Министерство образования и науки Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ»

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

«Разработка эффективной структуры данных, поддерживающей инкрементальные изменения, для параллельной обработки деревьев»

Автор: У Цзюньфэн _____
Направление подготовки (специальность): 01.04.02 Прикладная математика и информатика
Квалификация: Магистр
Руководитель: Станкевич А.С., канд. техн. наук, без звания _____

К защите допустить
Зав. кафедрой Васильев В.Н., докт. техн. наук, проф. _____

«____» _____ 20_____ г.

Санкт-Петербург, 2016 г.

Студент У Цзюньфэн **Группа** М4236 **Кафедра** компьютерных технологий **Факультет** информационных технологий и программирования

Направленность (профиль), специализация Технологии проектирования и разработки программного обеспечения

Консультанты:

а) Буздалов М.В., канд. техн. наук, без звания _____

Квалификационная работа выполнена с оценкой _____

Дата защиты «20» июня 2016 г.

Секретарь ГЭК _____

Листов хранения _____

Демонстрационных материалов/Чертежей хранения _____

Министерство образования и науки Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ»

УТВЕРЖДАЮ
Зав. каф. компьютерных технологий
докт. техн. наук, проф.
_____ Васильев В.Н.
«____» _____ 20_____ г.

ЗАДАНИЕ
НА МАГИСТЕРСКУЮ ДИССЕРТАЦИЮ

Студент У Цзюньфэн **Группа** М4236 **Кафедра** компьютерных технологий **Факультет** информационных технологий и программирования **Руководитель** Станкевич Андрей Сергеевич, канд. техн. наук, без звания, доцент кафедры компьютерных технологий Университета ИТМО

1 Наименование темы: Разработка эффективной структуры данных, поддерживающей инкрементальные изменения, для параллельной обработки деревьев

Направление подготовки (специальность): 01.04.02 Прикладная математика и информатика

Направленность (профиль): Технологии проектирования и разработки программного обеспечения

Квалификация: Магистр

2 Срок сдачи студентом законченной работы: «31» мая 2016 г.

3 Техническое задание и исходные данные к работе.

Требуется разработать структуру данных для параллельной обработки деревьев, поддерживающую инкрементальные изменения (добавление нового узла, подвешивание узла в качестве ребенка другого узла, отцепление узла от его родителя, изменение меток на вершинах и ребрах). В разработанной структуре данных требуется снять ограничение на максимальную степень вершины, имеющееся у существующих реализаций. Требуется поддержка эффективного выполнения запросов на определения корня дерева по представителю дерева, определение групповой суммы меток в поддереве данной вершины и на пути между двумя данными вершинами. При этом требуется поддерживать произвольные типы меток на вершинах и ребрах, являющихся моноидами (произвольный моноид для ребер и коммутативный для вершин). Структуру данных требуется реализовать на языке C++ с использованием библиотеки PASL для реализации параллельных операций.

4 Содержание магистерской диссертации (перечень подлежащих разработке вопросов)

а) обзор источников с целью выявления наиболее перспективной структуры данных для поддержки операций с динамическими деревьями;

б) реализация выбранной структуры данных в виде контейнера языка C++, поддерживающего необходимые запросы и операции модификации;
в) реализация эффективного параллельного применения пакетов модификаций;
г) экспериментальные исследования эффективности разработанной реализации.

5 Перечень графического материала (с указанием обязательного материала)

Не предусмотрено

6 Исходные материалы и пособия

1 *Sleator D. D.*, *Tarjan R. E.* A data structure for dynamic trees // Journal of Computer and System Sciences. — 1983. — Vol. 26, no. 3. — P. 362–391.

2 *Reif J.*, *Tate S.* Dynamic Parallel Tree Contraction // Proceedings of SPAA. — 1994. — P. 114–121.

3 Dynamizing Static Algorithms with Applications to Dynamic Trees and History Independence / U. Acar [et al.] // Proceedings of SODA. — 2004. — P. 531–540.

7 Календарный план

№№ пп.	Наименование этапов магистерской диссертации	Срок выполнения этапов работы	Отметка о выполнении, подпись руков.
1	Изучение исходных материалов и пособий	ноябрь 2014	
2	Разработка программного интерфейса для реализации параллельных динамических деревьев	февраль 2015	
3	Реализация последовательного алгоритма	июнь 2015	
4	Реализация ядра параллельного алгоритма	ноябрь 2015	
5	Реализация параллельных операций с помощью библиотеки PASL	декабрь 2015	
6	Реализация параллельных операций с помощью альтернативной библиотеки	декабрь 2015	
7	Экспериментальные исследования и доработка параллельных алгоритмов	апрель 2015	
8	Оформление магистерской диссертации	май 2015	

8 Дата выдачи задания: «01» сентября 2014 г.

Руководитель _____

Задание принял к исполнению _____ «01» сентября 2014 г.

Министерство образования и науки Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ»

АННОТАЦИЯ
МАГИСТЕРСКОЙ ДИССЕРТАЦИИ

Студент: У Цзюньфэн

Наименование темы работы: Разработка эффективной структуры данных, поддерживающей инкрементальные изменения, для параллельной обработки деревьев

Наименование организации, где выполнена работа: Университет ИТМО

ХАРАКТЕРИСТИКА МАГИСТЕРСКОЙ ДИССЕРТАЦИИ

1 Цель исследования: Разработка структуры данных на основе Rake-and-Compress деревьев для параллельной обработки деревьев, поддерживающей инкрементальные изменения: добавление нового узла, подвешивание узла в качестве ребенка другого узла, отцепление узла от его родителя, изменение меток на вершинах и ребрах.

2 Задачи, решаемые в работе:

а) снятие ограничения на число детей вершины, присущее параллельным реализациям Rake-and-Compress деревьев;

б) уменьшение использования памяти с $\Theta(n \log n)$ до $\Theta(n)$, где n — общее число вершин в деревьях, при сохранении эффективности параллельных операций;

в) обеспечение поддержки меток произвольных типов для вершин и ребер, при условии что метки на вершинах являются элементами коммутативного моноида, а метки на ребрах — элементами произвольного моноида.

3 Число источников, использованных при составлении обзора: _____

4 Полное число источников, использованных в работе: 17

5 В том числе источников по годам

Отечественных			Иностранных		
Последние 5 лет	От 5 до 10 лет	Более 10 лет	Последние 5 лет	От 5 до 10 лет	Более 10 лет

6 Использование информационных ресурсов Internet: _____

7 Использование современных пакетов компьютерных программ и технологий: В реализации был использован язык C++ стандарта C++11. Для обеспечения целостности и надежного хранения кода использовалась система контроля версий Git. Для реализации параллельных операций использовалась библиотека Parallel Algorithm Scheduling Library (PASL).

8 Краткая характеристика полученных результатов: Разработана эффективная реализация Rake-and-Compress деревьев. Ограничение на число детей вершины снято путем раздваивания каждой вершины и организации декартова дерева для детей каждой вершины. Использование памяти уменьшено до $\Theta(n)$ при сохранении эффективности параллельных операций путем хранения вершин в контейнерах переменной длины (векторах) отдельно для четных и нечетных раундов сжатия Rake-and-Compress деревьев, что позволило разделить области памяти для чтения и для записи различными потоками.

9 Гранты, полученные при выполнении работы: гранты отсутствуют.

10 Наличие публикаций и выступлений на конференциях по теме работы: По теме выпускной работы были сделаны выступления на двух всероссийских конференциях, по итогам которых имеются следующие публикации:

1 *У Ц.* Обобщенная реализация RC-деревьев с параллельными обновлениями // V Всероссийский конгресс молодых ученых. — 2016.

2 *У Ц.*, *Буздалов М.* Обобщенная реализация укорененных Rake-and-Compress деревьев // Труды всероссийской конференции СПИСОК. — 2016.

Выпусник: У Цзюньфэн _____

Руководитель: Станкевич А.С. _____

«____» _____ 20_____ г.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Структуры данных для поддержания динамических деревьев имеют прямое приложение к эффективному решению многих практическим задач, таких как поиск максимального потока [1]. Такие структуры данных должны поддерживать лес деревьев на заданных вершинах, операции обновления (добавление и удаление ребер, а в случае укорененных деревьев — подвешивание вершины к вершине и отцепление вершины). Также, как правило, на вершинах, ребрах, или и тех, и других имеются пометки. Структуры данных также должны уметь отвечать на запросы вида «достижима ли вершина A из вершины B по ребрам деревьев», «найти длину пути из вершины A в вершину B», «найти сумму пометок на пути из вершины A в вершину B», «найти сумму пометок в поддереве вершины A» и другие. С практической точки зрения, полезно также уметь строить такие структуры данных параллельно, а также параллельно применять совокупности, или пакеты, операций обновления.

Впервые задача поддержания динамических деревьев была поставлена в работе Слейтора и Тарьяна [2], в которой была предложена структура данных, известная сейчас как ST-дерево или Link-Cut дерево. Эта структура данных поддерживает операции добавления и удаления ребер. Также она поддерживает пометки на ребрах и запросы вида «найти сумму пометок на пути из одной вершины в другую».

В то же время в ряде работ [3–6] Миллером и Рейфом был разработан подход параллельной обработки деревьев, известный как «параллельное сжатие деревьев» (англ. Parallel Tree Contraction). Суть этого подхода заключается в построении структуры данных, описывающей данное дерево и позволяющей выполнять запросы различного вида на этом дереве, таким образом, чтобы основная часть работы выполнялась параллельно. Тем самым, при наличии многопроцессорных вычислительных систем, сокращается время между получением входных данных и моментом, начиная с которого возможно эффективное выполнение запросов к дереву.

Различные варианты параллельного сжатия деревьев применяются для реализации динамических деревьев для ускорения применения изменений при наличии многопроцессорных вычислительных систем. Так, известны структуры данных Topology Trees [7], Top Trees [8] и RC-Trees [9], поддерживающие параллельное построение, при этом некоторые из них также поддерживают

применение различных операций модификации, в том числе параллельно. В работе [10] исследовалась возможность добавления поддержки инкрементальных изменений дерева путем привнесения идей из работы [3]. В то же время, эти структуры накладывают различные ограничения на вид поддерживаемого дерева или на поддерживаемые операции модификации.

Мотивация. Задача поддержки динамических деревьев все еще не может считаться решенной удовлетворительно. Так, многие из существующих структур данных не поддерживают параллельное обновление, работу с метками на ребрах или на вершинах, агрегационные запросы на путях или поддеревьях, или работу с деревьями, имеющими вершины произвольной степени. Остается открытым также вопрос эффективной реализации таких структур данных — в частности, по времени выполнения запросов, времени применения операций обновления, используемой памяти, использованию кешей различных уровней, эффективности использования ядер многопроцессорных систем.

Кроме того, для структур данных, поддерживающих параллельное построение или обновление, отсутствуют их реализации, удобные для использования в приложениях. Существующие программные реализации имеют исключительно исследовательский характер и трудны для переиспользования, в то время как для пользователей была бы удобна реализация, устроенная по образцу контейнеров или коллекций из стандартных библиотек языков программирования (таких как `std::vector` в *C++* или `java.util.ArrayList` в *Java*).

Контейнерную реализацию дополнительно усложняет тот факт, что использование отложенного обновления (в том числе параллельного) требует от структуры данных поддержки в том или ином виде минимум двух версий данных — последней стабильной версии и текущей версии, содержащей еще не примененные изменения. В дополнение к этому, вновь применяемые изменения крайне желательно проверять на корректность, причем в некоторых случаях такая проверка может занимать большую часть времени работы со структурой данных, поэтому также требуется предусмотреть режим, в котором корректность не проверяется.

Наличие указанных проблем мотивируют создание эффективной реализации структуры данных, поддерживающей параллельное применение пакетов

изменений, реализованной в виде переиспользуемого обобщенного контейнера данных, поддерживающего опциональную проверку корректности применяемых изменений.

Цели исследования. В настоящей диссертации, оцениваются преимущества структуры данных RC-Trees и анализируются требованию к разработке новой структуры данных на основе RC-Trees для решения динамических задач.

Целью данной диссертации является разработка и реализация эффективного параллельного алгоритма для сжатия дерева, поддерживающего инкрементные изменения данных. Для этого необходимо улучшить базовую структуру данных RC-Tree для поддержки коммутативных и некоммутативных данных, а также направленной и ненаправленной функций. Также требуется разработать параллельный алгоритм и оценить его эффективность. Для решения динамических задач в параллельных вычислениях в данной работе были использованы библиотеки OpenMP и PASL (Parallel Algorithm Scheduling Library).

Основной проблемой исследования является то, что неясно какую структуру данных следует использовать для достижения поставленных целей и какие ограничения на нее налагаются? Второй проблемой исследования является то, что какой алгоритм, основанный на разработанной структуре данных позволяет поддерживать динамические деревья?

Ожидаемые результаты. Задачи для которых применяются алгоритмы параллельного сжатия дерева, описанного в диссертации, могут быть использованы для оценки эффективности используемой структуры данных. Одной из задач, которая может быть решена с помощью алгоритма параллельного сжатия дерева является Change propagation [11]. Change propagation автоматически адаптирует выходные данные алгоритма к изменению входных. Другой задачей, которая решается алгоритмом сжатия дерева является вычисление выражений. Параллельное выполнение сжатия дерева могло бы сделать вычисления более эффективными.

Ожидаемыми результатами этой диссертации являются разработка новой структуры данных на основе RC-Tree, поддерживающей требуемый тип данных. Эффективность новой структуры данных должна быть приемлемой для последовательных вычислений. Должна существовать возможность установки

максимального времени вычисления для ответа на запрос. Также должны быть возможности последовательного и параллельного, с использованием OpenMP и PASL, выполнения. Также дожна быть возможность оценки эффективности с использованием различных вычислительных типов. При параллельной работе алгоритм должен отвечать на запросы быстрее, чем при последовательной обработке, иначе разработка параллельного алгоритма не имеет смысла.

Таким образом, ожидаемым результатом является разработка и реализация улучшенной версии базовой структуры данных, эффективность которой удовлетворяет требованиям диссертации.

ГЛАВА 1. ОБЗОР ИСТОЧНИКОВ

В данной главе приведен обзор литературных источников, используемых в настоящей работе. В разделе 1.1 описывается задача поддержания динамических деревьев и известные структуры данных для решения этой задачи. В разделе 1.2 описываются алгоритмы параллельного сжатия деревьев. В разделе 1.3 описываются вспомогательные алгоритмы и структуры данных, которые будут использованы при решении задач, поставленных в настоящей диссертации.

1.1. Методы решения задачи поддержания динамических деревьев

Структуры данных для поддержания динамических деревьев имеют прямое приложение к эффективному решению многих практическим задач, таких как поиск максимального потока [1]. Такие структуры данных должны поддерживать лес деревьев, операции обновления (добавление и удаление ребер, а в случае укорененных деревьев — подвешивание вершины к вершине и отцепление вершины). Также, как правило, на вершинах, ребрах, или и тех, и других имеются пометки. Структуры данных также должны уметь отвечать на запросы вида «достижима ли вершина A из вершины B по ребрам деревьев», «найти длину пути из вершины A в вершину B», «найти сумму пометок на пути из вершины A в вершину B», «найти сумму пометок в поддереве вершины A» и другие. С практической точки зрения, полезно также уметь строить такие структуры данных параллельно, а также параллельно применять совокупности, или пакеты, операций обновления.

1.1.1. Link-Cut Trees

ST-Tree является лучшей известной структурой данных для решения динамических задач, предложенной Sleator D D, Tarjan R E. (1983). Важными операциями этой структуры данных являются link и cut. Поэтому такая структура данных также известна как link-cut tree.

ST-Trees совершают операции на корневом дереве. В ST-Trees вводятся три операции: операция link (v, w), которая соединяет корневую вершину v и вершину w из другого дерева, добавляя ребро (v, w) и делая w родителем v; операция cut(v) удаляет ребро между некорневой вершиной v и его родителем, затем разделяет дерево на два дерева; операция evert (v) делает вершину v

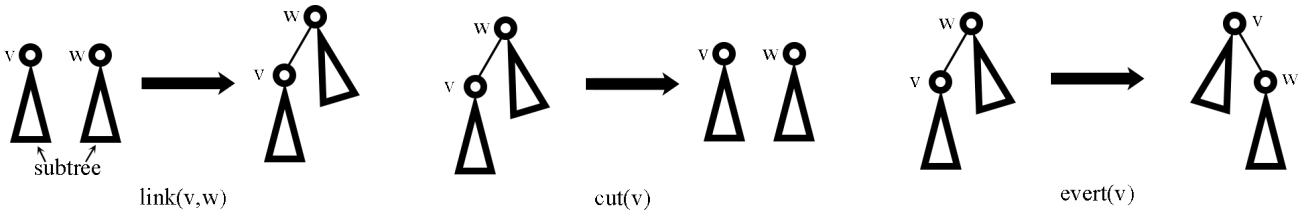

Рисунок 1 – Операции Link-Cut деревьев: `Link`(v, w), `Cut`(v), `Evert`(v)

корнем, поворачивая дерево относительно вершины v. Операции с ST-Trees изображены на рисунке 1.

ST-Trees поддерживают операции пути, требующие $O(\log n)$ времени на одну операцию на сбалансированном дереве в худшем случае, и амортизированно $O(\log n)$ времени на одну операцию на несбалансированном дереве. ST-Trees может также поддерживать другие запросы, но информация должна агрегироваться только над путями. ST-Trees является первой структурой данных в которой динамические запросы выполняются за логарифмическое время. Поэтому структуры данных для решения динамических задач основываются ST-Trees.

1.1.2. Topology Trees

Первой структурой данных, основанной на сжатии, является Topology Tree, которую ввел Frederickson, G. N. (1997). Topology Tree является деревом поддержки. Рассмотрим дерево сжатия S на основе дерева T. Каждый узел в S является поддеревом в T, и кластер сжатия дерева T является поддеревом, соответствующим узлу в S. Пример топологического дерева представлен на рисунке 2. Вверху изображено дерево T, под ним представлено дерево T', являющегося кластером сжатия S. Ребро в T соответствует инцидентному ребру T'. Степенью кластера T' является число инцидентных ребер. Вершина кластера T' из S является граничной вершиной, если она имеет инцидентное ребро. Число граничных вершин кластера T' является размером его граница (Acar U.et al. 2003).

Frederickson, G. N. (1997) предложил интерпретировать кластеры как вершины вместо ребер в качестве дерева поддержки. В Topology Trees, кластер имеет не более трех степений. Направленная структура данных Topology Trees разработана для поддержания динамических бинарных деревьев. Время работы операции в таких деревьях занимает $O(\log n)$, где n является количеством вершин в деревьях. Topology Trees являются реализацией link-cut trees,

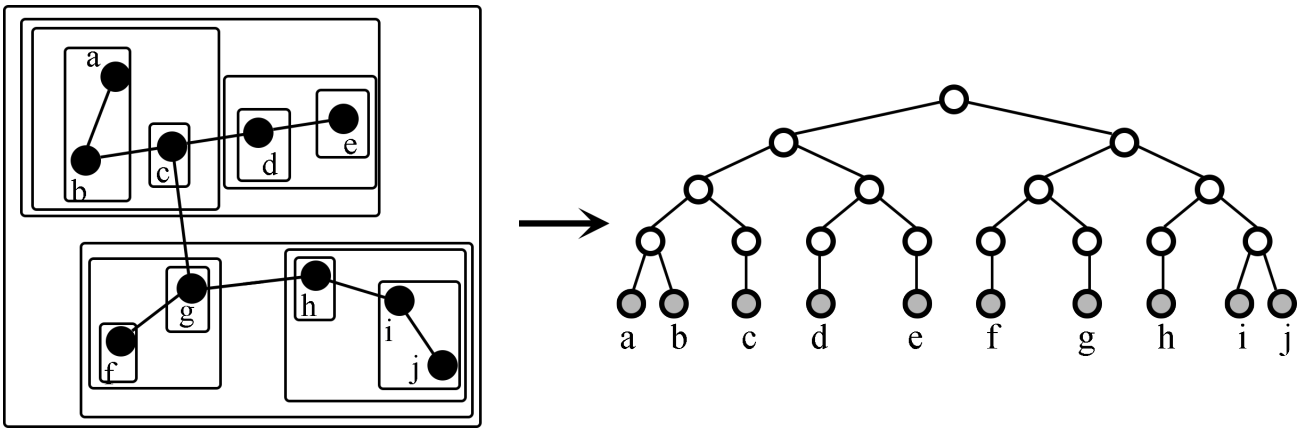

Рисунок 2 – Пример Topology Tree

так что основными операциями в Topology Trees также являются link и cut. Хотя Topology Trees упрощают сжатие динамического дерева с точки зрения алгоритма, требование что степень всех вершин в лесу ограничена константой, ограничивает разработку. Для более эффективной работы алгоритма, структура данных должна поддерживать изменяемую степень вершин в дереве поддержки.

1.1.3. Top Trees

Структура данных Top Trees представляет собой структуру данных разработанную на основе двоичного дерева для некорневых динамических деревьев (Alstrup S. et al. 1997). В некорневом дереве, ребра, смежные с каждой вершиной произвольно расположены в фиксированном круговом порядке. Кластером в Top Trees является поддерево пути первоначального базового дерева. В Top Trees кластеры имеют размер не более двух. Операция сжатия занимает $O(\log n)$ времени, и преимущество состоит в том, что время не зависит от степени дерева. Ограничением Top Trees является то, что она поддерживает только бинарные деревья.

Сжатие дерева состоит в том, чтобы соединить все кластеры в дереве поддержки в единый кластер. Top Trees поддерживает операцию сжатия для бинарного дерева. Два кластера (u, v) и (v, w) могут быть объединены, если v является общей вершиной этих двух кластеров и имеет степень равную двум. Alstrup S. et al. (1997) использует понятие компресса, объединенный кластер (u, w) из двух кластеров называется компрессом кластера. Затем, если (w, x) является преемником (v, x) и v имеет степень один, эти кластеры могут быть объединены в объединенный кластер. После комбинации, объединенный кла-

стер имеет граничные вершины w и x. Каждый объединенный или компресс кластер может рассматриваться как родитель, который агрегирует информацию, содержащуюся в ее детях. Корень верхнего дерева представляет собой кластер, который полностью лежит в основе дерева. При динамических изменениях лежащего в основе дерева, например, при применении операций link или cut, структура данных обновляет сжатие.

Структура данных Top Trees позволяет отвечать на динамические запросы. При выполнении путевого запроса, запрашивается путь между вершиной v и w, затем выполняется операция expose (v, w), которая возвращает корневой кластер с вершинами v и w в качестве конечных вершин, а в случае если v и w находятся в разных деревьях, операция будет возвращать нуль. Однако, даже если v и w находятся одном дереве, сжатие верхнего дерева может потребоваться изменить, потому что корневым кластером станет путь от v до w. Top Trees поддерживает агрегацию по путям или по деревьям напрямую. Они не требуют определенную степень вершины, в отличие от Topology Trees что является преимуществом.

1.1.4. RC-Trees

Структура данных Rake-and-Compress Tree [6] — одна из наиболее подходящих структур данных для решения задач настоящей диссертации, теоретически поддерживающая параллельное построение и параллельное обновление (включая не только добавление и удаление листьев дерева, но и произвольные подвешивания и отцепления вершин), а также запросы как на путях, так и на поддеревьях.

Идея Rake-and-Compress деревьев, в изложении для укорененных деревьев, заключается в следующем. Для дерева последовательно строится несколько «уменьшенных» копий дерева, причем i-тая копия получается из $(i-1)$-ой применением ко всем допустимым вершинам одной из следующих операций:

— Rake. Вершина, имеющая родителя и не имеющая детей, удаляются вместе с ребром, ведущим к родителю (рисунок 3).

— Compress. Вершина, имеющая родителя и ровно одного ребенка, удаляется вместе со смежными ребрами, а вместо этого ребро проводится от ребенка удаляемой вершины к родителю этой вершины (рисунок 4). При

этом родитель не должен подвергаться операции Compress, а ребенок не должен подвергаться ни операции Compress, ни операции Rake.

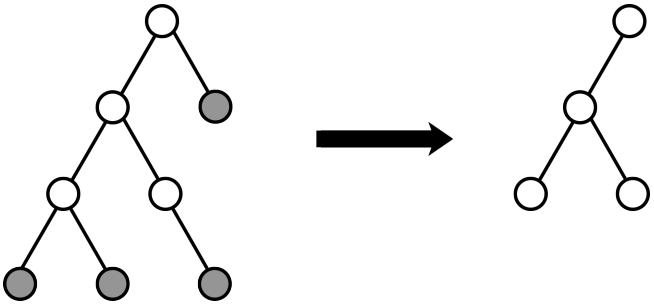

Рисунок 3 – Операция Rake

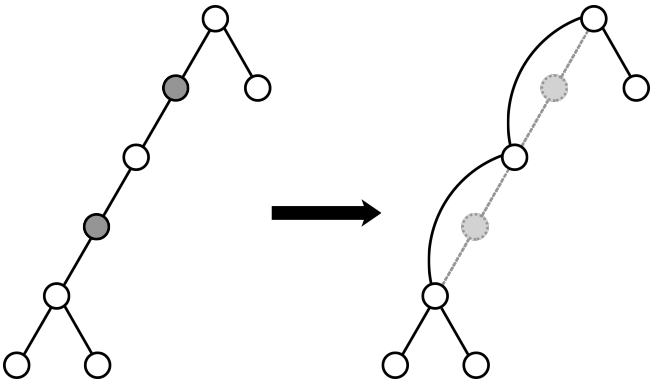

Рисунок 4 – Операция Compress

С каждым раундом число вершин в дереве, если оно больше единицы, уменьшается как минимум в $8/7$ раз в каждой компоненте связности. В конце работы для каждой компоненты связности остается ровно одна вершина, для чего требуется произвести $O(\log n)$ раундов (рисунок 5). RC-дерево можно представить как дерево с двумя типами вершин — первый тип соответствует вершинам оригинального дерева, второй тип — его ребрам (рисунок 6).

Такое многоуровневое представление дерева позволяет отвечать на запросы на путях и поддеревьях за время, пропорциональное числу раундов (а следовательно, $O(\log n)$), даже если дерево изначально было несбалансировано (например, существенно вытянутое). В то же время решение о выполнении операции Rake или Compress можно сделать на основе локальной информации (о самой вершине и ее ближайших соседях), что позволяет эффективно распараллеливать операции в рамках каждого раунда. Эффективное применение операций обновления также возможно с использованием этой структуры данных [10].

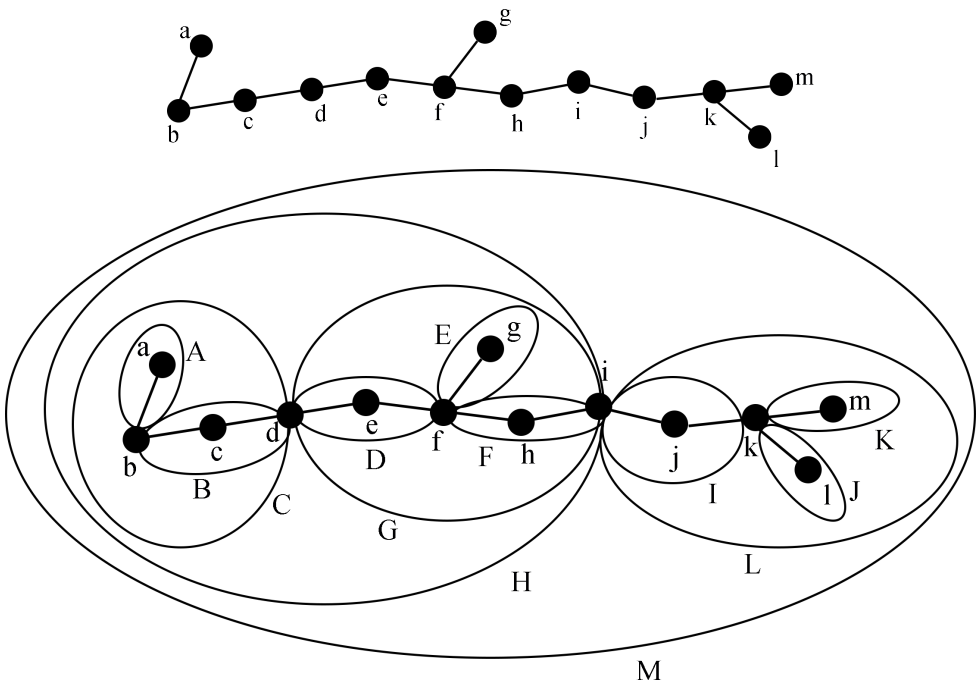

Рисунок 5 – Исходное дерево и группировка вершин, полученная последовательным применением операций Rake и Compress

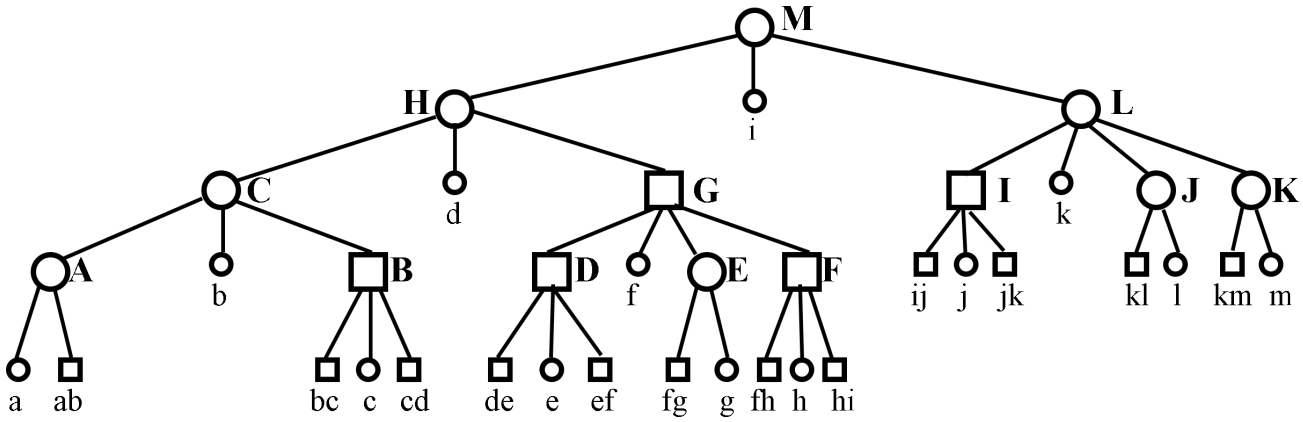

Рисунок 6 – Представление RC-дерева как дерева с двумя типами вершин, соответствующим вершинам (круглые вершины) и ребрам (квадратные вершины) оригинального дерева

1.2. Алгоритмы параллельного сжатия деревьев

Динамические деревья решают предназначенные для них задачи последовательно. Тем не менее, сжатие дерева позволяет этим структурам данных работать более эффективно. Сжатием дерева является операция для уменьшения дерева путем удаления некоторых из узлов (Atallah, M. J. (Ed.). 1998). Чтобы операция сжатия дерева поддерживала изменяемые процессоры [3, 4], было введено параллельное сжатие дерева [5, 6]. Фреймворк для параллельного сжатия дерева поддерживает эффективные параллельные алгоритмы на деревьях [12]. Эффективный алгоритм параллельного сжатия дерева позволя-

ет собирать значения в дереве эффективно и без конфликтов, что в результате, приводит к нескольким эффективным параллельным алгоритмам [13].

Параллельное сжатие дерева является подходом обработки деревьев снизу вверх. Это облегчает реализацию новых алгоритмов с меньшим количеством процессоров и меньшим временем выполнения, и упрощает решение сложных задач с помощью параллельных алгоритмов [3]. Параллельное сжатие дерева занимает $O(\log n)$ шагов независимо от формы дерева, что в результате ускоряет выполнение операций над деревом [14].

При параллельном сжатии дерева, существуют две абстрактные операции, выполняемые на деревьях [3], называемые Rake и Compress. Операция Rake перемещает все листья с дерева T. Операция Compress удаляет все максимальные цепи дерева T за один шаг, где максимальная цепь означает, что каждый узел имеет родителя и одного ребенка, а последний узел цепи имеет ребенка, и этот ребенок не является листом. Таким образом операция Compress удаляет каждые два ребра между двумя узлами в цепи дерева T. Например, в бинарном дереве, операция Compress удаляет узел, у которого есть только один ребенок и у его родителя тоже ровно один ребенок. Дополнительная операция сжатия, является одновременным применением операций Rake и Compress по всему дереву. Задача сжатия заключается в уменьшении дерева до одного узла. Для начала определяется корневое дерево T с n узлами и корнем r. Операции Rake и Compress выполняются параллельно и в любом порядке. Время работы сжатия с помощью $O(n/\log n)$ процессоров занимает $O(\log n)$ времени [3].

Морихата и Мацудзаки [12] ввели алгоритм сжатия, называемый Shunt, основанный на алгоритме параллельного сжатия дерева из работы [3]. Алгоритм Shunt состоит из операций Rake и Compress, а также новой операции Shunt. Операция Shunt удаляет лист и его родителя и соединяет брата листа с родителем его предка. Операция Shunt приводит к простому и эффективному параллельному сжатию бинарных деревьев.

1.3. Вспомогательные алгоритмы и структуры данных

Для того, чтобы новая структура данных была эффективной, при ее реализации используются следующие вспомогательные алгоритмы и структуры данных:

— Декартово дерево, используемое в качестве вспомогательного дерева для хранения данных, чтобы избавиться от ограничения на степен вершин;

— параллельный алгоритм вычисления префиксных сумм, который используется для параллельного формирования списка активных вершин при переходе с одного уровня RC-дерева на другой.

1.3.1. Декартово дерево

Декартово дерево является двоичной древовидной структурой данных, которая может быть построена из последовательности чисел. В работе [15] впервые введена структура данных декартового дерева для эффективного выполнения запросов на интервалах. Декартовые деревья широко применяется для поиска на интервалах, наименьшего общего предка и других задач. Операция построения декартового дерева из последовательности чисел может быть выполнена за линейное время. Пример декартового дерева приведен на рисунке 7.

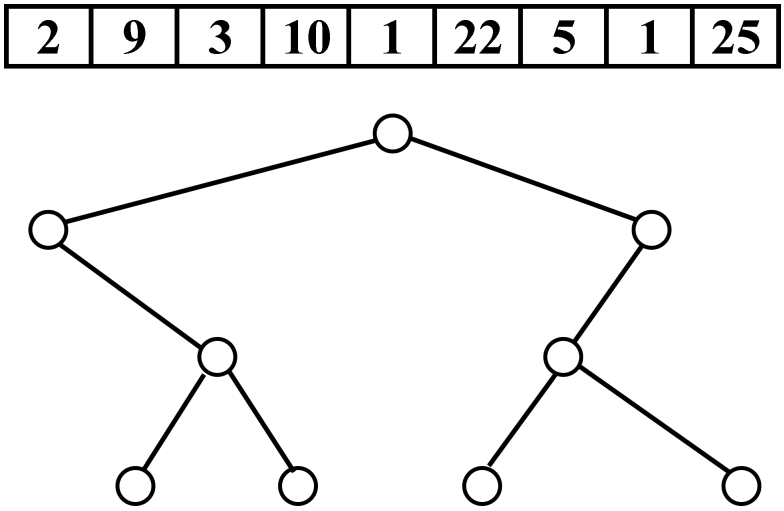

Рисунок 7 – Пример декартова дерева

Декартово дерево выполняет базовые операции, такие как вставка, удаление, поиск элемента, разрезание и склеивание, в среднем за $O(\log n)$ [15]. Декартовые деревья, построенные из последовательности различных чисел обладают следующими свойствами: узлы в декартовом дереве соответствуют элементам в исходной последовательности, то есть каждый элемент в последовательности соответствует единственному узлу в дереве, каждый узел в дереве также соответствует единственному элементу в последовательности. Используя упорядоченный обход, можно вывести исходную последовательность чисел: любой нижний индекс элемента из последовательности, представленной в левом поддереве дерева узла меньше, чем нижний индекс элемента из последовательности, представленной узлом, любой нижний индекс элемента из

последовательности, представленной в правом поддереве дерева узла больше, чем нижний индекс элемента из последовательности, представленной узлом.

1.3.2. Параллельный алгоритм вычисления префиксных сумм

Параллельный алгоритм вычисления префиксных сумм является одним из наиболее важных «кирпичиков» для параллельной работы с данными [16]. Оно было введено в [17]. Данный алгоритм вычисляет все префиксные суммы по данному массиву чисел. Идея алгоритма приведена на листинге 1.

Листинг 1 – Алгоритм параллельного вычисления префиксных сумм

```
1: for (j:=1 to log n do) do
2:     for (all k in parallel do) do
3:         if k ⩾ 2^j then
4:             x[k] := x[k − 2^(j − 1)] + x[k]
5:         end if
6:     end for
7: end for
```

Асимтотическая сложность данного алгоритма при использовании $O(N)$ процессоров при работе с массивом размера N составляет $O(\log n)$. На рисунке 8 показан пример работы параллельного алгоритма вычисления префиксных сумм.

Выводы по главе 1

В главе приведен обзор методов решения задач поддержания динамических деревьев, алгоритмов параллельного сжатия деревьев, а также некоторых вспомогательных алгоритмов и структур данных. В частности, подробно рассмотрены RC-Trees и показана необходимость разработки динамического алгоритма их перестроения. Описаны принципы построения декартова дерева, а также параллельный алгоритм вычисления префиксных сумм, что необходимо для разработки требуемого алгоритма.

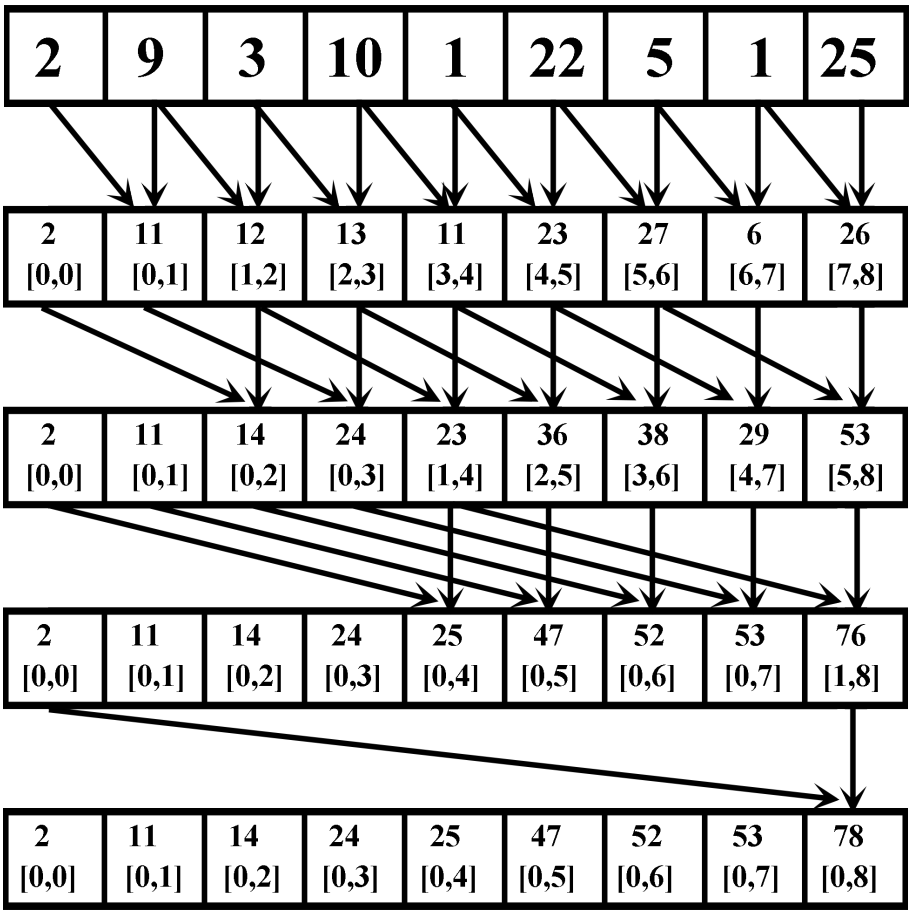

Рисунок 8 – Иллюстрация идеи параллельного алгоритма вычисления префиксных сумм

ГЛАВА 2. ПОСТАНОВКА ЗАДАЧИ

Проблемы исследования заключаются в том, что структура данных может быть использована для достижения поставленных целей и то, какие ограничения существуют и каков алгоритм, основанный на структуре данных решающих задачи динамического дерева. В результате обзора литературы, структуры данных сжатия параллельного дерева не достаточно для решения исследовательской задачи, потому что необходима эффективная структура динамического дерева данных. Для того, чтобы ответить на динамические запросы и продлить параллельную обработку, должна быть разработана эффективная динамическая структура дерева данных. Сравнивая динамические структуры дерева данных, RC-Trees имеют особенности, которые имеют так же ST-Trees, Topology Tree и Top Tree. RC-Trees улучшенные по Acar U. et al.(2003) использующие распространение изменения могут поддержать изменения в обновлении. В результате, структура данных RC-Trees подходит как базовая структура данных, используемая для покрытия дополнительных изменений в параллельной обработке.

Особенность RC-Trees сама по себе требует дополнительного осуществления RC-Trees, таким образом, первое улучшение должно завершить реализацию RC-Trees. Во-вторых, структура данных RC-Trees нуждается в более технически продвинутой реализации, чтобы сделать код пригодным для использования в современных приложениях. В-третьих, RC-Trees в параллельном процессе недостаточно эффективны, реализация должна сделать параллельный процесс в каждом Rake и Compress. Другими словами, в каждом вычислительном раунде каждая операция Rake должна выполняться параллельно, так же как и операция Compress. Тогда Rake и Compress должны выполняться параллельно.

Кроме того, в настоящее время в теории RC-Trees и структура не поддерживают данные некоммутативного типа, например, матрицы. Таким образом, в реализации и улучшении эта функция должна быть добавлена, и лучше поддержать направленные и ненаправленные данные тоже. Структура данных должна работать эффективно и с простыми данными и с данными больших объемов.

2.1. Требования к реализации структуры данных

Задачи исследования заключаются в необходимости решить то, как структура данных может быть использована для достижения поставленных целей и какие ограничения существуют и каков алгоритм, основанный на структуре данных решения задач динамических деревьев. В частности, выбранная структура данных должна поддерживать динамические запросы с постепенными изменениями. По причине того, что текущая работа состоит в том, чтобы структура данных работала в параллельной обработке, следует ответить какие данные параллельных вычислений идут на поддержку структуры. Тогда следующая проблема, если структура данных уже выбрана, заключается в том, что алгоритм должен быть выбран так, чтобы заставить структуру данных работать.

По сравнению с перечисленными структурами данных, RC-Tree структура данных пригодна. Она поддерживает динамическое дерево и сжатие параллельного дерева. Она вцелом способна решить задачи, и RC-Tree структура данных представленая Acar.U. поддерживает изменения в процессе обновления. RC-Trees реализуют алгоритм сжатия параллельного дерева, включая Rake и Compress. Распространение изменения делает возможным поддержку дополнительных изменений.

Хотя первоначальная структура данных RC-Tree в основном отвечает требованиям для решения исследовательской задачи, но она не поддерживает коммутативные и некоммутативные данные. Кроме того, направленные ребра с информацией может оказаться сложными для вычислений. Сложность структуры данных ограничивает расширение для переключения на параллельные вычисления. Ограниченная степень RC-Tree поддерживает не все виды деревьев в лесу.

В результате, RC-Trees не удовлетворяют поставленной задаче, и новая структура данных должна быть разработана на основе RC-Trees и должна поддерживать коммутативные и некоммутативные данных. Для того, чтобы избежать ограничения для ограниченной степени, вспомогательная структура может быть использована для изменения исходного дерева в используемое и хранимое дерево. В следующих разделах будет описана разработка новой структуры данных.

2.2. Интерфейс для запросов и модификаций

Новая структура данных, разрабатываемая в данной работе должны поддерживать динамические деревья, так что операции выполняются в лесу. В пользовательском слое должен быть создан корневой лес, потому что все запросы основаны на этом лесу. В корневом лесу, есть корневые деревья, такие что каждая вершина в лесу имеет корень. В общем, вершина, которая не имеет родителя является корнем. Как правило, в динамическом дереве, алгоритм в основном поддерживает запросы на получение номера вершин, ребер и корней в лесу. Кроме того, алгоритм поддерживает запросы к вершине на число её детей, возвращает его родителя, проверяя, если он является корнем, получает информацию, которую она несет, или несут её ребра. Вершина несет информацию о самой себе, которая в этой диссертации называется информацией вершины, а также информацию о её реберах, которые соединяют её с родителем или детьми, которая называется информация ребра. Информация о вершине поддерживает коммутативную информацию и отвечает на запросы о поддереве. Информация о ребре имеет две части, вверхнюю информацию, которая направлена к родителю, и низнюю информацию, которая исходит от детей. Ребра могут нести гораздо больше информации, чем вершины, при этом ребра предназначены для поддержки как коммутативный и некоммутативную информации.

В реализации алгоритма, существуют три основных интерфейса поддерживающие ответы на запросы о динамическом дереве: получить корень, получить путь и получить поддерево. Интерфейс для получения корня ищет родителя вершины, от которой был сделан запрос. В случае, если родитель вершины не является корнем, рекурсивно запрашивается родитель родителя вершины и таким образом в конечном счете находится корневая вершина дерева. Данная операция занимает O (1) времени. Интерфейс, отвечающий на запрос пути, возвращает сводную информацию о ребрах на пути между двумя вершинами запроса. Очевидно, что только если две вершины расположены в одном и том же дереве, путь между ними может быть найден. Для того, чтобы получить путь, если две вершины находятся в одном и том же маленьком поддереве, то просто нужно сделать несколько шагов навстречу друг другу, вверх и вниз соответственно, пока они не встретятся; если две вершины не в одном и том же маленьком поддереве, они должны дойти до самого нижнего по уровню обще-

го предка. Чтобы получить поддерево вершины нужно добавить информацию о его поддереве и нем самом. Ясно, что в RC-Tree только вершина, которая не является корнем, и которая должна быть сжата имеет поддерево, информация о поддереве является объединением информации ее ребенке и поддереве этого ребенка.

Чтобы отвечать на динамические запросы более эффективно, был добавлен механизм расписаний, который обновляет дерево в расписании. Каждый раз, когда некоторые изменения сделаны в лесу, он будет проверять, если изменения могут быть сделаны. Каждое изменение может быть записано, но, покуда изменения не внесены в расписание, изменения не будут внесены в лес. Операции, производиные пользователями, будут вызывать функцию расписания. Так же они могут вносить сведения о параметрах вершины или ребра. Так же ребра между двумя вершинами могут быть подключены или удалены. Изменения, сделанные в дереве могут быть отменены или применены, функция отмены обновляет дерево, как и другие операции, применение функции выполняет все изменения, сделанные на дереве и, наконец, изменяет дерево. Функция расписаний делает последующую параллельную работу более интуитивной и удобной. Для ускорения вычислений при использовании функции расписаний, параллельные операции выполняются в функции, когда применяемая функция вызывает их.

Выводы по главе 2

В главе приводится постановка задачи разработки эффективной структуры данных, поддерживающей инкрементальные изменения. Сформулированы требования к реализации структуры данных, описан интерфейс для запросов и модификаций.

ГЛАВА 3. РАЗРАБОТКА СТРУКТУРЫ ДАННЫХ

3.1. Снятие ограничения на степень вершины

В этой диссертации предполагается, что каждая вершина в лесу имеет не более трех детей. Установка ограничения на степень используется для более простого вычисления. В бинарном дереве каждая вершина имеет не более двух детей, третий ребенок используется для внутренней структуры данных, используемой для выполнения вычислений.

Пользователи могут создавать любые деревья, динамическое дерево не обязано быть бинарным. В этой диссертации, внутреннее дерево — декартово дерево, используется для того чтобы хранить фактические данные. Когда выполняется операция присоединения, она обращается к декартову дереву, чтобы соединить две вершины. В декартовом дереве, индекс вершины не совпадает с индексом вершины в представленном дереве, а информация о ребре будет храниться в ребенке в типе данных вершины. Например, вершина v, в декартовом дереве имеет индекс 2v. Вершина в декартовом дереве не влияет на вершину в представленном дереве, но содержит информацию о ней. Дочерняя вершина в представленном дереве будет располагаться в поддереве его родителя в декартовом дереве. Каждое изменение в представленном дереве, такие как вложение, отсоединение и добавление информации о вершине будет вызывать изменение декартового дерева, необходимое для обновления фактического сохраненного дерева.

Основной проблемой имеющихся реализаций является константное ограничение на максимальную степень вершины [10]. В настоящей работе его предлагается преодолевать с помощью следующего приема. Каждая вершина оригинального дерева разбивается на две вершины — *вершину данных* (или «нижнюю» вершину) и *вершину связей* (или «верхнюю» вершину). Вершина данных всегда имеет родителем соответствующую вершину связей. При этом все дети той или иной вершины организуются в двоичное дерево поиска, корнем которого служит вершина данных родителя, а остальными вершинами — вершины связей детей. В качестве двоичного дерева поиска использовалось декартово дерево [15]. При таком подходе степень каждой вершины не превосходит трех, что позволяет использовать алгоритмы для вершин с константной степенью. На рисунке 10 приведено оригинальное дерево и дерево, получаю-

щееся путем удвоения вершин и организации детей каждой вершины в виде дерева.

Каждая вершина оригинального дерева снабжена меткой, тип которой является коммутативным моноидом, при этом структура данных поддерживает операцию «вычислить моноидную сумму меток всех вершин в поддереве данной вершины». Аналогично, каждое ребро оригинального дерева снабжено двумя метками (метка, соответствующая направлению от ребенка к родителю, или «вверх», и аналогичная метка «вниз»), тип которых совпадает и является моноидом (возможно, некоммутативным), при этом структура данных поддерживает операцию «вычислить моноидную сумма меток всех ребер на пути от вершины A к вершине B для данных вершин A и B». При удвоении вершин, метки вершин и ребер, выходящих вверх из вершин оригинального дерева, переходят в соответствующие метки соответствущих вершин данных, в то время как метки вершин и выходящих вверх ребер у вершин связей равны нейтральным элементам соответствующих моноидов. Таким способом достигается корректность выполнения запросов независимо от структуры деревьев, организующих детей вершин.

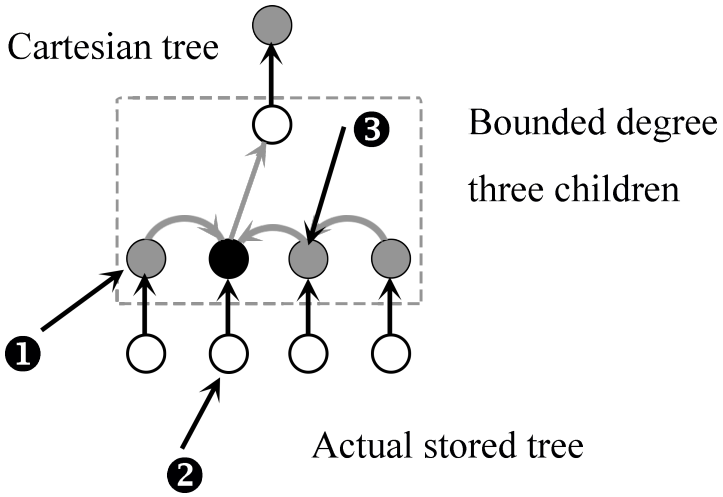

Рисунок 9 – Представление вершины и ее детей с помощью декартова дерева

3.2. Хранение данных с использованием $O(N)$ оперативной памяти

Структура данных, разработанная в диссертации, поддерживает операции динамических изменений. Рассмотрим механизм использования структуры данных расписания. Уровни используются для разъяснения операций. В каждой вершине хранится структура данных типа «столбец», в которой записывается каждое изменение вершины, как показано на рисунке. Все новые

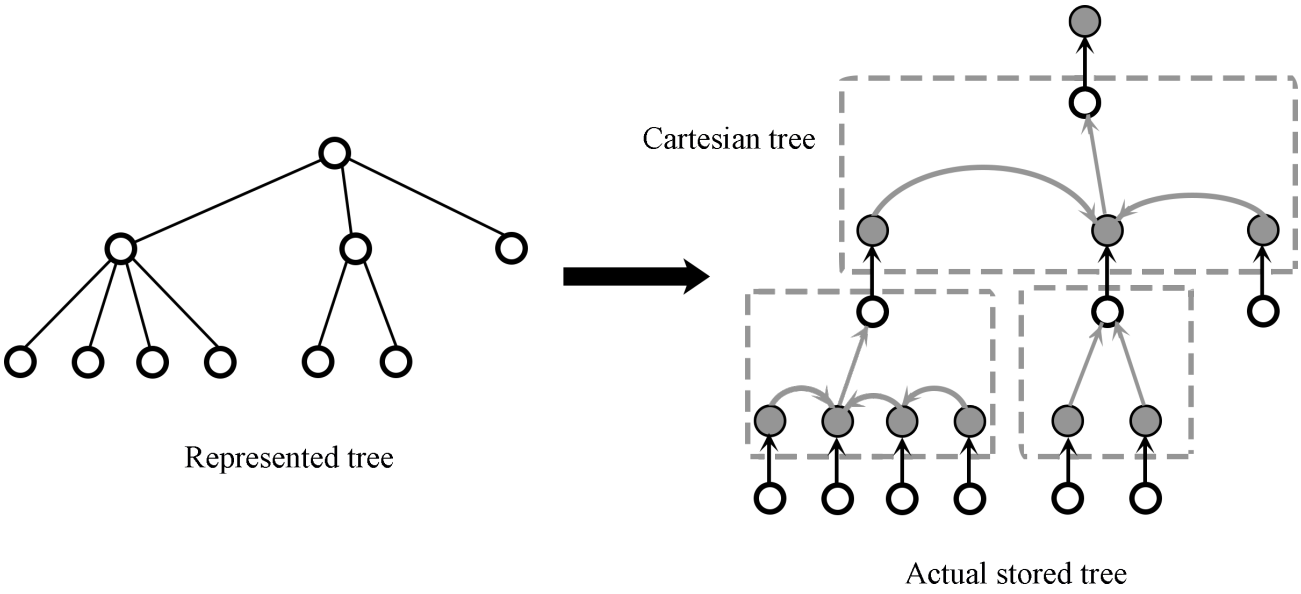

Рисунок 10 – Исходное RC-дерево и дерево, которое хранится в памяти

созданные вершины на первом уровне структуры, как только они меняются, выталкиваются на следующий уровень, это выглядит как во второй матрице. Вершины типа столбца имеют свою информацию об уровне. В результате упрощается проверка последнего активного уровня, в котором вершина может корректно выполнять операции.

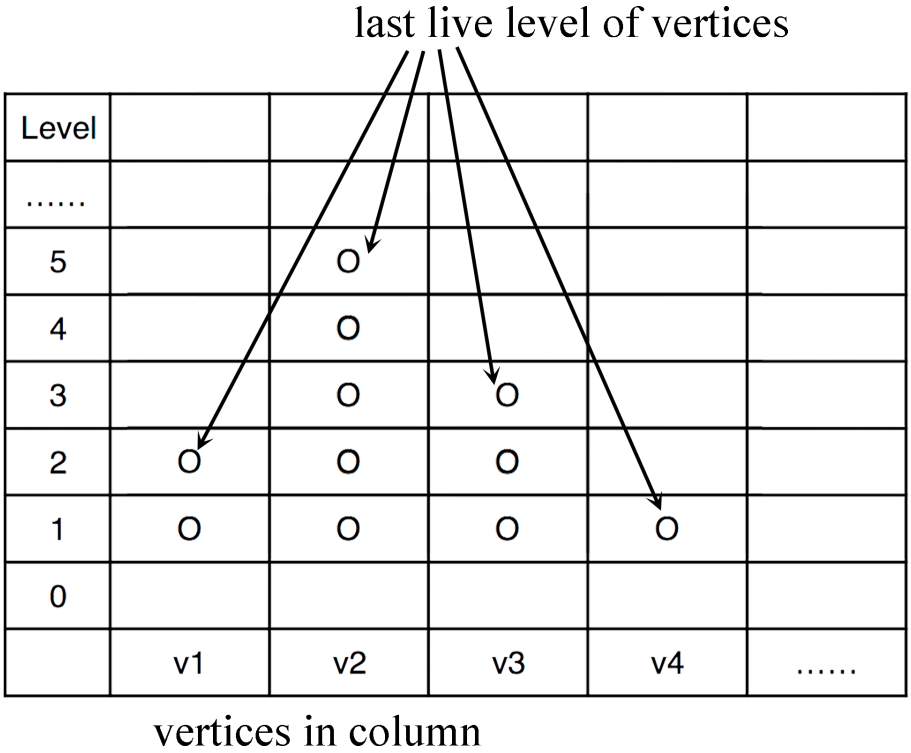

Рисунок 11 – Хранение данных с использованием $O(N)$ оперативной памяти

3.3. Проверка применимости модификаций

Для того, чтобы избежать ошибочных операций, сделанных на дереве, предлагается интерфейс для проверки ошибок динамической связи. Как правило, операции, выполняемые на динамическом дереве с двумя вершинами, требуют проверки их отношений. Операция проверки связи и проверки соединения между двумя вершинами проверяют, связаны ли вершины между собой или не связаны, но находятся в одном дереве. Проверка занимает $O(\log n)$ времени каждый раз, когда вызывается операция обновления, однако она не всегда необходима. Например, при операции отсоединения достаточно проверить только, является ли родительская вершина корнем. В операции разрезания необходимо проверить степень первой вершины, так как операция может быть выполнена только если вторая вершина является ее ребенком. Таким образом, проверка ошибок во время операции отсоединения занимает от $O(\log n)$ до $O(\log d)$, где d является степенью родителя.

3.4. Применение обновлений

Как уже упоминалось выше, уровни используются для обозначения модифицированных вершин на каждом раунде операций. Когда вызывается операция, последние использованные вершины вершины размещаются на нулевом и первом уровнях. Затем выполняется цикл, чтобы применить операции на каждом уровне. В соответствии с числом модифицированных вершин, вызывается функция обработки вершины. Функция проверяет состояние вершины, если текущая вершина является последней активной вершиной в его колонке, то функция будет выполнять операции Rake, Compress или другие изменения непосредственно. Если текущая вершина не активна, происходит рекурсивный вызов функции и текущая вершина перемещается на следующий уровень. Иными словами, функция фактически обновляет динамическое дерево. После обработки вершин одного уровня, необходимо проверить число следующих затрагиваемых вершин, чтобы обновить числа следующих затрагиваемых вершин текущей вершины на следующем уровне.

Перед тем, как внести изменения, необходимо проверить, применима ли операция к вершине. Функция `will_*` проверяет, станет ли вершина корнем, или к ней будут применены операции Rake или compress, или вершина будет изменена, но может участвовать в Rake или Compress на следующем уровне. Если результат проверки положительный, то функция `do_*` выпол-

няет необходимые операции. Если вершина подвергается операции Rake, это влияет на ее родителей, и число ее следующих затрагиваемых вершин должно быть увеличено на единицу. Если к вершине применяется операция compress, эта операция влияет как на ее родителей, так и на детей. Однако поскольку фактически используется декартово дерево, будет затронут только первый ребенок. В остальных случаях вызывается обработка вершин на следующем уровне.

3.5. Реализация параллельных вычислений

Как уже упоминалось выше, параллельное вычисление сделано в цикле с помощью вызова функции обработки вершины из расписания. Функция обработки обходит уровни для проверки всех измененных или посещенных вершин. Параллельные вычисления будут работать на каждом уровне для каждой вершины. В результате в новом алгоритме, разработанном в данной работе, на каждом уровне операции Rake, compress, turn to root, а также другие необходимые операции, выполняются параллельно, когда они находятся на том же уровне. Каждое изменение на определенном уровне будет влиять на следующий уровень, поэтому лучше не выполнять операции на разных уровнях параллельно.

Выводы по главе 3

В главе описана разработанная структура данных. Представлен механизм снятия ограничения на степень вершины, описаны принципы хранения данных с использованием $O(n)$ оперативной памяти. Разработана процедура проверки применимости модификаций. Описано, как происходит применение обновлений. Приводится описание реализации параллельных вычислений.

ГЛАВА 4. ЭКСПЕРИМЕНТАЛЬНОЕ ИССЛЕДОВАНИЕ

Ранее была описана разработка алгоритма, в этом разделе рассматриваются эксперименты для оценки его эффективности. Время работы последовательной реализации оценивается как базовая величина для сравнения со временем работы параллельной реализации. Для параллельной обработки данных используются две библиотеки, PASL и OpenMP. Они также сравниваются друг с другом. Эксперименты запускаются на компьютере с 4 ядрами процессора и 4 Гб памяти.

4.1. Тестовые сценарии

Эксперименты по измерению времени построения разработанной реализации и выполнения ею запросов производятся на следующих тестах:

а) дерево-«палка»: n вершин, при этом i-тая вершина является ребенком $(i-1)$-ой вершины при $i > 0$.

б) дерево-«пучок»: n вершин, при этом i-тая вершина является ребенком вершины 0 при $i > 0$.

в) дерево-«два пучка»: n вершин при $n = 2m$, m целое, при этом вершина 0 является родителем вершин с 1 по $(m-1)$, вершина m является родителем вершин с $(m+1)$ по $(n-1)$, а вершины 0 и m дополнительно соединены.

г) построение дерева-«палки» в десять этапов, каждый из которых состоит в присоединении $n/10$ вершин.

4.2. Экспериментальное исследование последовательной реализации

Выполнено четыре теста:

— построение n вершин линейного дерева — тест 1 «длинная цепь»;

— построение n вершин дерева-«пучка» — тест 2 «большая степень»;

— построение двух деревьев-«пучков» с n вершинами — тест 3 «две большие степени»;

— построение дерева-«палки» с десятью уровнями, в каждом из которых соединены $n/10$ вершин — тест 4 «инкрементальная длинная цепь». цепью.

Рассматриваются различные значения числа вершин. Каждый раз после построения дерева из n вершин выполняется n запросов к поддеревьям и n запросов к путям. В таблице приведено время работы структуры данных на тестах. Средняя сложность алгоритма составляет $O(\log n)$.

Таблица 1 – Результаты экспериментов для последовательной реализации

	$n = 10000$	$n = 100000$	$n = 1000000$	$n = 3000000$
1	0.063533	0.936087	9.8946	31.3906
2	0.031963	0.473844	5.04014	16.8301
3	0.03333	0.495401	5.19158	16.8656
4	0.068615	0.840358	11.1828	36.3289

4.3. Экспериментальное исследование реализации с использованием OpenMP

Оценим время работы параллельной реализации с применением OpenMP. Используются те же тесты, что и для последовательной реализации. Тем не менее, OpenMP может использовать процессоры с различным числом ядер. Проводятся тесты, проверяющие эффективность OpenMP на числе процессоров 1, 2, 4 и 8.

Параллельный метод с использованием OpenMP применяется в зацикливающей функции. Перед циклом добавляется строка `#pragma omp parallel for schedule (guided, 100)`. Для функции compute prefix sum сначала параллельно вычисляются предварительные значения; затем производится проверка, достигнут ли конец цикла и начинается ли последовательно следующий цикл, чтобы убедиться, что можно использовать предварительно посчитанные значения. Проверка выполняется в одном потоке. В таблицах 2–5 показано время работы, полученное с использованием OpenMP.

Таблица 2 – Результаты экспериментов для OpenMP, один процессор

	$n = 10000$	$n = 100000$	$n = 1000000$	$n = 3000000$
1	0.07072	1.01285	11.4329	35.1043
2	0.033578	0.48801	5.46487	17.5146
3	0.035731	0.515872	5.68942	18.0354
4	0.075208	0.891592	12.389	40.6775

Таблица 3 – Результаты экспериментов для OpenMP, два процессора

	$n = 10000$	$n = 100000$	$n = 1000000$	$n = 3000000$
1	0.081845	1.10473	11.6696	35.7506
2	0.04275	0.510112	5.47687	18.0637
3	0.044991	0.529194	5.59677	17.8966
4	0.11298	1.00143	12.9886	41.9446

Таблица 4 – Результаты экспериментов для OpenMP, четыре процессора

	$n = 10000$	$n = 100000$	$n = 1000000$	$n = 3000000$
1	0.089218	1.22818	13.169	40.9343
2	0.07859	0.553611	5.78005	18.871
3	0.047683	0.583647	5.94402	18.8319
4	0.159757	1.12068	14.4973	47.2622

Таблица 5 – Результаты экспериментов для OpenMP, восемь процессоров

	$n = 10000$	$n = 100000$	$n = 1000000$	$n = 3000000$
1	0.103436	1.61389	17.3962	54.0412
2	0.067749	0.638183	6.74281	21.2547
3	0.072668	0.638638	7.00279	21.7108
4	0.340943	1.39135	17.9712	58.4357

4.4. Экспериментальное исследование реализации с использованием PASL

Эксперименты проводятся аналогично экспериментам для OpenMP, однако в случае PASL тестируется также возможность использования процессоров с различным числом ядер. Параллельный метод с использованием PASL применяется в цикле. Используется инструкция `pasl::sched::native::parallel_for`. Для функции compute prefix sum сначала параллельно вычисляются предварительные значения; затем производится проверка, достигнут ли конец цикла и начинается ли последовательно следующий цикл, чтобы убедиться, что можно использовать предварительно посчитанные значения. В таблицах 6–9 показано время работы, полученное с использованием PASL с различным числом процессоров.

Таблица 6 – Результаты экспериментов для PASL, один процессор

	$n = 10000$	$n = 100000$	$n = 1000000$	$n = 3000000$
1	0.073684	1.09323	12.3956	38.1296
2	0.036832	0.522955	5.75152	18.5603
3	0.038104	0.548474	6.03994	19.1851
4	0.078299	0.953348	13.0366	42.883

Таблица 7 – Результаты экспериментов для PASL, два процессора

	$n = 10000$	$n = 100000$	$n = 1000000$	$n = 3000000$
1	0.141362	1.61106	15.5824	47.6541
2	0.077958	0.859135	8.91695	28.6172
3	0.078529	0.88774	8.77827	29.134
4	0.155524	1.90973	18.7348	58.2101

Таблица 8 – Результаты экспериментов для PASL, четыре процессора

	$n = 10000$	$n = 100000$	$n = 1000000$	$n = 3000000$
1	0.28281	2.71137	22.9835	69.6526
2	0.149832	1.5798	14.5509	49.3806
3	0.162006	1.61922	14.9153	48.653
4	0.313789	3.73799	31.7739	93.2229

Таблица 9 – Результаты экспериментов для PASL, восемь процессоров

	$n = 10000$	$n = 100000$	$n = 1000000$	$n = 3000000$
1	0.563715	5.00583	39.7913	121.166
2	0.306949	2.98072	27.0531	89.4415
3	0.322895	3.15966	28.327	87.608
4	0.644581	7.4977	57.6276	167.604

4.5. Сравнение результатов экспериментов

Эксперимент с последовательной реализацией выполняется в одном потоке. На рисунках 12–14 показано сравнение времени работы последовательной реализации, OpenMP и PASL.

На рисунке 12 высота столбца соответствует времени работы алгоритма. Можно видеть, что последовательная реализация в случае использования одного процессора является наилучшей. Кроме того, можно видеть, что на тесте «длинная цепь» OpenMP занимает меньше времени. В целом, в случае использования одного процессора PASL может занять много времени.

На рисунках 13–15 представлены результаты в случае использования нескольких процессоров. Можно видеть, что в этом случае OpenMP эффективнее, чем PASL.

Выводы по главе 4

В главе описано экспериментальное исследование эффективности предложенной структуры данных, поддерживающей инкрементальные изменения и

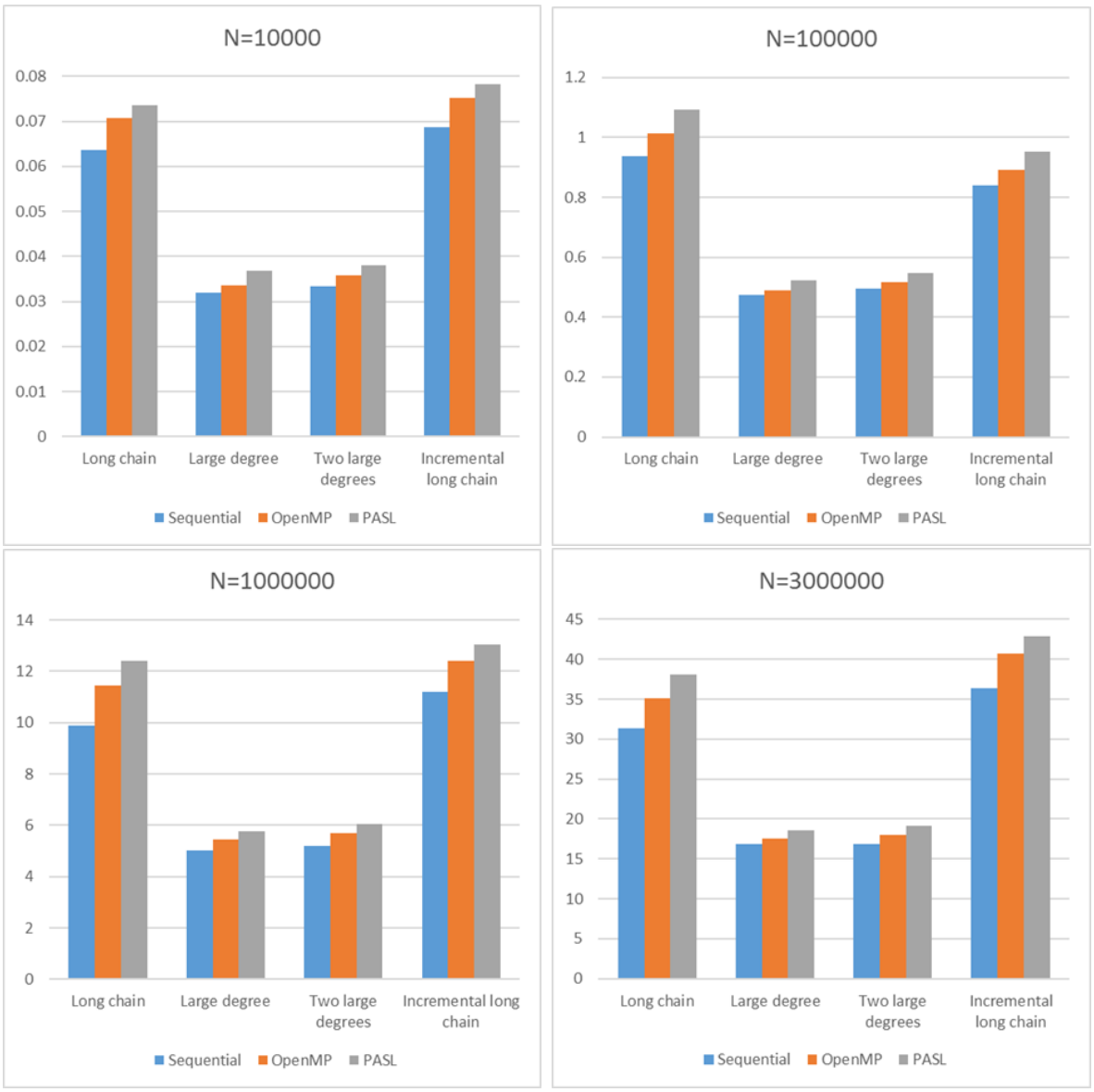

Рисунок 12 – Сравнение результатов, один процессор

предназначенной для параллельной обработки деревьев. Приводятся использованные тестовые сценарии. Выполняется экспериментальное исследование последовательной реализации, а также параллельных реализаций с использованием OpenMP и PASL. Библиотека PASL позволяет использовать различное число ядер процессора, однако демонстрирует низкую производительность в случае использования одного процессора. В большинстве рассмотренных случаев наиболее эффективной является реализация на основе OpenMP.

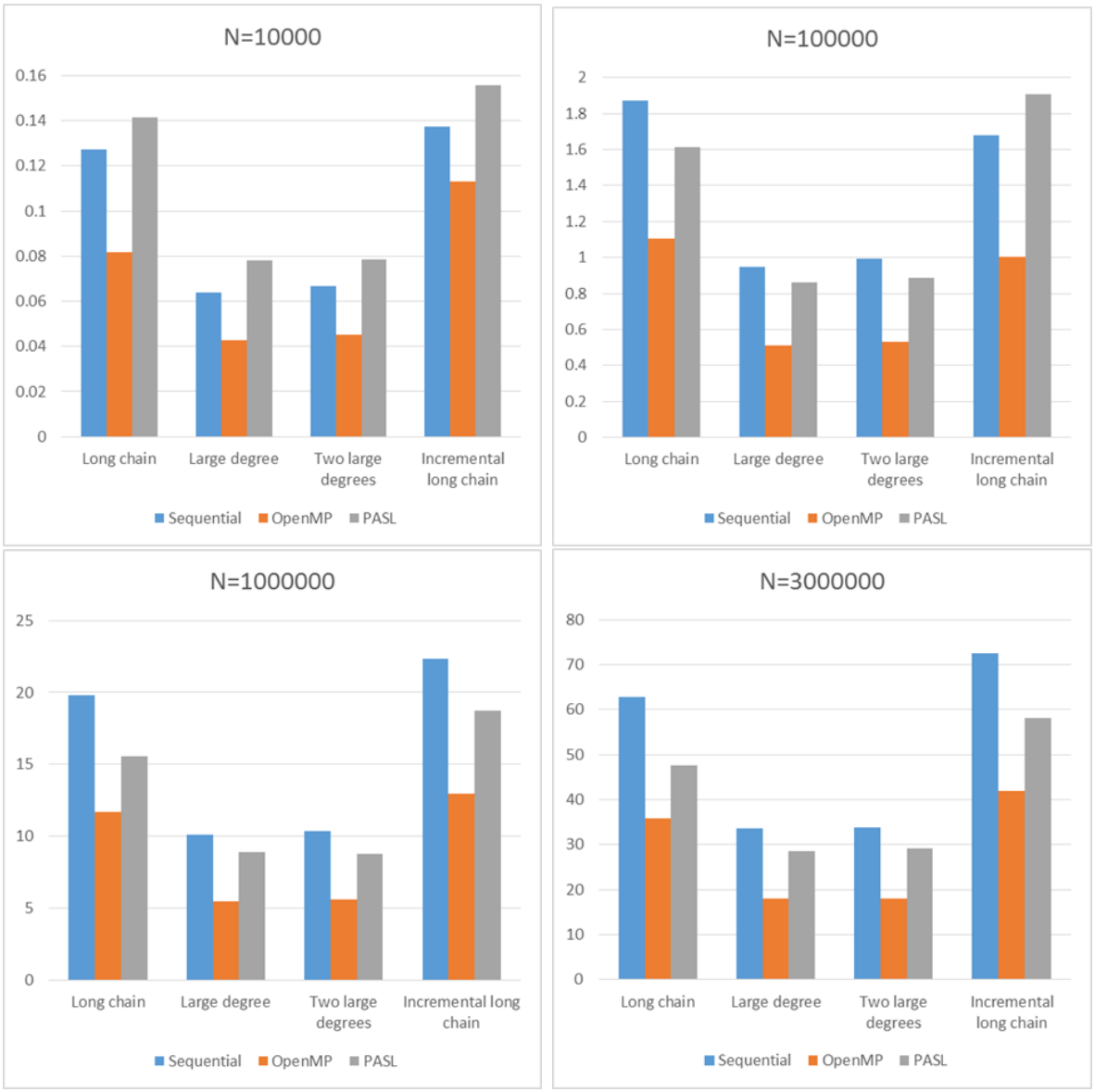

Рисунок 13 – Сравнение результатов, два процессора

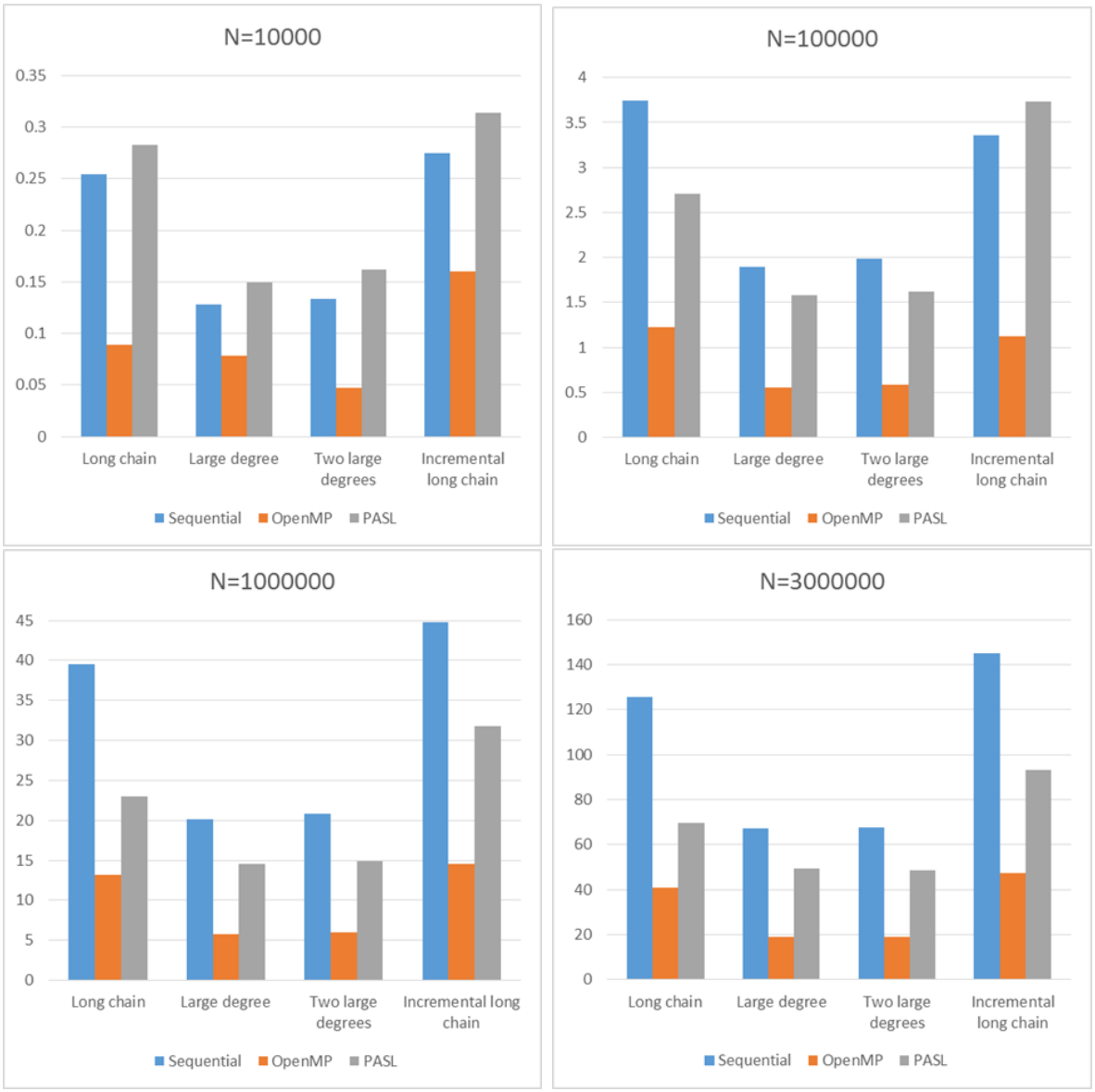

Рисунок 14 – Сравнение результатов, четыре процессора

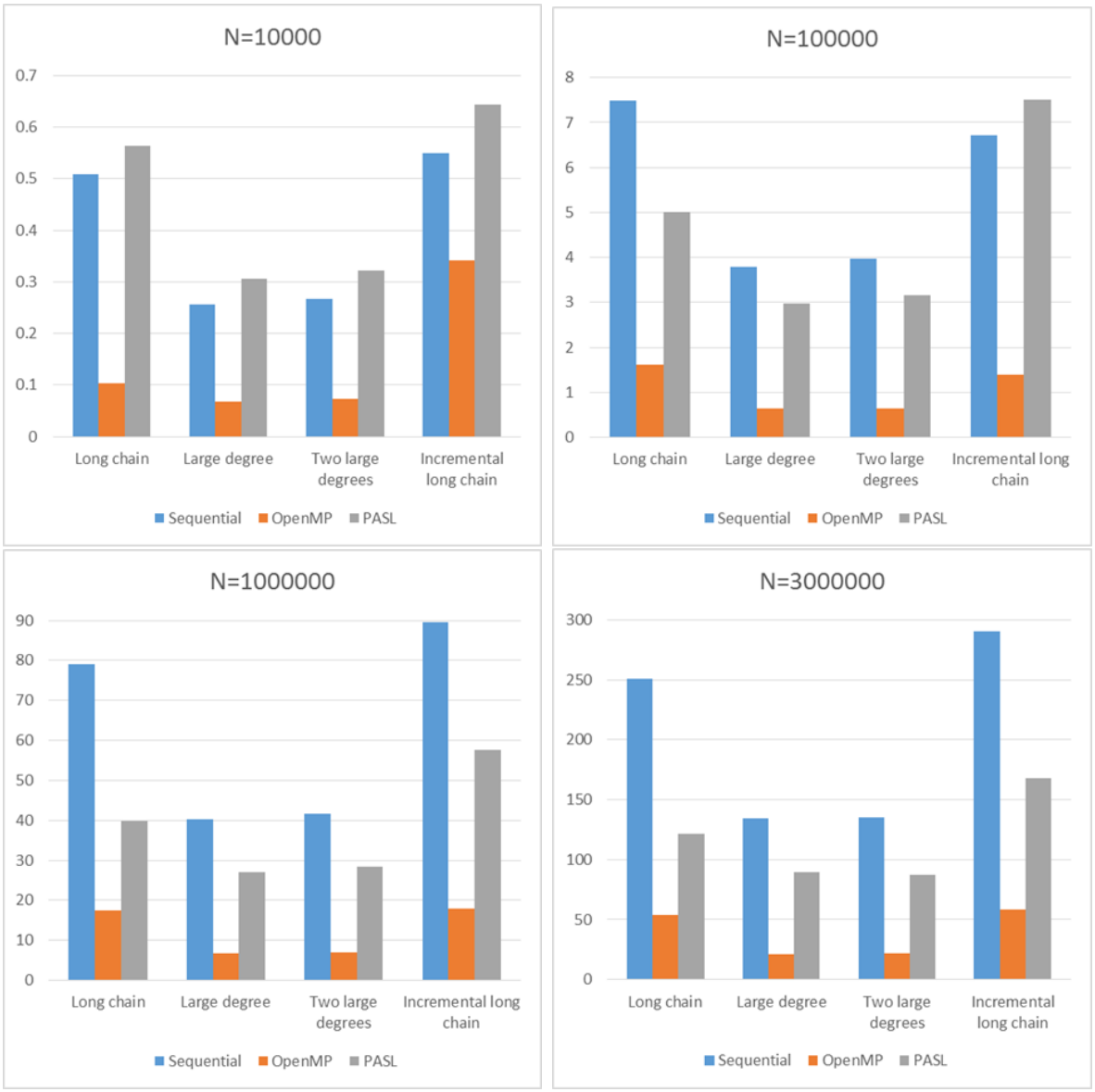

Рисунок 15 – Сравнение результатов, восемь процессоров

ЗАКЛЮЧЕНИЕ

В диссертации предложена структура данных, поддерживающая инкрементальные изменения и предназначенная для параллельной обработки деревьев. Предложенная структура основана на дереве Rake and Compress, а также декартовых деревьях и параллельном алгоритме вычисления префиксных сумм. Реализовано хранение данных с использованием $O(n)$ оперативной памяти, разработана процедура проверки применимости модификаций, поддерживается возможность выполнения параллельных вычислений.

Эффективность предложенной структуры данных проверяется экспериментально на примере четырех тестовых конфигураций. Производится сравнение с последовательной реализацией, а также исследуется эффективность реализаций на основе PASL и OpenMP. Результаты экспериментов подтверждают эффективность предложенной структуры данных в случае использования OpenMP.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 *Goldberg A. V.*, *Tarjan R. E.* A new approach to the maximum-flow problem // Journal of the ACM. — 1988. — Vol. 35, no. 4. — P. 921–940.

2 *Sleator D. D.*, *Tarjan R. E.* A data structure for dynamic trees // Journal of Computer and System Sciences. — 1983. — Vol. 26, no. 3. — P. 362–391.

3 *Miller G. L.*, *Reif J. H.* Parallel tree contraction and its application // Proceedings of the 26th Annual Symposium on Foundations of Computer Science. — 1985. — P. 478–489.

4 *Miller G. L.*, *Reif J. H.* Parallel Tree Contraction Part 1: Fundamentals // Randomness and Computation. Vol. 5. — JAI Press, 1989. — P. 47–72.

5 *Miller G. L.*, *Reif J. H.* Parallel Tree Contraction Part 2: Further Applications // SIAM Journal on Computing. — 1991. — Vol. 20, no. 6. — P. 1128–1147.

6 *Reif J.*, *Tate S.* Dynamic Parallel Tree Contraction // Proceedings of SPAA. — 1994. — P. 114–121.

7 *Frederickson G. N.* A data structure for dynamically maintaining rooted trees // Journal of Algorithms. — 1997. — Vol. 24, no. 1. — P. 37–65.

8 Minimizing diameters of dynamic trees / S. Alstrup [et al.] // Automata, Languages and Programming. — 2005. — P. 270–280. — (Lecture Notes in Computer Science ; 1256).

9 *Acar U. A.*, *Blelloch G. E.*, *Vittes J. L.* Separating structure from data in dynamic trees: tech. rep. / Department of Computer Science, Carnegie Mellon University. — 2003. — CMU-CS-03-189.

10 Dynamizing Static Algorithms with Applications to Dynamic Trees and History Independence / U. Acar [et al.] // Proceedings of SODA. — 2004. — P. 531–540.

11 *Acar U.*, *Blelloch G.*, *Vittes J.* An Experimental Analysis of Change Propagation in Dynamic Trees // Workshop on Algorithms Engineering and Experiments. — 2005.

12 *Morihata A.*, *Matsuzaki K.* A parallel tree contraction algorithm on non-binary trees: tech. rep. / Department of Mathematical Informatics, University of Tokyo. — 2008. — METR 2008-27.

13 *Morihata A.*, *Matsuzaki K.* A practical tree contraction algorithm for parallel skeletons on trees of unbounded degree //. — P. 7–16. — (Procedia Computer Science ; 4).

14 *Morihata A.*, *Matsuzaki K.* Parallel Tree Contraction with Fewer Types of Primitive Contraction Operations and Its Application to Trees of Unbounded Degree // IPSJ Online Transactions. — 2014. — Vol. 7. — P. 148–156.

15 *Vuillemin J.* A Unifying Look at Data Structures // Communications of ACM. — 1980. — Vol. 23, no. 4. — P. 229–239.

16 *Sengupta S.*, *Lefohn A. E.*, *Owens J. D.* A work-efficient step-efficient prefix sum algorithm // Workshop on edge computing using new commodity architectures. — 2006. — P. 26–27.

17 *Hillis W. D.*, *Steele Jr G. L.* Data parallel algorithms // Communications of the ACM. — 1986. — Vol. 29, no. 12. — P. 1170–1183.