МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

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

Факультет информационных технологий

Кафедра параллельных вычислений

Направление подготовки: 09.03.01 ИНФОРМАТИКА И ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА

Образовательная программа: 09.03.01 ИНФОРМАТИКА И ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА

РАЗРАБОТКА И РЕАЛИЗАЦИЯ ПЕРЕНОСИМЫХ АЛГОРИТМОВ РАСПРЕДЕЛЁННОГО ИСПОЛНЕНИЯ ФРАГМЕНТИРОВАННЫХ ПРОГРАММ

утверждена распоряжением проректора по учебной работе №309 от «6» декабря 2016 г.

Ажбаков Артем Альбертович, группа 13201 _____

(подпись студента)

«К защите допущена»

Заведующий кафедрой,

д.т.н., профессор

Малышкин Виктор Эммануилович/……..

(подпись)

«……»……2017г.

Руководитель ВКР

к.т.н., доцент

секретарь кафедры ПВ ФИТ НГУ

Маркова Валентина Петровна/……...

(подпись)

«……»……2017г.

Дата защиты: «……»……20…г.

Новосибирск, 2017г.

МИНОБРНАУКИ РОССИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НОВОСИБИРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
(НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ, НГУ)
Факультет информационных технологий

Кафедра параллельных вычислений

Направление подготовки: 09.03.01 ИНФОРМАТИКА И ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА

УТВЕРЖДАЮ
Зав. кафедрой Малышкин Виктор Эммануилович
…….
(подпись)
«……»……20…г.

ЗАДАНИЕ

НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ БАКАЛАВРА

Студент*у (ке)* Ажбакову Артему Альбертовичу, группы 13201

Тема: Разработка и реализация переносимых алгоритмов распределённого исполнения фрагментированных программ

утверждена распоряжением проректора по учебной работе от 06.12.2016 №309

Срок сдачи студентом готовой работы……20.. г.

Исходные данные (или цель работы): разработка переносимой распределенной среды исполнения, способной задействовать разнородные вычислительные ресурсы, для системы фрагментированного программирования задач численного моделирования LuNA.

Структурные части работы: обзор существующих систем автоматизации параллельного программирования, термины и определения, предлагаемая архитектура экземпляра среды исполнения и алгоритм работы среды исполнения, описание реализации.

Руководитель ВКР
секретарь кафедры ПВ ФИТ НГУ,
к.т.н., доцент
Маркова Валентина Петровна/……..
(подпись)

«…»……20…г.

Задание принял к исполнению
Ажбаков Артем Альбертович./……..
(подпись)

«…»……20…г.

СОДЕРЖАНИЕ

Введение

Параллельное программирование. Существует множество задач численного моделирования, для решения которых мощностей одного компьютера не хватает. Например, задачи распределения тепла в материалах, гидродинамические расчеты, обработка больших данных, моделирование атмосферных процессов, расчет движения космических тел и многие другие. Велика роль численного моделирования в решении научных и прикладных задач. Для моделирования с высокой точностью часто требуется огромное количество памяти и вычислительных операций. Памяти одного компьютера может не хватить для решения задачи, либо расчеты будут неприемлемо долгими. В таких случаях используется параллельный подход, задача разбивается на подзадачи, подзадачи и данные распределяются по множеству вычислительных устройств.

Параллельное программирование - трудоемкий процесс, программист должен учитывать непредсказуемость последовательности срабатывания операторов при распределенном исполнении программы, решать такие проблемы как проблема обеспечения конкурентного доступа к ресурсам, проблемы мертвых блокировок, в отдельных случаях - проблемы отказоустойчивости.

Кроме того, имеется множество тонкостей, которые должен понимать разработчик параллельных программ. Так, например, параллельное исполнение влечет накладные расходы на обмен данными между вычислительными устройствами, что может привести к замедлению исполнения по сравнению с последовательной программой. Потери во времени возникают не только непосредственно в момент передачи данных, но и в периоды ожидания разрешения зависимостей по данным. Производительность системы значительно падает, если распределение вычислений и данных по устройствам организовано неэффективно, без учета скорости коммуникаций, вычислительной мощности каждого узла и его загрузки. Для каждой конфигурации вычислителя требуется тщательный анализ при разработке

параллельной программы вручную. Задача распределения ресурсов становится особенно сложна в современных условиях, когда доступно множество неоднородных ресурсов и для решения одной задачи используются вычислительные устройства разных архитектур, с разной мощностью и объемом памяти. Для написания эффективной параллельной программы программист должен владеть знаниями различных технологий параллельного программирования, понимать особенности конкретного вычислителя, владеть инструментальными средствами. Объем требуемых навыков параллельного программирования зачастую превышает подготовку прикладных специалистов, которым для решения их задач требуется использовать параллельное оборудование.

Для упрощения параллельного программирования создаются системы автоматизации параллельного программирования. Они позволяют описывать алгоритм решения задачи на языке высокого уровня и получать готовую параллельную программу, в значительной степени скрывая от пользователя технические трудности параллельного программирования.

Проблема интеграции разнородных ресурсов в параллельном программировании. Одной из первостепенных проблем в области автоматизации параллельного программирования является проблема совместного использования разнородных вычислительных ресурсов.

Современные вычислительные кластера уже разнородны и тенденция к росту неоднородности вычислительного оборудования сохраняется. Узлы мультикомпьютеров - параллельных вычислительных систем с распределенной памятью - выходят из строя, заменяются или дополняются новыми, более производительными, с другими архитектурами и операционными системами. В то же время, имеется возможность применять одновременно несколько географически удаленных, разных параллельных систем для решения одной задачи.

Использование разнородных ресурсов порождает определенную специфику, которую необходимо учитывать и исследовать, возникает проблема

эффективного использования ресурсов. Особую сложность обретают задачи динамической балансировки нагрузки между узлами мультикомпьютера и эффективной передачи данных между его узлами, “сборка мусора”, динамическое подключение узлов к вычислительной сети и обеспечение отказоустойчивости. Множество исследований посвящается изучению этих вопросов (например, [1-10]). Часто используется практический подход - разрабатываются и испытываются на разных конфигурациях и задачах различные оптимизирующие алгоритмы, выполняются многочисленные тестовые запуски.

Облегчить проведение таких исследований может подходящая для практических экспериментов система исполнения параллельных программ, в которой исследователь мог бы легко модифицировать компоненты, которые отвечают за изучаемые задачи, реализовывать в них свои алгоритмы, варьировать ход исполнения программы, проводить тестовые запуски и получать информацию о ходе исполнения.

Приведенный в данной работе обзор известных систем автоматизации параллельного программирования Charm++, OpenTS, SMP Superscalar/GRID Superscalar/Cell Superscalar показал, что они слабо подходят для использования в качестве такой экспериментальной платформы. Система автоматизации параллельного программирования LuNA в большей степени подходит для этой цели, однако низкая переносимость среды исполнения, используемой в данной системе препятствует этому.

Цель работы. Целью данной работы является разработка переносимой распределенной среды исполнения, способной задействовать разнородные вычислительные ресурсы, для системы фрагментированного программирования задач численного моделирования LuNA.

К разрабатываемому алгоритму работы среды исполнения предъявляются следующие требования:

- Корректное распределенное исполнение LuNA-программ;

- Масштабируемость, т.е. отсутствие принципиальных ограничений для работы системы при любом количестве используемых вычислительных устройств;
- Возможность одновременного использования разнородных вычислительных ресурсов.

К реализации среды исполнения предъявляются требования:

- Модульность, возможность модификации алгоритмов сетевого взаимодействия и динамической балансировки нагрузки между узлами мультикомпьютера;
- Поддержка в качестве узлов мультикомпьютера мобильных устройств, управляемых операционными системами Android и iOS, компьютеров Windows/Linux/Mac;

Для достижения цели выполнены следующие шаги:

1. Выбор технологий, обеспечивающих переносимость и интеграцию разнородных вычислительных ресурсов;
2. Разработка алгоритма работы среды исполнения, реализующего распределенное исполнение фрагментированных LuNA-программ;
3. Реализация разработанных алгоритмов в программной системе с использованием выбранных технологий;
4. Разработка графического интерфейса наблюдения за ходом работы экземпляра среды исполнения;
5. Тестовые запуски.

В результате было получено:

1. Алгоритм работы узла мультикомпьютера, реализующий распределенное исполнение LuNA-программ;
2. Переносимая, не требующая установки, модифицируемая среда исполнения LuNA-программ, запускаемая в окружении JavaScript (веб-браузеры, серверные интерпретаторы JavaScript и др.).

Результаты работы были представлены на 55-ой Международной научной студенческой конференции (МНСК-2017), а также на Конференции Молодых

учёных ИВМиМГ СО РАН 2017 в виде устных докладов. Тезисы докладов опубликованы в сборниках материалов конференций.

1. Обзор существующих систем автоматизации параллельного программирования

Рассмотрим существующие системы автоматизации параллельного программирования с точки зрения пригодности для проведения экспериментальных исследований на разнородных вычислительных ресурсах. В рамках обозначенной цели представляются важными следующие свойства систем параллельного программирования: возможность модификации различных компонентов программной системы для практического применения различных подходов к обеспечению динамических и статических свойств программы, возможность использования неоднородных вычислительных ресурсов.

1.1 Charm++

Объектно-ориентированная модель параллельного программирования Charm++ [1,2] разработана в Лаборатории Параллельного Программирования Иллинойского университета. Система основывается на языке C++. Пользователь пишет программу в объектно-ориентированном стиле, используя специальные объекты - “чары”. Runtime-система распределяет “чары” по узлам мультикомпьютера. Как и обычные объекты в C++ “чары” имеют состояние (значения полей), поведение (методы), но также имеют специальные методы, которые могут быть вызваны удаленно, с любого узла, как обычные методы. Удаленный вызов реализуется средой исполнения через асинхронную посылку сообщения. Среда исполнения также обеспечивает маршрутизацию доставку сообщения между не связанными напрямую узлами, учитывает количество доступных вычислительных узлов, осуществляет динамическую балансировку вычислительной нагрузки по узлам мультикомпьютера.

На базе системы создан набор пакетов для отдельных задач из разных областей суперкомпьютерного моделирования [3].

Детали вычислителя, такие как количество узлов, их характеристики, способ связи (Ethernet, InfiniBand, и.т.д.), скрыты от программиста и не фиксируются в программе.

План выполнения вычислений вариативен. Имеются мощные средства динамической балансировки вычислительной нагрузки между узлами мультикомпьютера, что создает потенциал к использованию неоднородных ресурсов. Однако отсутствует поддержка статического распределения нагрузки между узлами мультикомпьютера. В целом архитектура системы слабо располагает к модификации системных компонентов, реализующих технологические алгоритмы, подлежащие исследованию.

1.2 OpenTS

Система параллельного программирования OpenTS [4-6] - российская разработка Института программных систем им. А.К. Айламазяна РАН, базируется на функциональной парадигме, предоставляет бесконфликтную модель динамического распараллеливания, в которой невозможны взаимные блокировки. Система представляет собой современную реализацию T-системы. Основная идея T-системы - использование так называемых "*неготовых значений*". Вводится понятие T-функции, которая является гранулой параллелизма. Она возвращает неготовое значение, не вызывая блокировку программы. Блокировка произойдет только в момент обращения к этому значению. Когда T-функция завершает свое исполнение, неготовое значение становится готовым и повторяет поведение значения обычного типа. В используемом языке T++ (расширение C++) отсутствуют явные распараллеливающие конструкции, понимаемые в привычном смысле, а T-функции динамически распределяются системой по узлам мультикомпьютера. Используемый "макро-планировщик" способен обеспечивать эффективную загрузку разнородных кластеров.

Для коммуникации используется среда MPI, поддерживаются архитектуры IA-32, x86-64, PowerPC, PowerPC-64, ОС Linux или Windows. Для

разработки необходима установка специального компилятора и библиотек. Предоставляется расширение, позволяющее запускать T-приложения без перекомпоновки с разными реализациями MPI-библиотеки. Тем не менее, использование MPI накладывает ограничение на набор аппаратных платформ.

Система успешно опробована на широком круге задач и на вычислительных установках различного масштаба. Наиболее пригодно использование системы в задачах, где затруднительно статическое распараллеливание, и где вычислительная схема близка к функциональной модели, то есть может быть эффективно представлена с помощью совокупности функций, рекурсивно вызывающих друг друга [4].

T-система освобождает программиста от явной организации параллелизма, обмена данными и синхронизации, от ручного распределения вычислений. Однако необходимо декомпозировать программу в функциональном стиле и тщательно рассчитывать размер гранулы параллелизма, распределяя работу по T-функциям. Одна из целей проекта OpenTS - обеспечение возможности создания новых планировщиков для систем с распределенной и с общей памятью. Возможно подключение различных планировщиков без перекомпиляции, использование разных планировщиков для разных сегментов вычислительной сети. Это, в совокупности с развитым открытым инструментарием для отладки и визуализации и возможностью работы с неоднородными устройствами, делает OpenTS хорошей платформой для исследования динамических свойств, однако отсутствуют способы статического управления распределением вычислительной нагрузки по узлам мультикомпьютера, которое может дать ускорение для большого класса задач.

1.3 SMP Superscalar/GRID Superscalar/Cell Superscalar

Системы автоматизации параллельного программирования для вычислительных систем SMP Superscalar (вычислители с общей памятью), GRID Superscalar (вычислители с распределенной памятью), Cell Superscalar (адаптация для процессоров Cell) [8] интересны используемой математической

моделью. Программа представляется в виде ориентированного бесконтурного графа (Directed Acyclic Graph, DAG), вершины - операции, дуги - информационные зависимости. Программист пишет последовательную программу на языке C, отмечая функции, которые служат крупными элементами параллелизма и распределяются средой исполнения по вычислительным ядрам для параллельного выполнения. Эти указания обозначают потенциальный параллелизм и среда исполнения на их основе динамически составляет граф зависимостей и планирует распределение. Для работы с системой необходим специальный компилятор, а также компиляция и установка самой среды исполнения. Количество процессоров указывается при запуске программы.

Автоматическое формирование графа программы значительно упрощает разработку программы. Управление ресурсами скрыто от программиста, однако обратной стороной является отсутствие возможности статического управления ресурсами. Это не позволяет в полной мере задействовать свойства численных расчетов для оптимизации вычислений.

1.4 LuNA

Система параллельного программирования LuNA [9,10] разработана в лаборатории синтеза параллельных программ ИВМ и МГ СО РАН в рамках проекта “Технология фрагментированного программирования”, который посвящен созданию системы параллельного программирования, ориентированной на реализацию больших численных моделей для суперкомпьютеров. Система включает язык фрагментированного программирования LuNA, компилятор и среду исполнения.

Программа описывается как множество фрагментов вычислений (операций) и фрагментов данных (аргументы операций) и зависимости между ними. Есть языковые средства задания указаний и рекомендаций для статического распределения фрагментов по узлам мультикомпьютера - можно указать какие фрагменты желательно размещать по соседству (в смысле

сетевой топологии). Используя эту информацию компилятор осуществляет частичное статическое распределение фрагментов по узлам мультикомпьютера, что выгодно отличает систему LuNA от аналогов.

Компилятор преобразует программу на языке LuNA в специальную внутреннюю структуру, содержащую описание фрагментов и указания по распределению фрагментов по узлам мультикомпьютера. Эта структура не является императивной последовательностью команд и допускает множественные пути выполнения средой исполнения, что позволяет реализовать в среде исполнения поддержку ряда динамических свойств.

Экземпляры среды исполнения на узлах мультикомпьютера выполняют фрагменты вычислений, осуществляет коммуникации, настраивает исполнение на имеющиеся ресурсы, динамически распределяет нагрузку. Важно отметить, что алгоритм решения прикладной задачи на языке LuNA не содержит реализации самих фрагментов вычислений, он лишь задает отношения между фрагментами вычислений и фрагментами данных. Для исполнения программы, необходимо предоставить последовательные процедуры, реализующие фрагменты вычислений. Среда исполнения вызовет и исполнит этот код в момент выполнения фрагмента.

Существующая реализация среды исполнения выполнена на языке C++ (стандарт языка 2014 года) и использует библиотеку Boost [11]. Реализована поддержка специализированных ускорителей CUDA [12] на узлах мультикомпьютера.

Модель фрагментированного программирования, наличие нескольких этапов обработки и анализа программы, наличие средств динамического и статического распределения нагрузки по узлам мультикомпьютера в сочетании с языковыми возможностями управления распределением делают систему LuNA наиболее подходящей программной платформой для практических исследований из рассмотренных. Переносимость существующей реализации системы ограничена выбранными технологиями, но наличие среды исполнения дает возможность расширить множество поддерживаемых вычислительных

устройств и сделать его более разнородным, разработав новую среду исполнения на основе технологий, обеспечивающих большую переносимость.

1.5 Вывод

Таким образом, существует множество самых разных подходов к автоматизации параллельного программирования, которые эффективны на разных классах задач, задействуют мультикомпьютеры разного масштаба, используют различные математические модели и парадигмы программирования, по-разному выявляют и используют динамические и статические свойства программы.

Большинство представленных систем осуществляют динамическое управление процессом вычислений и в большой степени обеспечивают интеграцию различных по производительности вычислителей. Однако ни одна из систем не предоставляет возможности задействовать широкий класс мобильных устройств и домашних компьютеров без специального программного обеспечения для высокопроизводительных вычислений. Затруднительно проводить эксперименты с одновременным использованием таких устройств и высокопроизводительных кластеров. Большинство систем используют расширения языка программирования C/C++, специальные компиляторы и системы исполнения на узлах мультикомпьютера для использования динамических свойств. Далеко не все решения поддерживают использование графические ускорителей и специализированных процессоров (например, Cell), мобильных устройств.

Поддержка статического управления распределением вычислительной нагрузки практически отсутствует во многих системах автоматизации, что делает их малопригодными для проведения исследований статических свойств программ, а это весьма востребовано для задач численного моделирования.

Существующие системы практически не декларируют возможностей модификации и выбора подходов динамического и статического управления

распределением нагрузки на вычислительные узлы, не представляются удобными для проведения экспериментальных исследований.

Большинство систем используют расширения языка программирования C/C++, специальные компиляторы и среду исполнения для использования динамических свойств. Немногие решения поддерживают использование графические ускорителей и специализированных процессоров (например, Cell), мобильных устройств. Имеется потенциал к увеличению переносимости систем автоматизации.

Среди представленных систем выгодно отличается система LuNA - в ее основе лежит располагающая к использованию разнородных ресурсов концепция фрагментированного программирования, возможно как статическое управление распределением нагрузки по узлам мультикомпьютера, так и динамическая балансировка нагрузки во время исполнения, реализация имеет простую программную архитектуру и допускает модификацию. Препятствует использованию системы в качестве платформы для исследований ограниченная переносимость существующей среды исполнения.

Таким образом, разработка переносимой модифицируемой платформы для практических исследований динамических и статических механизмов управления параллельным исполнением программ в неоднородной среде является актуальной задачей. Наиболее подходящей для этой цели основой выглядит система фрагментированного программирования LuNA.

2. Термины и определения

В разделе представлено понятие фрагментированного программирования, введена формализация выполнения фрагментированной программы, определен функционал узла мультикомпьютера.

2.1 Технология фрагментированного программирования

Система автоматизации параллельного программирования LuNA основывается на технологии фрагментированного программирования, которая является одним из новых подходов к автоматизации параллельного программирования в области численного моделирования [9,10]. Основная идея подхода заключается в отделении алгоритма решения прикладной задачи от управления ресурсами. Алгоритм решения задачи представляется в виде набора фрагментов вычислений - операций - и фрагментов данных - аргументов операций. Фрагментированный алгоритм описывает зависимости между фрагментами, но не определяет их отображения на физические ресурсы. Задача эффективного распределения ресурсов возлагается на систему автоматизации параллельного программирования, а программист лишь описывает алгоритм решения прикладной задачи.

Достоинства подхода: автоматическая генерация параллельной программы, обеспечение ряда динамических свойств программ - динамическая балансировка загрузки узлов мультикомпьютера, обеспечение коммуникаций на фоне вычислений, настройка на аппаратные ресурсы и, как следствие, способность одновременно задействовать неоднородные вычислительные устройства.

Наиболее пригодные для использования фрагментированного программирования задачи характеризуются заданием вычислений одного и того же типа для большого объема данных. Параллелизм в таком случае концептуально прост, но вручную корректно организовать его с точки зрения распределения ресурсов бывает весьма сложно.

Концепция частично реализуется в ряде систем программирования [1-6] и является предметом активного изучения.

2.2. Определение фрагментированного алгоритма и фрагментированной программы

Технология фрагментированного программирования разрабатывалась с целью автоматизации параллельного программирования и использует представление программы, удобное для автоматического управления ресурсами. Прикладной алгоритм представляется в виде, называемом фрагментированным алгоритмом, который загружается на узлы мультикомпьютера и анализируется и выполняется средой исполнения.

Фрагментированный алгоритм - ориентированный граф, вершины которого - *фрагменты вычислений* и *фрагменты данных*. Фрагменты данных - это переменные единственного присваивания, фрагменты вычислений - операции над фрагментами данных. Дугами фрагмент вычислений соединяется с входными и выходными фрагментами. При исполнении фрагмента вычислений в качестве аргументов используются входные фрагменты данных и порождаются выходные фрагменты вычислений и фрагменты данных. Граф фрагментированного алгоритма порождается в ходе исполнения программы, в начальный момент времени множество фрагментов данных пусто, а множество фрагментов вычислений содержит только главный фрагмент (*cfmain*), который является точкой входа в пользовательскую программу, не имеет входных зависимостей и обязательно присутствует в каждой программе на языке LuNA.

Таким образом, фрагментированный алгоритм описывает зависимости между фрагментами, но не определяет их размещения.

Процесс исполнения фрагментированного алгоритма можно формализовать путем описания начального состояния графа, конечного состояния и элементарного шага. Состояние характеризуется следующими множествами:

T - фрагменты вычислений, которые должны быть выполнены;

Dfs - множество сгенерированных фрагментов данных.

Начальное состояние $S_0(T_0, Dfs_0)$: $T_0 = \{cfmain\}$, $Dfs_0 = \{\varnothing\}$, где $cfmain$ - фрагмент вычислений, реализующий функцию main, не имеет входных зависимостей.

Элементарный шаг $S_n(T_n, Dfs_n) \to S_{n+1}(T_{n+1}, Dfs_{n+1})$ - выполнение одного фрагмента вычислений $Cf_i\ (dfin_1, \dots, dfin_{ni}, cfout_1, \dots, cfout_{mi}, dfout_1, \dots, dfout_{ki})$, где $dfin_1, \dots, dfin_{ni} \in Dfs_n$, $cfout_1, \dots, cfout_{ni} \in T_{n+1}$, $dfout_1, \dots, dfout_{ni} \in Dfs_{n+1}$, такое, что

$T_n = Cf_i \cup \{\dots\}$, $Dfs_n = \{\dots\}$ и $T_{n+1} = \{cfout_1, \dots, cfout_{ni}\} \cup T_n \,/\, Cf_i$, $Dfs_{n+1} = \{dfout_1, \dots, dfout_{ni}\} \cup Dfs_n$

То есть, $dfin_1, \dots, dfin_{ni}$ являются входными фрагментами данных для выполнения Cf_i, а фрагменты вычислений $cfout_1, \dots, cfout_{mi}$ и фрагменты данных $dfout_1, \dots, dfout_{ni}$ - выходные фрагменты, порождаемые при выполнении Cf_i.

Конечное состояние $S_z(T_z, Dfs_z)$: $T_z = \{\varnothing\}$, $Dfs_z = \{\dots\}$.

Фрагментированная программа P состоит из множества *фрагментов кода* p: $P = \{p_1, p_2 \ \dots, p_n\}$.

Фрагменты кода имеют параметры и могут быть структурированными или атомарными. *Структурированный* фрагмент кода p содержит операторы op, задающие фрагменты данных и фрагменты вычислений. Пример структурированного фрагмента - процедура, цикл for, цикл while. *Атомарный* фрагмент кода не задает фрагментов, он реализуется некоторой функцией и может означивать фрагменты данных:

$p_{struct} = \{op_1, op_2, \dots\}$, где op_i - фрагмент вычислений cf или фрагмент данных df;

$p_{atom} = externFunc$.

Фрагменты вычислений являются вызовами фрагментов кода, применением фрагмента кода к набору аргументов:

$$cf = \{arg_1, arg_2, \ldots\} \rightarrow p, \text{ где } arg = df \mid const.$$

Фрагмент вычислений также можно называть структурированным или атомарным, в зависимости от того, какому фрагменту кода он соответствует.

Таким образом, фрагментами кода задаются вершины и дуги графа фрагментированного алгоритма. В ходе выполнения фрагментированной программы из фрагментов кода извлекаются и исполняются фрагменты вычислений, выполнение фрагмента вычислений соответствует элементарному шагу исполнения фрагментированного алгоритма.

2.3 Виды фрагментов вычислений в системе LuNA

В системе автоматизации параллельного программирования LuNA выделены следующие виды фрагментов вычислений:

1) **Subroutine** - фрагментированная процедура, имеет параметры. При вызове в качестве аргумента может быть передана константа, выражение, фрагмент данных. Тело процедуры состоит из операторов объявления переменных и операторов вызова фрагментов кода. Важно понимать, что объявление переменной здесь не означает создание фрагмента данных, а лишь является синтаксической конструкцией описания фрагмента данных. Создание фрагмента данных произойдет при означивании переменной в ходе выполнения внешней функции или в результате выполнения цикла while (см. далее фрагмент вычислений while). При выполнении фрагмента вычислений типа subroutine происходит создание фрагментов вычислений, соответствующих элементам тела подпрограммы.
2) **For** - оператор for, характеризуется названием переменной-счетчика, начальным и конечным значением счетчика, телом цикла. При исполнении фрагменты вычислений-элементы тела цикла порождаются одновременно для каждого значения счетчика.

3) **While** - оператор while, характеризуется названием переменной-счетчика, начальным значением счетчика, условным выражением, а также именем фрагмента данных, куда нужно сохранить значение счетчика, когда условное выражение перестанет быть истинным. Если условное выражение истинно, то при исполнении порождаются фрагменты вычислений-элементы тела цикла для текущего значения счетчика, а также копия текущего фрагмента вычислений while с инкрементированным значением счетчика. Значение счетчика - неявный параметр для структурированного фрагмента кода. Если условное выражение ложно, порождается фрагмент данных с указанным именем и значением, равным текущему значению счетчика.
4) **Atom** - внешняя функция, предоставленная пользователем. Аргументы фрагмента вычислений являются аргументами при вызове внешней функции. Такие функции обычно инициализируют переменные, что приводит к созданию фрагмента данных. При выполнении фрагмента вычислений такого типа, среда исполнения вызывает внешнюю функцию и, если присутствовал выходной параметр, порождает фрагмент данных с именем, соответствующим значению этого параметра, и значением, присвоенным в ходе выполнения внешней функции.

2.4 Устройство распределенной среды исполнения

В ходе работы распределенной среды исполнения должно происходить исполнение пользовательской фрагментированной программы. Распределенная среда исполнения это множество вычислительных устройств, на которых работают специальные программы - *экземпляры среды исполнения*. Каждый экземпляр имеет свою локальную память, общей памяти нет. Экземпляры устанавливают соединения между собой для обмена данными, причем граф соединений является связным, но не обязательно полным.

Каждый экземпляр среды исполнения выполняет следующие задачи:

1) **Хранение фрагментов вычислений и фрагментов данных.** Распределенная среда исполнения функционирует как распределенное хранилище данных - в каждом экземпляре хранятся подмножества множества фрагментов вычислений (заданий) *T* и множества фрагментов данных *Dfs*. Экземпляр для хранения каждого фрагмента определяется в соответствии с алгоритмом распределения, реализованном в так называемой функции Pathfinder. Разработка эффективных алгоритмов распределения - тема для большого исследования и не рассматривается в рамках данной работы.
2) **Исполнение фрагментов вычислений.** На каждый экземпляр среды исполнения загружается для выполнения пользовательская программа, представленная в виде внутренней структуры, описывающей фрагментированный алгоритм. Структура получается в результате компиляции программы на языке LuNA и содержит описание фрагментов и отношения между ними, из нее извлекается информация о типе фрагмента вычислений, названия переменных, рекомендации по распределению фрагментов по экземплярам среды исполнения.
3) **Ретрансляция информации.** Поскольку возможна ситуация, когда два экземпляра среды исполнения не имеют прямого соединения друг с другом, необходимо обеспечивать передачу информации между ними посредством других экземпляров.

Такая постановка обеспечивает переносимость и масштабируемость - конфигурация сети не зафиксирована, коммуникации децентрализованы, отсутствуют глобальные структуры данных.

Необходимо уточнить формализацию исполнения алгоритма с учетом параллельности работы экземпляров среды исполнения и непрерывности времени: элементарный шаг считается совершенным в момент сохранения последнего выходного фрагмента в памяти назначенного экземпляра среды исполнения.

Таким образом, было определено корректное исполнение фрагментированной программы и изложен функционал экземпляра распределенной среды исполнения.

3. Предлагаемая архитектура экземпляра среды исполнения и алгоритм работы среды исполнения

В разделе представлена предлагаемая архитектура экземпляра среды исполнения и описан алгоритм работы экземпляра среды исполнения. Детально алгоритм приведен в приложении А.

3.1 Архитектура экземпляра среды исполнения

Для достижения цели обеспечения модифицируемости среды исполнения предлагается выполнить экземпляр среды исполнения в виде набора модулей. В соответствии с функциями экземпляра среды исполнения, описанными в предыдущем разделе, предлагается выделить следующие модули (для каждого модуля приведены также методы, предоставляемые им):

1) Модуль исполнения фрагментов вычислений
    - Принять фрагмент вычислений к исполнению
2) Модуль распределенного хранилища
    - Достать фрагмент из распределенного хранилища
    - Поместить фрагмент в распределенное хранилище
3) Модуль распределения фрагментов
    - Получить идентификатор экземпляра среды исполнения, хранящий фрагмент
4) Модуль сетевого взаимодействия
    - Отправить сообщение
    - Добавить обработчик сообщений

Каждый модуль характеризуется набором внешних методов, которые он предоставляет другим модулям. Посредством этих методов модули взаимодействуют друг с другом.

Алгоритм работы экземпляра среды исполнения представляется в виде совокупности алгоритмов работы взаимодействующих модулей.

Далее для каждого модуля приводится описание алгоритма его работы и предоставляемые этим модулем внешние методы.

3.2 Модуль исполнения фрагментов вычислений

Осуществляет исполнение фрагментов вычислений в соответствии с видом фрагмента (см. пункт 2.3). Для этого именно в данном модуле хранится структура, описывающая фрагментированный алгоритм. При выполнении фрагмента вычислений из нее извлекается необходимая для выполнения фрагмента информация, в том числе информация о фрагментах-зависимостях. Фрагменты-зависимости запрашиваются из распределенного хранилища путем вызова метода "Достать фрагмент из распределенного хранилища", предоставляемого модулем распределенного хранилища. После выполнения фрагмента полученные результирующие фрагменты помещаются в распределенное хранилище вызовом метода "Поместить фрагмент в распределенное хранилище", предоставляемого модулем распределенного хранилища.

Модуль предоставляет метод "Принять фрагмент вычислений к исполнению", в качестве аргумента принимающий фрагмент вычислений. Этот метод должен реализовать выполнение фрагмента вычислений, переданного в качестве аргумента, согласно вышеизложенной логике.

При запуске системы инициирующий экземпляр среды исполнения начинает исполнение фрагмента вычислений $cfmain$.

3.3 Модуль распределенного хранилища

Модуль представляет из себя локальное хранилище фрагментов вычислений и фрагментов данных, которые назначены на данный экземпляр среды исполнения, то есть реализует часть распределенного хранилища. Осуществляет хранение подмножества фрагментов, помещенных в распределенное хранилище, и предоставляет доступ к любому фрагменту, помещенному в хранилище. Скрывает сложность распределенности хранилища от модуля исполнения фрагментов вычислений, позволяет работать с

распределенным хранилищем как с нераспределенным, предоставляя методы "Поместить фрагмент в распределенное хранилище" (аргумент - фрагмент вычислений или фрагмент данных) и "Достать фрагмент из распределенного хранилища" (аргумент - идентификатор фрагмента). При вызове этих методов обращается к модулю распределения фрагментов, используя предоставляемый этим модулем метод "Получить идентификатор экземпляра среды исполнения, хранящий фрагмент", и получает информацию о том, на каком экземпляре среды исполнения должен храниться фрагмент, который необходимо разместить в распределенном хранилище или достать из него. Если назначенный экземпляр совпадает с текущим, то происходит обращение к локальному хранилищу модуля, иначе осуществляется обращение к хранилищу другого экземпляра по сети, путем вызова метода "Отправить сообщение" у модуля сетевого взаимодействия.

При добавлении фрагмента вычислений в локальное хранилище экземпляра среды исполнения модуль распределенного хранилища вызывает метод “Принять фрагмент вычислений к исполнению” модуля исполнения фрагментов вычислений, используя добавляемый фрагмент в качестве аргумента. То есть при назначении фрагмента вычислений для хранения на данном экземпляре среды исполнения начинается его исполнение модулем исполнения фрагментов вычислений.

3.4 Модуль распределения фрагментов

Реализует функцию Pathfinder - логику динамического распределения фрагментов вычислений и фрагментов данных по распределенной среде исполнения. Предоставляет метод "Получить идентификатор экземпляра среды исполнения, хранящий фрагмент", который принимает идентификатор фрагмента и возвращает идентификатор экземпляра среды исполнения, у которого в локальной части распределенного хранилища должен быть размещен данный фрагмент.

3.5 Модуль сетевого взаимодействия

Отвечает за получение сообщений и отправку сообщений другим экземплярам среды исполнения. Имеет каналы связи с некоторым непустым подмножеством множества всех экземпляров среды исполнения (соседи) и осуществляет маршрутизацию сообщения - по адресату сообщения определяет, какому из соседей переслать сообщение, чтобы сообщение достигло адресата, и пересылает его. Выполняет ретрансляцию сообщения, если сообщение предназначается другому экземпляру среды исполнения.

Чтобы другие модули могли осуществлять сетевое взаимодействие, предоставляются методы "Отправить сообщение" и "Добавить обработчик сообщений". Метод "Отправить сообщение" принимает в качестве аргументов само сообщение и экземпляр среды исполнения, на который оно должно быть доставлено, применяет маршрутизацию и передает сообщение. Метод "Добавить обработчик сообщений" принимает тип сообщения и функцию, которая должна быть вызвана при получении сообщения такого типа. Так, например, по умолчанию установлен обработчик для сообщений, содержащих фрагмент вычислений или фрагмент данных, который производит размещение фрагмента в локальном хранилище (модуль распределенного хранилища) данного экземпляра среды исполнения.

3.6 Анализ предложенной архитектуры экземпляра среды исполнения

Предложенная совокупность модулей обеспечивает выполнение функций исполнения фрагментов вычислений, хранения фрагментов в распределенной вычислительной сети и ретрансляции (по построению).

Модули могут иметь разную реализацию. Так, например, в зависимости от реализации модуль исполнения фрагментов вычислений может выполнять принятые к исполнению фрагменты последовательно или параллельно. Модуль распределенного хранилища может использовать разные структуры данных для хранения фрагментов. Модуль распределения может реализовать различные

алгоритмы динамической балансировки. Модуль сетевого взаимодействия может использовать различные сетевые топологии и технологии передачи данных, а также реализовывать различные алгоритмы сетевого взаимодействия.

Таким образом, обеспечивается модифицируемость распределенной среды исполнения.

Следует добавить, что логика использования функции Pathfinder (модуль распределения фрагментов) в предлагаемой архитектуре отличается от действующей в исходной реализации среды исполнения на языке C++. Исходная логика использования подразумевает, что функция Pathfinder указывает какому из экземпляров-соседей передать фрагмент. В предлагаемой архитектуре она может указать любой экземпляр среды исполнения, не обязательно соседний, а путь до него будет определен в ходе маршрутизации модулем сетевого взаимодействия. Это было сделано из соображений модульности, чтобы перенести ответственность за работу с топологией сети на модуль сетевого взаимодействия, а также для упрощения реализации функции Pathfinder. При необходимости модуль распределения фрагментов и модуль сетевого взаимодействия могут быть изменены для соответствия исходной логике использования функции Pathfinder.

3.7 Анализ представленного алгоритма работы экземпляра среды исполнения

Оценим алгоритм работы экземпляра среды исполнения с точки зрения предъявленных требований.

Распределенное хранилище, представляющее собой совокупность локальных хранилищ (модулей распределенного хранилища) экземпляров среды исполнения, реализует множества *T* и *Dfs* (см. пункт 2.2). В начальный момент времени оно содержит только фрагмент вычислений $cfmain$, что соответствует начальному состоянию фрагментированного алгоритма (графа). Выполнение одного фрагмента вычислений модулем исполнения фрагментов вычислений включает извлечение фрагмента вычислений из хранилища,

выполнение всех операторов фрагмента вычислений и пополнение распределенного хранилища продуктами исполнения, что соответствует выполнению элементарного шага исполнения фрагментированного алгоритма. Конечное состояние фрагментированного алгоритма (графа) наступает после выполнения всех фрагментов вычислений и всегда достигается, если пользовательский алгоритм, реализуемый исполняемой программой, завершается за конечное число операций. Таким образом, алгоритм осуществляет исполнение фрагментированной программы согласно определению, то есть происходит исполнение фрагментированного алгоритма. Выполняется требование корректности распределенного исполнения LuNA программ.

Отсутствие центрального сервера для передачи сообщений и возможность ретрансляции сообщений позволяет экспериментировать с топологией сети и применять различные техники горизонтального масштабирования. Отсутствуют глобальные структуры данных, коммуникации децентрализованы. Таким образом, алгоритм удовлетворяет требованию масштабируемости.

Исполняемая внутренняя структура может содержать рекомендации и указания по распределению ресурсов, но их несоблюдение не делает невозможным исполнение программы. Описание фрагментированного алгоритма не фиксирует конфигурацию вычислительной сети, его исполнение в ходе работы среды исполнения также не накладывает принципиальных ограничений на вычислитель. Сохранение производительности при смене конфигурации обеспечивается стратегией распределения фрагментов вычислений и фрагментов данных по вычислительным устройствам, алгоритм работы среды исполнения позволяет применять разные алгоритмы распределения. Исходя из этого, принципиальных ограничений переносимости среды исполнения не заложено и требование переносимости выполняется.

Таким образом, представленный алгоритм работы распределенной среды исполнения фрагментированных LuNA-программ соответствует предъявляемым требованиям.

4. Описание реализации

В разделе дано краткое описание выбранных для реализации технологий и обоснование выбора. Приведены детали реализации модулей, возможные улучшения, описание и результаты тестовых запусков.

4.1 Web-технологии

Для реализации переносимой среды исполнения необходимо выбрать технологии, которые удовлетворяют требованиям поддержки разнородных устройств и простоты развертывания. Оценим с этой позиции web-технологии.

Современные веб-технологии решают масштабные задачи организации одновременного доступа к информации множества пользователей, динамично развиваются и оказывают большое влияние на развитие информационных технологий. Веб-технологии изначально разрабатывались для создания распределенных информационных систем, обеспечивают платформонезависимое взаимодействие, им присуща переносимость и масштабируемость. Значительные вычислительные ресурсы вовлечены как на серверной, так и на клиентской стороне.

Одним из основных конечных звеньев глобальной разнородной распределенной сети Интернет является веб-браузер. Сохраняется тенденция обеспечения функцией просмотра веб-страниц самых разных устройств, практически каждая операционная система включает веб-браузер. Таким образом, имеется огромная база разнородных вычислительных устройств, объединенных единой коммуникационной сетью, которые потенциально могут быть задействованы в высокопроизводительных вычислениях.

4.1.1 Javascript

Возможности и производительность на клиентской стороне возрастают, в простом веб-браузере уже можно обрабатывать и передавать аудио- и видеопотоки, просматривать и редактировать документы, играть в трехмерные игры, выполнять математические расчеты, программировать. Браузеры сейчас

являются не только средством визуализации, но и средой исполнения программ. Основным средством программирования в браузерной среде является язык программирования JavaScript [13]. Он также широко применяется в области высокопроизводительного серверного ПО, что свидетельствует о потенциале использования этой технологии для организации разнородной вычислительной сети, включающей как простые мобильные устройства, так и мощное серверное оборудование.

Перспективность web-технологий для высокопроизводительных вычислений подтверждается заинтересованностью крупных компаний в их использовании для этой цели. Так, например, компания Cisco анонсировала проект “Supercomputer in your browser” [14], позволяющий использовать браузер в качестве элемента вычислительной сети. Для этого разрабатывается JavaScript-версия интерфейса MPI. Существуют и другие проекты, посвященные высокопроизводительным вычислениям и использующие JavaScript и браузерное окружение [15-16]. Компании Google, Mozilla и Microsoft участвуют в разработке технологии WebAssembly [17], которая является аналогом языка ассемблер для JavaScript и обещает значительно повысить производительность JavaScript. Существуют также средства доступа к графическому ускорителю из среды JavaScript, а также средства для работы с SIMD-вычислениями [18,19].

Использование JavaScript для разработки среды исполнения позволит запускать ее как в простом веб-браузере, так и на производительном серверном и кластерном оборудовании, и приводит к удовлетворению требований поддержки разнородных вычислительных устройств, а также обеспечивает простоту развертывания системы.

4.1.2 WebRTC

Коммуникации в среде web строятся по принципу “клиент-сервер”. Такая организация неприемлема для масштабируемой децентрализованной

вычислительной системы. Нужны средства прямой связи между узлами вычислительной сети.

WebRTC [20] - развивающийся проект с открытым исходным кодом, стандартизованный W3C и IETF, предназначенный для потоковой передачи данных между веб-браузерами по технологии точка-точка (peer-to-peer). Технология используется, например, в приложениях для видеочата в браузере. Предоставляется программный интерфейс с набором функций для вызова из JavaScript для установления прямого соединения между браузерами. WebRTC поддерживается не только в браузерах, но и на серверной стороне.

WebRTC подходит для обеспечения связи между экземплярами разрабатываемой среды исполнения.

Таким образом, web-технологии, в частности, язык программирования JavaScript могут объединить разнородные ресурсы, обеспечить высокую переносимость, простоту развертывания и производительность и подходят для создания на их основе среды исполнения для системы автоматизации параллельного программирования.

Была разработана программа на языке JavaScript, реализующая представленные алгоритмы и служащая средой исполнения для системы LuNA. Коммуникации между экземплярами среды исполнения реализованы с использованием технологии WebRTC.

4.2 Детали реализации

Сигнальный сервер. Установление прямого соединения между экземплярами среды исполнения с помощью WebRTC требует наличия дополнительного канала связи для передачи сервисной информации. Для этого с использованием технологии NodeJS [21] в ходе данной работы был разработан специальный сигнальный сервер. Для присоединения к вычислительной сети экземпляр подключается к этому серверу и получает от него идентификатор и “контактные данные” экземпляров-соседей, формирует ответные данные на основе полученных и отсылает их на сервер. Сервер

переадресует их соседям и прямая связь между этими экземплярами будет установлена. Сигнальный сервер является элементом централизации, но функционирует лишь на этапе инициализации сети и не снижает производительность в ходе выполнения программы. Для децентрализации возможно также использование сети сигнальных серверов.

Способы запуска. Наиболее распространенные среды исполнения JavaScript, пригодные для запуска реализации - Google Chrome (начиная с 28 версии), Mozilla Firefox (начиная с 22 версии), Microsoft Edge (начиная с 12 версии), Opera (начиная с 18 версии). Частично поддерживается NodeJS - может выполняться одиночный экземпляр среды исполнения, но осуществление сетевого взаимодействия на данный момент возможно лишь при использовании экспериментальной версии WebRTC с ограниченным функционалом, т.к. технология находится в разработке.

Графический интерфейс. Для использования среды исполнения в браузере был разработан HTML-интерфейс, предоставляющий возможность в наблюдать за функционированием данного экземпляра среды исполнения.

Доступна следующая информация:

- Хранимые в экземпляре среды исполнения фрагменты вычислений и фрагменты данных
- Выполняемые на экземпляре среды исполнения фрагменты вычислений
- Значения фрагментов данных
- Состояние подключения к соседним экземплярам и к сигнальному серверу
- Лог сетевого взаимодействия

Для удобства отладки реализован режим пошагового исполнения, когда очередной фрагмент вычислений принимается к исполнению только по указанию пользователя.

Особенности реализации модулей.

1) Модуль исполнения фрагментов вычислений - согласно алгоритму работы среды исполнения возможно одновременное выполнение нескольких фрагментов вычислений, назначенных на экземпляр (параллелизм на уровне экземпляра среды исполнения), но в текущей версии реализации среды исполнения он не задействован. Однако применены средства JavaScript для повышения скорости выполнения сетевых запросов за счет асинхронности - JavaScript Promise. Этот интерфейс позволяет отправлять несколько сетевых запросов, не дожидаясь ответа, и обрабатывать ответы в произвольном порядке. Последовательный интерпретатор JavaScript при этом переключается от одного запроса к другому.
2) Модуль сетевого взаимодействия - в текущей версии реализации модуль сетевого взаимодействия использует логическую организацию экземпляров “линейка”, когда экземпляры среды исполнения связаны последовательно в цепочку. Может быть изменено в другой реализации модуля сетевого взаимодействия и изменениями в сигнальном сервере.
3) Модуль распределения фрагментов - функция Pathfinder в текущей версии реализована как значение хеш-функции от идентификатора фрагмента. Может быть изменено в другой реализации модуля распределения фрагментов.

4.3 Возможные улучшения

Разработанная реализация среды исполнения представляет из себя базу для дальнейшего развития. Готова основа, обеспечивающая исполнение фрагментированных программ с использованием заменяемых модулей. Модули на данный момент имеют простейшую реализацию и предназначены для многократной замены в ходе применения системы в соответствии с целью разработки - в ходе исследования различных особенностей исполнения параллельных программ в неоднородной среде.

В качестве возможных будущих улучшений реализации среды исполнения следует указать обеспечение многопоточности средствами JavaScript и отделение системы отладки и журналирования от непосредственного выполнения вычислений. Следует отметить также, что текущая версия WebRTC имеет ограничение на максимальный размер передаваемого за одну передачу пакета данных. Это значит, что для пересылки больших пакетов, в частности, больших фрагментов данных, необходимо реализовать фрагментирование пакета и передачу по частям. Это может быть достигнуто модификацией модуля сетевого взаимодействия.

4.4 Тестовый запуск

Поскольку разрабатываемая система задумана как платформа для исследования особенностей выполнения параллельных программ в неоднородной среде, требуется проведение тестовых запусков для проверки интеграции разнородных вычислительных устройств. Такой тест был проведен. Кроме того, были выполнены также тесты производительности, чтобы увидеть как будет зависеть время выполнения программы от числа задействованных экземпляров среды исполнения для оценки возможного будущего практического применения для численных расчетов.

Тестовая программа - решение уравнения Пуассона методом Якоби в трехмерной области. Размер области - 10x10x10 блоков, количество фрагментов - 10, значение $\varepsilon = 0.001$. Значения аргументов выбраны так, чтобы не превышать ограничения размера сообщения при передаче между экземплярами среды исполнения, временно присутствующие в текущей реализации коммуникационного модуля (см. пункт 4.3).

Выбор тестовой программы объясняется тем, что данная численная задача может быть легко разделена на вычислительные фрагменты произвольного размера. Выбором разбиения удобно регулировать как вычислительную нагрузку на экземпляры среды исполнения, так и нагрузку на передачу данных между экземплярами, что позволяет обеспечить приемлемый

размер передаваемых по сети данных и достаточно продолжительное время вычислений, чтобы минимизировать влияние побочных процессов (например, временные затраты на инициализацию). Исходя из этих соображений подбирались значения аргументов. Кроме этого, при выборе значений аргументов учитывалось также то, что максимальный размер сообщения, передаваемого по сети, ограничен в текущей реализации коммуникационного модуля (см. пункт 4.3).

Как было сказано в пункте 2.3, пользователь должен предоставить реализацию внешних фрагментов вычислений. Для тестовой программы была взята существующая реализация внешних фрагментов вычислений для исходной реализации среды исполнения на языке Си++ и переписана на языке JavaScript.

Проверка интеграции разнородных вычислительных устройств. Тестовая программа была запущена в распределенной среде исполнения, включающей следующие устройства (по убыванию приблизительно оцениваемой производительности):

1) Компьютер с операционной системой Windows 7 и браузером Google Chrome (процессор Intel Core i5, 8 Гб оперативной памяти);
2) Ноутбук с ОС Ubuntu 16.04 и браузером Mozilla Firefox (процессор Intel Core i3, 6 Гб оперативной памяти);
3) Планшетный компьютер с ОС Android 4.4 и мобильным браузером Google Chrome (процессор Qualcomm Snapdragon, 2 Гб оперативной памяти);
4) Планшетный компьютер с ОС Android 4.2 и мобильным браузером Google Chrome (процессор Intel Atom, 2 Гб оперативной памяти).

Программа была успешно исполнена, результат работы программы - количество итераций для схождения алгоритма (53 итерации) и значение максимального расхождения (0.00988...) - совпадает с эталонным, полученным при запуске этой же программы в исходной среде исполнения на языке C++.

Таким образом, программа корректно выполнилась и достигается основная цель - интеграция разнородных вычислительных устройств.

Тестирование производительности. Тестирование производилось на домашнем компьютере с 4-х ядерным процессором Intel Core i5 и 8 Гб оперативной памяти. Запуски проводились с выделением разного числа процессов. Оценивалось время выполнения программы, а также распределение фрагментов по процессам. Следует отметить, что в разработанном прототипе системы исполнения вычислительная логика не в полной мере отделена от инструментов отладки и визуализации, что существенно снижает производительность, но представляется легко устранимым в будущих версиях. На данном этапе цель достижения высокой производительности не ставилась. Результаты представлены в таблице 1.

Таблица 1 — Результаты тестирования производительности.

Количество процессов	Время выполнения программы (сек.)	Количество фрагментов данных, загруженных на узлы (*номер узла*: *количество*)	Количество фрагментов вычислений, назначенных на узел (*номер узла*: *количество*)
1	19	1: 5343	1: 3262
2	33	1: 3988 2: 4032	1: 1548 2: 1714
3	41	1: 3011 2: 3007 3: 2947	1: 1120 2: 1109 3: 1033
4	49	1: 2376 2: 2401 3: 2269 4: 2295	1: 812 2: 905 3: 735 4: 810
8	108	1: 1334 2: 1289 3: 1146 4: 1234	1: 418 2: 489 3: 315 4: 492

		5: 1233 6: 1139 7: 1227 8: 1334	5: 407 6: 317 7: 407 8: 417

Падение производительности с увеличением числа узлов объясняется ростом затрат на передачу данных из-за несогласованности распределения вычислений и данных, ввиду использования хеш-функции в качестве функции Pathfinder. Не соблюдается принцип локальности, и фрагменты данных, содержащие большие массивы, закачиваются на новые экземпляры среды исполнения, вместо выполнения соответствующих вычислений на тех экземплярах, где уже лежат данные. Такое же поведение наблюдается в исходной среде исполнения на языке C++. Задача может быть решена с помощью новой реализации функции Pathfinder и является наглядным примером проблемы, экспериментальное исследование которой может проводиться с использованием разработанной платформы.

Таким образом, тест интеграции разнородных устройств подтвердил соответствие выбранных технологий цели работы, а также продемонстрировал работоспособность реализации среды исполнения. Тест производительности выявил ожидаемую зависимость времени работы от числа экземпляров среды исполнения и распределение фрагментов по экземплярам среды исполнения, что свидетельствует в пользу правильности работы реализации среды исполнения на данной задаче.

Заключение

Был разработан алгоритм исполнения программ в неоднородной распределенной среде для системы автоматизации параллельного программирования LuNA. Разработан и реализован алгоритм работы среды распределенного исполнения LuNA программ, предложена архитектура среды исполнения. Среда исполнения:

- Позволяет задействовать разнородные вычислители
- Является модифицируемой, может служить платформой для практического исследования особенностей исполнения параллельных программ в неоднородной среде

На защиту выносится:

1. Алгоритм работы среды распределенного исполнения LuNA-программ;
2. Архитектура среды распределенного исполнения LuNA-программ;
3. Реализация среды исполнения в программной системе на языке JavaScript;
4. Проведенное тестирование функционирования на разнородных вычислительных устройствах и тестирование производительности.

Дальнейшее развитие работы включает усовершенствование реализации среды исполнения и применение системы для исследования на практике специфики исполнения параллельных программ в неоднородной среде (разработка и тестирование модулей динамической балансировки, реализация различных подходов к организации связи между узлами вычислительной сети, и др.).

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».

Ажбаков Артем Альбертович _____

ФИО студента Подпись студента

« ____ »_____20 __г.

(заполняется от руки)

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ И ЛИТЕРАТУРЫ

1. Laxmikant V. Kale, Sanjeev Krishnan, CHARM++: A portable concurrent object oriented system based on C++, University of Illinois at Urbana-Champaign, Champaign, IL, 1993
2. Charm++. Parallel Computer Network. URL: http://charmplusplus.org/ (дата обращения: 10.11.2016)
3. Charm++. Applications. URL: http://charmplusplus.org/applications/ (дата обращения: 10.11.2016)
4. Т-система с открытой архитектурой. Краткое введение для пользователей. URL: http://www.botik.ru/~abram/temp-T-system/t-system-intro.u.html (дата обращения 10.11.2016)
5. Т-система с открытой архитектурой. Адаптивная модель обеспечения отказоустойчивости с оптимизацией по используемым ресурсам / В. А. Роганов, А. А. Кузнецов, Г. А. Матвеев, В. И. Осипов // Программные продукты, системы и алгоритмы. — 2014. — № 1.
6. Система параллельного программирования OpenTS. URL: http://www.opents.net/index.php/ru (дата обращения: 10.11.2016)
7. Opents: An outline of dynamic parallelization approach / S. Abramov, A. Adamovich, A. Inyukhin et al. // Berlin etc. Springer, 2005. - Lecture Notes in Computer Science. — Vol. 3606. — 2005. — P. 303–312.
8. Badia, R.M., Labarta, J., Sirvent, R. et al. Journal of Grid Computing (2003) 1: 151. doi:10.1023/B:GRID.0000024072.93701.f3
9. V. Malyshkin, V. Perepelkin. Optimization methods of parallel execution of numerical programs in the LuNA fragmented programming system // The Journal of Supercomputing, Springer, Volume 61, Number 1 (2012), pp. 235-248.
10. V.E. Malyshkin, V.A. Perepelkin. LuNA Fragmented Programming System, Main Functions and Peculiarities of Run-Time Subsystem //

Proceedings of the 11th International Conference on Parallel Computing Technologies, LNCS 6873. - pp. 53-61, Springer, 2011.

11. Boost. C++ libraries. URL: http://www.boost.org/ (дата обращения 10.11.2016)
12. NVIDIA. CUDA. URL: http://www.nvidia.ru/object/cuda-parallel-computing-ru.html (дата обращения 10.11.2016)
13. W3Techs. Usage of client-side programming languages for website. Дата обновления 01.05.2017. URL: https://w3techs.com/technologies/overview/client_side_language/all (дата обращения 12.05.2017)
14. Cisco Blogs. High Performance Computing. A supercomputer in your browser. Дата обновления 01.04.2015. URL: https://blogs.cisco.com/performance/a-supercomputer-in-your-browser (дата обращения 10.11.2016)
15. BMC Bioinformatics. QMachine: commodity supercomputing in web browsers. Дата обновления 09.06.2014. URL: https://bmcbioinformatics.biomedcentral.com/articles/10.1186/1471-2105-15-176 (дата обращения 10.11.2016)
16. Skale-engine. URL: https://skale-me.github.io/skale-engine/ (дата обращения 10.11.2016)
17. WebAssembly. URL: http://webassembly.org/ (дата обращения 10.11.2016)
18. GPU.JS. GPU Accelerated JavaScript. URL: http://gpu.rocks/ (дата обращения 10.11.2016)
19. Turbo.js. URL: https://turbo.github.io/ (дата обращения 10.11.2016)
20. WebRTC. URL: https://webrtc.org/ (дата обращения 03.09.2016)
21. Node.js. URL: https://nodejs.org/en/ (дата обращения 10.11.2016)

ПРИЛОЖЕНИЕ А

Алгоритмы работы узла вычислительной сети

1. Алгоритм исполнения фрагмента вычислений

1.1. Определение типа фрагмента вычислений

1.2. Обработка фрагмента вычислений:

1.2.1. Тип фрагмента вычислений - subroutine:

1.2.1.1. Для каждого элемента тела:

1.2.1.1.1. Обработка элемента тела (алгоритм обработки элемента тела)

1.2.2. Тип фрагмента вычислений - for:

1.2.2.1. Получить начальное значение счетчика (алгоритм получения значения выражения)

1.2.2.2. Получить конечное значение счетчика (алгоритм получения значения выражения)

1.2.2.3. Для каждого значения счетчика от начального до конечного значения:

1.2.2.3.1. Для каждого элемента тела:

1.2.2.3.1.1. Обработка элемента тела (алгоритм обработки элемента тела)

1.2.3. Тип фрагмента вычислений - while:

1.2.3.1. Получить значение счетчика (алгоритм получения значения выражения)

1.2.3.2. Получить значение условного выражения (алгоритм получения значения выражения):

1.2.3.2.1. Если условное выражение истинно:

1.2.3.2.1.1. Для каждого элемента тела:

1.2.3.2.1.1.1. Обработка элемента тела (алгоритм обработки элемента тела)

1.2.3.2.1.2. Породить фрагмент вычислений while - копию текущего while с

инкрементированным на 1 значением счетчика

1.2.3.2.1.3. Сохранить этот фрагмент вычислений в распределенном хранилище

1.2.3.2.2. Если условное выражение ложно:

1.2.3.2.2.1. Получить значение имени выходного фрагмента вычислений (алгоритм получения имени фрагмента)

1.2.3.2.2.2. Породить фрагмент данных с этим именем и значением, равным текущему значению счетчика

1.2.3.2.2.3. Сохранить фрагмент данных в распределенном хранилище

1.2.4. Тип фрагмента вычислений - atom:

1.2.4.1. Для каждого i-го параметра внешней функции:

1.2.4.1.1. Если тип параметра - name (выходной фрагмент данных):

1.2.4.1.1.1. Получить значение имени фрагмента данных - i-го аргумента обрабатываемого фрагмента вычислений

1.2.4.1.2. Иначе:

1.2.4.1.2.1. Получить значение выражения - i-го аргумента обрабатываемого фрагмента вычислений

1.2.4.2. Вызвать внешнюю функцию, передав полученные значения в качестве аргументов

1.2.4.3. Если имелся параметр типа name (выходной фрагмент данных):

1.2.4.3.1. Породить фрагмент данных с полученным именем и значением, присвоенным в ходе исполнения внешней функции

1.2.4.3.2. Сохранить фрагмент данных в распределенном хранилище

2. Алгоритм обработки элемента тела

2.1. Породить фрагмент вычислений, описываемый элементом тела

2.2. Сохранить фрагмент вычислений в распределенном хранилище

3. Алгоритм получения значения выражения

3.1. Определить тип выражения

3.2. Обработать выражение:

3.2.1. Если тип выражения - константа:

3.2.1.1. Вернуть значение константы

3.2.2. Если тип - фрагмент данных:

3.2.2.1. Получить значение имени фрагмента данных (алгоритм получения значения имени фрагмента)

3.2.2.2. Запросить значение фрагмента данных в распределенном хранилище по имени

3.2.2.3. Вернуть значение фрагмента данных

3.2.3. Если тип выражения - условное выражение <, <=, =, >=, >:

3.2.3.1. Получить значение первого операнда

3.2.3.2. Получить значение второго операнда

3.2.3.3. Вернуть значение сравнения

3.2.4. Если тип выражения - математическая операция +, -, *, /:

3.2.4.1. Получить значение первого операнда

3.2.4.2. Получить значение второго операнда

3.2.4.3. Вернуть значение математической операции

4. Алгоритм получения значения имени фрагмента

4.1. Для каждого индекса в имени фрагмента:

4.1.1. Получить значение индекса (алгоритм получение значения выражения)

4.2. Вернуть имя фрагмента со значениями всех индексов

5. Алгоритм получения значения фрагмента данных из распределенного хранилища по имени

5.1. Получить номер узла, вызвав функцию Pathfinder для имени фрагмента данных:

5.1.1. Если номер узла совпадает с номером текущего узла - искать фрагмент данных в локальном хранилище:

5.1.1.1. Если найден - вернуть значение

5.1.1.2. Иначе - ждать появления фрагмента в локальном хранилище и вернуть значение при появлении

5.1.2. Иначе - запросить значение по сети у узла с полученным номером и вернуть его

6. Алгоритм сохранения фрагмента в распределенном хранилища

6.1. Получить номер узла, вызвав функцию Pathfinder для имени фрагмента:

6.1.1. Если номер узла совпадает с номером текущего узла - сохранить фрагмент в локальном хранилище

6.1.2. Иначе - отправить фрагмент по сети узлу с полученным номером

7. Алгоритм обработки сообщений

7.1. Определить адресата сообщения:

7.1.1. Если адресат - не текущий узел - переслать сообщение

7.2. Определить тип сообщения

7.3. Обработать сообщение:

7.3.1. Если тип сообщения - запрос значения фрагмента данных:

7.3.1.1. Искать фрагмент данных в локальном хранилище:

7.3.1.1.1. Если фрагмент данных не найден - ждать появления фрагмента в локальном хранилище

7.3.1.2. Послать значение фрагмента данных на узел, создавший сообщение

7.3.2. Если тип сообщения - посылка с фрагментом:

7.3.2.1. Поместить фрагмент в локальное хранилище