МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
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
Направленность (профиль): Программная инженерия и компьютерные науки

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА

Макаренко Данила Евгеньевича

Тема работы:

РАЗРАБОТКА И РЕАЛИЗАЦИЯ ЧАСТНЫХ АЛГОРИТМОВ УПРАВЛЕНИЯ РАСПРЕДЕЛЁННЫМИ ДАННЫМИ В СИСТЕМЕ LUNA

«К защите допущена»
Заведующий кафедрой,
д.т.н., профессор
Малышкин В.Э. /……..
(ФИО)/ (подпись)
« 30 »…мая…2021г.

Руководитель ВКР
д.т.н., профессор
заведующий каф. ПВ ФИТ НГУ
Малышкин В.Э. /……...
(ФИО)/ (подпись)
« 27 »… …мая……2021г.

Соруководитель
ст. преп. каф. ПВ ФИТ НГУ
Перепёлкин В.А. / ……...
(ФИО)/ (подпись)
« 27 »……мая……2021г.

Новосибирск, 2021

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НОВОСИБИРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
(НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ, НГУ)
Факультет информационных технологий

Кафедра параллельных вычислений……
(название кафедры)

Направление подготовки 09.03.01 Информатика и вычислительная техника
Направленность (профиль): Программная инженерия и компьютерные науки

УТВЕРЖДАЮ

Зав. кафедрой Малышкин В.Э.
(фамилия, И., О.)
…….
(подпись)
« 18 »……декабря……2020г.

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ БАКАЛАВРА

Студент*у(ке)*…Макаренко Данилу Евгеньевичу……, группы…17202…
(фамилия, имя, отчество, номер группы)

Тема…Разработка и реализация частных алгоритмов управления распределёнными данными в системе LuNA……
(полное название темы выпускной квалификационной работы)

утверждена распоряжением проректора по учебной работе от 17.12.2020 № 0451
Срок сдачи студентом готовой работы……31 мая…2021 г.
Исходные данные (или цель работы): разработать и реализовать в системе фрагментированного программирования LuNA алгоритм автоматического распределения ресурсов……
Структурные части работы: обзор литературы, постановка задачи, разработка и реализация алгоритмов, тестирование……

Руководитель ВКР
заведующий. каф. ПВ ФИТ НГУ,
д.т.н, доцент
Малышкин В.Э. /……..
(ФИО)/(подпись)
« 18 »……декабря……2020г.

Задание принял к исполнению
Макаренко Д.Е. /……..
(ФИО)/(подпись)
« 18 »…декабря……2020г.

Соруководитель
ст. преп. каф. ПВ ФИТ
Перепёлкин В.А. / ……..
(ФИО)/(подпись)
« 18 »……декабря……2020г.

Содержание

ВВЕДЕНИЕ

Разработка параллельных программ численного моделирования сложна, трудоемка и требует специальной квалификации. В процессе написания параллельной программы специалисту необходимо решить проблемы, такие как декомпозиция данных и вычислений, распределение данных и вычислений по узлам, организация параллельной обработки распределенных данных, настройка на доступные ресурсы и обеспечение масштабируемости. Для облегчения процесса написания параллельной программы разрабатываются системы автоматизированного программирования, которые упрощают создание параллельных реализаций крупномасштабных численных моделей для распределенных вычислительных систем. В таких системах параллельная программа конструируются из блоков вычислений и их параметров. Каждый блок вычислений определяет независимый процесс, получающий новые значения из входных параметров. Программы, полученные с помощью данных систем, управляются исполнительными системами, использующими различные алгоритмы для распределений вычислений и данных, необходимые для улучшения эффективности потребления ресурсов мультикомпьютера и времени исполнения программы, что позволяет улучшить производительность и масштабируемость параллельных программ численного моделирования. Под эффективностью понимается эффективность по времени. В свою очередь качественное распределение ресурсов должно обеспечивать равномерную нагрузку на компоненты вычислительной системы и минимизировать коммуникацию между компонентами. Также для увеличения производительности исполнения параллельных программ часто необходимо обеспечение динамического свойства параллельных программ, такого как динамическая балансировка загрузки.

Проблемой является то, что нет общего алгоритма, который бы эффективно распределял ресурсы на всех системах автоматизации

параллельного программирования и был бы применим к широкому кругу задач, реализуемых на данных системах.

Среди существующих систем выделяется система конструирования параллельных программ LuNA [1 с. 4, 2 с. 60]. Ее преимуществами являются распределенная память, универсальная модель вычислений, крупноблочный параллелизм, параллельное представление алгоритма, а также наличие управляемого исполнения программы. Исходя из этих преимуществ было принято решение реализовать и протестировать новый алгоритм распределения ресурсов в этой системе

Цель работы заключается в разработке алгоритма распределения фрагментов данных в системе конструирования параллельных программ LuNA.

Достижение поставленной цели требует решения следующих задач:

1. Анализ существующих алгоритмов распределения
2. Выделение основных требований для алгоритма распределения
3. Разработка и реализация алгоритма распределения фрагментов данных в системе LuNA
4. Оценка эффективности реализованного алгоритма

Научная новизна заключается в решении необходимых задач для конструирования фрагментированных программ с использованием алгоритма распределения ресурсов на основе статического анализа.

Практической ценностью является разработка алгоритма распределения ресурсов на основе статического анализа, который повысит эффективность исполнения фрагментированных программ.

1 Разработка алгоритма распределения фрагментов данных

1.1 Необходимые определения

Системы автоматизированного параллельного программирования на вход получают исходный код с алгоритмом. Алгоритм представлен в виде множества блоков вычислений, имеющих входные и выходные параметры. Изначально алгоритм имеет управление, которое определяется зависимостями по данным, и не включает в себя распределение ресурсов. Из этого следует что существует множество способов исполнения алгоритма, которые отличаются между собой используемым распределением фрагментов данных и вычислений. Из этого следует, что системе необходимо самостоятельно отобразить блоки вычислений и данные на узлы определенной вычислительной системы. Множество способов распределения дает системе свободу выбора, что может сказаться на эффективности исполнения как положительно, так и отрицательно, поэтому в системы автоматизированного параллельного программирования внедряют алгоритмы, которые бы строили отображение блоков вычислений и данных на вычислительные узлы конкретной системы. Такие алгоритмы позволяют улучшить эффективность исполнения программы.

1.2 Анализ существующих алгоритмов

Одним из подходов распределения данных являются алгоритмы статического анализа [3 с. 115, 4 с. 226, 5 с. 827], которые используются в компиляторах. Ограниченность данных алгоритмов заключается в том, что принятие решение происходит во время компиляции. Следовательно, они не могут принять решения во время исполнения программы, что отражается на эффективности исполнения.

Еще одним подходом является использование масштабируемых диффузионных алгоритмов [6 с. 757, 7 с. 424, 8 с. 884]. Основная идея диффузионных алгоритмов состоит в распределении нагрузки от более нагруженных узлов менее нагруженным в некоторой окрестности узла. В общем виде такой алгоритм состоит из двух этапов. Первый этап заключается в том, что

каждый узел определяет количество избыточной нагрузки, в соответствии с нагрузкой узлов в окрестности. Затем во втором этапе производится передача нагрузки в соответствии с методикой расчета количества нагрузки, необходимой для передачи. Плюсами таких алгоритмов являются децентрализованность и хорошая масштабируемость. Однако недостатком является то, что диффузионные алгоритмы могут допускать возникновение глобального дисбаланса в рамках допустимого градиента нагрузки.

Балансировка нагрузки также может быть реализована с помощью рекурсивной бисекции [9 с. 110, 10 с. 33]. Такие алгоритмы основаны на рекурсивном фрагментировании пространства моделирования для уменьшения коммуникационных затрат. Основной идеей данных алгоритмов является то, что пространство о разбивается на равные по нагрузке части. Получившиеся части, равные по вычислительной нагрузке, параллельно исполняются на узлах мультикомпьютера. При возникновении дисбаланса нагрузки в процессе исполнения производится уточнение границ частей области, в которой возник дисбаланс. Минусом данного алгоритма является то, что он подходит не для всех задач, так как иногда может потребоваться повторное использование этого алгоритма для устранения дисбаланса.

Распределения данных в распределенной базе данных имеет много схожих черт. В статьях [11 с. 1339, 12 с. 79] рассматриваются разные типы фрагментации: горизонтальная, вертикальная и смешанная. Также рассматривается несколько методов распределения фрагментированных данных. Наиболее эффективным из всех является метод определения не избыточного распределения. Его реализация заключается в том, чтобы дать оценку эффективности каждого возможного распределения и дать один узел из всех. Данный способ дает возможность размещения части данных на определенном узле. Минусом такого алгоритма является не масштабируемость на большое число данных и вычислений.

Хорошее увеличение эффективности получается с помощью распределения данных определенной структуры. Примером такой структуры являются плиточные массивы [13 с. 483]. То есть массив данных сначала разбивается на прямоугольные блоки(“плитки”), затем эти блоки распределяются по узлам мультикомпьютера. Разбиение на такие блоки может быть иерархическим, регулярным и произвольным. При регулярном разбиении все блоки имеют одинаковый размер и выровнены относительно друг друга. Из-за такой структуры возникает ограничение, требующие прямоугольной формы плиток, что иногда мешает осуществить сбалансированное разбиение данных на плитки. Вторым ограничением является предполагаемая статичность разбиения, следовательно, динамическое изменение в ходе работы программы невозможно. Из этих двух ограничений можем понять, что этот метод подойдет малому числу задач.

Рассмотрим алгоритм распределения данных Rope из статьи [14 с. 3]. Алгоритм использует отображение сетки фрагментов данных на одномерный числовой диапазон. Каждому фрагменту данных назначается его (целочисленная) координата в диапазоне. Отображение фрагментов на диапазон делается таким образом, чтобы соседние фрагменты данных отображались на одну и ту же координату или на близкие координаты в диапазоне. Отображение фиксируется перед началом исполнения фрагментированной программы и не меняется в ходе ее исполнения.

Для распределения фрагментов данных по узлам мультикомпьютера диапазон разбивается на непересекающиеся смежные сегменты по числу вычислительных узлов, каждый узел получает свой сегмент. Предполагается, что узлы объединены в линейную топологию, что позволяет осуществить отображение сегментов на узлы один-в-один. Резиденцией фрагментированных данных является тот узел, сегменту которого в данный момент принадлежит координата этого фрагмента данных. Таким образом, поиск резиденции каждого фрагмента данных заключается в поиске узла с нужной координатой. Так как

координаты упорядочены по узлам, поиск сегмента может быть осуществлен с помощью прохода по линейке узлов, с использованием только локальных взаимодействий между вычислительными узлами. Такой алгоритм не всегда возможно реализовать, так как иногда такое отображение не удается построить.

Алгоритм Rope сохраняет соседство не по всем измерениям, так как используется одномерный числовой диапазон. Чтобы избежать этого алгоритм Patch использует отображение на многомерную область координат. Область представляется в виде регулярной декартовой n-мерной сетки ячеек, каждая ячейка имеет свою n-мерную целочисленную координату. Фрагменты данных отображаются на ячейки. Отображение фрагментов на ячейки задается перед стартом программы и далее не меняется.

Для динамической балансировки нагрузки в алгоритме Patch [15 с. 2], как и в алгоритме Rope, применяется диффузионный подход. Все вычислительные узлы распределяются между перекрывающимися группами, каждая группа узлов включает в себя центральный узел группы и все смежные с ним узлы в топологии; центральный узел каждой группы может быть одновременно смежным узлом в других группах, что обеспечивает взаимодействие между группами. Отличие заключается в том, что в Rope нагрузка рассчитывается для каждой координаты, а в Patch для каждой ячейки. Передача нагрузки производится путем миграции координат/ячеек. Плюсом является масштабируемость на большое число узлов и снижение общего коммуникаций в ходе исполнения. Так как используется диффузионный подход, то алгоритм имеет минусы, перечисленные выше в диффузионных алгоритмах, но их можно устранить, изменяя частоту вызова балансировки и объем нагрузки, передаваемой между узлами.

Проанализировав вышеперечисленные алгоритмы можем заметить, что они хорошо повышают эффективность исполнения параллельных программ, но у всех есть свои недостатки такие как: Ограниченность кругом задач,

необходимость запускать алгоритм несколько раз для исключения дисбаланса, что сказывается на эффективности в худшую сторону, невозможность использования в процессе исполнения программ и не масштабируемость на большое число узлов.

Все эти недостатки дают возможность для улучшения или создания алгоритмов с новыми свойствами, которые бы исключали некоторые из выше вышеперечисленных минусов.

1.3 Постановка задачи

Программы численного моделирования часто содержат внутри себя большое число вычислений, из-за чего требуют много вычислительных ресурсов для исполнения. Поэтому такие программы часто запускают на мультикомпьютерах. Мультикомпьютер - это множество вычислительных узлов, связанных по сети. Распараллеливание происходит за счет распределения крупных задач по вычислительным узлам мультикомпьютера. Поэтому программы разбивают на фрагменты вычислений. Фрагмент вычислений - это независимая единица программы, которая состоит из фрагмента кода и входных/выходных фрагментов данных. Фрагментом данных называется часть общей структуры данных. Следовательно, нам необходимо распределить фрагменты вычислений и данных по вычислительным узлам. Для того, чтобы сконструировать распределение нам необходимо построить отображение фрагментов данных и вычислений на узлы вычислительной системы. Полученное отображение должно удовлетворять следующим критериям:

1. Высокий уровень производительности системы при использовании полученного отображения
2. Уменьшение трудозатрат на разработку фрагментированной программы
3. Удовлетворение обширному кругу задач

Распределение может быть сконструировано динамически с помощью системы или же статически за счет добавления рекомендаций по распределению.

Рекомендации - это специальные аннотации в исходном коде программы, которые задают начальное распределение фрагментов данных и вычислений. Источником рекомендацию может быть специалист, разрабатывающий программу или модуль системы, который конструирует распределение автоматически. Мы можем анализировать данные рекомендации и использовать их для формирования распределения.

Таким образом, задача распределения фрагментов данных сводится к конструированию новых рекомендаций для назначения местоположения фрагментов данных на вычислительных узлах мультикомпьютера.

1.4 Идея предлагаемого решения

Идея предполагаемого решения основывается на том, что у нас есть фрагментированная программа, в которой уже частичного могут быть расставлены рекомендации по распределению фрагментов данных. Мы можем определить данные, которые используются вместе, если часть из их имеет рекомендации по распределению, то можно распространить эти рекомендации для фрагментов данных, для которых нет таких рекомендаций в исходном коде программы, за счет чего будут снижаться трудозатраты пользователя на написание фрагментированных программ.

Предлагаемое решение можно разделить на 3 основные части:

1. Выявление совместно используемых данных
2. Выявление данных имеющих распределение
3. Расстановка новых рекомендаций по распределению

Для выявления совместно используемых данных необходимо проанализировать входные и выходные фрагменты данных в каждом фрагменте вычислений, чтобы проанализировать необходимые нам данные мы должны рассмотреть абстрактное синтаксическое дерево нашей программы, которое строит система в процессе анализа исходного кода. Это дерево описывает подпрограммы и процедуры, которые были объявлены в программе. Процедуры и подпрограммы в свою очередь состоят из списка аргументов, списка операторов и списка рекомендаций. Оператором в этом дереве являются циклы,

вызовы подпрограмм, объявление фрагментов данных и фрагментов вычислений. Операторы включают в себя список аргументов, которые являются выражениями. Оператор описывающий фрагмент вычислений является массовым оператором и потребляет фрагменты данных на вход или выход, а также массивы, состоящие из фрагментов данных. Следовательно, для чтения аргументов нам нужно спуститься по дереву до оператора описывающий фрагмент вычислений и рассматривать уже какие фрагменты данных он потребляет. Также в нашем абстрактном синтаксическом дереве есть ветви, которые определяют отображение фрагмента данных на узлы вычислительной системы. Допустим у нас есть фрагмент данных на вход, для него происходит поиск в дереве записи, которая соответствует данному фрагменту и задает его распределение, если такая запись существует, то фрагмент данных считается распределенным, а если запись отсутствует, то мы считаем его нераспределенным. Тогда если мы имеем два фрагмента данных, один из которых является распределенным, а другой нет, и если они имеют одинаковый индексный вид, то мы можем распространить распределение с одного фрагмента на другой. Следовательно, для распространения распределения мы должны модифицировать абстрактное синтаксическое дерево и добавить записи, которые будут соответствовать фрагментам данных, которые могут быть распределены схожим образом, но еще не были распределены. Для того, чтобы узнать могут ли данные использовать схожее распределение необходимо произвести анализ фрагментов данных. Заключительный этап состоит из обобщения результатов, полученных на предыдущих этапах работы и выбора распределения для совместных данных, для которых не были проставлены аннотации в исходном коде.

1.5 Описание алгоритма

Алгоритм на вход получает абстрактное синтаксическое дерево программы. Алгоритм совершает обход этого дерева, то есть спускается каждый раз на уровень вниз и рассматривает операторы для объявления фрагмента вычислений. В объявлении фрагмента вычислений мы рассматриваем входные и

выходные фрагменты данных, то есть смотрим наличие начального распределения и производим анализ на возможность распространить это распределение на фрагменты данных, которые являются аргументами того же фрагмента вычислений и являются нераспределенными. Рекомендации по начальному распределению для нужного нам фрагмента данных могут содержаться в ветвях, находящихся на более высоких уровнях, поэтому нам необходимо их хранить информацию о том на каком уровне мы находимся, и если на нашем уровне нет необходимых нам рекомендаций, то нам нужно пройти вверх по дереву и поискать необходимую рекомендацию. Описание алгоритма на псевдокоде приведено на рисунке 1.

```
Добавление_новых_рекомендаций(Абстрактное синтаксическое дерево АСД)
{
        Пока появляются новые рекомендации
        {
                Уровень = 0
                Контекст = массив рекомендаций с распределениями
                Для каждой подпрограммы из АСД
                {
                        Обработка списка операторов(Тело подпрограммы, уровень + 1, контекст)
                }
        }
}
Обработка списка операторов(список операторов, уровень, контекст)
{
                Для каждого оператора из списка операторов
                {
                        Если на текщем уровне есть рекомендации по распределию
                        {
                                Контекст[уровень] = список рекомендаций по распределению из оператора
                        }

                        Если тип оператора = вызов процедуры
                        {
                                Получаем аргументы оператора
                                Если у аргументов есть распределение
                                {
                                        Если можно распространить распределение
                                        {
                                                Добавляем новые рекомендаци для нераспределенных данных
                                        }
                                }Иначе
                                {
                                        локальный уровень = уровень
                                        Если контекст[локальный уровень] не имеет распределение
                                        {
                                                локальный уровень -= 1
                                        }
                                        Если нашлось распределение
                                        {
                                                Если можно распространить распределение
                                                {
                                                        Добавляем новые рекомендаци для нераспределенных данных
                                                }
                                        }
                                }
                        }
                        Если у оператора есть тело
                        {
                                Обработка списка операторов(тело , уровень + 1, контекст)
                        }
                }
}
```

Рисунок 1 – описание алгоритма с помощью псевдокода

Анализ возможности использовать схожее распределение может быть простой задачей если входные и выходные аргументы имеют одинаковую структуру и отличаются лишь названием. Пример такой ситуации представлен на рисунке 2. Допустим у нас фрагмент данных X имеет статическое распределение, после анализа мы понимаем, что можем использовать такое же распределение для фрагментов данных Y и Z, после чего для них добавляем в

исходный код рекомендации с таким же распределением, как и у фрагмента данных X.

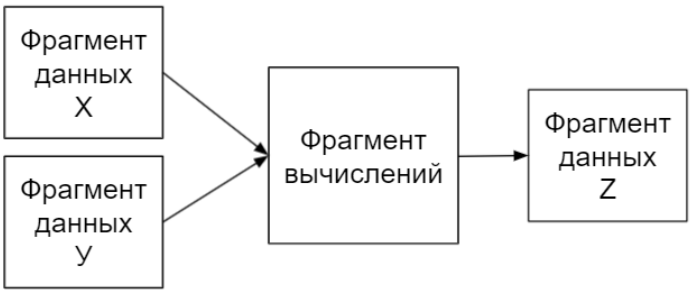

Рисунок 2 – пример совместно используемых данных

Также анализ может быть сложной задачей если в процессе выбора распределения возникают конфликтные ситуации. Такие ситуации могут возникать если структура входных и выходных фрагментов данных отличается, например, один из фрагментов данных в своей структуре имеет смещение и/или масштабирование, такой пример можно увидеть на рисунке 3. В такой ситуации сложно выбрать распределение для фрагмента данных Y[i]. Нам необходимо учесть, как этот фрагмент данных используется в других фрагментах вычислений, если этот фрагмент данных больше нигде не используется, то мы можем распределить их также как фрагмент X[i] и Z[i], а если используется, то можем взять среднее арифметическое.

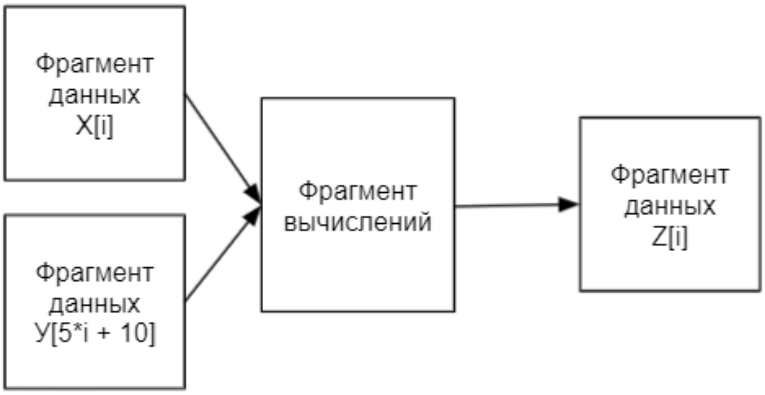

Рисунок 3 – пример конфликтной ситуации

Конфликтные ситуации также могут возникать если в одном фрагменте вычислений используются два фрагмента данных отличающихся лишь индексом. Пример такой ситуации представлен на рисунке 4. В таком случае

решением будет распределить фрагмент данных Y[i] на i вычислительном узле, если фрагмент данных X[i] расположен тоже на i вычислительном узле.

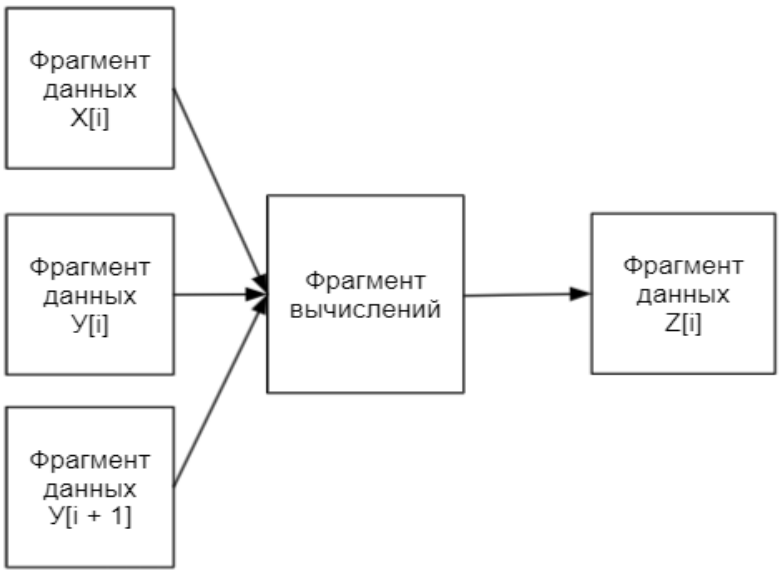

Рисунок 4 – пример конфликтной ситуации

1.6 Характеристики предлагаемого решения

Данный алгоритм снижает трудозатраты специалиста на разработку фрагментированной программы, за счет отсутствия необходимости указывать распределение для каждого фрагмента данных. Алгоритм может использоваться для широкого круга задач, так как для каждой задачи разработчик самостоятельно определяет необходимое распределение для фрагментов, путем добавления рекомендаций по распределению в исходный код. Распределение по вычислительным узлам будет равномерным и обеспечит высокую производительность системы, так как начальное распределение задается за счет специалиста, который знает необходимые условия для эффективного исполнения программы.

2 Реализация алгоритма

2.1 Система фрагментированного программирования LuNA

Входной язык LuNA – теоретико-множественный, единственного присваивания и единственного исполнения фрагментов вычислений [16 с. 48]. Фрагменты данных и вычислений задаются рекурсивно перечислимыми множествами с использованием индексных выражений. Управление в LuNA-программе задается отношением частичного порядка на множестве фрагментов вычислений. Дополнительно имеются операторы-рекомендации по распределению ресурсов вычислителя, по определению требуемого порядка исполнения фрагментов вычислений. Средства задания приоритетов исполнения фрагментов вычислений используются run-time системой для выбора наиболее подходящего фрагмента на исполнение.

LuNA имеет два уровня преобразования фрагментированного алгоритма в программу: Компиляция, Исполнение:

• Компилятор формирует исполняемое представление исходного файла с алгоритмом,в которое вкладывает статические решения о способе исполнения, используя только информацию о свойствах входного алгоритма.

• Run-time система принимает решения в процессе вычислений, которые могут быть сделаны только динамически. В их числе выбор фрагмента вычислений на исполнение, динамическая балансировка нагрузки узлов вычислителя, динамическое распределение ресурсов, включая назначение процессора для исполнения фрагмента вычислений и многое другое.

На рисунке 5 представлена структура системы LuNA.

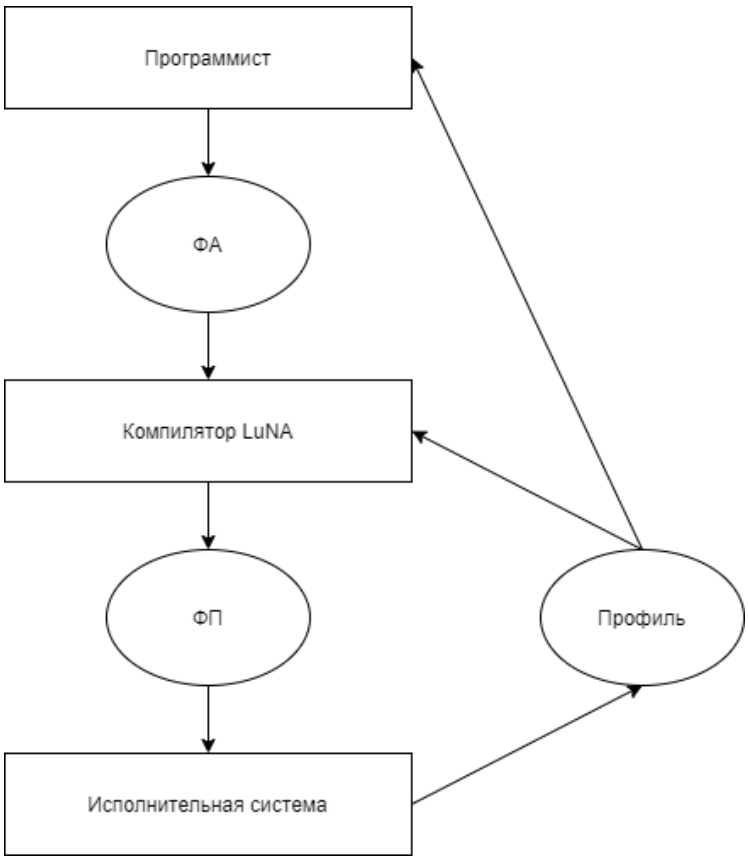

Рисунок 5 – структура системы LuNa

2.2 Этапы работы с LuNA-программой

Для получения фрагментированной параллельной программы необходимы следующие этапы:

1. Описание чистого фрагментированного алгоритма
2. Добавление рекомендаций по исполнению
3. Компиляция
4. Исполнение
5. Пост-анализ

Каждый из этих этапов вносит свой вклад в эффективность получаемой на выходе параллельной программы.

2.3 Процесс компиляции в системе LuNA

На вход компилятор получает файл с программой на языке LuNa, которую переводит в файл с программой на языке C++. Новый файл состоит из блоков, каждый из которых представляет фрагмент вычислений в виде отдельной функции. Также компилятор учитывает рекомендации, расставленные

пользователем и добавляет изменения в получаемый на выходе C++ файл с итоговой реализацией алгоритма. В дальнейшем эти изменения помогают runtime системе LuNA эффективнее выполнить необходимый алгоритм. Получаемый C++ файл представлен на рисунке 6.

```
1: #include "ucenv.h"
2:
3: extern "C" void c_helloworld();
4: // MAIN
5: BlockRetStatus block_0(CF &self)
6: {
7:      self.NextBlock=1;
8:      return CONTINUE;
9: }
10:
11: // STRUCT: sub main()
12: BlockRetStatus block_1(CF &self)
13: {
14: // GEN BODY: sub main()
15:     { // FORK_BI: cf _l7: hello_world();
16:             CF *child=self.fork(2);
17:     }
18:     return EXIT;
19: }
20: // BI_EXEC: cf _l7: hello_world();
21: BlockRetStatus block_2(CF &self)
22: {
23:     if (self.migrate(CyclicLocator(0))) {
24:             return MIGRATE;
25:     }
26:     {
27:             // EXEC_EXTERN cf _l7: hello_world();
28:             c_helloworld();
29:     }
30:     return EXIT;
31: }
32: extern "C" void init_blocks(BlocksAppender add)
33: {
34:     bool ok=true;
35:
36:     ok = ok && add(block_0)==0;
37:     ok = ok && add(block_1)==1;
38:     ok = ok && add(block_2)==2;
39:
40:     assert(ok);
41: }
```

Рисунок 6 – листинг скомпилированной LuNA программы.

2.4 Рекомендации по исполнению

При описании фрагмента вычислений в его теле указываются различные рекомендации по обработке этого фрагмента. Они несут в себе дополнительную информацию об алгоритме и используются для принятия статический решений с помощью компилятора, улучшая нефункциональные свойства полученной реализации.

Все рекомендации могут быть одного из следующих видов:

1. Рекомендации, определяющие расположение фрагментов (locator_cyclic)
2. Рекомендации, определяющие время жизни фрагментов (request, req_count, delete)

Пример LuNA-программы с использованием рекомендаций показан на рисунке 7.

```
1: import c_init(int, name) as init;
2: import c_iprint(int) as iprint;
3:
4: sub main()
5: {
6:      df x;
7:      cf a: init(7, x) @ {
8:              req_count x=1;
9:              locator_cyclic: 1;
10:     };
11:     iprint(x) @ {
12:             request x;
13:             locator_cyclic: 2;
14:     };
15: } @ {
16:     locator_cyclic x => 3;
17: }
```

Рисунок 7 – листинг программы на языке LuNA

2.5 Описание реализации

Алгоритм был реализован на языке Python. Для реализованного алгоритма было создано руководство программиста в приложении А. Реализация состоит из двух основных частей:

1. Добавление в компилятор языка LuNA механизма определения совместно используемых данных

2. Добавление механизма расстановки новых рекомендаций для совместно используемых данных.

Для определения совместно используются данные или нет необходимо сделать проход по исходному коду программы и просмотреть аргументы используемых функции, но тут нужно учесть работу уже существующего компилятора. Как мы знаем он преобразует изначальный алгоритм в C++ программу. Но этот процесс разделяется на несколько этапов. Сначала текст переводится в формат JSON, в котором содержится абстрактное синтаксическое дерево программы, затем происходит анализ этого файла и его последующая модификация. Поэтому мы добавляем в работу компилятора модуль, который обходит абстрактное синтаксическое дерево, для того чтобы узнать аргументы фрагмента вычислений, затем с помощью существующих инструментов компилятора узнаем, какие из них имеют статическое распределение, которое было задано с помощью рекомендации locator_cyclic. При одном обходе мы узнаем не все совместно используемые данные, так как у нас останутся не охвачены данные между которыми транзитивная зависимость. Для исключения этого недостатка было решено сделать режим компилятора, в котором он совершает обходы АСД, пока появляется новая информация о совместно используемых данных.

Для расстановки новых рекомендаций необходимо понять какое распределение мы будем использовать для совместных данных, так как могут возникать конфликтные ситуации, связанные с масштабированием, смещением и неоднозначным выбором распределения. Так как эта задача является не совсем простой в реализации, было принято решение сначала реализовать алгоритм, который будет работать с фрагментами данных имеющими одинаковую структуру и не будет учитывать совместные данные из-за которых могут возникнуть конфликтные ситуации.

3 Тестирование

Для проверки работоспособности алгоритма и его эффективности были проведены тесты с использованием программ написанных на языке LuNA. Тестирование было проведено на высокопроизводительном кластере MVS-10p МСЦ РАН [17].

3.1 Тестирование на задаче сложения векторов

Для проверки работоспособности алгоритма была написана программа, вычисляющая сумму двух векторов, в которой задается статическое распределение для одного из векторов, с помощью рекомендации locator_cyclic. Мы распределяем фрагмент данных x[i] на i-ый вычислительный узел. Листинг программы приведен на рисунке 8.

```
1: import c_init(int, name) as init;
2: import c_iprint(int) as iprint;
3: import c_vector_sum(value, value, name) as sum;
4:
5: sub main()
6: {
7:      df z, y, x;
8:
9:      for i=1..Counter{
10:                     cf a[i]: init(i, x[i]) @ {
11:                             locator_cyclic: 0;
12:                     };
13:             }
14:
15:     for i=1..Counter{
16:                     cf c[i]: init(i, y[i]) @ {
17:                             locator_cyclic: 0;
18:                     };
19:             }
20:
21:     for i=1..Counter{
22:                     cf d[i]: sum(x[i], y[i], z[i]) @ {
23:                     req_count z[i]=2;
24:                     locator_cyclic: i;
25:                     };
26:             }
27:
28:     cf b: display(z) @ {
29:             request z[Counter];
30:             locator_cyclic: 0;
31:     };
32: } @ {
33:     locator_cyclic x[i]=>i;
34: }
```

Рисунок 8 – листинг программы сложения векторов на языке LuNA

Программа запускалась на 4 узлах, с размерностью векторов 5000.

Результаты приведены в таблице 1.

Таблица 1 – результаты тестирования на задаче сложения векторов

Время работы без рекомендаций, с	Время работы с рекомендацией для x[i],с	Время работы с использованием алгоритма, с
30,945	27,039	23,658

Также для проверки работоспособности нам необходимо проверить наличие изменений в абстрактном синтаксическом дереве программы, так как алгоритм должен добавить новые рекомендации для распределения.

```
{
    "ruletype": "map",
    "property": "locator_cyclic",
    "type": "rule",
    "id": ["x", {"ref": ["i"] }]
},
{
    "ruletype": "map",
    "property": "locator_cyclic",
    "type": "rule",
    "id": ["y",{"ref": ["i"]}]
},
{
    "ruletype": "map",
    "property": "locator_cyclic",
    "type": "rule",
    "id": ["z",{"ref": ["i"]}]
}
```

Рисунок 9 – рекомендации по распределению после работы алгоритма

В результате работы алгоритма были добавлены рекомендации для фрагментов данных y и z, представленные на рисунке 9, а также получено небольшое ускорение программы.

3.2 Тестирование на задаче решения уравнения Пуассона

Для проверки алгоритма на работоспособность с более сложными структурами программ была взята задача с решением уравнения Пуассона. В реализации этой программы мы добавили статическое распределение для всех аргументов функции poi_part, кроме Fi[t][i], что можно увидеть на рисунке 10.

```
sub main()
{
        df Ro, Fi, iters, max, FiU, FiD, lmax, dummy;
        while max[t]>$EPS, t=0..out iters
        {
                poi_part(Ro[0], Fi[t][0], FiU[t][0], FiU[t][1], 0,
                                Fi[t+1][0], FiU[t+1][0], FiD[t+1][0], lmax[t+1][0])
                        --> (Fi[t][0], FiU[t][0], FiU[t][1]) @ {
                };
                for i=1..$FG_COUNT-2 {
                        poi_part(Ro[i], Fi[t][i], FiD[t][i-1], FiU[t][i+1], i,
                                        Fi[t+1][i], FiU[t+1][i], FiD[t+1][i], lmax[t+1][i])
                                -->(Fi[t][i], FiD[t][i-1], FiU[t][i+1]) @ {
                                locator_cyclic:i/11;
                        };
                } @ {
                        locator_cyclic:0;
                        unroll_at_once;
                }
                poi_part(
                        Ro[$FG_COUNT-1], Fi[t][$FG_COUNT-1],
                        FiD[t][$FG_COUNT-2], FiD[t][$FG_COUNT-1],
                        $FG_COUNT-1,
                        Fi[t+1][$FG_COUNT-1], FiU[t+1][$FG_COUNT-1],
                        FiD[t+1][$FG_COUNT-1], lmax[t+1][$FG_COUNT-1])
                                --> (
                                        Fi[t][$FG_COUNT-1],
                                        FiD[t][$FG_COUNT-2],
                                        FiD[t][$FG_COUNT-1])
                                @ {
                                        stealable;
                                locator_cyclic:0;
                        };
        }
} @ {
        locator_cyclic Ro[i]=>i;
        locator_cyclic iters=>0;
        locator_cyclic max[t]=>0;
        locator_cyclic FiU[t][i]=>i-1;
        locator_cyclic FiD[t][i]=>i+1;
        locator_cyclic dummy[i]=>0;
        locator_cyclic lmax[t][i]=>0;
}
```

Рисунок 10 – листинг алгоритма решения уравнения Пуассона.

Программа запускалась на 4 узлах вычислительного кластера. Результаты приведены в таблице 2.

Таблица 2 – результаты тестирования на задаче решения уравнения Пуассона

Время работы без алгоритма, с	Время работы с алгоритмом, с
56,396	50,248

В результате работы алгоритма была добавлена рекомендация для распределения аргумента Fi, представленная на рисунке 11.

```
{
    "ruletype": "map",
    "property": "locator_cyclic",
    "type": "rule",
    "id": ["Fi", {"ref": ["t"]},{ "ref": ["i"]}
}
```

Рисунок 11 – листинг рекомендации после работы алгоритма

3.3 Тестирование на задаче перемножения матриц

Для того, чтобы показать, что не всегда получается применить алгоритм была взята задача умножения матриц. Реализация алгоритма не учитывает совместные данные, которые имеют сложную структуру. Например, было бы логичным задать начальное распределение для одной из матриц и проверить получится ли распространить это распределение на вторую и итоговую матрицу. Для проверки результата работы алгоритма в такой ситуации был взят алгоритм листинг, которого приведен на рисунке 12.

В результате работы алгоритма не было добавлено новых рекомендаций, так как алгоритм не смог выявить совместные данные, которые могут быть распределены одинаково.

```
sub calc_mat(name A, name B, name C, int i, int j, int N)
{
        df Ctmp, Csum;

        for k=0..N-1
        {
                cf f[i][j][k]: mult_mat(A[i][k], B[k][j], Ctmp[k]);// @
                //{
                        //locator_cyclic: k; //add
                //};
        }

        if N>1
                sum_mat(Ctmp[0], Ctmp[1], Csum[1]);
        if N==1
                copy_mat(Ctmp[0], Csum[0]);

        for k=2..N-1
                cf sum[k]: sum_mat(Ctmp[k], Csum[k-1], Csum[k]); //@
                //{
                        //locator_cyclic: k;
                //};

        copy_mat(Csum[N-1], C);
}
sub main()
{
        df A, B, C, N, M;

        init($FG_COUNT, N);
        init($FG_SIZE, M);

        for i=0..N-1
                for j=0..N-1
                {
                        cf initA[i][j]: init_mat(0, i*M, j*M, M, M, A[i][j]) @ {
                                //locator_cyclic: j; // add
                        };
                        cf initB[i][j]: init_mat(1, i*M, j*M, M, M, B[i][j]) @ {
                                //locator_cyclic: j; // add
                        };

                        cf calc[i][j]: calc_mat(A, B, C[i][j], i, j, N);
                }
}
```

Рисунок 12 – листинг алгоритма перемножения матриц

3.4 Итоги тестирования

Тестирование показало работоспособность разработанного алгоритма, которая заключается в появлении новых рекомендаций по распределению, а также показало, что происходит улучшение эффективности исполнения программ, но есть ситуации, с которыми алгоритм не может справится, что дает в свою очередь возможность для последующего улучшения.

ЗАКЛЮЧЕНИЕ

В ходе работы был разработан алгоритм распределения ресурсов на основе статического анализа, анализирующий рекомендации по распределению в исходном коде. Алгоритм был реализован и протестирован в системе LuNA.

Тестирование показало, что алгоритм улучшает нефункциональные свойства конструированных параллельных программ.

Данная работа была представлена на 59-й Международной научно-студенческой конференции, г. Новосибирск, 2021 г. [18]

Защищаемые положения:

1. Разработан алгоритм распределения ресурсов по вычислительным узлам на основе статического анализа
2. реализован алгоритм распределения ресурсов в компиляторе системы LuNA
3. проведено исследование эффективности исполнения конструированных программ с разработанным алгоритмом распределения

Разработка и реализация алгоритма распределения фрагментов данных на основе статического анализа в системе LuNA улучшила эффективность исполнения программ, а также снизила трудозатраты специалистов на написание параллельных алгоритмов математического вычисления.

В дальнейшем планируется развивать алгоритм на основе синтаксического анализа. Возможные направления развития:

1. Реализация версии алгоритма с учетом конфликтных ситуаций
2. Добавление рекомендации для указания класса задач, к которому относится программа
3. Добавление статического анализа циклов на возможность их распределения

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. Optimization Methods of Parallel Execution of Numerical Programs in the LuNA Fragmented Programming System [Электронный ресурс] / – Режим доступа: http://ssd.sscc.ru/sites/default/files/malyshkin_perepelkin2012.pdf (дата обращения: 17.04.2021)
2. Malyshkin, V. E. LuNA Fragmented Programming System, Main Functions and Peculiarities of Run-Time Subsystem / V. E. Malyshkin, V. A. Perepelkin // Proceedings of the 11th Conference on Parallel Computing Technologies. – 2011. – V. 6873. – P. 53–61.
3. Anderson, J. M., Lam, M. S. Global Optimizations for Parallelism and Locality on Scalable Parallel Machines / ACM-SIGPLAN PLDI'93. – 1993. ACM New York, USA. – P. 112–125.
4. Li, J., Chen, M. The Data Alignment Phase in Compiling Programs for Distributed-Memory Machines // J. Parallel and Distributed Computing. – 1991. – V. 13. – P. 213–221.
5. Lee, P. Efficient Algorithms for Data Distribution on Distributed Memory Parallel Computers // J. IEEE Transactions on Parallel and Distributed Systems. – 1997. – V. 8. – P. 825–839.
6. Kraeva, M.A. Assembly technology for parallel realization of numerical models on MIMD multicomputers / M.A. Kraeva, V.E. Malyshkin // Future Generation Computer Systems. – 2001. – V. 17. – P. 755–765.
7. Hu, Y. F., Blake, R. J. An Improved Diffusion Algorithm for Dynamic Load Balancing. J. Parallel Computing. – 1999. – V. 25. – P. 417–444.
8. Corradi, A., Leonardi, L., Zambonelli F. Performance Comparison of Load Balancing Policies Based on a Diffusion Scheme / Euro-Par'97 Parallel Processing. – 1997. – P. 882–886.

9. Barnard, Stephen T. Fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems / Barnard, Stephen T., Simon, Horst D. // Concurrency: Practice and Experience. – 1994. – V. 6. – P. 101–117.
10. Van Driessche Rafael. An improved spectral bisection algorithm and its application to dynamic load balancing / Van Driessche Rafael, Roose Dirk // Parallel Computing. – 1995. – V. 21. – P. 29–48.
11. Yu-Kwong Kwok, Ahmad, I. Design and Evaluation of Data Allocation Algorithms for Distributed Multimedia Database Systems // IEEE Journal on Selected Areas in Communications. – 1997. – V. 14. – P. 1332–1348.
12. Iacob, N. M. Fragmentation and Data Allocation in the Distributed Environments // Annals of the University of Craiova Mathematics and Computer Science Series. – 2011. – V. 38. – P. 76–83.
13. Furtado P., Baumann P. Storage of Multidimensional Arrays Based on Arbitrary Tiling // 15th International Conference on Data Engineering. – 1999. – P. 480–489.
14. Малышкин В.Э., Перепелкин В.А., Щукин Г.А. Распределенный алгоритм управления данными в системе фрагментированного программирования LuNA // Проблемы информатики. 2017. №1 [Электронный ресурс] / – Режим доступа: https://cyberleninka.ru/article/n/raspredelennyy-algoritm-upravleniya-dannymi-v-sisteme-fragmentirovannogo-programmirovaniya-luna (дата обращения: 12.05.2021)
15. Малышкин В.Э., Щукин Г.А. Распределенный алгоритм распределения многомерных сеток данных на многомерном мультикомпьютере в системе фрагментированного программирования LuNA // Проблемы информатики. 2018. №1 [Электронный ресурс] / – Режим доступа: https://cyberleninka.ru/article/n/raspredelennyy-algoritm-raspredeleniya-mnogomernyh-setok-dannyh-na-mnogomernom-multikompyutere-v-sisteme-fragmentirovannogo (дата обращения: 14.04.2021)

16. Малышкин, В. Э. Технология фрагментированного программирования / В. Э. Малышкин // Вестник ЮУрГУ. Серия: Вычислительная математика и информатика. – 2012. – N 46(305). – С. 45 – 55.
17. Межведомственный Суперкомпьютерный Центр Российской Академии Наук [Электронный ресурс] / – Режим доступа: http://www.jscc.ru/ (дата обращения: 12.05.2021)
18. Макаренко, Д. Е. Разработка алгоритма распределения фрагментов данных в системе конструирования LuNA/ Д. Е. Макаренко // Информационные технологии: Материалы 59-й Междунар. науч. студ. конф. 12–23 апреля 2021 г. / Новосиб. гос. ун-т. — Новосибирск: ИПЦ НГУ, 2021. — С. 115.

ПРИЛОЖЕНИЕ А

МОДУЛЬ АВТОМАТИЧЕСКОЙ РАССТАНОВКИ РЕКОМЕНДАЦИЙ ДЛЯ СОВМЕСТНО ИСПОЛЬЗУЕМЫХ ДАННЫХ ДЛЯ КОМПИЛЯТОРА В СИСТЕМЕ «LuNA»

РУКОВОДСТВО ПРОГРАММИСТА

Листов 9

Новосибирск 2021

СОДЕРЖАНИЕ

АННОТАЦИЯ

В данном документе приведено руководство программиста для модуля автоматической расстановки рекомендаций для совместно используемых данных во время компиляции программы в системе LuNA. Исходным языком программы является Python. Средство разработки – редактор исходного кода Visual Studio Code от компании Microsoft.

Основной функцией программы является расстановка новых рекомендаций для совместно используемых фрагментов данных с целью улучшения эффективности исполнения программы.

Оформление программного документа «Руководство оператора» произведено по требованиям ГОСТ 19.504-79 «ЕСПД. Руководство программиста» и ГОСТ 19.105-78 «Единая система программной документации (ЕСПД). Общие требования к программным документам (с Изменением N 1)».

1 Назначение и условия программы

1.1 Назначение программы

Модуль расстановки новых рекомендаций для совместно используемых данных входит в состав компилятора системы LuNA. Пользователь не осуществляет явной работы модулем, а пользуется компилятором системы как обычно.

1.2 Функции, выполняемые программой

Программа позволяет расширить компиляцию языка LuNA с помощью анализа кода, анализа использования фрагментов данных и автоматической генерации кода рекомендаций для распределения нераспределенных фрагментов данных, тем самым упрощая написание программ.

1.3 Условия, необходимые для выполнения программы

Чтобы программа могла исполняться, необходимо наличие у программиста интерпретатора Python версии 3.7 и выше, а также исходного алгоритма с некоторыми начальными рекомендациями по распределению фрагментов данных.

2 Характеристика программы

2.1 Описание основных характеристик программы

Модуль расстановки новых рекомендаций входит в состав компилятора системы LuNA и производит анализ кода компилируемой программы с целью определения возможности использовать начальное распределение, заданное пользователем на нераспределенные фрагменты данных.

2.2 Описание основных особенностей программы

Разработанная версия модуля на момент написания руководства не поддерживает расстановку рекомендаций для данных, которые используются совместно, но имеют разный вид индексных выражений.

3 Обращение к программе

3.1 Описание процедур вызова программы

Анализ на возможность использовать распределение, заданное в исходном коде, для нераспределенных фрагментов данных и модификация абстрактного синтаксического дерева программы выполняются с помощью вызова функции req_for_collectivly_used_data. Для использования функции необходимо импортировать её из файла с исходным кодом программы.

Для чтения абстрактного синтаксического дерева программы используется класс AbstractTreeParser. Во время чтения дерева, вся необходимая информация для анализа потреблений фрагментов данных информация записывается в класс Context.

4 Входные и выходные данные

4.1 Организация используемой входной информации

Входная информация – абстрактное синтаксическое дерево компилируемой программы на языке системы LuNA в формате json.

4.2 Организация используемой выходной информации

Выходной информацией программы является модифицированное абстрактное синтаксическое дерево программы на языке системы LuNA в формате json.

5 Сообщения

В случае, когда программа в системе LuNA компилируется с флагом --verbose, позволяющим компилятору выводить информацию в терминал, если во время анализа программы не получилось распространить начальное распределение будет показано сообщение “Cannot add new recommendations for collectively used data”. Это сообщение не влияет на работу программы, однако для достижения большей эффективности исполнения, программисту предлагается либо обработать незатронутые фрагменты данных вручную, либо переписать программу без использования неподдерживаемых конструкций языка.

6 Лист регистрации изменений

Таблица 3 – Лист регистрации изменении руководства программиста

Лист регистрации изменений									
Номера листов (страниц)					**Всего листов (страниц) в докум.**	**№ докумен-та**	**Входящий № сопроводит. докум. и дата**	**Подп.**	**Дата**
Изм.	**изменен-ных**	**заменен-ных**	**новых**	**аннули-рованных**					