Бланк титульного листа выпускной квалификационной работы бакалавра

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НОВОСИБИРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
(НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ, НГУ)

Факультет информационных технологий

Кафедра Параллельных вычислений

Направление подготовки: 09.03.01 ИНФОРМАТИКА И ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА

Образовательная программа: 09.03.01 ИНФОРМАТИКА И ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА**

# РАЗРАБОТКА И РЕАЛИЗАЦИЯ ВЕБ-СРЕДЫ ВИЗУАЛЬНОЙ РАЗРАБОТКИ ФРАГМЕНТИРОВАННЫХ ПРОГРАММ

утверждена распоряжением проректора по учебной работе № 179 от « 16 » мая 2017 г.

скорректирована распоряжением проректора по учебной работе № 19 от « 22 » января 2019 г.

Ижицкий Руслан Леонидович, группа 15201 _____

(подпись студента)

| **«К защите допущена»** | **Руководитель ВКР** |
|---|---|
| Заведующий кафедрой Параллельных вычислений ФИТ НГУ, | к.т.н., |
| д.т.н., профессор | доцент кафедры Параллельных вычислений ФИТ НГУ |
| Малышкин В.Э. /…….. | Маркова В.П. /……... |
| (подпись) | (подпись) |
| «……»……2019 г. | «……»……2019 г. |

Дата защиты: «……» июня 2019 г.

Новосибирск, 2019 г.

Бланк задания на выполнение выпускной квалификационной работы бакалавра

МИНОБРНАУКИ РОССИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НОВОСИБИРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
(НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ, НГУ)

Факультет информационных технологий

Кафедра Параллельных вычислений
Направление подготовки: 09.03.01 ИНФОРМАТИКА И ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА

УТВЕРЖДАЮ

Зав. кафедрой Малышкин В.Э.

…….
(подпись)
«……»……20…г.

**ЗАДАНИЕ**

**НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ БАКАЛАВРА**

Студенту Ижицкому Руслану Леонидовичу, группы 15201

Тема «Разработка и реализация веб-среды визуальной разработки фрагментированных программ»

утверждена распоряжением проректора по учебной работе от «16» мая 2017 г. № 179
скорректирована распоряжением проректора по учебной работе от «22» января 2019 г. №19

Срок сдачи студентом готовой работы……2019 г.

Исходные данные (или цель работы): разработка визуального языка и визуальной веб-среды разработки, позволяющие разрабатывать фрагментированные программы для исполнения в системе LuNA.

Структурные части работы: изучение технологии фрагментированного программирования и системы фрагментированного программирования LuNA, обзор существующих визуальных языков и проблем их разработки, разработка визуального языка описания фрагментированных алгоритмов, разработка и реализация веб-среды для визуальной разработки фрагментированных алгоритмов.

Руководитель ВКР
доц. каф. ПВ ФИТ,
к.т.н
Маркова В.П. /……..
(подпись)
«…»……20…г.

Задание принял к исполнению
Ижицкий Р.Л. /……..
(подпись)
«…»……20…г.

СОДЕРЖАНИЕ

# ВВЕДЕНИЕ

Существуют два вида программирования – системное и прикладное. Прикладное является основной целью программиста (расчет какой-либо прикладной задачи), когда системное требуется для работы программы: например, нужно позаботиться о выделении и освобождении памяти (в некоторых высокоуровневых языках освобождением памяти программист не занимается). В параллельном программировании системной частью может является управление памятью и балансировка данных между узлами суперкомпьютера.

Существует общемировая тенденция к повышению уровня программирования путем автоматизации системного программирования. При высокоуровневом программировании задача системного программирования отдается компилятору или исполнителю системы (интерпретатору), из-за чего уменьшается число ошибок программиста, не связанных напрямую с прикладной задачей, и повышается эффективность программирования. Эта задача довольно сложна в случае параллельного программирования.

В ИВМиМГ разрабатывается система и язык LuNA, преследующие цель повышения уровня параллельного программирования путем автоматизации создания реализаций больших численных моделей для суперкомпьютеров [1]. В языке исполнительной семантикой является data-flow (по готовности данных), а алгоритм можно представить графом информационных зависимостей, состоящих из переменных единственного присваивания и операций единственного срабатывания. Как у большинства систем программирования, у данной системы есть проблема низкой доступности внешним пользователям: требуется установка системы, а также существует определенный порог вхождения в язык. Эта работа посвящена решению этой проблемы.

Для data-flow семантики хорошо подходит визуальное представление алгоритма. Исходя из этого поставлена следующая цель работы: создание визуальной веб-среды разработки LuNA-программ. Визуальная разработка программ традиционно является средством повышения уровня программирования, а наличие веб-среды повышает доступность системы.

Задачами работы являются анализ требований к визуальному языку, его разработка на базе языка LuNA, реализация визуальной веб-среды разработки и реализация транслятора из визуального представления в текстовое.

Научная новизна заключается в применении подхода визуального программирования к разработке LuNA-программ. Наличие визуальной среды разработки позволит повысить уровень программирования в системе LuNA и упростить ее изучение для незнакомых с системой программистов, что является практической ценностью данной работы.

Работа состоит из введения, четырех глав и заключения. Введение раскрывает актуальность, цель работы, научную новизну, практическую ценность и дает обзор родственных работ. Первая глава содержит анализ других решений, вводит основные компоненты языка LuNA и приводит постановку задачи. Вторая глава описывает разработку визуального языка и компонентов визуальной веб-среды. Третья описывает разработку визуальной веб-среды. В четвертой приводятся результаты разработки визуальной веб-среды. В заключении приводятся итоги и выводы по данной работе.

# 1 Анализ проблемной области и постановка задачи

## 1.1 Обзор визуальных языков

Визуальных языков существует большое количество, типичными представителями являются, например, ДРАКОН [2], LabVIEW [3], Simulink [4], Pifagor [5]. Некоторые из них активно используются в своих предметных областях. Например, LabVIEW используется для системного проектирования в отраслях, где требуется проведение испытаний, измерений и осуществление управления, а также быстрый доступ к оборудованию и результатам анализа данных (в физике, электромашиностроении, обработке сигналов, медицине и других). Simulink используется для моделирования динамических систем (таких как электрических схем, химических индустриальных схем и механических контролируемых систем - машин, самолетов).

ДРАКОН является control-flow языком, а не data-flow. В нем нет элемента схемы, соответствующего данным, что является необходимым для реализации алгоритма на распределенной памяти. Семантика существующих визуальных data flow языков далека от семантики языка LuNA, следовательно, ни один из вышеперечисленных языков не подходит для визуальной разработки LuNA-программ.

## 1.2 Проблемы разработки визуальных data flow языков

В разработке визуальных data flow языков существует множество открытых проблем: например, представление циклов, структур данных и больших участков программного кода.

### 1.2.1 Представление циклов

Проблема представления циклов заключается в том, что компактное представление циклических конструкций, не нарушая принципа data flow (единственное присваивание переменных), не является тривиальной задачей. В статье [6] выполнено сравнение способов представления итерационных конструкций.

Например, некоторые визуальные языки (Show-And-Tell [7], LabVIEW [3], Prograph [8]) используют специальный блок, выделяющий часть кода для итеративного исполнения. В блоке число входов равно числу выходов, а также есть блок условия цикла. Пока это условие выполняется, выходы блока идут на его входы.

Другой подход реализован в Cantata [9]: цикл скрывается в одной вершине, где указывается последовательность присваиваний (причем присваивания могут не соответствовать data flow семантике: одной переменной может быть присвоено несколько значений).

Еще одно решение проблемы было представлено в VEE [10]: граф представляется через явные циклы в графе зависимостей с использованием вспомогательных блоков: `for` генерирует последовательность индексов, `next` активирует следующую итерацию цикла, `break` прекращает выполнение цикла.

### 1.2.2 Представление структур данных

Другой проблемой визуальных языков является представление структур данных. Проблема представления заключается в том, что при изменении одного элемента структуры нельзя менять другие элементы из-за правила единственного присваивания. Обзор решений этой проблемы представлен в

статье [11]. Одним из решений является Dennis’s Method [12], где используется представление динамической памяти в качестве направленного графа с дугами от массивов к их данным. При смене одного значения в массиве создается новая вершина для нового элемента в массиве и еще одна для массива, указывающая на новый элемент и старые элементы с исключением измененного элемента. Однако в этом подходе может происходить создание большого числа ненужных конструкций, например, при последовательном изменении структуры данных в цикле.

Другим решением являются I-Structures [13], в котором все поля структуры по умолчанию не инициализированы, а запись каждого допускается один раз. У этого решения есть та же проблема, что и у предыдущего подхода.

Существует также гибридный подход [14]. Он использует шаблон массива, ссылающийся либо на оригинальный массив, либо на созданный блок элементов, в котором хотя бы один элемент был модифицирован. При обращении в шаблон массива происходит перенаправление либо в элемент оригинального массива, либо в соответствующий блок. При этом подходе также происходят лишние операции копирования данных, но их меньше, чем при других подходах.

### 1.2.3 Представление большого программного кода

Представление большого кода является еще одной проблемой в визуальном программировании. Чем больше одновременно показывается графических элементов, тем сложнее в них ориентироваться. Самым простым решением проблемы является использование подпрограмм. Другим решением является возможность сворачивания блока кода в одну операцию. Наконец, можно взять опыт картографических систем: при достаточном удалении от

полотна можно показывать меньше деталей (скрыть дуги, оставить только модули или кластеры кода).

## 1.3 Описание языка LuNA

Язык LuNA создавался с целью записывать численные алгоритмы на высоком уровне абстракции и сократить время программиста на запись алгоритмов, уменьшив время на программирование прикладной части программы.

Алгоритм, записанный на языке LuNA, называется фрагментированным алгоритмом. Он является фрагментированным в том смысле, что элементами алгоритма преимущественно являются не отдельные числа и операции над ними, а целые блоки данных (фрагменты) и операции над фрагментами. Фрагментированный алгоритм состоит из потенциально бесконечного множества переменных единственного присваивания и операций единственного срабатывания, связанных отношениями in и out. Исполнительной семантикой языка LuNA является data flow (по готовности данных). Порядок исполнения фрагментированного алгоритма влияет на расход памяти и доступный параллелизм в разные моменты времени. Data flow семантика хорошо представляется через граф зависимостей с переменными и операциями.

Далее перечислены элементы языка LuNA.

### 1.3.1 Переменные

В языке LuNA переменные единственного присваивания. В языке нет массивов, но есть индексированные переменные (имена, содержащие произвольное число индексов), которые вычисляются к моменту исполнения. В частности, эти переменные можно интерпретировать как массивы.

Пример объявления переменных `x` и `y` в языке LuNA:

```
df x, y;
```

### 1.3.2 Операции

Операцией является применение внешней подпрограммы к заданным переменным. При реализации операции указывается реализующая ее подпрограмма и множество переменных, к которым применяется операция. Также может быть указано имя операции.

Пример объявления операции и реализующей подпрограммы:

```
import c_add(int, int, name) as add;
cf a: add(#in: x, y, #out: y);
```

### 1.3.3 Условные операторы

Условные операторы в языке позволяют описывать алгоритмы с ветвлениями. В операторе указываются условие (логическое выражение, содержащее переменные языка LuNA) и операции, которые могут сработать при выполнении этого условия. После выполнении условия операции внутри блока помечаются готовыми к исполнению. Условные операторы влияют явно только на заданное множество операций в алгоритме и косвенно на все элементы, зависимые от ближайшего элемента, из-за data flow семантики.

Пример условного оператора:

```
if (x>0) {sqrt(#in x, #out y);}
```

### 1.3.4 Циклы for

В языке LuNA существуют циклы двух видов: for и while. Циклы for принимают на вход границы счетчика и множество операций, которые будут

выполнены при всех значениях счетчика. В циклах часто используются индексированные имена, описанные выше.

Пример оператора for:

```
for i=1..N { func(i, y[i]); }
```

### 1.3.5 Циклы while

Циклы while принимают на вход нижнюю границу счетчика и условие завершения. Результатом работы цикла является минимальное значение счетчика, на котором условие цикла не выполнилось, это значение помещается в заданную выходную переменную.

Пример оператора while:

```
while x[i-1]>0, i=1..out N { func(i, x[i]); }
```

## 1.4 Постановка задачи

Целью работы является разработка визуального языка и визуальной веб-среды разработки, позволяющие разрабатывать LuNA-программы.

К языку предъявляются следующие требования. Визуальный язык должен быть спроектирован таким образом, чтобы обеспечивать создание LuNA-программ.

Визуальная веб-среда разработки должна обеспечивать рисование и удаление графических примитивов и связей между ними, преобразовывать граф в текстовое представление на языке LuNA, позволять добавлять методы из подключаемого модуля на C++, после чего генерировать его шаблон, позволять экспортировать программу на визуальном языке в текстовое представление и загружать граф из текстового представления.

# 2 Разработка визуального языка

Так как язык LuNA является data flow языком, то представление визуального языка в виде графа зависимостей будет одним из естественных представлений. Переменные и операции логично отображать блоками, связи между элементами языка в виде дуг ориентированного графа.

Визуальный язык должен включать базовые примитивы, соответствующие компонентам языка LuNA: переменные, операции, структуры, условные операторы, циклы и связи между примитивами. Операции должны позволять именование (имя соответствующего модуля и опциональное имя данной операции) и упорядочение аргументов, условные операторы визуального языка - вложенные условия, а циклы - включать несколько визуальных компонент (связи между элементами в том числе) в одну циклическую конструкцию и настройку границ счетчика.

В качестве визуального языка для записи фрагментированных алгоритмов был разработан язык, основанный на языке LuNA и имеющий в основе граф информационных зависимостей.

Далее рассматриваются компоненты разработанного визуального языка.

## 2.1 Переменные и операции

На рисунке 1 показано визуальное представление алгоритма, состоящего из трех переменных и одной операции. В качестве визуального представления переменной был выбран круг с именем переменной. В качестве визуального представления операции был выбран прямоугольник с именем реализующей ее подпрограммы. Имя операции было решено не отображать в первом варианте визуального языка, чтобы не перегружать изображение несущественными деталями. Поддержка имен операций запланирована для реализации в веб-среде без визуального отображения.

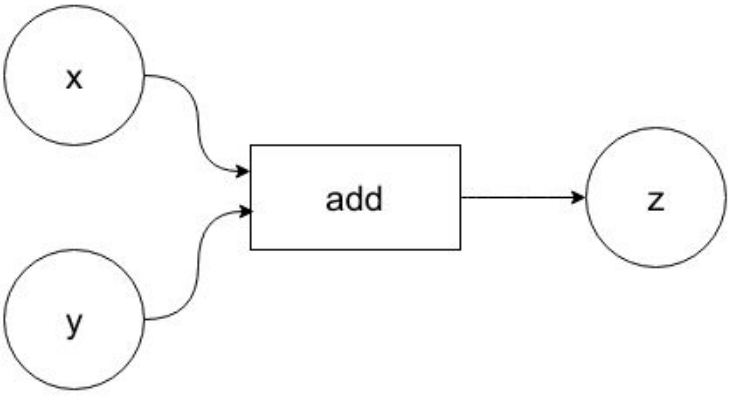


Рисунок 1 – переменные, операция и связи визуального языка

## 2.2 Связи между компонентами

В качестве обозначения информационной связи между компонентами (зависимостей по данным) (рисунок 1) была выбрана сплошная дуга ориентированного графа. Каждой дуге ставится в соответствие число, определяющее порядок соответствующей переменной в соответствующей операции. При отсутствии данного числа порядок аргументов в операции может оказаться произвольным. Связи вида зависимости по данным допускаются только между переменными и операциями. Также допускаются связи особого вида между началом цикла и операцией, обозначаемые дугой с пунктирной линией (рисунок 4). Эти связи должны соединять начало цикла с такими вершинами из подграфа цикла, что все вершины в подграфе станут достижимыми из начала цикла при обходе по направлению этих и информационных связей. Эти связи не являются типа data flow.

## 2.3 Условные операторы

В качестве условного оператора (рисунок 2) был выбран блок условия и два индикатора условий (для положительного - со знаком “+”, и отрицательного срабатывания условия - со знаком “-”). Условие является операцией особого

вида, а индикаторы – переменными особого вида. Переменные особого вида не входят в список аргументов операций при наличии связи.

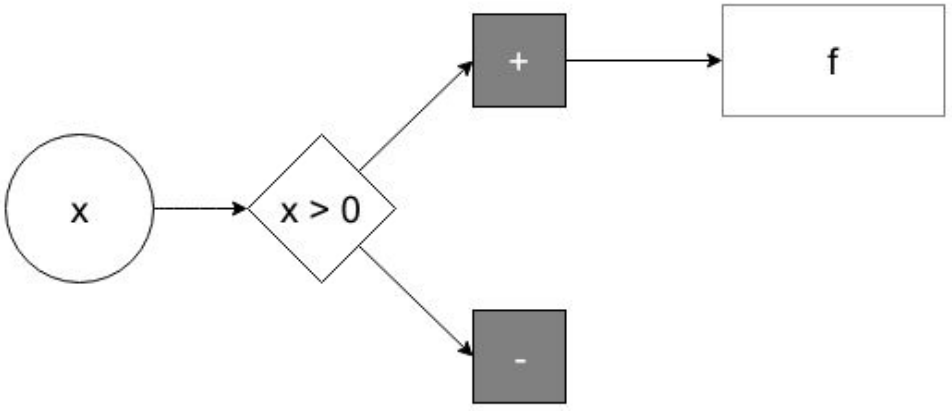


Рисунок 2 – условный оператор и индикаторы условий

## 2.4 Структуры данных

Структуры данных и их компоненты в языке обозначаются различными переменными. Переход от одних к другим обозначается специальными операциями (графически представляемым как серый квадрат). Например, переход от массива к элементу массива обозначается специальной операцией, показанной на рисунке 3. Переход от элемента в цикле к массиву обозначается специальной операцией, показанной на рисунке 4.

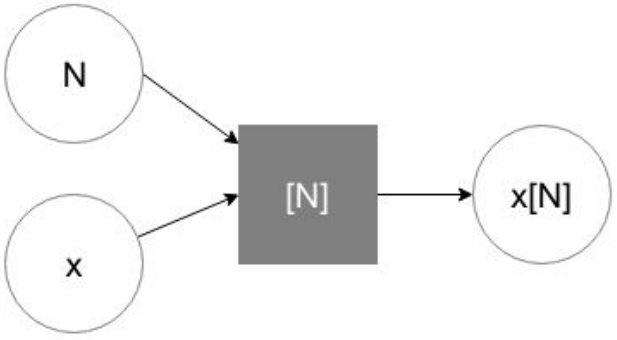


Рисунок 3 – переход от массива к элементу

## 2.5 Циклы for

В языке luna в качестве циклических конструкций используются циклы с множеством операций. В качестве первого прототипа для циклов for (рисунок 1г) было выбрано представление в виде одного блока начала цикла (шестиугольник) и нескольких блоков границ цикла (двойная окружность). Блок начала цикла интерпретируется как операция особого вида. Из вершины цикла позволяется проводить пунктирные линии, обозначающие связь в виде принадлежности циклу, а не связь по данным в смысле data flow. Блоками границ цикла неявно задаются границы подграфа, составляющего тело цикла: вершина находится в цикле тогда и только тогда, когда обход графа по направлению дуг (как по сплошным, так и по пунктирным) начиная с вершины цикла доходит до этой вершины, не посещая границы цикла. В будущих реализациях планируется расширенная семантика для границ цикла: взятие по индексу, редукция или взятие последнего элемента.

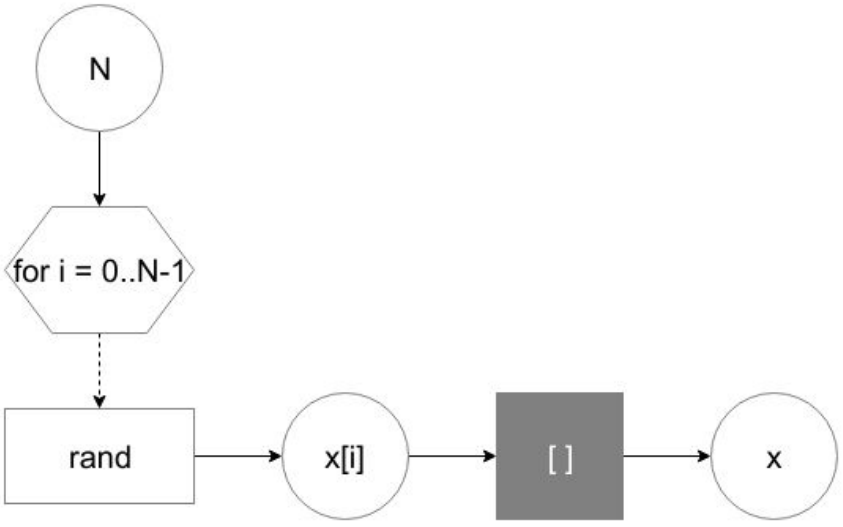


Рисунок 4 – цикл for

## 2.6 Циклы while

В качестве первого прототипа для циклов while (рисунок 5) было выбрано представление, аналогичное представлению для цикла for. Отличие состоит только в блоке начала цикла: у него указывается условие, а вместо конечного значения указывается переменная, куда сохранится последняя итерация цикла.

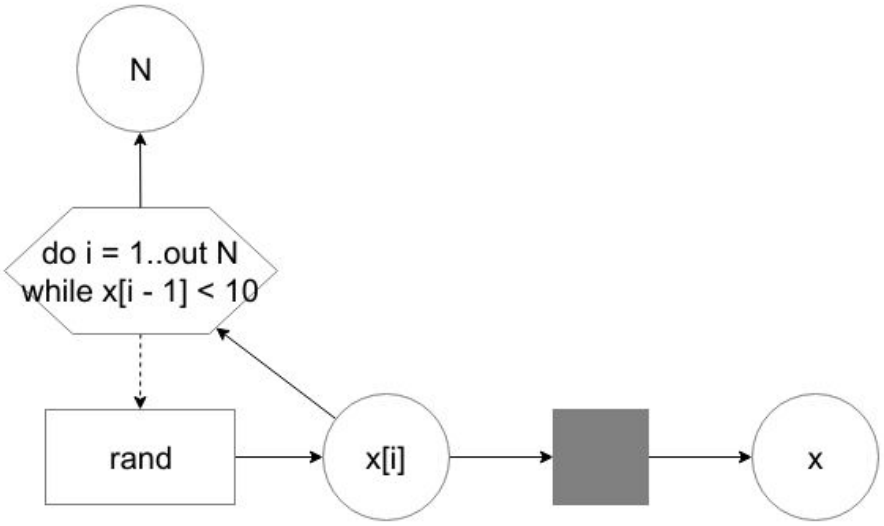


Рисунок 5 – цикл while

## 2.7 Атрибуты

Некоторые атрибуты элементов (например, имена операций) не отображаются в визуальном языке и оставлены на реализацию в визуальной веб-среде. Другие атрибуты (например, название переменной счетчика цикла) отображены явно в языке в соответствующих компонентах визуального языка.

# 3 Разработка визуальной веб-среды

## 3.1 Выбор средств разработки веб-среды

В качестве языка разработки был выбран JavaScript, так как он хорошо подходит для разработки для браузеров, а также позволит встроить разрабатываемую систему в существующий проект HPC Community Cloud [15], упрощающий взаимодействие пользователей с высокопроизводительными компьютерами через веб-интерфейс.

Далее встала задача выбора библиотеки для рисования графов. К ней были предъявлены следующие требования:

1. возможность рисовать заданное графическое представление фрагментированного алгоритма, спроектированное на основе языка LuNA;
2. бесплатность;
3. возможность динамического редактирования графов;
4. наличие развитой документации библиотеки.

Были рассмотрены 20 библиотек, позволяющие рисовать графы. С помощью представленных критериев из них была выбрана наиболее подходящая - MxGraph. Остальные были отсеяны по следующим причинам:

1. Не свободно распространяемые: JointJS, Rappid, GoJS, Mindfusion Diagram, JsPlumb;
2. Невозможность рисовать приведенное выше графическое представление фрагментированного алгоритма: jsUML2, Nomnoml, State.js, Cytoscape.js;
3. Невозможность динамического редактирования графов в реальном времени: Mermaid.js, Dagre-d3;

4. Низкоуровневая библиотека (является векторным редактором графических примитивов): Raphael, Fabric.js, Paper.js, P5.js;

5. Неудобный интерфейс для пользователя во время создания новых вершин и ребер: D3, Vis.js;

6. Плохая документация: Diagram.js, Draw2D.

Выбранная библиотека MxGraph свободно распространяется (по лицензии Apache 2.0), с ее помощью можно интерактивно строить графы, а ее документация содержит много примеров использования.

## 3.2 Внутреннее представление программы

Одним из требований к веб-среде является отображение фрагментированного алгоритма с возможностью настраивать параметры некоторых его элементов. Поэтому ставится задача разработки внутреннего представления визуальных фрагментов языка на основании внутреннего представления графических примитивов библиотеки.

Так как в библиотеке MxGraph граф хранится в XML-виде, было выбрано представление вершин графа в виде XML-элементов с атрибутами, что позволяет каждый элемент визуального языка представить своим XML-элементом, а атрибуты позволяют сохранить параметры компонентов.

Для каждого визуального элемента было разработано внутреннее представление на базе XML, которое содержит собственные атрибуты, связанные с хранением алгоритма, а также атрибуты, связанные с отображением элемента с помощью библиотеки: координаты элемента и его стиль (ширина, высота, форма, цвет). Например, XML-элемент Operation имеет собственные атрибуты для имени импортируемой функции из подключаемого модуля и имени данного элемента в графе. Дуги хранят в своих атрибутах

число, которое позволит упорядочить переменную для аргументов в соответствующей дуге операции.

Представление графа в виде XML позволяет средствами библиотеки делать импорт и экспорт XML-кода для сохранения графа между сеансами работы. В разработанной веб-среде была добавлена эта функциональность.

Следующим требованием является возможность добавления описаний подпрограмм, реализующих операции фрагментированного алгоритма. При добавлении описания подпрограммы информация о ней добавляется в две структуры: одна для отображения пользователю уже добавленных описаний, другая содержит сгенерированный шаблон соответствующей подпрограммы на языке C++ (подпрограмма с пустым телом).

В качестве первого решения проблемы представления программ, состоящих из большого числа компонентов, была реализована возможность масштабирования холста для рисования.

## 3.3 Генерация LuNA-программы

LuNA-программа состоит из двух частей: код на LuNA и код на C++. Код на LuNA включает структуру разработанного алгоритма (переменные, операции и связи между ними). Код на C++ содержит реализации операций алгоритма в виде подпрограмм на языке C++.

### 3.3.1 Генерация кода C++

Генератор кода для подпрограмм принимает на вход список подпрограмм, задаваемых пользователем веб-среды, и выдает на выход шаблон с прототипами подпрограмм.

В веб-среде есть возможность добавлять подпрограммы в соответствующий список, задав имя метода, имя соответствующей операции в графе, число и типы аргументов.

### 3.3.2 Генерация кода на языке LuNA

Генератор LuNA-кода принимает на вход внутреннее библиотечное представление алгоритма, и выдает на выход LuNA-код.

Средства библиотеки позволяют получить список всех элементов графа (вершин и ребер), отсортированных в порядке их добавления на поле рисования, для дуги получить target и source (вершины, которые соединяет дуга), а для любой вершины получить список всех инцидентных дуг. Этих возможностей достаточно для генерации кода.

Алгоритм генерации LuNA-кода представлен далее.

1. Происходит получение списка всех элементов алгоритма (вершин и дуг графа) средствами библиотеки.
2. Для каждого стартового элемента цикла происходит генерация кода этого цикла (с пометкой посещенных элементов). По атрибутам стартового элемента цикла генерируется заголовок цикла for, а множество операций для тела цикла получается с помощью обхода графа. Обход графа тела цикла (из-за data flow семантики не имеет значения какой используется обход: в глубину или ширину) происходит по прямым дугам, обрабатывая только операции и границы тела цикла. При встрече границы тела цикла происходит завершение данной ветви обхода. Если в графе будет вложенный цикл, то поведение генерации считается неопределенным.
3. После этого происходит генерация кода для всех непомеченных операций и переменных.

Для каждой переменной код генерируется следующим образом: имя переменной добавляется в блок кода, объявляющий все переменные алгоритма.

Алгоритм для генерации кода для операций следующий.

1. Для операции ищутся все условные операторы, от которых она зависит, и при их наличии формируется код условия.
2. Генерируется код для входных (по прямым дугам) и выходных (по обратным дугам) аргументов операции с их упорядочиванием по числовым атрибутам на дугах (чем меньше на дуге число, тем переменная идет раньше в аргументах). При отсутствии аргументов на дугах атрибут считается равным 0 по умолчанию.

Временная сложность алгоритма является O(N+M) (где N - число вершин, M - число ребер), так как происходит два прохода по списку длинной N+M, а при обходе каждое ребро посещается только один раз.

# 4 Результаты

## 4.1 Внешний вид веб-среды

Был создан прототип веб-среды разработки (рисунок 6). Вверху веб-среды располагается меню с командами, в центре – поле для рисования, слева панель с перетаскиваемыми на поле компонентами визуального языка. Веб-среда поддерживает импорт и экспорт схемы через XML-формат с использованием меню (XML-представление графа сохраняет расположение всех объектов на поле, все зависимости между объектами и все их атрибуты).

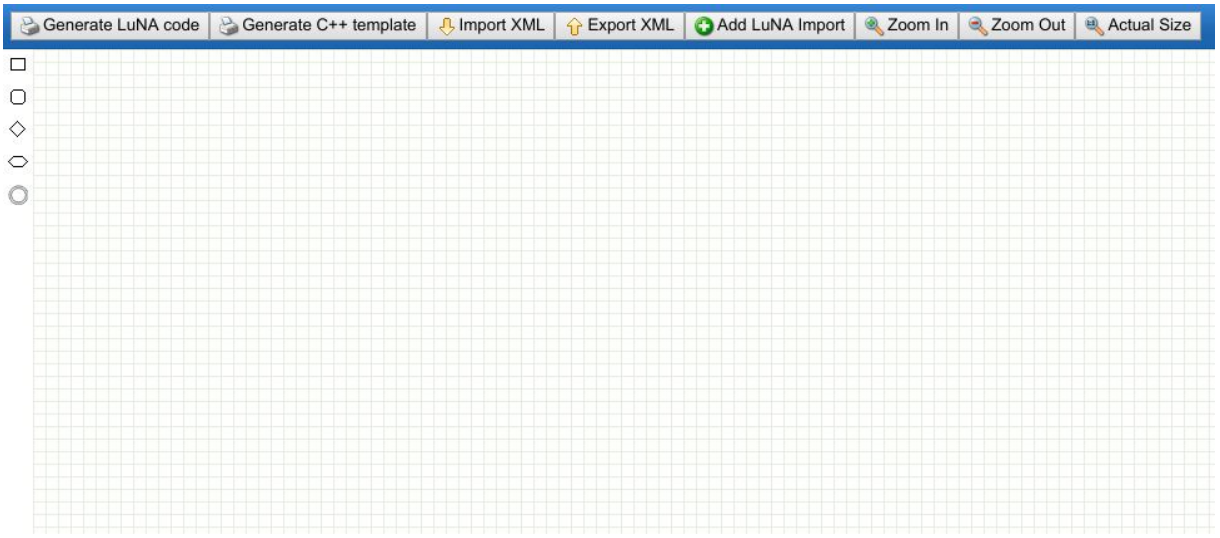


Рисунок 6 – прототип веб-среды разработки

## 4.2 Пример генерации LuNA-программы

В качестве примера программы представлен расчет скалярного произведения (рисунок 7). В нем происходит инициализация двух массивов `a` и `b`, затем цикл вычисляет выражение `a[i] * b[i]` для каждого `i`, записывая результаты в отдельный массив. Следующий цикл суммирует все элементы массива с произведениями, используя вспомогательный массив `s`, далее итоговый результат `s[N]` выводится на печать.

Сгенерированный LuNA-код (отступы расставлены вручную для читаемости кода):

```
import c_initA as initA;
import c_initB as initB;
import c_init0 as init0;
import c_mul as mul;
import c_sum as sum;
import c_print as print;

sub main()
{
    df a, b, ab, s, N;
    for i = 0..N-1
    {
        mul(#in i, a[i], b[i], #out ab[i]);
    }
    for i = 0..N-1
    {
        sum(#in s[i], ab[i], i, #out s[i+1]);
    }
    initA(#in #out a);
    initB(#in #out b);
    init0(#in #out s[0]);
    print(#in s[N]);
}
```

Сгенерированный C++-код (для компактности представлены только две подпрограммы):

```
#include "ucenv/ucenv.h"
extern "C"
void c_initA(OutputDF& dfo0)
{
    dfo0.setValue<int>(0);
}
extern "C"
void c_sum(int i0, int i1, int i2, OutputDF& dfo0)
{
    dfo0.setValue<int>(0);
}
```

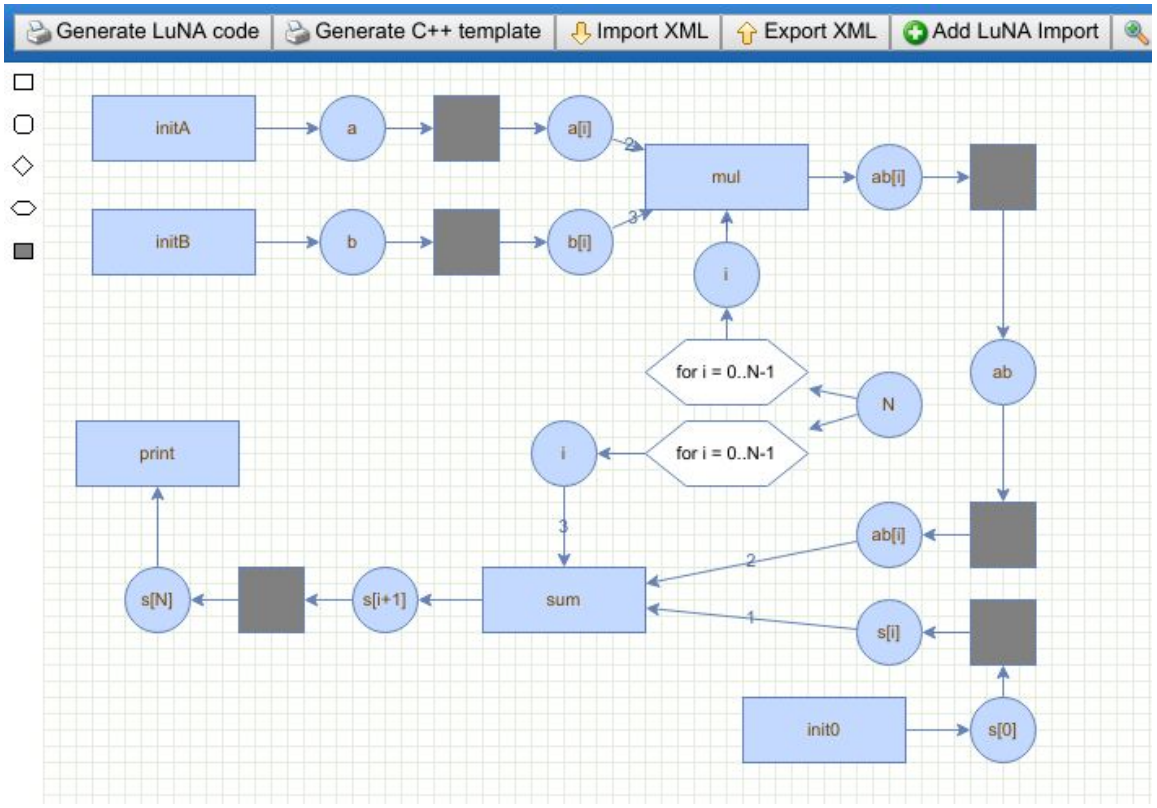


Рисунок 7 – скалярное произведение

# ЗАКЛЮЧЕНИЕ

В итоге был создан прототип визуального языка, позволяющий разрабатывать ограниченный подкласс фрагментированных алгоритмов. Также был создан прототип визуальной веб-среды разработки, позволяющий разрабатывать и редактировать фрагментированные алгоритмы на разработанном языке, транслировать их в текст программы на языке LuNA и шаблон модуля на языке C++, а также осуществлять их загрузку и сохранение в виде XML-представления.

В работе был применен подход визуального программирования к разработке LuNA-программ. Наличие возможности рисовать программы упростит знакомство с языком для программистов, а генерация кода может ускорить написание базовых программ.

Работа была представлена на 57-ой Международной научной студенческой конференции в секции “Информационные технологии” в подсекции “Программная архитектура и теоретическое программирование” [16].

Проект является пилотным. В планах реализация поддержки вложенных циклов и while-циклов, проверка на ошибки при редактировании программы, возможность запуска кода (локального или удаленного) из веб-среды, встраивание в систему HPC Community Cloud.

Выпускная квалификационная работа выполнена мной самостоятельно и с соблюдением правил профессиональной этики. Все использованные в работе материалы и заимствованные принципиальные положения (концепции) из опубликованной научной литературы и других источников имеют ссылки на них. Я несу ответственность за приведенные данные и сделанные выводы.

Я ознакомлен с программой государственной итоговой аттестации, согласно которой обнаружение плагиата, фальсификации данных и ложного цитирования является основанием для не допуска к защите выпускной квалификационной работы и выставления оценки «неудовлетворительно».

| _____ | _____ |
|---|---|
| ФИО студента | Подпись студента |

« ____ »_____20 __г.

*(заполняется от руки)*

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ И ЛИТЕРАТУРЫ


1. Malyshkin V. E., Perepelkin V. A. LuNA fragmented programming system, main functions and peculiarities of run-time subsystem // International Conference on Parallel Computing Technologies. – Springer, Berlin, Heidelberg, 2011. – С. 53-61.
2. Визуальный язык ДРАКОН. URL: https://drakon.su (дата обращения: 31.05.2019).
3. LabVIEW User Manual. National Instruments Corporation. URL: http://www.ni.com/pdf/manuals/320999b.pdf (дата обращения: 31.05.2019).
4. Ong C. M. et al. Dynamic simulation of electric machinery: using MATLAB/SIMULINK. – Upper Saddle River, NJ : Prentice hall PTR, 1998. – Т. 5.
5. Легалов А. И. Функциональный язык для создания архитектурно-независимых параллельных программ // Вычислительные технологии. – 2005. – Т. 10. – №. 1.
6. Mosconi M., Porta M. Iteration constructs in data-flow visual programming languages // Computer languages. – 2000. – Т. 26. – №. 2-4. – С. 67-104.
7. McLain P., Kimura T. D. Show and Tell user's manual. – 1986.
8. Cox P. T., Giles F. R., Pietrzykowski T. Prograph: a step towards liberating programming from textual conditioning // [Proceedings] 1989 IEEE Workshop on Visual Languages. – IEEE, 1989. – С. 150-156.
9. Rasure J. R., Williams C. S. An integrated data flow visual language and software development environment // Journal of Visual Languages & Computing. – 1991. – Т. 2. – №. 3. – С. 217-246.
10. Helsel R. Cutting your test development time with HP Vee: an iconic programming language. – Prentice-Hall, Inc., 1994.

11. Johnston W. M., Hanna J. R., Millar R. J. Advances in dataflow programming languages // ACM computing surveys (CSUR). – 2004. – Т. 36. – №. 1. – С. 1-34.

12. Dennis J. B. First version of a data flow procedure language // Programming Symposium. – Springer, Berlin, Heidelberg, 1974. – С. 362-376.

13. Arvind R. E., Thomas I. structures: An efficient data type for functional languages // Laboratory for Computer Science, MIT, TM. – 1980. – Т. 178.

14. Hurson A. R., Lee B., Shirazi B. Hybrid structure: A scheme for handling data structures in a data flow environment // International Conference on Parallel Architectures and Languages Europe. – Springer, Berlin, Heidelberg, 1989. – С. 323-340.

15. Городничев М. А., Малышкин В. Э., Медведев Ю. Г. HPC Community cloud: эффективная организация работы научно-образовательных суперкомпьютерных центров // Научный вестник Новосибирского государственного технического университета. – 2013. – №. 3. – С. 91-96.

16. Ижицкий Р. Л. Разработка и реализация веб-среды визуальной разработки фрагментированных программ // Информационные технологии : Материалы 57-й Междунар. науч. студ. конф. 14–19 апреля 2019 г. / Новосиб. гос. ун-т. — Новосибирск : ИПЦ НГУ, 2019. — С. 101.