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

Ивакина Александра Олеговича

Тема работы:

РАЗРАБОТКА ЯЗЫКОВЫХ СРЕДСТВ ОПИСАНИЯ НЕФУНКЦИОНАЛЬНЫХ ПАРАМЕТРОВ ПОДПРОГРАММ И ИХ ПОДДЕРЖКА В СИСТЕМЕ LUNA

«К защите допущена»
Заведующий кафедрой,
д.т.н., профессор
Малышкин В.Э. / _____
«31» мая 2023 г.

Руководитель ВКР
к.т.н.,
доцент каф. ПВ ФИТ НГУ
Власенко А.Ю. / _____
«20» мая 2023 г.

Соруководитель ВКР
ст. преп. каф. ПВ ФИТ НГУ
Перепёлкин В.А. / _____
«20» мая 2023 г.

Новосибирск, 2023

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

УТВЕРЖДАЮ
Зав. кафедрой ПВ ФИТ НГУ
Малышкин В.Э.
_____
07 ноября 2022 г.

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ БАКАЛАВРА

Студенту Ивакину Александру Олеговичу, группы 19202.

Тема Разработка языковых средств описания нефункциональных параметров подпрограмм и их поддержка в системе LuNA

утверждена распоряжением проректора по учебной работе от 07.11.2022 № 0330.

Срок сдачи студентом готовой работы: 20 мая 2023 г.

Исходные данные (или цель работы): разработка системы нефункциональных параметров подпрограмм для системы LuNA.

Структурные части работы: обзор и анализ предметной области, постановка задачи, формулирование требований к системе, разработка и реализация системы, тестирование.

Руководитель ВКР

к.т.н.,
доцент каф. ПВ ФИТ НГУ
Власенко А.Ю. / _____
07 ноября 2022 г.

Задание принял к исполнению

Ивакин А.О. / _____
07 ноября 2022 г.

Соруководитель ВКР

ст. преп. каф. ПВ ФИТ НГУ
Перепёлкин В.А. / _____
07 ноября 2022 г.

СОДЕРЖАНИЕ

Введение **5**

1 Анализ предметной области **9**

1.1 Системы параллельного программирования 9

1.2 PaRSEC 9

1.3 Legion 10

1.4 Заключение 11

2 Проектирование системы нефункциональных параметров **12**

2.1 Основные понятия 12

2.1.1 Понятия системы LuNA 12

2.1.2 Общие понятия 13

2.2 Дизайн системы нефункциональных параметров 13

2.2.1 Общая схема-модель системы LuNA 13

2.2.2 Схема передачи параметров в подпрограмму 13

2.2.3 Аннотации в LuNA-программе 14

2.2.4 Требования к нефункциональным параметрам 15

2.2.5 Расширение схемы передачи данных в подпрограмму 15

2.2.6 Расширение логики аннотаций 16

2.2.7 Алгоритм передачи данных 17

2.2.8 Использование целочисленных нефункциональных параметров 18

2.2.9 Анализ расширенной схемы передачи данных в подпрограмму 20

2.3 Специальные параметры и их взаимодействие с системой 20

2.3.1 Специализация нефункциональных параметров 20

2.3.2 Автоматический выбор параметра в адаптивной подпрограмме 21

2.3.3 Алгоритм автоматической подстановки fgsize 22

2.3.4 Использование автоматического выбор параметра 22

3 Реализация системы нефункциональных параметров.....25

3.1 Устройство системы LuNA.....25

3.2 Реализация целочисленных нефункциональных параметров.....25

3.3 Реализация автоматического выбора параметра..... 27

3.3.1 Модификация исполнительной системы..... 27

3.3.2 Модификация компилятора.....27

4 Тестирование.....30

4.1 Проверка целочисленного нефункционального параметра..... 30

4.2 Проверка автоматической подстановки fgsize.....31

4.3 Проверка автоматической подстановки fgsize на умножении матриц..... 32

Заключение.....35

Список использованных источников и литературы..... 37

Приложение А..... 39

Приложение Б.....45

ВВЕДЕНИЕ

С каждым годом в информационных технологиях появляется всё больше систем, позволяющих облегчить и ускорить разработку программного обеспечения. У развития инструментов разработки есть важная цель – экономия сил и времени программиста.

Одним из инструментов разработки являются системы параллельного программирования. Такие системы позволяют работать над производственными и исследовательскими задачами программистам с высокой квалификацией в области производства или исследования, но низкой в области системного параллельного программирования, что также упрощает процесс разработки программного обеспечения в целом.

Разработчики систем автоматического программирования сталкиваются с распространенной проблемой. Эффективная реализация автоматически сгенерированной программы по описанию алгоритма является алгоритмически труднорешаемой.

Эта задача является сложной, так как построить эффективное решение системе трудно. Ввиду этого, качество системы программирования зависит от используемых в ней алгоритмов. Разработчики таких систем пытаются расширить количество классов задач, для которых их система работает эффективно, а также улучшить работу уже готовых решений.

Более того, система программирования может генерировать решение задачи по-разному и от того, как хорошо решение адаптировано под условия исполнения, зависит качество его работы. Условиями исполнения являются, например, различные характеристики вычисляющего устройства или особенности описываемого численного алгоритма. К примеру, если система способна адаптировать свою работу под устройства с различным размером кэш-памяти, то она будет эффективнее использовать предоставленную память.

Тем не менее, системе сложно самостоятельно генерировать адаптивную программу, так как она не знает о способах адаптации ввиду ее незнания

численного алгоритма. В это же время, часто человек, описывающий системе задачу, обладает знаниями и идеями по адаптации и улучшению работы.

Чтобы системе стали доступны знания о способах улучшения программы, человеку необходимо иметь возможность для их передачи. Этого можно достичь, создав для системы интерфейс, через который она сможет влиять на различные параметры. В это время человек должен предоставить интерфейс в виде параметров, в которых он заложит свое решение по оптимизации работы, ведь система все равно не сможет их проанализировать.

Обеспечить такое взаимодействие пользователя и системы – актуальная задача.

В ИВМиМГ СО РАН разрабатывается система фрагментированного программирования LuNA. Она используется для задач научного моделирования на мультикомпьютерах. Она способна самостоятельно обеспечить параллельность выполняемой программы. В ней отсутствуют рычаги управления нефункциональными свойствами программы. Поэтому система LuNA подходит для исследования этой проблемы.

Одним из подходов решения данной проблемы является использование явной параметризации нефункциональных свойств, с помощью которых пользователь и система смогут взаимодействовать друг с другом посредством адаптивных подпрограмм. Иными словами, в системе программирования должна быть предусмотрена система нефункциональных параметров, которые имели бы смысл, известный и пользователю, и системе.

Используя такой интерфейс, пользователь сможет писать адаптивные программы. Если система знакома с представленными пользователем параметрами, то она сможет менять их значения в зависимости от условий исполнения программы, выигрывая в эффективности.

Важно сказать, что подобная система программирования во многих случаях будет применять не лучшие параметры, которое мог бы реализовать специалист с глубокими знаниями в области параллельного

программирования. В частности, быстродействие и использование памяти в решении системы часто будут часто хуже лучшего решения. Но при грамотном написании адаптивной подпрограммы результаты будут близки к хорошему.

Ключевым условием для введения такой возможности в систему программирования является возможность системы анализировать нефункциональные параметры. Это может быть достигнуто, если существует договоренность о смысле параметров. То есть в системе должны иметься возможности для явной параметризации подпрограммы.

Целью работы является разработка системы нефункциональных параметров подпрограмм для системы LuNA.

Для достижения цели работы были поставлены следующие задачи:

- Исследовать возможности работы с нефункциональными свойствами в других системах программирования.
- Разработать набор нефункциональных параметров, определяющих интерфейс взаимодействия с подпрограммой в системе LuNA.
- Реализовать разработанный интерфейс в системе.

Научная новизна работы заключается в том, что в ее результате впервые был получен интерфейс работы с нефункциональными параметрами фрагментированной программы. В частности, этот интерфейс подходит для системы LuNA.

Практическая ценность работы состоит в том, что полученная система нефункциональных параметров является полностью работоспособным модулем, встроенным в систему LuNA. Он позволяет создавать адаптивные подпрограммы и внедрять пользовательские нефункциональные параметры, что ускоряет процесс написания программного кода.

Настоящая работа состоит из введения, четырех глав и заключения. Во введении описываются возможности и проблемы параметризации нефункциональных свойств программ в ручном и системном программировании, актуальность проблемы, цель работы, задачи,

практическая ценность и научная новизна. В первой главе представлен анализ предметной области, составлен обзор систем с точки зрения организации их работы с нефункциональными свойствами. Во второй главе разобраны требования, предъявляемые к системе нефункциональных параметров, предложены и проанализированы решения по проектированию такой системы. В третьей главе описываются детали реализации принятого решения. В четвертой главе приведена информация о тестировании системы нефункциональных параметров.

1 Анализ предметной области

1.1 Системы параллельного программирования

Существует множество различных систем программирования, выполняющих определенные задачи. У них могут быть различные архитектуры и подходы к работе с данными, различные пользовательские интерфейсы. В зависимости от целей, которые преследовали разработчики, опыт работы с той или иной системой может сильно отличаться.

Целью исследования систем параллельного программирования было выделение возможностей управления нефункциональными свойствами выполнения программ. Это необходимо для того, чтобы перед разработкой системы для LuNA почерпнуть опыт разработчиков других систем.

Были исследованы следующие системы: Charm++ [10], PaRSEC [11] и DPLASMA [12], Legion [13] и Regent [14].

В результате анализа удалось выделить две системы, которые в большей степени, чем остальные, схожи с системой LuNA. Они были более подробно проанализированы с точки зрения организации их работы с нефункциональными свойствами.

1.2 PaRSEC

PaRSEC – это система для планирования и управления микрозадачами с учетом архитектуры на распределенных системах. PaRSEC распределяет вычислительные потоки по ядрам и использует динамический, полностью распределенный планировщик, основанный на архитектурных и алгоритмических особенностях, таких как повторное использование данных [4].

Система PaRSEC позволяет пользователю, задавая параметры, описать задачу в виде ориентированного бесконтурного графа, то есть графа, в котором нет циклов, учитывающих направление дуг. В результате пользователь предоставляет системе краткое описание всего потока данных между узлами, которые являются источником таких данных, а также узлами, которые

являются получателем. Таким образом, создается сжатое алгебраическое представление графа задач, который затем преобразуется в код на языке C с помощью компилятора [5].

Описывая задачи, пользователь указывает входные данные, описывает локальные переменные, разбиение данных и их отображение на ресурсы. Все вышеперечисленное является функциональными параметрами. Из настроек нефункциональных свойств задачи имеется приоритет. Приоритет – это целое число, с помощью которого можно указать, в каком порядке система должна выполнять задачи. Таким образом можно, например, задать высокий приоритет объемной задаче, которая будет выполнятся одновременно с более легкой [6].

Система PaRSEC позволяет удобно описывать топологию задач и работать с параметрами. Ее можно использовать как пример с грамотным синтаксисом настройки задач и их взаимодействия. Однако, с точки зрения работы с нефункциональными свойствами, система PaRSEC имеет скромный инструментарий. Несмотря на то, что в ней существует параметр, позволяющий приоритезировать задачи, этого недостаточно, чтобы строить новую систему нефункциональных параметров, основываясь лишь на нем.

1.3 Legion

Legion – это проект с открытым исходным кодом. Разработчики данной системы приоритезируют достижение высокой производительности и энергоэффективности на будущих архитектурах с помощью систем программирования, способных судить о структуре данных программы для эффективного размещения и перемещения данных [7].

Модель программирования Legion ориентирована на данные для написания высокопроизводительных приложений для распределенных архитектур [8].

Legion предоставляет программистам абстракции для явного объявления свойств данных программы, включая организацию, разделение, привилегии и согласованность. В отличие от существующих систем программирования, в

которых эти свойства неявно управляются программистами, Legion делает их явными и предоставляет реализацию программистам.

У системы Legion есть явная сильная сторона – возможность явной параметризации по нефункциональным свойствам реализаций структур данных. То есть в системе имеется возможность вручную настраивать, как именно будут храниться данные в системе. Это может повлиять на ход выполнения программы, но не на ее результат [3].

Система ориентирована на работу с данными, и ее возможности по настройке нефункциональных свойств нацелены на управление именно ими. Эти настройки являются хорошим инструментом в рамках системы Legion, но принцип работы системы LuNA и цели, которые она преследует, отличаются от работы и целей Legion. По этой причине систему нефункциональных параметров из Legion можно использовать лишь как хороший пример из смежной области, но не как опору в разработке.

1.4 Заключение

Рассмотренные системы так или иначе дают пользователю возможность влиять на нефункциональные свойства, но не предоставляют полноценной системы явных нефункциональных параметров, которую можно было бы переложить на систему LuNA, которая работает с фрагментированными программами.

Следовательно, для LuNA необходимо сделать уникальную систему нефункциональных параметров. Некоторые решения, которые применяются в системах, рассмотренных в обзоре, можно использовать в качестве дополнительной опоры при разработке, но получится взять за основу.

2 Проектирование системы нефункциональных параметров

2.1 Основные понятия

2.1.1 Понятия системы LuNA

Фрагмент вычислений – это независимая единица программы, содержит описание входных/выходных переменных и кода фрагмента [2].

Фрагмент данных – это иммутабельная (единственного присваивания) переменная, значением которой может выступать некоторая часть данных прикладного алгоритма [1]. Это данные, которые являются входными/выходными данными фрагментов вычислений.

Подпрограмма – это именованный программный модуль, который может быть многократно вызван из разных частей программы.

Узел – это узел мультикомпьютера, на котором производятся вычисления.

В системе LuNA существует система аннотаций. *Аннотации* – это инструмент, позволяющий дополнить подпрограмму информацией об алгоритме и способе его отображения на ресурсы мультикомпьютера.

Фрагментированную программу можно представить как описание ориентированного бесконтурного графа, в котором каждая вершина является фрагментом вычислений или фрагментом данных. Дуги отражают зависимость фрагментов вычислений от фрагментов данных и наоборот.

Этот граф всегда двудольный, а направление дуг показывает, для какого фрагмента вычислений фрагмент данных является входным, а для какого выходным. На рисунке 1 проиллюстрирован подобный граф. На нем прямоугольники обозначают фрагменты вычислений, а круги – фрагменты данных.

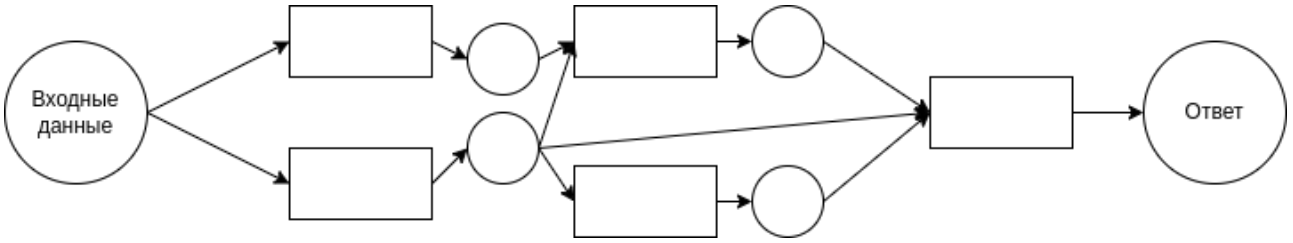

Рисунок 1 – Представление фрагментированной программы в виде графа

2.1.2 Общие понятия

Функциональные параметры – это параметры, влияющие на вычисляемое значение. Это данные, с которыми работает алгоритм, как входные, так и выходные. На основе этих и только этих параметров производится вычисление итогового значения.

Нефункциональные параметры – это параметры, влияющие на детали вычисления. Такие параметры могут влиять на распределение нагрузки, степень фрагментации, способ вычисления. Эти параметры не могут влиять на выходное значение.

2.2 Дизайн системы нефункциональных параметров

2.2.1 Общая схема-модель системы LuNA

Система LuNA состоит из двух модулей, каждый из которых имеет свою зону ответственности в виде отдельного этапа исполнения LuNA-программы. В рамках данной работы были модифицированы LuNA-компилятор (далее – компилятор) и исполнительная система.

Работа компилятора состоит в анализе и преобразовании кода, написанного на языке LuNA.

Исполнительная система начинает свою работу перед запуском готового к исполнению кода, и заканчивает, когда программа полностью отработала и результат был получен.

2.2.2 Схема передачи параметров в подпрограмму

Переиспользуемой единицей кода в LuNA-программе является подпрограмма. Подпрограмму можно многократно вызывать из разных частей кода, передавая в нее параметры.

Стандартная схема передачи данных в подпрограмму предусматривает только параметры, передающиеся в скобках после названия подпрограммы. Например:

Листинг 1 – Вызов подпрограммы

```
cf a: subroutine_name(a, b);
```

Здесь **a** и **b** являются параметрами, с помощью которых данные передаются в подпрограмму. Возвращаемое значение тоже получается через **a** и/или **b**, в них может быть произведена запись результата.

2.2.3 Аннотации в LuNA-программе

В языке LuNA существует специальный инструмент – аннотации. Он используется для того, чтобы влиять на конструирование и исполнение фрагментированных программ. Синтаксис аннотаций выглядит следующим образом:

Листинг 2 – Вызов подпрограммы с аннотацией

```
cf a: subroutine_name(a, b) @ {
    recommendation_name: value;
};
```

Различные аннотации имеют различный смысл, они помогают настроить фрагментированную программу. К аннотациям относятся директивы и рекомендации.

Директивы напрямую влияют на ход исполнения программы. Например, директива **locator_cyclic** позволяет пользователю разместить фрагмент данных на произвольном узле. При этом при неправильном их использовании может быть нарушена работа программы, что может повлечь за собой даже аварийную остановку.

Рекомендации же, как следует из названия, задают условия, которые система не обязана выполнять. Если система распознает, что она может выполнить условия из рекомендации, при этом не нарушив логику работы программы, то она, скорее всего, выполнит эти условия. Рекомендации могут

быть использоваться для описания важных с точки зрения эффективности, но трудно автоматически выставляемых свойств прикладного алгоритма [9].

2.2.4 Требования к нефункциональным параметрам

При формулировании требований к системе нефункциональных параметров были выделены характеристики, которыми она должна обладать.

Нефункциональные параметры должны быть синтаксически выделены. Это необходимо для того, чтобы система разделяла параметры на функциональные и нефункциональные. Кроме того, важно, чтобы пользователь мог легко отличить разные типы параметров при чтении кода.

Система нефункциональных параметров должна быть гибкой и расширяемой. Требуется предусмотреть возможность ее расширения или модификации в целях выполнения различных задач, возникающих при использовании системы LuNA.

Для создания возможности передачи дополнительных параметров должна быть расширена схема передачи параметров в подпрограмму.

2.2.5 Расширение схемы передачи данных в подпрограмму

В исходной схеме передачи параметров передать данные в подпрограмму можно было только через основные параметры, указанные в скобках после имени подпрограммы.

Для системы нефункциональных параметров подпрограмм схема передачи данных была расширена.

Так как нефункциональные параметры должны быть синтаксически отделены от функциональных, то для них было решено использовать механизм, отличный от того, что используется для функциональных параметров. Для передачи нефункциональных параметров было решено модифицировать функциональность аннотаций, которые уже используются в грамматике языка LuNA. Таким образом, параметры различных типов визуально и логически разделяются.

Тип параметра было решено сделать целочисленным. Это позволит пользователям договариваться о смысле параметра и выражать через него необходимую для оптимизации решения информацию.

Для удобства использования было решено сделать нефункциональные параметры именованными, а не позиционными. Таким образом, человек, работающий с ними, будет ассоциировать значения с именами, в которых заложено назначения параметра. Система, в свою очередь, во время обработки параметров сможет ориентироваться на строковые имена, что упростит работу с ними.

2.2.6 Расширение логики аннотаций

Для использования аннотаций в нуждах передачи нефункциональных параметров было необходимо воспользоваться предусмотренной возможностью их расширения и добавить к ним новую аннотацию. После этого, работа новой функциональности была поддержана в системе.

Был введен новый тип аннотаций с названием **nfparam**. С помощью новой аннотации можно передавать целочисленные переменные в подпрограмму. Разработчик, который пишет фрагмент вычислений может использовать эти параметры в своем коде наравне с обычными.

Стандартная схема передачи данных в подпрограмму предусматривает следующий вид вызова подпрограммы:

Листинг 3 – Вызов подпрограммы с локатором

```
cf a: subroutine_name(a, b) @ {
    locator_cyclic x[i] => n;
};
```

Передающиеся данные – **a** и **b**. Аннотация **locator_cyclic** нужна лишь для того, чтобы поместить фрагмент данных **x[i]** на узле **n**.

Обновленная схема передачи данных предусматривает возможность использования новой аннотации для передачи параметра.

Листинг 4 – Вызов подпрограммы с нефункциональным параметром

```
cf a: subroutine_name(a, b) @ {
    recommendation_name: value_1;
    nfparam chunk_size => 20;
};
```

Здесь, в подпрограмму передается размер чанка для вычислений. Он не влияет на итоговый результат, но может повлиять на время исполнения. То есть этот параметр является нефункциональным, поэтому его следует указывать в аннотации.

Теперь, в области видимости внутри подпрограммы будут видны не только **a** и **b**, но и **value_nfparam**. На этом примере наглядно отображено визуальное разделение функциональных параметров (**a**, **b**) и нефункциональных (**value_nfparam**).

Теперь разработчикам осталось лишь прийти к соглашению об именовании нефункциональных параметров и задокументировать их использование для пользователей проекта или библиотеки.

2.2.7 Алгоритм передачи данных

В языке LuNA не был предусмотрен способ передачи целочисленных параметров через аннотации, поэтому при расширении схемы передачи данных в подпрограмму.

Был разработан алгоритм передачи дополнительных данных в подпрограмму. Алгоритм выглядит следующим образом:

Листинг 5 – Алгоритм передачи данных в подпрограмму

```
argument_number = 0          // Номер аргумента
arguments// Список числовых и строковых аргументов подпрограммы
annotations        // Список аннотаций подпрограммы

for argument in arguments:
    if argument.type == integer or real or string:
        assign_argument_number(argument, argument_number)
        argument_number += 1

for annotation in annotations:
    if annotation.type == nfparam:
        assign_argument_number(annotation.value, argument_number)
        argument_number += 1
```

Кроме **arguments**, в котором хранятся числовые и строковые аргументы, существует массив **ids**, который хранит ссылочные типы данных. Здесь обработка **ids** опущена, так как нумерация новых параметров продолжает нумерацию именно не ссылочных данных.

На этапе компиляции одна из функций ставит в соответствие каждому вызову подпрограммы связанный с ним фрагмент вычислений. После этого компилятор пронумеровывает все численные и строчные функциональные параметры и по этим номер сопоставляет значения параметров.

Для передачи нефункциональных параметров это нумерация продолжается и отображается теперь уже на имена нефункциональных параметров. Таким образом, с точки зрения способа передачи данных функциональные и нефункциональные параметры не отличаются.

2.2.8 Использование целочисленных нефункциональных параметров

Нефункциональные параметры должны работать как параметры, переданные через аннотации.

Представим следующий случай использования: пользователь хочет написать фрагмент вычислений, принимающий в аргументе **name x**, и печатающий **x[0]**. Допустим, **x** размещён на узле 1.

Листинг 6 – Размещение на узле в подпрограмме

```
cf a: print_value(x) @ {
        locator_cyclic x[i] => 1;
}
```

Теперь, в фрагменте вычислений необходимо написать такую аннотацию:

Листинг 7 – Размещение на узле в фрагменте вычислений

```
sub print_value(name x) {
        print(x[0]);
} @ {
        locator_cyclic x[i] => 1;
}
```

Таким образом, получился фрагмент, работающий для **x**, который размещен только на первом узле. Но что, если есть необходимость написать универсальный фрагмент, например, печатающий **name**, который размещен на произвольном узле? В таком случае предполагается применение аннотации **nfparam**.

В аннотациях подпрограммы надо передать информацию о номере узла, на котором расположены данные для этой подпрограммы в частности:

Листинг 8 – Передача целого числа в подпрограмму через nfparam

```
cf a: print_value(x) @ {
        nfparam n => 1;
};
```

А в фрагменте вычислений получить это значение и применить к **locator_cyclic**:

Листинг 9 – Получение числа в фрагменте вычислений через nfparam

```
sub print_value(name x) {
        print(x[0]);
} @ {
        locator_cyclic x[i] => n;
        nfparam n;
}
```

Таким образом, получается фрагмент данных, способный работать с данными на произвольном узле. Теперь, код внутри **print_value** лучше параметризован, и его можно модифицировать независимо от значений, связанных с **locator_cyclic**.

2.2.9 Анализ расширенной схемы передачи данных в подпрограмму

Отсутствие возможности использовать типы данных отличных от целых чисел в целочисленном нефункциональном параметре ограничивает его удобство использования.

Жертвуя удобством использования, целочисленный параметр получает преимущество в гибкости, так как такой параметр легко использовать в различных нуждах. Например, целое число можно использовать для обозначения размеров фрагментов памяти, номера узла, временной отметки.

Расширенная схема передачи параметров позволяет пользователю и, что важнее, системе отделять функциональные параметры от нефункциональных. Пользователь получает возможность визуально отделять параметры при написании кода. С точки зрения системы, это нововведение является гораздо более значимым, так как это открывает возможности для системы обрабатывать нефункциональные параметры отдельно.

Также такой параметр является фундаментом для расширения всей системы нефункциональных параметров. Взяв его за пример несложно добавить другой параметр, например, строковый, или параметр-массив. Также на основе целочисленного параметра разработчики LuNA смогут разработать

другие нефункциональные параметры, например, добавив новый синтаксис передачи целых чисел или набора целых чисел.

2.3 Специальные параметры и их взаимодействие с системой

2.3.1 Специализация нефункциональных параметров

Теперь, когда в системе есть нефункциональный параметр, который можно использовать в различных целях, открываются возможности использования подмножества возможных имен параметров для уточнения их специализации.

В рамках этой работы было разработано соглашение, которое заключается в том, что нефункциональный параметр с именем **fgsize** будет обозначать рекомендованный размер фрагмента вычислений для вызываемой подпрограммы.

С одной стороны, такая специализация делает некоторые имена нефункциональных параметров «занятыми». С другой стороны, система получает выигрыш в двух других местах.

Во-первых, такой подход повышает совместимость программных модулей, написанных разными разработчиками. Если тот или иной параметр с фиксированным именем и смыслом используется в разных библиотеках, то для организации совместимости библиотек не придется писать прослойку, которая будет интерпретировать значения из одной библиотеки для другой.

Во-вторых, за счет такого решения параметризация нефункциональных параметров становится частично явной. Если параметр с определенным именем всегда обозначает одно и то же, то система может идентифицировать и анализировать его.

2.3.2 Автоматический выбор параметра в адаптивной подпрограмме

Явная параметризация нефункциональных параметров открывает возможности для реализации адаптивных подпрограмм в системе LuNA. Благодаря этой возможности, был спроектирован и разработан модуль, с помощью которого система может автоматически выбирать параметр.

Вернемся к примеру из предыдущего раздела. Теперь в системе есть нефункциональный параметр с именем **fgsize,** который обозначает размер фрагмента вычислений для задачи из вызываемой подпрограммы. Автоматизация выбора значения была применена к нему следующим образом:

Если при вызове подпрограммы, значение этого параметра было задано, то исполнение происходит как обычно. Но что, если пользователь, вызывающий подпрограмму не уверен, что сможет правильно задать значение, поскольку не обладает достаточными знаниями. В таком случае, он может передать задачу выбора значения системе, не задав его вручную.

Поскольку система «знакома» с этим параметром, то на этапе компиляции она распознает, что параметр не задан, и сможет задать его самостоятельно по какому-либо правилу. В случае с **fgsize**, правилом может служить установка значением размера кэша вычисляющего устройства, так как тогда частота кэш-попаданий увеличится.

Таким образом, в системе LuNA появляется новое соглашение об использовании определенных нефункциональных параметров. Пользователи системы смогут использовать в своих программах эти параметры и рассчитывать, что другие пользователи и сама система поймут их смысл и смогут корректно обработать.

2.3.3 Алгоритм автоматической подстановки fgsize

Для того, чтобы реализовать функциональность автоматической подстановки, необходимо расширить исполнительную систему и компилятор. Разработанный алгоритм, по которому система выполняет подстановку, описан далее.

На этапе компиляции система проверяет, есть ли в LuNA-программе не заданные параметры **fgsize**. И, если такие есть, то отмечает их.

После этого, перед запуском программы исполнительная система узнает размер кэш-памяти текущего вычисляющего устройства и подставляет это

значение. Таким образом, такое поведение программы не отличимо от того, как если бы значение **fgsize** было передано изначально.

2.3.4 Использование автоматического выбор параметра

Допустим, пользователь LuNA в рамках своей работы решает задачу блочного умножения матриц. Ему будет необходимо написать фрагмент вычислений, который будет производить само умножение. В его теле будет необходимо использовать какое-либо значение в качестве размера блока матрицы. Это нефункциональный параметр, поэтому пользователь будет принимать его в фрагмент вычислений с помощью соответствующей функциональности (через механизм аннотаций). Также, этот параметр он назовет **fgsize**, так как в системе уже существует договоренность об использовании именно этого имени для параметра, обозначающего размер фрагмента:

Листинг 10 – Использование fgsize в фрагменте вычислений

```
sub matrix_mult(name matrix_A, name matrix_B) {
        …
        block_size = fgsize;
        …
} @ {
        nfparam fgsize;
}
```

Таким образом, когда возникнет необходимость использовать этот фрагмент вычислений, его можно будет вызвать следующим образом:

Листинг 11 – Использование fgsize при вызове подпрограммы

```
cf a: matrix_mult(matrix_A, matrix_B) @ {
    nfparam fgsize => 1000;
};
```

Здесь пользователь решил, что размер блока матрицы будет равен 1000. Но что, если он не уверен, что это хорошее значение. Например, на другом,

более слабом, вычисляющем устройстве эта величина может оказаться слишком большой и устройство не сможет выделить столько памяти. В таком случае пользователь может не задавать значение параметра и вызвать подпрограмму так:

Листинг 12 – Автоматическое задание fgsize при вызове подпрограммы

```
cf a: matrix_mult(matrix_A, matrix_B);
```

Тогда система сама подставит в **fgsize** значение, равное размеру кэш-памяти устройства.

3 Реализация системы нефункциональных параметров

В данном разделе описана реализация разработанной функциональности. В подразделе 3.1 описаны сущности, используемые в программном коде системы LuNA, которые были использованы в реализации. Подраздел 3.2 описывает детали реализации целочисленных параметров, а подраздел 3.3 описывает детали реализации автоматического выбора специализированного параметра **fgsize**.

3.1 Устройство системы LuNA

Система LuNA состоит из двух основных модулей, это компилятор и исполнительная система.

name – это ссылочный тип выражений, обозначающий идентификатор фрагмента данных. Он используется для того, чтобы обозначать выходные фрагменты данных в фрагментах вычислений.

gja – это абстрактное синтаксическое дерево языка LuNA.

ja – это отдельная ветка синтаксического дерева gja. В ней хранится информация о фрагменте вычислений, который обрабатывается в данный момент. В нем есть следующие поля:

rules – это список аннотаций фрагмента вычислений.

3.2 Реализация целочисленных нефункциональных параметров

Компилятор преобразует логику LuNA программы в генерируемый файл, написанный на C++. С точки зрения генерируемого C++ кода нефункциональные параметры – такие же, как и обычные аргументы. Разница в том, что нефункциональные параметры берутся из аннотаций фрагмента вычислений, а не из списка параметров в скобках.

При создании базовой функциональности нефункциональных параметров был затронут только код компилятора. Были расширены функции, отвечающие за обработку аргументов фрагмента вычислений и генерации структуры, описывающей вызываемую подпрограмму. Также была добавлена

функция, ставящая каждому нефункциональному параметру в соответствие его индекс.

Для корректной обработки нефункциональных параметров на этапе компиляции необходимо сопоставить параметры, передающиеся в вызываемую подпрограмму, и параметры, принимаемые в фрагменте вычислений. Объект **gja** является глобальным и в нем хранятся данные о каждой вызываемой подпрограмме. В объекте подпрограммы хранятся данные о ее аннотациях. Здесь в поле **arg_index** теперь хранятся индексы нефункциональных параметров, которые были получены после нумерации всех аргументов, как это было описано в разделе 2.2.7.

Функция, которая обрабатывает аргументы вызываемого фрагмента вычислений была модифицирована следующим образом: к логике обработки параметров был добавлен обход списка аннотаций (**rules**) фрагмента вычислений. Во время обхода выбираются только аннотации типа **nfparam**. Если такие аннотации есть, то проверяется наличие поля **arg_index** у этой аннотации в глобальном **gja**.

Поле **arg_index** необходимо для того, чтобы передать значение параметра как обычного аргумента внутри сгенерированного C++ файла, но так, чтобы это было скрыто от глаз пользователя LuNA. Нумерация **arg_index** продолжает нумерацию числовых аргументов, но не аргументов типа **name**. Поэтому вычисление **arg_index** происходит после обработки аргументов. Это также соответствует алгоритму, описанному в 2.2.7.

Заранее неизвестно, что будет обработано первым – вызываемая подпрограмма или фрагмент вычислений, но нефункциональный параметр в обоих местах один и тот же. Поэтому перед обработкой такого параметра необходимо проверять, есть ли в **gja** данные об этом параметре или нет. Если данные уже есть, то индекс берется из **gja**. Если же во время обработки параметров оказывается, что у аннотации в **gja** отсутствует поле **arg_index**, то

в специальной функции **assign_nfparams** происходит присваивание ей нового индекса **arg_index** и производится запись в глобальный **gja**.

Функция, отвечающая за генерацию структуры вызываемой подпрограммы, которая в будущем будет выполняться языком C++ была модифицирована: код, связанный с **nfparams**, встроен после обработки аргументов. В этой функции алгоритм также проходит по аннотациям, но уже по аннотациям подпрограммы. Логика алгоритма схожа: индекс нефункционального параметра берется из **gja** или, если ещё не имеется, генерируется с помощью **assign_nfparams**.

Эта структура преобразуется в код C++, в котором функциональные и нефункциональные параметры уже не отличимы.

3.3 Реализация автоматического выбора параметра

Система LuNA автоматически заполняет параметр **fgsize**, используя размер кэша вычисляющего устройства. Автозаполнение параметра происходит на этапе компиляции, а размер кэша вычисляется перед запуском программы. Поэтому для реализации автоматического выбора параметра были модифицированы два основных модуля системы LuNA: компилятор и исполнительная система.

3.3.1 Модификация исполнительной системы

Для определения размера кэша была написана функция **get_cache_size**. Она ищет и обрабатывает файл, в котором хранится информация о вычисляющем устройстве (узле).

Поскольку в файле хранятся многие данные об устройстве, сначала необходимо найти то, что нужно нам. Как только нужная строка найдена, она парсится и сохраняется в двух частях: числовая величина и единица измерения (например «KB» или «MB»). Найденный размер кэш-памяти третьего уровня переводится в байты и возвращается функцией **get_cache_size**.

Эта функция вызывается перед исполнением программы, и значение размера кэша записывается в объект класса **RTS**. Этот класс описывает

текущее вычисляющее устройство (узел), а также предоставляет функции для взаимодействия между узлами. Объект этого класса доступен во всей исполнительной системе, но на каждом узле свой.

3.3.2 Модификация компилятора

Как уже было сказано выше, компилятор преобразует код LuNA программы в генерируемый файл, написанный на C++. Именно на стадии компиляции происходит проверка, соответствует ли каждому «принимаемому» параметру **fgsize** «передающийся» параметр **fgsize**.

Эта проверка происходит в функции, которая генерирует C++ код из данных о фрагменте вычислений. Алгоритм выглядит следующим образом:

Листинг 13 – Алгоритм проверки соответствия нефункциональных параметров

```
annotations_outer        // Список аннотаций, переданных в подпрограмму
annotations_inner        // Список аннотаций, содержащихся в фрагменте вычислений

for annotation_inner in annotations_inner:
    is_pair_found = false
    for annotation_outer in annotations_outer:
        …
        // Здесь остается логика, описанная в разделе 3.2.
        …
        if annotation_inner.name == annotation_outer.name == "fgsize":
            is_pair_found = true
    if not is_pair_found:
        // Запись размера кэша в параметр
```

Алгоритм заключается в проверке каждого передаваемого нефункционального параметра с каждым принимаемым. Когда пара найдена, то значение флага **is_pair_found** становится истинным.

Если после сопоставления флаг **is_pair_found** не стал истинным, то это означает, что для принимаемого параметра **fgsize** нет передаваемого параметра с таким же именем. Тогда в его значение надо подставить размер кэша.

Так как на этапе компиляции нельзя знать этих величин, то никакое значение не подставляется в параметр на данном этапе. Вместо этого, генерируется код на языке C++. Потом, уже перед запуском программы, для получения значения кэша будет получено значение поля из объекта **RTS**. В этом месте связываются модификации в работе компилятора и исполнительной системы.

4 Тестирование

4.1 Проверка целочисленного нефункционального параметра

Для того, чтобы проверить работоспособность функциональности передачи нефункциональных параметров по новой схеме, было разработана следующая наглядная программа:

Листинг 14 – Тестовая программа для целочисленного параметра

```
sub main()
{
    cf print_nfparam_value() @ {
        nfparam n => 10;
  }
}
```

Здесь вызывается подпрограмма **print_nfparam_value**, в которую передано 10 посредством нефункционального параметра **n**. Фрагмент вычислений, соответствующий **print_nfparam_value** описан далее:

Листинг 15 – Тестовый фрагмент вычислений для целочисленного параметра

```
sub print_nfparam_value()
{
    int_print(n);
} @ {
    nfparam n;
}
```

int_print – это функция, печатающее целое число в терминал. Таким образом, от программы ожидается, что она выведет в терминал «10», что и происходит. Таким образом было проверено, что функциональность работает согласно задумке.

При встраивании новой функциональности в систему LuNA, было необходимо дополнить систему Unit-тестами. Это поможет в дальнейшем

внедрять новые модули или модифицировать старые, при этом не конфликтуя с логикой старых модулей.

Представленный выше тест был взят за основу ряда незначительно видоизмененных тестов, которые были использованы как Unit-тесты нефункциональных параметров в системе LuNA.

4.2 Проверка автоматической подстановки fgsize

Для проверки автоматической подстановки параметра **fgsize** было сконструирована схожая программа:

Листинг 16 – Тестовая программа для автоматической подстановки параметра

```
sub main()
{
    cf print_nfparam_value();
}
```

Здесь мы вызываем **print_nfparam_value** без передачи какого-либо нефункционального параметра. Фрагмент вычислений, соответствующий **print_nfparam_value** выглядит так:

Листинг 17 – Тестовый фрагмент вычислений для автоматической подстановки

```
sub print_nfparam_value()
{
    int_print(fgsize);
} @ {
    nfparam fgsize;
}
```

Также нам известно, что на устройстве, на котором запускалась программа, объем кэш-памяти составляет 512 KB. От программы ожидается вывести в терминал «524288», что соответствует объему в байтах. Программа действительно выводит верное значение. При проверках на других вычисляющих устройствах результаты соответствовали их размерам кэш-памяти.

Таким образом было проверено, что функциональность работает исправно и соответствует требованиям.

Представленный выше тест был взят за основу ряда незначительно видоизмененных тестов, которые были использованы как Unit-тесты автоматического выбора нефункциональных параметров в системе LuNA.

4.3 Проверка автоматической подстановки fgsize на умножении матриц

Для того, чтобы проверить работу параметра **fgsize** и оценить потенциальное ускорение, которое он может предоставить, был проведен ряд экспериментов. В качестве примера для проведения замеров была взята задача блочного умножения матриц, размер матриц 1024 на 1024 целых чисел.

В основной подпрограмме, в которой вызываются остальные подпрограммы, выбирается параметр, задающий размер фрагмента вычислений. Этот параметр может быть задан вручную, как это делалось раньше, или, как стало возможно теперь, размер фрагмента можно вычислить, основываясь на размере кэша вычисляющего устройства.

Эксперимент был проведен на двух компьютерах, размер кэша у которых отличается (8 MB и 16 MB). Очевидно, что для каждого устройства и размера задачи можно вручную подобрать оптимальный параметр, при котором задача будет выполняться быстрее всего. Однако, такое значение является оптимальным лишь для одного вычисляющего устройства и его размера кэша.

Заметим, что оптимальное значение параметра связано с размером кэша устройства, тогда возможно проследить связь размера кэша и оптимального размера фрагмента. Таким образом, для задачи выводится приблизительная формула, которая может перевести размер кэша устройства в «хороший» (не сильно отличающийся от оптимального) размер фрагмента. В итоге, возникает возможность написать адаптивную подпрограмму, которая будет выбирать «хороший» размер фрагмента для устройства с произвольным размером кэш-памяти.

Для экспериментальной задачи формула была выведена следующим образом.

Оптимальный размер фрагмента зависит от размера кэша.

$fg_size = F(cache_size)$

Поскольку измерения проводились на двух разных устройствах, то можно построить прямую, аппроксимирующую эту функцию.

$F(8) = 96$

$F(16) = 134$

$F(x) = b * x + c$

В результате получается система уравнений. После ее решения, становится известна формула.

$fg_size = 4,75 * cache_size + 58$

Эти размышления подкрепляются результатами эксперимента, которые представлены в таблице 1:

Таблица 1 – Время исполнения программы при разных размерах фрагмента для матриц 1024 на 1024

	Время, за которое выполняется программа, размер фрагмента которой выбран…		
Размер кэш-памяти вычисляющего устройства	Вручную, как **среднее значение** оптимальных величин двух устройств	Вручную, как **оптимальное** значение для каждого устройства	С помощью выведенной **формулы** и информации о размере кэш-памяти
8 MB	1.7 с	1.4 с	1.5 с
16 MB	1.2 с	0.7 с	0.75 с

Как можно видеть из таблицы 1, результаты, полученные с помощью выведенной формулы, отличаются от подобранных вручную в лучшую сторону и время работы не критически увеличивается в сравнении с оптимальным.

Можно улучшить аппроксимацию, произведя замеры оптимальных размеров фрагмента для матрицы 2048 на 2048. Получив зависимость оптимального размера фрагмента от размера матрицы, можно такой же

аппроксимацией добавить еще один параметр. Формула меняется на следующую:

$$fg_size = (4,75 * cache_size + 58) * (matrix_size/1024) * 1.45$$

В результате применения этой формулы время вычислений почти не изменилось, по сравнению с представленным в таблице 1.

Ниже приведена таблица времени вычислений для более больших матриц:

Таблица 2 – Время исполнения программы при разных размерах фрагмента для матриц 2048 на 2048

	Время, за которое выполняется программа, размер фрагмента которой выбран…		
Размер кэш-памяти вычисляющего устройства	Вручную, как **среднее значение** оптимальных величин двух устройств	Вручную, как **оптимальное** значение для каждого устройства	С помощью выведенной **формулы** информации о **размере кэш-памяти и размере задачи**
8 MB	53.0 с	4.0 с	4.1 с
16 MB	5.75 с	2.6 с	2.8 с

Проанализировав результаты, можно сделать вывод, что в рамках данной задачи и данных размеров метод оправдывает свое использование. Для других задач и условий разработчики могут применять другие аппроксимации.

Этот подход имеет уникальную сильную сторону: в отличие от подобранных вручную значений, которые подбирались с помощью двух вычисляющих устройств, выведенная формула применима к любому размеру кэш-памяти и может быть использована при вычислениях на устройствах с любым размером кэша.

ЗАКЛЮЧЕНИЕ

В результате выполненной работы была спроектирована и реализована система нефункциональных параметров подпрограмм в системе LuNA. Также спроектирована и поддержана в системе функциональность автоматического заполнения специализированного параметра.

Защищаемые положения:

- Расширена схема передачи параметров в подпрограмму.
- Разработан алгоритм передачи дополнительных данных в подпрограмму.
- Разработан алгоритм проверки специализированных нефункциональных параметров и автоматической подстановки значений.
- Разработанные алгоритмы реализованы в составе системы LuNA.

Система нефункциональных параметров подпрограмм улучшила опыт использования системы LuNA и открыла возможности по написанию адаптивных подпрограмм.

Возможные направления развития:

- разработка новых специализированных нефункциональных параметров;
- разработка нефункциональных параметров, содержащих новые типы данных;
- разработка интерфейсов передачи нефункциональных параметров.

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. Ахмед-Заки Д. Ж., Лебедев Д. В., Малышкин В. Э., Перепелкин В. А. Автоматизация конструирования распределенных программ численного моделирования в системе LuNA на примере модельной задачи // Журнал «Проблемы информатики», 2019, № 4, с.53-64. DOI: 10.24411/2073-0667-2019-00017

2. Малышкин В.Э. Технология фрагментированного программирования // Вестник ЮУрГУ. Серия «Вычислительная математика и информатика», № 46(305), 2012, С. 45-55.

3. Documentation. [Электронный ресурс] // Официальный сайт системы Legion – Режим доступа: https://legion.stanford.edu/documentation/ (дата обращения 02.05.2023).

4. PaRSEC. Официальный репозиторий bitbucket [Электронный ресурс]. – Режим доступа: https://bitbucket.org/icldistcomp/parsec/wiki/Home (дата обращения 02.05.2023).

5. Hoque, R., T. Herault, G. Bosilca, J. Dongarra. Dynamic Task Discovery in PaRSEC- A data-flow task-based Runtime // Proceedings of ScalA17. – 2017. – DOI 10.1145/3148226.3148233.

6. Introduction to PaRSEC [Электронный ресурс]. Режим доступа: https://icl.utk.edu/projectsfiles/parsec/pubs/parsec-tutorial-1.pdf (дата обращения 02.05.2023).

7. Legion Programming System [Электронный ресурс] // Legion Programming System: официальный сайт системы. Режим доступа: https://legion.stanford.edu/ (дата обращения 02.05.2023).

8. M. Bauer, S. Treichler, E. Slaughter, A. Aiken. Legion: Expressing locality and independence with logical regions [Электронный ресурс] // Proc. IEEE Int. Conf. High Perform. Comput. Netw. Storage Anal. pp. 1-11 2012. Режим доступа: https://legion.stanford.edu/pdfs/sc2012.pdf (дата обращения 02.05.2023).

9. Перепелкин В. А. Система LuNA автоматического конструирования параллельных программ численного моделирования на мультикомпьютерах // Проблемы информатики. – 2020. – №. 1. – С. 66-74.

10. Kale, Laxmikant V. and Bhatele, Abhinav. Parallel Science and Engineering Applications: The Charm++ Approach / Taylor & Francis Group. — CRC Press, 2013. — ISBN 9781466504127.

11. George Bosilca, Aurélien Bouteiller, Anthony Danalis [и др.]. PaRSEC: A programming paradigm exploiting heterogeneity for enhancing scalability // Computing in Science and Engineering. — 2013. — Т. 15. — С. 36–45

12. George Bosilca, Aurelien Bouteiller, Anthony Danalis [и др.]. Flexible Development of Dense Linear Algebra Algorithms on Massively Parallel Architectures with DPLASMA // IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum. — 2011. — DOI 10.1109/IPDPS.2011.299.

13. Michael Bauer, Sean Treichler, Elliott Slaughter and Alex Aiken. Legion: expressing locality and independence with logical regions // Conference on High Performance Computing Networking, Storage and Analysis, SC'12. — Salt Lake City, UT, USA, November 11 – 15. — 2012. — DOI 10.1109/SC.2012.71.

14. E. Slaughter, W. Lee, S. Treichler, M. Bauer and A. Aiken. Regent: a high-productivity programming language for HPC with logical regions // SC '15: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. — 2015. — С. 1–12. — DOI 10.1145/2807591.2807629.

ПРИЛОЖЕНИЕ А

СИСТЕМА НЕФУНКЦИОНАЛЬНЫХ ПАРАМЕТРОВ ПОДПРОГРАММ ДЛЯ СИСТЕМЫ «LUNA»

Руководство программиста

Листов 6

Новосибирск 2023

СОДЕРЖАНИЕ

Аннотация.....41
1. Назначение программы.....42
1.1 Функциональное назначение программы..... 42
1.2 Эксплуатационное назначение программы..... 42
1.3 Состав функций.....42
2. Условия выполнения программы..... 43
2.1 Минимальный состав аппаратных средств.....43
2.2 Минимальный состав программных средств..... 43
2.3 Требования к персоналу..... 43
3. Выполнение программы..... 44
3.1 Загрузка и запуск программы.....44
3.2 Выполнение программы..... 44
3.3 Завершение работы программы.....44

АННОТАЦИЯ

В данном документе приведено руководство программиста для системы нефункциональных параметров подпрограмм в системе LuNA. Программа написана на двух языках: Python и C++.

Оформление программного документа «Руководство программиста» 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 программы

1.1 Функциональное назначение программы

Система нефункциональных параметров подпрограмм предназначена для создания схемы передачи данных, влияющих на нефункциональные свойства, и расширения возможностей написания адаптивных подпрограмм.

1.2 Эксплуатационное назначение программы

Система нефункциональных параметров подпрограмм предназначена для облегчения написания программ на языке LuNA и/или ускорения их работы. Конечными пользователями системы должны быть физические лица независимо от их рода деятельности.

1.3 Состав функций

- Предоставление инструментов для составления и применения пользовательских нефункциональных параметров.
- Предоставление специализированных нефункциональных параметров с автоматическим выбором значения.
- Предоставление расширенной схемы передачи данных в подпрограмму.

2. Условия выполнения программы

2.1 Минимальный состав аппаратных средств

Программа предусмотрена для использования на персональном компьютере или вычислительном кластере. Устройство, на котором запускается программа, должно иметь:

- Оперативную память объемом 4 Гб или выше.
- Процессор с тактовой частотой 1 ГГц или выше.
- Жесткий диск объемом 32 Гб или выше.
- Монитор.
- Клавиатура.

2.2 Минимальный состав программных средств

Системные программные средства, используемые системой нефункциональных параметров подпрограмм должны быть представлены:

- Операционной системой, поддерживающей интерпретатор языка Python версии 3.8 и выше и компилятор GCC.
- Интерпретатор языка Python версии 3.8 и выше.
- Компилятор языка C++ GCC.

2.3 Требования к персоналу

Конечный пользователь программы (оператор) должен обладать практическими навыками использования интерфейса командной строки и опытом использования системы LuNA.

3. Выполнение программы

3.1 Загрузка и запуск программы

Запуск системы нефункциональных параметров подпрограмм происходит одновременно с запуском системы LuNA, так как является его составной частью.

3.2 Выполнение программы

После запуска система нефункциональных параметров произведет передачу параметров в подпрограммы и подстановку значений в специализированные параметры.

3.3 Завершение работы программы

Завершение программы происходит автоматически после окончания вычислений. Программу можно прервать в любой момент, используя сочетание клавиш Ctrl+C, это вызовет сигнал SIGINT.

ПРИЛОЖЕНИЕ Б

СИСТЕМА НЕФУНКЦИОНАЛЬНЫХ ПАРАМЕТРОВ ПОДПРОГРАММ ДЛЯ СИСТЕМЫ «LuNA»

Описание программы

Листов 12

Новосибирск 2023

СОДЕРЖАНИЕ

Аннотация.....47
1. Общие сведения.....48
1.1 Обозначение и наименование программы.....48
1.2 Программное обеспечение, необходимое для функционирования программы.....48
1.3 Языки программирования.....48
2. Функциональное назначение.....49
2.1 Назначение программы.....49
2.2 Сведения о функциональных ограничениях на применение.....49
3. Описание логической структуры.....50
3.1 Алгоритм программы.....50
3.2 Структура программы.....50
3.3 Связи между составными частями программы.....50
3.4 Связи программы с другими программами.....51
4. Используемые технические средства.....52
5. Вызов и загрузка.....53
6. Входные данные.....54
7. Выходные данные.....55
8. Лист регистрационных изменений.....56

АННОТАЦИЯ

В данном документе приведено описание программы системы нефункциональных параметров подпрограмм в системе LuNA. Программа написана на двух языках: Python и C++. Средство разработки – интегрированная среда разработки Visual Studio Code от компании Microsoft.

Оформление программного документа «Описание программы» произведено по требованиям ГОСТ 19.402-78 «ЕСПД. Описание программы» и ГОСТ 19.105-78 «Единая система программной документации (ЕСПД). Общие требования к программным документам (с Изменением N 1)»

1. Общие сведения

1.1 Обозначение и наименование программы

Полное наименование программы – система нефункциональных параметров подпрограмм для системы LuNA.

1.2 Программное обеспечение, необходимое для функционирования программы

- Интерпретатор языка Python версии 3.8 и выше.
- Компилятор языка C++ GCC.

1.3 Языки программирования

Программа написана на двух языках: Python и C++. Средство разработки – интегрированная среда разработки Visual Studio Code от компании Microsoft.

2. Функциональное назначение

2.1 Назначение программы

Система нефункциональных параметров подпрограмм предназначена для создания схемы передачи данных, влияющих на нефункциональные свойства, и расширения возможностей написания адаптивных подпрограмм.

2.2 Сведения о функциональных ограничениях на применение

Программа не предназначена для запуска на платформах, не поддерживающих Python 3.8.

3. Описание логической структуры

3.1 Алгоритм программы

Программа встроена в два основных модуля системы LuNA: компилятор исполнительную систему.

В компиляторе встроенная программа генерирует код на языке C++ из LuNA-кода. Функциональность, встроенная в исполнительную систему, сохраняет данные о вычисляющем устройстве и производит подстановку значений, если необходимо.

3.2 Структура программы

Программа является расширением двух основных модулей системы LuNA.

Таблица 3 – Модифицированные модули

Модуль	Описание
Компилятор	Модифицированы файлы fcmp и fcmp2. Добавлен новый тип аннотаций. Разработан и встроен алгоритм, позволяющий передавать данные в подпрограмму по расширенной схеме. Добавлена проверка специализированного нефункционального параметра fgsize, которая позволяет системе автоматически заполнять его значение, если оно не указано вручную.
Исполнительная система	Модифицированы файлы rts.cpp, rts.h, cf.cpp и cf.h. Добавлено измерение размера кэш-памяти вычисляющего устройства и сохранение этой величины в специальный объект. Это сделано для того, чтобы во время исполнения было возможно произвести подстановку значение, если необходимо.

3.3 Связи между составными частями программы

Связь между составными частями выполняется следующим образом: компилятор генерирует код, в котором вызываются функции. Эти функции

написаны при расширении исполнительной системы, они будут вызваны в будущем, при исполнении сгенерированного кода.

3.4 Связи программы с другими программами

Для запуска программы необходимы интерпретатор языка Python версии 3.8 и выше, а также компилятор языка C++ GCC.

4. Используемые технические средства

Программа эксплуатируется на персональном компьютере. Работа выполняется в консоли.

Устройство, на котором запускается программа, должно иметь:

- Оперативную память объемом 4 Гб или выше.
- Процессор с тактовой частотой 1 ГГц или выше.
- Жесткий диск объемом 32 Гб или выше.
- Монитор.
- Клавиатура.

5. Вызов и загрузка

Запуск системы нефункциональных параметров подпрограмм происходит одновременно с запуском системы LuNA, так как является его составной частью.

6. Входные данные

Входных данных программа не принимает, так как она занимается лишь обработкой кода и системных файлов.

7. Выходные данные

Выходные данные зависят от задачи, для которой использовалась система нефункциональных параметров подпрограмм. Сама система параметров влияет лишь на нефункциональные свойства.

8. Лист регистрационных изменений

Таблица 4 – Лист регистрационных изменений

Лист регистрационных изменений									
Номера листов (страниц)					Всего листов (страниц) в докум.	№ докум-ента	Входящий № сопроводит. докум. и дата	Подп.	Дата
Изм.	измене-нных	замене-нных	нов-ых	аннулиро-ванных					