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

Препринты ИПМ • Препринт № 23 за 2016 г.

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

Краснов М.М., Ладонкина М.Е., Тишкин В.Ф.

Разрывный метод Галёркина на трёхмерных тетраэдральных сетках. Использование операторного метода программирования

Рекомендуемая форма библиографической ссылки: Краснов М.М., Ладонкина М.Е., Тишкин В.Ф. Разрывный метод Галёркина на трёхмерных тетраэдральных сетках. Использование операторного метода программирования // Препринты ИПМ им. М.В.Келдыша. 2016. № 23. 27 с. doi:10.20948/prepr-2016-23
URL: http://library.keldysh.ru/preprint.asp?id=2016-23

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

М.М. Краснов, М.Е. Ладонкина, В.Ф. Тишкин

Разрывный метод Галёркина на трёхмерных тетраэдральных сетках. Использование операторного метода программирования

Москва — 2016

М.М. Краснов, М.Е. Ладонкина, В.Ф. Тишкин

Разрывный метод Галёркина на трёхмерных тетраэдральных сетках. Использование операторного метода программирования

Операторный метод программирования позволяет компактно записывать математические формулы в программах и облегчает перенос программ на параллельные архитектуры, такие как NVidia CUDA и Intel Xeon Phi. Ранее операторный метод программирования был реализован для регулярных трёхмерных декартовых сеток и трёхмерных локально-адаптивных сеток. В данной работе этот метод переносится на трёхмерные тетраэдральные сетки. На этом примере иллюстрируется возможность эффективной реализации данного метода на произвольных трёхмерных сетках. Кроме того, в работе демонстрируется применение методов шаблонного метапрограммирования языка C++ для ускорения вычислений.

Ключевые слова: операторной метод программирования, трёхмерные тетраэдральные сетки, разрывный метод Галёркина, CUDA, шаблонное метапрограммирование

Mikhail Mikhailovich Krasnov, Marina Evgenievna Ladonkina, Vladimir Fedorovich Tishkin

Discontinuous Galerkin method on three-dimensional tetrahedral meshes. The usage of the operator programming method

Operator programming method allows the compact writing of mathematical formulas in programs and helps to transfer the programs to parallel architectures, such as NVidia CUDA and Intel Xeon Phi. Earlier the operator programming method was implemented for regular three-dimensional Cartesian grids and tree-dimensional locally adaptive grids. In this work, this method is transferred to three-dimensional tetrahedron meshes. This example illustrates the possibility of method implementation on arbitrary tree-dimensional meshes. Besides, in this work we demonstrate the usage of template metaprogramming methods of the C++ programming language to speed-up calculations.

Key words: operator programming method, three-dimensional tetrahedral mesh, discontinuous Galerkin method, CUDA, template metaprogramming

Работа выполнена при поддержке грантов РФФИ №14-01-00145, № 16-01-00333

Оглавление

1. Введение

В задачах математического моделирования широко используются сеточные функции – величины, определённые в каждом узле некоторой сетки. Для численного решения задач математического моделирования с этими сеточными функциями делаются определённые преобразования. В математической литературе эти преобразования описываются операторами, такими, например, как оператор Лапласа, градиента, дивергенции. Кроме того, в уравнениях могут встречаться линейные комбинации и композиции операторов. Например, при решении эллиптического уравнения многосеточным методом итерирующий оператор для двух уровней имеет вид (см. [1]):

$$Q = S_p \left(I - P A_H^{-1} R A_h \right) S_p ,$$

где A_H и A_h – операторы (матрицы) на подробной и грубой сетках соответственно, P – оператор интерполяции (продолжения), R – оператор сборки (проектирования), S_p – сглаживающий оператор, p – число пре- и пост-сглаживающих шагов (число применений оператора A_h), I – единичный оператор.

В теоретических работах описание алгоритмов выглядит очень компактно и элегантно. При реальном программировании текст программы выглядит гораздо более громоздко и часто требует дополнительной памяти для сохранения промежуточных результатов вычислений. Операторный метод программирования (см. [2]) позволяет приблизить, насколько это возможно, внешний вид программы к формулам в теоретических работах. Ещё одна немаловажная проблема заключается в том, что в настоящее время большое распространение получили гибридные суперкомпьютеры с вычислительными

графическими платами, а их эффективное использование затруднено, т.к. требует освоения большого объёма новой и непривычной информации о методах программирования на них.

Первоначально операторный метод был реализован в виде библиотеки для трёхмерных регулярных декартовых сеток с целью переноса многосеточного метода на параллельные архитектуры, такие как NVidia CUDA [3] и Intel Xeon Phi [4]. Позже операторный метод был реализован для решения параболических уравнений на локально-адаптивных сетках, которые уже не являются регулярными, но всё ещё имеют некоторую структуру. В данной работе делается дальнейшее развитие операторного метода – перенос на нерегулярные трёхмерные тетраэдральные сетки. Обобщение на произвольные трёхмерные сетки теперь уже является сравнительно несложной задачей.

В качестве модельной задачи для реализации операторного метода на трёхмерных тетраэдральных сетках был выбран разрывный метод Галёркина [5-10]. У этого метода относительно сложная математика, которая хорошо формулируется в терминах операторов (лимитирования, вычисления объёмных интегралов и потоков через грани), и на нём операторный метод может показать все свои преимущества. Кроме того, реализация метода Галёркина имеет и практическую ценность. Как известно, разрывный метод Галёркина характеризуется высоким порядком точности решения, что влечет за собой и высокую вычислительную сложность, что является большим недостатком метода Галёркина. В данной работе активно применяется механизм метапрограммирования на языке C++, позволяющий вынести часть вычислений на стадию компиляции. Было интересно оценить, насколько это позволит ускорить метод и, таким образом, частично сгладить этот недостаток. Интересно было также посмотреть, какое ускорение получится на графических ускорителях CUDA, перенос программы на которые при использовании операторного метода программирования является тривиальной задачей (это делается практически простой перекомпиляцией исходного текста).

2. Описание разрывного метода Галеркина для уравнений Эйлера

Мы будем рассматривать трёхмерное уравнение Эйлера, записанное в консервативной форме:

$$\partial_t U + \nabla \cdot F(U) = 0, \qquad (1)$$

и дополненное подходящими начально-краевыми условиями, вид которых зависит от конкретной задачи, и которые будут конкретизированы далее.

Консервативные переменные $\boldsymbol{U}$ и компоненты потоковой функции $F(U)$ заданы в виде

$$U = \begin{Bmatrix} \rho \\ \rho u \\ \rho v \\ \rho w \\ E \end{Bmatrix}, \quad F(U) = \begin{pmatrix} F_x(U) \\ F_y(U) \\ F_z(U) \end{pmatrix},$$

$$F_x(U) = \begin{Bmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ \rho uw \\ (E+p)u \end{Bmatrix}, \quad F_y(U) = \begin{Bmatrix} \rho v \\ \rho uv \\ \rho v^2 + p \\ \rho vw \\ (E+p)v \end{Bmatrix}, \quad F_z(U) = \begin{Bmatrix} \rho w \\ \rho uw \\ \rho vw \\ \rho w^2 + p \\ (E+p)w \end{Bmatrix}, \tag{2}$$

где ρ – плотность жидкости, u, v, w – компоненты скорости v, p – давление, ε – удельная внутренняя энергии и $E = \rho\left(e + \frac{u^2+v^2+w^2}{2}\right)$ – полная энергия на единицу объёма.

Для определения давления p будем использовать уравнение состояния идеального газа:

$$p = (\gamma - 1)\rho\varepsilon, \tag{3}$$

где γ – показатель адиабаты.

Для применения разрывного метода Галеркина покроем область Ω, на которой ищется решение, тетраэдральной сеткой T_h. На каждом элементе T_j приближённое решение системы уравнений (1) будем искать в виде полиномов $P(\mathrm{x})$ степени N с зависящими от времени коэффициентами:

$$U_h(\mathrm{x},t) = \sum_{k=0}^{st} U_k(t)\phi_k(\mathrm{x}), \tag{4}$$

где st – размерность пространства полиномов, а $\phi_k(\mathrm{x})$ – соответствующая базисная функция. В данной работе в качестве базисных функций используется базис Тейлора:

$$\begin{aligned}
&\phi_0 = 1 \\
&\phi_1 = \frac{(x-x_c)}{\Delta x} \qquad \phi_2 = \frac{(y-y_c)}{\Delta y} \qquad \phi_3 = \frac{(z-z_c)}{\Delta z} \\
&\phi_4 = \frac{(x-x_c)^2}{(\Delta x)^2} \qquad \phi_5 = \frac{(y-y_c)^2}{(\Delta y)^2} \qquad \phi_6 = \frac{(z-z_c)^2}{(\Delta z)^2} \\
&\phi_7 = \frac{(x-x_c)(y-y_c)}{\Delta x \Delta y} \qquad \phi_8 = \frac{(x-x_c)(z-z_c)}{\Delta x \Delta z} \qquad \phi_9 = \frac{(y-y_c)(z-z_c)}{\Delta y \Delta z}
\end{aligned} \tag{5}$$

где x_c, y_c, z_c – координаты центра масс соответствующего тетраэдра, $\Delta x, \Delta y, \Delta z$ – проекция ячейки на оси x, y и z.

$$x_c = \frac{x_1 + x_2 + x_3 + x_4}{4}, \qquad y_c = \frac{y_1 + y_2 + y_3 + y_4}{4}, \qquad z_c = \frac{z_1 + z_2 + z_3 + z_4}{4}.$$

Приближённое решение системы (1) в разрывном методе Галёркина ищется как решение следующей системы [5]:

$$\begin{aligned} &\frac{d}{dt}\int_{T_j} \phi_k(\mathrm{x}) \cdot U_h(\mathrm{x},t)\, d\Omega - \\ &-\int_{T_j} \left(\frac{\partial \phi_k(\mathrm{x})}{\partial x} F_x\left(U_h(\mathrm{x},t)\right) + \frac{\partial \phi_k(\mathrm{x})}{\partial y} F_y\left(U_h(\mathrm{x},t)\right) + \frac{\partial \phi_k(\mathrm{x})}{\partial z} F_z\left(U_h(\mathrm{x},t)\right) \right) d\Omega + \\ &+\oint_{\partial T_j} \phi_k(\mathrm{x}) \cdot h_F\left(U_h^+, U_h^-, n\right) d\sigma = 0, \end{aligned} \qquad (6)$$

где $U_h(\mathrm{x},t)$ – вектор решения, n – вектор внешней единичной нормали по границе элемента ∂T_j. $h_F\left(U_h^+, U_h^-, n\right)$, – потоковые функции, вычисленные на границе элемента ∂T_j. Величины, обозначенные через U_h^-, вычисляются на границе ∂T_j элемента T_j по значениям внутри элемента T_j, в то время как величины, обозначенные через U_h^+, вычисляются на границе ∂T_j по значениям в соседней к данному элементу T_j ячейке. $h_F(U_h^+, U_h^-, n)$ – численная потоковая функция, зависящая от значений приближённого решения по обе стороны границы элемента и от направления единичного нормального вектора n. Функция $h_F(\cdot,\cdot,\cdot)$ – функция монотонного численного потока, для которой выполнено условие согласования

$$h_F\left(U_h(\mathrm{x},\mathrm{t}), U_h(\mathrm{x},\mathrm{t}), n\right) = F(U_h(\mathrm{x},\mathrm{t})). \qquad (7)$$

В данной работе использовались потоки Русанова–Лакса–Фридрихса [12,13]

$$\begin{aligned} &h_F\left(U_h(\mathrm{x}^l,\mathrm{t}), U_h(\mathrm{x}^r,\mathrm{t}), n\right) = \\ &\qquad \frac{1}{2}\left(F(U_h(\mathrm{x}^l,t) + F(U_h(\mathrm{x}^r,t)) - A \cdot (U_h(\mathrm{x}^r,t) - U_h(\mathrm{x}^l,t))) \right), \\ &\qquad A = \max\left(\left|\mathrm{v}^l\right| + c^l, \left|\mathrm{v}^r\right| + c^r \right), \end{aligned} \qquad (8)$$

где v – скорость, c – скорость звука.

Все интегралы, появляющиеся в элементных уравнениях, вычисляются в среднем численными квадратурными формулами Гаусса, число интеграционных точек соответствует необходимой точности [14,15].

Система обыкновенных дифференциальных уравнений (6), которая регулирует изменения во времени дискретного решения, может быть записана в виде

$$\frac{dU^p}{dt} = R(U^p), \quad R(U^p) = -A^{-1}\Pi_k^p, \tag{9}$$

где A обозначает матрицу масс, U^p – глобальный вектор степеней свободы (коэффициенты разложения всех искомых функций), k – порядковый номер уравнения в системе (1) и $R(U^p)$ – вектор правых частей.

3. Лимитирование

Как известно, для обеспечения монотонности решения, полученного данным методом, необходимо вводить так называемые ограничители наклона, или лимитеры, в особенности в том случае, если решение содержит сильные разрывы. Однако применение лимитеров может отрицательно сказаться на точности получаемого решения. Например, при использовании классического лимитера Кокбурна [5], к сожалению, снижается точность решения [7-9]. Идея данного лимитера легко реализуется как в одномерном, так и в двумерном и трёхмерном случаях, причём и на произвольных тетраэдральных сетках.

3.1. Лимитер Кокбурна

Перейдем к описанию лимитера Кокбурна $\Lambda\Pi_h$ в трёхмерном случае. Рассмотрим кусочно-линейную функцию u_h.

Лимитер $\Lambda\Pi_h$ для кусочно-линейных функций u_h должен удовлетворять условию сохранения «массы»:

для любого элемента $T_j \in T_h$

$$\int_{T_j} \Lambda\Pi_h u_h dxdy = \int_{T_j} u_h dxdy$$

Построим ограничитель $\Lambda\Pi_h$ на тетраэдральном элементе.

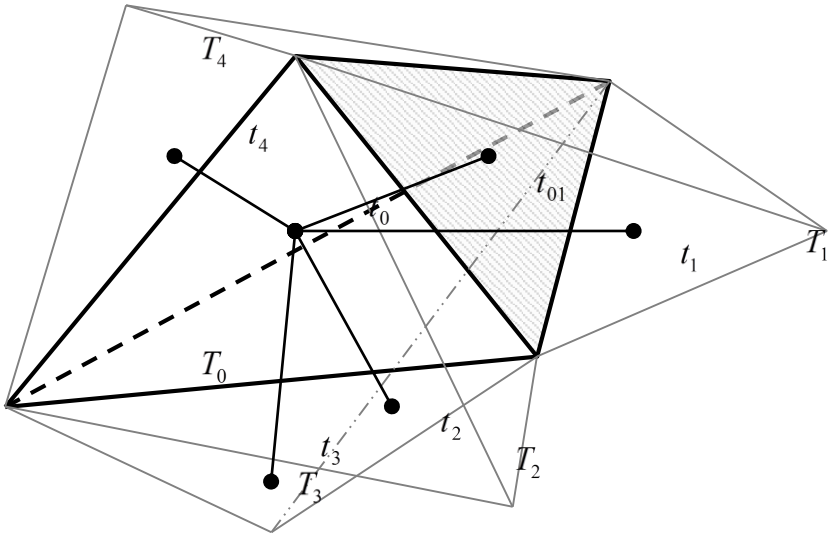

Рис. 1. Тетраэдральный элемент сетки, на котором проводится лимитирование

Введём обозначение:

t_0 – центр масс рабочего тетраэдра T_0, в котором необходимо провести ограничение функций, T_i – тетраэдры, имеющие общую грань с T_0 .

t_i – центр масс тетраэдра T_i.

t_{0i} – центр масс грани с номером i.

$$u_0 = u(t_0),$$
$$u_i = u(t_i),$$
$$u_{0i} = u(t_{0i}).$$

Далее работаем с линейным разложением функции $u_h(x,y,z) = \sum_{i=0}^{9} u_i^k \, \phi_i(x,y,z)$.

$$u_h^{\eta}(x,y,z) = \tilde{u}_0^k + \tilde{u}_1^k (x - x_c) + \tilde{u}_2^k (y - y_c) + \tilde{u}_3^k (z - z_c), \qquad (10)$$

где

$$\tilde{u}_0 = \frac{1}{|T_i|} \int_{T_i} u_h(x,y,z)\,dxdydz$$

$$\int_{T_i} u_h(x,y,z)\phi_1(x,y,z)\,dxdydz = \tilde{u}_1 \int_{T_i} \phi_1(x,y,z)\phi_1(x,y,z)\,dxdydz +$$

$$\tilde{u}_2 \int_{T_i} \phi_2(x,y,z)\phi_1(x,y,z)\,dxdydz + \tilde{u}_3 \int_{T_i} \phi_3(x,y,z)\phi_1(x,y,z)\,dxdydz$$

$$\int_{T_i} u_h(x,y,z)\phi_2(x,y,z)\,dxdydz = \tilde{u}_1 \int_{T_i} \phi_1(x,y,z)\phi_2(x,y,z)\,dxdydz +$$

$$\tilde{u}_2 \int_{T_i} \phi_2(x,y,z)\phi_2(x,y,z)\,dxdydz + \tilde{u}_3 \int_{T_i} \phi_3(x,y,z)\phi_2(x,y,z)\,dxdydz$$

$$\int_{T_i} u_h(x,y,z)\phi_3(x,y,z)dxdydz = \tilde{u}_1 \int_{T_i} \phi_1(x,y,z)\phi_3(x,y,z)dxdydz +$$

$$\tilde{u}_2 \int_{T_i} \phi_2(x,y,z)\phi_3(x,y,z)dxdydz + \tilde{u}_3 \int_{T_i} \phi_3(x,y,z)\phi_3(x,y,z)dxdydz$$

Решая систему уравнений, получаем коэффициенты $\tilde{u}_0, \tilde{u}_1, \tilde{u}_2$ и $\tilde{u}_3$ линейного разложения функции $u_h(x,y,z)$.

Построив вектора:

$$l_{0i} = t_{0i} - t_0 = (x_{0i} - x_0, y_{0i} - y_0, z_{0i} - z_0),$$
$$l_j = t_j - t_0 = (x_j - x_0, y_j - y_0, z_j - z_0),$$

рассмотрим 4 трёхгранных угла: первый образован векторами: $\bar{l}_1, \bar{l}_2, \bar{l}_3$; второй, третий и четвертый – векторами: $\bar{l}_1, \bar{l}_2, \bar{l}_4$; $\bar{l}_1, \bar{l}_3, \bar{l}_4$; и $\bar{l}_2, \bar{l}_3, \bar{l}_4$ соответственно. Выберем среди них тот, в котором все коэффициенты разложения вектора l_{01} по векторам, образующим данный трёхгранный угол, неотрицательны (тот трёхгранный угол, в котором лежит вектор $\bar{l}_{oi}$)

$$l_{oi} = \sum_{\substack{j=1 \\ k \neq j}}^{3} \alpha_{ij}^k \bar{l}_j, \quad i = 1,4, \ \alpha_{i1}^k \geq 0, \quad \alpha_{i2}^k \geq 0, \quad \alpha_{i2}^k \geq 0.$$

Введём обозначения

$$\Delta u_{0i} = u_{0i}^{л} - u_0^{л},$$

$$\Delta \tilde{u}_{0i} = \sum_{j=1}^{3} \alpha_{ij}^{k_0} (u_j^{л} - u_0^{л}).$$

Пусть t_{oi}, $i = 1,2,3,4$ – центры масс граней тетраэдра T_0. Тогда для $(x,y,z) \in T_0$ можно записать

$$u_h^{л}(x,y,z) = \sum_{i=1}^{4} u_{oi}^{л} \cdot \varphi_i(x,y,z) = \tilde{u}_0 + \sum_{i=1}^{4} \Delta u_{oi} \cdot \varphi_i(x,y,z), \qquad (11)$$

где функция $\varphi_i(x,y,z) = \begin{cases} 1, & \text{в} \quad t_{0i}. \\ 0, & \text{в} \quad t_{0j}. \end{cases} \quad j \neq i.$

Вычислим величины

$\Delta u_{0i}^* = \bar{m}(\Delta u_{0i}, \nu \Delta u_{0i})$, где $\nu > 1$, а $\bar{m}$ – функция «*TVB* minmod» (в случае *TVB* – ограничителя).

$$\bar{m}(a_1, a_2) = \begin{cases} a_1, & \text{если} \ |a_1| \leq M(|l_{0i}|)^2 \\ m(a_1, a_2), & \text{иначе} \end{cases},$$

M – заданная константа, $|l_{0i}|$ – длина вектора l_{0i}.

$$m(a_1,a_2)=\begin{cases} s\cdot\min(|a_1|,|a_2|), & \text{если } s=\operatorname{sign}(a_1)=\operatorname{sign}(a_2),\\ 0, & \text{иначе.}\end{cases}$$

TVD – ограничитель.

Если $\sum_{i=1}^{4}\Delta u_{0i}^{*}=0,$ то полагаем

$$\Lambda\Pi_h u_h=\tilde{u}_0+\sum_{i=1}^{4}\Delta u_{0i}^{*}\varphi_i(x,y,z). \qquad (13)$$

Если $\sum_{i=1}^{4}\Delta u_{0i}^{*}\neq 0,$ то использовать формулу (13) нельзя, т.к. $\tilde{u}_0\neq\frac{1}{|T_i|}\int_{T_i}u_h(x,y,z)dxdydz$. В этом случае величины Δu_{0i}^{*} подвергаются коррекции в сторону уменьшения.

Вычисляем суммарный вклад положительных членов $\sum_{i=1}^{4}\Delta u_{0i}^{*}$

$$pos=\sum_{i=1}^{4}\max(0,\Delta u_{0i}^{*})$$

и суммарный вклад отрицательных членов $\sum_{i=1}^{4}\Delta u_{0i}^{*}$

$$neg=\sum_{i=1}^{4}\max(0,-\Delta u_{0i}^{*}).$$

Полагаем

$$\theta^{+}=\min\left(1,\frac{neg}{pos}\right),\qquad \theta^{-}=\min\left(1,\frac{pos}{neg}\right).$$

Определим величину

$$\Delta u_{i0}^{*}=\theta^{+}\max(0,\Delta u_{0i}^{*})-\theta^{-}\max(0,-\Delta u_{0i}^{*}).$$

Далее, если

$$\tilde{u}_0+\sum_{i=1}^{4}\Delta\tilde{u}_{i0}^{*}\cdot\varphi_i(x,y,z)=u_h^{л},$$

то ограничитель полагаем равным тождественному оператору, то есть исходная функция $u_h(x,y,z)$ после действия оператора $\Lambda\Pi_h u_h(x,y,z)$ не изменится:

$$\Lambda\Pi_h u_h(x,y,z) = u_h(x,y,z).$$

В противном случае полагаем

$$\Lambda\Pi_h u_h(x,y,z) = \tilde{u}_0 + \sum_{i=1}^{4} \Delta\tilde{u}_{i0}^{*} \cdot \varphi_i(x,y,z). \qquad (14)$$

При применении данных лимитеров важную роль играет выбор параметра $1 \le \alpha \le 2$. При α=1 получаем наиболее «жёсткий» ограничитель, обеспечивающий монотонность решения, при α=2 – «менее строгий» ограничитель $\Lambda\Pi_h$.

Замечание. Для методов высокого порядка точности необходимо использовать схемы высокого порядка по времени. Мы использовали схему Рунге-Кутта третьего порядка [5].

$$U^{*} = \Lambda\Pi_h\left\{U^n + \Delta t L(U^n)\right\}$$

$$U^{**} = \Lambda\Pi_h\left\{\frac{3}{4}U^n + \frac{1}{4}U^{*} + \frac{1}{4}\Delta t L(U^{*})\right\}$$

$$U^{n+1} = \Lambda\Pi_h\left\{\frac{1}{3}U^n + \frac{2}{3}U^{**} + \frac{2}{3}\Delta t L(U^{**})\right\}.$$

4. Реализация операторов

4.1. Оператор объёмного интегрирования

В методе Галёркина для каждого тетраэдра требуется вычислить пять объёмных интегралов:

$$I_{kj}^{1ev} = \int_{T_k}\left[(\rho u)\frac{\partial\varphi_j}{\partial x} + (\rho v)\frac{\partial\varphi_j}{\partial y} + (\rho w)\frac{\partial\varphi_j}{\partial z}\right]dV;$$

$$I_{kj}^{2ev} = \int_{T_k}\left[(\rho u^2 + p)\frac{\partial\varphi_j}{\partial x} + (\rho uv)\frac{\partial\varphi_j}{\partial y} + (\rho uw)\frac{\partial\varphi_j}{\partial z}\right]dV;$$

$$I_{kj}^{3ev} = \int_{T_k}\left[(\rho vu)\frac{\partial\varphi_j}{\partial x} + (\rho v^2 + p)\frac{\partial\varphi_j}{\partial y} + (\rho vw)\frac{\partial\varphi_j}{\partial z}\right]dV;$$

$$I_{kj}^{4ev} = \int\limits_{T_k} \left[(\rho w u) \frac{\partial \varphi_j}{\partial x} + (\rho w v) \frac{\partial \varphi_j}{\partial y} + (\rho w^2 + p) \frac{\partial \varphi_j}{\partial z} \right] dV ;$$

$$I_{kj}^{5ev} = \int\limits_{T_k} \left[(E + p) u \frac{\partial \varphi_j}{\partial x} + (E + p) v \frac{\partial \varphi_j}{\partial y} + (E + p) w \frac{\partial \varphi_j}{\partial z} \right] dV .$$

Все эти интегралы подходят под следующий единый шаблон:

$$I_{kj}^{nev} = \int\limits_{T_k} \left[f_x^n \frac{\partial \varphi_j}{\partial x} + f_y^n \frac{\partial \varphi_j}{\partial y} + f_z^n \frac{\partial \varphi_j}{\partial z} \right] dV ,$$

где $f^n=f^n(\rho, u, v, w, p, E)$, а φ_j – базисные функции.

Интеграл по тетраэдру вычисляется методом Гаусса, для чего нужно найти значение выражения в квадратных скобках в гауссовых точках. На входе в процедуру имеются коэффициенты разложения по базисным функциям величин ρ, ρu, ρv, ρw, E.

Вначале по этим коэффициентам находятся сами величины в гауссовых точках, затем вычисляются значения переменных $u=\rho u/\rho$, $v=\rho v/\rho$, $w=\rho w/\rho$. После этого вычисляется значение переменной e: $e=E/\rho-(u^2+v^2+w^2)/2$. Давление p вычисляется по уравнению состояния $p=p(\rho, e)$.

Поставим целью реализовать объёмное интегрирование один раз, а выражения для вычисления функций f^n передавать как параметры. По этим выражениям в процессе интегрирования будут вычисляться конкретные значения. Механизм, позволяющий передавать выражения для последующих вычислений по ним, называется «шаблоны выражений» (expression templates). Создадим набор классов, реализующих следующую простую грамматику:

«выражение»::=«элементарное выражение» | (*«выражение»*) |
«выражение»+«выражение» | *«выражение»-«выражение»* |
«выражение»«выражение»* | *«выражение»/«выражение»;*
«элементарное выражение»::=ρ|ρu|ρv|ρw|E|u|v|w|e|p;

При вычислении значений каждое выражение будет получать в качестве параметра объект, хранящий все необходимые для вычисления величины:

```
struct integrate_data_type {
     data_type rho, rho_u, rho_v, rho_w, E, u, v, w, e, p;
};
```

Перед тем как показать реализацию выражений, опишем ещё один приём программирования. Как известно, одним из свойств объектно-ориентированного программирования является полиморфизм. Стандартным (и единственным в первоначальной версии языка) средством реализации полиморфизма в языке C++ является механизм виртуальных функций. По ряду причин для наших целей он плох. Причин две. Первая – он не эффективен по быстродействию из-за наличия косвенной адресации. Вторая причина более

существенная. Мы собираемся использовать графические ускорители CUDA. CUDA позволяет копировать объекты из памяти host-системы в память GPU (графического ускорителя), если объекты содержат только простые типы данных (числа, указатели). Но объект, содержащий виртуальные функции, скопирован быть не может (копировать таблицу виртуальных функций бессмысленно, т.к. у GPU своё адресное пространство).

С появлением шаблонов в языке C++ появился другой механизм реализации полиморфизма – т.н. «шаблонный полиморфизм». Он основан на шаблоне проектирования CRTP – Curiously Recurring Template Pattern (см. [17]). При использовании этого шаблона проектирования базовому классу в качестве параметра шаблона передаётся конечный класс. Это позволяет привести имеющуюся ссылку на базовый класс к ссылке на конечный класс и обращаться к любым переменным и методам конечного класса. Мы используем для этой цели единый базовый класс. Вот его реализация:

```
template<class O>
struct math_object_base {
  O& self(){ return static_cast<T&>(*this); }
  const O& self() const {
    return static_cast<const T&>(*this);
  }
};
```

Метод self этого класса как раз и делает приведение объекта базового класса к ссылке на конечный класс. Наш класс «выражение» пронаследован от этого класса:

```
template<class IE>
struct integrate_expression : math_object_base<IE>{};
```

Класс выражения не несёт в себе никакого функционала – это чисто маркерный класс. Его параметр шаблона IE (Integrate Expression) – это конечный класс выражения. Затем определяются арифметические операции с выражениями. Все арифметические операции реализуются с помощью единственного класса:

```
template<class IE1, class OP, class IE2>
struct integrate_expression_binary_op :
  integrate_expression<integrate_expression_binary_op<IE1, OP, IE2> >
{
  integrate_expression_binary_op(
    const integrate_expression<IE1> &ie1_,
    OP op_,
    const integrate_expression<IE2> &ie2_
  ) : ie1(ie1_.self()), op(op_), ie2(ie2_.self()){}

  data_type operator()(const integrate_data_type &id) const {
    return op(ie1(id), ie2(id));
  }
```

```
private:
  const IE1 ie1;
  OP op;
  const IE2 ie2;
};
```

Обратите внимание, что конструктор класса в качестве параметра принимает ссылки на объекты базового класса (ingtegrate_expression), а сохраняет их как копии объектов конечных классов (ie1 и ie2). Арифметическая операция также передаётся как параметр – нет необходимости для разных арифметических операций создавать разные объекты. Базовому классу этот объект передаёт в качестве параметра самого себя. Ну и, наконец, для выражений переопределяются стандартные арифметические операторы:

```
#define DECLARE_IE_OPERATOR(op, op_impl) \
  template<class IE1, class IE2> \
  integrate_expression_binary_op<IE1, op_impl<data_type>, IE2> \
  operator op(const integrate_expression<IE1> &ie1, \
    const integrate_expression<IE2> &ie2){ \
    return integrate_expression_binary_op<IE1, op_impl<data_type>, IE2> \
      (ie1, op_impl<data_type>(), ie2); \
  }

DECLARE_IE_OPERATOR(+, plus)
DECLARE_IE_OPERATOR(-, minus)
DECLARE_IE_OPERATOR(*, multiplies)
DECLARE_IE_OPERATOR(/, divides)

#undef DECLARE_IE_OPERATOR
```

Осталось определить элементарные выражения:

```
#define INTEGRATE_DATA_VAR(name) \
  struct _##name##_t : integrate_expression<_##name##_t> { \
    data_type operator()(const integrate_data_type &id) const { \
      return id.name; \
    } \
  }; \
  extern _##name##_t _##name

INTEGRATE_DATA_VAR(rho);
INTEGRATE_DATA_VAR(rho_u);
INTEGRATE_DATA_VAR(rho_v);
INTEGRATE_DATA_VAR(rho_w);
INTEGRATE_DATA_VAR(E);
INTEGRATE_DATA_VAR(e);
INTEGRATE_DATA_VAR(p);
INTEGRATE_DATA_VAR(u);
INTEGRATE_DATA_VAR(v);
INTEGRATE_DATA_VAR(w);
```

```
#undef INTEGRATE_DATA_VAR
```

Элементарное выражение просто достаёт из структуры integrate_data_type переменную с соответствующим именем.

Оператор объёмного интегрирования написан в соответствие с операторным методом программирования (см. [1]). Вот основной скелет его текста:

```
template<class G, class EOS, class IE1, class IE2, class IE3, class I>
struct volume_integral_operator : ugrid_operator<G,
  volume_integral_operator<G, EOS, IE1, IE2, IE3, I> >
{
  typedef ugrid_operator<G, volume_integral_operator> super;
  volume_integral_operator(const G &ugrid, const EOS &eos_,
    const integrate_expression<IE1> &ie1_,
    const integrate_expression<IE2> &ie2_,
    const integrate_expression<IE3> &ie3_
  ) : super (ugrid), eos(eos_), integrate(),
    ie1(ie1_.self()), ie2(ie2_.self()), ie3(ie3_.self()){}

  polynom_t operator()(unsigned icell, const dense_ugrid_function_proxy<
    G, grid_data_type<polynom_t> > &U) const
  {
    const grid_type::proxy_type &grid = super::get_grid();
    const cell_value_type &cell = grid.m_cells[icell];
    for_each(vector_t gp){ // Цикл по всем Гауссовым точкам
      const vector_t p = (gp - cell.center) / cell.dx;
      const integrate_data_type int_data(eos, U[icell], p);
      const data_type
        val1 = ie1(int_data),
        val2 = ie2(int_data),
        val3 = ie3(int_data);
      … // Реализация оператора
    }
  }
private:
     const EOS eos;
     const IE1 ie1;
     const IE2 ie2;
     const IE3 ie3;
     const I integrate;
};
```

В качестве параметров шаблона этому оператору передаются:

- класс сетки (G);
- класс, реализующий уравнение состояния (EOS);
- три класса для выражений (IE1, IE2, IE3);

- класс, реализующий гауссово интегрирование по тетраэдру (I). Есть несколько реализаций такого интегрирования, отличающиеся количеством гауссовых точек (4, 5 и 15).

Чтобы не определять явно классы IE1, IE2, IE3, используется вспомогательная функция:

```
template<class I, class G, class EOS, class IE1, class IE2, class IE3>
volume_integral_operator<G, EOS, IE1, IE2, IE3, I>
volume_integral(const G &ugrid, const EOS &eos,
  const integrate_expression<IE1> &ie1,
  const integrate_expression<IE2> &ie2,
  const integrate_expression<IE3> &ie3){
    return volume_integral_operator<G, EOS, IE1, IE2, IE3, I>(
      ugrid, eos, ie1, ie2, ie3);
}
```

С её помощью обращение к оператору объёмного интегрирования можно записать следующим образом:

```
I1ev = volume_integral<integrate_tetra_t>(ugrid, eos,
  _rho_u, _rho_v, _rho_w)(U);
I2ev = volume_integral<integrate_tetra_t>(ugrid, eos,
  _rho_u * _u + _p, _rho_u * _v, _rho_u * _w)(U);
I3ev = volume_integral<integrate_tetra_t>(ugrid, eos,
  _rho_v * _u, _rho_v * _v + _p, _rho_v * _w)(U);
I4ev = volume_integral<integrate_tetra_t>(ugrid, eos,
  _rho_w * _u, _rho_w * _v, _rho_w * _w + _p)(U);
I5ev = volume_integral<integrate_tetra_t>(ugrid, eos,
  (_E + _p) * _u, (_E + _p) * _v, (_E + _p) * _w)(U);
```

При обращении к функции volume_integral достаточно указать первый параметр шаблона – класс для гауссова интегрирования по тетраэдру. Остальные параметры шаблона (в том числе типы IE1, IE2, IE3) компилятор определит автоматически, исходя из типов переданных параметров.

4.2. Оператор вычисления потоков через грани

Вычисление потоков через грани можно произвести двумя различными способами. Первый – организовать цикл по всем граням (как внутренним, так и граничным, рассчитать поток через грань и прибавить (или вычесть) этот поток к текущему значению в обеих (или только в одной, если грань граничная) ячейках, которой принадлежит эта грань. Одна из двух ячеек, которой принадлежит данная грань, для этой грани первая, а другая – вторая. К первой ячейке поток прибавляется, а из второй – вычитается. Сам поток рассчитывается соответствующим образом. Второй способ – организовать цикл по ячейкам сетки, для каждой ячейки пройтись по всем её граням, для каждой грани рассчитать поток и прибавить или вычесть этот поток к текущему значению в ячейке в зависимости от того, является данная ячейка для грани

первой или второй. При последовательных вычислениях первый способ, очевидно, эффективнее, т.к. во втором случае поток через каждую грань вычисляется дважды. Однако если мы хотим распараллелить вычисления, то ситуация меняется. В первом случае распараллеливание оказывается невозможным (или сильно затруднённым), т.к. если одновременно будут обрабатываться две грани, принадлежащих одной и той же ячейке, и они захотят прибавить (или вычесть) значение потока к этой ячейке, то возникнет проблема одновременного доступа. Попытаться решить эту проблему можно, но это непростая задача.

Операторный метод программирования изначально нацелен на параллельные вычисления, и в нём цикл всегда организуется по элементам сеточной функции, которой присваивается вычисляемое выражение. Если сеточная функция определена на ячейках сетки (а она может быть определена на любых элементах сетки, например, ячейках, гранях, вершинах, граничных гранях и т.д.), то это означает, что цикл будет вестись по ячейкам и поток через каждую грань будет вычисляться дважды. Бонусом за это будет возможность распараллелить вычисления. Заметим, что распараллелить вычисления можно не только на графических ускорителях, но и на обычных многоядерных процессорах (например, Intel Xeon или Intel Core i7). В этом случае операторная библиотека распараллеливает вычисления с помощью OpenMP. Всё, что нужно для этого, – включить опцию компилятора -openmp.

Итак, вычисления ведутся в цикле по ячейкам сетки. Для каждой ячейки берутся все ей грани. Для каждой грани вычисляется поток через неё и этот поток прибавляется или вычитается к суммарному потоку для данной ячейки. Поток через грань вычисляется как поверхностный интеграл потоковой функции, помноженной на базисную функцию, по этой грани, который вычисляется по гауссовым точкам грани:

$$I_j = \int\limits_{\partial T_k} \Phi \varphi_j dS,$$

где Ф – потоковая функция, φ_j – базисные функции ячейки, для которой считается поток.

Обратим внимание на то, что данный интеграл для одной грани, но разных ячеек – соседей этой грани может дать разные результаты, т.к. в базисную функцию, входящую в этот интеграл, входят координаты центра и характерные размеры ячейки. В частности, если характерные размеры ячеек одинаковые, а центры ячеек расположены симметрично относительно плоскости грани, то значения потоков будут одинаковые.

Потоковая функция может вычисляться по разным алгоритмам. В настоящее время реализованы две потоковые функции – Лакса-Фридрихса и Годунова. В любом случае в потоковую функцию нужно передать значения всех переменных (плотность, импульс, скорость, давление, энергия) в соседних (для грани) ячейках. Для вычисления этих значений нужно для соседних ячеек

грани знать коэффициенты разложения по базисным функциям, координаты центров ячеек и их характерные размеры. Значение консервативной переменной A (это может быть плотность ρ, импульсы ρu, ρv, ρw или энергия E) в точке с координатами (x, y, z) (в частности, в гауссовых точках) вычисляется по формуле:

$$A = \sum_{i=0}^{9} a_i \varphi_i = \sum_{i=0}^{9} a_i \cdot \left(\frac{x - x_c}{\Delta x}\right)^{\alpha_i} \cdot \left(\frac{y - y_c}{\Delta y}\right)^{\beta_i} \cdot \left(\frac{z - z_c}{\Delta z}\right)^{\gamma_i},$$

где a_i – коэффициенты разложения переменной A по базисным функциям.

Если грань внутренняя, то проблем нет, все нужные значения известны. В случае же, если грань граничная, для первой ячейки все нужные данные есть, а для второй – нет. Поэтому для всех граничных граней нужно задать способ вычисления значений консервативных переменных в наружной фиктивной ячейке. Например, можно на этапе инициализации для каждой фиктивной ячейки вычислить её центр и установить связь с одной из внутренних ячеек, откуда впоследствии можно будет брать коэффициенты разложения по базисным функциям и характерные размеры ячейки. Вообще говоря, задание граничных условий – отдельная непростая задача. Цель этой задачи – при вычислениях уметь быстро определять значения консервативных переменных для фиктивных граничных ячеек.

Покажем функцию, вычисляющую одну консервативную переменную по коэффициентам разложения по базисным функциям:

```
data_type calc(const polynom_t &u, const vector_t &p){
  data_type result = 0;
  #define CALC(z, i, unused) \
  { \
    typedef bf_t::bf_type##i type; \
    result += u[i] * \
      pow<type::alpha>(p.value_x()) * \
      pow<type::beta>(p.value_y()) * \
      pow<type::gamma>(p.value_z()); \
  }
  BOOST_PP_REPEAT(BASE_FUNCTIONS_COUNT, CALC, ~)
  #undef CALC
  return result;
}
```

Здесь u – коэффициенты разложения консервативной переменной по базисным функциям, p – векторная величина, $p=((x-x_c)/\Delta x, (y-y_c)/\Delta y, (z-z_c)/\Delta z)$.

И в заключение покажем, как вычисляется поток через внутреннюю грань для всех консервативных переменных одновременно. Для каждой гауссовой точки gp грани вычисляется значение следующей функции, которые затем суммируются:

```
grid_data_type<polynom_t> calc_flow(
  const vector_t &gp, const flow_t &flow,
```

```
  const vector_t &normal, const vector_t &tau1, const vector_t &tau2,
  const vector_t &center, const vector_t &dx,
  const grid_data_type<polynom_t> &data0,
  const vector_t &center0, const vector_t &dx0,
  const grid_data_type<polynom_t> &data1,
  const vector_t &center1, const vector_t &dx1)
{
  const integrate_data_type
    int_data0(data0, (gp - center0) / dx0),
    int_data1(data1, (gp - center1) / dx1);
  // Поток через грань в данной Гауссовой точке.
  grid_data_type<data_type> f = flow(int_data0, int_data1,
    normal, tau1, tau2);
  const vector_t pt = (gp - center) / dx;
  grid_data_type<polynom_t> result;
  #define CALC_INT(z, i, unused) { \
    typedef bf_t::bf_type##i type; \
    grid_data_type<data_type> r = f * \
      pow<type::alpha>(pt.value_x()) * \
      pow<type::beta>(pt.value_y()) * \
      pow<type::gamma>(pt.value_z()); \
    result.ro[i] = r.ro; \
    result.rov.value_x()[i] = r.rov.value_x(); \
    result.rov.value_y()[i] = r.rov.value_y(); \
    result.rov.value_z()[i] = r.rov.value_z(); \
    result.E[i] = r.E; \
  }
  BOOST_PP_REPEAT(BASE_FUNCTIONS_COUNT, CALC_INT, ~)
  #undef CALC_INT
  return result;
}
```

Параметры этой функции следующие:

- gp – координаты Гауссовой точки грани;
- flow – объект, вычисляющий поток (например, Лакса-Фридрихса);
- normal – вектор единичной нормали к грани;
- tau1, tau2 – ортогональные единичные векторы в плоскости грани;
- center, dx – центр и характерные размеры ячейки, для которой считается поток;
- data0, center0, dx0 – коэффициенты разложения консервативных переменных по базисным функциям, центр и характерные размеры первой ячейки;
- data1, center1, dx1 – коэффициенты разложения консервативных переменных по базисным функциям, центр и характерные размеры второй ячейки.

Для вычисления потока через граничную грань параметров data1, center1, dx1 может не быть, а вместо них могут быть какие-то другие параметры. Соответственно, переменная int_data1 должна считаться каким-то другим способом. В реальной программе было три типа граничных условий: константные, циклические и сдвиг по диагонали. В первом случае (константные граничные условия) переменная int_data1 рассчитывалась только с использованием координат гауссовой точки gp, в двух других случаях каждой фиктивной ячейке ставилась в соответствие некоторая внутренняя ячейка, из которой и брались требуемые параметры, кроме центра, который хранился в самой фиктивной ячейке.

4.3. Оператор лимитирования

В настоящей работе для лимитирования был реализован лимитер Кокбурна в двух модификациях – в простейшем виде и с выбором направлений лимитируемых производных согласно градиенту плотности в центре ячейки. Часть параметров лимитера зависит только от геометрии сетки. Такие параметры вычисляются один раз на стадии инициализации задачи. Использование лимитера можно задавать из командной строки. Тип используемого лимитера зашит в программе жёстко. Для изменения типа используемого лимитера требуется перекомпиляция программы.

Параметры лимитера M и ν задаются из командной строки и имеют значения по умолчанию.

Лимитер, в соответствии с операторным методом программирования, реализован в виде оператора.

5. Примеры решения задач

Рассмотрим численное решение трёхмерной системы уравнений газовой динамики (1) разрывным методом Галёркина. В качестве примера возьмём задачу Римана о распаде произвольного разрыва. Область, на которой решается задача, представляет собой куб со стороной 1.

Начальные условия плотности, скорости и давления распределены следующим образом в Задаче 1:

$$(\rho,u,v,w,p)=\begin{cases}1,0,0,0,1.2,\ (0\le x\le 0.5,\ 0\le y\le 1,\ 0\le z\le 1)\\ 2,0,0,0,2.4,\ (0.5<x\le 1,\ 0\le y\le 1,\ 0\le z\le 1)\end{cases} \tag{15}$$

и

$$(\rho,u,v,w,p)=\begin{cases}1,0,0,0,1.2,\ (0<x\le 1,\ x\le y\le 1,\ 0\le z\le 1)\\ 2,0,0,0,2.4,\ (0\le x\le 1,\ 0\le y\le x,\ 0\le z\le 1)\end{cases} \tag{16}$$

в Задаче 2.

Адиабатический показатель γ равен 1,4 в обоих случаях.

В качестве граничных условий в Задаче 1 на границах y=0 и y=1 выберем условия, соответствующие отражению волны от стенок, и периодические граничные условия на остальных границах.

В Задаче 2 на границах x=0, x=1, y=0 и y=1 поставлены граничные условия продолжения области путём снесения значений на границе, а на границах z=0 и z=1 – периодические граничные условия.

При интегрировании по времени в данном случае использована схема Рунге-Кутты 3-го порядка точности.

Далее на рис.2–3 приведены графики распределения плотности в задаче 1 о распаде разрыва (15) в момент времени t=0.2. Расчёты выполнены с использованием линейного N=1 и квадратичного N=2 базисов как без лимитирования, так и с использованием лимитера с параметрами M=60 и α=1.4. Расчёты сделаны на тетраэдральной сетке с шагом 0.01. В качестве более «точного» решения был выбран расчёт на сетке с шагом 0.0001, использующий линейные полиномы в качестве базисных функций с лимитером.

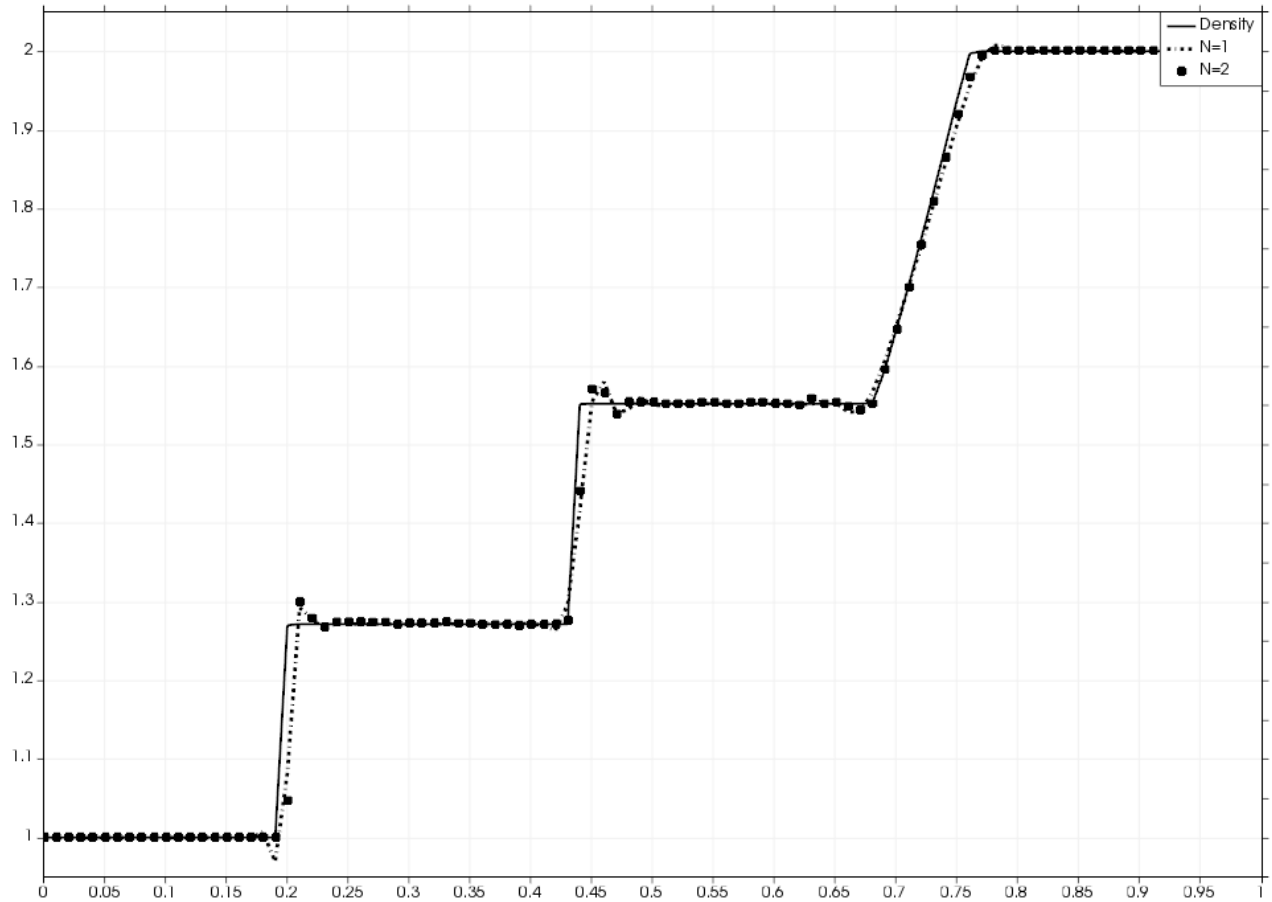

Рис. 2. Профили плотности вдоль y=0.5, на плоскости z=0.5 в Задаче 1 для полиномов степени N=1, N=2 без использования лимитера в сравнении с

«точным» решением, полученным с использованием лимитера с параметрами M=0 и α=1 на сетке с шагом 0.0001

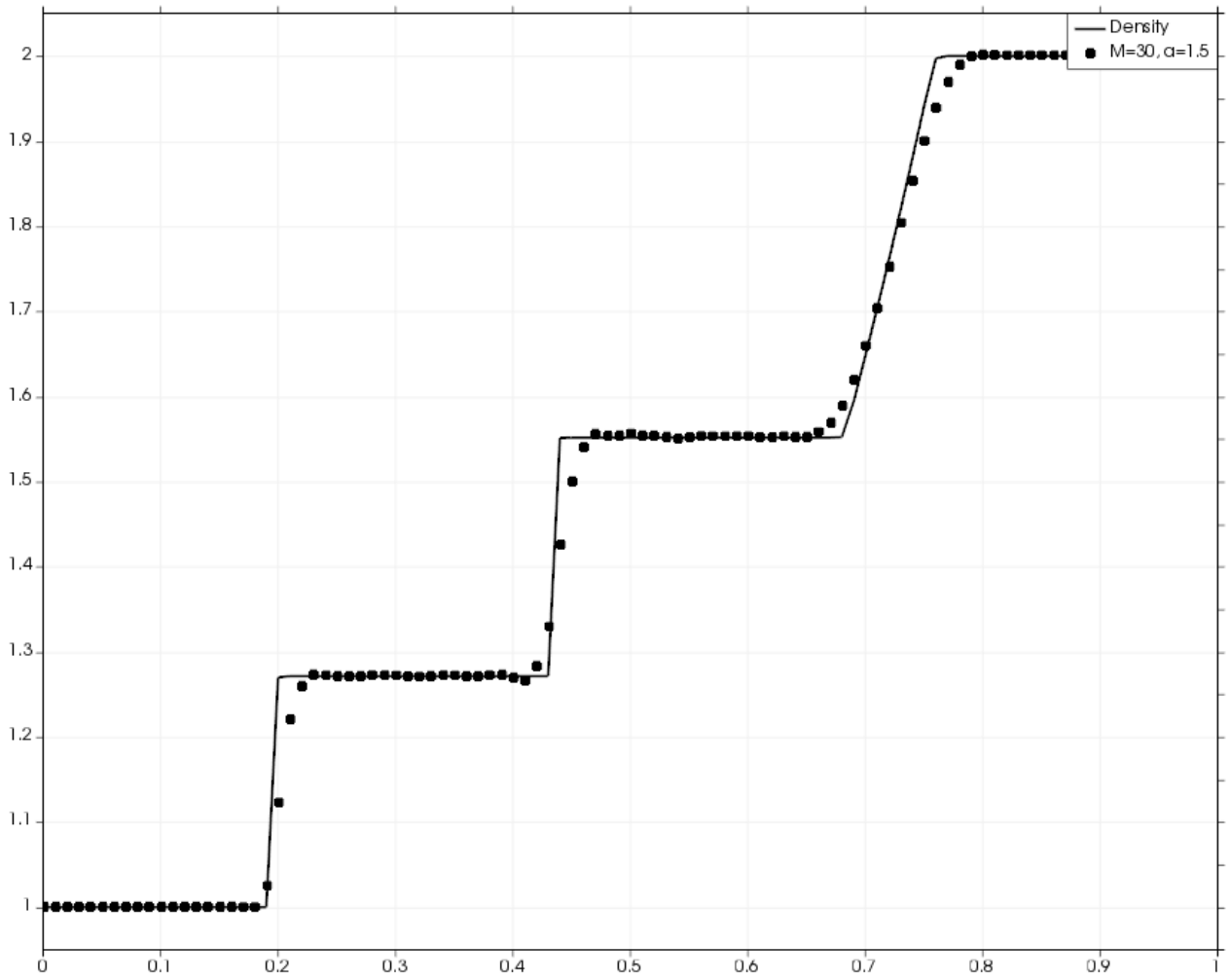

Рис. 3. Профили плотности вдоль y=0.5, на плоскости z=0.5 в Задаче 1 для полиномов степени N=1 с использованием лимитера с параметрами M=30 и α=1.5 в сравнении с «точным» решением

Далее на рис.4–6 приведены графики распределения плотности в задаче 2 о распаде произвольного разрыва (16) в момент времени t=0.2. Расчёты выполнены с использованием линейного N=1 и квадратичного N=2 базисов как без лимитирования, так и с использованием лимитера с параметрами M=0 и α=1.3. Расчёты сделаны на тетраэдральной сетке с шагом 0.01.

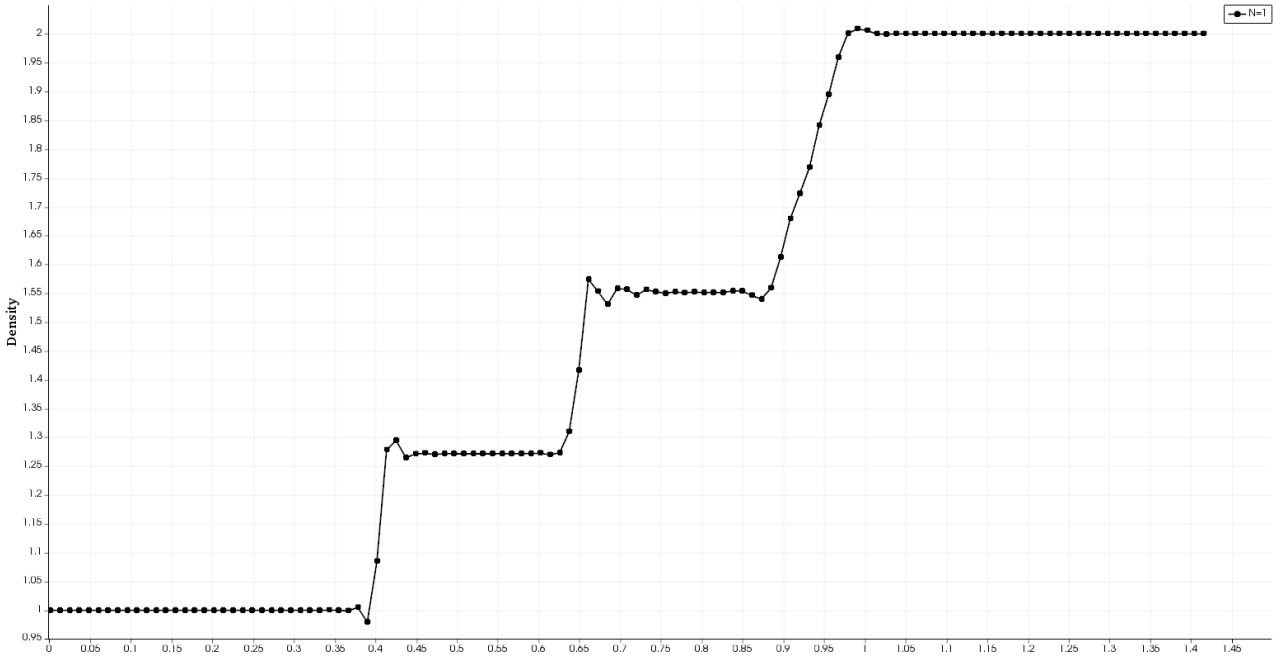

Рис. 4. Профили плотности вдоль y=1-x, на плоскости z=0.5 в Задаче 2 для полиномов степени N=1, без использования лимитера

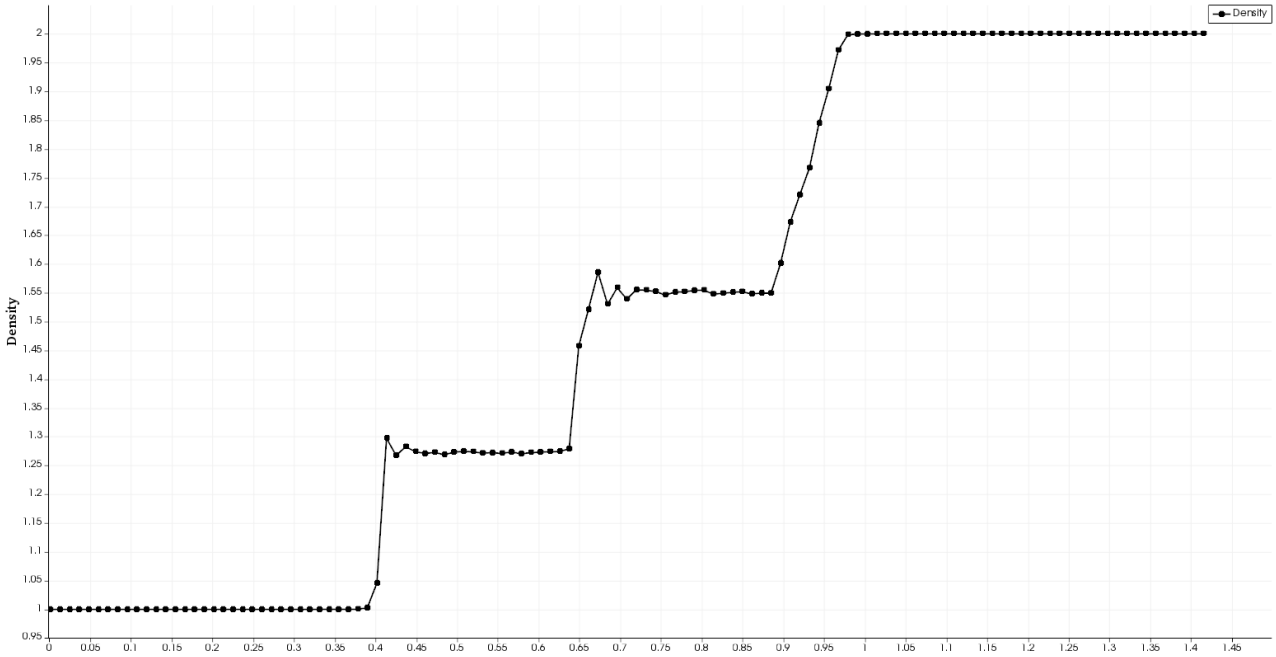

Рис. 5. Профили плотности вдоль y=1-x, на плоскости z=0.5 в Задаче 2 для полиномов степени N=2, без использования лимитера

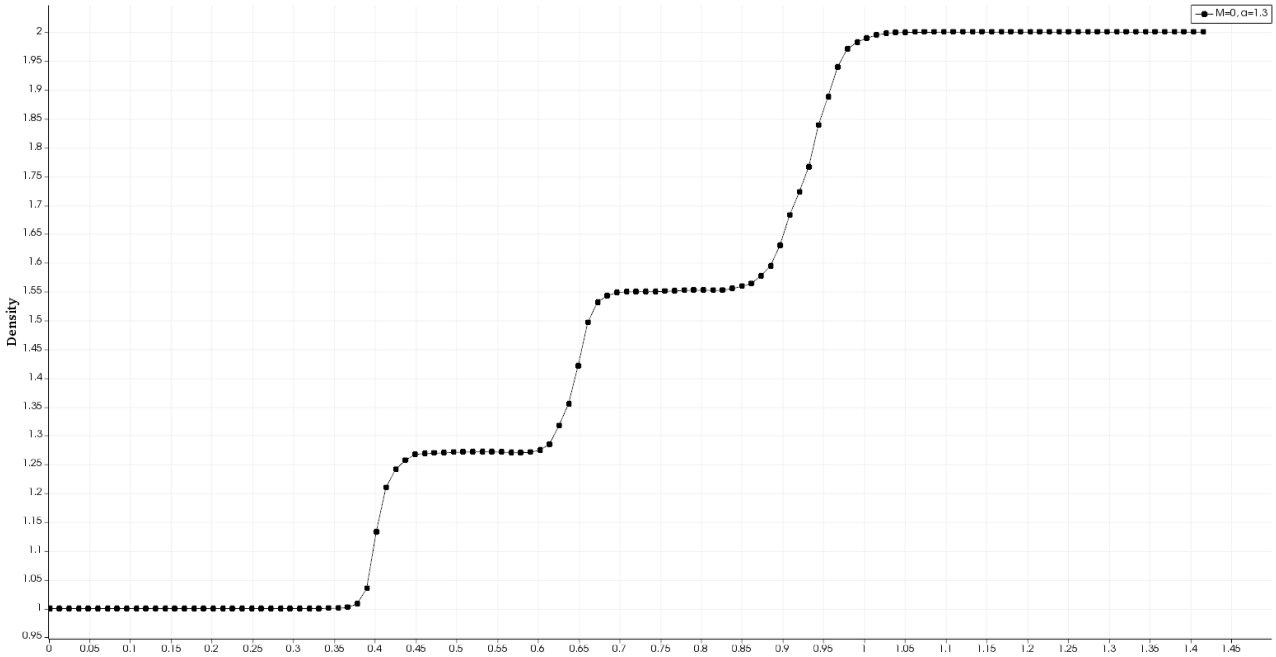

Рис. 6. Профили плотности вдоль *y=1-x*, на плоскости *z=0.5* в Задаче 2 для полиномов степени N=1, с использованием лимитера с параметрами M=0 и α=1.3

Как видно из графиков рис.2,3, численное решение хорошо восстанавливает структуру газодинамического течения (ударную волну, контактный разрыв и волну разряжения) в задаче о распаде произвольного разрыва. Также хорошо видно, что решение и при N=1, и N=2 сильно осциллирует (рис.2), что согласуется с теоретическими исследованиями, а использование лимитера (рис.3) решение сглаживает, причём вне зависимости от типа базисных функций лимитированные решения мало отличаются.

6. Заключение

Операторный метод программирования неоднократно показывал свою эффективность при решении многомерных задач математической физики (см. [19, 20]). Открытым оставался вопрос о том, насколько он будет применим в случае неструктурных сеток. В настоящей работе был выбран для реализации достаточно сложный как логически, так и вычислительно разрывный метод Галёркина.

В соответствие с операторным методом программирования была написана операторная библиотека для трёхмерных неструктурных тетраэдральных сеток. За основу при написании этой библиотеки были взяты операторные библиотеки для трёхмерных регулярных (прямоугольных) сеток и для трёхмерных локально-адаптивных сеток. Перенос библиотеки на новый тип сеток показал, что сделать это можно достаточно формально (соответственно, несложно) с сохранением всех основных идеологических принципов операторного метода. Полученный опыт переноса библиотеки позволяет утверждать, что

операторный метод программирования эффективно реализуем для любых двух и трёхмерных сеток.

Операторный метод программирования нацелен на максимальное распараллеливание программ на общей памяти (технологии OpenMP, CUDA). Для того чтобы воспользоваться этим операторным методом, нужно чётко выделить в алгоритме части, которые могут быть оформлены в виде операторов. В методе Галёркина такими частями являются объёмное интегрирование, расчёт потоков через грани и лимитирование. Чёткая математическая формулировка этих частей позволила их реализовать относительно легко и весьма эффективно.

Об эффективности реализации метода Галёкрина операторным методом говорят следующие цифры. По сравнению с имеющейся альтернативной последовательной (на общей памяти) реализацией метода Галёркина последовательная версия на операторной библиотеке показала ускорение примерно в 6 раз. Применение технологии OpenMP позволило ускорить последовательную программу ещё в 4-9 раз (в зависимости от числа ядер). CUDA-версия показала ускорение по сравнению с последовательной версией в 6-8 раз. При этом надо заметить, что последовательная, OpenMP и CUDA версии программы компилируются из одного исходного текста, но разными компиляторами (например, icpc для последовательной и OpenMP версий и nvcc для CUDA) или с разными опциями компиляторов (например, -openmp для включения OpenMP).

Что касается распараллеливания на кластере с несколькими узлами через MPI, то операторная библиотека с этим напрямую не работает. Реализовывать поддержку MPI нужно на уровне приложения. Это относительно несложно и было успешно сделано для локально-адаптивных сеток (см. [20]). В планах авторов стоит реализация распараллеливания по MPI и для метода Галёркина. Одна их основных проблем при этом – декомпозиция сетки с учётом всех типов граничных условий. Имеющиеся у авторов программы для декомпозиции сеток, в частности, не умеют учитывать циклические граничные условия.

Библиографический список

1. Жуков В.Т., Новикова Н.Д., Феодоритова О.Б. Параллельный многосеточный метод для разностных эллиптических уравнений. Часть I. Основные элементы алгоритма // Препринты ИПМ им. М.В. Келдыша. 2012, № 30. URL: http://library.keldysh.ru/preprint.asp?id=2012-30
2. Краснов М.М. Операторная библиотека для решения трёхмерных сеточных задач математической физики с использованием графических плат с архитектурой CUDA. // Математическое моделирование, 2015, т. 27, № 3, с. 109-120.
3. NVidia CUDA.URL: http://www.nvidia.com/object/cuda_home_new.html

4. Intel Xeon Phi. URL: http://www.intel.com/content/www/us/en/processors/xeon/xeon-phi-detail.html
5. Bernardo Cockburn, An Introduction to the Discontinuous Galerkin Method for Convection - Dominated Problems, Advanced Numerical Approximation of Nonlinear Hyperbolic Equations (Lecture Notes in Mathematics), 1998, V. 1697, pp. 151-268.
6. Галанин М.П., Савенков Е.Б., Токарева С.А. Применение разрывного метода Галеркина для численного решения квазилинейного уравнения переноса, 2005, Препринт 105, ИПМ им. М.В. Келдыша РАН, Москва.
7. Ладонкина М.Е., Неклюдова О.А.,Тишкин В.Ф. Исследование влияния лимитера на порядок точности решения разрывным методом Галеркина // Препринты ИПМ им. М.В.Келдыша. 2012. № 34. 31 с. URL: http://library.keldysh.ru/preprint.asp?id=2012-34
8. Ладонкина М.Е., Неклюдова О.А., Тишкин В.Ф. Исследование влияния лимитера на порядок точности решения разрывным методом Галеркина // Математическое моделирование. 2012. т.24. №12.
9. Ладонкина М.Е., Неклюдова О.А., Тишкин В.Ф. Лимитер повышенного порядка точности для разрывного метода Галеркина на треугольных сетках // Препринты ИПМ им. М.В.Келдыша. 2013. № 53. 26 с. URL: http://library.keldysh.ru/preprint.asp?id=2013-53
10. Krivodonova Lilia, Limiters for high-order discontinuous Galerkin methods, 2007, Journal of Computational Physics, vol. 226, pp. 879-896.
11. Волков А.В. Метод численного исследования обтекания пространственных конфигураций путем решения уравнений Навье-Стокса на основе схем высокого порядка точности. // Диссертация на соискание ученой степени доктора физико-математических наук, М., 2010
12. Русанов В.В. Расчет взаимодействия нестационарных ударных волн с препятствиями. 1961, // Журнал вычислительной математики и математической физики, т. I, № 2, с. 267- 279.
13. Lax P.D. Weak solutions of nonlinear hyperbolic equations and their numerical computation, 1954, Communications on Pure and Applied Mathematics. 7, №1, с. 159 -193.
14. Мысовских И.П. Интерполяционные кубатурные формулы.– М.: Наука, 1981.
15. Li Ben Q. Discontinuous finite elements in fluid dynamics and heat transfer.–Springer, 2006
16. Abrahams David, Gurtovoy Aleksey. C++ Template Metaprogramming. Addison-Wesley. — 2004.
17. Curiously recurring template pattern. URL: http://en.wikipedia.org/wiki/Curiously_recurring_template_pattern
18. Boost C++ Libraries. URL: http://www.boost.org/

19. Жуков В.Т., Краснов М.М., Новикова Н.Д., Феодоритова О.Б. Сравнение эффективности многосеточного метода на современных вычислительных архитектурах. // Программирование, 2015, № 1, с. 21-31.
20. Жуков В.Т., Краснов М.М., Новикова Н.Д., Феодоритова О.Б. Численное решение параболических уравнений на локально-адаптивных сетках чебышевским методом.
// Препринты ИПМ им. М.В.Келдыша. 2015. № 87. 26 с.
URL: http://library.keldysh.ru/preprint.asp?id=2015-87