Московский государственный университет имени М.В. Ломоносова
Факультет вычислительной математики и кибернетики
Кафедра суперкомпьютеров и квантовой информатики

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

Разработка функциональных основ системы визуализации информационных графов алгоритмов

Выполнила: студентка 423 группы
Гадиева Т.Р.
Научный руководитель: к.ф-м.н.
Антонов А.С.

Москва, 2023

Содержание

1. Введение

Современное стремительное развитие вычислительной техники влечёт за собой ряд проблем, одной из них является проблема возможная потеря актуальности разработанных пользователем программ с течением времени. Это происходит из-за того, что вместе с изменениями особенностей архитектур вычислительных систем языки программирования становятся всё более зависимыми от этих особенностей. С целью минимизации данной проблемы существует идея исследовать самые подробные детали структуры алгоритмов и программ посредством построения их *тонкой информационной структуры*[1].

В данной работе в качестве инструмента исследования особенностей алгоритмов используется *информационный граф алгоритма*. Он представляет собой ациклический граф, вершины которого соответствуют операциям алгоритма, а дуги - связям по данным между этими операциями. Анализ информации, полученной с помощью исследования таких графов, эффективен с точки зрения выявления возможностей лучшей организации параллельных вычислительных процессов. Таким образом, с целью облегчения процесса исследования информационных графов алгоритмов естественным представляется выбор создания системы 3D-визуализации таких графов с последующим интерактивным выделением особенностей его структуры.

Данная работа является частью создаваемой системы визуализации интерактивных моделей графов алгоритмов по программному коду их референсных реализаций. Под референсной реализацией понимается реализация без каких-либо изменений алгоритма с целью его оптимизации либо использования ресурса параллелизма алгоритма (последовательная реализация).

Система визуализации представляет собой цепочку следующего вида (см. Рис. 1):

Рис. 1. Схема работы системы визуализации информационных графов алгоритмов

Таким образом, разрабатываемая система является двухэтапной. На первом этапе на вход программе поступает описание информационной структуры алгоритма на языке Algolang в формате XML, на основе которого строится некоторое текстовое 3D-модели информационного графа этого алгоритма. На втором этапе по полученному текстовому описанию создаётся интерактивная модель этого графа.

Автоматическая визуализация информационного графа алгоритма, как элемента описания алгоритма, в виде интерактивной 3D-модели с выявлением особенностей структуры графа направлена на решение следующего ряда задач.

- Обеспечение наглядного представления внутренней структуры алгоритма, в том числе и для лиц, не являющихся специалистами в области, в которой применяется описываемый алгоритм.
- Облегчение процесса подробного анализа алгоритма для составления его полного описания с упоминанием всех характерных особенностей как на уровне собственно алгоритма, так и на уровне его конкретных реализаций.
- Освобождение от необходимости воспроизводить визуализацию вручную.

2. Постановка задачи

Целью данной работы является автоматизация создания описания многомерной модели информационного графа любого заданного алгоритма в рамках общей системы визуализации графов алгоритмов. Для достижения данной цели необходимо выполнить следующие задачи:

1. Изучить понятие информационного графа алгоритма, его свойства и способы применения и анализа, изучить способы описания графа алгоритма. Изучить синтаксис языка Algolang, предназначенного для описания информационной структуры алгоритма в виде XML-файла. Изучить необходимые программные библиотеки для работы с форматом XML и вычисления математических выражений, используемых при описании информационных графов алгоритмов.
2. Разработать способы получения и обработки необходимой информации об алгоритме из описания на языке Algolang с целью использования его в качестве входных данных для данной работы.
3. Разработать формат выходных данных, на основе которого информационный граф может быть легко визуализирован.
4. Реализовать алгоритм, который на основе информации, полученной из описания алгоритма на языке Algolang, строит описание модели многомерного графа, по которой он может быть визуализирован. Модель должна представлять собой описание расположения вершин и дуг графа определённым образом, а также их свойства, в некотором формате, который может быть использован в качестве входных данных для второй части общей системы визуализации графов алгоритмов.

3. Обзор понятий предметной области

3.1. Обзор графов, применимых к анализу работы программ

Графы представляют собой математическую абстракцию реальной системы, объекты которой обладают взаимными связями. Они широко используются как средство анализа информации и зависимостей любого рода. Таким образом, в рамках технической предметной области существуют характерные виды графов для исследований, примеры которых будут приведены ниже.

Граф потока управления — множество всех возможных путей исполнения программы, представленное в виде графа. Блоки графа представляют собой прямолинейные участки кода, а дуги между ними — передачу управления следующему участку программы[2](см. Рис. 2[2]). Граф потока управления используется в статическом анализе, а также в приложениях компилятора, так он наглядно отражает поток внутри программного модуля.

```
Procedure TriangleType
 S1  char *returnString = "not triangle\n";
 S2  if (((i + j)> k)&&((j + k)> i)&&((k + i)> j))
 S3     if ((i + j + k) == (3 * i))
 S4        returnString =  "equilateral\n";
        else
 S5        if ((i != j) && (j != k) && (k != i))
 S6           returnString = "scalene\n";
           else
 S7           returnString = "isosceles\n";
           endif
        endif
     endif
 S8  return returnString;
```

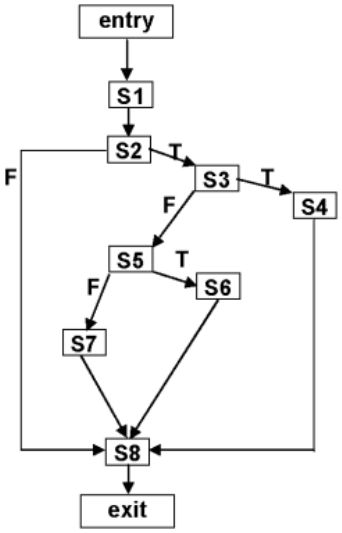

Рис. 2. Процедура определения типа треугольника по заданным значениям длин и граф потока управления, составленный по данной процедуре

Граф зависимостей — это ориентированный граф, представляющий зависимости нескольких объектов друг относительно друга. Они являются полезными инструментами для управления сложными цепочками вычислений и широко используются для обозначения любого рода программных[3][4] и алгоритмических[5] зависимостей. Например, графы зависимостей применяются в компиляторах для анализа взаимосвязей элементов программы с целью проведения её оптимизации[6]. В таких графах вершины отражают низкоуровневые

операции, дуги — зависимость от доступа к памяти и зависимости между определением и использованием некоторой переменной(см. Рис. 3[6]).

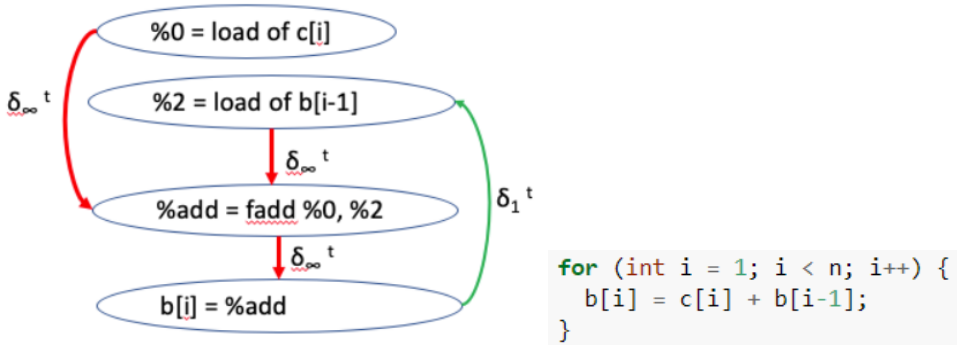

Рис. 3. Пример построения графа зависимостей для приведённого фрагмента программы (зелёная дуга - зависимость от доступа к памяти, красная дуга - зависимость использования переменной от её определения)

Граф-схема алгоритма — это ориентированный связный граф, задающий последовательность выполнения операций данного алгоритма. Он состоит из входной, выходной, операторных и условных вершин. Операторной называется вершины, сопоставляются с одной или несколькими микрооперациями, а условной — вершина, которой сопоставляется некоторое логическое условие (см. Рис. 4). Подобное представление алгоритмов используется при построении систем логического управления (логический контроллер)[7], реализующих заданные управляющие алгоритмы, в задачах распараллеливания вычислений и т. д.

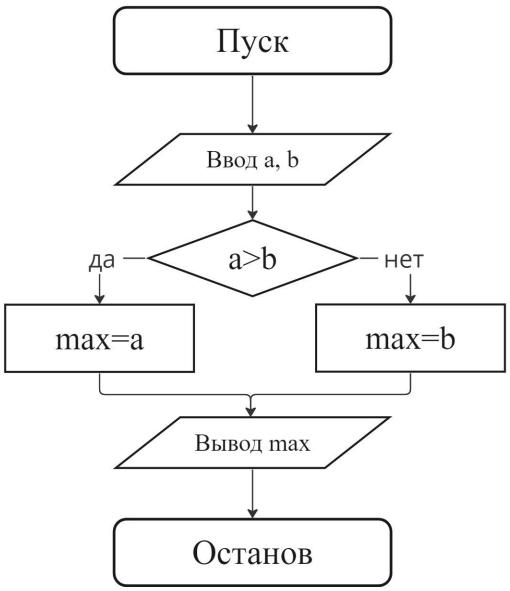

Рис. 4. Пример граф-схемы для алгоритма определения максимума из двух чисел

В рамках данной работы будет рассмотрен граф, позволяющий исследовать тонкую информационную структуру алгоритмов.

3.2. Информационный граф

3.2.1. Понятие информационного графа

Свойства и характеристики, по которым сейчас исследуются алгоритмы, например число операций, объём используемой памяти и точность, применимы при рассмотрении именно последовательной их реализации, подход к исследованию же параллельных алгоритмов требует дополнительного анализа их особенностей. Одним из инструментов исследования параллельных алгоритмов является граф информационной зависимости реализации алгоритма при фиксированных входных данных, или, кратко, информационный граф алгоритма.

Информационный граф — это ациклический ориентированный граф, вершины которого соответствуют операциям алгоритма, а дуги — связям по данным между этими операциями. Две вершины связываются дугой, если вторая использует данные, вычисленные в первой.

Его ацикличность следует из того, что в любых программах реализуются только явные вычисления и никакая величина не может определяться через саму себя, в том числе и рекурсия реализует однотипные, но разные операции.

Анализ информационных графов алгоритмов играет важную роль в исследовании внутреннего параллелизма алгоритмов. Под внутренним параллелизмом подразумевается заложенный в их последовательной реализации потенциал к распараллеливанию. Также он даёт возможность определить множества независимых друг от друга операций, найти подходящее распределение операций по процессорам вычислительной системы, обнаружить узкие места и т.д.[1]

3.2.2. Свойства информационного графа

Одним из важнейших свойств информационного графа алгоритма является его представление в виде ярусно-параллельной формы.

Ярусно-параллельная форма[8] — это представление графа алгоритма, в котором:

- все вершины разбиты на пронумерованные подмножества ярусов;
- начальная вершина каждой дуги расположена на ярусе с номером меньшим, чем номер яруса конечной вершины;
- между вершинами, расположенными на одном ярусе, не может быть дуг.

Высота ЯПФ — это число ярусов. Минимальной высотой всех возможных ЯПФ графа является его критический путь. Канонической ярусно-параллельной формой называется ЯПФ, высота которой на единицу больше длины критического пути, а все входные вершины расположены на первом ярусе. Для заданного графа его каноническая ЯПФ единственна. Высота канонической ЯПФ соответствует параллельной сложности алгоритма.

Для ЯПФ графа алгоритма важным является тот факт, что операции, которым соответствуют вершины одного яруса, не зависят друг от друга (не находятся в отношении связи), и поэтому заведомо существует параллельная реализация алгоритма, в которой они могут быть выполнены параллельно на разных устройствах вычислительной системы.

3.2.3. Класс программ, представимый в виде информационного графа

В работе “Параллельные вычисления” 2002 г. В.В.Воеводина и Вл.В.Воеводина[1] дано определение так называемого линейного класса алгоритмов, для которых возможно их представление в виде информационного графа, в терминах программ, написанных на языке программирования Fortran:

“Всюду в дальнейшем, если не сделано специальных оговорок, будем считать, что алгоритм записан с помощью следующих средств языка:

1. В программе может использоваться любое число простых переменных и переменных с индексами.
2. Единственным типом исполнительного оператор может быть оператор присваивания, правая часть которого есть арифметическое выражение; допускается любое число таких операторов.
3. Все повторяющиеся операции описываются только с помощью циклов DO; структура вложенности циклов может быть произвольной; шаги изменения параметров циклов всегда равны +1; если у цикла нижняя граница больше верхней, то цикл не выполняется.
4. Допускается использование любого числа условных и безусловных операторов перехода, передающих управление “вниз” по тексту; не допускается использование побочных выходов из циклов
5. Все индексные выражения переменных, границы изменения параметров циклов и условия передачи управления задаются, в общем случае, неоднородными формами, линейными как по параметрам циклов, так и по внешним переменным программы; все коэффициенты линейных форм являются целыми числами.

6. Внешние переменные программы всегда целочисленные, и вектора их значений принадлежат некоторым целочисленным многогранникам; конкретные значения внешних переменных известны только перед началом работы программы и неизвестны в момент её исследования.

Программы, удовлетворяющие описанных условиям, будем называть линейными или принадлежащими линейному классу.”

Также в этой работе изложены примеры приёмов, позволяющих сводить к линейному фрагменты программы, формально не принадлежащие к этому классу.

В рамках данной работы рассматривается так называемый расширенный линейный класс, построенный на основе описанного выше линейного класса. Расширенный линейный класс снимает ограничения на линейность выражений, описанных в пункте 5.

3.2.4. Применение информационного графа

На данный момент понятие информационного графа алгоритма активно используется в рамках проекта Algowiki[9] - открытой библиотеки по свойствам алгоритмов и особенностям их реализации на различных программно-аппаратных платформах - для осуществления детального анализа свойств параллельных алгоритмов[10].

3.3. Язык описания информационной структуры алгоритмов

В качестве инструмента для создания формального описания особенностей алгоритма в качестве входных данных используется язык Algolang[11]. Он предназначен для достаточно компактной записи информационной структуры фрагмента программы, по которой затем можно построить визуализацию его информационного графа. Формальная структура описания языка представлена в расширенной форме Бэкуса-Наура (см. Рис. 5[11]).

Вся информация об алгоритме заключается в тег <algo>. Внутри него первым следует перечисление параметров алгоритма под тегами <param> с такими атрибутами, как name — имя параметра, type — тип параметра, который может быть представлен целым числом или числом с плавающей точкой, и, опционально, value — значение параметра. Весь список параметров заключён в тег <params>. После него следует набор “блоков операций” под тегами <block>, описывающих опорные многогранники[1] для операторов присваивания заданного алгоритма. Атрибутами каждого блока является, опционально, id — идентификатор блока и dims — размерность блока.

Описание каждого опорного многогранника задаётся:

1. Границами изменения параметров его опорного цикла. Они перечислены под тегом <arg> и задаются такими атрибутами, как name — имя параметра и val — значение параметра.
2. Линейным пространством итераций. Оно указано под тегом <vertex> с такими атрибутами, как condition — условие, определяющее вхождение или невхождение текущей точки в опорную область и type — тип этой точки.
3. Зависимостями по данным. Они описаны под тегом <in> с указанием координат точки, от которой зависимость строится к текущей, под атрибутом src, и, если точка указана из опорного цикла другого многогранника, id “блока операций”, соответствующего этому многограннику.

```
<Algorithm> ::== <algo> <Parameters> { <Block> } </algo>
<Parameters> ::== <params> { <Parameter> } </params>
<Parameter> ::== <param name = “<Name>” type = “<Type> [value = “<Number>”]></param>
<Type> ::== int | float
<Block> ::== <block [id = “<Number>“] dims = “<Number>”>{Argument}{Vertex}</block>
<Argument> ::== <arg name = “<Name>” val = “<RegExp>..<RegExp>”></arg>
<Vertex> ::== <vertex condition = “<RegExp>” type = “<Number>”>{Source}</vertex>
<Source> ::== <in [bsrc = “<Number>“] src = “<RegExp>{,<RegExp>}”></in>
```

Рис. 5. Основные конструкции языка Algolang

То есть, описание информационной структуры на примере алгоритма умножения матрицы на вектор (см. Рис. 6 и Рис. 7[9]) состоит из:

1. Перечисления параметров, используемых в работе и их значений, соответствующих программным переменным алгоритма.
2. Списка “блоков операций” разной размерности, соответствующих циклам алгоритмов и их вложенности.
3. Описание опорных циклов многогранников, соответствующих пространству итераций цикла алгоритма. (внутри каждого блока)
4. Список условий вхождения точек в опорную область, соответствующий ограничениям, накладываемых на переменные, итерируемые во время выполнения цикла алгоритма. (внутри каждого блока)
5. Список информационных зависимостей для текущей вершины, соответствующий зависимости текущей операции от значений, вычисленных на предыдущих шагах цикла алгоритма. (внутри каждого условия существования вершины)

```
int n = 5;
int m = 4;
for(int i = 1; i <= m; i++)
  for(int j = 1; j <= n + 1; j++)
    vec_out[i] += matrix[i][j] * vec _in[j];
```

Рис. 6. Алгоритм умножения матрицы на вектор на языке программирования C

```
<algo>
    <params>
        <param name="n" type="int" value="5"></param>
        <param name="m" type="int" value="4"></param>
    </params>

    <block dims="2">
        <arg name="i" val="1..m"></arg>
        <arg name="j" val="1..n+1"></arg>

        <vertex condition="j>1" type="2">
            <in src="i,j-1"></in>
        </vertex>
    </block>
</algo>
```

Рис. 7. Описание информационной структуры алгоритма умножения матрицы на вектор на языке AlgoLang

4. Обзор существующих решений обработки файлов формата XML

4.1. Синтаксические анализаторы SAX и DOM

Существует два основных подхода анализа файлов формата XML - DOM и SAX API (от англ. Application Programming Interface).

DOM анализаторы осуществляют обработку XML-файла, предварительно загрузив данные этого файла в программу в виде DOM (от англ. Document Object Model) дерева[12]. Оно является представлением HTML-документа в виде дерева тегов (см. Рис. 8).

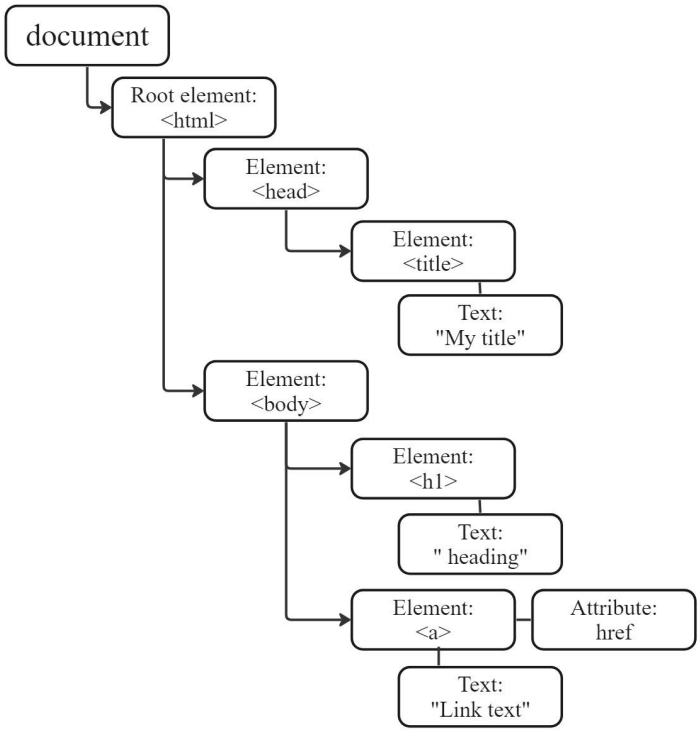

Рис. 8. Пример структуры DOM дерева для HTML файла

SAX (от англ. Simple API for XML) анализаторы, в свою очередь, осуществляют поточную событийную обработку XML-документов, не загружая данные во внутренние структуры программы[13]. Таким образом, более подробно разница между данными анализаторами представляет собой следующее[14]:

1. Способ работы

Первое и основное различие между способами разбора файлов DOM и SAX заключается в том, как они работают. Синтаксический анализатор DOM загружает в память полный XML-файл и создает древовидное представление XML-документа, в то время как

SAX представляет собой анализатор XML, основанный на событиях, и не загружает весь XML-документ в память.

Соответственно, формат SAX более удобен для обработки файлов больших объемов.

2. Размер XML-файлов

Для небольших и средних XML-документов DOM работает намного быстрее, чем SAX, благодаря работе в памяти.

3. Тип интерфейсов программного взаимодействия (API)

Анализатор DOM создает дерево в памяти и позволяет получать доступ к элементам, в то время как тип API SAX основан на потоковой передаче.

4. Простота использования

DOM относительно проще в использовании, чем парсер SAX в смысле применения программных интерфейсов.

5. Возможности доступа к данным

DOM подход позволяет использовать доступ к элементам по заданному пути, что недоступно в синтаксическом анализаторе SAX. В большинстве случаев DOM анализаторы используются из-за данной возможности.

6. Направление

SAX — это анализатор, работающий только в прямом направлении, что означает, что нет возможности вернуться к пройденным данным, в то время как DOM позволяет вам двигаться в любом направлении.

Таким образом, SAX анализаторы лучше подходят для потоковой обработки событий в больших XML-файлах, а DOM анализаторы для полной обработки маленьких XML-файлов.

4.2. Примеры библиотек для обработки XML-файлов в стиле DOM

В рамках данной работы в качестве инструмента для разработки необходимо использовать DOM подход к обработке XML-файлов, поэтому рассмотрены различные библиотеки с открытым кодом, предоставляющие именно такие возможности. Наиболее используемыми для языка C++ являются следующие из них[15]:

1. RapidXML
2. PugiXML
3. TinyXML

Как следует из названия, RapidXML в основном ориентирован на уменьшение времени выполнения и представляет собой стабильный и быстрый парсер, работающий со скоростью приближающейся к скорости работы функции strlen (функция подсчёта длины в байтах), выполняемой на тех же данных[16]. Данная библиотека поддерживает кодировки UTF-8 и частично UTF-16, UTF-32 и использует небольшой объём памяти для своей работы.

PugiXML — это библиотека, которая предоставляет программный интерфейс DOM с широкими возможностями обхода/модификации, чрезвычайно быстрый синтаксического анализатор, и возможность доступа к данным по заданному пути[17]. Также доступна полная поддержка Unicode с преобразованиями между различными кодировками (которые происходят автоматически во время синтаксического анализа/сохранения).

TinyXML — это минимальная версия для анализа данных XML-файлов, не такая быстрая, как две предыдущие. Данная библиотека была создана из соображений простоты в использовании и освоении, поэтому хорошо подходит в том случае, если большая скорость выполнения не является приоритетным фактором при выборе инструментов для работы[18].

Также для приведённых в работе и других библиотек разбора XML-файлов, в том числе работающих в силе SAX, проведены тесты производительности и потребления памяти и приведены диаграммы с результатами по отношению к результатам работы библиотеки PugiXML[19]. Из них следует, что PugiXML является одним из лучших решений по времени работы и достаточно хорошим по использованию памяти. Библиотека RapidXML также показывает хорошие результаты, очень близкие к результатам PugiXML, в то время как результаты TinyXML значительно хуже, особенно по показателям времени выполнения.

4.3. Компилятор привязки данных

Помимо SAX и DOM анализаторов существует способ обработки при помощи компилятора привязки данных XSD Schema. Такой способ был предложен компанией CodeSynthesis, как альтернативное решение обработки XML-файлов с некоторыми преимуществами по сравнению с традиционными подходами[20]. При наличии спецификации экземпляра XML (XML-схема) он генерирует классы C++, представляющие заданный словарь, а также код синтаксического анализа и сериализации XML. По сравнению с API-интерфейсами, такими как DOM и SAX, привязка данных XML позволяет пользователю получать доступ к данным в XML-документах, используя сгенерированный доменный словарь, а не общие элементы, атрибуты и текст.

XSD Schema поддерживает два подхода сопоставления XML-схемы и C++: C++/Tree, работающий наподобие DOM анализатора, то есть представляющий информацию XML-файла в виде древовидной модели в памяти программы, и C++/Parser, представляющий данные в виде иерархии событий синтаксического анализа, подобно SAX. Выбор привязки может осуществляться по таким же доводам, как и выбор между DOM и SAX подходами обработки XML-файлов, поскольку они имеют аналогичные особенности и преимущества.

Таким образом, традиционные API-интерфейсы доступа к XML имеют ряд недостатков, которых компилятор привязки данных XSD Schema позволяет избежать.

Недостатки традиционных API — DOM и SAX	*Преимущества, предоставляемые компилятором привязки данных XSD Schema*
1. Необходим объёмный связующий код, идентифицирующий и преобразующий информацию, закодированную в XML.	1. Сгенерированный код скрывает всю сложность, связанную с разбором и сериализацией XML.
2. Управление потоком на основе строк откладывает обнаружение ошибок до времени выполнения.	2. Статистическая типизация помогает обнаруживать ошибки во время компиляции, а не во выполнения.
3. Полученные приложения трудно отлаживать, изменять и поддерживать.	3. Автоматическая генерация кода сводит к минимуму усилия, необходимые для адаптации приложения к изменениям в структуре документа.

4.4. Дополнительные решения для обработки XML-файлов

Помимо способов работы напрямую с XML-файлами существует возможность осуществления тех же манипуляций (доступ/анализ/обработка) с данными, только предварительно осуществив конвертацию XML формата к JSON (от англ. JavaScript Object Notation) — стандартному текстовому формату обмена данными. В таком случае для обработки данных уже используются библиотеки для работы с форматом JSON, поэтому рассмотрим самую используемую из них — RapidJSON.

Данная библиотека предлагает пользователю следующий ряд возможностей и преимуществ[21]:

1. Небольшая, но полноценная. Она поддерживает API в стиле SAX и DOM.
2. Быстро работает. Её производительность может быть сравнима с функцией strlen() на тех же данных.
3. Удобна по памяти. Каждое значение JSON занимает ровно 16 байт для большинства 32/64-разрядных машин (исключая текстовую строку). По умолчанию библиотека использует быстрый распределитель памяти, поэтому синтаксический анализатор компактно выделяет память во время своей работы.
4. Поддерживает Unicode. RapidJSON поддерживает UTF-8, UTF-16, UTF-32, а также их внутреннее обнаружение, проверку и транскодирование.

Также важным в рамках данной работы оказался способ представления данных при работе с программным интерфейсом DOM. RapidJSON работает с сущностью Value (значение), которая может быть двух типов - Object (объект) и Array (массив). С точки зрения исходного XML-файла — все его теги являются объектами и все одноимённые теги объединяются в массивы. Таким образом, древовидная структура, в виде которой данные хранятся в программе, у библиотеки RapidJSON отличается от структур библиотек, работающих напрямую с форматом XML (см. Рис. 9[21]). И тогда обход этой древовидной структуры, а в частности переход между разноимёнными тегами, расположенными на одной глубине, отличается и представляется более удобным и безопасным с точки зрения доступа к памяти.

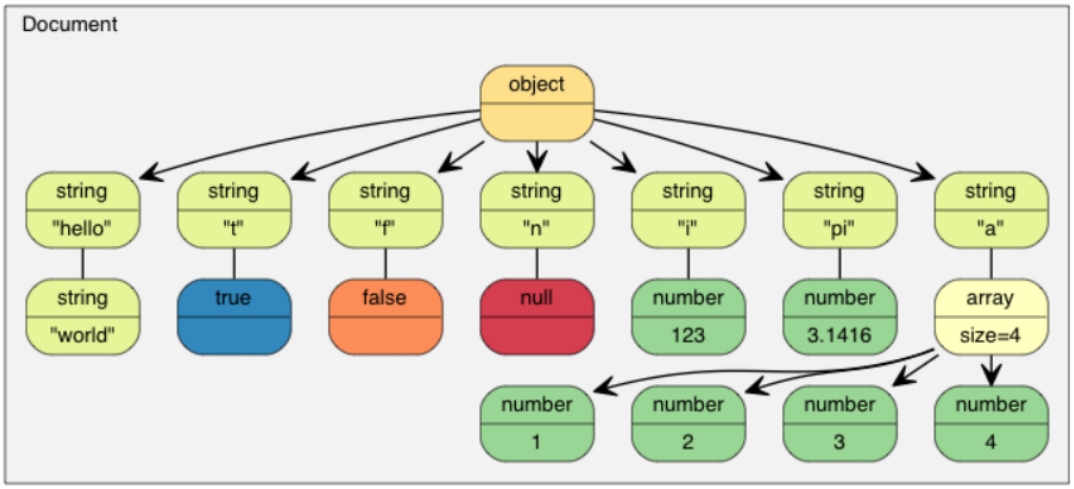

Рис. 9. Формат DOM структуры данных, использующийся программной библиотекой RapidJSON

5. Обзор библиотек для вычисления значений математических выражений

В рамках данной работы существует необходимость вычисления и оценки математических выражений, с помощью которых описывается информационная структура алгоритма. Для этих целей на языке C++ существует набор внедряемых в проекты сторонних библиотек с открытым кодом. Самыми используемыми и подходящими в рамках данной работы являются:

1. ExprTk
2. muParser
3. Math Expression Toolkit Library (METL)

5.1. Библиотека ExprTk

ExprTk является наиболее полной и используемой библиотекой для разбора и оценки математических выражений из приведённых выше. Механизм синтаксического анализа поддерживает множество форм функциональной и логической семантики обработки и легко расширяется. Документация библиотеки представляет развернутый список возможностей данной библиотеки и большое количество примеров, что позволяет быстро освоить её использование[22]. Библиотека ExprTk позволяет работать со скалярными, строковыми и векторными типами данных с числовыми типами Float, Double и MPFR (от англ. multiple-precision floating-point) и поддерживает обработку следующих операций и функций[23]:

1. Арифметические операции и операторы присваивания
2. Равенства и неравенства
3. Логические операции
4. Функции общего назначения (abs, avg, ceil, exp, log, max, min и т.д.)
5. Тригонометрические функции
6. Операции со строками (=, !=, in, +=, +, [] и т.д.)
7. Структуры управления (if, if-else, switch, while и т.д.)

ExprTk считается самым быстрым решением для вычисления стандартных операций со скалярными значениями. Возможность замера времени работы включена в функционал библиотеки, а некоторые примеры замеров приведены в самой документации к ней (см. рис. 10[22]).

```
./exprtk_benchmark test.txt 1000000
Expr 1 of 5 86.563 ns 8656300ns (296417859.6) '1/sqrt(2x)*e^(3y)'
Expr 2 of 5 40.506 ns 4050600ns (296417859.6) '1/sqrt(2x)*exp(3y)'
Expr 3 of 5 14.248 ns 1424799ns (    44267.2) '1/sqrt(2x)'
Expr 4 of 5 88.840 ns 8884000ns (615985286.9) 'e^(3y)'
Expr 5 of 5 29.267 ns 2926699ns (615985286.9) 'exp(3y)'
[*] Number Of Evals:        5000000
[*] Total Time:             0.260sec
[*] Total Single Eval Time: 0.000ms
```

Рис. 10. Результаты замеров времени работы библиотеки ExprTk при обработке скалярных выражений

Принцип работы данной библиотеки заключается в последовательном использовании следующих трёх её компонентов: таблица переменных, экземпляр выражения и анализатор. Сначала происходит добавление имён переменных и их значений, затем выражение регистрируется с созданной ранее таблицей переменных, а после - экземпляр выражения передаётся анализатору. В результате обработки, если она прошла успешно, по данному выражению создаётся абстрактное синтаксическое дерево[24], которое и позволяет оценить его значение. Таким образом, математическое выражение проходит следующую цепочку преобразований в рамках работы с библиотекой ExprTk (см. Рис. 11):

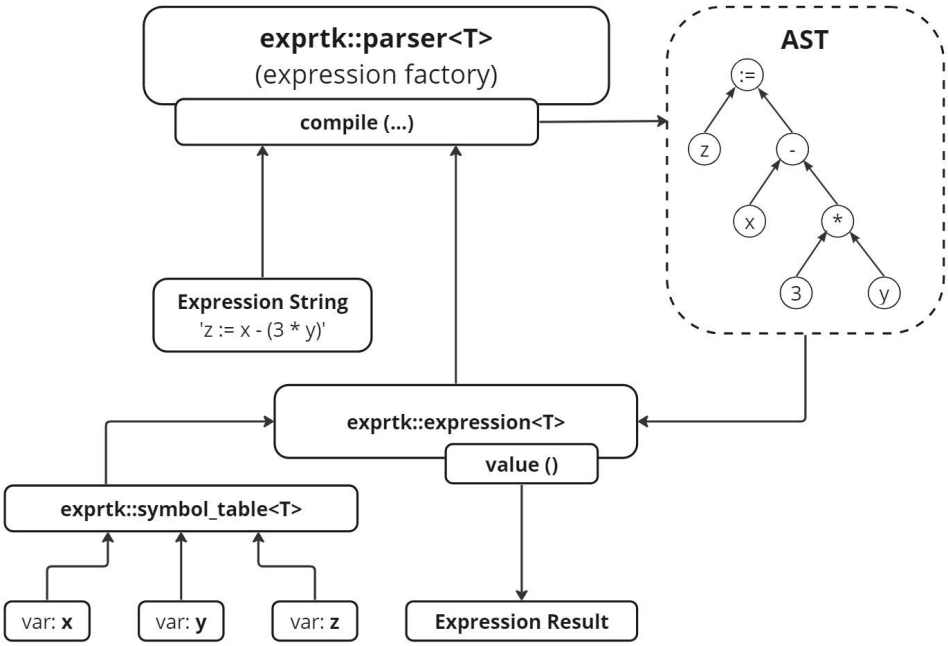

*Рис. 11. Пример работы библиотеки ExprTk для нахождения значения выражения z:= x - (3*y)*

5.2. Серия библиотек muParser

MuParser — это серия расширяемых высокопроизводительных библиотек, предоставляющих пользователю разные возможности в зависимости от его нужд (см. Рис. 12[25]). Для достижения большей скорости работы со скалярными значениями с ограничением на количество и тип используемых параметров и точность вычислений используется библиотека muparserSSE. Для преодоления данных ограничений с увеличением времени работы используется muparser. Для работы с векторными и строковыми типами данных и комплексными числами применима версия библиотеки muparserX.

Parser	Data types				Precision	User defined operators			User defined functions		Localization	Licence	Performance (Expr. per second)
	complex	scalar	string	vector		Binary	Postfix	Infix	Strings as parameters	Arbitrary number of parameters			
muparser	✗	✓	✓ (1)	✗	double	✓	✓	✓	✓	✓	✓	BSD 2-Clause	~ 10.000.000
muparserSSE	✗	✓	✗	✗	float	✓	✓	✓	✗	max. 10	✓	BSD 2-Clause	~ 20.000.000
muparserX	✓	✓	✓	✓	double	✓	✓	✓	✓	✓	✗	BSD 2-Clause	~ 1.600.000

Рис. 12. Возможности библиотек обработки математических выражений серии muParser

Библиотеки серии muParser предоставляют следующие возможности для работы:

1. Типы данных: двойные, целые, сложные, логические, строка, массив.
2. Расширяемы для пользовательских операторов (бинарных, инфиксных или постфиксных)
3. Расширяемы для пользовательских функций с произвольным количеством аргументов функции
4. Поддерживают неограниченного количества переменных и констант
5. Считывают двоичные, шестнадцатеричные, комплексные, целые и строковые значения из выражений и может быть расширен для чтения пользовательских значений.
6. Большое разнообразие предопределенных операторов, функций и констант.
7. Написаны в стандартном совместимом коде C++ без внешних зависимостей.
8. Версия muparser позволяет распараллелить оценку выражений с помощью OpenMP[26]
9. Работают как на 32-разрядных, так и н 64-разрядных архитектурах
10. Поддерживают сборку с помощью компиляторов clang и gcc

5.3. Библиотека METL

Math Expression Toolkit Library — небольшая библиотека для стандарта языка c++14 для анализа математических выражений. Она спроектирована таким образом, чтобы быть достаточно гибкой, но и то же время эффективной. Под гибкостью подразумевается, что выражения могут использовать все типы переменных с разумным поведением (например, полезно для работы с векторами и матрицами) и при этом добавление и редактирование операторов и функций очень просто.

Библиотека METL представляет собой набор заголовочных файлов и единственной зависимостью для неё является библиотека PEGTL, библиотека шаблонов для парсинга грамматики выражений, поэтому METL легко интегрируется в проекты.

Основные достоинства данной библиотеки[27]:

1. Собрать единожды, вызывать часто: METL компилирует входное выражение в std::function, которое можно вызывать так часто, как того требует задача
2. Исключительная гибкость: существует возможность создать любой оператор или функцию и использовать их с любым типом данных
3. Сборка доступна с помощью использования компиляторов msvc, gcc и clang
4. Поддерживает обработку выражений, включающих литералы с суффиксами или без них, переменные и константы с произвольными именами
5. Интеллектуальная настройка значений по умолчанию, которая распознает стандартные типы (с целым числом и с плавающей запятой)
6. Возможность использования бинарных операторов, унарных операторов, функций
7. Встроенный оператор присваивания новым или существующим переменным и константам

5.4. Сравнение библиотек

Для библиотек осуществлено сравнение с помощью тестов производительности (бенчмарка), разработанных для тестирования корректности вычислений и скорости работы библиотек для парсинга математических выражений, написанных на языке C++ и работающих по принципу POEM (от англ. Parse Once Evaluate Many times)[28]. Ниже представлены характеристики системы, на которой производились запуски тестов (см. Таблицу 1)

Таблица 1. Характеристики системы, на которой производились запуски тестов производительности для библиотек обработки математических выражений

CPU Name	Intel(R) Xeon(R) CPU E5-2687W 0 @ 3.10GHz	Build	32-bit
Num. cores	8	L1D assoc.	8-way
Num. logical	16	L2 assoc.	8-way
Total logical	16	L3 assoc.	20-way
L1 D cache	32 KB	L1D line size	64 bytes
L1 I cache	32 KB	L2 line size	64 bytes
L2 cache	256 KB	CPU clock	3092 MHz

По результатам замеров времени вычислений лидерами среди библиотек для парсинга математических выражений стабильно становились muparserSSE и ExprTkFloat, работающие с типом данных Float, а с учётом также точности вычислений - ExprTk, работающая с типом данных Double (см. Рис. 13[28])

```
Expression 411 of 6617: "10^log(3+b)"; Progress: #############
[01] ExprTk               (   55.653 ns,      44.530608078220737411, 3514652.345715506002306938)
[02] muparserSSE          (   58.024 ns,      44.530609130859375000, 3514652.156829833984375000)
[03] ExprTkFloat          (   58.647 ns,      44.530609130859375000, 3514652.156829833984375000)
[04] atmsp 1.0.4          (   58.938 ns,      44.530608078220737411, 3514652.345715506002306938)
[05] METL                 (   60.547 ns,      44.530608078220737411, 3514652.345715506002306938)
[06] muparser 2.2.4       (   61.203 ns,      44.530608078220737411, 3514652.345715506002306938)
[07] FParser 4.5          (   66.560 ns,      44.530608078220751622, 3514652.345715506002306938)
[08] TinyExpr             (   66.723 ns,      44.530608078220737411, 3514652.345715506002306938)
[09] muparser 2.2.4 (omp) (   67.292 ns,      44.530608078220737411, 3514652.345715506002306938)
[10] MTParser             (   69.176 ns,      44.530608078220737411, 3514652.345715506002306938)
[11] MathExpr             (   80.103 ns,      44.530608078220737411, 3514652.345715506002306938)
[12] Lepton               (  164.500 ns,      44.530608078220737411, 3514652.345715506002306938)
[13] muparserx            (  230.696 ns,      44.530608078220737411, 3514652.345715506002306938)
```

Рис. 13. Результаты замеров времени работы программных библиотек для вычисления выражения 10^log(3+b)

(по столбцам указаны: имя библиотеки, среднее время для однократного вычисления выражения, результат вычисления, сумма результатов N-кратного вычисления)

Также в качестве результатов тестирования автором бенчмарка приведена диаграмма, отображающая результаты выполнения в секунду на тестах с набором случайных математических выражений (см. Рис. 14[28]). В результатах отражены только библиотеки,

работающие с типом Double, запуск теста включал 5000000 итераций на выражение на процессоре Intel Xeon E5-2687W 3 ГГц. Вертикальная мера на диаграмме — достигнутая точность вычислений, горизонтальная — это длина выражения в байтах.

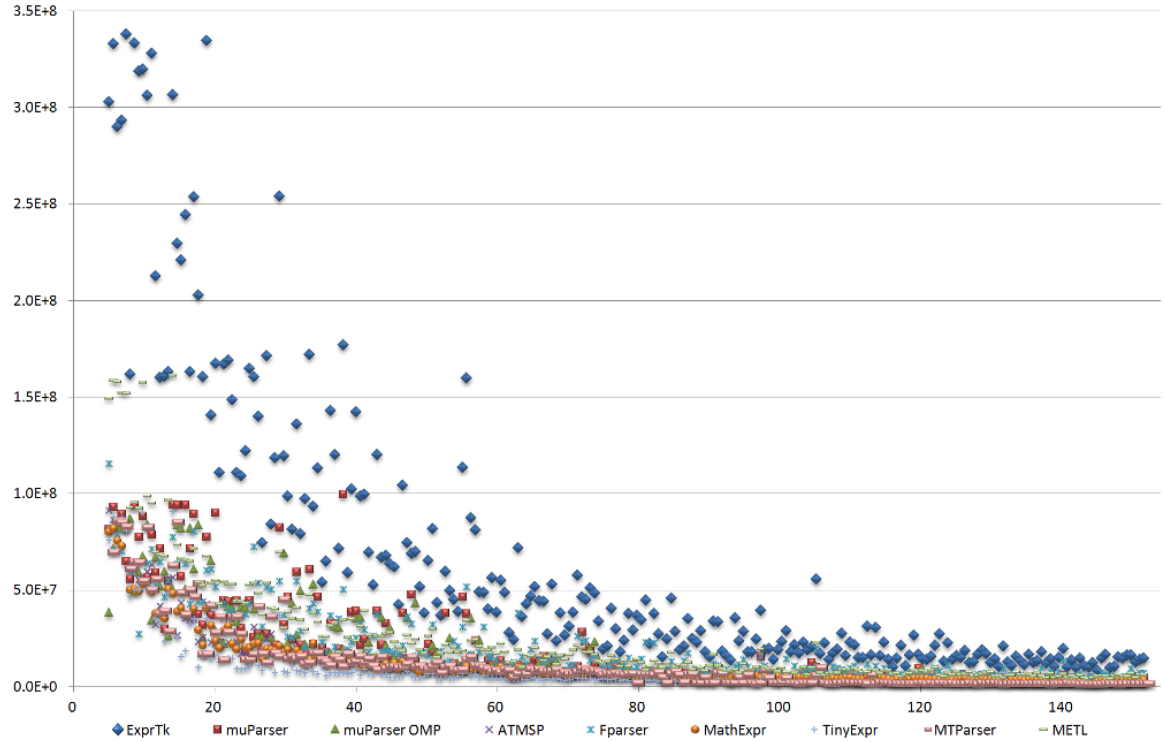

Рис. 14. Результаты работы программных библиотек в секунду, полученные при запуске теста производительности для случайных математических выражений

6. Программная реализация

Для начала сделаем несколько оговорок. В процессе описания работы разработанной программы используемое понятие ярусно-параллельной формы информационного графа будет подразумевать именного каноническую ЯПФ этого графа. Информационный граф алгоритма, построенный в результате работы программы не является таковым в классическом его понимании, поскольку содержит в себе также и вершины с входными данными, что противоречит определению, в котором вершинами являются только операции алгоритма. То есть понятие графа, используемое в рамках реализованной программы, является как бы надграфом, образованным путём добавления к графу алгоритма вершин с входными данными.

Также существует ограничение, накладываемое на входные данные, а именно для блоков алгоритма(тег <block> на языке Algolang), описывающих опорные многогранники его графа, наложено ограничение не более чем трёхмерной размерности.

6.1. Схема работы программы

По результатам данной работы была реализована программа, создающая описание модели информационного графа алгоритма на основе описания его информационной структуры.

Из соображений возможности масштабирования данного проекта и увеличения скорости выполнения разработка программы осуществлялась на языке C++. Процесс работы итоговой программы описан далее.

В качестве входных данных разработанная программа принимает описание алгоритмов на языке Algolang в формате XML. Следующим этапом происходит конвертация исходного файла в формат JSON. После этого при помощи средств библиотеки RapidJSON в памяти создаётся древовидная структура DOM, по которой возможен доступ ко всем данных первоначального XML файла. По данной структуре производится обход с целью сбора всей информации о графе в программные классы с выполнением некоторых промежуточных вычислений для удобства использования, а именно сразу при обходе осуществляется вычисление диапазонов значений пространства итераций. Для подсчёта значений данных выражений используется библиотека ExprTk. Вычисление всех имеющихся в описании алгоритма выражений на данном этапе невозможно, поскольку они обладают зависимостью от текущих значений пространства итераций и будут вычисляться в дальнейшем. Такие выражения при обходе сохраняются целиком в виде строк в качестве программных элементов классов. После сбора всех данных входного файла и проведения предварительной их

обработки запускается "основной цикл". Этот цикл выполняет содержательную работу всей программы. Внутри него происходит проход по всему пространству итераций, осуществляются вычисления, зависимые от текущих его значений, происходит формирование координат вершин, определение их уровня ярусно-параллельной формы и определение вершин, связанных информационной дугой.

По результатам работы этого этапа формируются списки объектов классов вершин и дуг с описанием их свойств. Далее на основе этой информации формируется JSON файл в качестве формата выходных данных. Этот файл, в свою очередь, может быть использован как входные данные для второй части общей системы визуализации информационных графов алгоритмов (см. Рис. 15).

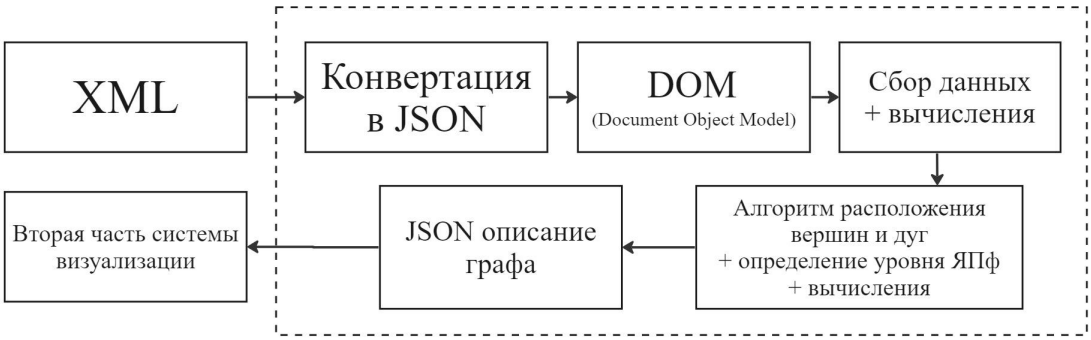

Рис. 15. Схема работы разработанной программы

6.2. Основные этапы работы программы

6.2.1. Конвертация в JSON

На первом этапе работы происходит мгновенная конвертация XML-файла в JSON с помощью небольшой библиотеки xml2json. Таким образом, ошибкой, обрабатывающей этим этапом, может быть только неверная структура XML-файла.

6.2.2. Сбор входных данных

На этапе сбора входных применяются средства библиотеки RapidJSON, а именно программный интерфейс DOM, поскольку требуется сбор полной информации небольшого файла, для чего не применяется подход SAX. Данная библиотека была выбрана в качестве инструмента для работы из-за некоторых особенностей формата входного. В большинстве случаев в структуре XML-файлов одноимённые теги внесены под отдельный общий тег, и если на одной глубине находятся разноимённые теги, то тегов каждого имени не более, чем один. В случае же с языком Algolang допускается наличие нескольких одноимённых тегов на одном уровне с тегами других имён. При использовании библиотек, работающих с доступом

к данным напрямую в XML-файлах возникает проблема при обработке таких ситуаций. Поскольку в таких библиотеках обход построенной в процессе работы древовидной структуры данных осуществляется с помощью указателей, работа с ними не была безопасной, возникали ошибки доступа к памяти по пустым указателям в ситуациях, описанных выше. В связи с этим было принято решение изменить формат входных данных, чтобы использовать средства библиотеки RapidJSON. В случае программного интерфейса DOM данной библиотеки - древовидная структура имеет другой формат, который помогает избежать поставленной проблемы. Библиотека работает с так называемой сущностью Value (значение), которая может являться Object (объектом) или Array (массивом). Каждый тег XML файла представляет собой объект с доступом к его атрибутам, и одноимённые теги объединяются в массив объектов с определённым размером. Таким образом, в самой структуре DOM в памяти программы одноименные и разноимённые теги заведомо разделяются по способу хранения, что облегчает доступ к ним и делает его более безопасным(см. Рис. 16).

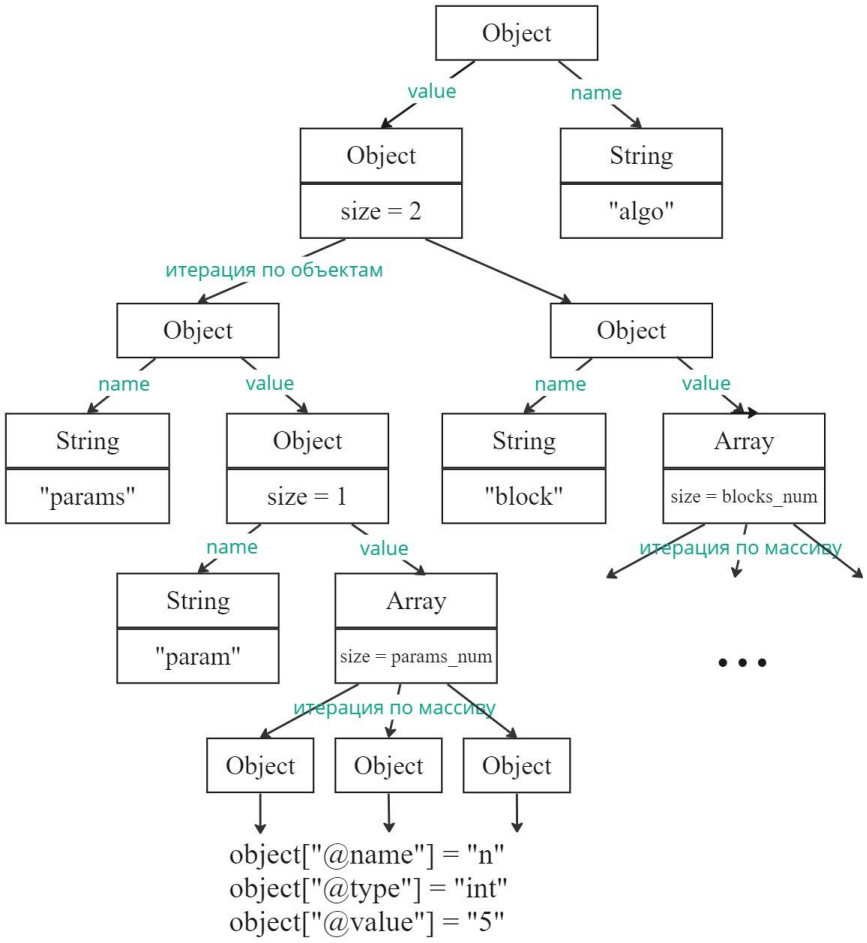

Рис. 16. Фрагмент DOM структуры, построенной библиотекой RapidJSON на основе формата входных данных - описания информационной структуры алгоритма на языке Algolang

Также библиотека показывает хорошие результаты скорости работы и имеет встроенные средства для обработки ошибок, с помощью которых была реализована некоторая валидация входных данных (например, повторное использование имени аргумента внутри одного блока, заданы некорректные границы аргументов и т.д.).

Таким образом, в процессе работы данного этапа осуществляется сбор всей информации, содержащейся во входном файле на языке Algolang, и формирования на основе неё программных классов. Также в процессе обхода для удобства в отдельной функции производятся промежуточные вычисления значений математических выражений, зависящих от значений параметров, а именно диапазоны значений пространства итераций, которые также сохраняются в программе в качестве элементов классов. Остальные математические выражения сохраняются в заданном виде - в виде строк и вычисляются при помощи той же функции во время выполнения другой части программы. На этом этапе завершается работа с входными данными, дальнейшая работа происходит только на основе обработанных данных.

6.2.3. Вычисление математических выражений

В рамках данной работы вычисление математических выражений необходимо для обработки выражений, описанных во входных данных. Инструментом для этих целей была выбрана библиотека ExprTk, которая легко интегрируется в проекты, поскольку представляет собой один заголовочный файл без зависимостей, предлагает большую базу обрабатываемых функций и операций, а также быстро выполняет вычисления. Все вычисления в программе осуществляются одной отдельной функцией, принимающей на вход выражение в виде строки и карту имён и значений параметров, участвующих в вычислении. Таким образом, в качестве выражений в процессе работы программы данная функция обрабатывает:

1. Диапазон значений пространства итераций (на этапе обхода DOM дерева)
2. Условия существования вершины при текущих значениях пространства итераций (на этапе работы "основного цикла")
3. Координаты вершины, из которой дуга ведёт в вершину с текущими значениями пространства итераций (на этапе работы "основного цикла")

6.2.4. Основной цикл

Так называемый “основной цикл” программы осуществляет содержательную её часть и формирует программные классы с итоговой информацией о модели графа алгоритма, которые достаточно переписать в неком стандартном виде для возможности дальнейшего использования.

Работа "основного цикла" реализует обработку одного блока алгоритма, соответственно данный фрагмент программы также работает в цикле для всех блоков. Когда на обработку подаётся блок, как программный класс со всей информацией о нём, для создаётся поле id-номеров вершин будущего информационного графа на основе диапазонов пространства итераций и определяется сдвиг всех координат по одной оси в зависимости от размерности предыдущего блока, чтобы блоки не накладывались друг на друга. На каждом шаге итерации выбирается текущее значение координат. Далее в цикле проверяются условия существования вершин с помощью функции вычисления математических выражений для этих координат. Если условие выполняется, то для данных значений создаётся вершина, которой присваивается очередной порядковый id и, на основе информации, полученной из описания на языке Algolang, её тип, или, если тип не был указан, то ему присваивается значение по умолчанию. После этого осуществляется обработка всех вершин-источников, из которых дуга ведёт в текущую. Проверяется, существует ли эта вершина, путём поиска её в поле id-номеров вершин блока, к которому она принадлежит, по заданным координатам. Если такой вершины не существует, значит она не является операцией алгоритма, а обозначает зависимость от входных данных. Такая вершина создаётся со специально определённым типом входных данных (тип "0") и значению уровня ярусно-параллельной формы данной вершины присваивается нуль. Если такая вершина существовала, то по её координатам программа получается id, также обращаясь к полю вершин. После этого для каждой вершины-источника, по её id и id текущей вершины, создаётся дуга. После обработки всех вершин-источников, для текущей вершины возможно определение уровня её ярусно параллельной формы. Оно вычисляется по формуле

$$\max_{(\text{вершины-источники})} \text{уровень ЯПФ} + 1$$

Далее происходит следующая итерация цикла(см. Рис. 17).

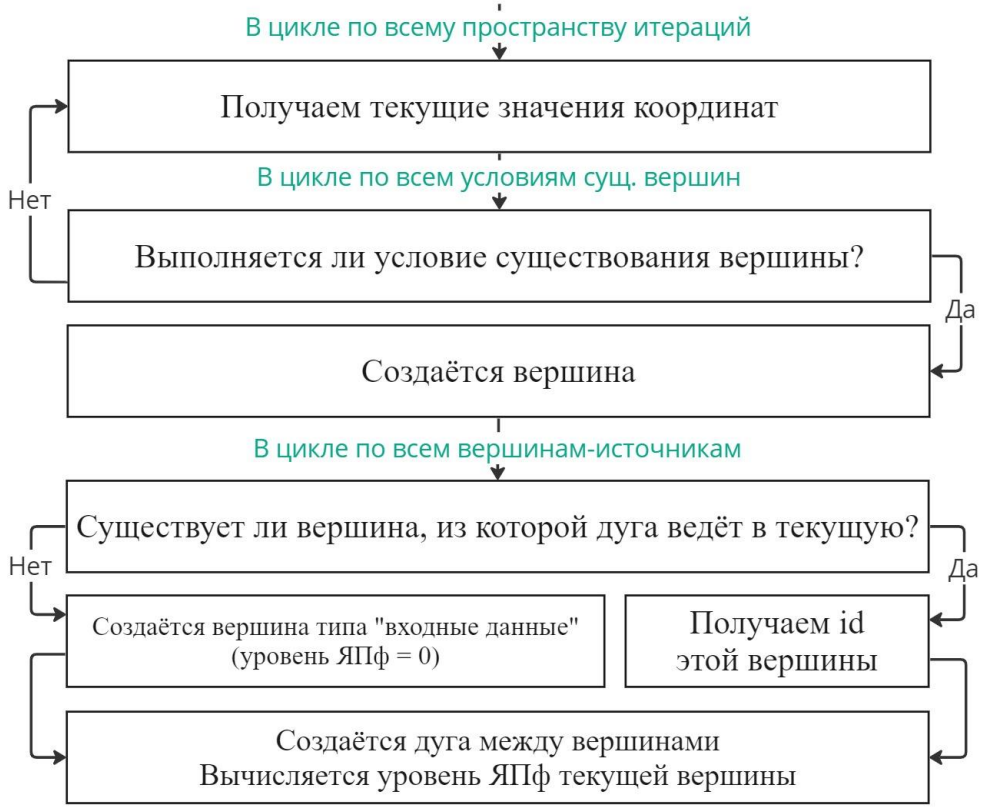

Рис. 17. Схема работы "основного цикла" программы

6.2.5. Формирование выходных данных

В результате работы "основного цикла" программа формирует список объектов программных классов вершин и дуг с описанием их свойств. На основе этих списков формируются выходные данные в формате JSON, поскольку он является стандартным форматом для передачи текстовой информации. Таким образом, выходные данные реализованной программы представляют собой список вершин и дуг с перечислением их свойств. Для Вершин — это id, координаты, тип и уровень ярусно-параллельной формы, для дуг — id, id вершин, которые она соединяет, с направлением от первой названой вершины ко второй, и тип (который задан по умолчанию для данной реализации) (см. Рис. 17 и Рис. 18).

```
{
    "vertices": [
        { "id": 0, "coordinates": [0, 0, 0], "type": "0", "level" : 0 },
        { "id": 1, "coordinates": [1, 0, 0], "type": "1", "level" : 1 },
        { "id": 2, "coordinates": [2, 0, 0], "type": "1", "level" : 2 }
    ],
    "edges": [
        { "id": 0, "sourceVertexId": 0, "targetVertexId": 1, "type": "0" },
        { "id": 1, "sourceVertexId": 1, "targetVertexId": 2, "type": "0" }
    ]
}
```

Рис. 18. Пример выходных данных

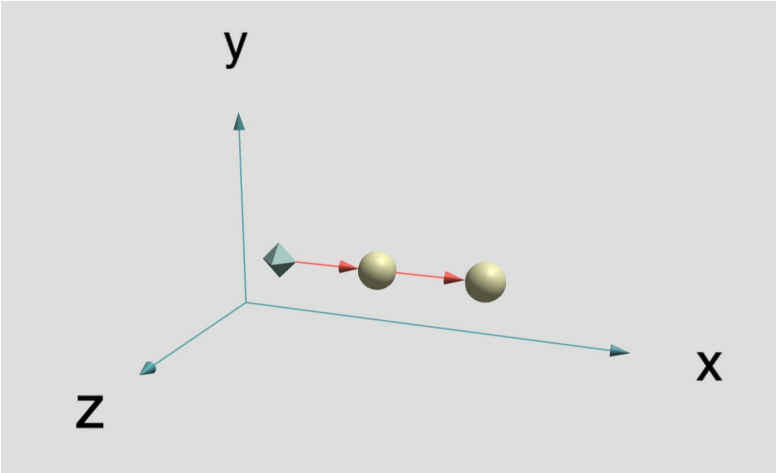

Рис. 19. Результат работы системы на примере выходных данных

6.3. Расположение вершин информационного графа

Для расположения вершин и дуг информационного графа алгоритма в трёхмерном пространстве был выбран следующий принцип. Вводится трёхмерная декартова система координат и сетка фиксированного размера на ней, зависящая от диапазона значений пространства итераций (см. Рис. 18 и Рис. 19).

1. Любая вершина графа алгоритма располагается в уникальном узле сетки и задаётся координатами этого узла.
2. Любая дуга задаётся парой узлов сетки - её началом и концом.

```
<arg name="i" val="1..3"></arg>
<arg name="j" val="1..4"></arg>
<arg name="k" val="1..5"></arg>
```

Рис. 18. Пример диапазона пространства итераций

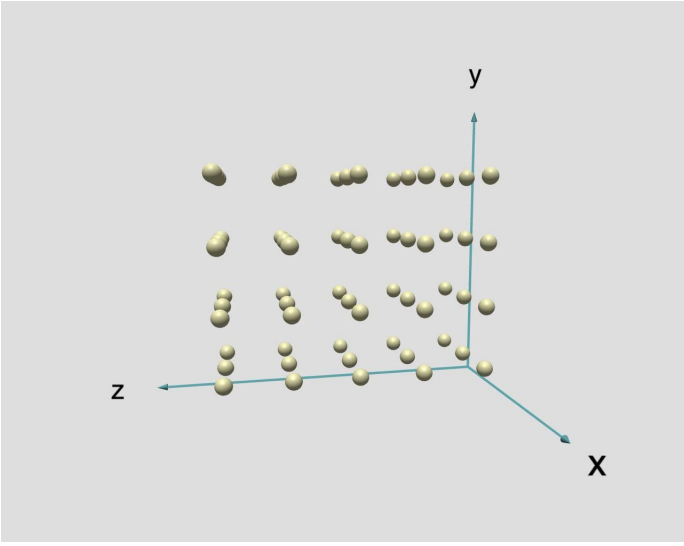

Рис. 19. Координатная сетка, построенная на основе примера диапазона пространства итераций

Таким способом можно отобразить в трёхмерной пространстве любой информационный граф алгоритма без наложений вершин. Данное решение не является наиболее подходящим для осуществления анализа информационного графа, поскольку не отражает распределение операций по уровням ярусно-параллельной формы, но оно существенно проще реализуемо и в достаточной степени визуально понятно. Для определения же принадлежности вершины к уровню ярусно-параллельной форме будут использованы средства интерактивной визуализации итоговой модели.

6.4. Примеры результатов выполнения программы

Система визуализации информационных графов алгоритмов в рамках которой была выполнена данная работа производит следующую цепочку действий с заданным алгоритмом(см. Рис. 20):

1. Существует описание некоторого алгоритма, подходящего под определение расширенного линейного класса, на выбранном языке программирования. Пользователь переводит его в описание на языке Algolang, следуя синтаксису языка и правилам написания на нём.
2. По результатам реализованной в рамках данной работы программы описание информационной структуры алгоритма на языке Algolang преобразуется в JSON-файл с описанием модели многомерного графа алгоритма.
3. По этим выходным данным вторая часть системы визуализации создает интерактивное изображение.

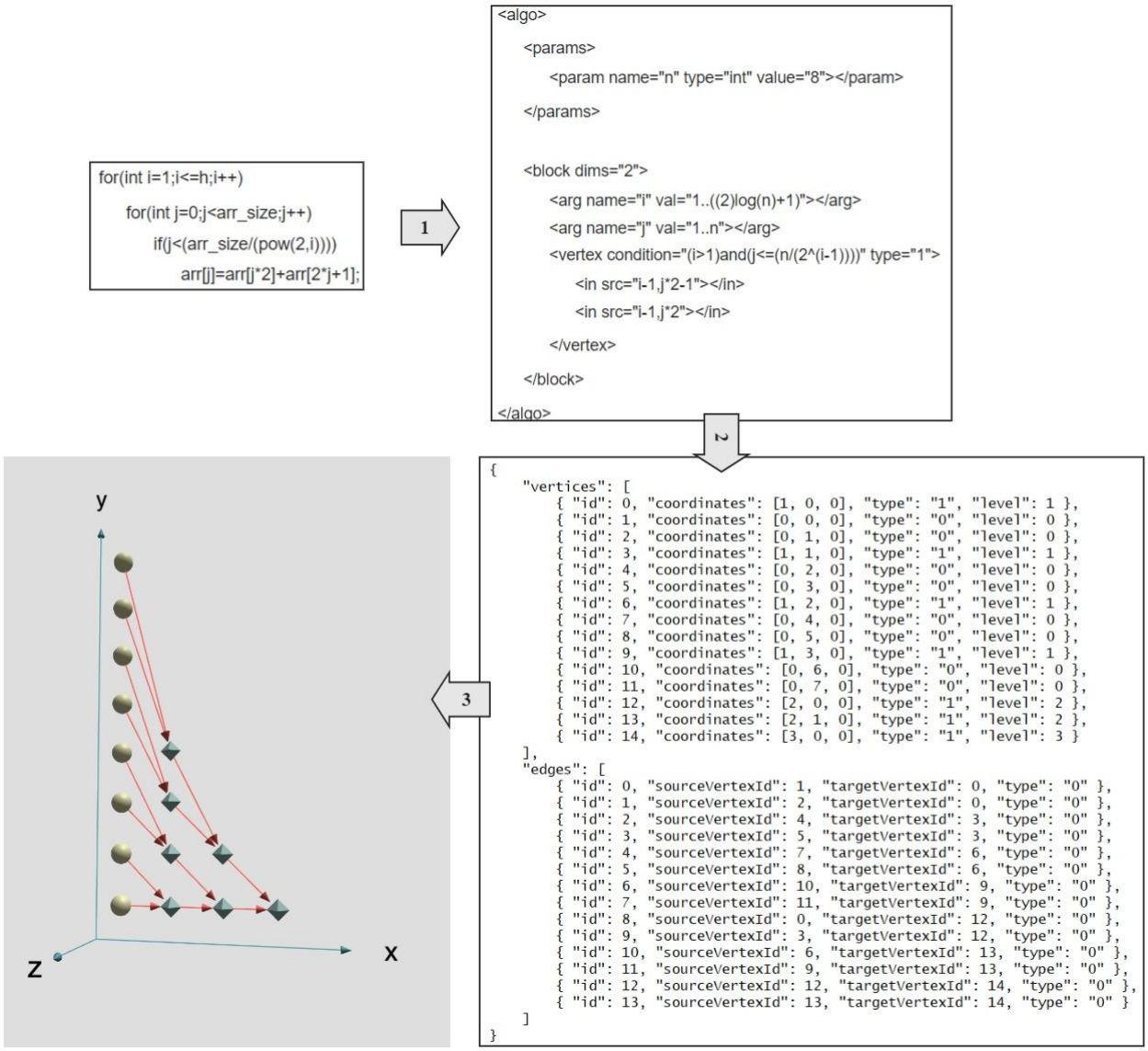

Рис. 20. Цепочка преобразований алгоритма нахождения суммы элементов массива сдваиванием

(алгоритм на языке C -> визуализация модели графа алгоритма)

Далее приведены результаты работы общей системы визуализации для некоторых алгоритмов с разным количеством блоков и их размерностью (см Рис. 21, Рис. 22, Рис. 23, Рис. 24, Рис. 25 и Рис. 26).

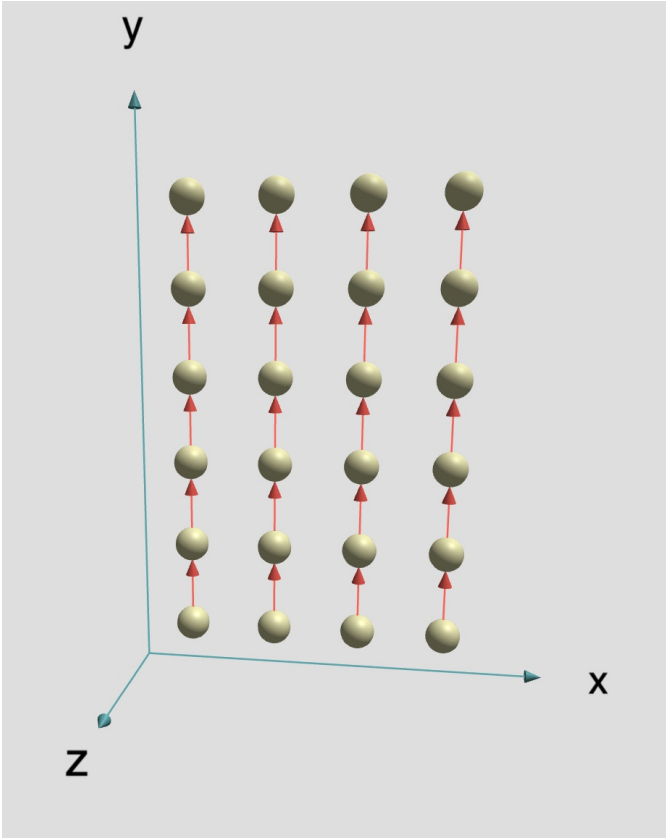

Рис. 21. Умножение матрицы на вектора

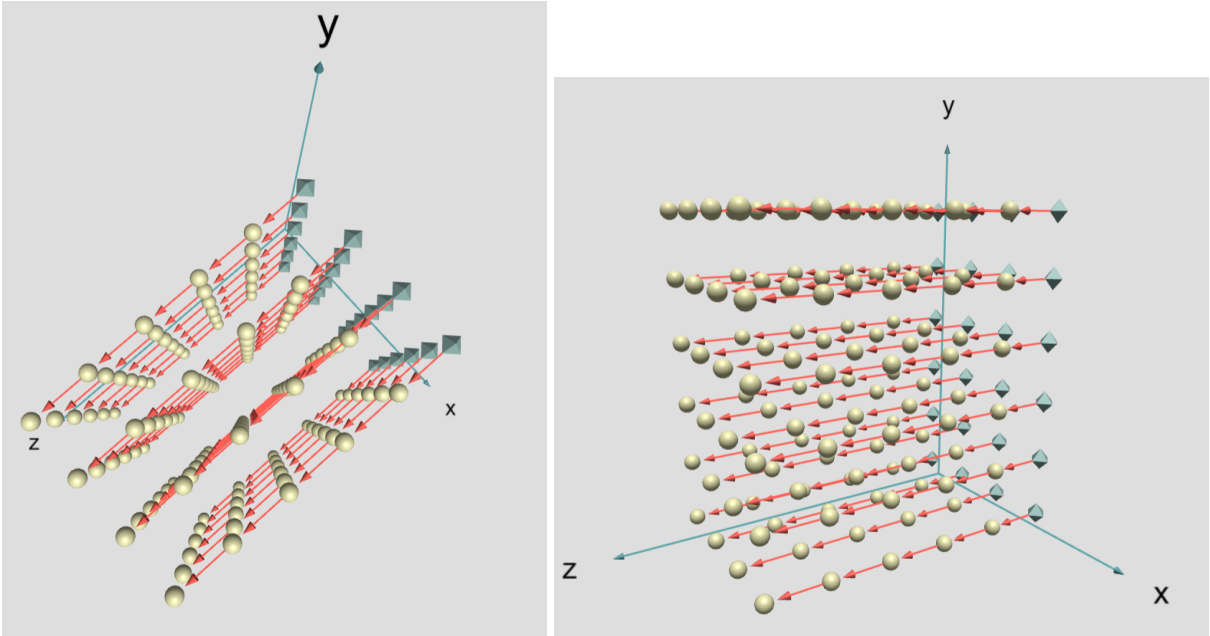

Рис. 22. Перемножение матриц

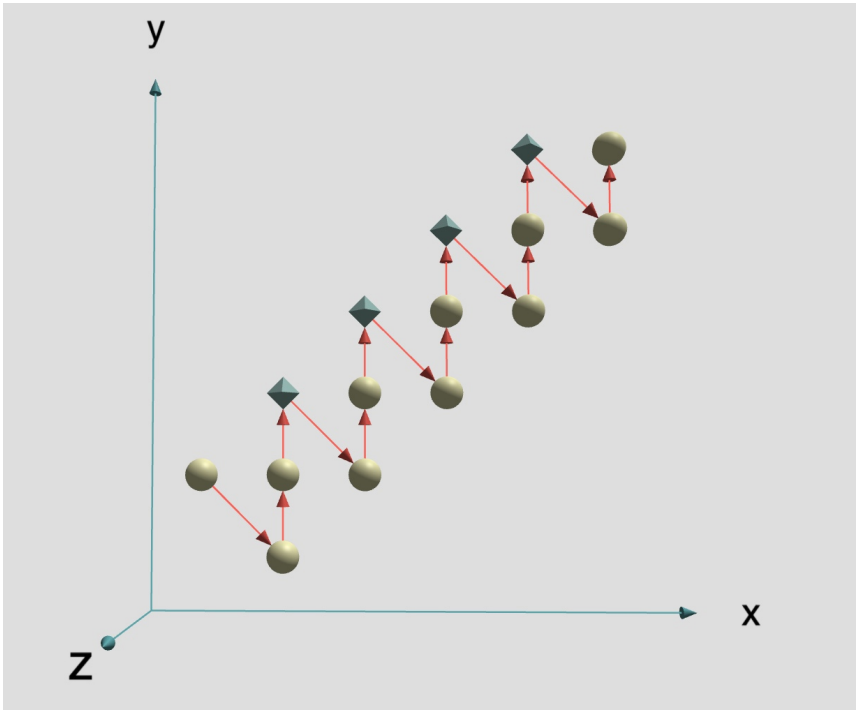

Рис. 23. Компактная схема метода Гаусса для трёхдиагональной матрицы

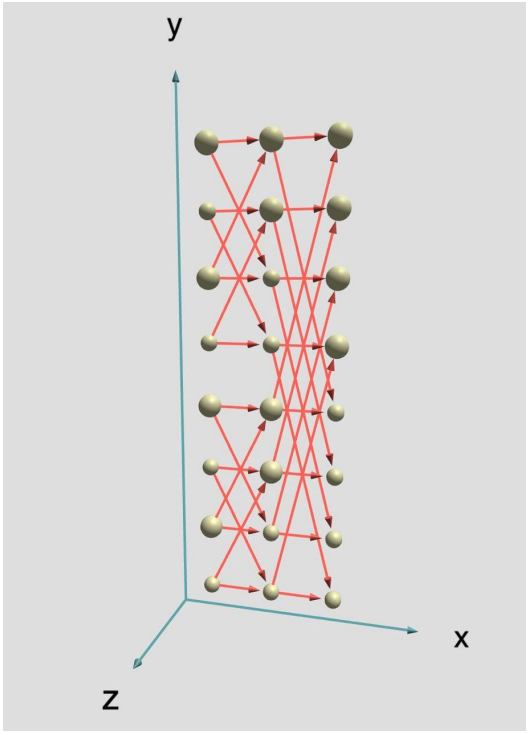

Рис. 24. Простой алгоритм Кули-Тьюки быстрого преобразования Фурье для степеней двойки

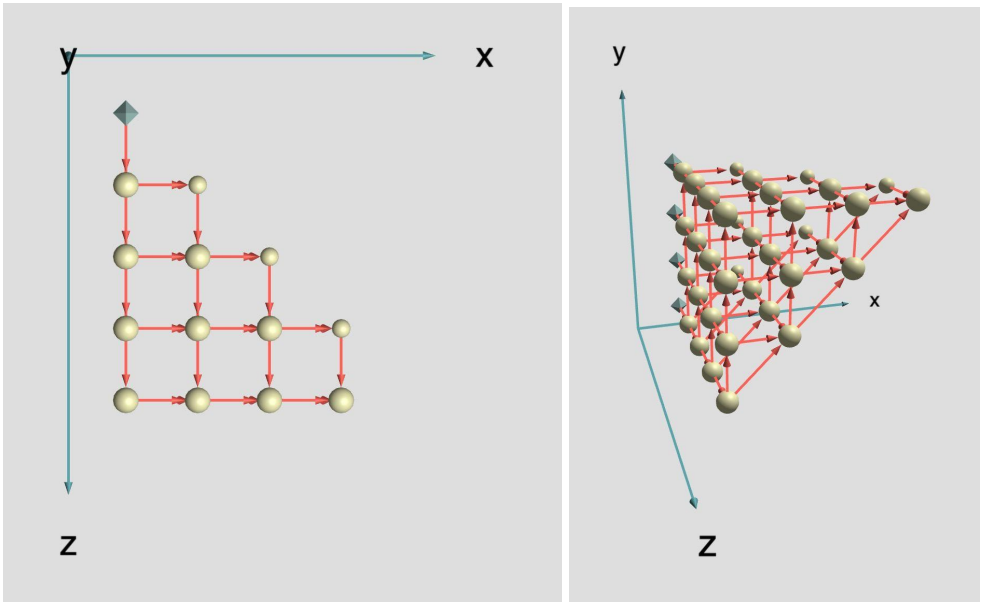

Рис. 25. Метод Гивенса (вращений) QR-разложения квадратной матрицы

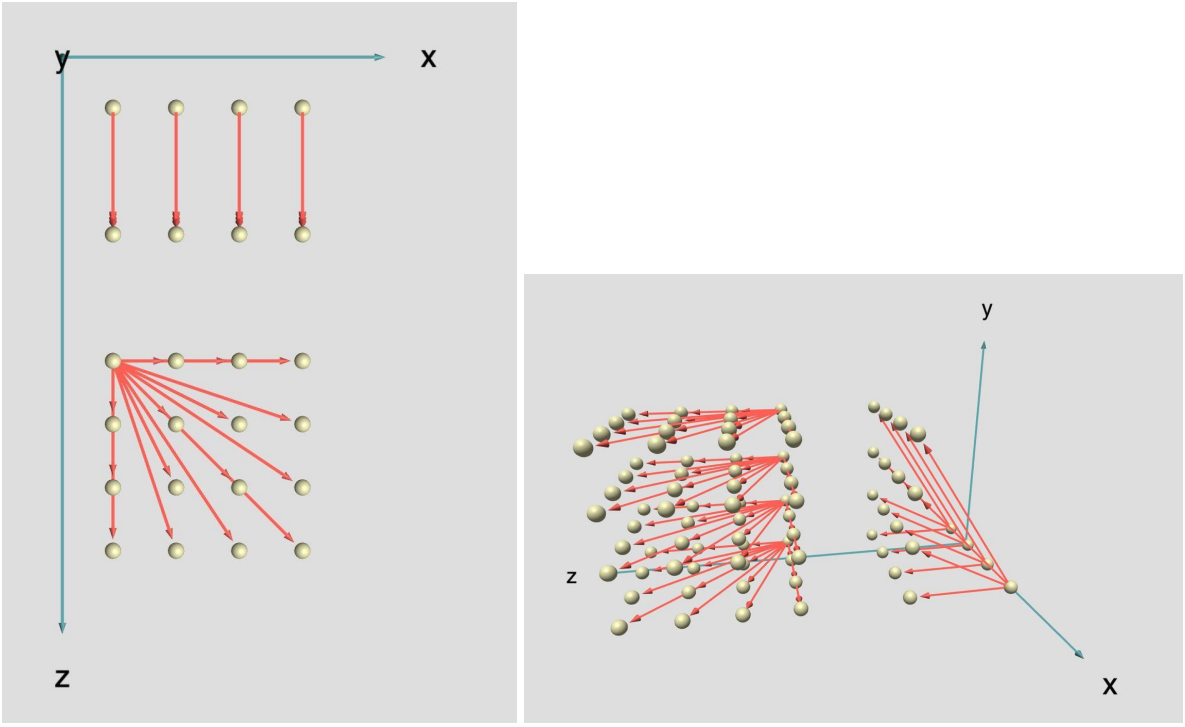

Рис. 26.Пример визуализации нескольких блоков разной размерности
(в программе на языке С нескольких циклов разной вложенности)

7. Результаты работы

В рамках данной работы была реализована программа, создающая описание многомерной модели информационного графа алгоритма. Для создания данной программы были решены следующие задачи:

1. Реализован сбор информации о графе алгоритма из файлов, написанных на языке Algolang, с использованием изученных библиотек для работы с форматом XML
2. Реализована обработка информации о графе алгоритма с использованием изученных библиотек для работы с математическими выражениями
3. Реализован алгоритм, который на основе собранных данных из описания графа на языке Algolang, строит описание многомерной модели графа, включающее в себя расположение вершин и дуг определённым образом и описание их свойств. Данное описание модели может быть использовано второй частью общей системы визуализации информационных графов алгоритмов для создания итогового интерактивного изображения

8. Список литературы

[1]. В.В. Воеводин, Вл.В. Воеводин. Параллельные вычисления. — СПб.: БХВ-Петербург, 2002. — 608 с.

[2]. James F.Bowring. Modeling and predicting software behaviors: Doctor of Philosophy in the College of Computing. — Georgia, 2006. — 165 с.

[3]. Mark Higgins. What Are Dependency Graphs and Why Are They Important? [Электронный ресурс]. — Электрон. дан. — URL: https://www.beacon.io/what-are-dependency-graphs-and-why-are-they-important/. (дата обращения: 21.03.2023).

[4]. А.С. Новиков. Построение графа информационных зависимостей по машинному коду // Известия Тульского государственного университета.Технические науки. — 2016. № 2. — с. 180-187

[5]. N.A. Likhoded, A. A. Tiunchik. Method of constructing parallel forms of algorithms based on locally parallel, globally sequential partition. // Cybernetics and Systems Analysis. — Том 32. №2. — с. 173-184

[6]. Dependence Graphs in LLVM. [Электронный ресурс]. — Электрон. дан. — URL: https://llvm.org/docs/DependenceGraphs/index.html (дата обращения: 25.03.2023).

[7]. Э.И. Ватутин, В.С. Титов. Анализ областей качественного превосходства последовательных эвристических методов синтеза разбиений при проектировании логических мультиконтроллеров. // Известия высших учебных заведений. Приборостроение. — 2015. Том 58. № 2 — с. 115-122

[8]. AlgoWiki. Глоссарий. [Электронный ресурс]. — Электрон. дан. — URL: https://algowiki-project.org/ru/Глоссарий (дата обращения: 10.12.2022).

[9]. AlgoWiki. Открытая энциклопедия свойств алгоритмов. [Электронный ресурс]. — Электрон. дан. — URL: https://algowiki-project.org/ru/Открытая_энциклопедия_свойств_алгоритмов (дата обращения: 03.04.2023)

[10]. Alexander S. Antonov, Alexey V. Frolov, Hiroaki Kobayashi, Igor N. Konshin, Alexey M. Teplov, Vadim V. Voevodin, Vladimir V. Voevodin. Parallel Processing Model for Cholesky Decomposition Algorithm in AlgoWiki Project // Supercomputing frontiers and innovations. — 2016. Том 3. №3. — с. 61-70

[11]. Волков Н.И. Документация языка Algolang. [Электронный ресурс]. — Электрон. дан. — URL: https://parallel.ru/sites/default/files/info/education/opisanie_yazyka_algolang.pdf (дата обращения: 11.12.2022)

[12]. Peter-Paul Koch. The Document Object Model: an Introduction [Электронный ресурс]. — Электрон. дан. — URL: https://www.digital-web.com/articles/the_document_object_model/ (дата обращения: 20.04.2023)

[13]. David Brownell. SAX2. — Gravenstein Highway North, Sebastopol: O'Reilly Media, 2002. — 240 с.

[14]. Difference between DOM vs SAX Parser in Java — XML Parsing in Java. [Электронный ресурс]. — Электрон. дан. — URL: https://www.java67.com/2012/09/dom-vs-sax-parser-in-java-xml-parsing.html#ixzz7xtkySKLe. (дата обращения: 20.04.2023)

[15]. Sukumar Paul. How to Parse XML in C++. [Электронный ресурс]. — Электрон. дан. — URL: https://linuxhint.com/parse_xml_in_cpp/ (дата обращения: 15.02.2023)

[16]. Документация библиотеки RapidXML. [Электронный ресурс]. — Электрон. дан. — URL: https://rapidxml.sourceforge.net/ (дата обращения: 23.04.2023)

[17]. Документация библиотеки PugiXML. [Электронный ресурс]. — Электрон. дан. — URL: https://pugixml.org/ (дата обращения: 23.04.2023)

[18]. Документация библиотеки TinyXML. [Электронный ресурс]. — Электрон. дан. — URL: https://github.com/leethomason/tinyxml2 (дата обращения: 23.04.2023)

[19]. PugiXML. Benchmarks. [Электронный ресурс]. — Электрон. дан. — URL: https://pugixml.org/benchmark.html (дата обращения: 25.04.2023)

[20]. XSD: XML Data Binding for C++. [Электронный ресурс]. — Электрон. дан. — URL: https://www.codesynthesis.com/products/xsd/ (дата обращения: 23.04.2023)

[21]. Документация библиотеки RapidJSON [Электронный ресурс]. — Электрон. дан. — URL: http://rapidjson.org/ (дата обращения: 10.04.2023)

[22]. Документация библиотеки ExprTk. [Электронный ресурс]. — Электрон. дан. — URL: https://github.com/ArashPartow/exprtk (дата обращения: 18.02.2023)

[23]. C++ Mathematical Expression Parsing And Evaluation Library. [Электронный ресурс]. — Электрон. дан. — URL: http://www.partow.net/programming/exprtk/ (дата обращения: 18.02.2023)

[24]. Joel Jones. Abstract Syntax Tree Implementation Idioms. [Электронный ресурс]. — Электрон. дан. — URL: https://hillside.net/plop/plop2003/Papers/Jones-ImplementingASTs.pdf (дата обращения: 02.05.2023)

[25]. Документация серии библиотек muParser. [Электронный ресурс]. — Электрон. дан. — URL: https://beltoforion.de/en/muparserx/ (дата обращения: 17.04.2023)

[26]. Документация библиотеки muparser. [Электронный ресурс]. — Электрон. дан. — URL: https://beltoforion.de/en/muparser/ (дата обращения: 17.04.2023)

[27]. Документация библиотеки METL. [Электронный ресурс]. — Электрон. дан. — URL: https://github.com/TillHeinzel/METL (дата обращения: 28.04.2023)

[28]. C++ Mathematical Expression Parser Benchmark. [Электронный ресурс]. — Электрон. дан. — URL: https://github.com/ArashPartow/math-parser-benchmark-project (дата обращения: 07.05.2023)