РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

Факультет физико-математических и естественных наук

Кафедра прикладной информатики и теории вероятностей

УТВЕРЖДАЮ

Заведующий кафедрой прикладной
информатики и теории вероятностей
д.т.н., профессор

_____ К.Е. Самуйлов

«____»_____ 2017 г.

КУРСОВАЯ РАБОТА

на тему

Численное решение произвольных задач теплопроводности для однородного тела с граничными условиями 1 рода с применением MPI и ANTLR4

Дифференциальные уравнения

шифр и наименование учебной дисциплины

Выполнил

Студент группы НП-201

Студенческий билет №: 1032153553

А.В. Лукьянов

«__» _____ 2017 г.

Руководитель

преподаватель кафедры
прикладной математики

Аносова Н.П. И.О. Фамилия

Москва 2017

СОДЕРЖАНИЕ

1. Введение

К дифференциальным уравнениям и их системам сводятся очень многие прикладные задачи естествознания. Однако далеко не всегда возможно решить дифференциальное уравнение или получить из него иные данные аналитически. В таких случаях применяют численные, приближённые методы нахождения решения, причём как правило реализованные в виде алгоритмов для ЭВМ. Однако нахождение решения с приемлемой точностью может представлять трудность в вычислении даже для ЭВМ; по этой причине для нахождения решения сложных задач используются кластеры ЭВМ. Наиболее популярным средством разделения задачи для одновременного выполнения на нескольких машинах является MPI.

Численные методы также обладают тем достоинством что позволяют ставить эксперимент и получить результат близкий к точному в то время как настоящий эксперимент мог бы обойтись слишком дорого, был бы чересчур сложен или же вовсе невозможен. Однако у численных методов есть и недостатки: неточность, неустойчивость некоторых методов, кроме того получить функцию в явном виде обычно намного предпочтительнее. Также при реализации метода на одном из языков программирования бросается в глаза неуниверсальность решений, приводящая к тому что для каждой задачи требуется своя решающая её программа. Яркой иллюстрацией к этому тезису может послужить список задач из последнего параграфа учебного пособия [1].

В данной работе я реализовал численный метод решения нескольких видов задач для одномерного и двумерного уравнения теплопроводности с граничными условиями первого рода, создав относительно универсальное решение. В ходе работы был написан комплекс из двух программ, произведено его тестирование и получены результаты. Также в качестве примера использования программы была решена задача из учебного пособия «Высокопроизводительные вычисления на кластерах» [1].

2. Уравнение теплопроводности

Уравнение теплопроводности является уравнением в частных производных параболического типа. Оно описывает процессы нагревания или охлаждения тел с точностью, достаточной в большинстве случаев, хотя и не идеальной. В m-мерном пространстве R^m данное уравнение имеет следующий вид:

$$\frac{dT}{dt} - \alpha\left(\frac{\partial^2 T}{\partial x_1^2} + \frac{\partial^2 T}{\partial x_2^2} + \ldots + \frac{\partial^2 T}{\partial x_m^2}\right) = f(x,t)$$

Где $x = (x_1, x_2, \ldots, x_m)$.

В постановке задачи для уравнения теплопроводности также задаются:

1. геометрические условия — область тела $D \subset R^m, x \in D$;
2. физические условия — значение коэффициента температуропроводности α, вычисляемого также по формуле $\alpha = \frac{\chi}{\rho \cdot c_p}$, где χ — коэффициент теплопроводности материала, ρ — плотность материала, c_p – его удельная теплоёмкость;
3. начальное условие: $T(x,0) = \varphi(x)$, являющееся уравнением распределения температуры в момент времени t=0;
4. граничные условия, которые бывают следующих видов:
   - 1 рода — описывают температуру на границе тела;
   - 2 рода — задают тепловой поток на границе тела;
   - 3 рода — чрез границу тела происходит обмен с окружающей средой по закону Ньютона.

2.1. Численное решение

Для реализации был выбран явный сеточный метод конечных разностей с граничными условиями первого рода как достаточно простой для распараллеливания. Кроме того, на граничные условия было наложено ограничение: температура на границе тела в каждой точке не меняется со временем, хотя в разных точках может различаться — это необходимо для выноса вычислений температуры границы за пределы основного цикла программы.

Устойчивость вычислительной задачи отражает зависимость полученного решения от небольших изменений входных данных. Если малые изменения входных значений алгоритма ведут к небольшим изменениям в ответе, то такой метод считается устойчивым; если же малые изменения входных данных приводят к значительным изменениям в выдаваемых результатах, то перед нами неустойчивое решение, непригодное для вычислений. То, какие изменения следует считать «малыми» зависит от конкретной задачи.

Основным минусом явного метода является тот факт, что он обладает устойчивостью только при выполнении определённых условий. А именно, требуется, чтобы шаг по времени был строго меньше $\frac{h^2}{\alpha}$, где h – шаг сетки по пространству. В противном случае метод практически бесполезен.

Суть самого метода заключается в следующем. Мы берём куб $G=[x_{10,}x_{1e}]\times[x_{20,}x_{2e}]\times\ldots\times[x_{m0,}x_{me}]$ в R^m, такой что $D\subset G$, этот куб мы будем считать областью исследования. Для каждой размерности куба G_i находим её длину $|G_i|=x_{ie}-x_{i0}$, задаём число точек разбиения по этой размерности $K_i\in N, K_i>0$ и вычисляем шаг сетки: $\Delta x_i=|G_i|/(K_i-1)$. Произведём теперь дискретизацию области исследования, для этого построим равномерную сетку:

$$\omega=\{\omega_{i_1\ldots i_m}=(x_{10}+i_1\Delta x_{1,}x_{20}+i_2\Delta x_{2,}\ldots,x_{m0}+i_m\Delta x_m), i_k\in[0,k-1]\}$$

Далее введём длину шага по времени $\tau \in R^+, 0 < \tau < \frac{1}{\alpha}\prod_{i=1}^{m}\Delta x_i^2$, где верхняя граница — условие устойчивости, обозначим n-тый шаг по времени как $t_n = n \cdot \tau, n \in Z^+$, а также введём обозначение температуры в точке $\omega_{i_1,\dots,i_m}$ на n-тый шаг по времени $T^n_{i_1,\dots,i_m} = T(\omega_{i_1,\dots,i_m}, t_n)$.

Теперь в соответствие обеим частям уравнения поставим алгебраические выражения по явной схеме Эйлера. Производная по времени аппроксимируется выражением:

$$\left(\frac{dT}{dt}\right)^n_{i_1,\dots,i_m} \approx \frac{T^{n+1}_{i_1,\dots,i_m} - T^n_{i_1,\dots,i_m}}{\tau}$$

Вторая производная по каждой переменной x_k расстояния аппроксимируется выражением:

$$\left(\frac{\partial^2 T}{\partial x_k^2}\right)^n_{i_1,\dots,i_m} \approx \frac{T^n_{i_1,\dots,i_k-1,\dots,i_m} - 2T^n_{i_1,\dots,i_m} + T^n_{i_1,\dots,i_k+1,\dots,i_m}}{\Delta x_k^2}$$

Явной эта схема называется по причине того, что из неё возможно выразить температуру в любой внутренней точке следующего временного слоя $T^{n+1}_{i_1,\dots,i_m}$ как явную функцию от значений точек предыдущего временного слоя. Сделаем это:

$$\frac{T^{n+1}_{i_1,\dots,i_m} - T^n_{i_1,\dots,i_m}}{\tau} \approx \alpha \sum_{k=1}^{m} \frac{T^n_{i_1,\dots,i_k-1,\dots,i_m} - 2T^n_{i_1,\dots,i_m} + T^n_{i_1,\dots,i_k+1,\dots,i_m}}{\Delta x_k^2}$$

$$T^{n+1}_{i_1,\dots,i_m} \approx T^n_{i_1,\dots,i_m} + \tau\alpha \sum_{k=1}^{m} \frac{T^n_{i_1,\dots,i_k-1,\dots,i_m} - 2T^n_{i_1,\dots,i_m} + T^n_{i_1,\dots,i_k+1,\dots,i_m}}{\Delta x_k^2}$$

Для случаев m=1 и m=2 выражение будет иметь вид, соответственно:

$$T^{n+1}_i \approx T^n_i + \frac{\tau\alpha}{\Delta x^2}\left(T^n_{i-1} - 2T^n_i + T^n_{i+1}\right)$$

$$T^{n+1}_{i_1,\dots,i_m} \approx T^n_{i_1,\dots,i_m} + \tau\alpha\left(\frac{T^n_{i-1,j} - 2T^n_{i,j} + T^n_{i+1,j}}{\Delta x^2} + \frac{T^n_{i,j-1} - 2T^n_{i,j} + T^n_{i,j+1}}{\Delta y^2}\right) =$$
$$= T^n_{i_1,\dots,i_m} + \frac{\tau\alpha}{\Delta x^2}\left(T^n_{i-1,j} - 2T^n_{i,j} + T^n_{i+1,j}\right) + \frac{\tau\alpha}{\Delta y^2}\left(T^n_{i,j-1} - 2T^n_{i,j} + T^n_{i,j+1}\right)$$

Итак, получили окончательную формулу для вычисления нового

временного слоя. Сам алгоритм вычислений прост: зная температуру в каждой точке временного слоя t=0 из начального условия, по формуле выше находим значения для каждой точки следующего временного слоя и т. д. до тех пор, пока не достигнем требуемого момента времени.

Температуру в точках по границе области исследования этим способом, очевидно, вычислять нельзя (не существует точки $T^{n}_{i_1=-1,i_{2...},i_m}$ для того чтобы найти по ней значение в точке $T^{n+1}_{i_1=0,i_{2...},i_m}$, например), поэтому к области D обычно добавляется искусственная граница в одну точку, значения которой описываются иным способом, как правило, граничными условиями. Точно также значения всех внутренних точек области G, не принадлежащих телу, не пересчитываются по разностной схеме, а задаются теми же граничными условиями.

Описанная выше схема для одномерного случая называется трёхточечной, т. к. значение в точке вычисляется на основе трёх других известных точек. Для n-мерного случая уравнение будет (n*2+1)-точечным.

2.2. Распараллеливание вычислений

Полученное решение хорошо тем, что значение в каждой точке не зависит от соседних точек того же временного слоя, что идеально для разделения их вычисления между несколькими потоками, процессами или даже машинами.

Разделить можно по-разному. Я выбрал разделение по столбцам для одномерного случая и по строкам для двумерного случая на некоторое количество одинаковых частей, задаваемое пользователем. Разделение работы требует небольших изменений в алгоритме: теперь каждый процессор (MPI-процессор, т. к. для распараллеливания используется MPI) может иметь левого и правого соседей, которые обрабатывают соответственно предыдущую и следующую части матрицы. Однако каждому процессору для вычисления крайних значений – верхней и нижней строк матрицы для двумерной задачи

или первого и последнего элементов массива для одномерной — требуются значения, находящиеся в соседнем процессоре, поэтому процессоры обмениваются соответствующими блоками данных с соседями после вычислений очередного временного слоя. Для хранения этих блоков используются дополнительные, «фиктивные» строки сверху и снизу матрицы (двумерная задача) и в начале и конце массива (одномерная задача), что позволяет не обрабатывать вычисление значений, использующих эти данные, как отдельный случай.

Для разделения в двумерном случае выбраны именно строки, т.к. MPI может обменивать через сообщения только сплошные области данных. Было сделано так, чтобы передаваемые данные были сплошным блоком памяти (строками, а не столбцами), что избавляет от циклов копирования при каждом обмене данными.

Описанное выше разделение работы между процессорами называется одномерным, т. к. матрица разделяется только по одному измерению. Можно разделить по обоим и обмениваться данными с восемью процессорами: с верхним, нижним, правым и левым по строке/столбцу (кроме угловых элементов), а также четырьмя угловыми процессорами по одной ячейке данных (находящихся в углах матриц). Такая схема декомпозиции называется двумерной. Если посчитать суммарный объём обмениваемых данных, то выяснится что при числе процессоров более четырёх программа, использующая двумерную декомпозицию, обменивается меньшим количеством данных и, соответственно, такая схема могла бы показать лучшие результаты на большом кластере машин, однако она показывает худшие результаты на одной машине, особенно при сравнительно небольших объёмах обрабатываемых данных. Соответственно, была выбрана одномерная декомпозиция MPI как более подходящая для не самой быстрой машины автора работы.

2.3. Другие способы численного решения

Также существует неявный метод, вычисляемый на основе другого шаблона с тем же числом точек:

$$T^{n+1}_{i_1,\dots,i_m} \approx T^{n}_{i_1,\dots,i_m} + \tau\alpha \sum_{k=1}^{m} \frac{T^{n+1}_{i_1,\dots,i_k-1,\dots,i_m} - 2\,T^{n+1}_{i_1,\dots,i_m} + T^{n+1}_{i_1,\dots,i_k+1,\dots,i_m}}{\Delta x_k^2}$$

Уравнение в итоге приводится к системе уравнений с симметрической трёхдиагональной (пяти при m=2, семидиагональной при m=3 и т.д.) матрицей и решается на каждой итерации по времени. Я счёл что этот метод хуже распараллеливается из-за того что единственная возможная для разделения операция это решение СЛАУ. Для параллельного решения СЛАУ рекомендуется использовать метод сопряжённых градиентов, но я решил остановиться на более простом явном методе.

Существуют также явно-неявные (смешанные) разностные схемы и другие решения, как сеточные, так и нет.

3. Описание разработанного программного обеспечения

Для курсовой работы был написан комплекс из двух программ: пользовательского интерфейса и вычислительной части на языке программирования C++. Программа-интерфейс требует для корректной работы установленного пакета «Visual C++ Redistributable for Visual Studio 2015». Вычислительная часть — установленной реализации MPI, по умолчанию Microsoft MPI.

В решении были использованы следующий библиотеки:

1. MPI, «Message Passing Interface»
2. MFC, «Microsoft Foundation Classes»
3. ANTLR, «Another Tool for Language Recognition»
4. STL, «Standard Template Library» языка C++

3.1. MPI

Это широко распространённый API для разработки параллельных программ под системы с распределённой памятью. Основным плюсом MPI является возможность разделения задачи между целым кластером машин, в отличие от OpenMP, позволяющего задействовать ресурсы лишь одной машины. Однако эта же особенность является его недостатком: для каждой нити выполнения даже на одной машине MPI создаёт отдельный процесс, а обмен данными между нитями (называемыми MPI-процессорами) происходит только посредством сообщений, что делает его слишком громоздким для использования в системе с общей памятью (т. е. для одной машины), для которых более предпочтительны другие средства организации параллельных вычислений, например OpenMP. Существует несколько реализаций MPI, как коммерческих, так и нет.

MPI как средство распараллеливания был выбран в связи с требованиями,

указанными в задаче, а также потому что автор данной работы хотел научиться работе с ним. Кроме того, вычислительные мощности одной машины ограничены, а наиболее простой способ увеличения производительности это просто увеличить число машин, следовательно, умение работать с MPI является актуальным в прикладных задачах. Однако тестирование производительности на кластере машин не производилось, поскольку кластера для этого автор курсовой работы не имеет.

3.2. MFC

MFC является библиотекой от Microsoft для разработки под ОС Windows, в первую очередь пользовательских интерфейсов. Разработка на MFC требует больших трудозатрат чем разработка с применением Windows Forms или VCL, однако созданное в итоге приложение меньше и работает быстрее. С использованием данной библиотеки разработаны программы пакета Microsoft Office.

3.3. ANTLR4 и язык выражений

Для того чтобы иметь возможность задавать в условии задачи произвольную форму тела, произвольную начальную температуру тела в зависимости от точки и, аналогично, температуру окружающей среды, возникла необходимость в разработке языка, позволяющего выразить это и передать в программу, а также в необходимости парсера для его распознавания и сборки соответствующих внутренних структур программы в соответствии с заданным пользователем выражением.

Разработка такого парсера собственными силами «с нуля» прямо в коде — задача долгая, кропотливая и плохо расширяемая с изменением языка. Кроме того, автор курсовой работы уже выполнял её раньше в учебных целях и делать снова счёл нецелесообразным. Вместо этого было решено использовать

генератор парсеров.

ANTLR, «Another Tool for Language Recognition» — генератор парсеров формальных языков на основе контекстно-свободных грамматик, описывающих эти языки.

Почему именно ANTLR:

1. Это open-source продукт, распространяемый по лицензии BSD. Он бесплатен.

2. ANTLR генерирует быстрый, качественный код, прошёл проверку временем (первая публичная сборка в 1992 году) и до сих пор активно поддерживается разработчиком. Это серьёзный продукт, используемый как в коммерческих продуктах (например, компаниями Oracle и JetBrains), так и в продуктах, разрабатываемых Open-source сообществом (NetBeans C++).

3. Удобство разработки грамматики: простой и понятный синтаксис, большое количество учебных материалов в интернете, существование своей IDE на основе NetBeans, значительно ускоряющей процесс создания, тестирования грамматики и поиска в ней ошибок. Например, IDE позволяет визуализировать синтаксическое дерево, построенное по конкретной строке.

Проблема передачи уравнений в программу была решена с использованием синтезированных атрибутов: при разборе переданной строки парсер компонует структуру из объектов внутренних классов программы, описывающую переданное выражение. UML-схема этих классов приведена ниже (Иллюстрация 1). Полное описание грамматики языка приведено в приложении.

Созданный язык поддерживает основные арифметические операции (+, -, *, /), сравнения (<, >, =, <=, >=, !=), скобки «(« и «)», логические выражения (or, and, not), ветвления (if-else-if-else), переменные, а также встроенные функции, точный состав которых определяется программной частью независимо от грамматики языка и может быть легко расширен при необходимости. В данный

момент реализованы следующие функции: sin(x), cos(x), tg(x), ctg(x), ln(x), lg(x), log(основание, выражение), exp(степень), abs(x), pow(x, степень), все функции допускают в качестве своих параметров любые выражения вещественного типа. Между лексемами допускаются пробельные символы аналогично правилам C++. Язык регистронезависим.

Следует также заметить что вычисление передаваемых в программу формул не используется в вычислительном цикле, вместо этого для каждой точки вычислительной сетки определяется признак принадлежности телу один раз до запуска основного цикла. Следовательно, связанные со введением языка издержки по времени нахождения решения практически отсутствуют и составляют O(1).

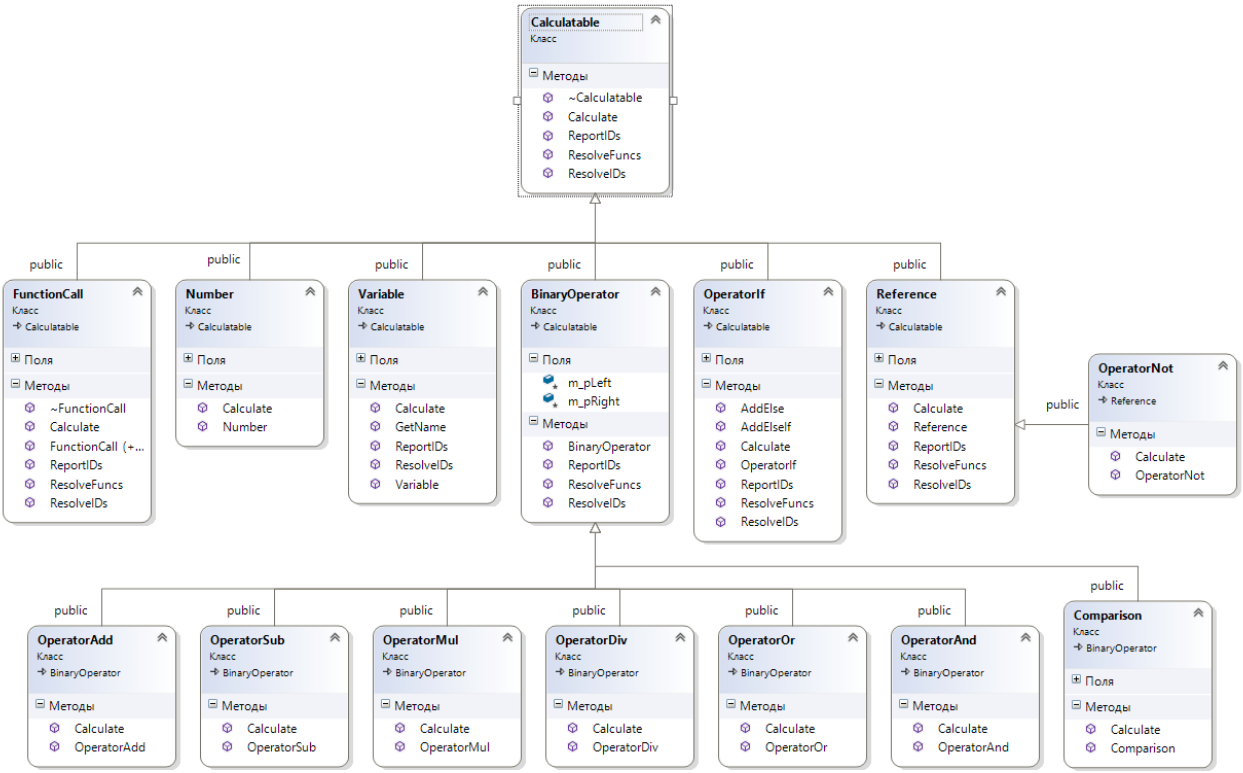


Иллюстрация 1: UML-схема иерархии классов, из которых парсер производит сборку выражений при разборе строки

3.4. Описание интерфейса и возможностей программы

Вычислительная часть выполнена как консольная MPI-программа, в то время как программа-интерфейс является приложением для Windows. Ниже приведён обзор его интерфейса.

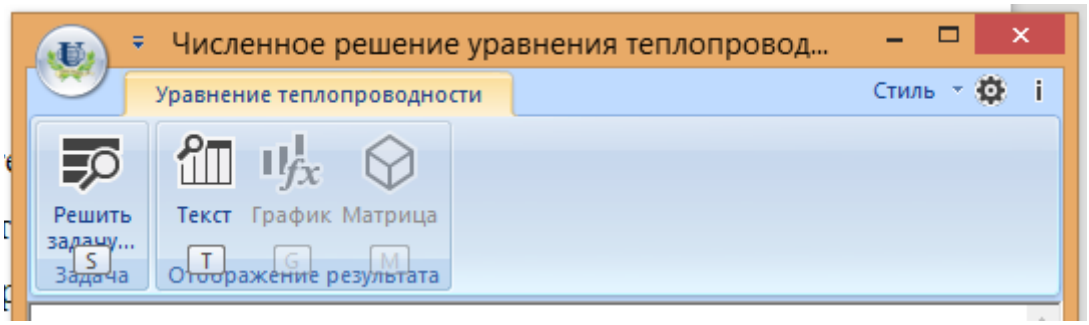


Иллюстрация 2: Внешний вид программы-интерфейса

S – открытие диалогового окна ввода условий задачи, оно будет разобрано ниже.

T, G, M – виды визуализации данных после получения ответа. Текстовое представление доступно для любого типа задачи. Для одномерной задачи полного распределения температуры по телу, а также для задачи изменения температуры в точке с течением времени возможно построение графика. Для полного распределения температуры в двумерной задаче доступно матричное представление данных.

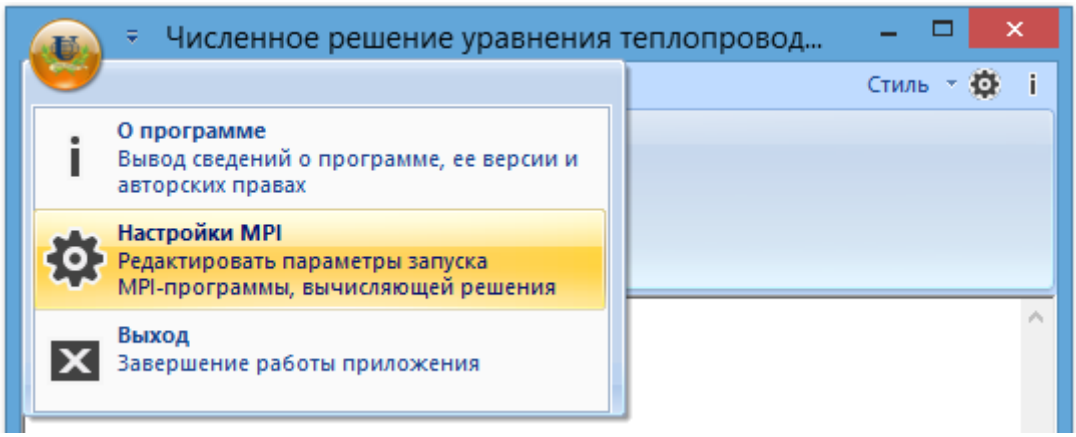


Иллюстрация 3: Внешний вид программы-интерфейса

Ввод условий задачи выполнен в виде пошагового мастера, назначение каждого шага поясняется в самом диалоговом окне и комментариев не требует. Рассмотрим все шаги на примере одномерной краевой задачи из учебника [2] «Разностные методы решения задач теплопроводности: учебное пособие»: происходит теплопередача через плоскую бесконечную пластину, при этом на границах пластины поддерживается постоянная температура (может быть различной на разных концах). Дана начальная температура пластины. Внутри пластины источники тепловыделения отсутствуют. Материал из которого изготовлена пластина — сталь. Математическая формулировка задачи:

$$\rho c \frac{\partial T}{\partial t} = \lambda \frac{\partial^2 T}{\partial x^2}, \quad 0 < x < L. \qquad (3)$$

Начальные и граничные условия запишутся следующим образом:

$$\begin{aligned} &t = 0: \; T = T_0, \; 0 \le x \le L; \\ &x = 0: \; T = T_л, t > 0; \\ &x = L: \; T = T_п, t > 0. \end{aligned} \qquad (4)$$

Далее подставляем заданные значения: начальная температура — 20°C, левый край имеет температуру 300°C, правый — 100°C. Найти распределение температуры в пластине через 60 секунд.

Соответствующий задаче ввод данных в программу:

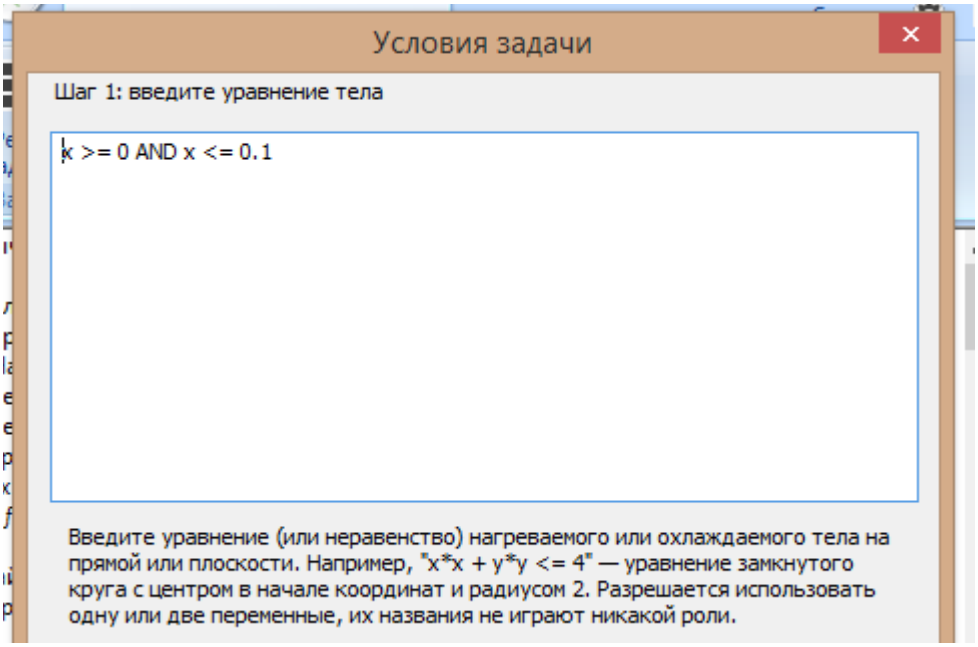


Иллюстрация 4: Первый шаг мастера ввода условий задачи

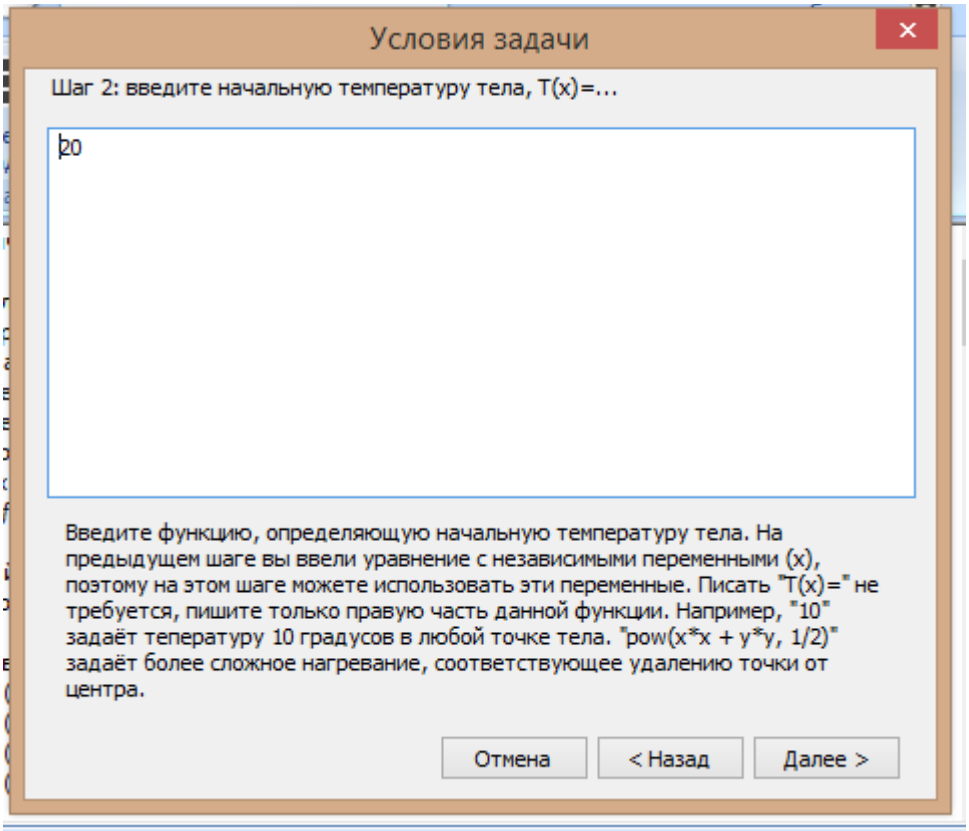


Иллюстрация 5: Второй шаг мастера ввода условий задачи

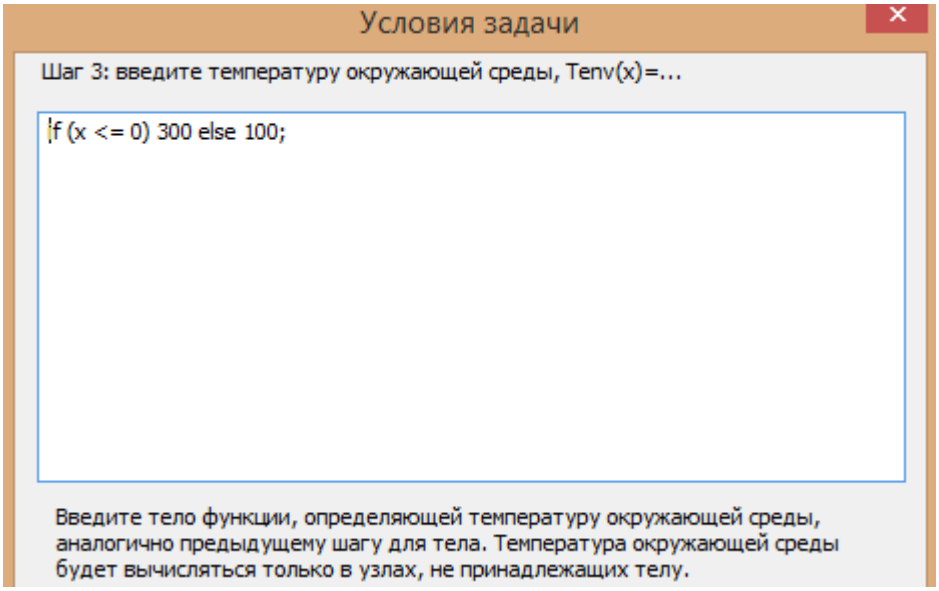


Иллюстрация 6: Третий шаг мастера ввода условий задачи

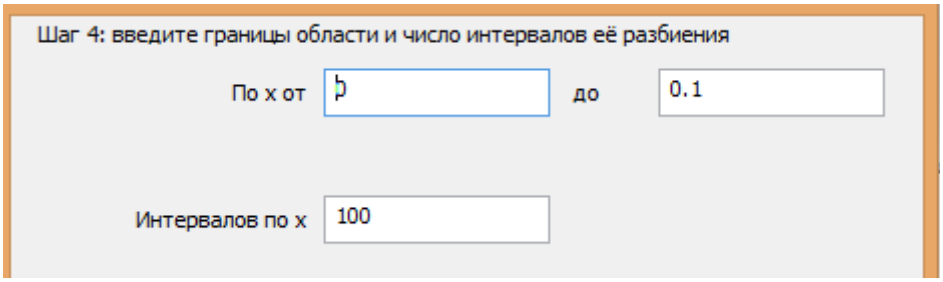


Иллюстрация 7: Четвёртый шаг мастера ввода условий задачи

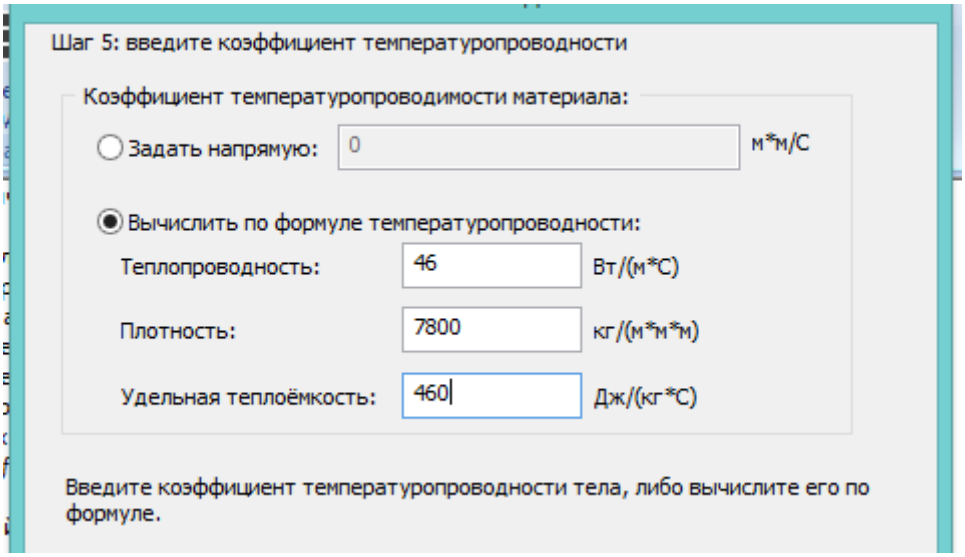


Иллюстрация 8: Пятый шаг мастера ввода условий задачи

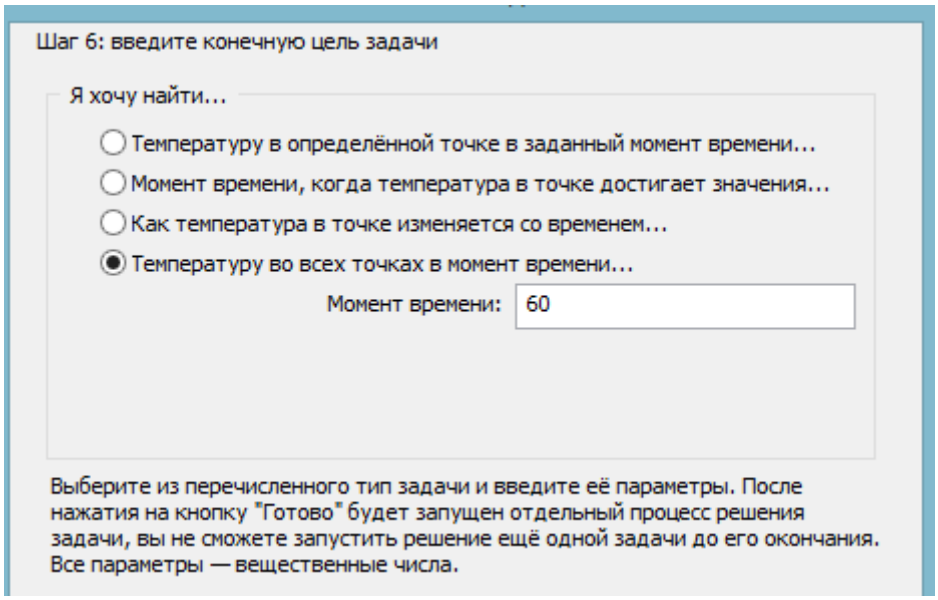
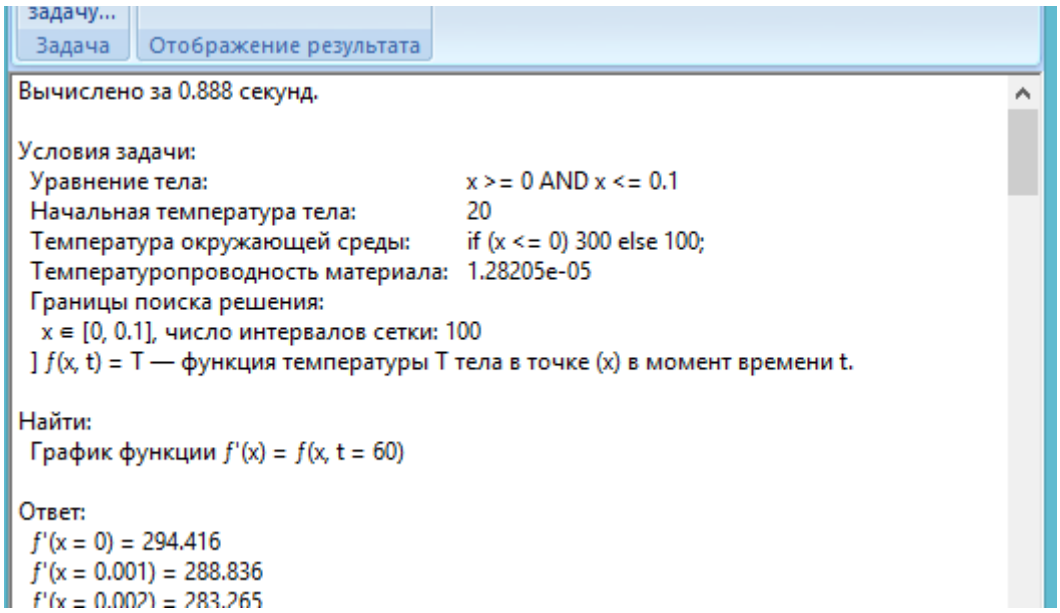
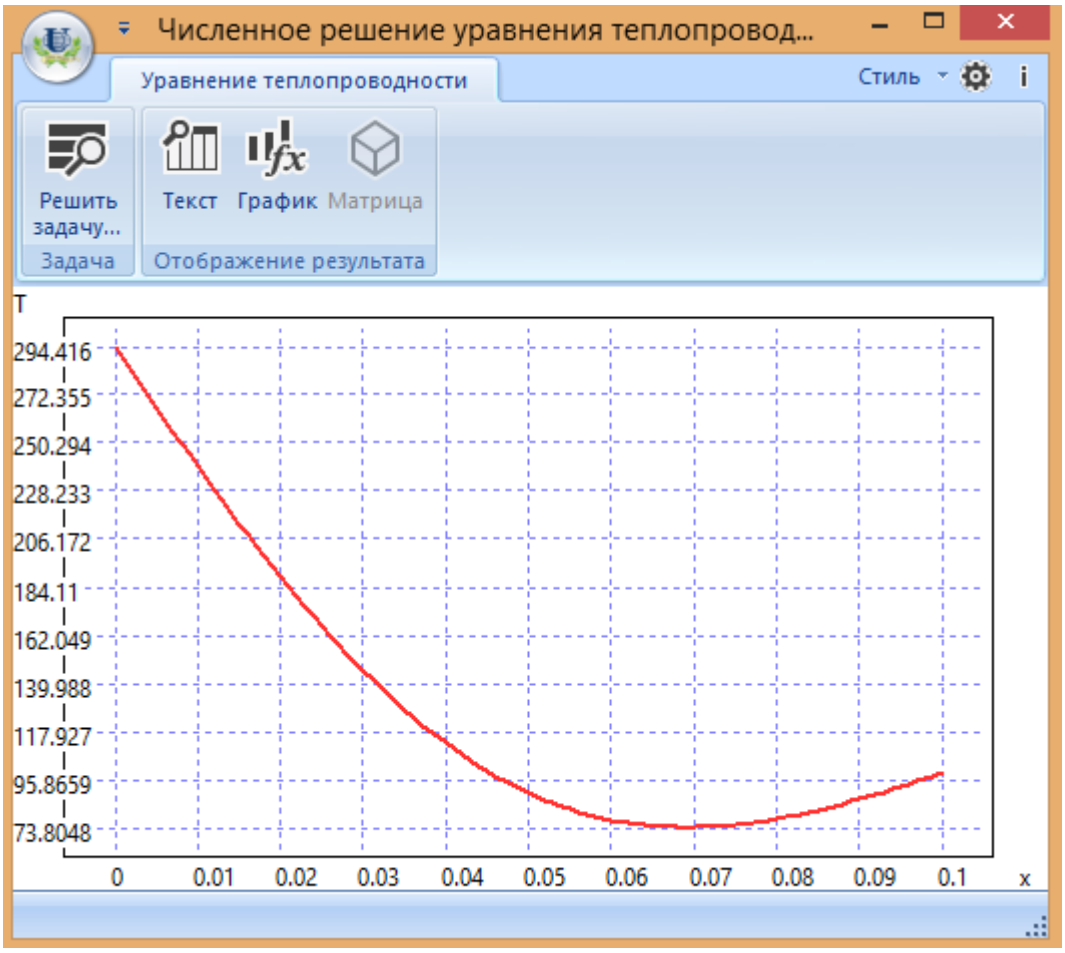


Иллюстрация 9: Шестой шаг мастера ввода условий задачи



На последнем шаге выбирается тип задачи, всего реализовано 4 вида задач как для одномерного, так и для двумерного случая. После нажатия кнопки «Готово» будет запущена вторая программа, вычисляющая результат, и как только она завершит выполнение, интерфейс примет вид как на изображении 10. По умолчанию, представление результата текстовое, но для данного типа задачи также доступно отображение результата в виде графика:



Как можно заметить, у авторов учебного пособия получился в точности тот же результат, что доказывает правильность работы программы:

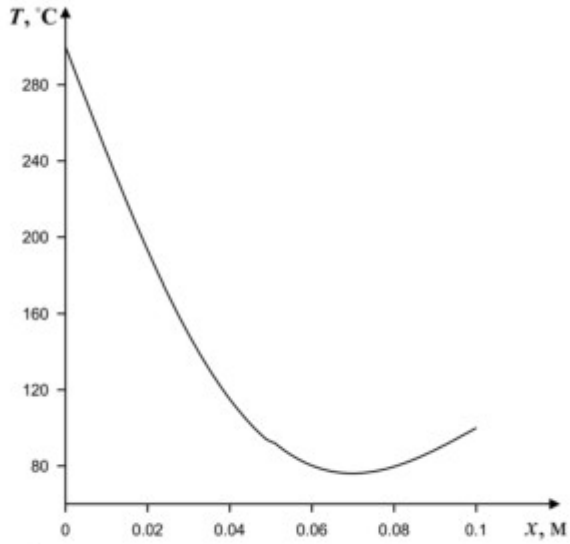


Рис. 4. Распределение температуры по толщине пластины в момент времени t = 60 с

Иллюстрация 12: Распределение, приведённое для задачи о нагревании пластины в учебном пособии [2]

Для двумерного случая этого же типа задачи программа может отображать результат в виде матрицы, например:

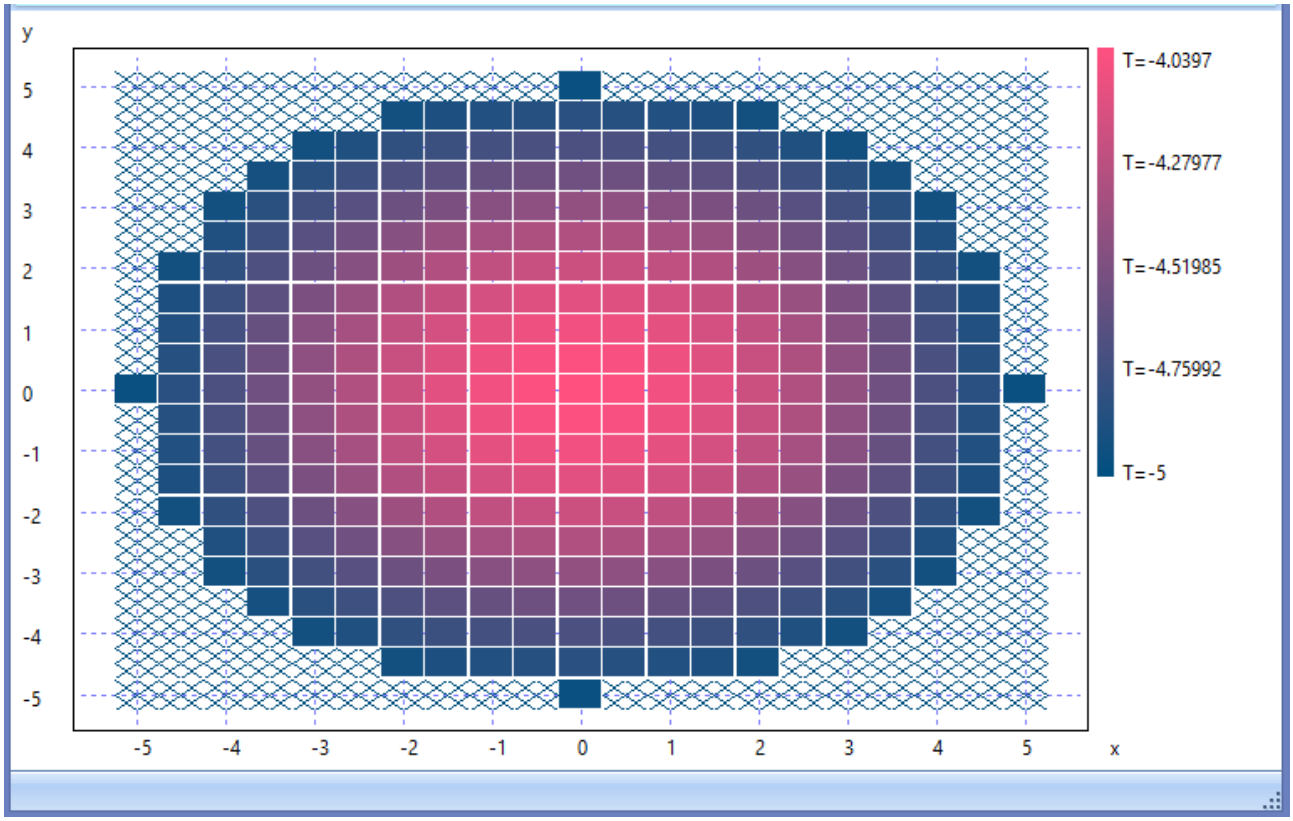


Иллюстрация 13: Матричное представление данных

4. Решение задачи

Для примера решим следующую задачу из учебника «Высокопроизводительные вычисления на кластерах» [1]:

«Определить конечное распределение температуры в обручальном кольце с радиусами R=2,0 см и r=1,8 см, если температура в помещении 20°C, а температуру тела человека можно принять равной 36,6°C. Разработать численный метод решения задачи. Написать параллельную программу и провести ее тестирование для различных размеров вычислительной сетки и числа используемых процессов. $\alpha = 126 \cdot 10\text{-}6$ м^2/с.»

Пусть F(x,y,z,t) – искомая функция распределения температуры в момент времени t. В задаче ничего не сказано про высоту кольца, поэтому будем считать что распределение температуры в кольце не зависит от высоты z и одинаково на всех уровнях, т. е. F(x,y,z1,t) = F(x,y,z2,t), поэтому мы будем искать функцию T(x,y,t)=F(x,y,z,t). Начальная температура на конечное распределение роли не играет и может быть любой. Для определённости, мы возьмём T_0=30°C. Также будем считать что внутренний/внешний края кольца уже имеют температуру тела/воздуха соответственно и эта температура в них постоянно поддерживается окружающей средой, поэтому коэффициентом теплоотдачи тело-кольцо и кольцо-воздух можно пренебречь (кстати, они и не указаны в задаче) и считать что процесс происходит целиком на внутренней части кольца, т.е. будем считать что задача имеет граничные условия 1 рода (а не третьего, как в случае с теплоотдачей).

Итак, полная математическая постановка для исходной задачи имеет следующий вид:

$$\left\{\begin{array}{c} D=\{(x,y):0.018^2\le x^2+y^2\le 0.02^2\} \\ I=\{(x,y):x^2+y^2<0.018^2\} \\ E=\{(x,y):0.02^2<x^2+y^2\} \\ pc\frac{\partial T}{\partial t}=\lambda(\frac{\partial^2 T}{\partial x^2}+\frac{\partial^2 T}{\partial y^2}),(x,y)\in D \\ t=0:(x,y)\in D,T=30 \\ t\ge 0:(x,y)\in I,T=36.6 \\ t\ge 0:(x,y)\in E,T=20 \\ T_{end}=\{t:T(x,y,t)=T(x,y,t+e),\forall\ e>0,\forall\ x,\forall\ y\} \\ \mathring{T}(x,y)=T(x,y,t_{end}),t_{end}\in T_{end} \\ \mathring{T}(x,y)=? \end{array}\right.$$

Теперь заметим что при указанной форме тела и заданных граничных условиях температура во всех точках одного радиуса r=r0 будет одинакова, что даёт возможность упростить задачу, сведя её к одномерному уравнению теплопроводности. После чего можно найти её решение, выразить через него решение исходной задачи и проверить правильность решения проведением эксперимента для исходной задачи. Сведение задачи к одномерному случаю выгодно с точки зрения производительности, т. к. численное решение задачи для двухмерного кольца займёт значительно большее время.

Итак, обозначим ширину кольца как w = R – r. Проведём из центра кольца луч радиуса R (в любую сторону), он пересекает кольцо на отрезке [r, R]=[r, r+w]. Сделаем перенос начала координат в точку r и получим отрезок P=[0, w]. Пусть $T_1(r, t)$ — функция распределения температуры на отрезке P. Также нам известно что температура в точках исходного двухмерного тела, находящихся на одном удалении от центра, одинакова, а значит может быть выражена через температуру на отрезке следующим образом:

$$T(x,y,t)=T_1(\sqrt{x^2+y^2}-r,t) \qquad (*)$$

При этом, по условию задачи, один край этого отрезка имеет температуру 36,6°C, а другой — 20°C. Теперь найдём распределение температуры на отрезке P.

Математическая формулировка для новой задачи выглядит так:

$$\left\{\begin{array}{c} pc\frac{\partial T_1}{\partial t}=\lambda\frac{\partial^2 T_1}{\partial x^2}, x\in[0,0.002] \\ t=0: T_1=30, x\in(0,0.002) \\ x=0, T_1=36.6, t\geq 0 \\ x=0.002, T_1=20, t\geq 0 \\ T_{end}=\{t: T_1(r,t)=T_1(r,t+e), \forall\ e>0, \forall\ r\} \\ \mathring{T}_1(r)=T_1(r,t_{end}), t_{end}\in T_{end} \\ \mathring{T}_1(r)=? \end{array}\right\}$$

Введём соответствующие значения в программу и вычислим полное распределение для t=0.0001, t=10 и t = 100. Для t=0.0001 получим:

Условия задачи:
Уравнение тела: x >= 0 AND x <= 0.002
Начальная температура тела: 30
Температура окружающей среды: if (x <= 0) 36.6 else 20;
Температуропроводность материала: 0.000126
Границы поиска решения:
x ∈ [0, 0.002], число интервалов сетки: 100
] $f(x, t)$ = T — функция температуры T тела в точке (x) в момент времени t.

Найти:
График функции $f'(x) = f(x, t = 0.0001)$

Иллюстрация 14: Условия производной одномерной задачи для t=0.0001

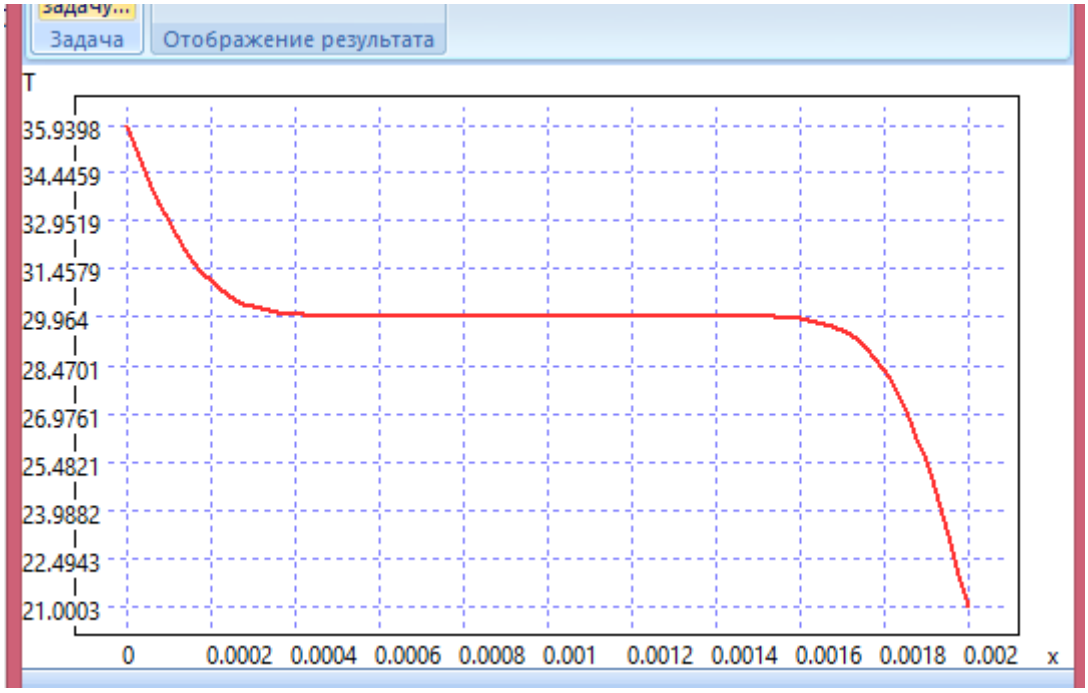


Иллюстрация 15: Результат для t=0.0001 в виде графика

Для t=10:

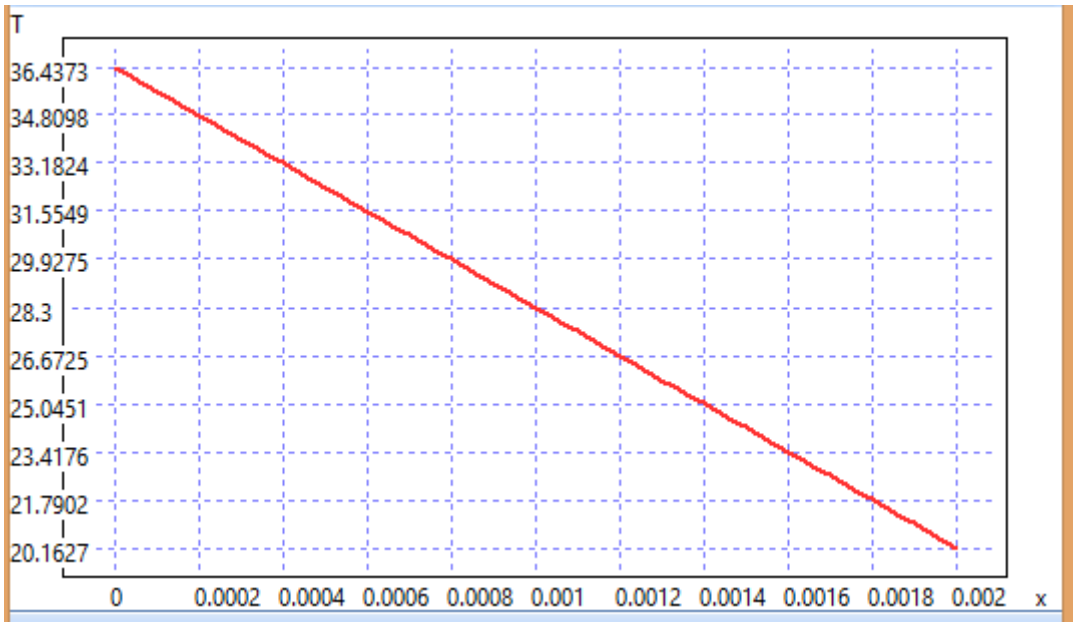


Иллюстрация 16: Результат для t=10 в виде графика

Для t=100 были получены те же значения с точностью до десятитысячных. Взяв более мелкую сетку, получается в целом тот же результат, но температура на границах отрезка становится ближе к заданным в граничных условиях, для сетки со 150 интервалами верхняя и нижняя границы, например, составляют 36.4908 и 20.1092 соответственно, а при 200 — 36.5178 и 20.0822. Логично предположить, что при увеличении сетки график и дальше будет приближаться именно к граничным значениям, причём бесконечно близко. Заметим теперь что график полученного распределения имеет вид прямой, а значит сама функция распределения задаётся уравнением y=kx+b. Подставив граничные точки в уравнение, найдём его коэффициенты:

$$\begin{Bmatrix} 36{,}6=0\,k+b \\ 20=0.002k+b \end{Bmatrix}$$
$$\begin{Bmatrix} b=36.6 \\ k=-8300 \end{Bmatrix}$$

Теперь, подставив ответ в формулу (*), получим:

$$F(x,y,z,t)=T(x,y,t)=36.6-8300\left(\sqrt{x^2+y^2}-r\right),t\in T_{end} \qquad (1)$$

Убедимся, что найденное уравнение (1) действительно является решением, причём сделаем это снова при помощи данной программы. Для этого выберем в качестве начального распределения температуры тела функцию (1) и понаблюдаем за изменением температуры с течением времени. Чтобы иметь численную характеристику изменений температуры за некоторый промежуток времени t_1, введём обозначения:

$$\delta(x,y,t_1)=|T(x,y,0)-T(x,y,t_1)|$$
$$\delta_{mid}(t_1)=\frac{1}{n}\sum_{(x,y)\in D}|T(x,y,0)-T(x,y,t_1)|$$
$$\delta_{max}(t_1)=max_{(x,y)\in D}(|T(x,y,0)-T(x,y,t_1)|)$$

Тут n - число пар (x, y), принадлежащих телу для вычислительной сетки текущего размера.

Постановка эксперимента на двумерном теле для t более 0.01 секунды с достаточно большой сеткой по каждому из измерений проблематична из-за слишком долгого вычисления (для t=0.2 — более суток), однако мы можем запустить вычисления на меньшие промежутки времени и сравнить результаты с ожидаемыми. Для этого вычислим конечное распределение температуры для тела в момент времени t_1 при условии что начальное распределение температуры задано указанной функцией T_0, скопируем полученные значения в столбцы таблицы Excel и, найдя для них приведённые выше характеристики при помощи табличного процессора Excel, получим:

1: $T_0 = 40$ $t_1 = 0.0001$:	$\delta_{mid}(t_1) = 0.5457$, $\delta_{max}(t_1) = 20$
2: $T_0 = 38.0-8300((x^2+y^2)^{1/2}-0.018)$ $t_1 = 0.0001$:	$\delta_{mid}(t_1) = 0.0727$, $\delta_{max}(t_1) = 1.4$
3: $T_0 = 36.7-8300((x^2+y^2)^{1/2}-0.018)$ $t_1 = 0.0001$:	$\delta_{mid}(t_1) = 0.0508$, $\delta_{max}(t_1) = 0.3329$
4: $T_0 = 36.6-8300((x^2+y^2)^{1/2}-0.018)$ $t_1 = 0.0001$:	$\delta_{mid}(t_1) = 0.0502$, $\delta_{max}(t_1) = 0.3191$
5: $T_0 = 36.5-8300((x^2+y^2)^{1/2}-0.018)$ $t_1 = 0.0001$:	$\delta_{mid}(t_1) = 0.0509$, $\delta_{max}(t_1) = 0.3278$
6: $T_0 = 35.0-8300((x^2+y^2)^{1/2}-0.018)$ $t_1 = 0.0001$:	$\delta_{mid}(t_1) = 0.0733$, $\delta_{max}(t_1) = 1.6$
7: $T_0 = 20$ $t_1 = 0.0001$:	$\delta_{mid}(t_1) = 0.3188$, $\delta_{max}(t_1) = 16.6$

Значения в t=0 вычислялись по указанным формулам для T_0 в Excel для соответствующих x и y сетки. При прогонах использовались следующие начальные условия (на примере строки №4):

Условия задачи:
Уравнение тела: (x*x + y*y <= 0.0004) AND (x*x + y*y >= 0.000324)
Начальная температура тела: 36.6 - 8300 * (pow(x*x + y*y, 0.5) - 0.018)
Температура окружающей среды: if (x*x + y*y >= 0.0004) 20 else 36.6;
Температуропроводность материала: 0.000126
Границы поиска решения:
x ∈ [-0.02, 0.02], число интервалов сетки: 100
y ∈ [-0.02, 0.02], число интервалов сетки: 100
] f(x, y, t) = T — функция температуры T тела в точке (x, y) в момент времени t.

В таблице: столбцы E и F – x и y точки, G – $T_0(x,y)$ независимо от того, принадлежит ли точка телу. M – признак принадлежности точки телу. I, J, K – вывод программы, который она записывает в файл result.txt, L – $\delta(x, y, t_1)$. Содержимое ячеек приведено для случая №3.

= 36,7 - 8300 * ((E10203*E10203 + F10203*F10203)^(1/2) - 0,018)

C	D	E	F	G	H	I	J	K	L	M	N
		0,0124	0,02	-9,216477543		0,0124	0,02	20	29,21647754	ЛОЖЬ	
		0,0128	0,02	-10,98611722		0,0128	0,02	20	30,98611722	ЛОЖЬ	
		0,0132	0,02	-12,79543383		0,0132	0,02	20	32,79543383	ЛОЖЬ	
		0,0136	0,02	-14,64335456		0,0136	0,02	20	34,64335456	ЛОЖЬ	
		0,014	0,02	-16,52882322		0,014	0,02	20	36,52882322	ЛОЖЬ	
		0,0144	0,02	-18,45080151		0,0144	0,02	20	38,45080151	ЛОЖЬ	
		0,0148	0,02	-20,40827005		0,0148	0,02	20	40,40827005	ЛОЖЬ	
		0,0152	0,02	-22,40022926		0,0152	0,02	20	42,40022926	ЛОЖЬ	
		0,0156	0,02	-24,42570009		0,0156	0,02	20	44,42570009	ЛОЖЬ	
		0,016	0,02	-26,48372468		0,016	0,02	20	46,48372468	ЛОЖЬ	
		0,0164	0,02	-28,57336677		0,0164	0,02	20	48,57336677	ЛОЖЬ	
		0,0168	0,02	-30,69371209		0,0168	0,02	20	50,69371209	ЛОЖЬ	
		0,0172	0,02	-32,84386861		0,0172	0,02	20	52,84386861	ЛОЖЬ	
		0,0176	0,02	-35,0229667		0,0176	0,02	20	55,0229667	ЛОЖЬ	
		0,018	0,02	-37,23015918		0,018	0,02	20	57,23015918	ЛОЖЬ	
		0,0184	0,02	-39,46462134		0,0184	0,02	20	59,46462134	ЛОЖЬ	
		0,0188	0,02	-41,72555081		0,0188	0,02	20	61,72555081	ЛОЖЬ	
		0,0192	0,02	-44,01216743		0,0192	0,02	20	64,01216743	ЛОЖЬ	
		0,0196	0,02	-46,32371308		0,0196	0,02	20	66,32371308	ЛОЖЬ	
		0,02	0,02	-48,65945135		0,02	0,02	20	68,65945135	ЛОЖЬ	
						сумма отклонений:			76,0468691		
						ячеек принадлежит области:			1496		
						среднее:				0,050833469	
						максимальное:				0,3329	

В результате серии запусков мы получили результаты, говорящие что изменение температуры по обеим характеристикам тем сильнее, чем дальше начальное распределение температуры от найденного решения, что подтверждает его правильность. Увы, численные характеристики изменений температуры со временем для уравнения (1) не равны нулю, но являются наименьшими из возможных. Такое отклонение от ожидаемых значений объясняется неточностью вычислений с плавающей точкой и неточностью самого численного метода.

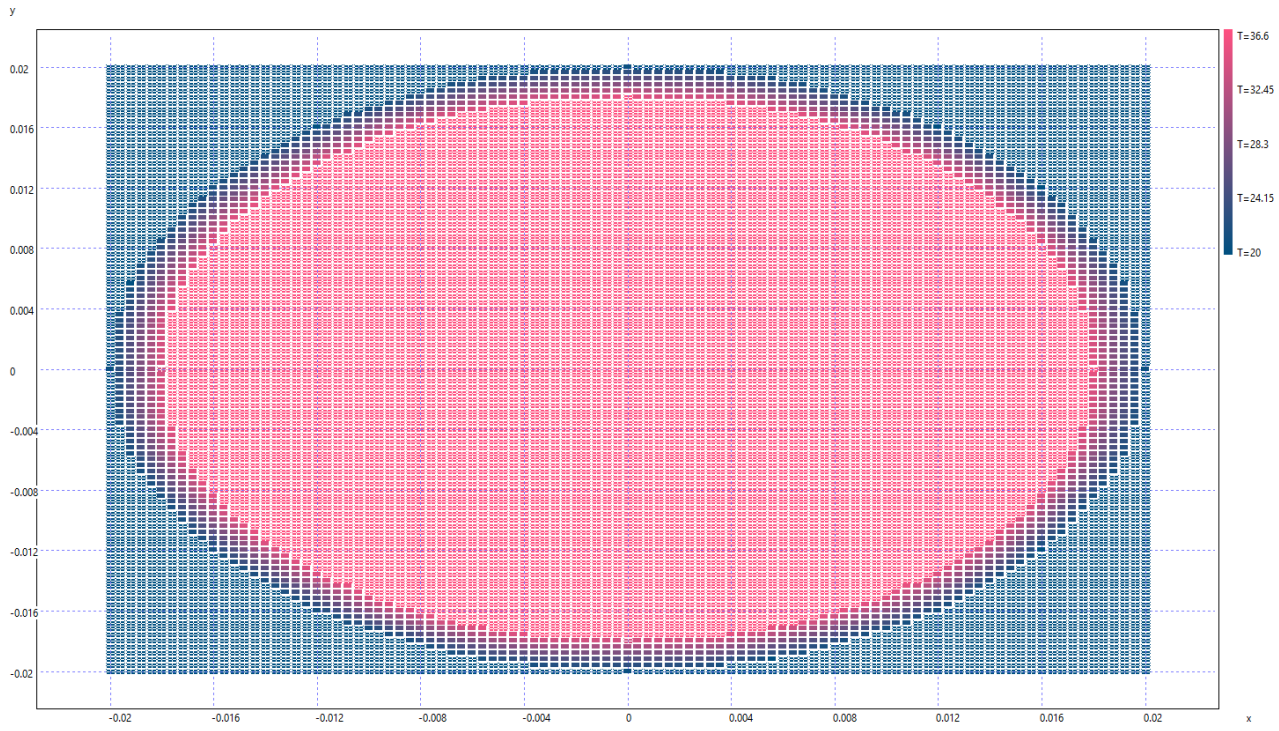
Теперь запустим ту же задачу для t=0.001 и t=0.2; в момент времени t=0.2 система для одномерного случая уже достигла равновесия и перестала изменяться при увеличении t. Время вычисления составило 466 секунд (7,77 минут) и 102871 секунд (28,57528 часов) соответственно. Результаты запусков:

8: $T_0 = 35.0-8300((x^2+y^2)^{1/2}-0.018)$ $t_1 = 0.001$: $\delta_{mid}(t_1) = 0,3252$, $\delta_{max}(t_1) = 1,4411$

9: $T_0 = 35.0-8300((x^2+y^2)^{1/2}-0.018)$ $t_1 = 0.2$: $\delta_{mid}(t_1) = 0,6624$, $\delta_{max}(t_1) = 2,7645$

Следовательно, средняя абсолютная погрешность для данного метода и данной задачи с вычислительной сеткой 101x101 точек составляет 0,6624, а максимальная абсолютная погрешность — 2,7645. При этом, если двумерная система достигает равновесия дольше одномерной, то эти значения могут быть немного больше.

Конечное распределение температуры в кольце может быть отображено графически как (матрица 101x101):



5. Оценка скорости работы программы

В ходе, например, вычисления двумерной задачи из пункта 4 программа запускалась неоднократно и с различными вычислительными условиями.

Например, сравним скорость решения задачи для t=0.0001 и с сеткой 100x100 для различного числа MPI-процессов на четырёхядерном процессоре. Ниже приведены параметры запуска программы и усреднённое примерное время её работы:

mpiexec -n 1	в среднем 70 секунд
mpiexec -n 2	в среднем 40 секунд
mpiexec -n 3	в среднем 50 секунд
mpiexec -n 4	в среднем 35-50 секунд, большой разброс
mpiexec -n 5	более 10 минут

Тут польза от распараллеливания очевидна. Однако для матрицы 10x10 мы получим совсем другую картину:

mpiexec -n 1	в среднем 0.075
mpiexec -n 2	в среднем 0.09
mpiexec -n 3	в среднем 0.1
mpiexec -n 4	в среднем 0.11
mpiexec -n 5	в среднем 0.14
mpiexec -n 10	в среднем 0.9

Вывод: распараллеливание невыгодно при числе MPI-процессоров, превышающем число физических ядер, а также при малом размере матрицы.

В первом случае резкое уменьшение производительности происходит из-за невозможности параллельного выполнения всех процессов одновременно, в результате чего происходит их попеременная работа. Но на каждой итерации по времени MPI-процессоры обмениваются данными друг с другом и если

выполнение одного из них было приостановлено, то он не сможет передавать данные, требуемые другим MPI-процессам для продолжения работы, а также принимать данные, передаваемые от них. В результате, все остальные процессоры будут простаивать, ожидая возобновления приостановленного процессора и завершения операций обмена данными.

Также в этом случае на производительность влияют постоянные перезагрузки контекста потоков внутри ядра ОС, и чем больше потоков требуется выполнить одновременно на одной машине, тем сильнее перезагрузки контекста будут сказываться на общей скорости.

Во втором случае выигрыш от разделения вычислений просто не окупает времени, требуемого на пересылку данных между процессами.

6. Заключение

Итак, программа работает в целом верно, выдаёт достаточно близкий к ожидаемому результат и может принести некоторую пользу (впрочем, не очень большую). У программы много недостатков: невозможность поставить вычислительный процесс на паузу и возобновить при следующем запуске, нет поддержки трёхмерных задач и задач большей размерности, отсутствие поддержки граничных условий 2 и 3 рода, отсутствие поддержки неоднородных тел и т. д. Однако самым существенным и фундаментальным её недостатком, безусловно, является выбранный алгоритм, который недостаточно универсален из-за условия устойчивости и обладает недостаточно высоким быстродействием. Причём быстродействие сильно зависит от вводимых числовых данных, например, от длины шага сетки по каждой переменной — именно длины, а не числа шагов.

Однако разработанная программа и не создавалась для промышленного применения; она представляет собой так называемую «грязную задачу»: в ходе её решения становятся ясны особенности более универсального и общего «чистого решения», а также намечаются проблемы, возникающие при его реализации.

В ходе разработки программы был освоен генератор парсеров ANTLR4, построена грамматика для него, а сгенерированный по ней парсер был интегрирован в обе программы, что позволило задавать произвольные формы тел в задаче, произвольное начальное распределение температуры и произвольную температуру в точках, не принадлежащих телу. Хотя при помощи программы всё равно можно решать только простейшие задачи, её всё же можно назвать инструментом решения некоторого достаточно большого класса задач, а не программой решения одной-единственной задачи с различным коэффициентам.

Для инструмента был разработан оконный интерфейс с использованием библиотеки MFC, визуализирующий полученные результаты. Для задач,

решаемых данным комплексом программ, такой визуализации вполне достаточно.

Были также сделаны выводы по целесообразности распараллеливания задач. Как следует из замеров времени работы, оно невыгодно при числе MPI-процессоров, превышающем число физических ядер, а также при малом количестве вычислений, достающихся каждому процессу. Также следует заметить что выбор именно MPI как средства распараллеливания был выбран в соответствии с требованиями задачи, однако это не самое оптимальное решение для вычислений на одной машине и OpenMP выполнил бы ту же работу быстрее.

MPI действительно ускорил вычисление решения при достаточно больших матрицах, однако польза от него в комбинации с явным методом аппроксимации выглядит сомнительной, т.к. ощутимо уменьшать время вычислений даже на одной машине (т.е. без накладных расходов на пересылку данных по сети) MPI начинает лишь при достаточно большой матрице. Но очень большая матрица также означает очень малый шаг по пространству, а малый шаг по пространству может привести к микроскопически малому максимальному шагу по времени, что диктуется условием устойчивости явной схемы. При таком шаге даже разделив задачу между большим кластером машин ответа можно ждать очень долго.

Явный метод не является полностью бесполезным, т.к. существуют задачи где его ограничений достаточно, но круг таких задач слишком мал чтобы считать его действительно универсальным, что говорит о необходимости изучения более эффективных методов.

7. Список использованной литературы

1. «Высокопроизводительные вычисления на кластерах», Учебное пособие. Под ред. А.В. Старченко. – Томск: Изд-во Том. ун-та, 2008. – 198 с.
2. «Разностные методы решения задач теплопроводности: учебное пособие» Г.В. Кузнецов, М.А. Шеремет. – Томск: Изд-во ТПУ, 2007. – 172с
3. Официальная документация по ANTLR4:

   https://github.com/antlr/antlr4/blob/master/doc/index.md
4. Статья «The Definitive ANTLR Reference»:

   https://media.pragprog.com/titles/tpantlr/errors.pdf
5. Статья «The ANTLR mega tutorial»: https://tomassetti.me/antlr-mega-tutorial/
6. Статья «Обмен данными с использованием MPI. Работа с библиотекой MPI на примере Intel® MPI Library»:

   https://habrahabr.ru/company/intel/blog/251357/
7. Круглински Д., Уингоу С., Шеферд Дж «Программирование на Microsoft Visual C++ 6.0 для профессионалов», Питер, 2001, 864 с.

8. Приложение

8.1. Полное описание грамматики языка

```
grammar MathGrammar;

@header{
#include "stdafx.h"
#include "../RK_frontend/lang/Calculatables.h"
#include <vector>
}

// NB: данная продукция включает в себя окружающие её пробелы,
// в то время как продукции более нижнего уровня включают только внутренние.
root
returns[lang::Calculatable* node, lang::ExpressionType type]:
opt_WS
(real_expression{ $node = $real_expression.node; $type = lang::ExpressionType::Double; }
| bool_expression{ $node = $bool_expression.node; $type = lang::ExpressionType::Bool; }
) opt_WS EOF
;

// ---------------- BOOL -------------------------------------------------------

bool_expression
returns[lang::Calculatable* node]:
first = bool_atom{ $node = $first.node; }
(opt_WS AND opt_WS next = bool_atom{ $node = new lang::OperatorAnd($node, $next.node); }
| opt_WS OR  opt_WS next = bool_atom{ $node = new lang::OperatorOr($node, $next.node); }
)*;

bool_atom
returns[lang::Calculatable* node]:
comparison{ $node = $comparison.node; }
| NOT opt_WS bool_atom{ $node = new lang::OperatorNot($bool_atom.node); }
| BR_OP opt_WS bool_expression opt_WS BR_CL{ $node = $bool_expression.node; }
;

comparison
returns[lang::Calculatable* node]:
left = real_expression opt_WS EQVAL      opt_WS right = real_expression{ $node = new
lang::Comparison($left.node, $right.node, lang::CmpType::Equal); }
| left = real_expression opt_WS NOT_EQVAL  opt_WS right = real_expression{ $node = new
lang::Comparison($left.node, $right.node, lang::CmpType::NotEqual); }
| left = real_expression opt_WS GREATER    opt_WS right = real_expression{ $node = new
lang::Comparison($left.node, $right.node, lang::CmpType::Greater); }
| left = real_expression opt_WS LESS       opt_WS right = real_expression{ $node = new
lang::Comparison($left.node, $right.node, lang::CmpType::Less); }
| left = real_expression opt_WS GREATER_EQ opt_WS right = real_expression{ $node = new
lang::Comparison($left.node, $right.node, lang::CmpType::GreaterOrEqual); }
| left = real_expression opt_WS LESS_EQ    opt_WS right = real_expression{ $node = new
lang::Comparison($left.node, $right.node, lang::CmpType::LessOrEqual); }
;

// ---------------- REAL -------------------------------------------------------
////////////////////////////////////////////////////////////////////////////////
// Оператор ветвления.
if_expr
returns[lang::OperatorIf* node]:
IF opt_WS
BR_OP opt_WS expr1 = bool_expression opt_WS BR_CL
opt_WS exec1 = real_expression
{ $node = new lang::OperatorIf(); $node->AddElseIf($expr1.node, $exec1.node); }
```

```
(opt_WS ELSE WS IF opt_WS
       BR_OP opt_WS expr_next = bool_expression opt_WS BR_CL
       opt_WS exec_next = real_expression
{ $node->AddElseIf($expr_next.node, $exec_next.node); }
)*
opt_WS ELSE
opt_WS last = real_expression opt_WS SEMIC
{ $node->AddElse($last.node); }
;

////////////////////////////////////////////////////////////////////////////
real_expression
returns[lang::Calculatable* node]:
add_expr{ $node = $add_expr.node; };

////////////////////////////////////////////////////////////////////////////
add_expr
returns[lang::Calculatable* node]:
first = mult_expr{ $node = $first.node; }
(opt_WS PLUS  opt_WS next = mult_expr{ $node = new lang::OperatorAdd($node, $next.node); }
| opt_WS MINUS opt_WS next = mult_expr{ $node = new lang::OperatorSub($node, $next.node); }
)*;

////////////////////////////////////////////////////////////////////////////
mult_expr
returns[lang::Calculatable* node]:
first = atom{ $node = $first.node; }
(opt_WS MUL opt_WS next = atom{ $node = new lang::OperatorMul($node, $next.node); }
| opt_WS DIV opt_WS next = atom{ $node = new lang::OperatorDiv($node, $next.node); }
)*;

////////////////////////////////////////////////////////////////////////////
atom
returns[lang::Calculatable* node]:
number{ $node = $number.node; }
| id{ $node = $id.node; }
| function{ $node = $function.node; }
| if_expr{ $node = $if_expr.node; }
| BR_OP opt_WS add_expr opt_WS BR_CL{ $node = $add_expr.node; };

////////////////////////////////////////////////////////////////////////////
function
returns[lang::Calculatable* node]:
id opt_WS
(BR_OP opt_WS BR_CL{ $node = new lang::FunctionCall($id.node); }
| BR_OP opt_WS param_list opt_WS BR_CL{ $node = new lang::FunctionCall($id.node,
$param_list.args); }
);

////////////////////////////////////////////////////////////////////////////
// Список параметров функции. Не может быть пуст.
param_list
returns[std::vector<lang::Calculatable*> args]:
(param = add_expr opt_WS COMMA opt_WS{ $args.push_back($param.node); }
       )*
       last = add_expr{ $args.push_back($last.node); };

////////////////////////////////////////////////////////////////////////////
// Идентификатор. Самостоятельный идентификатор становится переменной,
// входящий в функцию — именем функции.
id
returns[lang::Calculatable* node]:
ID{ $node = new lang::Variable($text); };

////////////////////////////////////////////////////////////////////////////
```

```
// Вещественное число.
number
returns[lang::Calculatable* node]:
(MINUS opt_WS) ? P_FLOAT{ $node = new lang::Number(atof(($text).c_str())); };

////////////////////////////////////////////////////////////////////////////
// Пробельные символы, которые могут как быть, так и нет.
opt_WS: WS | nothing;

////////////////////////////////////////////////////////////////////////////
nothing:;

// ----------------- LEXER ----------------------------------------------------
// Мы не различаем регистр идентификаторов.
fragment A : 'a' | 'A';
fragment B : 'b' | 'B';
fragment C : 'c' | 'C';
fragment D : 'd' | 'D';
fragment E : 'e' | 'E';
fragment F : 'f' | 'F';
fragment G : 'g' | 'G';
fragment H : 'h' | 'H';
fragment I : 'i' | 'I';
fragment J : 'j' | 'J';
fragment K : 'k' | 'K';
fragment L : 'l' | 'L';
fragment M : 'm' | 'M';
fragment N : 'n' | 'N';
fragment O : 'o' | 'O';
fragment P : 'p' | 'P';
fragment Q : 'q' | 'Q';
fragment R : 'r' | 'R';
fragment S : 's' | 'S';
fragment T : 't' | 'T';
fragment U : 'u' | 'U';
fragment V : 'v' | 'V';
fragment W : 'w' | 'W';
fragment X : 'x' | 'X';
fragment Y : 'y' | 'Y';
fragment Z : 'z' | 'Z';

IF:         I F;
ELSE:       E L S E;
AND:        A N D;
OR:         O R;
NOT:        N O T;
ID:         ([a - zA - Z] | '_') ([a - zA - Z] | '_' | [0 - 9])*;
P_FLOAT:    [0 - 9] + (DOT[0 - 9] + ) ? (E(PLUS | MINUS) ? [0 - 9] + ) ? ; // P == Positive
SEMIC:      ';';
PLUS:       '+';
MINUS:      '-';
MUL:        '*';
DIV:        '/';
DOT:        '.';
COMMA:      ',';
EQVAL:      '=';
NOT_EQVAL:  '!=';
GREATER:    '>';
LESS:       '<';
GREATER_EQ: '>=';
LESS_EQ:    '<=';
BR_OP: '(';
BR_CL: ')';
WS: [\n\t\r] + ;
```

8.2. Фрагмент кода программы, производящей численное решение задач

main.cpp

```
#include <mpi.h>
#include <iostream>
#include <windows.h>
#include <fstream>
using namespace std;

#include "../RK_frontend/lang/MathExpression.h"
#include "../RK_frontend/lang/TaskConditions.h"
using namespace lang;

#include "Matrix2D.h"
#include "GenericSolution.h"
#include "FullBuilderSolution.h"
#include "TemperatureChangesSolution.h"
#include "TemperatureCalculatorSolution.h"
#include "TimeCalculatorSolution.h"
using namespace solution;

////////////////////////////////////////////////////////////////////////////
char szOutputFile[] = "result.txt";

////////////////////////////////////////////////////////////////////////////
int main(int argc, char **argv)
{
	if (int error = MPI_Init(&argc, &argv))
	{
		cout << "Error. MPI error number: " << error << endl;
		MPI_Abort(MPI_COMM_WORLD, error);
	}

	int rank, size;
	MPI_Comm_rank(MPI_COMM_WORLD, &rank);
	MPI_Comm_size(MPI_COMM_WORLD, &size);
	cout << "Started processor #" << rank << " of " << size << endl;

	try
	{
		// Производим разбор параметров задачи.
		TaskConditions task(argc, argv, 1);

		// Выбираем тип задачи.
		GenericSolution *pSolution;
		switch (task.taskType)
		{
		case FIND_TEMPERATURE_AT:
			pSolution = new TemperatureCalculatorSolution(task, rank, size);
			break;
		case FIND_TIME_WHEN_TEMPERATURE_IS:
			pSolution = new TimeCalculatorSolution(task, rank, size);
			break;
		case BUILD_TEMPERATURE_CHANGES_FOR_POINT:
			pSolution = new TemperatureChangesSolution(task, rank, size);
			break;
		case BUILD_FULL_BODY_FOR_TIME:
			pSolution = new FullBuilderSolution(task, rank, size);
			break;
		default:
			throw exception("Unknown task type found.");
		}
```

```
		// Решаем задачу.
		pSolution->Solve();

		// Выводим ответ, если мы - последний процессор.
		if (rank == size - 1)
		{
			ofstream out(szOutputFile);
			out << TaskResult::RESULT_SUCCESS << endl;
			pSolution->WriteAnsver(out);
			out.close();
		}

		// Очищаем память.
		delete pSolution;
	}
	catch (std::exception ex)
	{
		cout << "Processor #" << rank << " error: " << ex.what() << endl;

		ofstream out(szOutputFile);
		out << TaskResult::RESULT_ERROR << endl
			<< "Processor #" << rank << " error: " << ex.what() << endl;
		out.close();

		MPI_Abort(MPI_COMM_WORLD, 0);
	}

	MPI_Finalize();
	return 0;
}

////////////////////////////////////////////////////////////////////////////////
```

GenericSolution.h

```
#pragma once
#include <mpi.h>
#include <iostream>
#include <fstream>
using namespace std;

#include "../RK_frontend/lang/MathExpression.h"
#include "../RK_frontend/lang/TaskConditions.h"
using namespace lang;

#include "Matrix2D.h"
using namespace solution;


namespace solution {

	////////////////////////////////////////////////////////////////////////////
	class GenericSolution
	{
	public:
		enum PontLocation { ENVIRONMENT_POINT = 0, BODY_POINT = 1, CALCULATE }; //
Нумерация оставлена чтобы при случайной перетасовке значения не поменялись.

	protected:
		// MPI
		const int mpiRank;          // Номер данного процессора внутри группы
		const int mpiSize;          // Количество процессоров в группе
		const int mpiLeftID; // Левый MPI-сосед (если есть), с которым будем
```

```
обмениваться данными
        const int mpiRightID;       // Правый MPI-сосед (если есть), с которым будем
обмениваться данными

        MathExpression expressionBody;
        MathExpression expressionInitBodyTemperature;
        MathExpression expressionEvironmentTemperature;

        const size_t dimensionCount;

        // общее число точек тела
        const size_t pointCountX;
        const size_t pointCountY;

        // число точек, которое вычисляет наш процессор
        // одномерные делим по X, двумерные по Y
        const size_t myPointCountX;
        const size_t myPointCountY;

        // число точек, которое вычисляют все процессоры кроме последнего, по X (без
границ)
        const size_t stdPointCountX;
        // число точек, которое вычисляют все процессоры кроме последнего, по Y (без
границ)
        const size_t stdPointCountY;

        // число точек, которое вычисляет наш процессор, включая границу
        const size_t nMyPointCountWithBorderX;
        const size_t nMyPointCountWithBorderY;

        Matrix2D<double> mtxTemperature;
        Matrix2D<double> mtxTemperatureNew;
        Matrix2D<bool>   mtxIsBodyPoint;   // Принадлежит ли точка телу.

        const double begX;
        const double begY;

        const double stepX;
        const double stepY;

        // Коэффициент температуропроводности, везде обозначается как "альфа".
        const double alpha;


    public:
        ////////////////////////////////////////////////////////////////////////
        GenericSolution(const TaskConditions& task, const int MPI_rank, const int
MPI_size) :
                mpiRank(MPI_rank),
                mpiSize(MPI_size),
                mpiLeftID(MPI_rank ? MPI_rank - 1 : MPI_PROC_NULL),
                mpiRightID(MPI_rank < (MPI_size - 1) ? MPI_rank + 1 : MPI_PROC_NULL),

                expressionBody(task.strBodyExpression),
                expressionInitBodyTemperature(task.strTemperatureAt0),

    expressionEvironmentTemperature(task.strEvironmentTemperatureExpression),

                dimensionCount(expressionBody.GetVariableCount()),

                // При разбиении отрезка число точек разбиения всегда больше числа
интервалов ровно на 1.
                pointCountX(task.nVar0IntervalCount + 1),
                pointCountY((dimensionCount == 2) ? task.nVar1IntervalCount + 1 : 1),
```

```
                        // В одномерных задачах делим по X, в двумерных по Y.
                        // Находим число элементов по первой переменной, с которыми будем
работать.
                        // Мы делим между MPI-процессорами всегда итерации именно первой
переменной, т.к. она есть всегда.
                        // Делим поровну, но последний MPI-процессор получает также остаток от
деления.
                        myPointCountX(
                                (dimensionCount == 1) ?
                                ((MPI_rank < MPI_size - 1) ? (pointCountX / MPI_size) :
(pointCountX / MPI_size + pointCountX % MPI_size))
                                : pointCountX),

                        myPointCountY(
                                (dimensionCount == 2) ?
                                ((MPI_rank < MPI_size - 1) ? (pointCountY / MPI_size) :
(pointCountY / MPI_size + pointCountY % MPI_size))
                                : pointCountY),

                        stdPointCountX((dimensionCount == 1) ? (pointCountX / MPI_size) :
pointCountX),
                        stdPointCountY((dimensionCount == 2) ? (pointCountY / MPI_size) :
pointCountY),

                        // Если у нас есть левый или правый сосед, крайний влево/вправо столбец
мы будем принимать от него.
                        // Иначе левый/правый столбец будет содержать температуру окружающей
среды.
                        // Так или иначе, край в 1 столбец с обеих сторон есть всегда и никогда
не пересчитывается.
                        nMyPointCountWithBorderX(myPointCountX + 2),
                        // По Y сверху и снизу мы всегда имеем край в 1 строку, заполненный
температурой окружающей среды.
                        nMyPointCountWithBorderY(myPointCountY + 2),

                        mtxTemperature(nMyPointCountWithBorderX, nMyPointCountWithBorderY),
                        mtxTemperatureNew(nMyPointCountWithBorderX, nMyPointCountWithBorderY),
                        mtxIsBodyPoint(nMyPointCountWithBorderX, nMyPointCountWithBorderY),

                        begX(task.dblVar0From),
                        begY((dimensionCount == 2) ? task.dblVar1From : 0),

                        stepX(abs(task.dblVar0To - task.dblVar0From) /
task.nVar0IntervalCount),
                        stepY((dimensionCount == 2) ? abs(task.dblVar1To - task.dblVar1From) /
task.nVar1IntervalCount : 0),

                        alpha(task.dblThermalConductivity)
                {
                        if (task.dblVar0From >= task.dblVar0To
                        || (dimensionCount == 2 && task.dblVar1From >= task.dblVar1To))
                                throw std::exception("Invalid field borders: FROM must be <
TO.");

                        if (dimensionCount != 1 && dimensionCount != 2)
                                throw std::exception("Invalid dimension count of the body
expression. Can be 1 or 2.");

                        if (task.nVar0IntervalCount < mpiSize)
                                throw std::exception("Invalid interval count of the first
variable. It must be greater or equal to count of MPI-processors.");

                        if (task.dblThermalConductivity <= 0)
                                throw exception("Thermal conductivity must be >= 0.");
```

```
                InitMatrixes();
        }

        ////////////////////////////////////////////////////////////////////////
        virtual ~GenericSolution() {}

        ////////////////////////////////////////////////////////////////////////
        virtual void Solve() = 0;

        ////////////////////////////////////////////////////////////////////////
        virtual void WriteAnsver(ofstream& out) = 0;


    protected:
        ////////////////////////////////////////////////////////////////////////
        // Максимальный шаг по времени, вычисляется на основе параметров задачи.
        inline double MaxTimeDelta() const
        {
            // Максимальный шаг определяется условием устойчивости:
            //              step * step
            // deltaTime < -----------.
            //              2 * alpha
            // Мы же берём половину от максимального:
            return 0.25 * stepX * stepX * ((dimensionCount >= 2) ? stepY * stepY :
1) / alpha;
        }

        ////////////////////////////////////////////////////////////////////////
        // Коэффициенты уравнения аппроксимации для указанного шага по времени.
        inline double Qx(const double timeDelta) const { return
alpha * timeDelta / (stepX * stepX); }
        inline double Qy(const double timeDelta) const { return (dimensionCount ==
2) ? alpha * timeDelta / (stepY * stepY) : 0; }
        inline double Qt(const double timeDelta) const { return 1 - 2 * Qx(timeDelta)
- 2 * Qy(timeDelta); }

        ////////////////////////////////////////////////////////////////////////
        // Устанавливает значение точки с заданными координатами mtxIsBodyPoint[] и
mtxTemperature[].
        // Если точка принадлежит среде, то функция вычисляет для неё температуру
внешней среды в данной точке.
        // Если точка принадлежит телу, то функция находит для неё начальную
температуру тела в данной точке.
        // Если location установлена в CALCULATE, то вычисляет принадлежит ли точка
телу самостоятельно.
        // Используется для инициализации матриц.
        inline void SetPointAs(const PontLocation location, const size_t indexX, const
size_t indexY)
        {
            // Число столбцов/строк, которые мы пропустили из-за MPI-разбиения.
            const int offsetX = (dimensionCount == 1) ? (stdPointCountX *
mpiRank) : 0;
            const int offsetY = (dimensionCount == 2) ? (stdPointCountY *
mpiRank) : 0;

            // Чтобы избежать накопления ошибок округления после каждого сложения,
            // вычисляем данные значения умножением. Благо что эта функция
вызывается только
            // при инициализации, а не на каждой итерации основного цикла,
            // так что оверхед за точность относительно небольшой.
            //
            // NB: (indexX - 1), т.к. сетка у нас начинается с единицы, а нулевой
индекс - край.
            double args[] = {
                begX + stepX * (((int)indexX) - 1 + offsetX),    // -1 для
```

```
нулевого индекса
                        begY + stepY * (((int)indexY) - 1 + offsetY)    // -1 для
нулевого индекса
                };

                // Принадлежит ли эта точка телу?
                const bool isBody =
                        (location == PontLocation::CALCULATE) ?
(expressionBody.Calculate(args).boolValue)
                        : location;

                mtxIsBodyPoint(indexX, indexY) = isBody;

                mtxTemperature(indexX, indexY) = (isBody) ?
                        expressionInitBodyTemperature.Calculate(args).doubleValue
                        : expressionEvironmentTemperature.Calculate(args).doubleValue;
        }

        ////////////////////////////////////////////////////////////////////////
        // Инициализирует матрицы начальными (t=0) и граничными (T окружающей среды)
условиями.
        // AHTUNG: Инициализирует ВСЕ границы температурой внешней среды.
        // После такой инициализации обязательно требуется обмен с соседними
элементами и замена значений внутренних границ ДО начала вычислений.
        void InitMatrixes()
        {
                // Внутренняя часть массивов.
                // Не факт что с обеих сторон всё решение окружено "внешней средой" в
одну ячейку шириной.
                // Возможно что слева у нас есть сосед и тогда левый край тоже следует
рассчитать.
                // Границы перезапишем ниже.
                for (size_t j = 0; j < nMyPointCountWithBorderY; j++)
                        for (size_t i = 0; i < nMyPointCountWithBorderX; i++)
                                SetPointAs(PontLocation::CALCULATE, i, j);

                // Границы. ---------------
                // слева
                // Если делим по Y, то всегда есть, если по X, то если слева нет соседа
                if (dimensionCount == 2 || mpiLeftID == MPI_PROC_NULL)
                        for (size_t j = 0; j < nMyPointCountWithBorderY; j++)
                                SetPointAs(PontLocation::ENVIRONMENT_POINT, 0, j);

                // справа
                if (dimensionCount == 2 || mpiRightID == MPI_PROC_NULL)
                        for (size_t j = 0; j < nMyPointCountWithBorderY; j++)
                                SetPointAs(PontLocation::ENVIRONMENT_POINT,
nMyPointCountWithBorderX - 1, j);

                // сверху
                if (dimensionCount == 1 || mpiLeftID == MPI_PROC_NULL)
                        for (size_t i = 0; i < nMyPointCountWithBorderX; i++)
                                SetPointAs(PontLocation::ENVIRONMENT_POINT, i, 0);

                // снизу
                if (dimensionCount == 1 || mpiRightID == MPI_PROC_NULL)
                        for (size_t i = 0; i < nMyPointCountWithBorderX; i++)
                                SetPointAs(PontLocation::ENVIRONMENT_POINT, i,
nMyPointCountWithBorderY - 1);

                mtxTemperatureNew = mtxTemperature;
        }

        ////////////////////////////////////////////////////////////////////////
};
```

```
	//////////////////////////////////////////////////////////////////////////
}
```

FullBuilderSolution.h

```
#pragma once
#include "GenericSolution.h"
#include <limits.h>

namespace solution
{
	//////////////////////////////////////////////////////////////////////////
	class FullBuilderSolution : public GenericSolution {
	protected:
		static const size_t MAX_ALLOWED_ITERATION_COUNT = UINT_MAX;

	protected:
		const double endTime;

		double time;

		bool mustBuildResult;
		Matrix2D<double>* pResult;

	public:
		//////////////////////////////////////////////////////////////////////////
		FullBuilderSolution(const TaskConditions& task, const int MPI_rank, const int
MPI_size) :
			GenericSolution(task, MPI_rank, MPI_size),
			endTime(task.BUILD_FULL_BODY_FOR_TIME.dblEndTime),
			pResult(0),
			mustBuildResult(1)
		{}

		//////////////////////////////////////////////////////////////////////////
		void WriteAnsver(ofstream& out) override
		{
			if (mpiRank != mpiSize - 1)
				return;

			if (dimensionCount == 1)
			{
				// 1D
					for (size_t i = 0; i < pointCountX; i++)
						out << '\n'
						<< begX + stepX * i << ' '
						<< 0 << ' '
						<< (*pResult)(i, 0);
			}
			else
			{
				// 2D
				for (size_t j = 0; j < pointCountY; j++) // по Y без смещения
					for (size_t i = 1; i < pointCountX + 1; i++) // 1 ==
смещение, бордюр не печатаем
						out << '\n'
						<< begX + stepX * (i - 1) << ' '
						<< begY + stepY * j << ' '
						<< (*pResult)(i, j);
			}
		}

		//////////////////////////////////////////////////////////////////////////
```

```
		void Solve() override
		{
			// Вычисление коэффициентов.
			double deltaTime = MaxTimeDelta();
			double qx = Qx(deltaTime);
			double qy = Qy(deltaTime);
			double qt = Qt(deltaTime);

			// Собственно, расчёты.
			bool endIsNotReached = true;	// Пока true вычисления продолжаются

			// Иначе получим все границы заполненными температурой внешней среды.
			ExchangeBorderValuesWithNeighbors();

			// Собственно, цикл вычисления новых временных слоёв с шагом по времени deltaTime.
			for (size_t step = 1; endIsNotReached; step++)
			{
				// NB: первая итерация - не в нуле, а в deltaTime.

				// Текущий слой времени (для всех итераций кроме последней).
				// Если вычислять его через сложение то получим накопление погрешности и тем большее, чем больше итераций.
				// Через умножение чуть медленнее, но по сравнению с пересчётом матрицы это ничто.
				time = step * deltaTime;
				if (time >= endTime)
				{
					deltaTime = endTime - (step - 1) * deltaTime; // Dt = end - время_предыдущего_временного_слоя
					time = endTime;
					// Обновляем коэффициенты.
					qx = Qx(deltaTime);
					qy = Qy(deltaTime);
					qt = Qt(deltaTime);
					// Это последняя итерация.
					endIsNotReached = false;
				}

				// Пересчитываем только точки, принадлежащие телу.
				// Делаем отступ в 1 ячейку с начала и конца по всем измерениям чтобы не обрабатывать границу как отдельный случай.
				for (size_t j = 1; j < nMyPointCountWithBorderY - 1; j++)
					for (size_t i = 1; i < nMyPointCountWithBorderX - 1; i++)
						// Это точка не принадлежит телу - пропускаем.
						// Принадлежит - пересчитываем её значение в новом временном слое.
						if (mtxIsBodyPoint(i, j))
							mtxTemperatureNew(i, j) =
							qt * mtxTemperature(i, j) +
							qx * mtxTemperature(i + 1, j) +
							qx * mtxTemperature(i - 1, j) +
							((dimensionCount == 2) ?
								(qy * mtxTemperature(i, j + 1) +
									qy * mtxTemperature(i, j - 1))
								: 0);

				mtxTemperature = mtxTemperatureNew;

				// При достижении максимального числа итераций считаем что результат не может быть достигнут за разумное время.
				if (step == MAX_ALLOWED_ITERATION_COUNT)
					throw std::exception("The iteration limit has reached. Aborting process.");
```

```
                        ExchangeBorderValuesWithNeighbors();

                        AdditionalTimeSliceThings();
                }

                //
---------------------------------------------------------------------------
                // Далее собираем результат в одну большую матрицу pResult, если это
требуется
                if (!mustBuildResult)
                        return;

                // По-хорошему, следовало бы разделить 1D и 2D как-то иначе, например
запихнуть в разные классы, либо же унифицировать.

                // 1D
                if (dimensionCount == 1)
                {
                        if (mpiRank == mpiSize - 1)
                                pResult = new Matrix2D<double>(pointCountX, 1);

                        MPI_Gather(mtxTemperature.GetRawPointer(1, 1), stdPointCountX,
MPI_DOUBLE,
                                (pResult) ? pResult->GetRawPointer(0, 0) : 0,
stdPointCountX, MPI_DOUBLE, mpiSize - 1, MPI_COMM_WORLD);

                        // Дописываем остаток, (myPointCountX - stdPointCountX) строк.
                        // Уже записано в матрицу stdPointCountX * mpiSize строк.
                        if (mpiRank == mpiSize - 1)
                        {
                                int rowInResult = stdPointCountX * mpiSize;
                                int rowInMyMatrix = stdPointCountX + 1;

                                for (size_t i = 0; i < myPointCountX - stdPointCountX; i+
+)
                                {
                                        (*pResult)(i + rowInResult, 0) = mtxTemperature(i
+ rowInMyMatrix, 1);
                                }
                        }
                }
                // 2D
                else
                {
                        // В ответе - полное тело + граница по X. Границы по Y скипаем.
                        if (mpiRank == mpiSize - 1)
                                pResult = new Matrix2D<double>(pointCountX + 2,
pointCountY);

                        int blockSize = stdPointCountY * (stdPointCountX + 2);

                        // Собираем данные на последнем MPI-процессоре Почему именно на
нём? Потому что у него больше элементов,
                        // чем у любого другого, т.к. именно ему достаётся остаток. И
этот остаток он должен дописать.
                        MPI_Gather(mtxTemperature.GetRawPointer(0, 1), blockSize,
MPI_DOUBLE,
                                (pResult) ? pResult->GetRawPointer(0, 0) : 0, blockSize,
MPI_DOUBLE, mpiSize - 1, MPI_COMM_WORLD);

                        // Дописываем остаток, (myPointCountY - stdPointCountY) строк.
                        // Уже записано в матрицу stdPointCountY * mpiSize строк.
                        if (mpiRank == mpiSize - 1)
                        {
                                int lineInResult   = stdPointCountY * mpiSize;
```

```
                                int lineInMyMatrix = stdPointCountY + 1; // 1 == бордюр,
который при сборе ответа мы скипаем у всех блоков

                                for (size_t i = 0; i < myPointCountY - stdPointCountY; i+
+)
                                        for (size_t j = 0; j < nMyPointCountWithBorderX;
j++)
                                                (*pResult)(j, i + lineInResult) =
mtxTemperature(j, i + lineInMyMatrix);
                        }
                }
        }

        ////////////////////////////////////////////////////////////////////////
        virtual void AdditionalTimeSliceThings() { }

        ////////////////////////////////////////////////////////////////////////
        void ExchangeBorderValuesWithNeighbors()
        {
                // Обмениваемся данными с остальными MPI-процессорами, если они
существуют.
                MPI_Status status;
                // 1D
                if (dimensionCount == 1)
                {
                        // Y == 1 для одномерного случая
                        MPI_Sendrecv(
                                mtxTemperature.GetRawPointer(1, 1), 1, MPI_DOUBLE,
mpiLeftID, 100,      // Отправляем [1] по X
                                mtxTemperature.GetRawPointer(0, 1), 1, MPI_DOUBLE,
mpiLeftID, 100,      // Принимаем  [0] по X
                                MPI_COMM_WORLD, &status);

                        MPI_Sendrecv(
                                mtxTemperature.GetRawPointer(nMyPointCountWithBorderX -
2, 1), 1, MPI_DOUBLE, mpiRightID, 100, // отправляем [last-1] по X
                                mtxTemperature.GetRawPointer(nMyPointCountWithBorderX -
1, 1), 1, MPI_DOUBLE, mpiRightID, 100, // принимаем  [last]   по X
                                MPI_COMM_WORLD, &status);
                }
                // 2D
                else
                {
                        // Для 2D делим по Y
                        MPI_Sendrecv(
                                // Шлём ПЕРВУЮ (не нулевую!) строку левому соседу. Мы
только что вычислили этот блок.
                                mtxTemperature.GetRawPointer(0, 1),
nMyPointCountWithBorderX, MPI_DOUBLE, mpiLeftID, 100,
                                // Принимаем НУЛЕВУЮ строку от левого соседа.
                                mtxTemperature.GetRawPointer(0, 0),
nMyPointCountWithBorderX, MPI_DOUBLE, mpiLeftID, 100,
                                MPI_COMM_WORLD, &status);

                        MPI_Sendrecv(
                                // Шлём ПРЕДПОСЛЕДНЮЮ строку правому соседу.
                                mtxTemperature.GetRawPointer(0, nMyPointCountWithBorderY
- 2), nMyPointCountWithBorderX, MPI_DOUBLE, mpiRightID, 100,
                                // Принимаем ПОСЛЕДНЮЮ строку него.
                                mtxTemperature.GetRawPointer(0, nMyPointCountWithBorderY
- 1), nMyPointCountWithBorderX, MPI_DOUBLE, mpiRightID, 100,
                                MPI_COMM_WORLD, &status);
                }
        }
```

```
	};

	////////////////////////////////////////////////////////////////////////////
}
```

TimeCalculatorSolution.h

```
#pragma once
#include "GenericPointSolution.h"

namespace solution
{
	////////////////////////////////////////////////////////////////////////////
	// Температура в ближайшем узле сетки к заданному.
	class TimeCalculatorSolution : public GenericPointSolution {
		const double temperature;
		const bool conditionT0lessT1;

		double timeAnsver;

	public:
		////////////////////////////////////////////////////////////////////////
		TimeCalculatorSolution(const TaskConditions& task, const int MPI_rank, const
int MPI_size) :
			GenericPointSolution(task, MPI_rank, MPI_size),
			temperature(task.FIND_TIME_WHEN_TEMPERATURE_IS.dblTemperature),
			conditionT0lessT1(mtxTemperatureNew(indexX, indexY) < temperature)
		{
		}

		////////////////////////////////////////////////////////////////////////
		void WriteAnsver(ofstream& out) override
		{
			if (mpiRank != mpiSize - 1)
				return;

			out << timeAnsver << endl;
		}

		////////////////////////////////////////////////////////////////////////
		void Solve() override
		{
			const int allArgCount = 6;
			lang::Variant allArgs[allArgCount];

			double& time      = allArgs[0].doubleValue;
			double& deltaTime = allArgs[1].doubleValue;
			double& qx        = allArgs[2].doubleValue;
			double& qy        = allArgs[3].doubleValue;
			double& qt        = allArgs[4].doubleValue;
			bool& mustContinue = allArgs[5].boolValue;

			// Вычисление коэффициентов.
			time = 0;
			deltaTime = MaxTimeDelta();
			qx = Qx(deltaTime);
			qy = Qy(deltaTime);
			qt = Qt(deltaTime);
			mustContinue = true;

			ExchangeBorderValuesWithNeighbors();

			// Поехали
			for (size_t step = 0; mustContinue; step++)
```

```
                {
                        time += deltaTime;

                        const double oldTemperature = mtxTemperature(indexX, indexY);

                        // Пересчитываем только точки, принадлежащие телу.
                        // Делаем отступ в 1 ячейку с начала и конца по всем измерениям
чтобы не обрабатывать границу как отдельный случай.
                        for (size_t j = 1; j < nMyPointCountWithBorderY - 1; j++)
                                for (size_t i = 1; i < nMyPointCountWithBorderX - 1; i++)
                                        // Это точка не принадлежит телу - пропускаем.
                                        // Принадлежит - пересчитываем её значение в новом
временном слое.
                                        if (mtxIsBodyPoint(i, j))
                                                mtxTemperatureNew(i, j) =
                                                qt * mtxTemperature(i, j) +
                                                qx * mtxTemperature(i + 1, j) +
                                                qx * mtxTemperature(i - 1, j) +
                                                ((dimensionCount == 2) ?
                                                        (qy * mtxTemperature(i, j + 1) +
                                                         qy * mtxTemperature(i, j - 1))
                                                        : 0);

                        mtxTemperature = mtxTemperatureNew;

                        // ----------------------------------------------------
                        // Алгоритм.
                        // Делаем шаг. Проверяем. Перешагнули?
                        // Нет: шагаем дальше.
                        // Да: откатываемся, делим шаг на 2 и повторяем.
                        //     '--> Число перестало меняться? Мы достигли предела,
заканчиваем.
                        // Навернули <over 9000> циклов -> аварийный выход.
                        if (step == MAX_ALLOWED_ITERATION_COUNT)
                                throw std::exception("The iteration limit has reached.
Aborting process.");

                        ExchangeBorderValuesWithNeighbors();

                        // Если мы — процессор-владелец точки, то устанавливаем признак
завершения вычислений, если это требуется.
                        // Также рассылаем новые значения коэффициентов, если уменьшаем
шаг.
                        // NB: т.к. далее идёт Bcast, то использовать
break/return/continue до него нельзя.
                        if (pointIsMy)
                        {
                                // ----------------------------------------------------
                                // Вычислили слой. Ну что там как с температурой?
                                // Как определить что мы перешагнули порог?
                                // T  - требуемая температура
                                // T* - текущая, T0 - в нуле (t=t0)
                                // { T0 < T == T* < T } <=> не достигли
                                // { T0 < T != T* < T } <=> достигли
                                // { T0 < T != T* < T } <=> { T0 < T == T* >= T }
                                const bool hit = (conditionT0lessT1 ==
(mtxTemperature(indexX, indexY) >= temperature));

                                // Перешагнули?
                                if (hit)
                                {       // Таки да, перешагнули.
                                        // При преодолении порога температура не
изменилась?
                                        // 0.000001 - точность до миллионных.
                                        if (abs(oldTemperature - mtxTemperature(indexX,
```

```
indexY)) < 0.0000001)
                                        {
                                                // Не меняется. Значит уточнить ответ ещё
больше нельзя.
                                                // Прекращаем вычисления, в другие
процессоры отсылаем сигнал останова.
                                                // Запоминаем ответ:
                                                timeAnsver = time;
                                                // Останавливаем все процессоры:
                                                mustContinue = false;
                                        }
                                        else
                                        {
                                                // Нет, при уменьшении шага ответ продолжает
изменяться => уменьшаем шаг ещё сильнее.
                                                // Откатываемся, делим шаг на 2 и повторяем
                                                time -= deltaTime;
                                                deltaTime /= 2;
                                                qx = Qx(deltaTime);
                                                qy = Qy(deltaTime);
                                                qt = Qt(deltaTime);
                                        }
                                }
                        }

                        // Передаём все аргументы от владельца точки остальным
процессорам.
                        MPI_Bcast(allArgs, allArgCount, MPI_DOUBLE, pointOwnerRank,
MPI_COMM_WORLD);
                } // for (...)

                // Ок, закончили. Если ответ у нас, то отсылаем его в последний
процессор.
                // Ах да, делаем это только эти два процессора на самом деле не одни и
тот же.
                // Если один и тот же, то ответ уже вычислен, действий не требуется.
                if (pointOwnerRank != mpiSize - 1)
                {
                        if (pointOwnerRank == mpiRank)     // Передаём
                        {
                                MPI_Send(&timeAnsver, 1, MPI_DOUBLE, mpiSize - 1, 105,
MPI_COMM_WORLD);
                        }

                        if (mpiSize - 1 == mpiRank) // Получаем
                        {
                                MPI_Status status;
                                MPI_Recv(&timeAnsver, 1, MPI_DOUBLE, pointOwnerRank, 105,
MPI_COMM_WORLD, &status);
                        }
                }

        }

        ////////////////////////////////////////////////////////////////////////////
};

////////////////////////////////////////////////////////////////////////////////
}
```