ИПМ им.М.В.Келдыша РАН • Электронная библиотека

Препринты ИПМ • Препринт № 118 за 2016 г.

ISSN 2071-2898 (Print)
ISSN 2071-2901 (Online)

Меньшов И.С., Никитин В.С.,
Шевердин В.В.

Параллельная трехмерная ЛАД модель на декартовых сетках вложенной структуры

Рекомендуемая форма библиографической ссылки: Меньшов И.С., Никитин В.С., Шевердин В.В. Параллельная трехмерная ЛАД модель на декартовых сетках вложенной структуры // Препринты ИПМ им. М.В.Келдыша. 2016. № 118. 32 с. doi:10.20948/prepr-2016-118
URL: http://library.keldysh.ru/preprint.asp?id=2016-118

Ордена Ленина
ИНСТИТУТ ПРИКЛАДНОЙ МАТЕМАТИКИ
имени М.В.Келдыша
Российской академии наук

И.С. Меньшов, В.С. Никитин, В.В. Шевердин

Параллельная трехмерная ЛАД модель на декартовых сетках вложенной структуры

Москва — 2016

Меньшов И.С., Никитин В.С., Шевердин В.В.

Параллельная трехмерная ЛАД модель на декартовых сетках вложенной структуры

Настоящий препринт посвящен обобщению технологии декартовых локально-адаптивных (ЛАД) сеток вложенной структуры на трехмерный случай и описанию соответствующей библиотеки прикладных программ. Библиотека написана на языке программирования C++ с использованием принципов объектно-ориентированного программирования и адаптирована на параллельные вычисления на многоядерных процессорах с общей памятью с помощью программного интерфейса приложения OpenMP. Библиотека учитывает специфику многопотокового счета трехмерных задач на декартовых сетках. Это позволяет существенно минимизировать загружаемую память компьютера путем отказа от хранения сеточной информации. Данные сетки, такие как координаты узлов, нормали, площади и объемы, не хранятся, а вычисляются при необходимости. Порядок обхода представляется специальным списком, который упрощает параллельную реализацию директивами интерфейса OpenMP. Препринт включает описание древовидной структуры данных на ЛАД сетке, основные положения дискретной модели и основные функции разработанной библиотеки. Приводятся результаты предварительного тестирования и оценка эффективности параллельного счета представленной трехмерной ЛАД методики на решении задачи о развитии объемного взрыва в замкнутом пространстве.

Ключевые слова: локально-адаптивные декартовые сетки, библиотека прикладных программ, параллельные вычисления на общей памяти

Работа выполнена при поддержке Российского научного фонда, номер проекта 14-11-00872.

Igor Stanislavovich Menshov, Viacheslav Sergeevich Nikitin, Victor Victorovich Sheverdin

Parallel Three-Dimensional LAD Model on Cartesian Grids of Nested Structure

This preprint addresses a generalization of the technology of nested-type Locally Adaptive (LAD) Cartesian grids to the three-dimensional case and a description of the corresponding software library. The library is written in C++ programming language using object-oriented programming principles and is adapted for parallel computing on multi-core shared memory processes with the OpenMP application interface. The library takes into account the specificity of multi-threaded Cartesian-grid calculations of 3D problems. This can significantly minimize the usage of computer memory by avoiding the storage of grid information. Data grid as the coordinates of the nodes, normals, areas, and volumes are not stored, and are evaluated as needed. The order of cell traversing is represented by a special list, which simplifies parallel implementation by means of the OpenMP directives. This preprint contains a description of tree-like data structure on the LAD grid, the basic principles of the discrete model, and main functions of the library developed. Preliminary results of testing the library and the estimation of the effectiveness of parallel calculations are given for calculations of the problem of the evolution of an explosion in a closed space.

Key words: locally adaptive Cartesian grid, application software library, shared memory parallel calculations

The work was supported by the Russian Science Foundation, project No 14-11-00872.

Оглавление

Введение

Вопрос динамической адаптации сеток путем дополнительного подразбиения счетных ячеек с целью повышения точности численных решений для задач гидродинамики и газовой динамики интенсивно исследуется уже на протяжении последних двадцати лет [1-6]. Однако с развитием современных компьютеров на основе высокопроизводительных многопроцессорных вычислительных систем подобная сеточная адаптация приобретает дополнительную специфику, которая делает необходимым развитие новых подходов. Это связано с тем, что реализация алгоритмов решения на параллельных многоядерных системах сверхвысокой производительности сопряжена с преодолением ряда трудностей и особенностей организации счета и обмена данными. Например, желание использовать графические многопоточные ускорители приводит к необходимости поиска простых вычислительных алгоритмов и узкой локализации расчетных шаблонов. Повышение точности расчетов требует рассмотрения очень подробных сеток с высоким пространственным разрешением, что в свою очередь сильно увеличивает расчетное время для получения требуемого решения.

Декартовы сетки идеально подходят для алгоритмов, приспособленных для использования многоядерных параллельных вычислительных систем благодаря простоте программной реализации методов расщепления по направлениям, регулярным шаблонам обращения в память, отсутствию необходимости вычисления нормалей и прочих геометрических характеристик расчетных ячеек. Вместе с тем, использование структурированных регулярных декартовых сеток для получения качественного результата зачастую требует расчетные сетки столь большого размера, что выполнение конкретных вычислительных экспериментов даже на современных суперкомпьютерах может занимать практически неприемлемое время.

Чтобы снять это ограничение, можно воспользоваться тем фактом, что математическое моделирование гидро- газодинамических процессов описывается кусочно-гладкими функциями, где особенности с большими градиентами локализованы в узких подобластях области нахождения решения. Применение в этом случае сеток с высоким пространственным разрешением во всей области нерационально. Уменьшение размера всех расчетных ячеек приводит к значительному росту вычислительной сложности задачи.

В этой связи более разумно использовать сетки с возможностью выборочного измельчения сеточных элементов. Признак деления крупной ячейки на несколько малых может быть разным: большой локальный градиент сеточной функции, пограничный слой вблизи обтекаемого тела, граница раздела фаз в многофазной задаче. Так мы приходим к локально-адаптивным (далее ЛАД) сеткам вложенной структуры.

Технология декартовых ЛАД сеток порождает большое количество проблем, связанных с ее программной реализацией. Во-первых, важным с

практической точки зрения моментом становится выбор формата представления данных на ЛАД сетках. Этот формат должен сохранять однородность доступа к памяти, минимизировать используемые для счета ресурсы, при этом обеспечивая возможность написания всего необходимого функционала с помощью достаточно простых алгоритмов.

Описание связанности сеточных элементов и, соответственно, поиск соседних ячеек не должны становиться сложными в вычислительном плане операциями. Кроме того, требуют отдельного рассмотрения такие вопросы, как: хранить или вычислять координаты адаптированных ячеек, находить всех соседей при необходимости или хранить внутри ячейки ссылки на некоторых из них, как расположить в динамической памяти поля физических величин, чтобы доступ к ним не требовал много лишнего времени из-за неконформности сетки? Отдельной проблемой служит распараллеливание алгоритмов на ЛАД декартовых сетках, которые фактически являются и неструктурированными, и неконформными. Для вычислительных систем с общей памятью актуальны вопросы размещения счетных ячеек в памяти, порядка обхода этих ячеек. Для систем с распределенной памятью встает вопрос динамической балансировки памяти между ее узлами.

Другой вопрос, который естественно возникает, когда речь заходит о декартовых сетках, – это представление геометрии задачи и учет граничных условий, которые в общем случае не согласованы с сеточными узлами.

Настоящий препринт посвящен описанному выше кругу вопросов, связанных с использованием декартовых ЛАД сеток. Решение этих вопросов реализовано в библиотеке прикладных программ для работы с ЛАД декартовыми сетками. Библиотека написана в объектно-ориентированном стиле на языке программирования C++ и адаптирована на параллельные вычисления на многоядерных процессах с общей памятью (технология OpenMP). Она учитывает специфику трехмерных задач: заточена под минимизацию загружаемой памяти (координаты ячеек не хранятся внутри них, а вычисляются при необходимости) и учитывает возможность параллелизации (порядок обхода предварительно записывается в специальный список с целью упрощения параллелизации). Для работы с геометрией задачи мы используем оригинальный подход, основанный на локальной адаптации несогласованных декартовых сеток к требуемой геометрии и учете граничных условий на полученных в результате адаптации несогласованных сетках методом свободной границы [7,8]. Библиотека тестируется на задаче Л.И. Седова о сильном взрыве, происходящем в замкнутом ограниченном жесткими стенками пространстве.

1. Представление неконформной сетки в памяти компьютера

Определение оптимального и удобного формата представления данных – одна из ключевых задач при использовании сеток с локальной адаптацией. Для адаптации изначально структурированных декартовых сеток, где разбиение производится по какому-то заранее выбранному закону, удобно использовать различные древовидные структуры. Кроме того, для них хорошо известны быстрые рекурсивные алгоритмы обхода и перестроения [9].

Для описания трехмерной сетки с возможностью локальной адаптации предлагается следующий формат. Ячейка может разбиваться только на восемь равновеликих частей делением пополам по каждому из трех координатных направлений. В этом случае удобно говорить о наличии восьми потомков данной ячейки. В качестве базовой сетки берется первоначальная регулярная декартовая сетка в параллелепипеде, представляющем расчетную область. Размер этой сетки составляет $M \times N \times K$ ячеек, которым приписывается нулевой уровень.

Каждая ячейка адаптивной сетки описывается своим уровнем *lvl* (т.е. сколько разбиений начиная с нулевого уровня было проведено для ее получения) и виртуальным положением на этом уровне – тройкой индексов (*i, j, k*), таких, какими они были бы в случае полного заполнения данного уровня. Таким образом, ячейки ЛАД декартовой сетки представляются восьмеричным графом (деревом). Каждый узел дерева хранит флаг наличия подразбиения (потомков). Если он равен единице, ячейка обязательно имеет 8 потомков и хранит указатели на них. Для листовых ячеек дерева (не имеющих подразбиений более низкого уровня) эти указатели нулевые. Эти ячейки являются расчетными, и для них устанавливаются ненулевые указатели на координаты и вектор физических параметров.

Кроме того, из соображений точности аппроксимации исходных уравнений принимается еще одно дополнительное ограничение на конфигурацию сетки, состоящее в том, что соседние листовые ячейки не должны отличаться более чем в 2 раза. Другими словами, разница уровней двух соседних ячеек по модулю должна быть меньше или равна единице.

Такой тип данных позволяет описывать адаптивные сетки различных конфигураций без использования лишней памяти. Этот формат будем называть восьмеричным деревом по числу потомков родительской ячейки. Рис. 1 иллюстрирует пример декартовой ЛАД сетки в двумерном случае. Здесь число потомков равно 4 и, соответственно, дерево, представляющее формат данных, является четвертичным. Описанный выше формат характеризуется рядом свойств, которые становятся очень полезными при разработке численных методов и алгоритмов на рассматриваемых декартовых ЛАД сетках. К ним относятся следующие.

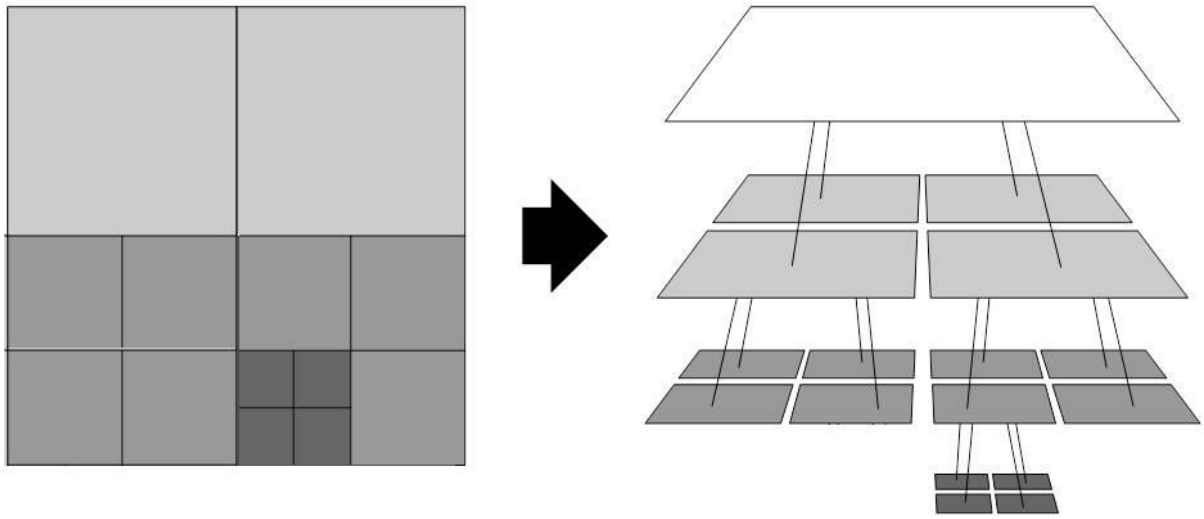

Рис. 1. Четверичное дерево как формат представления двумерной декартовой сетки с измельчением.

Ячейка 1-го уровня с индексами (i, j, k) обязательно имеет своим родителем ячейку с индексами $\left([i/2],[j/2],[k/2]\right)$, где $[\cdot/\cdot]$ – операция целочисленного деления и нумерация обоих индексов начинается с нуля, $i = 0, M-1;\ j = 0, N-1;\ k = 0, K-1.$

Число соседей первого уровня для ячейки ЛАД сетки варьируется от 26 до 52, число прямых соседей (по граням) – от 6 до 12.

Если записать виртуальные координаты ячейки (*i, j, k*) на ее уровне *lvl* в двоичной системе исчисления так, чтобы число знаков двоичной записи составляло *lvl*, то можно заметить следующее полезное свойство. Тройка цифр, стоящих на s-й позиции двоичных записей индексов *i* , *j* и *k*, определяет положение s-го прародителя (*s*< *lvl*) в ячейке *s-1*–го. Например, если тройка оказывается (0,0,0), то *s*-й прародитель является нижней, левой и фронтальной ячейкой ячейки *s-1*–го уровня, (1,0,0) – верхней, левой и фронтальной, (0,1,0) – нижней, правой и фронтальной, (1,0,1) – верхней, левой и задней и т.д.

Последнее свойство иллюстрируется рис. 2 для двумерного случая. Для ячейки с *lvl* = 2 и (i,j)=(1,2), показанной оранжевым, двоичная запись виртуальных индексов есть (012,102). Первые цифры индексов – 0 и 1 – показывают, что сначала рассматривается нижний правый потомок базовой ячейки (бордовая стрелка), вторые цифры – 1 и 0 – адресуют ячейку как верхнего левого подпотомка выбранной ячейки 1-го уровня (белая стрелка).

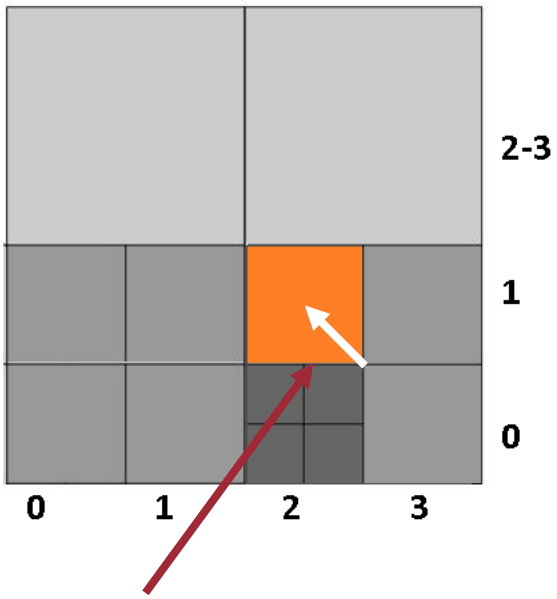

Рис. 2. Иллюстрация к определению адреса ячейки через двоичную запись ее виртуальных индексов на уровне адаптации в двумерном случае.

Таким образом, 4 адресные координаты ячейки, (*lvl, i, j, k*) однозначно определяют топологическое положение рассматриваемой ячейки на многоуровневой локально-адаптивной сетке.

Исходя из вышеприведенного, цикл по ячейкам декартовой ЛАД сетки можно реализовать следующей простой процедурой. В тройном цикле по индексам базовой сетки обходятся все ячейки базовой сетки. Если в этом цикле текущая ячейка является конечной (листовой), то к ее вектору решения применяется конкретная расчетная функция солвера. В противном случае вызывается процедура обхода по всем восьми потомкам данной ячейки с анализом флага подразбиения. Эта процедура рекурсивно повторяется до тех пор, пока не будет найден уровень с листовыми ячейками, к которым уже и применяется соответствующая функция солвера.

При организации цикла по соседям некоторой текущей листовой ячейки необходимо иметь эффективный алгоритм поиска множества соседних к текущей листовых ячеек. Для этого удобно использовать свойство 3 используемого формата данных, изложенного выше. Рассмотрим некоторую листовую ячейку с координатами (*lvl, i, j, k*) в качестве текущей, для которой необходимо осуществить поиск всех ее соседей. Предположим, что соседями по i-му направлению является ячейка такого же размера, что и текущая. Тогда, очевидно, соседом по этому направлению (с тыльной стороны) будет одна листовая ячейка с координатами (*lvl, i+1, j, k*). При поиске ячейки с такими координатами (прохождении по определяемому координатами пути) возможны 3 случая. (1) Ячейка оказалась листовой, т.е. является физической ячейкой ЛАД сетки, и мы ее берем в качестве соседа. (2) Найденная ячейка имеет флаг, равный единице. Тогда она имеет потомков, которые, в силу ограничения на уровень адаптации для соседей, и будут листовыми. В этом случае текущая

ячейка имеет 4 соседа с тыльной стороны, которые относятся к уровню *lvl+1,* и с координатами, определяемыми по свойству 3). (3) В процессе поиска мы не дошли до искомой ячейки и остановились на уровне *lvl-1*. Тогда эта ячейка и будет правым соседом исходной, просто предыдущего уровня *lvl-1*.

2. Основные положения дискретной модели

Рассматривается численное решение системы уравнений Эйлера, описывающей движение идеальной сжимаемой жидкости. Система определяющих уравнений в декартовых координатах записывается в консервативной форме

$$\frac{\partial \mathbf{q}}{\partial t} + \frac{\partial \mathbf{f}_k}{\partial x_k} = 0, \tag{1}$$

где подразумевается суммирование по индексу координатного направления *k*, *k=1,2,3*

$\mathbf{q} = \left(\rho, \rho u_m, \rho E\right)^T$ – вектор консервативных переменных,

$\mathbf{f}_k = \left(\rho u_k, \rho u_k u_m + \delta_{km} p, \rho u_k H\right)^T$ – векторы потоков,

$m = 1,2,3$, δ_{km} – символ Кронекера, ρ, u_k, p – плотность, компоненты вектора скорости и давление соответственно, $E = e + 0.5 u_k u_k$, $H = e + p/\rho$ – удельные полная энергия и энтальпия. Система уравнений (1) замыкается уравнением состояния, связывающим термодинамические параметры среды, которое в настоящей работе берется в форме уравнения состояния идеального совершенного с показателем адиабаты γ: $p = (\gamma - 1)\rho e$. В расчетах, которые будут представлены ниже, показатель адиабаты берется для случая воздуха, $\gamma = 1.4$.

Рассмотрим сначала случай без локальной сеточной адаптации, когда дискретизация по пространству делается регулярной базовой декартовой сеткой, ориентированной вдоль координатных направлений, с шагами h_k, которые, вообще говоря, могут быть переменными. Применяя метод конечного объема к уравнениям (1), приходим к следующей системе полудискретных уравнений:

$$\frac{d\mathbf{q}_i}{dt} = -\left(\frac{\Delta \mathbf{F}_k}{h_k}\right)_i, \tag{2}$$

где i является обобщенным индексом ячейки, зависящим от индекса координатного направления k и принимающим значение порядкового номера ячейки в этом направлении.

Разность потоков в правой части (2)

$$\Delta\mathbf{F}_k = \mathbf{F}_{k,i+1/2} - \mathbf{F}_{k,i-1/2}, \tag{3}$$

где $\mathbf{F}_{k,i+1/2}$ – численный поток, аппроксимирующий дифференциальный поток $\mathbf{f}_k$ на грани между ячейками i и $i+1$. Этот численный поток берется в виде функции двух векторных аргументов

$$\mathbf{F}_{k,i+1/2} = \mathbf{F}_k\left(\mathbf{z}_i^+, \mathbf{z}_{i+1}^-\right), \tag{4}$$

удовлетворяющей условию аппроксимации $\mathbf{F}_k\left(\mathbf{z},\mathbf{z}\right) = \mathbf{f}_k(\mathbf{z})$. Здесь $\mathbf{z}$ обозначает вектор примитивных переменных, $\mathbf{z} = \left(\rho, u_k, p\right)$.

Верхние индексы «+» и «-» в правой части уравнений (3) указывают, что соответствующие величины берутся в центре грани $i+1/2$. Выбор этих величин определяется точностью схемы. Например, схема первого порядка точности по пространству получается при простом выборе

$$\mathbf{z}_i^+ = \mathbf{z}_i, \qquad \mathbf{z}_{i+1}^- = \mathbf{z}_{i+1}. \tag{5}$$

Для увеличения порядка аппроксимации схемы необходимо применять подсеточную реконструкцию параметров более высокого порядка точности. В настоящей работе мы используем кусочно-линейное подсеточное восполнение, обобщающее на неравномерные сетки интерполяцию схемы MUSCL (Monotone Upstream-Centered Scheme for Conservation Laws) [10]:

$$\mathbf{z}^{\pm} = \mathbf{z} \pm 0.5\delta^{\pm}\left[(1-k^{\pm})\Delta^{\mp} + (1+k^{\pm})\Delta^{\pm}\right] \tag{6}$$

с конечными разностями $\Delta^+ = \mathbf{z}_{i+1} - \mathbf{z}_i$, $\Delta^- = \mathbf{z}_i - \mathbf{z}_{i-1}$. В этом уравнении для сокращения записи опущен индекс ячейки i,
$\delta^+ = h_i / (h_i + h_{i+1})$, $\delta^- = h_i / (h_i + h_{i-1})$ – параметры неравномерности сетки,
$k^{\pm} = k(\delta^{\pm})$ – функция, определяющая порядок аппроксимации интерполяционной схемы. При выборе $k(\delta) = -1$ получается стандартная MUSCL схема второго порядка точности [10], $k(\delta) = 1$ дает неустойчивую центрально-разностную схему второго порядка, $k(\delta) = 0$ отвечает схеме Фромма [11], $k(\delta) = (12\delta^2 - 1)/(12\delta)$ приводит к MUSCL схеме третьего порядка аппроксимации по пространству. Ниже мы используем этот (последний) вариант схемы.

Схемы порядка точности по пространству второго и выше являются немонотонными и приводят к нефизичным осцилляциям в численных решениях вблизи поверхностей сильных разрывов [12]. Для подавления этих осцилляций используют ограничители производных [10, 13]. Действие ограничителя сводится к модификации разностей $\Delta^{\pm}$ таким образом, чтобы интерполяция (6) не приводила к образованию локальных экстремумов.

Наиболее распространенными являются *minmod* ограничитель [10, 13]:

$$\Delta^+ = \boldsymbol{minmod}(\Delta^+, \varphi\Delta^-), \quad \Delta^- = \boldsymbol{minmod}(\Delta^-, \varphi\Delta^+)$$

$$\varphi = \frac{3-k}{1-k}, \qquad \boldsymbol{minmod}(x,y) = \begin{cases} 0, \text{ если } xy < 0 \\ sign(x)\,\boldsymbol{min}(|x|,|y|), \text{ если } xy \geq 0 \end{cases} \tag{7}$$

и ограничитель Ван Альбада [14]:

$$\mathbf{z}^{\pm} = \mathbf{z} \pm 0.5 s \delta^{\pm} \left[(1 - sk^{\pm})\Delta^{\mp} + (1 + sk^{\pm})\Delta^{\pm} \right]$$

$$s = \boldsymbol{max}\left(0, \frac{2\Delta^+\Delta^-}{\Delta^+\Delta^+ + \Delta^-\Delta^- + \varepsilon}\right) \tag{8}$$

где ε – малое число, служащее для предотвращения деления на ноль ($\varepsilon \sim 10^{-12}$ для операций с двойной арифметикой, $\varepsilon \sim 10^{-6}$ – для одинарной). Первый ограничитель производных не является непрерывно дифференцируемой функцией и может приводить к зацикливанию невязки в процессе сходимости решения. Мы используем второй ограничитель в виде гладкой функции, который во многих задачах решает проблему зацикливания невязки.

Метод С.К. Годунова [15] применяется для аппроксимации функции численного потока. В этом подходе поток вычисляется на решении автомодельной задачи Римана для системы локально-одномерных уравнений:

$$\frac{\partial \mathbf{q}}{\partial t} + \frac{\partial \mathbf{f}_k}{\partial x} = 0 \tag{9}$$

с начальными данными $\mathbf{z}(0,x) = \begin{cases} \mathbf{z}_i^+, & x < 0 \\ \mathbf{z}_{i+1}^-, & x \geq 0 \end{cases}$.

Решение этой задачи подробно описывается в монографии [16]. На практике оно сводится к решению одного нелинейного уравнения относительно давления контактной зоны, что может быть реализовано ньютоновскими итерациями. Остальные компоненты решения имеют точные аналитические выражения. Обозначим это решение как $\mathbf{z}(t,x) = \mathbf{Z}^{R,k}\left(\lambda, \mathbf{z}_i^+, \mathbf{z}_{i+1}^-\right)$, где $\lambda = x/t$ – автомодельная переменная. Тогда стандартный годуновский численный поток будет иметь вид

$$\mathbf{F}_k\left(\mathbf{z}_i^+, \mathbf{z}_{i+1}^-\right) = \mathbf{f}_k\left[\mathbf{Z}^{R,k}\left(0, \mathbf{z}_i^+, \mathbf{z}_{i+1}^-\right)\right]. \tag{10}$$

Расчет численного потока выполняется на каждом ребре счетной ячейки и является по сути основной (в смысле временных затрат) операцией расчетного цикла. Поэтому в целях повышения эффективности схемы С.К. Годунова были разработаны различные безытерационные приближенные решения задачи

Римана [17]. Одно из таких приближений приводит к численному потоку, предложенному В.В. Русановым [18]. Он имеет следующий вид:

$$\mathbf{F}_k\left(\mathbf{z}_i^+,\mathbf{z}_{i+1}^-\right)=\frac{1}{2}\left\{\mathbf{f}_k\left(\mathbf{z}_i^+\right)+\mathbf{f}_k\left(\mathbf{z}_{i+1}^-\right)-\left(\left|u_k\right|+c\right)_{i+1/2}\left[\mathbf{q}\left(\mathbf{z}_{i+1}^-\right)-\mathbf{q}\left(\mathbf{z}_i^+\right)\right]\right\}, \qquad (11)$$

где индекс $i+1/2$ означает осреднение, например, $(*)_{i+1/2}=0.5\left[(*)_i^+ + (*)_{i+1}^-\right]$.

Интегрирование по времени системы полудискретных уравнений (2)-(4) с численным потоком в форме (10) проводится с использованием явной двухшаговой схемы типа предиктор-корректор. Здесь на первом шаге вычисляются значения предиктора на полушаге по времени по явной эйлеровой схеме:

$$\tilde{\mathbf{q}}_i=\mathbf{q}_i^n-\frac{\Delta t}{2}\left(\frac{\Delta\mathbf{F}_k(\mathbf{z}^n)}{h_k}\right)_i \qquad (12)$$

с потоком $\mathbf{F}_{k,i+1/2}$, определяемым по интерполированным на грани ячейки значениям,

$$\Delta\mathbf{F}_k(\mathbf{z}^n)=\mathbf{f}_k(\mathbf{z}_i^+)-\mathbf{f}_k(\mathbf{z}_i^-), \qquad (13)$$

где верхний индекс n обозначает дискретный уровень по временной переменной. Решение на новом временном слое $n+1$ получается по явной схеме второго порядка точности на шаге Δt с потоками, вычисляемыми по значениям предиктора:

$$\mathbf{q}_i^{n+1}=\mathbf{q}_i^n-\Delta t\left(\frac{\Delta\mathbf{F}_k(\tilde{\mathbf{z}})}{h_k}\right)_i;$$
$$\mathbf{F}_{k,i+1/2}=\mathbf{F}_k\left(\tilde{\mathbf{z}}_i^+,\tilde{\mathbf{z}}_{i+1}^-\right). \qquad (14)$$

Схема гарантирует (в линейном случае) устойчивость при выполнении условия Куранта-Фридрихса-Леви на шаг интегрирования по времени,

$$\Delta t\le\lambda_i(\mathbf{z}^n) \text{ для всех } i, \qquad (15)$$

где функция в правой части определяется локальными скоростью течения и скоростью звука:

$$\lambda_i(\mathbf{z}^n)=K_s\left(\frac{\left|u_k\right|+C}{h_k}\right)^{-1}. \qquad (16)$$

Здесь K_s – коэффициент запаса ($0<K_s\le 1$).

Модификация описанного выше численного метода на случай декартовых ЛАД сеток вложенной структуры связана с изменением схемы вычисления численного потока $\mathbf{F}_{k,i+1/2}$ в уравнениях (2) и (3). Теперь этот поток должен учитывать возможность неконформного (не грань в грань) соседства ячеек, когда по грани текущей ячейки примыкают два сеточных элемента меньшего уровня адаптации или соседний элемент относится к уровню, на единицу

большему, чем уровень текущей ячейки. В первом случае поток $\mathbf{F}_{k,i+1/2}$ вычисляется как сумма с весами ½ потоков между текущей ячейкой и двумя ее соседями. При этом значение вектора $\mathbf{z}_i$ интерполируется в текущей ячейке на соответствующую координату соседней ячейки.

Если соседний элемент является ячейкой большего уровня, чем уровень текущей, значение $\mathbf{z}_{i+1}$ интерполяцией в соседней ячейке приводится к соответствующей координате текущей ячейки. В остальном все формулы расчетной схемы, приведенные выше, как для аппроксимации 1-го порядка, так и аппроксимации 2-го порядка, остаются в силе.

3. Описание библиотеки для работы с трехмерными ЛАД сетками

Для программной реализации описанной выше численной методики была разработана библиотека прикладных программ для работы с трехмерными декартовыми ЛАД сетками. В ней собраны функции, необходимые для построения, измельчения и огрубления сетки, поиска соседних элементов, а также других функций, используемых в программной реализации численной методики.

В библиотеке используются два типа сеточных функций, которые условно называются геометрическая и физическая. Геометрическая сеточная функция служит для построения геометрии элементов сетки с использованием параметрических и заданных многоугольниками поверхностей. Физическая функция используется для описания численного решения физических задач.

Для построения сеточной геометрии задачи требуется определить расчетные области, которые задаются с использованием ограничивающих параметрических поверхностей, таких как сфера, цилиндр или более сложных, аппроксимируемых многоугольниками. При препроцессинге активно используется функция нахождения пересечения элементов сетки (прямоугольного параллелепипеда) и многоугольника. В основе используемого алгоритма лежит метод последовательного отсечения многоугольника плоскостями параллелепипеда.

После построения геометрической сетки с использованием геометрических поверхностей на ее основе инициализируется физическая сеточная функция. С ее помощью производится постановка дискретной задачи и реализуется численный расчет. Элементы сеточной функции содержат векторы состояния среды, векторы численных потоков и коэффициенты адаптации произвольной размерности. Во время расчета производится последовательный обход всех элементов, поиск соседей и расчет новых значений физических величин.

Библиотека организована с использованием шаблонов, что позволяет задавать произвольный размер векторов физических величин. Для этого

необходимо при определении объекта сетки указать в угловых скобках количество определяющих физическое состояние среды параметров в векторах примитивных значений, потока и коэффициентов адаптации: **TPhysGrid<5,4,2>.** Для деления элемента используется функция **Divide(),** для огрубления – функция **Merge().** Поиск соседей используется при помощи функции **FoundNeights().** Существует целый ряд других функций и процедур, используемых в программе, которые описываются ниже.

Как уже было сказано выше, существуют два типа сеточных функций: геометрическая и физическая. Для описания геометрической функции служит класс **CGeoGrid**. Он служит для построения геометрии сетки вокруг поверхностей объектов. С его помощью можно находить пересечения элементов сетки и элементов поверхности, определять, какие ячейки являются внутренними, внешними, и тому подобное. Физическая сеточная функция описывается с помощью класса **TPhysGrid**, который служит для описания модели расчетной сетки, содержащей физические параметры (такие как плотность, скорость, давление). Класс оперирует ячейками следующих типов: **SolidCell** – твердотельные ячейки, **VolumeCell** – ячейки объема, содержащие физическую информацию, **BorderCell** – граничные ячейки. Также имеется большой функционал для использования локально-адаптивных сеток с произвольными численными методами, в который входят различные функции манипуляции сеткой.

Программная реализация библиотеки разделена на пять основных частей с целью логической организации программных модулей по смыслу их использования. Каждая из них представляет собой динамически компонуемую библиотеку на языке C++: **BasicLib, MathLib, GeometryLib, GridLib, DrawLib.**

Библиотека **BasicLib** содержит базовые компоненты библиотеки для работы со строками, массивами, списками, словарями, указателями и другими подобными структурами. В ней содержатся базовые подпрограммы, используемые во всех остальных библиотеках комплекса. Библиотека **MathLib** содержит подпрограммы для работы с математическими объектами, такими как матрица, вектор, интервал, нормаль, угол, репер и так далее. Библиотека **GeometryLib** содержит классы и функции для работы с двумерными поверхностями в трехмерном пространстве. Как было сказано ранее, для представления поверхностей, ограничивающих объемы, в библиотеке обычно используется метод триангуляции. Поверхности представляются в виде своей аппроксимации, состоящей из многоугольников. Также существует возможность использования параметрического задания поверхностей, таких как сфера, прямоугольный параллелепипед, цилиндр и других. В библиотеке **GridLib** содержатся подпрограммы для работы с декартовыми трехмерными локально-адаптивными сетками. Библиотка **DrawLib** служит для отображения геометрии и результатов расчетов средствами OpenGL.

3.1. Устройство и принципы функционирования библиотеки

На рис. 3 приведен пример локально-адаптивной декартовой сетки. Из рисунка видно, что элементы декартовой сетки содержат внутри себя вложенные структуры декартовых сеток, что обеспечивает ее сгущение и разрежение. Это сделано с целью уменьшения количества используемых элементов и экономии вычислительных ресурсов. Равномерная декартовая сетка требует значительного количества элементов для удовлетворительного представления результатов расчета.

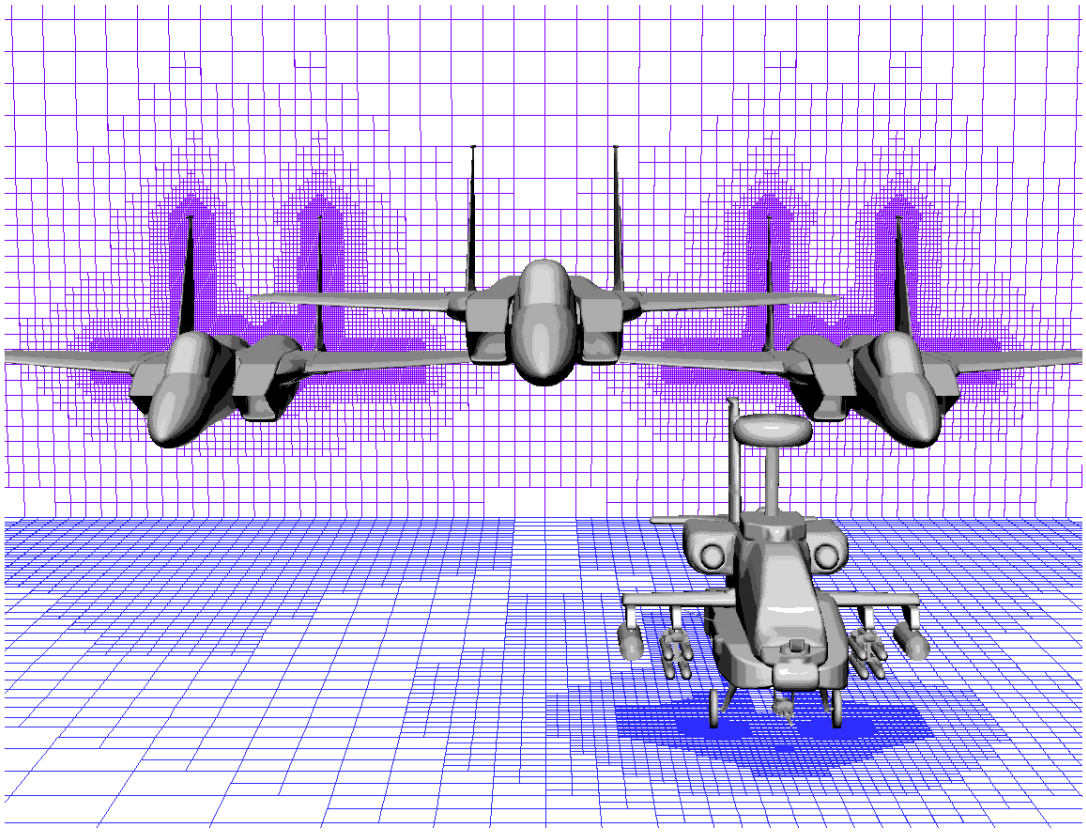

Рис. 3. Пример локально-адаптивной декартовой сетки

Таким образом, локально-адаптивная декартова сетка представляет собой структуру вложенных прямоугольных декартовых сеток. Предполагается многократная рекурсивная вложенность. Запишем на псевдоязыке структуру элемента сетки (здесь приведено упрощенное определение, в программе используется несколько расширенный подход):

```
class Cell {
   Cell *child[N,M,K];
};
```

Из описания видно, что элемент `Cell` содержит в себе указатели на массив таких же элементов `Cell`.

Каждая ячейка определяется координатами своих вершин. Эти координаты не хранятся, так как это потребовало бы использования значительного объема оперативной памяти. Поскольку каждая из ячеек является простой декартовой структурой, существует возможность ввести их нумерацию, используя номер уровня вложенности и индексный номер ячейки (*xNum, yNum, zNum*) вдоль осей *X, Y, Z* на однородной декартовой сетке, соответствующей уровню вложенности. Этой информации достаточно, чтобы определить геометрические координаты вершин. Таким образом, данное выше псевдо-определение расширяется еще несколькими элементами: номером уровня и номерами элементов вдоль осей *X, Y, Z*:

```
class Cell {
   Cell *child[N];
   int level;
   int xNum, yNum, zNum;
};
```

При таком подходе экономится память и отпадает необходимость хранения ссылок на соседние элементы сетки. При расчетах с использованием пространственных сеток требуется определить соседство для вычисления потоков между ячейками. Хранение ссылок на соседние элементы в ЛАД сетках потребовало бы огромного объема оперативной памяти, т.к. каждый элемент может соседствовать со множеством других, более мелких, а кроме того, при расчетах требуется постоянная перестройка сетки для выявления особенностей решения и вся информация о соседстве должна синхронно обновляться, что сложно реализовать. Вместо этого, имея номер ячейки, можно вычислить номера соседей, обойти иерархию сверху вниз и обнаружить их. Этот подход экономит память и не вносит больших задержек в расчет, так как не требуется много времени для обхода дерева, потому как номер, а значит и местоположение заранее известны.

Так же каждая из ячеек несет в себе дополнительную информацию, зависящую от типа ячейки. Например, ячейка объема должна содержать вектор физических значений, граничная ячейка содержит в себе еще и данные об условиях на границе, твердотельная ячейка содержит в себе нормаль и отсекаемые объемы для уточнения поведения решения вблизи поверхности твердого тела и так далее. Для реализации этого подхода можно было бы хранить в каждой ячейке всю информацию, которая только может потребоваться, но лучше, для экономии ресурсов, сохранять данные динамически в виде указателя на тип *void*, например. В программе используется

механизм виртуальных функций. В нашем псевдокоде определим дополнительные данные в виде указателя на неизвестные данные:

```
class Cell {
   Cell *child[N];
   int level;
   int xNum, yNum, zNum;
   enum type;
   void *additionalData;
};
```

Таким образом (в упрощенном представлении) определяется ячейка декартовой сетки.

Для манипуляций над ячейками сетки используется класс `Grid`. В этом классе определены методы для поиска соседей, расчета значений, вводится дополнительная определяющая информация об уровнях и тому подобное.

Главным элементом класса `Grid` является указатель на начальную ячейку сетки, потомками которой являются все остальные ячейки:

```
class Grid {
   Cell *head;
};
```

Таким образом, в упрощенном виде описано устройство и функционирование декартовой ЛАД сетки. Для поиска соседей необходимо вычислить номер уровня, номера ячеек соседей и обойти сетку сверху вниз для того, чтобы добраться до их места хранения. Для локального увеличения разрешения сетки нужно добавить дочерние элементы, для огрубления – наоборот, удалить дочерние элементы.

Для осуществления всех этих процедур и динамического создания/удаления потомков требуются структуры динамического управления памятью, такие как массив, список, словарь, множество. Для описания координат требуются такие математические объекты, как векторы и матрицы. Для описания поверхностей, определяющих твердотельные элементы, требуются такие объекты, как полигон, поверхность, примитивы (сфера, параллелепипед, цилиндр). Для вывода результатов требуются подпрограммы рисования и т.д. Все эти структуры описаны в Приложении 1. Приложение 2 содержит пример реализации численного метода на декартовой ЛАД сетке.

4. Численные результаты

В настоящем разделе приводятся результаты численных экспериментов по тестированию метода и библиотеки декартовых ЛАД сеток. В качестве теста была выбрана задача о распространении трехмерной взрывной волны в

замкнутом пространстве. Начальные данные для расчетов соответствовали решению задачи Л.И. Седова о сильном взрыве [19]. Расчетная область представляет собой параллелепипед с координатами противоположных углов (-0.8, -0.8, -0.25), (0.8, 0.8, 0.8). Начальный параллелепипед может последовательно быть разбит на 8 потомков вплоть до L раз, L – число уровней.

В начальный момент времени всюду в расчетной области задается поле примитивных переменных $(\rho, U_x, U_y, U_z, p) = (1, 0, 0, 0, 10^{-6})$, кроме шара радиуса 0.01 с центром в (0, 0, 0), где задается $p=(\gamma-1)/(1.033V_s)$, V_s – объем шара. Такое начальное давление вызывает волну, фронт которой проходит расстояние 1 за время 1. Вблизи шара сетка в начальный момент разбита до максимального уровня адаптации. Коэффициент запаса для шага по времени K_s =0.5.

В этих расчетах используется алгебраический критерий адаптации (сумма градиентов вдоль всех направлений для плотности и давления) с порогами 0.05 для измельчения и 0.01 для огрубления.

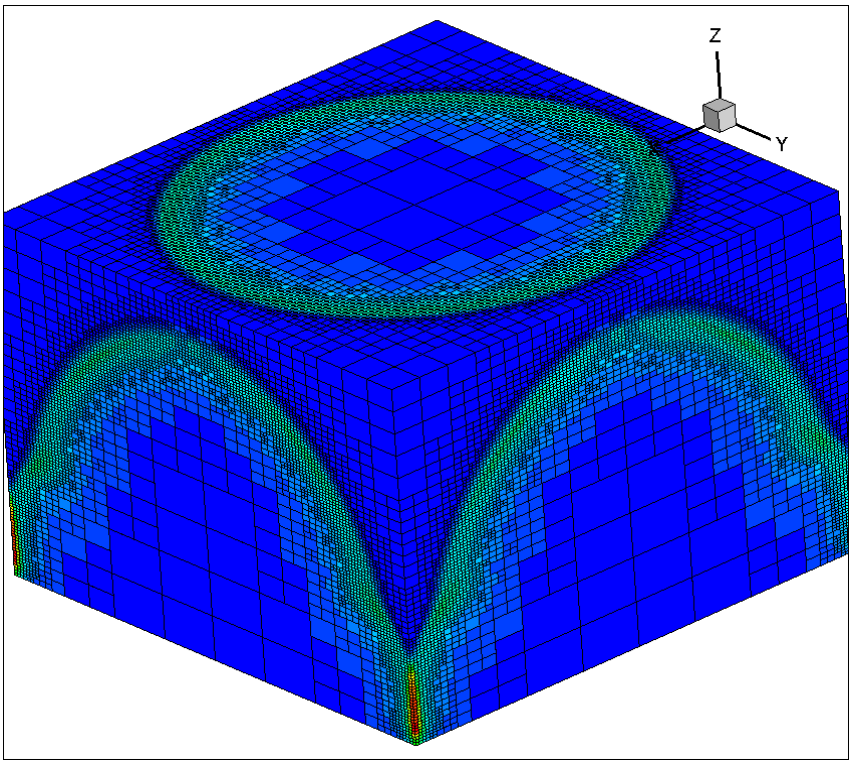

Рис. 4. Поле плотности на границах параллелепипеда
с 7 уровнями адаптации, момент времени t=1, вид сверху

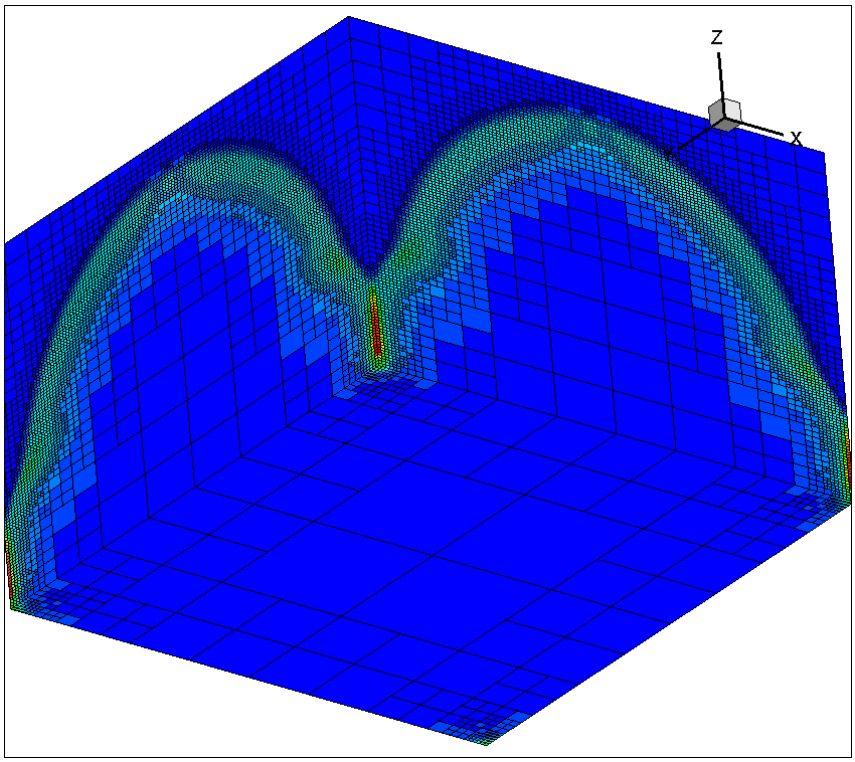

Рис. 5. Поле плотности на границах параллелепипеда
с 7 уровнями адаптации, момент времени t=1, вид снизу

На рисунках 4 и 5 видно, что такой критерий приводит к хорошей адаптации к поверхностям разрывов. На боковых стенках мы видим результат сложения ударной волны непосредственно от взрыва и отраженной волны от нижней стенки, самой ближней к точке начального вложения энергии (0, 0, 0). На верхней стенке видно отражение сферического фронта начальной ударной волны.

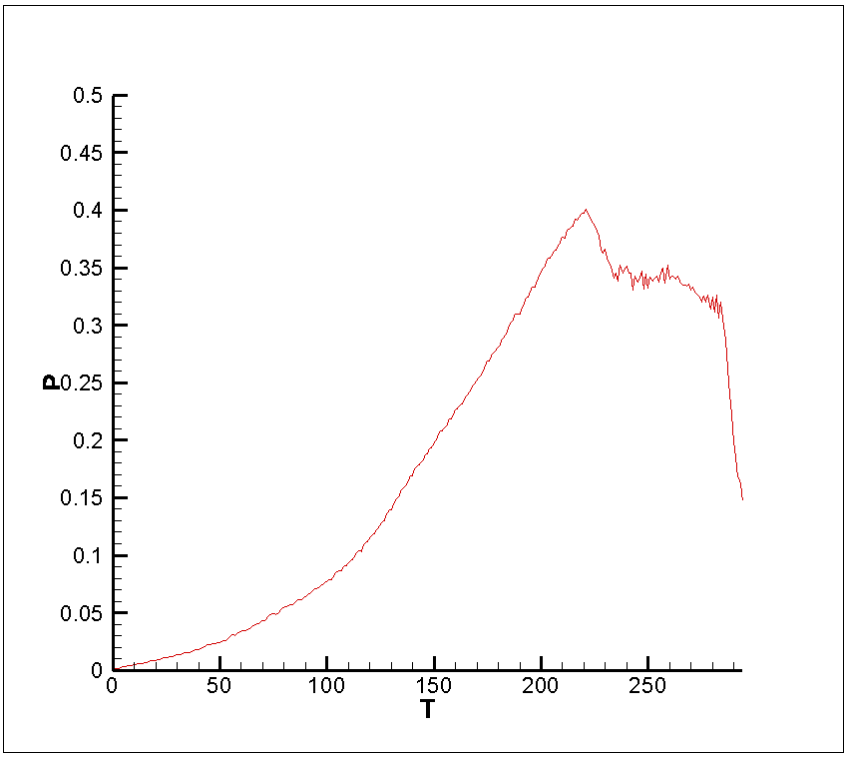

Рис. 6. График заполненности сетки P для расчета с 6 уровнями адаптации

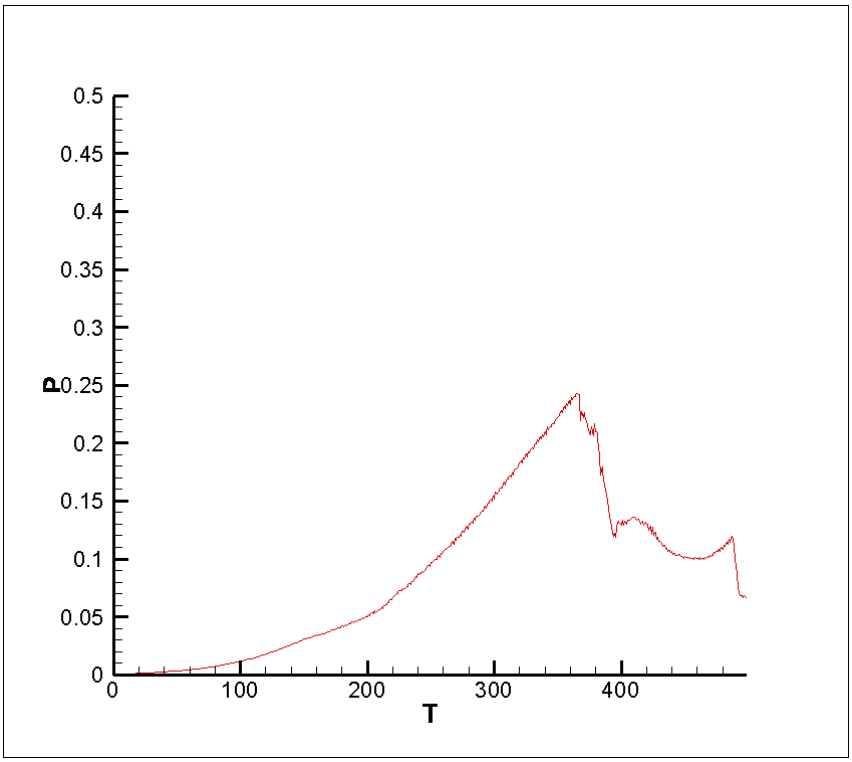

Рис. 7. График заполненности сетки P для расчета с 7 уровнями адаптации

Эффективность адаптивной сетки может быть выражена с помощью величины $P=N_c/8^L$, где N_c – общее число расчетных ячеек, L – число уровней, 8^L – максимальное число ячеек при данном числе уровней. На рисунках 6 и 7 изображены графики зависимости заполненности сетки P от шага по времени T. В среднем по шагам процент заполненности составил 18.6% для 6 уровней, 8.7% для 7 уровней.

Таблица 1. Тестирование распараллеливания задачи с 6 уровнями адаптации на общей памяти (OpenMP)

Число процессоров	Время счета, с	Ускорение	Эффективность распараллеливания
1	5724	-	100%
2	3597.5	59.1%	79.6%
4	2957.8	93.5%	48.4%

Таблица 1 показывает эффективность распараллеливания задачи при разном числе процессоров n. Величина ускорения A_n показывает увеличение скорости счета на данном числе процессоров по сравнению со скоростью счета на одном процессоре. Эффективность распараллеливания $E_n=(1+ A_n)/n$.

5. Заключение

В работе описана методика применения ЛАД сеток при численном моделировании трехмерных газодинамических задач и представлено описание программной реализации этих методов в виде прикладной C++ библиотеки, использующей интерфейс распараллеризации с общей памятью OpenMP. Результаты показывают высокую эффективность динамической адаптации с использованием ЛАД сеток и возрастание ее эффективности при увеличении числа уровней ЛАД сетки, т.к. при измельчении сетки вдвое число ячеек, принадлежащих поверхностям разрывов, растет на порядок медленнее общего, максимального числа ячеек.

Список литературы

1. *F. Bramkamp, Ph. Lamby, and S. Mueller.* An adaptive multiscale finite volume solver for unsteady and steady state flow computations. J. Comp. Phys., 197(2):460-490, 2004.
2. *A. Harten.* Multiresolution algorithms for the numerical solution of hyperbolic conservation laws. Comm. Pure Appl. Math., 48(12):1305-1342, 1995.
3. *A. Harten.* Multiresolution representation of data: A general framework. SIAM J. Numer.Anal, 33(3):1205-1256, 1996.

4. *G. Zumbusch.* Parallel multilevel methods. Adaptive mesh refinement and loadbalancing Advances in Numerical Mathematics. Teubner, Wiesbaden, 2003.
5. *S. Osher and R. Sanders.* Numerical approximations to nonlinear conservation laws with locally varying time and space grids. Math. Сотр., 41:321-336, 1983
6. *O.V. Vasilyev.* Solving multi-dimensional evolution problems with localized structures using second generation wavelets. Int. J. Comp. Fluid Dyn. 17:151–168, 2003.
7. *И.С. Меньшов, М.А. Корнев.* Метод свободной границы для численного решения уравнений газовой динамики в областях с изменяющейся геометрией // Матем. моделирование, 26:5 (2014), с. 99–112.
8. *И.С. Меньшов, П.В. Павлухин.* Численное решение задач газовой динамики на декартовых сетках с применением гибридных вычислительных систем // Препринты ИПМ им. М.В. Келдыша. 2014. № 92, 24 с.
9. *А.А. Сухинов.* Построение декартовых сеток с динамической адаптацией к решению //Математическое моделирование. – 2010. – Т. 22. – №. 1. – С. 86-98.
10. *B. Van Leer.* Towards the ultimate conservative difference scheme V: A second-order sequel to Godunov's method // J. Comp. Phys., 32 (1979), p. 101-136.
11. *J.E.Fromm.* A method for reducing dispersion in convective difference schemes // J. Comp. Phys., 3 (1968), p. 176-187.
12. *С.К. Годунов, В.С. Рябенький.* Разностные схемы // М.: Наука. 1977. 440 с.
13. *В.П. Колган.* Применение принципа минимальных значений производной к построению конечно-разностных схем для расчета разрывных решений газовой динамики //Ученые записки ЦАГИ, 3:6 (1972), с. 68–77.
14. *G.D. van Albada, B. van Leer, W. Roberts.* A comparative study of computational methods in cosmic gas dynamics // Astron. Astrophys, 108 (1982), p. 76-84.
15. *С.К. Годунов.* Разностный метод численного расчета разрывных решений уравнений гидродинамики // Мат. сборник, 47:3 (1957), с. 271–306.
16. *С.К. Годунов, А.В. Забродин, М.Я. Иванов, А.Н. Крайко, Г.П. Прокопов.* Численное решение многомерных задач газовой динамики // М.: Наука. 1976. 400 с.
17. *E. Toro.* Riemann solvers and numerical methods for fluid dynamics // Springer. 2009. p. 719.
18. *В.В. Русанов.* Разностные схемы третьего порядка точности для сквозного счета разрывных решений // Докл. АН СССР, 180:6 (1968), с. 1303–1305.
19. *Л.И. Седов.* Методы подобия и размерности в механике. Наука, М. -1977. –с 243-274.

Приложение 1. Описание основных структур библиотеки декартовых ЛАД сеток

Принятые обозначения

В целях унификации программного кода приняты следующие соглашения: имена переменных начинаются со строчной буквы, имена функций и классов начинаются с заглавной буквы. При этом перед именем класса указывается заглавная буква, обозначающая тип класса. `C` – обычный класс, `T` – шаблон, `I` – интерфейсный класс, содержащий чисто виртуальные функции. Имена переменных – членов класса начинаются с префикса `m_` (member). Если в имени класса имеется цифра, то обычно она отражает размерность объекта класса, например, `CVertex3` означает вершину в трехмерном пространстве, `CVector2` – вектор в двумерном пространстве и т.д.

Библиотека BasicLib

Для реализации функционала библиотеки требуются различные структуры данных, такие как массив, список, словарь и множество. Необходим также набор различных типов указателей для удобства работы с памятью. Все эти элементы содержатся в библиотеке BasicLib. Корме того, в ней содержатся некоторые вспомогательные подпрограммы для отладки, работы со строками и таймером.

Классы для работы со структурами данных:

`TFiniteVector <T_ElementsType, T_ElementsCount>` – шаблонный класс для представления массива конечной длины, известной на этапе компиляции. Параметр `T_ElementsType` определяет тип элементов массива, `T_ElementsCount` – количество элементов массива.

`TVector<T_ElementsType>` – массив для хранения элементов типа `T_ElementsType;`

`TList<T_ElementsType>` – список для хранения элементов типа T_ElementsType;

`TMap<T_ElementsType>` – словарь для хранения элементов типа T_ElementsType;

`TSet<T_ElementsType>` – множество для хранения элементов типа `T_ElementsType.`

Классы для работы с указателями:

`TSimplyPointer<T_VarType>` – простой указатель на тип `T_VarType` без какой-либо дополнительной функциональности;

`TValue<T_VarType>` – указатель на тип `T_VarType` обеспечивающий автоматическое создание/уничтожение данных;

`TSharedPointer<T_VarType>` – указатель, обеспечивающий разделяемое использование одних и тех же данных типа `T_VarType`, что означает

автоматическое копирование данных при изменении, а также подсчет ссылок и автоматическое удаление неиспользуемой памяти.

Прочие вспомогательные классы:

`CString` – класс для работы со строками;

`CDebug` – класс для проведения отладки программы;

`CTimer` – класс для проведения замеров времени выполнения программы и определения ее эффективности.

Библиотека MathLib

Классы для работы с векторами:

`CInt2, CInt3, CInt4` – классы для работы с векторами, содержащими целочисленные компоненты размерности 2, 3 и 4 соответственно;

`CReal2, CReal3, CReal4` – классы для работы с веществennозначными векторами размерности 2, 3 и 4;

`CNormal2, CNormal3` – классы для работы с векторами нормали размерности 2 и 3.

Классы для работы с матрицами:

`TMatrix<T_ElementsType>` – класс для работы с матрицами произвольного размера, составленными из элементов типа `T_ElementsType;`

`TMatrix2<T_ElementsType>` – класс для работы с матрицами размера 2x2, составленными из элементов типа `T_ElementsType;`

`TMatrix3<T_ElementsType>` – класс для работы с матрицами размера 3x3, составленными из элементов типа `T_ElementsType.`

Прочие классы:

`CRadian, CDegree` – классы для работы с углами в радианах и градусах соответственно;

`TInterval<T_ElementsType›` – класс для работы с интервалами типа `T_ElementsType.`

Библиотека GeometryLib

Классы для работы с поверхностями, состоящими из полигонов:

`CVertex3` – класс, определяющий вершину многоугольника поверхности в трехмерном пространстве. Вершина содержит информацию о координатах, нормали, координатах текстуры и материале. Полигоны, из которых состоят поверхности, определяются своими вершинами.

`CTriangle3` – класс, описывающий треугольник в трехмерном пространстве, определяемый вершинами типа `CVertex3;`

`CQuadrangle3` – класс, описывающий четырехугольник в трехмерном пространстве, определяемый вершинами типа `CVertex3;`

CPoly3 – класс, описывающий многоугольник в трехмерном пространстве, определяемый вершинами типа CVertex3;

`TRegularPoly3` – класс, описывающий полигон, обладающий специальными свойствами, в данном случае – плоский и выпуклый

многоугольник. Объекты этого класса не могут быть созданы непосредственно, т.к. их конструктор закрыт, но могут быть получены из объектов других классов, например, из класса нерегулярного многоугольника методом его деления на выпуклые, плоские полигоны.

`TSurface3` – класс, описывающий поверхность, состоящую из многоугольников.

Классы для работы с параметрическими поверхностями:

`TAABox` – класс для работы с выровненным по осям параллелепипедом;

`TAAPlane` – класс для работы с плоскостью в трехмерном пространстве;

`TSphere` – класс, описывающий сферу в трехмерном пространстве;

`TAACylinder` — класс, описывающий цилиндр в трехмерном пространстве, выровненный относительной координатных осей;

`TAALine2` – класс, описывающий прямую в двумерном пространстве, выровненную относительно координатных осей;

`TAALine3` – класс, описывающий прямую в трехмерном пространстве, выровненную относительно координатных осей;

`TCircle2` – класс, описывающий окружность на плоскости;

`TAACircle3` – класс, описывающий окружность в трехмерном пространстве, выровненную относительно координатных плоскостей;

`TAARect2` – класс, описывающий прямоугольник на плоскости, выровненный относительно координатных осей;

`TAARect3` – класс, описывающий прямоугольник в пространстве, выровненный относительно координатных осей;

`TAASegment2` – класс, описывающий отрезок на плоскости, выровненный относительно координатных осей;

`TAASegment3` – класс, описывающий отрезок в трехмерном пространстве, выровненный относительно координатных осей;

`TEdge3` – класс, описывающий ребро многоугольника в трехмерном пространстве.

Прочие классы:

`CColor3` – класс, описывающий значение цвета из трех компонент;

`CColor4` – класс, описывающий значение цвета из четырех компонент;

`CMaterial` – класс, описывающий характеристики материала поверхности;

`CObjFileLoader` – класс, отвечающий за загрузку модели из obj-файла;

`CAADirection2` – класс, описывающий направление на плоскости (X, Y);

`CAADirection3` – класс, описывающий направление в пространстве (X, Y, Z);

`CAAOrientation2` – класс, описывающий ориентацию на плоскости (-X, +X, -Y, +Y);

`CAAOrientation3` – класс, описывающий ориентацию в пространстве (-X, +X, -Y, +Y, -Z, +Z).

Библиотека GridLib

Классы библиотеки `GridLib` предназначены для проведения операций с декартовыми сетками. Существуют два вида сеток, они состоят из двух типов ячеек. Это физические сетки и ячейки (`PhysGrid/PhysCell`) и геометрические сетки и, соответственно, ячейки (`GeoGrid/GeoCell`). Физические сетки (`PhysGrid`) предназначены для проведения вычислений. Ячейки сетки хранят в себе векторы физических значений и содержат методы для поиска соседей, деления, огрубления сетки, проведения физических расчетов, установки/чтения значений и тому подобного. Геометрические сетки (`GeoGrid`) предназначены для построения декартовой адаптивной геометрии вокруг объектов, заданных поверхностями, состоящими из полигонов. Впоследствии геометрические сетки могут быть преобразованы в физические и использованы для расчетов. Сетки `GeoGrid` содержат методы для адаптации к поверхности, нахождения пересечений, подсеточного разрешения и т.п.

Ячейки сеток содержат как однотипную информацию, например, номер ячейки, уровень в иерархии и т.п., так и информацию, специфичную для конкретного элемента. Например, ячейки объема содержат вектор физических значений, ячейки поверхности – информацию об отсекаемых объемах, нормалях и поверхностях, граничные ячейки содержат информацию о значениях на границе и т.д.

Элементы физических сеток:

`TPhysCell` – класс, определяющий ячейку, принадлежащую сетке PhysGrid, используемой для численных расчетов;

`TPhysGrid` – класс, описывающий сетку, используемую для расчетов.

Типы, определяющие дополнительную информацию, хранящуюся в ячейках сетки:

`CPhysErrorData` – класс, содержащий дополнительные данные, описывающие ячейку, содержащую ошибку;

`CPhysHollowData` – класс, содержащий дополнительные данные, описывающие пустую ячейку, не содержащую никакой дополнительной информации;

`CPhysParentData` – класс, содержащий дополнительную информацию для ячейки, являющейся родительской и содержащей в себе другие ячейки;

`CPhysSolidData` – класс, содержащий дополнительную информацию для ячейки, являющейся "твердой" ячейкой;

`TPhysVolumeData` – класс, содержащий дополнительную информацию для ячейки, являющейся ячейкой объема;

`TPhysBorderData` – класс, содержащий в себе дополнительную информацию о граничной ячейке;

`TPhysSolidFNeightsData`, `TPhysVolumeFNeightsData`, `TPhysBorderFNeightsData` – классы, содержащие в себе дополнительную

информацию о ячейках, которые помимо описанной выше информации хранят в себе информацию об элементах, соседних спереди;

TPhysSolidBNeightsData, TPhysVolumeBNeightsData, TPhysBorderBNeightsData – то же, что и предыдущие, но хранится информация о соседях сзади;

TPhysSolidNeightsData, TPhysVolumeNeightsData, TPhysBorderNeightsData – хранится информация о всех соседях.

Элементы геометрических сеток:

CGeoCell – класс, определяющий ячейку, принадлежащую сетке GeoGrid, используемой для построения геометрии расчетной области;

CGeoGrid – класс, описывающий сетку, используемую для построения геометрии.

Типы, определяющие дополнительную информацию, хранящуюся в ячейках сетки:

CGeoErrorData – класс, содержащий дополнительные данные, описывающие ячейку, содержащую ошибку;

CGeoHollowData – класс, содержащий дополнительные данные, описывающие пустую ячейку, не содержащую никакой дополнительной информации;

CGeoParentData – класс, содержащий дополнительную информацию для ячейки, являющейся родительской и содержащей в себе другие ячейки;

CGeoUndefinedData – класс, содержащий дополнительную информацию для ячейки, тип которой пока не определен и нуждается в дополнительном тестировании;

CGeoInnerSurfaceData – класс, содержащий дополнительную информацию для ячеек поверхности, отделяющих внутренние области;

CGeoBorderSurfaceData – класс, содержащий дополнительную информацию для ячеек поверхности, определяющих границу области;

CGeoExcludeSurfaceData – класс, содержащий дополнительную информацию для ячеек поверхности, определяющих исключаемые области;

CGeoVolumeData – класс, содержащий дополнительную информацию для ячейки, не являющейся ячейкой поверхности.

Библиотека DrawLib

CTracerPoint3 – класс, определяющий точку, принадлежащую линии тока;

CTracer3 – класс, описывающий линию тока, служащую для визуализации результатов численных расчетов;

TTracers3 – класс, описывающий целый набор линий тока, позволяющий задавать сразу ряд трассирующих элементов;

TSliceElement – класс, описывающий элемент, принадлежащий срезу сетки;

TPlaneSlice – класс, описывающий срез, сделанный при помощи плоскости;

CPointLight – класс, описывающий точечный источник света;

CDirectionalLight – класс, описывающий направленный источник света;

CScene – класс, описывающий трехмерную сцену;

CDrawer – класс, отвечающий за рисование геометрических данных (моделей и сеток).

Наиболее важные классы и методы, используемые для работы с ЛАД-сетками

Наиболее важными классами в библиотеке являются TPhysGrid – для работы с физическими сетками, класс CGeoGrid – для работы с геометрическими сетками, класс ISolver1 – для реализации численного метода на адаптивных сетках. В этом разделе описаны наиболее важные и значимые их методы, необходимые для организации численного метода.

Класс TPysGrid используется для проведения расчетов, хранения физической информации и взаимодействия с классом реализации численного метода первого порядка ISolver1 и метода второго порядка ISolver2.

Основными методами класса сетки являются:

CellsList()– возвращает список всех рассчитываемых элементов сетки;

CellsVector()– возвращает массив всех рассчитываемых элементов сетки;

Refine()– метод, используемый для балансировки сетки с целью исключения соседства элементов, различающихся более чем на один уровень;

Divide()– разделить все ячейки сетки;

SetVolumeValues()– метод для установки значений для ячеек объема;

SetBorderValues()– метод для установки граничных значений сетки.

Основными методами класса ячеек являются:

LevelNumber()– функция возвращает номер уровня, которому принадлежит ячейка;

Parent()– функция возвращает родительскую ячейку для текущей;

Grid()– функция возвращает сетку, которой принадлежит ячейка;

Number()– функция возвращает номер ячейки;

Box()– функция возвращает ограничивающий параллелепипед ячейки;

Corn1()– функция возвращает "левый нижний угол" ячейки;

Corn2()– функция возвращает "правый верхний угол" ячейки;

XSquare()– функция возвращает площадь грани ячейки, расположенной в направлении оси OX;

`YSquare()`– функция возвращает площадь грани ячейки, расположенной в направлении оси OY;

`ZSquare()`– функция возвращает площадь грани ячейки расположенной в направлении оси OZ;

`Volume()`– функция возвращает объем ячейки;

`IsChildExist()` – функция возвращает факт наличия в ячейки потомков;

`Status()` – функция возвращает тип ячейки;

`Refine()` – функция производит балансировку данного элемента, чтобы исключить соседство данного элемента с элементами, отличающимися более чем на один уровень;

`FoundNeights()` – функция возвращает всех соседей для данной ячейки;

`FrontNeights()` – функция возвращает всех соседей "спереди" для данной ячейки;

`BackNeights()` – функция возвращает всех соседей "сзади" для данной ячейки;

`IsVolumeCell()` – функция возвращает флаг того, что ячейка принадлежит объему;

`IsBorderCell()` – функция возвращает флаг того, что ячейка принадлежит границе;

`IsSolidCell()` – функция возвращает флаг того, что это твердотельная ячейка.

Приложение 2. Пример реализации численного метода на ЛАД декартовой сетке

Рассмотрим программную реализацию алгоритма, который сводится к выполнению следующих функций.

1. Получить все ячейки сетки, принадлежащие объему.
2. Пройти их все и вычислить коэффициенты адаптации.
3. Выполнить измельчение/огрубление ячеек на основании коэффициентов адаптации.
4. Получить список всех ячеек.
5. Вычислить шаг по времени.
6. Пройти все ячейки, найти соседей, вычислить значения векторов потока для каждой ячейки.
7. Пересчитать новые значения примитивного вектора на основании вычисленных значений векторов потока.

Ниже дано описание соответствующего численного кода.

```
#include "grid.h"
```

```
typedef TPhysGrid<5,5,2> MyGrid;
int main(void)
{
  // Определение сетки
  MyGrid grid;
  // Получаем все ячейки объема сетки
  MyGrid::CellsVector cells = grid.VolumeCells();
  // Цикл по ячейкам
  for (int i=0; i<cells.Count(); i++) {
       // Текущая ячейка
       MyGrid::CellType cell = cells[i];
       // Ищем соседей ячейки
       MyGrid::CellsVector neights = cell.Neights();
       // Текущая соседняя ячейка
       MyGrid::CellType neightCell = neights[i];
       if (neightCell.IsVolumeCell())
            // Вычислить коэффициенты адаптации

  }
  // Цикл по ячейкам
  for (int i=0; i<cells.Count(); i++) {
       // Текущая ячейка
       MyGrid::CellType cell = cells[i];
       // Если текущая ячейка крайняя в иерархии
       if (cell.IsLastCell()) {
            if (cell.AFactor() > 0.1)
                 cell.Divide();
       }
       //  Если  ячейка  содержит  только  крайние  ячейки  в
качестве потомков
       else if (cell.IsFatherCell()) {
            if (cell.AFactor() < 0.05)
                 cell.Merge();
       }

  }
  // Получаем все ячейки сетки
  cells = grid.Cells();
  // Шаг по времени
  double dt = 0.0;
  // Цикл по всем ячейкам
  for (int i=0; i<cells.Count(); i++) {
```

```
        Вычислить шаг по времени
}
  // Цикл по всем ячейкам
  for (int i=0; i<cells.Count(); i++) {
        // Текущая ячейка
        MyGrid::CellType cell = cells[i];
        // Ищем соседей ячейки
        MyGrid::CellsVector neights = cell.Neights();
        // Если ячейка принадлежит объему
        if (cell.IsVolumeCell()) {
             // Обходим всех соседей
             for (int j=0; j<neights.Count(); j++) {
                  // Текущая соседняя ячейка
                  MyGrid::CellType neightCell = neights[i];
                  if (neightCell.IsVolumeCell())
                       // Выполнить действия
                  else if (neightCell.IsBorderCell())
                       // Выполнить действия
                  else if (neightCell.IsSurfaceCell())
                       // Выполнить действия
             }
        }
        // Иначе, если ячейка принадлежит поверхности
        else if (cell.IsSurfaceCell()) {
             // Обходим всех соседей
             for (int j=0; j<neights.Count(); j++) {
                  // Текущая соседняя ячейка
                  MyGrid::CellType neightCell = neights[i];
                  if (neightCell.IsVolumeCell())
                       // Выполнить действия
                  else if (neightCell.IsBorderCell())
                       // Выполнить действия
                  else if (neightCell.IsSurfaceCell())
                       // Выполнить действия
             }
        }
  }

  return 0;
}
```