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: 09.04.01 ИНФОРМАТИКА И ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА

Образовательная программа: 09.04.01 ИНФОРМАТИКА И ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА. ВЫСОКОПРОИЗВОДИТЕЛЬНЫЕ ВЫЧИСЛИТЕЛЬНЫЕ СИСТЕМЫ.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА МАГИСТРА

Софронов Иван Викторович

(Фамилия, Имя, Отчество автора)

Тема работы Разработка алгоритма оптимизации синтеза фрагментированных программ и его реализация для системы программирования LuNA

Тема утверждена распоряжением по НГУ №___ от «___»_____2017 г.

Тема скорректирована распоряжением по НГУ №___ от «___»_____2017г.

«К защите допущена»

Заведующий кафедрой,
д.т.н., профессор

……/……..
(фамилия , И., О.) / (подпись, МП)

«……»……2017 г.

Научный руководитель

д.т.н., профессор, руководитель лаборатории синтеза параллельных программ, ИВМиМГ СО РАН

……/……...
(фамилия , И., О.) / (подпись, МП)

«……»……2017 г.

Дата защиты: «……»……2017 г.

Новосибирск, 2017 г.

МИНОБРНАУКИ РОССИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

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

Факультет информационных технологий

Кафедра……Параллельных вычислений…….
(название кафедры)

Направление подготовки: 09.04.01 ИНФОРМАТИКА И ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА

Направленность (магистерская программа)…Высокопроизводительные вычислительные системы..

УТВЕРЖДАЮ

Зав. кафедрой……
(фамилия, И., О.)
…….
(подпись, МП)
«……»……20…г.

ЗАДАНИЕ

НА МАГИСТЕРСКУЮ ДИССЕРТАЦИЮ

Студенту *(ке)*……Софронову Ивану Викторовичу……...
(фамилия, имя, отчество)

Тема……Разработка алгоритма оптимизации синтеза фрагментированных программ и его реализация для системы программирования LuNA ……..…….....
(полное название темы магистерской диссертации)

Исходные данные (или цель работы)..… В работе ставится цель создания системы управления компиляцией программ, позволяющей автоматизировать настройку программы на определенную конфигурацию вычислителя. Для этого вводятся критерии оптимизации, в соответствии с которыми реализуется та или иная последовательность оптимизирующих преобразований. ……..
…….....

Структурные части работы…… Введение, Обзор существующих подходов, Постановка задач, Описание алгоритма и его реализация, Тестирование, Заключение ……..

…….....

Научный руководитель
должность, место работы,
ученая степень, звание
……/……..
(фамилия , И., О.) / (подпись)
«…»……20…г.

Задание принял к исполнению
……/……..
(ФИО студента) / (подпись)
«…»……20…г.

АННОТАЦИЯ

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА МАГИСТРА

Наименование темы: Разработка алгоритма оптимизации синтеза фрагментированных программ и его реализация для системы программирования LuNA

Выполнена студентом(кой) Софроновым Иваном Викторовичем

Факультет информационных технологий, Новосибирский государственный университет
Кафедра Параллельных вычислений Группа 15221
Образовательная программа 09.04.01 Информатика и вычислительная техника.
Высокопроизводительные вычислительные системы
(направленность)

Объем работы: 32 стр.
Количество иллюстраций: 9
Количество таблиц: 2
Количество литературных источников: 18
Количество приложений: 1
Ключевые слова: автоматический синтез параллельных программ, конструирование параллельных программ, вычислительные модели, оптимизация при компиляции, фрагментированное программирование

Объектом исследования являются алгоритмы и структуры данных для обеспечения автоматического построения параллельных программ.

Целью работы является создание подсистемы управления конструированием программ на основе высокоуровневого описания задачи в рамках систем параллельного программирования (СПП). Управление конструированием в таких системах необходимо, т.к. реализация высокоуровневого описания может осуществляться несколькими вариантами. Нахождение оптимального варианта в зависимости от заданного критерия оценки и условий, в которых будет выполняться результирующая программа (количество процессов, узлов, и т.п.) является актуальной проблемой.

Для достижения цели работы выполнены следующие задачи:

- выбрать представление для хранения описания ситуаций, в которых нужно выбирать тот или иной вариант реализации;
- создать язык для описания выбранного представления и реализовать алгоритмы для работы с ним;
- выбрать существующую СПП и выявить особенности реализации в ней подсистемы управления конструированием;
- реализовать подсистему управления конструированием и провести тестирование её работы на примере конструирования программы расчёта численной задачи.

В результате работы была создана система управления конструированием программ. Новизна заключается в том, что предложен подход к автоматизации управления конструированием параллельных программ.

Полученные результаты могут быть использованы специалистами в областях создания систем автоматической генерации программ и создания оптимизирующих компиляторов языков программирования для обеспечения автоматического выбора вариантов преобразований в процессе их работы.

Софронов И.В _____

СОДЕРЖАНИЕ

1 Введение

В настоящее время активно развивается численное моделирование как метод изучения сложных систем. Обусловлено это тем, что компьютерные модели удобнее исследовать в тех случаях, когда реальные эксперименты затруднены из-за финансовых или физических препятствий. Численная модель отличается от реального объекта тем, что она отражает только те его свойства, которые интересны с т.з. эксперимента, такие свойства называются параметрами модели. Адекватно построенная численная модель может помочь в выявлении основных факторов, определяющих свойства изучаемого объекта-оригинала (или целого класса объектов), в частности, исследовать отклик моделируемой физической системы на изменения ее параметров и начальных условий [1].

Зачастую, для расчёта модели сколько-нибудь приближенной к реальности требуется большое количество ресурсов вычислителя. Поэтому использование обычного персонального компьютера приводит к тому, что задача не может выполниться из-за нехватки памяти, либо выполнение занимает неприемлемое количество времени из-за нехватки вычислительной мощности процессора. Одним из решений этой проблемы является использование в расчётах суперкомпьютеров, однако такое решение приводит к появлению новой трудности – необходимости создания параллельных программ (ПП).

Сложность создания ПП численного моделирования обуславливается тем, что помимо реализации алгоритма расчёта некоторой численной модели, необходимо дополнительно программировать взаимодействия процессов, осуществлять декомпозицию данных, производить балансировку вычислительной нагрузки между узлами вычислителя и пр. Получается, что исследователям в области численного моделирования необходимо тратить дополнительное время для написания своих программ, кроме того обладать навыком написания параллельных программ. Всё это приводит к тому, что исследователи ограничиваются достаточно простыми ПП, что затрудняет разработку новых численных моделей, более сложных в реализации.

Для оптимизации труда исследователей, занимающихся численным моделированием, создаются различные системы, облегчающие конструирование параллельных программ путём автоматизации тех или иных этапов их построения. К примеру, в последовательной программе выделяются секции, которые распараллеливаются (например, цикл *for* с независимыми итерациями), либо, в более сложном случае, пользователь с помощью высокоуровневого языка описывает алгоритм решения задачи, а система производит генерацию ПП по этому описанию. В рамках работы будут рассматриваться системы второго типа.

Один и то же алгоритм можно запрограммировать разными способами, каждый из которых обладает разными нефункциональными свойствами. Соответственно каждая полученная таким образом ПП будет работать в разных условиях (характеристики вычислителя, параметры выполняемой задачи и т.п.) с разной степенью эффективности. Под эффективностью здесь понимается удовлетворение результирующей программой заданных требований (далее в работе эффективность любой программы будет пониматься в этом смысле). Оценку того, насколько программа удовлетворяет требованиям, производят на основе критериев оценки результата. Например, если указать в качестве критерия времени работы программы, а в качестве требований – минимизировать это время, то чем меньше будет время исполнения выбранной реализации алгоритма на заданном вычислителе, тем более эффективной она будет считаться. При этом если описание алгоритма состоит из нескольких этапов или подалгоритмов, то для каждого из них можно также выбрать различные варианты реализации. В этом случае реализация описания алгоритма будет комбинироваться из выбранных реализаций для каждого его этапа.

Работа посвящена проблеме выбора реализации алгоритма решения задачи в процессе генерации ПП, описанного с помощью языка высокого уровня, таким образом, чтобы обеспечить получение эффективной ПП.

2 Обзор родственных работ

Проблема выбора модулей для реализации алгоритма в процессе конструирования программы схожа с проблемой, возникающей в компиляторах императивных языков программирования при выборе оптимизирующих преобразований для улучшения характеристик программы.

Наиболее распространённым подходом для её решения является использование фиксированных наборов оптимизации. Наборы составляются таким образом, чтобы для большинства программ из определённого класса (или классов), получалась оптимальная программа. Для указания набора, который следует выбрать в процессе компиляции, вводят некоторые средства управления. Зачастую для этого создаются специальные ключи компиляции. Например, в компиляторе GCC (the GNU Compiler Collection)[2] для языка Си используются ключи *–Оуровень* [3].

В работе [4] для выбора оптимизирующих опций используется метод итеративного поиска. Заключается он в повторении следующих шагов: некоторым образом составляется набор оптимизаций, программа компилируется с составленным набором, запускается и результаты её работы сохраняются. Так повторяется для всех возможных наборов, составленных из доступных оптимизаций компилятора. После чего результаты запуска

программ, полученных применением разных наборов оптимизаций, сравниваются по некоторому критерию, например, времени работы. В работе рассматривается несколько алгоритмов для обхода пространства оптимизаций.

Обход всего пространства – длительная операция, поэтому приходится либо тратить много времени на компиляцию, либо выбирать некоторое подмножество оптимизаций, которые применяются чаще всего – уменьшать количество потенциально возможных вариантов наборов оптимизации.

Логическим продолжением этой работы стала работа [5], в которой для улучшения показателей подхода и ускорения перебора была создана распределённая система нахождения вариантов. Суть заключается в том, что каждая итерация поиска оформлена в виде задачи, которая отправляется на узел некоторого расчётного кластера. При этом параметры узла кластера должны быть сходны целевой платформе. Для контроля за ходом выполнения создан web-интерфейс. После выполнения проверки на узле, результаты собираются в одном месте и сравниваются.

Однако с точки зрения оптимизации ПП это слабое улучшение, т.к. выполнение такой программы может занять длительное время. Решение интересно с точки зрения организации системы для отладки компилятора, ускорения и организацию проведения тестирования отдельных опций, определение параметров новых опций.

Описанные далее подходы используют следующее преставление наборов оптимизации. Каждый набор оптимизаций представляется в виде n-мерного бинарного вектора $\boldsymbol{V}$ (значение n равняется количеству оптимизаций в компиляторе). Каждой оптимизации компилятора присваивается номер, который сопоставляется с индексом $\boldsymbol{V}$. Определение состава набора по данному вектору $\boldsymbol{V}$ происходит тривиальным образом: если в позиции i установлена 1, то оптимизация компилятора с номером i включена в набор, если же в этой позиции 0, то соответствующая оптимизация не входит в набор.

Некоторым улучшением метода итеративного поиска является использование генетического алгоритма [6]. В подходе [7] генетический алгоритм используется для нахождения наборов оптимизаций для процедур. Чтобы улучшить производительность в рамках этой работы была реализована параллельная версия генетического алгоритма, где каждая хромосома из одного поколения выполняется на кластере из идентично настроенных машин.

В подходе [8] предлагается использовать генетический алгоритм для сокращения счётного множества вариантов наборов оптимизаций. Предполагается, что для каждого класса программ можно выделить в этом множестве подмножество «хороших» оптимизаций, которые будут оптимальными для этого класса программ. Таким образом, все программы

могут быть объединены в небольшие классы, для каждого из которых будут использоваться небольшое количество наборов оптимизаций.

В следующих подходах применяются методы, позволяющие динамически создавать наборы оптимизаций и сужать область поиска оптимизаций, чтобы не делать полный перебор. Эти методы используют связи, которые существуют между применяемыми оптимизациями и получаемыми свойствами программы. Выявить эти связи можно эмпирическим путём: применение некоторой оптимизации к тестовой программе, компиляция, запуск, после чего сравнение времени исполнения и других характеристик с результатами, полученными при запуске других оптимизаций. Полученные связи сохраняются в компиляторе для последующего использования. Найденные в литературе подходы различаются способом хранения выявленных закономерностей и алгоритмами работы с ними.

В [9] описан ещё один подход для оптимизации итеративного поиска. В подходе применяются методы математической статистики, чтобы сделать вывод о вероятностном распределении оптимизаций компилятора, которые нужно использовать для достижения наилучшей производительности программы. После того, как распределение будет построено, производится итеративный процесс компиляции, который перебирает элементы этого распределения. Как и в подходах на основе машинного обучения, на основе множества примеров программ вырабатывается статистическое отношение между свойствами выполняемой программы и оптимизациями компилятора. Свойства выполняемой программы получают путём анализа профиля её выполнения. Сети Байеса используются для запоминания статистической модели. При появлении новой программы, которой нет в обучающей выборке, её свойства будут поданы на вход Байесовской сети как некоторый факт в распределении. Внесение этого факта вносит смещение в распределение, т.к. оптимизации компилятора связаны со свойствами программ. Полученное вероятностное распределение зависит от рассматриваемой программы и позволяет сужать область поиска для итеративного алгоритма компиляции, рассматривая только те оптимизации, которые вероятнее всего приведут к получению оптимальной программы.

Довольно популярными в последнее время стали алгоритмы машинного обучения. При их использовании, на основе некоторой конечной выборки примеров, создаётся модель предметной области, которая отражает зависимости, содержащиеся в этих примерах. Созданная модель должна имитировать поведение реальной системы при поступлении схожих данных. Суть заключается в том, что в каждой программе выделяются характеризующие её особенности, эти особенности представляются в виде *n*-мерного вектора, который подаётся на вход модели и на выходе получается набор оптимизаций,

которые нужно применить к программе. Преимуществом такого подхода перед итеративными методами компиляции в том, что однажды обученная система уже больше не требует множественных запусков.

В подходе [10] для хранения существующих особенностей программ применяются онтологии. Онтологии представляют собой множество понятий некоторой предметной области, для которого определено множество бинарных отношений. Таким образом, с помощью онтологий можно составить описание любой программы, которое способен распознать компьютер и использовать для выбора варианта методами машинного обучения.

Недостатком подхода является то, что обученная система работает только в рамках данных, которые схожи с обучающей выборкой. Поэтому если появятся данные, которые сильно отличаются от тех, на которых система была обучена, то для эффективной работы системы нужно производить переобучение, что является длительным и сложным процессом. К тому же собрать достаточно большую выборку примеров не всегда удаётся

Ввиду того, что описанные подходы обладают недостатками и не решают полностью проблемы, проведение данного исследования целесообразно.

3 Цель работы

Для решения проблемы выбора реализации алгоритма в процессе генерации ПП предлагается использовать подход, основанный на вычислительных моделях (ВМ), описанных в теории синтеза параллельных программ [11].

ВМ используются для организации знаний о некоторой предметной области. В предметной области выделяются величины, которые существенны с т.з. полноты модели, а также операции, которые позволяют получить из одних величин другие. Таким образом, получение одних величин из других может быть выражено в ВМ в виде последовательности вызовов операций. При этом для одних и тех же величин может быть зафиксировано несколько вариантов вычисления. В случае предметной области «конструирование ПП» подход на основе ВМ заключается в том, что операции представляют собой процедуру реализации конкретных частей алгоритма, а величинами становятся: исходное описание алгоритма, результирующая программа и все промежуточные состояния описания, которые получаются в процессе конструирования программы, когда для части описания уже выбрана реализация, а для другой – ещё нет. Т.е. можно построить цепочку из вызовов операций, которая приведёт от описания алгоритма, до результирующей программы. При этом, на каждом шаге этой цепочки, может встретиться несколько вариантов реализации некоторой части описания алгоритма.

При выборе варианта реализации необходимо учитывать его свойства, так как свойства всех выбранных вариантов определяют свойства результата генерации. Чтобы можно было понять, какие свойства важны для оценки результирующей программы, необходимы критерии оценки результата. Введение критерия позволяет оценить программу, производя анализ её свойств. Однако в процессе конструирования программы неизвестны свойства результирующей программы, а известны только свойства вариантов реализации. Значит, необходимо определить зависимость, при которой выбор варианта с определённым свойством, приведёт к генерации программы с теми свойствами, которые существенны с т.з. критерия.

Описанный подход предлагается рассмотреть в рамках СПП LuNA [12], которая предназначена для конструирования ПП численного моделирования для суперкомпьютеров на основе высокоуровневого описания. В ней реализована парадигма фрагментированного программирования, в рамках которой описание алгоритма решения задачи представляется в виде взаимодействующих частей – фрагментов. Каждый фрагмент представляет собой единицу описания алгоритма и для реализации всего алгоритма нужно подобрать реализации для всех фрагментов. Поэтому в СПП LuNA также присутствует обозначенная ранее проблема. Для её решения предлагается реализовать подсистему, которая будет работать совместно с компилятором системы LuNA и обеспечивать выбор подходящей реализации для каждого фрагмента, из которых состоит алгоритм.

Целью работы является создание подсистемы управления конструированием программ на основе описания алгоритма решения задачи в рамках СПП LuNA. Описание алгоритма содержит отдельные этапы расчёта, для каждой из которых (или для всего алгоритма в целом) в процессе генерации ПП производится выбор реализации.

К создаваемой подсистеме предъявляются следующие требования:

- возможность автоматического выбора реализации подзадачи на основе вводимых пользователем критерия и информации о вычислителе;
- наличие средств работы с ВМ;
- независимость создаваемой подсистемы от различных систем параллельного программирования.

В работе ставятся следующие задачи:

- создание средств описания вычислительных моделей, разработка и реализация алгоритмов работы с ними;
- изучение особенностей работы СПП LuNA для последующего встраивания в неё создаваемой подсистемы;
- проведение тестирования работы созданной подсистемы на примере конструирования некоторой численной задачи для работы на вычислителе с заданными параметрами.

4 Необходимые термины и определения

4.1 Вычислительные модели

Вычислительные модели описаны в теории синтеза параллельных программ на вычислительных моделях [11]. Формальное определение ВМ следующее:

1) конечное множество $X=\{x,y,z,\ldots\}$ переменных для представления вычисляемых и измеряемых величин;
2) конечное множество $F=\{a,b,c,\ldots\}$ символов операций арности $m\times n$, $m\geq 0$, $n\geq 0$;
3) с каждым символом операции a арности $m\times n$ связан набор $¿(a)=(x_1,\ldots,x_m)$ входных и набор $out(a)=(y_1,\ldots,y_n)$ выходных переменных, при этом $i\neq j \rightarrow y_i\neq y_j$.

Пара $C=(X,F)$ называется простой ВМ. Операция $a\in F$ описывает возможность вычисления переменных $out(a)$ из $¿(a)$ с помощью некоторой процедуры. Графически ВМ можно представить в виде двудольного ориентированного графа (см. рисунок 4.1).

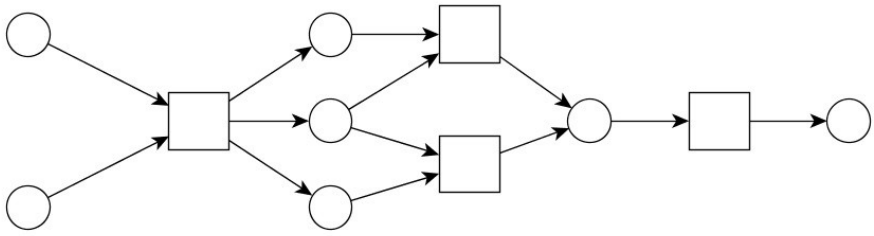

Рисунок 4.1 - Вычислительная модель; круги - переменные, квадраты - операции

На ВМ ставится задача – нахождение подграфа, который связывает одно подмножество вершин (исходное) с другим подмножеством (искомое), непересекающимся с первым. Процесс выполнения задачи на ВМ называется планированием, а результатом этого процесса является подграф – план. Для одной задачи может найтись несколько планов.

Расширим понятие ВМ понятием атрибутов. Атрибуты выражают определённую характеристику операции, существенную с т.з. заданного критерия оценки результата. Имея эту информацию, можно в процессе планирования на ВМ производить выбор операции, основываясь на критериях, относящихся к предметной области, а не на характеристиках плана, таких, например, как максимальная длина пути из исходного множества вершин в искомое. Любой операции в ВМ можно сопоставить множество атрибутов. Кроме того, введём функцию качества плана **Q**, которая ставит в соответствие каждому плану число – его оценку качества, на основании анализа атрибутов каждой операции плана. Для работы функции **Q** необходимо указать заданный критерий, который показывает как нужно оценивать каждый атрибут.

4.2 Поиск решения в вычислительной модели: планирование и выбор плана

Алгоритм планирования на ВМ описан в [11].

Пусть задана ВМ $C=(X,F)$, которая после трансляции представлена в виде двух таблиц *ТХ* и *ОП*. Каждая строка таблицы *ТХ* имеет вид $(x, A(x), comp(x))$, а таблицы *ОП* – $(a, \in(a), out(a))$. Здесь $x \in X$, $a \in F$, $A(x)=\{a \in F \mid x \in \in(a)\}$, $comp(x)=\{a \in F \mid x \in out(a)\}$. Алгоритм планирования состоит из двух частей: восходящей и нисходящей. В восходящей части алгоритма строятся множества переменных и операций, используемых в термах из множества $T_v=T(V,F)$ всех термов, порождённых V и F.

Обозначим $V_0=V$, тогда

$$F_0=\{a \in F \mid \in(a) \subseteq V_0\}=¿\, x \in V_0 \{a \in A(x) \mid \in(a) \subseteq V_0\}$$

содержит все операции ПВМ такие, что $¿(a) \subseteq V_0$. Далее формируется множество $V_1=\{x \in X \mid x \in out(a) \wedge a \in F_0\} \cup V_0$. На основе V_1 строится множество

$$F_1=¿\, x \in V_1 \{V_0 ¿ \{a \in A(x) \mid \in(a) \subseteq V_1\}$$

и т.д. до тех пор, пока при некотором целом положительном k не окажется, что $F_k=\varnothing$. На этом завершается восходящая часть алгоритма планирования. Если $W \not\subseteq V_k$, то планирование можно прекращать, так как в этом случае существует переменная в W, которая не вычисляется никаким термом из множества T_v, и, следовательно, не существует алгоритма решения сформулированной задаче на основе имеющихся знаний о ПО. В противном случае можно переходить к нисходящей части алгоритма.

Обозначим $F^{¿}=¿\, i=0 F_i$ и определим множества

$$G_1=¿\, x \in W \{a \in F^{¿} \mid a \in comp(x)\}, H_1=¿\, a \in G_1 ¿(a)$$

и далее, для $i=2,3,\ldots,$

$$G_i=¿\, x \in H_{i-1} ¿\, m=1 ¿\, i-1 G_m\} ¿, H_i=¿\, a \in G_i ¿(a).$$

Построение множеств G_i и H_i завершается, когда при некотором целом положительном r окажется $G_r=\varnothing$. После завершения планирования в таблицах *ТХ* и *ОП* остаются лишь переменные и операции из множеств G_i и H_i, остальные удаляются. Таким образом, результатом планирования является ВМ, оставшаяся от C после удаления из *ТХ* и *ОП* «лишних» переменных и операций.

Следующий после планирования этап – построение (V,W)-планов. Термы строятся из множества W, т.е для каждой переменной $x \in W$ выбирается из множества

$comp(x)$ одна из операций, которая вычисляет x, затем выбираются операции, которые вычисляют те входные переменные ранее выбранной операции, которые не входят в V и т.д., пока не будет построен терм t, вычисляющий x, и $¿(t) \subseteq V$.

4.3 Описание и исполнение алгоритма в СПП LuNA

Алгоритм решения задачи LuNA представляется в виде множества фрагментов данных (ФД) и множества фрагментов вычислений (ФВ) – такое представление называется фрагментированным алгоритмом (ФА), а его реализация – фрагментированной программой (ФП). ФД представляет собой переменную, которая может быть целым или вещественным числом, строкой, либо некоторым блоком данных. ФВ представляет собой операцию вычисления некоторой функции. Каждая такая функция реализуется некоторым программным модулем.

Модуль получает на вход набор входных ФД, на основе которых вычисляет набор выходных ФД. Подстановка ФД в качестве параметров модуля называется применением модуля к ФД (один и тот же модуль может применяться к различным ФД). Получается, что модуль используется для реализации ФВ. Соответствие между модулем и ФВ указывается явно.

Все ФВ занимают определённое место в ФА (ввиду порядка вычислений, заложенного в алгоритм), однако можно заменить реализацию ФВ – выбрать модуль, реализующий соответствующую функцию, т.е. выбрать реализацию для отдельных частей алгоритма решения задачи.

На рисунке 4.2 представлены элементы ФА и их связи в СФП LuNA, при этом для ФВ2 существует два модуля реализации.

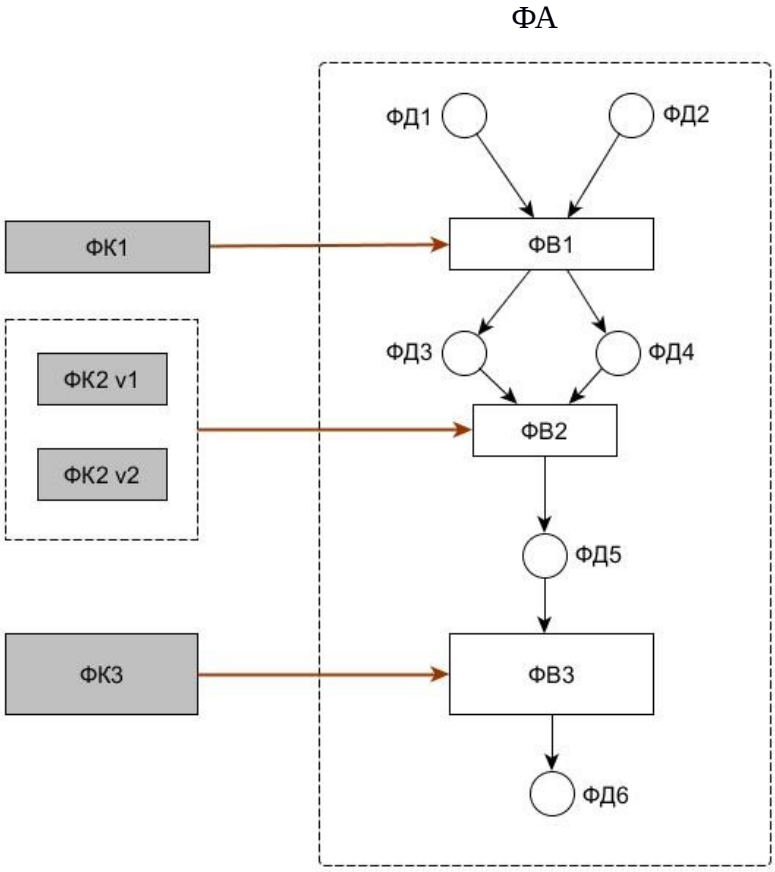

Рисунок 4.2 – Элементы ФА

Исполнение ФП, построенной по ФА, происходит на основе информационных зависимостей, которые присутствуют между ФВ: ФВ не начнёт выполняться, пока не будут вычислены все его входные ФД.

При описании ФА возможно описать структурированные ФВ, которые включают описания других ФВ. Отличием исполнения структурированного ФВ от простого или атомарного ФВ является то, что для него не назначается модуль, а происходит выполнение ФВ, которые описаны в нём. Таким образом можно строить иерархию вызовов ФВ.

5 Постановка задачи

Задача конструирования ПП в СПП LuNA ставится как задача выбора модуля для каждого ФВ, представленного в ФА. Выбор модуля необходимо осуществлять таким образом, чтобы результирующая ФП обладала требуемыми нефункциональными свойствами.

Каждый модуль обладает некоторым набором нефункциональных свойств, которые его отличают от остальных. Выбор модуля с определёнными нефункциональными свойствами влияет, в свою очередь, на нефункциональные свойства ФП и, соответственно, на её эффективность. Таким образом, существует связь между свойствами модулей и свойствами ФП. При зафиксированном критерии оценки результата, выявленную связь

можно использовать для выбора наилучшего варианта реализации алгоритма из доступных вариантов.

Для накопления и хранения связей требуется некоторое представление – база знаний, которую будет относительно просто модифицировать и использовать. При этом необходимо также автоматизировать процесс извлечения связей из базы. Для построения такой базы предлагается использовать ВМ: ВМ строится таким образом, что, в процессе построения плана, выбор операции, которая будет включена в план, соответствовал выбору определённого модуля. Тогда атрибуты операций в ВМ соответствуют свойствам модулей. При этом, предполагается, что конечный пользователь СПП ничего не знает о свойствах модулей, атрибутах или способах построения ВМ. Всё что ему нужно сделать – создать описание своего алгоритма на языке высокого уровня, поддерживаемого СПП, и указать критерии оценки результата.

Такая организация правил выбора позволяет автоматизировать сам процесс выбора, сделать его независимым от изменений, которые вносятся в правила, а также снизить вовлечённость конечного пользователя в процесс построения ПП.

6 Организация знаний о конструировании программ

Предлагаемое решение состоит из следующих частей. Во-первых, ВМ, описание создания которой приведено в разделе 6.1. С помощью неё описывается процесс выбора модуля для реализации ФВ и построения ПП на основе описания ФА. Во-вторых, создание специального языка для описания ВМ, синтаксис которого описан в 6.2. В-третьих, модули для реализации структурированных ФВ, описание которых приведено в 6.3.

6.1 Процесс создания вычислительных моделей

В процессе создания ВМ необходимо определить множество переменных и операций.

Переменная ВМ описывает величину в предметной области. Например, промежуточное состояние описания алгоритма в процессе конструирования или сгенерированные коды.

Операция ВМ содержит некоторый набор скриптов, которые в совокупности приводят к реализации функционала модуля в результирующей программе, т.е. через выбор операции, производится выбор модуля. Скрипты могут производить следующие действия: генерация кодов на языке C/C++; преобразование исходного описания алгоритма для уточнения и придания ему некоторых свойств; отсутствие действия. Использование операций без действий необходимо в некоторых случаях для реализации варианта «по умолчанию».

В описании операции ВМ указываются входные и выходные переменные. Входные переменные – это величины, которые необходимы для работы по встраиванию модуля, определённого в операции, а выходные – величины, получаемые в результате работы по встраиванию модуля. При этом, с помощью переменных, формируются информационные зависимости, которые задают порядок активизации модулей.

Если существует несколько вариантов получения некоторой величины, то выразить это можно с помощью операций ВМ, у которых совпадает множество входных переменных и совпадает множество выходных переменных. На рисунке 6.1 представлена часть ВМ, которая описывает эту ситуацию.

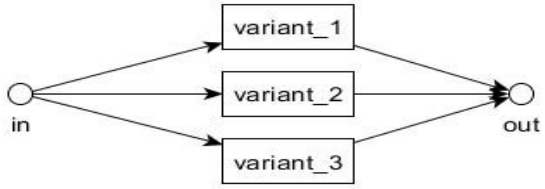

Рисунок 6.1 – Организация вариантов в ВМ

6.2 Синтаксис языка для описания вычислительных моделей

Чтобы можно было организовывать базу знаний на основе ВМ, содержащую связи между свойствами модулей и свойствами ФП, требуется язык для её описания, поэтому был разработан язык LCMD (Language for Computational Models Description).

Основными элементами описания служат объекты *model* и *task*. Объект *model* представляет собой описание переменных и операций в ВМ (см. Листинг 6.1).

Листинг 6.1 - Описание вычислительной модели

```
1.    model simple{
2.      variable input, output;
3.
4.       < attr1, attr2>
5.      { input } -> simple_op1 -> { output } :
6.         command 'Run command1' run 'some_prog1' with 'arg1', 'arg2' mod default;
7.
8.      { input } -> simple_op2 -> { output } :
9.         command 'Run command2' run 'some_prog2' mod default;
10.
11.   };
```

Переменные описываются после ключевого слова “*variable*” через запятую. Каждая операция начинается с описания множества входных переменных в фигурных скобках, после стрелки указывается имя операции, а далее, в фигурных скобках, - множество выходных переменных.

Над строкой с описанием операции в угловых скобках перечисляются атрибуты операции. Именно эти атрибуты будут учитываться при выставлении оценки плана. Значения атрибутов фиксированы и не могут быть изменены конечным пользователем.

Строка с описанием операции заканчивается двоеточием, после которого приводится список команд, разделённый точкой с запятой. Команда начинается с ключевого слова “*command*”, после которого следует строка, которая будет выводиться в консоли при запуске команды, далее следует ключевое слово “*run*”, после которого указывается строка с названием команды. Далее следует необязательная секция “*with*”, в которой через запятую перечисляются аргументы для вызываемой команды. Завершает строку секция с ключевым словом “*mod*”. Здесь указывается режим запуска команды. В данный момент поддерживается один режим - “*default*” - который запускает команды как есть, т.е. не добавляя никаких аргументов.

Для указания задания на планирование описывается объект *task*, который имеет две обязательные секции: множество неизвестных переменных, для нахождения которых строиться план (секция “*required*”) и множество известных переменных (секция “*given*”) (см. Листинг 6.2).

Листинг 6.2 - Описание задания на вычислительную модель

```
1.  task simple {
2.    given input;
3.    required output;
4.    using 2 of 4 in 3 nodes;
5.    criteria time_improvement;
6.  };
```

Далее следуют необязательные секции. Секция *“using”* используется для указания конфигурации вычислителя. Первое число указывает количество процессов, используемых на узле, второе число – максимальное возможное количество процессов на узле (количество процессорных ядер), третье – количество задействованных узлов с подобной конфигурацией.

В примере (Листинг 6.2) указано, что будет задействовано три узла, на каждом из которых будет запущено два процесса из возможных четырёх. В секции *“criteria”* указывается критерий оценки результата, в данном примере это улучшение времени работы программы, т.е. оценкой будет время работы результирующей программы.

Каждый объект *task* связан с объектом *model*, который определён в том же файле. Одной модели может соответствовать несколько заданий.

6.3 Уровни конструирования

Так как в ФА могут быть два типа ФВ, то имеется два уровня конструирования: конструирование атомарных ФВ и конструирование структурированных ФВ.

В первом случае из имеющегося набора готовых модулей выбирается один для реализации ФВ. Модуль может быть написан пользователем или подгружен из библиотеки готовых модулей, если такой имеется.

Во втором случае происходит конструирование части алгоритма, которая задана структурированным ФВ. Конструирование в этом случае означает реализация некоторой модели управления исполнением атомарных ФВ, которые описаны в структурированном ФВ. Например, выбор между моделями потока данных (dataflow) [14] и потока управления (controlflow) [15].

Если разным структурированным ФВ назначается разная модель исполнения, то необходимо обеспечить их совместную работу. Ввиду того, что ФВ работают с ФД, вводится некоторая общая среда, в которой определены связи между разными ФВ и каналы по обмену ФД.

7 Особенности реализации

7.1 Варианты реализации структурированных ФВ в СФП LuNA

В СФП LuNA существует два варианта реализации структурированных ФВ: базовый и FW, каждый из которых реализует определённую модель исполнения.

Базовый вариант реализует в процессе исполнения ПП потоковое управление (dataflow). В такой модели исполнительной системой LuNA производятся распределение ресурсов и управление вычислениями, на основе информационных зависимостей, которые присутствуют в описании алгоритма. При этом неизбежно появляются накладные расходы на организацию вычислений, приводящие к увеличению времени работы. С другой стороны, в ПП автоматически реализуются динамические свойства (такие как динамическая балансировка нагрузки на процессоры), что существенно упрощает разработку ПП, в сравнении с реализацией с помощью MPI [15].

FW (LuNA Framework) [16] реализует событийно-ориентированную модель управления (event-driven) [17], позволяет на стадии компиляции принять ряд решений по управлению вычислениями и распределению ресурсов и зафиксировать их в виде жесткой (без динамических свойств) управляющей программы.

7.2 Объединение структурированных модулей

При конструировании программы может возникнуть ситуация, когда разным структурированным ФВ будут назначены модули, реализующие разные модели управления вычислениями. В этом случае необходимо обеспечить обмен ФД между ними. Для этого используется контейнер LLRT (LuNA Library RunTime), который определяет логику взаимодействия разных вычислительных модулей путем описания способа передачи ФД между ними. На рисунке 7.1 приведён пример взаимодействия модулей LuNA и FW.

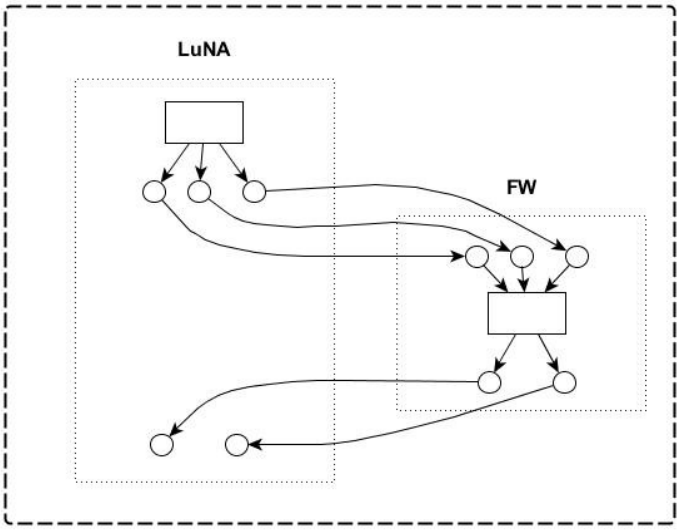

Рисунок 7.1 - Пример взаимодействия модулей LuNA и FW через контейнер LLRT

7.3. Использование атрибутов при расчёте оценки плана

Для оценки плана используются атрибуты операций, которые входят в него. Атрибуты являются характеристикой операции в ВМ и непосредственными пользователями не используются. Разработчик операции указывает атрибуты, чтобы выделить её ключевые особенности, существенные с т.з. результата конструирования. Каждый атрибут при анализе плана может учитываться или нет – это зависит от критерия. Если атрибут критерием учитывается, то возможны два варианта: атрибут подходит под критерий или нет. В первом случае к общей оценке плана добавляется единица, во втором – отнимается единица. Если атрибут не учитывается критерием, то оценка не меняется.

План с наибольшей оценкой признается системой как оптимальный по заданному критерию пользователя. В случае если оценки планов совпадают, то выбирается любой из них.

Для организации выбора реализации для ФВ был введён критерий *time_improvement*, который предписывает сконструировать программу с наилучшим временем исполнения, а также атрибуты:

- *few_proc* – операция будет удовлетворять критерию, если на узле будет использовано меньше половины от возможного числа процессов;
- *lots_proc* – операция будет удовлетворять критерию, если на узле будет использовано более половины от возможного числа процессов;
- *few_nodes* – операция будет удовлетворять критерию, при запуске на малом числе узлов (один или два);
- *lots_nodes* – операция будет удовлетворять критерию, при запуске на большом числе узлов (более двух).
- *default* – помечается операция, которую следует использовать при отсутствии явно указанных требований к программе.

7.4 Вычислительная модель процесса конструирования программы

Процесс конструирования ПП разделён на несколько частей: анализ, выбор реализации, агрегация. На рисунке 7.2 представлена часть ВМ, которая производит анализ введённого ФА.

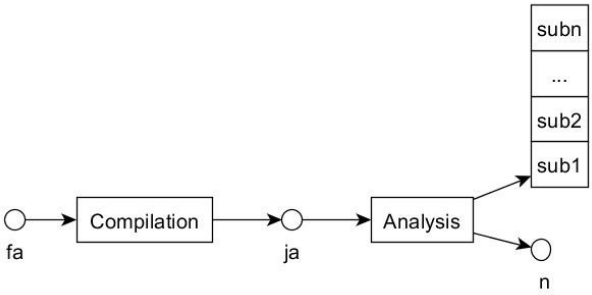

Рисунок 7.2 – Процесс анализа

Сначала производится компиляция во внутреннее представление системы, затем производится анализ количества структурированных ФВ, для которых будет производиться выбор реализации исполнения атомарных ФВ. В результате создаётся массив размера *n*, в котором хранятся ссылки на соответствующие описания структурированных ФВ – *sub*.

Далее следует этап выбора реализации для каждого *sub* и агрегация в единую программу всех сгенерированных исполняемых кодов. На рисунке 7.3 представлена остальная часть ВМ генерации.

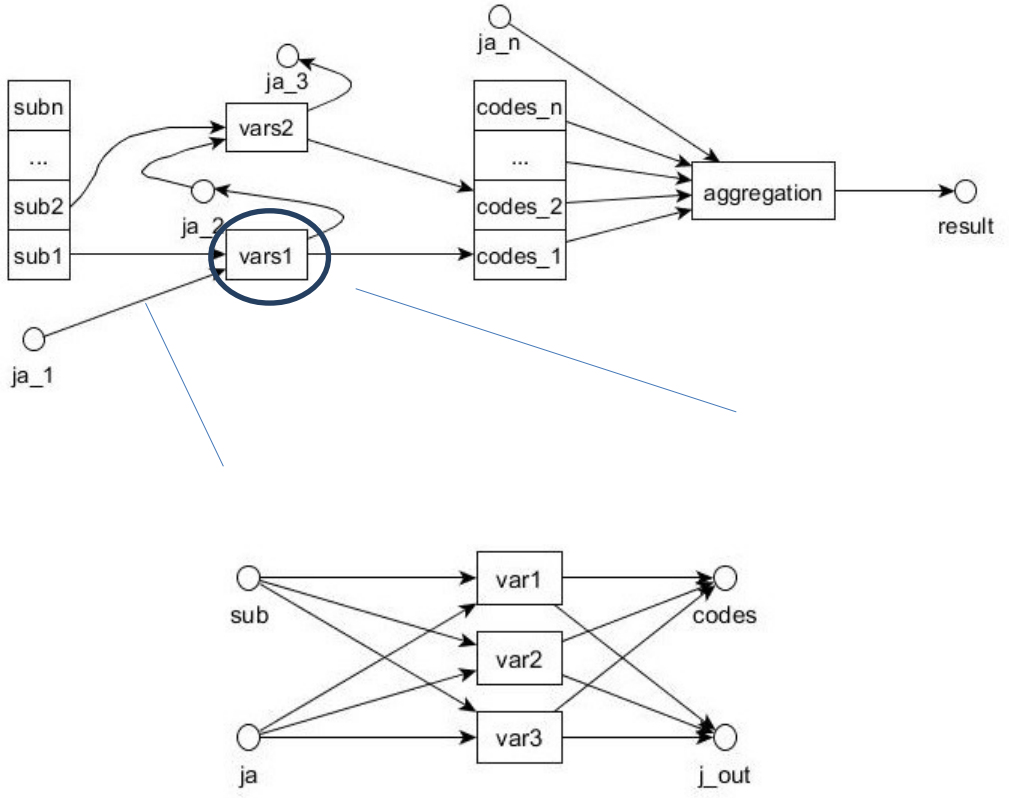

Рисунок 7.3 – Процесс выбора и агрегации

Для каждого *sub* и внутреннего представления (*ja*) производится реализация варианта (*vars*). При этом *vars* может быть структурированной операций и распадаться на несколько вариантов (*var* в нижней части рисунка 7.3). После того будет выбрана реализация для всех *n sub*, получится *n* элементов кода (*codes*), реализующих выбранный вариант исполнения для *sub* – все они, вместе с внутренним представлением алгоритма после *n-ой* итерации, поступают на вход операции агрегации, которая генерирует результат. На рисунке 7.4 представлена полная ВМ конструирования ПП.

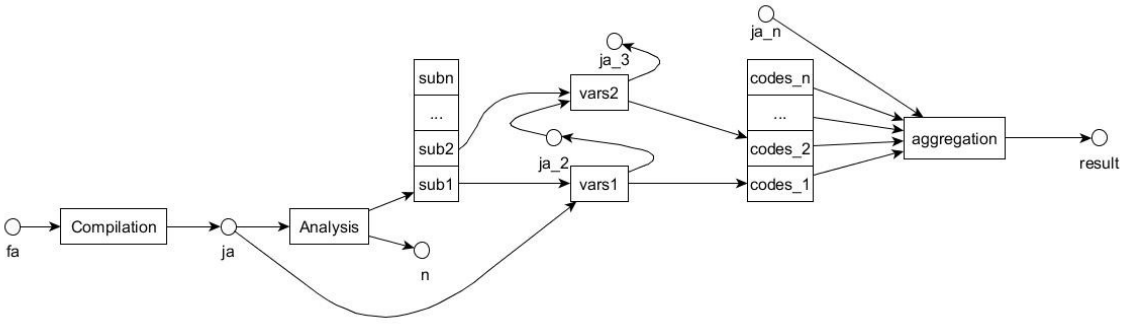

Рисунок 7.4 – ВМ процесса конструирования ПП

8 Применение подсистемы управления конструированием программ

Демонстрация работы созданной подсистемы производилась на одной из задач математической физики. Наиболее распространённым методом решения таких задач является

метод конечных разностей или метод сеток, который позволяет сводить приближённое решение дифференциальных уравнений в частных производных к решению систем алгебраических уравнений. Ввиду распространённости подхода, с некоторым приближением можно говорить о том, что полученные результаты тестирования распространяются на все задачи из этого класса.

Целью тестирования является продемонстрировать, что при указании конфигурации вычислителя, а именно: количества процессов, выделенных на узле; общее количество процессов на узле; количество задействованных узлов – будет выбрана наилучшая программа для этой конфигурации. В приложении А приведены описания ВМ, которые использовались в процессе тестирования.

8.1 Описание тестовой задачи

Для исследования эффективности предложенного алгоритма была использована фрагментированная реализация явной конечно-разностной схемы для решения уравнения Пуассона в трёхмерном пространстве с начальными краевыми условиями первого рода:

$$\nabla^2 \phi = f,$$
$$\phi|_G = F.$$

Трёхмерная регулярная сетка была разбита на фрагменты данных вдоль одной координаты. Значения в узлах сетки вычислялись по формуле:

$$\phi_{ijk}^{m+1} = \frac{\frac{\phi_{i+1\,jk}^m + \phi_{i-1\,jk}^m}{h_x^2} + \frac{\phi_{ij+1k}^m + \phi_{ij-1k}^m}{h_y^2} + \frac{\phi_{ijk+1}^m + \phi_{ijk-1}^m}{h_z^2} - f_{ijk}}{\frac{2}{h_x^2} + \frac{2}{h_y^2} + \frac{2}{h_z^2}}.$$

Запуск задач производился на кластере МВС-10П (2 процессора Xeon E5-2690, 64 ГБ оперативной памяти, коммуникационная сеть на базе FDR Infiniband) [18].

Параметры тестовой задачи: 128 итераций на равномерной сетке; размер расчётной области 1200x300x300; размер фрагмента 30x300x300 – на каждой итерации 40 фрагментов.

Две версии реализации основного цикла расчёта, использующие разные модели управления вычислениями: *luna*, в которой используется модель потоков данных (data-flow) и *fw*, в которой используется событийно-ориентированная модель (event-driven).

8.2 Результаты тестирования

Были проведены две серии тестов: в общей и распределённой памяти.

В таблице 8.1 приведены результаты запусков программы при разных моделях выполнения основного цикла расчёта. В зависимости от конфигурации вычислителя разные

реализации ведут себя по-разному: *fw* лучше работает на малом количестве процессов, *luna* – наоборот работает лучше, когда много процессов.

Таблица 8.1 – Результаты запусков различных реализаций основного цикла расчёта в общей памяти в зависимости от количества процессов

	1	2	4	8	16
luna	581,15	289,23	152,91	83,755	71,716
fw	301,1	168,88	85,55	86,186	109,07

Передавая в систему информацию о вычислителе в виде количества узлов и процессов и сообщая ей о намерении оптимизировать время выполнения, получаем версию программы, которая работает в любом случае хорошо, т.к. выбирается лучшая реализация для данных условий (график auto на рисунке 8.1).

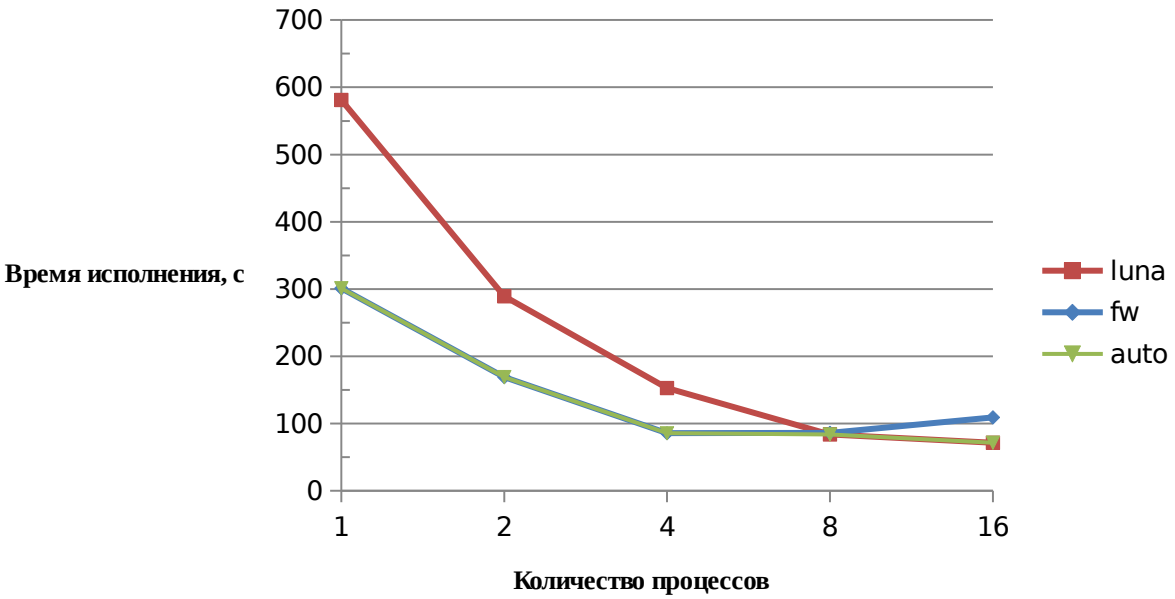

Рисунок 8.1 – Результат работы подсистемы выбора варианта реализации (зеленый график) в общей памяти

Во втором тесте исследовалось поведение системы при запуске на разном количестве узлов. Количество процессов оставалось постоянным и равнялось 8, количество узлов варьировалось от одного до четырёх. Для реализации основного цикла расчёта также было два варианта: *fw* и *luna*. В таблице 8.2 приведены время работы каждой из реализаций в разных условиях запуска, в первой строке первая цифра означает общее количество процессов, вторая – количество задействованных узлов.

Таблица 8.2 - Результаты запусков различных реализаций основного цикла расчёта в распределённой памяти в зависимости от количества узлов

	1/1	8/1	8/2	8/4
luna	591,966	88,143	82,236	81,101
fw	301,103	101,259	63,808	97,749

На рисунке 8.2 графиком *auto* отражён результат работы подсистемы. Также как и в предыдущем тесте подсистема, по введённым данным о вычислителе, принимала решение об оптимальной реализации расчётного цикла. Правила работали следующим образом: если используется много процессов на узле, то выбирается реализация *luna* (точка 8x1), если же на узле запускается мало процессов (в сравнении с общим количеством – восьмью) и мало узлов, то выбирается реализация *fw* (точка 8/2).

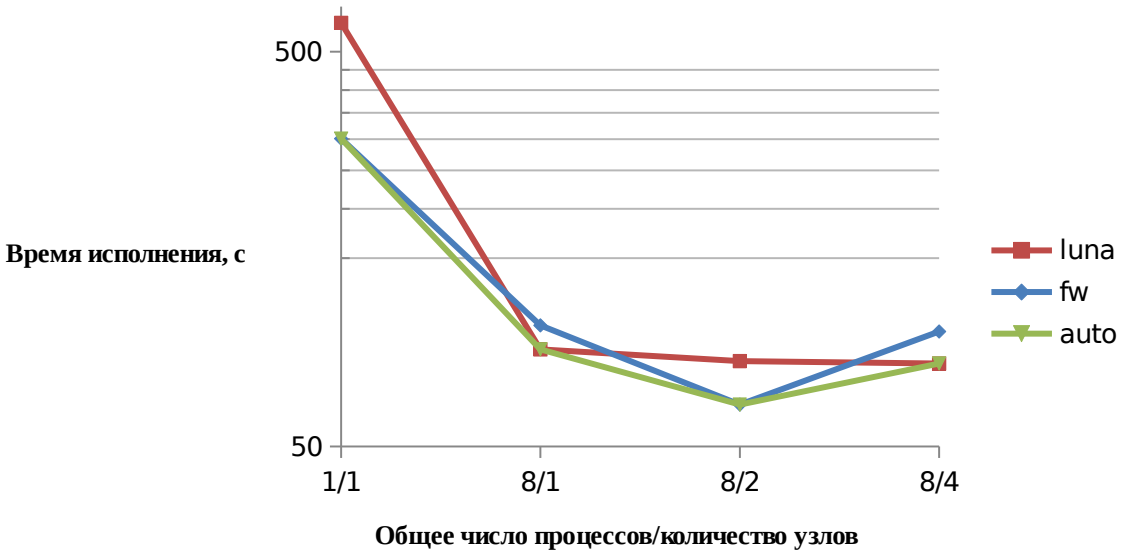

Рисунок 8.2 - Результат работы подсистемы выбора варианта реализации (зеленый график) в распределённой памяти

В результате тестирования была продемонстрирована возможность конструирования ПП в соответствии с критериями и параметрами вычислителя. Перед каждым запуском производился выбор той реализаций из доступных вариантов, которая была признана функцией качества как оптимальная. В результате, при каждом запуске, удалось получить наилучшую из возможных реализаций. Стоит отметить, что полученные результаты были достигнуты на узком классе программ, а введённые правила выбора реализации относительно параметров вычислителя не всегда дают положительный результат и

выбирается не самый оптимальный вариант реализации. Всё это указывает на сложность выработки критериев, атрибутов и правил для оценки вариантов реализации, которые бы покрывали широкий круг программ, и требует отдельного исследования.

9 Апробация результатов работы

Основные и промежуточные результаты работы были представлены на четырёх конференциях:

1. 54-я Международная Научная Студенческая Конференция (МНСК-2016), Новосибирск, 16-20 апреля 2016, доклад и тезисы;
2. XVII Всероссийская конференция молодых ученых по математическому моделированию и информационным технологиям (YM2016), Новосибирск, 30 октября - 3 ноября 2016, доклад;
3. 55-я Международная Научная Студенческая Конференция (МНСК-2017), Новосибирск, 16-20 апреля 2017, доклад и тезисы;
4. Конференция молодых учёных Института Вычислительной Математики и Математической Геофизики СО РАН, Новосибирск, 24-25.04.17, доклад и тезисы;

и двух школах:

1. XXVII Летняя школа по параллельному программированию, ИВМиМГ СО РАН, Новосибирск, 4-15 июля 2016, доклад;
2. The 1st International School on Heterogeneous Computing Infrastructure, Tomsk, 7-10 December 2016, постер и доклад;

Кроме того, в течение обучения, было сделано несколько докладов на студенческих научных семинарах кафедры Параллельных вычислений ФИТ НГУ

ЗАКЛЮЧЕНИЕ

Рассмотрена проблема выбора реализации алгоритма, описанного с помощью языка высокого уровня в системе параллельного программирования, в процессе генерации параллельной программы.

Проведён обзор родственных работ, решающих сходную проблему выбора оптимизаций компилятора для получения программы с требуемыми свойствами. Показано, что проблема является актуальной, однако рассмотренные подходы не решают её полностью.

Предложен подход для решения проблемы, основанный на применении теории структурного синтеза параллельных программ на вычислительных моделях. В рамках подхода были разработаны алгоритмы и структуры данных для обеспечения автоматического выбора варианта реализации алгоритма в процессе генерации параллельной программы.

Для управления построением параллельных программ для системы параллельного программирования LuNA была создана подсистема, реализующая предложенный подход.

Созданная подсистема удовлетворяет всем требованиям, которые были предъявлены к ней:

- реализован подход, в котором автоматический выбор реализации алгоритма осуществляется при указании пользователем только критерия оценки результата генерации и информации о вычислителе;
- создан язык для описания вычислительных моделей LCMD и его интерпретатор;
- разработанная архитектура подсистемы не зависит от конкретной системы параллельного программирования, что обеспечивает её переносимость.

Работа подсистемы продемонстрирована на примере конструирования программы для алгоритма решения уравнения Пуассона явным методом. Тестирование показало, что, в зависимости от указанной конфигурации, выбирается наиболее подходящая под условия реализация алгоритма, что позволяет автоматически генерировать наилучшую параллельную программу из возможных.

Выносится на защиту:

- подход к решению проблемы выбора реализации алгоритма решения задачи численного моделирования в процессе генерации параллельных программ, основанный на вычислительных моделях;
- средства для описания вычислительной модели и алгоритмы работы с ней;
- реализация алгоритмов выбора решения под заданные условия;
- описание и анализ результатов тестирования.

В дальнейшем планируется развивать предложенный подход и созданную подсистему: улучшить алгоритмы выбора вариантов и расширить класс программ, с которыми может работать подсистема.

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. Самарский, А.А. Введение в теорию разностных схем / А.А. Самарский // М.: Наука, 1971. – 553 с.
2. GCC, the GNU Compiler Collection – GNU Project - Free Software Foundation (FSF). URL: https://gcc.gnu.org/ (дата обращения 03.04.17)
3. Using the GNU Compiler Collection (GCC): Optimize Options. URL: https://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html (дата обращения 09.05.17)
4. Брусенцов, Л.Е. Автоматическая оптимизация при компиляции / Л.Е. Брусенцов // Открытые системы – 2011. – № 2. – С. 53–56.
5. Брусенцов, Л.Е. Распределенный итеративный поиск опций компиляции / Л.Е. Брусенцов // XIII Российская конференция с участием иностранных ученых «Распределенные, информационные и вычислительные ресурсы» (DICR'2010): материалы конференции – электронные данные. Новосибирск: ИВТ СО РАН, 2010 (CD-ROM). № гос. регистрации 0321100051. - http://conf.nsc.ru/dicr2010/ru/reportview/29810.
6. Генетический алгоритм. URL: http://www.machinelearning.ru/wiki/index.php?title=Генетический_алгоритм (дата обращения 10.08.16)
7. Jantz, M.R. Performance Potential of Optimization Phase Selection During Dynamic JIT Compilation/ M.R. Jantz, P.A. Kulkarni // In the ACM SIGPLAN Conference on Virtual Execution Environments (VEE '13). – 2013.
8. Purini, S. Finding Good Optimization Sequences Covering Program Space/ S. Purini, J. Lakshya //ACM Transactions on Architecture and Code Optimization – Volume 9, Issue No. 4, Article 39. – 2010.
9. Ashouri, A.H. A Bayesian Network Approach For Compiler Auto-tuning For Embedded Processors. / A.H. Ashouri et al. // ESTIMedia: Embedded Systems for Real-time Multimedia, 2014 IEEE 12th Symposium on Applied Machine Intelligence and Informatics, – 2014. – P. 90–97.
10. Hashimoto, M. Extracting Facts from Performance Tuning History of Scientific Applications for Predicting Effective Optimization Patterns / M. Hashimoto et al. // Proc. of the 12th Working Conf. on Mining Software Repositories. – 2015. – P.13-23.
11. Вальковский, В.А. Синтез параллельных программ и систем на вычислительных моделях / В.А. Вальковский, В.Э. Малышкин// Новосибирск: Наука, Сиб. отделение 1988. – 129 с.
12. Malyshkin, V.E. LuNA Fragmented Programming System, Main Functions and Peculiarities of Run-Time Subsystem / V.E. Malyshkin, V.A. Perepelkin // Proc. of the 11th Intern. Conf. on Parallel Computing Technologies (PaCT 2011), LNCS, Springer. – 2011 – Volume 6873. – P. 53—61.
13. What is a Data Flow Model? - Definition from Technopedia. URL: https://www.techopedia.com/definition/28523/data-flow-model (дата обращения 12.05.17)
14. What is controlflow? URL: https://www.computerhope.com/jargon/c/contflow.htm (дата обращения 29.05.17).

15. Message Passing Interface (MPI) URL: https://computing.llnl.gov/tutorials/mpi/ (дата обращения 25.05.17).
16. Ткачёва, А.А. Эффективное исполнение фрагментированных программ с помощью средств прямого управления в системе LuNA на примере задачи редуцирования данных / А.А. Ткачёва // Проблемы Информатики. – 2016. – №2(31). – с. 21-29.
17. What do you mean by “Event-Driven”?. URL: https://martinfowler.com/articles/201701-event-driven.html (дата обращения 02.06.17)
18. МСЦ РАН – Вычислительные системы. URL: http://www.jscc.ru/scomputers.html (дата обращения 18.05.17).

СПИСОК ПУБЛИКАЦИЙ

Софронова Ивана Викторовича

№ п/п	Наименование работы, её вид	Форма работы	Выходные данные	Объём в стр. или п.л.	Соавторы
1	2	3	4	5	6
1	Алгоритм управления компиляцией LuNA-программ, основанный на вычислительных моделях (тезисы)	печ.	Материалы 54-й Международной научной студенческой конференции МНСК-2016/ Новосибирский национальный исследовательский государственный университет. 2016. С. 123.	1	
2	Подсистема автоматизации конструирования параллельных программ для системы LuNA (тезисы)	печ.	Материалы 55-й Международной научной студенческой конференции МНСК-2017/ Новосибирский национальный исследовательский государственный университет. 2017. С. 107.	1	

Соискатель _____И.В.Софронов
(подпись)

Список верен:

Заведующий кафедрой
_____
профессор, д.т.н. В. Э. Малышкин

Секретарь ученого
совета ФИТ НГУ доцент О. А. Кутненко
__.06.2017

ПРИЛОЖЕНИЕ А

Вычислительная модель процесса построения программы

Листинг А.1 - Основная ВМ

```
model main {
  variable init, no_repo, loaded, decomp, llrt, cleaned, removed;

  { no_repo } -> load_repo -> { loaded }:
          command 'Loading projects' run './lcmdexec' with 'config/load.lcmd' mod default;

  { loaded } -> prepare -> { decomp }:
          command 'Preparing' run './lcmdexec' with 'config/prepare.lcmd' mod default;

  { decomp } -> constrution -> { llrt }:
          command 'Construction' run './lcmdexec' with 'config/construct.lcmd' mod default;

  { init } -> clean_prj -> { cleaned }:
          command 'Clean project' run 'bash' with '-c', 'config/scripts/clean.sh' mod default;

  { init } -> remove_bld -> { removed }:
          command 'Remove build' run 'bash' with '-c', 'config/scripts/remove.sh' mod default;
};

task full_build {
  given no_repo;
  required llrt;
};

task test_build {
  given loaded;
  required llrt;
  using 2 of 8 in 4 nodes;
  criteria time_improvement;
};

task load {
  given no_repo;
  required loaded;
};


task clean {
  given init;
  required cleaned;
};

task remove {
  given init;
  required removed;
};
```

Листинг А.2 - ВМ подготовки проекта

```
model prepare {
  variable fa, ja, dec, no_luna, luna,
          no_lurker, lurker,
          no_luna_fw, luna_fw,
          no_gtest, gtest;

  { no_luna } -> prepare_luna -> { luna }:
```

```
    command 'Prepare LuNA' run 'bash' with '-c', 'config/scripts/prepare_luna.sh' mod default;

  { no_luna_fw } -> prepare_luna_fw -> { luna_fw }:
    command 'Prepare LuNA FW' run 'bash' with '-c', 'config/scripts/prepare_fw.sh' mod default;

 { fa, luna } -> compilation -> { ja }:
    command 'Prepare poi fw' run 'bash' with '-c', 'config/scripts/prepare_poi_fw.sh' mod default;

 { ja, luna, luna_fw } -> decomposite -> { dec }:
    command 'Decomposition' run 'python' with 'config/scripts/decomp.py', 'poi1d_illrt/poid_illrt.ja' mod default;
};

task prepare_task {
 given no_luna,
       no_luna_fw,
       fa;
 required dec;
};
```

Листинг А.3 - Сгенерированная ВМ для программы решения уравнения Пуассона

```
model chooseExecutor {
        variable calc_max, main, puasson, find_maxdiff, prep_calc_max, prep_main, prep_puasson,
prep_find_maxdiff, main_llrt;

< lots_proc, lots_nodes, default >
{ calc_max } -> luna_calc_max -> { prep_calc_max }:


< lots_proc, lots_nodes, default >
{ main } -> luna_main -> { prep_main }:


< lots_proc, lots_nodes, default >
{ puasson } -> luna_puasson -> { prep_puasson }:

< few_proc, few_nodes>
{ puasson } -> fw_puasson -> { prep_puasson }:
        command 'Prepare puasson to run with FW' run 'bash' with  '-c', 'config/scripts/fw_gen.sh puasson' mod
default;

< lots_proc, lots_nodes, default >
{ find_maxdiff } -> luna_find_maxdiff -> { prep_find_maxdiff }:


{ prep_calc_max, prep_main, prep_puasson, prep_find_maxdiff } -> mkexec -> { main_llrt }:
        command 'Create LLRT main' run 'bash' with '-c', 'config/scripts/mkexecutable.sh' mod default;

};
task exe_task {
 given calc_max, main, puasson, find_maxdiff;
 required main_llrt;
 using 2 of 8 in 4 nodes;
 criteria time_improvement;
};
```