime.write()` отвечает за вывод полей в файл, который осуществляется в соответствующие моменты времени, как указано в `system/controlDict` (см. раздел 2.3.3). Методы `runTime.elapsedCpuTime()` и `runTime.elapsedClockTime()` возвращают затраченное процессорное время и время, пройденное с момента запуска программы, соответственно.

6.3. Пример расчёта с использованием собственного решателя

6.3.1. Постановка задачи

Рассмотрим пример решения задачи с использованием описанного в предыдущем разделе решателя. В качестве основы для постановки задачи будем использовать пример из дистрибутива OpenFOAM `$WM_PROJECT_DIR/tutorials/incompressible/shallowWaterFoam/squareBump`.

Пусть имеется квадратный «бассейн» (100×100 м), заполненный жидкостью, вблизи из одного углов которого возникает возмущение за счёт «обрушения» круглого столба жидкости. Схема постановки задачи представлена на рисунке 6.2.

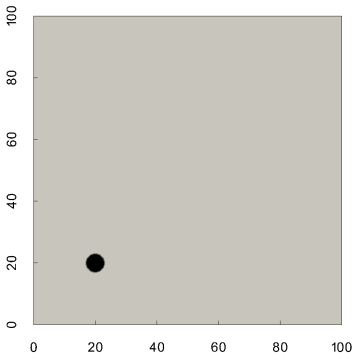

Рис. 6.2. Геометрия (в метрах) и начальное распределение h

В начальный момент времени уровень жидкости в бассейне равен $h = 1$ м, столб жидкости высотой $h = 1.5$ м от дна находится в точке $(20; 20)$ и имеет диаметр 6 м. Ускорение силы тяжести по модулю равно 9.81 м/с^2.

Для построения равномерной ортогональной расчётной сетки используется утилита blockMesh. Часть файла blockMeshDict, предназначенного для создания сетки размером 400×400 ячеек, представлена ниже.

Часть файла $FOAM_RUN/squareBump/system/blockMeshDict:

```
convertToMeters 100;
vertices
(
    (0 0 0)
    (1 0 0)
    (1 1 0)
    (0 1 0)
    (0 0 0.01)
    (1 0 0.01)
    (1 1 0.01)
    (0 1 0.01)
);
blocks (
    hex (0 1 2 3 4 5 6 7) (400 400 1) simpleGrading (1 1 1)
);
boundary (
    sides {
        type patch;
        faces
        (
            (3 7 6 2) (1 5 4 0)
            (0 4 7 3) (2 6 5 1)
        );
    }
    frontAndBack {
        type empty;
        faces
        (
            (0 3 2 1)
            (4 5 6 7)
        );
    }
);
```

Здесь выделены две граничные зоны: зона «frontAndBack» представляет поверхности в z-направлении, на которых ставится условие «empty», зона «sides» содержит четыре «края» квадратного бассейна.

На «краях» бассейна задаётся условие проскальзывания для скорости «slip» и условие нулевой производной по нормали для h (zeroGradient). Соответствующие граничные условия представлены в листинге.

```
Часть файла $FOAM_RUN/squareBump/0/U:
dimensions      [0 1 -1 0 0 0];
internalField   uniform (0 0 0);
boundaryField
{
    sides
    {
        type            slip;
    }
    frontAndBack
    {
        type            empty;
    }
}

Часть файла $FOAM_RUN/squareBump/0/h:
dimensions      [0 1 0 0 0 0 0];
internalField   uniform 0;
boundaryField
{
    sides
    {
        type            zeroGradient;
    }
    frontAndBack
    {
        type            empty;
    }
}
```

Чтобы задать описанное выше начальное условие для h, воспользуемся утилитой setFields; соответствующий файл `setFieldsDict` представлен ниже.

```
Часть файла $FOAM_RUN/squareBump/system/setFieldsDict:
defaultFieldValues ( volScalarFieldValue h 1 );
regions
(
    cylinderToCell
    {
        p1      (20 20 0);
        p2      (20 20 1);
        radius          3;
        fieldValues
        (
              volScalarFieldValue h 1.5
        );
    }
);
```

Файл с параметрами задачи (величиной модуля ускорения свободного падения) имеет следующий вид:

```
Файл $FOAM_RUN/squareBump/constant/gravitationalProperties:
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      gravitationalProperties;
}
g           g [0 1 -2 0 0 0 0]  9.81;
```

Параметры линейных решателей СЛАУ для соответствующих переменных ($\vec{V}$ и h), задаваемые в файле system/fvSolution, могут быть заданы так, как представлено в листинге.

```
Часть файла $FOAM_RUN/squareBump/system/fvSolution:
solvers
{
    h
    {
        solver          PBiCG;
        preconditioner  DILU;
        tolerance       1e-10;
        relTol          0.01;
```

```
    }
    hFinal
    {
        $h;
        tolerance       1e-10;
        relTol          0;
    }
    U
    {
        solver          smoothSolver;
        smoother        GaussSeidel;
        tolerance       1e-10;
        relTol          0.1;
    }
    UFinal
    {
        $U;
        tolerance       1e-10;
        relTol          0;
    }
}
PIMPLE
{
    nOuterCorrectors 10;
    residualControl {
        "(U|h)"
        {
            relTol          0.1;
            tolerance       1E-5;
        }
    }
}
```

Отметим, что поскольку в решателе SaintVenantFoam для организации итерационного цикла используется класс для алгоритма PIMPLE, то в файле имеется соответствующая секция с заданным значением параметра nOuterCorrectors, который определяет максимальное число итераций; для выхода из итерационного цикла также используются опция «residualControl», поэтому выход из итерационного цикла может быть осуществлён ранее, если достигнута соответствующая точность по невязкам

уравнений. Кроме того, при организации итераций при помощи PIMPLE последняя (final) итерация происходит с теми параметрами линейного решателя СЛАУ, которые указаны в «hFinal» и «UFinal». Подробнее о параметрах, задаваемых в файле system/fvSolution, см. раздел 3.1.6.

При решении рассматриваемой задачи использовалось несколько различных численных схем для некоторых слагаемых в уравнениях. В исходном варианте использовались следующие численные схемы: трёхслойная схема «разностью назад» для временной производной, линейная интерполяция при вычислении дивергенции и метод Грина-Гаусса для вычисления градиента. Соответствующий файл system/fvSchemes представлен ниже.

Часть файла $FOAM_RUN/squareBump/system/fvSchemes:

```
ddtSchemes
{
    default         backward;
}
gradSchemes
{
    default         Gauss linear;
}
divSchemes
{
    default         none;
    div(phi,h)      Gauss linear;
    div(phi,U)      Gauss linear;
}
```

6.3.2. Результаты расчёта

В качестве иллюстрации получаемого решения на рисунке 6.3 приведены распределения высоты уровня жидкости h в разные моменты времени, полученные на сетке размером 400×400 с шагом по времени $\Delta t = 0.02$ с. Здесь «тёплая» цветовая гамма (красный цвет) означает горбы волн, а «холодная» (синий цвет) — впадины. Момент времени для каждого распределения указан в верхней части соответствующего рисунка (5, 10, 20 и 40 секунд).

Отметим следующие особенности представленного решения. Во-первых, на распределении h, соответствующем 5 с, отчётливо видна сеточная чув-

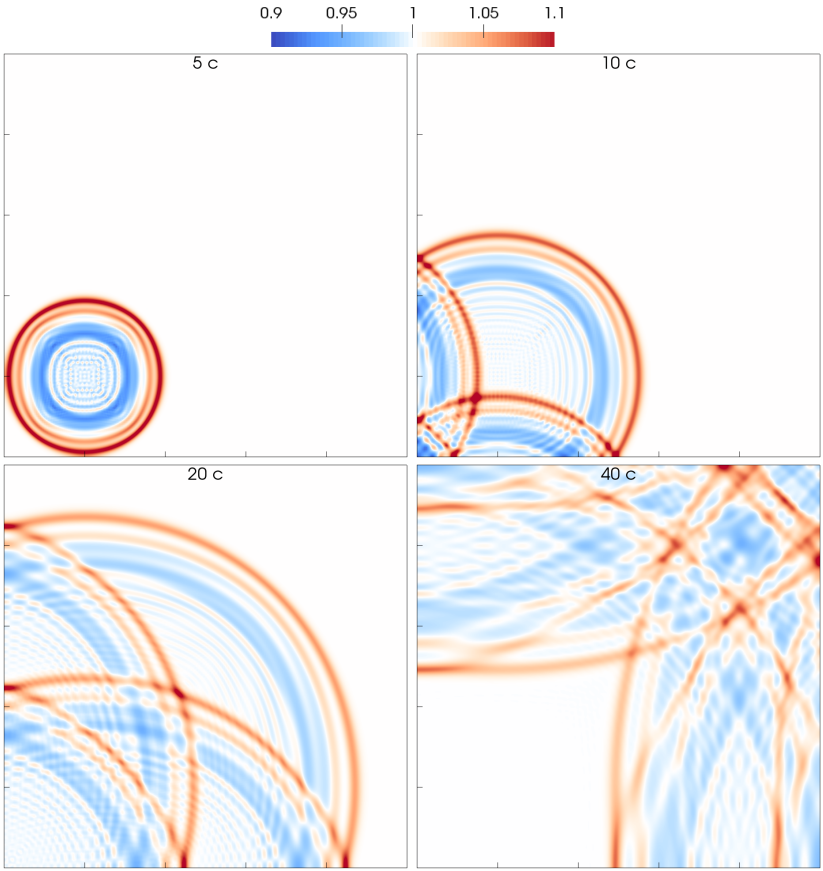

Рис. 6.3. Иллюстрация течения (распределение h в разные моменты времени)

ствительность решения: если фронт первых волн близок по форме к окружности, то форма последующих волн из-за неизбежных погрешностей, вносимых сеткой, трансформируется и становится похожа на квадрат. Также, при разрушении столба жидкости последние этапы этого процесса уже не разрешаются на данной сетке, что приводит к появлению «ряби» в центре столба, идущей также по сеточным линиям; очевидно, что данная «рябь» образуется вследствие численных погрешностей, а следовательно,

вряд ли претендует на «правильное» решение. Поскольку амплитуда её относительно мала, можно в некоторым смысле ею пренебречь, тем более, как следует из дальнейших распределений, она слабо влияет на решение.

Распределение h в момент времени 10 с характеризуется наличием отражённых от стенок «бассейна» волн, в результате взаимодействия которых с падающими волнами образуется интерференционная картина. После «двойного» отражения (от левой и нижней стенок в углу) возникает волновая картина (см. распределение при $t = 20$ с), которая сохраняется до тех пор, пока головная волна не дойдёт до верхней и правой стенок «бассейна», после чего возникает сложная картина взаимодействия падающих и отражённых волн, представленная в момент времени 40 с; отметим, что здесь область первоначального возмущения (левый нижний угол) практически не содержит волн (все они, отразившись от стенок, «ушли» в правый верхний угол).

Рассмотрим теперь влияние схемы дискретизации конвективных слагаемых на решение. На рисунке 6.4 представлено распределение h вдоль диагонали «бассейна» в момент времени 20 с, полученные в исходном варианте и в варианте с противопоточной схемой первого порядка для конвективных слагаемых. Здесь наблюдается сильное сглаживание колебаний при

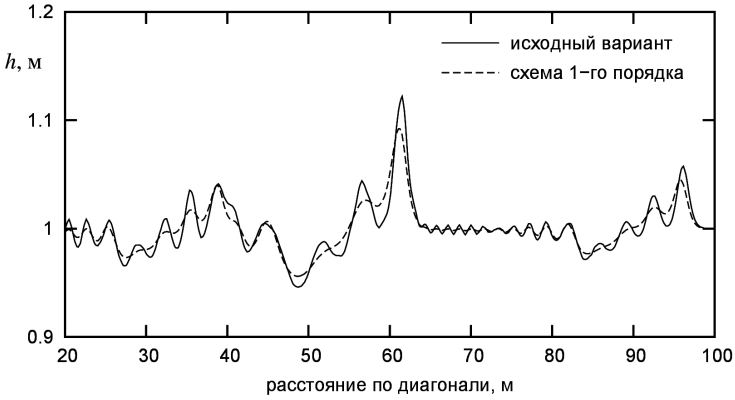

Рис. 6.4. Распределение h в момент времени 20 с по диагонали от нижнего левого угла к верхнему правому

использовании схемы первого порядка, что означает наличие большой численной диффузии.

В заключение необходимо сделать важное замечание насчёт рассматриваемой задачи: обрушение столба сопровождается возникновением в толще жидкости вертикального движения, которое в рамках рассматриваемой модели не учитывается, поэтому строго говоря данное решение не может полностью описать процессы генерации волн, соответствующие реальным. Тем не менее, распространение головной волны вполне описывается в рамках рассматриваемой модели. Кроме того, поскольку в начальный момент времени распределение h имеет разрыв, то наименьшая длина волны генерирующихся колебаний зависит от размера ячейки сетки, и при измельчении сетки возникают всё более короткие волны. Поэтому в данной задаче не удаётся достичь сеточной сходимости, хотя общая картина течения сохраняется. Отметим также, что хотя возникающие коротковолновые колебания являются решением уравнений (6.3), (6.4), сами эти уравнения выведены в предположении, что длина волны много больше глубины. Поэтому решения с коротковолновыми колебаниями строго говоря не отражают реальную картину генерации волн на поверхности воды, так что в данной задаче может идти речь только о качественной картине распространения волн.

6.4. Дополнительные сведения

6.4.1. Отладка программы

В случае, если написанная программа вылетает с ошибкой или выдаёт странные результаты, необходимо провести её отладку. Для этого в OpenFOAM можно использовать различные средства. Наиболее простым и быстрым способом является вставка в код программы инструкций, выводящих различную информацию в терминал, при помощи объекта Info, например:

```
Info << "My variable min: " << min(myVar).value() << endl;
```

Если же программа вылетает не из-за какой-либо инструкции в коде решателя, а в результате выполнения кода, скрытого где-то в «недрах»

библиотек OpenFOAM, то в этом случае добраться до ошибки существенно сложнее. Одним из способов решения проблемы является компиляция OpenFOAM в режиме отладки. Для этого необходимо в файле глобальных настроек OpenFOAM $WM_PROJECT_DIR/etc/bashrc указать:

```
export WM_COMPILE_OPTION=Debug
```

и перекомпилировать OpenFOAM (по умолчанию значение этой переменной равно Opt, т.е. компиляция в режиме оптимизации скорости выполнения программ). После этого можно запускать решатели OpenFOAM в режиме отладки при помощи свободно распространяемой программы-отладчика gdb, подробнее см. https://www.gnu.org/software/gdb.

Перекомпиляция OpenFOAM может занимать значительно время, поэтому если известно, в какой именно библиотеки возникает ошибка, можно скомпилировать только эту библиотеку в режиме отладки. Для этого необходимо в соответствующем каталоге с исходниками этой библиотеки добавить в файл Make/options к переменной окружения EXE_INC следующие параметры: -DFULLDEBUG -g -O0, после чего перекомпилировать библиотеку командой wmake libso.

OpenFOAM также предоставляет возможность выводить некоторую отладочную информацию (во время работы решателя) без необходимости перекомпиляции. Для этого служат специальные переменные, список которых приведён в файле $WM_PROJECT_DIR/etc/controlDict в секции DebugSwitches; ниже приводится часть этой секции:

```
DebugSwitches
{
    Analytical          0;
    APIdiffCoefFunc     0;
        ...
    zoneToFace          0;
    zoneToPoint         0;
}
```

Если установить для переменной ненулевое значение, то в коде, связанном с этой переменной, активируется вывод некоторой отладочной информации. Для задания значения какой-либо переменной не обязательно менять файл $WM_PROJECT_DIR/etc/controlDict, можно в каталоге с поста-

новкой задачи добавить в файл system/controlDict секцию DebugSwitchs с необходимыми переменными.

Более подробную информацию о способах отладки в OpenFOAM можно найти по адресу https://openfoamwiki.net/index.php/HowTo_debugging.

6.4.2. Программируемые граничные условия

OpenFOAM при помощи специальных граничных условий даёт возможность вставлять код (на языке C++) прямо в файлы граничных условий без необходимости добавлять/изменять библиотеки OpenFOAM. Такой способ позволяет реализовывать относительно сложные граничные условия, при этом не требуя детального программирования с неизбежным глубоким погружением в код библиотек OpenFOAM, что необходимо при создании новых пользовательских граничных условий (подробнее о создании собственных граничных условий можно найти в [12]).

Для вставки кода в граничное условие необходимо использовать одно из следующих граничных условий: codedFixedValue или codedMixed. Первое предназначено для программирования граничного условия первого рода, а второе — третьего. Рассмотрим использование программируемого граничного условия на примере codedMixed как более общего.

Синтаксис условия codedMixed, взятый из описания в файле $WM_PROJECT_DIR/src/finiteVolume/fields/fvPatchFields/derived/codedMixed/codedMixedFvPatchField.H, приведён ниже. В нём рассматривается программная реализация нестационарного граничного условия для векторной величины в виде задания на границе кусочно-линейного её изменения.

```
<patchName>
{
    type            codedMixed;
    refValue        uniform (0 0 0);
    refGradient     uniform (0 0 0);
    valueFraction   uniform 1;
    name    rampedMixed;   // name of generated BC
    code
    #{
        this->refValue() =
```

```
            vector(1, 0, 0)
            *min(10, 0.1*this->db().time().value());
        this->refGrad() = Zero;
        this->valueFraction() = 1.0;
    #};
}
```

Условие codedMixed применяется к граничной зоне с именем <patchName> (т.е. указанные выше строки следует вставить в соответствующий файл граничных условий). При этом при запуске программы будет произведена компиляция кода, содержащегося после ключевого слова «code» между символами `#{` и `#}`, и новое скомпилированное граничное условие будет иметь имя, указанное после ключевого слова «name» (в примере — rampedMixed). В самом коде необходимо вычислить переменные `this->refValue()` (т.е. значение величины, для которой ставится данное граничное условие, на грани), `this->refGad()` (значение градиента, в примере — нулевое) и `this->valueFraction()` (вес между refValue и refGrad; значение valueFraction равное 1 соответствует условию первого рода).

В заключение отметим, что подобным образом можно добавлять собственный код не только для граничных условий, но и в файл `controlDict` для создания «на лету» новых функциональных объектов, позволяющих производить различные манипуляции с переменными «изнутри» OpenFOAM (см., например, `$WM_PROJECT_DIR/tutorials/basic/potentialFoam/cylinder/system/controlDict`).

Литература

1. Лойцянский Л.Г. МЖГ. – М.: Дрофа, 2003

2. Рябенький В.С. Введение в вычислительную математику. 3-е изд., испр. и доп. – М.: ФИЗМАТЛИТ – 2008. – 286 с.

3. Учебник по МКЭ

4. Кацикаделис Дж. Граничные элементы. Теория и приложения. – Изд. «Ассоциации строительных вузов». – 2007. – 348 с.

5. Handbook of Grid Generation / Ed. by J.F. Thompson, B.K. Soni, N.P. Weatherill. CRC Press. – 1999. – 1136 p.

6. Самарский А.А. Теория разностных схем. М.Наука, 1989, 3-е изд., 616 с.

7. Флетчер К. Вычислительные методы в динамике жидкостей. М.: Мир, 1991. – Т. 2. – 552 с.

8. Ferziger J.M., Peric M. Computational Methods for Fluid Dynamics. Springer-Verlag, Berlin, Germany, 2002, 423 p.

9. Patankar S.V., Spalding D.B. A calculation procedure for heat, mass and momentum transfer in three-dimensional parabolic flows // Int. J. Heat Mass Transfer, 1972, Vol.15, No.10, pp.1787-1806.

10. Rhie C.M., Chow W.L. A numerical study of the turbulent flow past an isolated airfoil with trailing edge separation // AIAA Journal. – 1983. – Vol. 21. – P. 1525–1532.

11. Головачев Ю.П., Колешко С.Б. Численные методы решения уравнений динамики жидкости и газа: Учебное пособие. - Л.: Изд-во ЛПИ, 1990. – 126 с.

12. T. Maric, J. Höpken, K. Mooney. The OpenFOAM Technology Primer. – Sourceflux UG, 2014. – 442 p. – ISBN: 978-3-00-046757-8

13. Moukalled F., Mangani L., Darwish M. The Finite Volume Method in Computational Fluid Dynamics: An Advanced Introduction with OpenFOAM® and Matlab. – Springer. – 2015. – 817 p.

14. OpenFOAM 4.1 User Guide.

15. Y. Saad. Iterative Methods for Sparse Linear Systems. – SIAM. – 2003. – 547 p.

16. Barakos G., Mitsoulis E., Assimacopoulos D. Natural convection flow in a square cavity revisited: laminar and turbulent models with wall functions // International Journal for Numerical Methods in Fluids. – 1994. – Vol. 18. – P.695-719.

17. Гебхарт Б., Джалурия П., Махаджан Р., Саммакия Б. Свободноконвективные течения, тепло- и массообмен / В 2-х книгах: Пер. с англ. М.: Мир, 1991.

18. Трубецков Д.И. Турбулентность и детерминированный хаос // Соросовский образовательный журнал №1, 1998, с. 77-83

19. Гидродинамические неустойчивости и переход к турбулентности. Пер. с англ. / Под ред. Х. Суинни и Дж. Голлаба. – Москва: Мир, 1984. – 344 С.

20. Современные подходы к моделированию турбулентности: учебное пособие / А.В. Гарбарук, М.Х. Стрелец, А.К. Травин, М.Л. Шур – СПб: Изд-во Политехн. ун-та, 2016. – 234 с.

21. Spalart P.R., Allmaras S.R. (1994). A one-equation turbulence model for aerodynamic flows. La Recherche Aerospatiale, 1, 5-21.

22. Launder B. E., Sharma B. I. Application of the energy-dissipation model of turbulence to the calculation of flow near a spinning disc // Letters in heat and mass transfer, V.1, N2, 131-137, 1974.

23. Launder B. E., Spalding D. B. Lectures in mathematical models of turbulence – Academic Press, 1972.

24. Wilcox D. C. Turbulence Modeling for CFD, 2nd edition, DCW Industries, Inc., La Canada CA, 1998.

25. Menter F. R., Kuntz M., Langtry R. (2003). Ten Years of Industrial Experience with the SST Turbulence Model // Turbulence, Heat and Mass Transfer, V.4, ed: K. Hanjalic, Y. Nagano, & M. Tummers, Begell House, Inc., 625 - 632.

26. M. Popovac, K. Hanjalic. Compound wall treatment for RANS computation of complex turbulent flows and heat transfer // Flow Turbulence and Combustion. – 2007. – Vol. 78. – P. 177-202.

27. T. Esch, F. Menter. Heat transfer predictions based on two-equation turbulence models with advanced wall treatment // In: Turbulence, Heat and Mass Transfer 4 (Proc. 4th Int. Symp, 12 - 17 October 2003, Antalya, Turkey). – Begell House, 2003. – 1208 p. – P. 633-640.

28. Зайцев Д. К., Смирнов Е. М. Расчет течения и теплоотдачи в турбулентных потоках с локализованными отрывными областями на основе метода обобщенных пристенных функций / Труды VI Российской национальной конференции по теплообмену (РНКТ-6) (27-31 октября 2014, Москва). [CD-ROM]. - Москва, МЭИ, 2014. - 4с.

29. Spalding D.B. A single formula for the law of the wall. Journal of Applied Mechanics, Trans. ASME, Series E, Vol. 28, pp. 455–458, 1961.

30. Menter F., Esch T. Elements of Industrial Heat Transfer Prediction // 16th Brazilian Congress of Mechanical Engineering (COBEM), Nov. 2001.

31. Шлихтинг Г. Теория пограничного слоя. М. – Наука, 1974 г., 712 с.

32. Идельчик И. Е. Справочник по гидравлическим сопротивлениям / Под ред. М. О. Штейнберга. — 3-е изд., перераб. и доп.— М.: Машиностроение, 1992, 672 с.

33. S. C. C. Bailey, M. Vallikivi, M. Hultmark, A. J. Smits. Estimating the value of von Kármán's constant in turbulent pipe flow // Journal of Fluid Mechanics, vol. 749, pp. 79–98, 2014.

34. Vogel, J.C., Eaton, J.K. Combined Heat Transfer and Fluid Dynamic Measurements Downstream of a Backward-Facing Step // Journal of Heat Transfer. – 1985. – Vol. 107. – P. 922–929.

35. Adams, E.W., Johnston, J.P., Eaton, J.K. Experiments on the structure of turbulent reattaching flow // Stanford University. – 1984. – Report MD-43.

36. Волков К.Н., Емельянов В.Н. Вычислительные технологии в задачах механики жидкости и газа. – М.: ФИЗМАТЛИТ, 2012. – 468 с.

37. A. Vasiliev et al. High Rayleigh number convection in a cubic cell with adiabatic sidewalls // International Journal of Heat and Mass Transfer, vol. 102, pp. 201–212, 2016.

38. Abramov A.G., Ivanov N.G., Smirnov E.M. Numerical study of high-Ra Rayleigh-Benard mercury and water convection in confined enclosures using a hybrid RANS/LES technique / In: Heat transfer in unsteady and transitional flows / Proc. of the Eurotherm Seminar 74, March 23-26, 2003, Eindhoven, the Netherlands, ed. by H.C. de Lange and A.A. van Steenhoven, TUE, 2003. P.33-38.

39. Гиргидов А.Д. Техническая механика жидкости и газа. – Учеб. для вузов. – СПб.: Изд-во СПбГТУ, 1999. – 395 с.

40. Педлоски Дж. Геофизическая гидродинамика в 2-х т. Т. 1: Пер. с англ. – М.:Мир. – 1984 – 398 с.

41. Dingemans M.W. Wave propagation over uneven bottoms. Advanced Series on Ocean Engineering. – World Scientific, Singapore. – 1997. – 700 p.