Санкт-Петербургский государственный электротехнический университет
“ЛЭТИ” им. В. И. Ульянова (Ленина)
(СПбГЭТУ “ЛЭТИ”)

Направление подготовки: 09.04.01 “Информатика и вычислительная техника”
Магистерская программа: “Программное обеспечение информационных и вычислительных систем”

Факультет компьютерных технологий и информатики
Кафедра вычислительной техники

К защите допустить:
Заведующий кафедрой
д. т. н., профессор _____ М. С. Куприянов

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА МАГИСТРА

Тема: “Средства оптимизации программ на основе программной предвыборки данных”

Студент _____ Т. В. Кадыров

Руководитель
к. т. н. _____ А. А. Пазников

Консультант по защите интеллектуальной собственности
к. э. н., доцент _____ М. Н. Магомедов

Консультант от кафедры _____ М. Н. Гречухин

Санкт-Петербург
2023 г.

Санкт-Петербургский государственный электротехнический университет "ЛЭТИ" им. В. И. Ульянова (Ленина)
(СПбГЭТУ "ЛЭТИ")

Направление 09.04.01 «Информатика и вычислительная техника»
Программа «Программное обеспечение информационных и вычислительных систем»
Факультет компьютерных технологий и информатики
Кафедра вычислительной техники

УТВЕРЖДАЮ
Заведующий кафедрой ВТ
д. т. н., профессор
(М. С. Куприянов)
"___" _____ 202__г.

ЗАДАНИЕ
на выпускную квалификационную работу

Студент Кадыров Тимур Валерьевич Группа № 7308

1. Тема: Средства оптимизации программ на основе программной предвыборки данных

Место выполнения ВКР: Санкт-Петербургский государственный электротехнический университет «ЛЭТИ» им. В. И. Ульянова (Ленина)

2. Объект и предмет исследования: Оптимизация программ. Оптимизация программ с помощью предвыборки данных в кэш.
3. Цель: Исследование средств оптимизации программ на основе программной предвыборки данных.
4. Исходные данные: Алгоритмы предвыборки данных в кэш, средства для компиляторной оптимизации.
5. Содержание: Описание методов предвыборки данных, описание структуры LLVM, описание алгоритмов предвыборки данных, реализация и тестирование алгоритма предвыборки при помощи LLVM.
6. Технические требования: Программа должна работать под версией библиотеки LLVM 15.
7. Дополнительные разделы: Оценка и защита результатов интеллектуальной деятельности
8. Результаты: В результате работы нужно проанализировать различные алгоритмы программной предвыборки данных, а также реализовать и

протестировать один из этих алгоритмов. Отчетные материалы: пояснительная записка, аннотация, реферат, презентация.

Дата выдачи задания «_____» _____ 202__ г.

Дата представления ВКР к защите «_____» _____ 202__ г.

Руководитель
к. т. н. _____ А. А. Пазников

Студент _____ Т. В. Кадыров

Санкт-Петербургский государственный электротехнический университет
“ЛЭТИ” им. В. И. Ульянова (Ленина)
(СПбГЭТУ “ЛЭТИ”)

Направление 09.04.01 «Информатика и вычислительная техника»
Программа «Программное обеспечение информационных и вычислительных систем»
Факультет компьютерных технологий и информатики
Кафедра вычислительной техники

УТВЕРЖДАЮ
Заведующий кафедрой ВТ
д. т. н., профессор
(М. С. Куприянов)
“___” _____ 202__г.

КАЛЕНДАРНЫЙ ПЛАН
выполнения выпускной квалификационной работы

Тема Средства оптимизации программ на основе программной предвыборки данных

Студент Кадыров Тимур Валерьевич Группа № 7308

№ этапа	Наименование работ	Срок выполнения
1	Обзор литературы по теме работы	06.02 – 28.02
2	Изучение методов предвыборки данных	01.03 – 15.03
3	Изучение структуры LLVM	16.03 – 30.03
4	Изучение алгоритмов предвыборки данных	31.03 – 15.04
5	Реализация и тестирование алгоритма предвыборки данных	16.04 – 01.05
6	Оформление пояснительной записки	02.05 – 15.05
7	Предварительное рассмотрение работы	16.05
8	Представление работы к защите	23.05

Руководитель
к. т. н. _____ А. А. Пазников
Студент _____ Т. В. Кадыров

РЕФЕРАТ

Пояснительная записка содержит: 51 с., 18 рис., 5 табл., 2 приложения, 20 источников литературы.

Цель работы: исследование средств оптимизации программ на основе программной предвыборки данных.

В выпускной квалификационной работе приводится описание структуры кэш-памяти и методов предвыборки команд и данных. Также в этой работе описывается общая структура LLVM и технические особенности LLVM IR, а также API, предоставляемого LLVM для написания проходов.

В результате работы произведен анализ алгоритмов предвыборки данных и реализован один из них с помощью LLVM.

Результат работы может быть использован в качестве прохода LLVM для оптимизации программ с использованием компилятора Clang и утилит, которые предоставляет проект LLVM.

ABSTRACT

One of the most important criteria for users choosing the product to use is the speed of the program. According to statistics, most Internet users do not want to wait more than 3-4 seconds for a webpage to load. Also, the results of Google's surveys show that half of the users get annoyed by the low speed of a web page. In order to speed up the program, there are several different ways.

The purpose of this work is to study program optimization tools based on software cache prefetching.

The work describes the structure of cache and different methods for prefetching. It also contains the information about LLVM infrastructure and features of LLVM IR. It also describes the API provided by LLVM to create Passes.

As a result of this work an analysis of prefetching algorithms was conducted and one of them was implemented using LLVM library.

The result can be used as a LLVM pass with clang compiler and LLVM tools to optimize programs.

СОДЕРЖАНИЕ

ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

Allocator (аллокатор) – специальная функция, которая выделяет память для переменных по определенному алгоритму

Benchmark (бенчмарк) – Программа, используемая для тестирования работы алгоритма.

Cache hit – Попадание кэша, происходит в случае, когда данные, к которым происходит обращение находятся в кэше.

Cache line – Блок данных определенного размера. Минимальная единица загрузки данных в кэш.

Cache miss – Промах кэша, происходит в случае, когда данные, к которым происходит обращение не находятся в кэше и нужно искать их в основной памяти

Intrinsic function – функция, которая используется определенным компилятором

LIFO – Last In First Out, хранение данных по типу стэк

LRU – Least Recently Used, алгоритм выборки, при котором выбирается объект, который не использовался дольше всех

Overhead (оверхед) – накладные расходы при выполнении определенной функции или алгоритма

Prefetch (предвыборка, предзагрузка, префетч) – Загрузка данных из памяти в кэш до того, как они будут использоваться

The LLVM Project – инфраструктура для создания компиляторов

ВВЕДЕНИЕ

Одним из самых важных критериев при выборе программного продукта для пользователей является скорость работы этой программы. Согласно статистике, большая часть пользователей сети Интернет не хотят ждать загрузки страницы более 3-4 секунд [9]. Также результаты опросов Google показывают, что половину пользователей раздражает низкая скорость работы веб-страницы [10]. Для того, чтобы ускорить работу программы существует несколько различных способов.

Первым способом является улучшение оборудования, добавление дополнительных серверов и т.п. То есть решения на уровне «железа». Однако во многих случаях это либо дорого, либо не приводит к желаемому результату.

Вторым способом является оптимизация на уровне алгоритма. То есть неэффективные алгоритмы программы следует заменить на более эффективные. Однако часто это является слишком сложной задачей, или алгоритм уже оптимизирован настолько, насколько это возможно.

Поэтому часто прибегают к третьему способу – оптимизациям, связанным со структурой памяти программы, особенностями выполнения инструкций процессоров и так далее. Одной из таких оптимизаций является предвыборка данных.

Предвыборка данных представляет собой загрузку данных в кэш до того, как они будут использоваться, чтобы они потом были загружены быстрее. Программист может сам указать, где осуществлять предвыборку, но чаще всего эту задачу целесообразнее передать компилятору.

Целью работы является анализ методов программной предвыборки данных в кэш. Объектом работы является оптимизация программ. Предмет – оптимизация программ с помощью предвыборки данных в кэш.

Задачи:

1. Анализ методов предвыборки данных
2. Анализ структуры LLVM

3. Тестирование и анализ алгоритмов предвыборки данных

В первом разделе приведено описание структуры кэш-памяти, анализ методов предвыборки данных и команд. Описание структуры LLVM и особенностей LLVM IR приведено во втором разделе. Третий раздел содержит описание алгоритмов предвыборки данных и реализации. В четвертом разделе приведена оценка и защита результата интеллектуальной деятельности (программы ЭВМ). Общие выводы по работе содержатся в заключении.

1 Описание методов предвыборки данных

В данном разделе приведено описание механизма предвыборки данных, различные виды предвыборки и способы применения.

1.1 Описание структуры кэш-памяти

Прежде чем переходить к описанию предвыборки данных, следует рассказать о том, как устроена кэш-память. В данном подразделе приводится краткое описание устройства кэш-памяти.

1.1.1 Иерархическая структура памяти

Память в современных вычислительных системах имеет иерархическую структуру, то есть состоит из нескольких уровней, которые отличаются по размеру и скорости загрузки и выгрузки данных [11]. Пример иерархической структуры памяти представлен на рисунке 1.1.

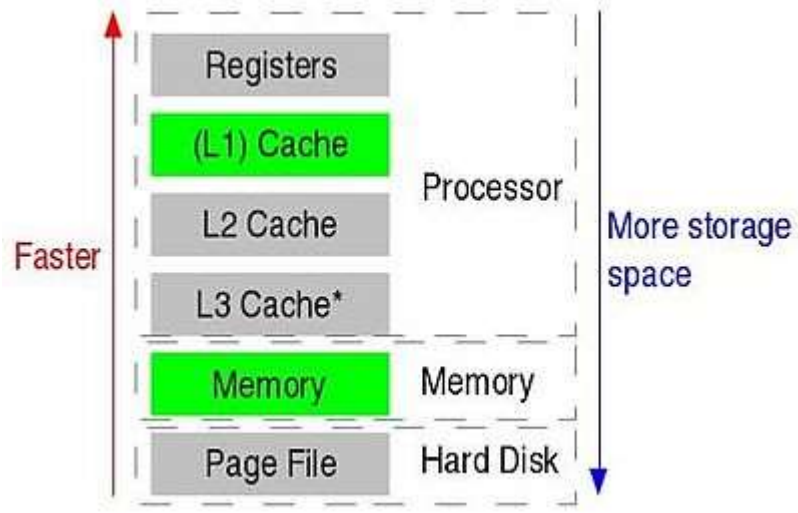

Рисунок 1.1 – Пример иерархии памяти

Обычно на самом высоком уровне находятся регистры процессора. К ним процессор имеет самый быстрый доступ, но при этом они являются самым маленьким по размеру видом памяти.

Кэш-память находится на следующем уровне иерархии после регистров. В ней дублируются определенные участки оперативной памяти, чтобы

ускорить доступ к ним. Обычно в современных процессорах кэш-память разделяется на 2-3 уровня. Самый маленький и быстрый L1-кэш, и более объемные, но медленные L2 и L3-кэши [12]. Стоит заметить, что кэш 2 и 3 уровня чаще всего разделяется между несколькими ядрами процессора.

Следующим уровнем после кэш-памяти является оперативная память, в которой хранятся основные данные, требуемые для работы программы. На самом низком уровне иерархии находится внешняя память для хранения долгосрочных данных.

1.1.2 Принципы работы кэш-памяти

Рассмотрим подробнее структуру кэш-памяти. Принцип работы кэш-памяти состоит в том, что при обращении к участку памяти (например, переменной) сначала проверяется наличие данных в кэше и если их нет, то происходит промах (cache miss) [12], а данные загружаются из оперативной памяти (либо из более высокого уровня кэша по такому же принципу). Данные в кэше обычно выделяются в виде блоков определенного размера, которые называются кэш-линиями (cache lines).

Для отображения данных из памяти в кэш используются три основных алгоритма: прямое отображение, полностью ассоциативное и наборно-ассоциативное отображение [13].

Принцип прямого отображения состоит в том, что основная память делится на блоки, размер которых равен размеру кэш-памяти, а каждый блок делится на кэш-линии. Соответственно кэш-линия в определенном блоке всегда отображается в соответственную кэш-линию в кэш-памяти. В кэш-памяти же дополнительно хранится тег (tag), который указывает на номер блока, данные из которого сейчас хранятся в кэш-линии. Основной минус состоит в том, что каждый из участков памяти жестко привязан к определённой кэш-линии. Пример прямого отображения представлен на рисунке 1.2.

В полностью ассоциативном кэше участок памяти может быть отображен в любую кэш-линию. Основной минус состоит в сложной аппаратной реализации проверки тега.

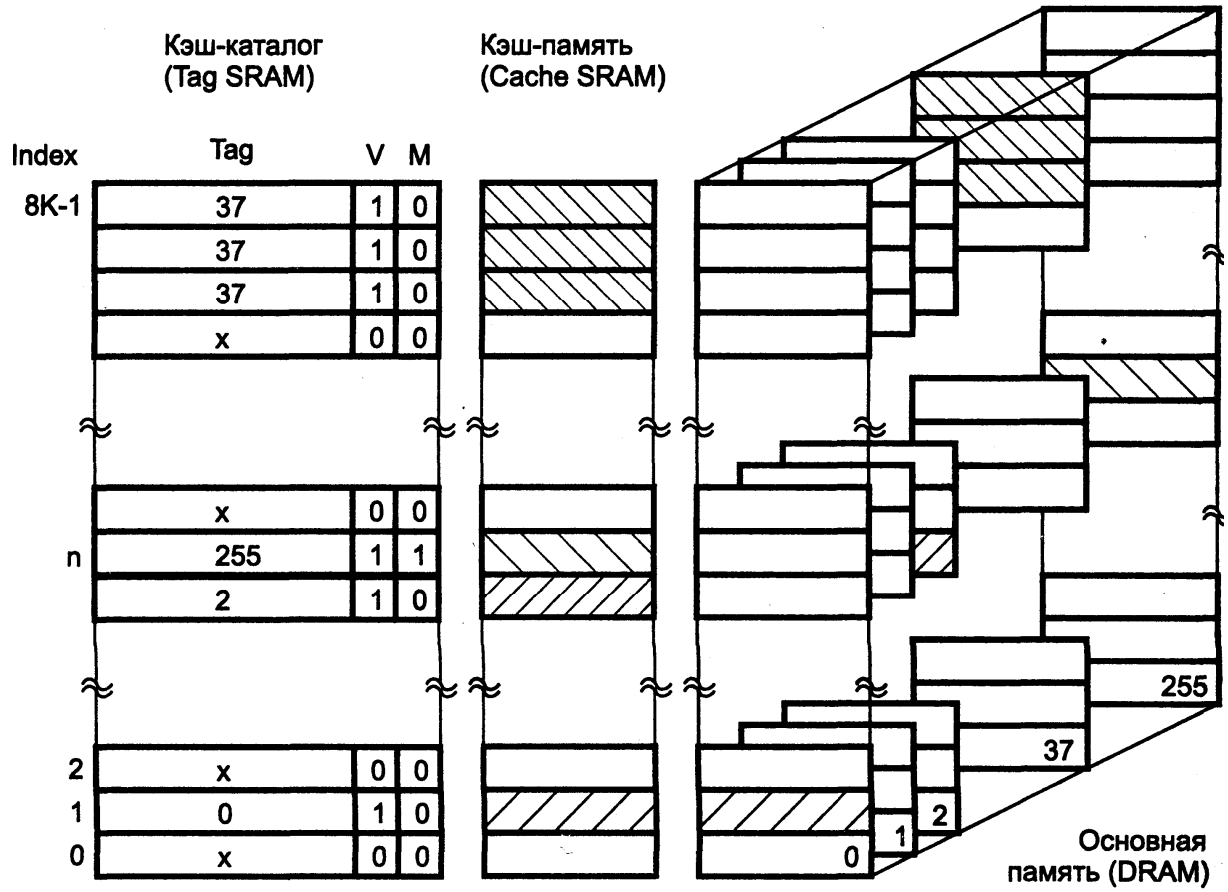

Рисунок 1.2 – Кэш прямого отображения

В наборно-ассоциативном кэше он делится на несколько банков (каналов), в каждом из которых выделяется набор кэш-линий. Соответственно каждый из этих банков действует по принципу прямого отображения, но теперь у каждой кэш-линии в основной памяти есть возможность «попасть» в несколько разных кэш-линий (число их соответственно равно количеству каналов). В таком кэше также требуется реализовать алгоритм замещения, обычно это LRU или LIFO. Чаще всего встречаются 4 и 8-канальные кэши. На рисунке 1.3 представлен пример наборно-ассоциативного отображения.

Из этих трех видов отображения наиболее применимым является наборно-ассоциативное. Оно сочетает себе плюсы прямого и полностью ассоциативного отображения.

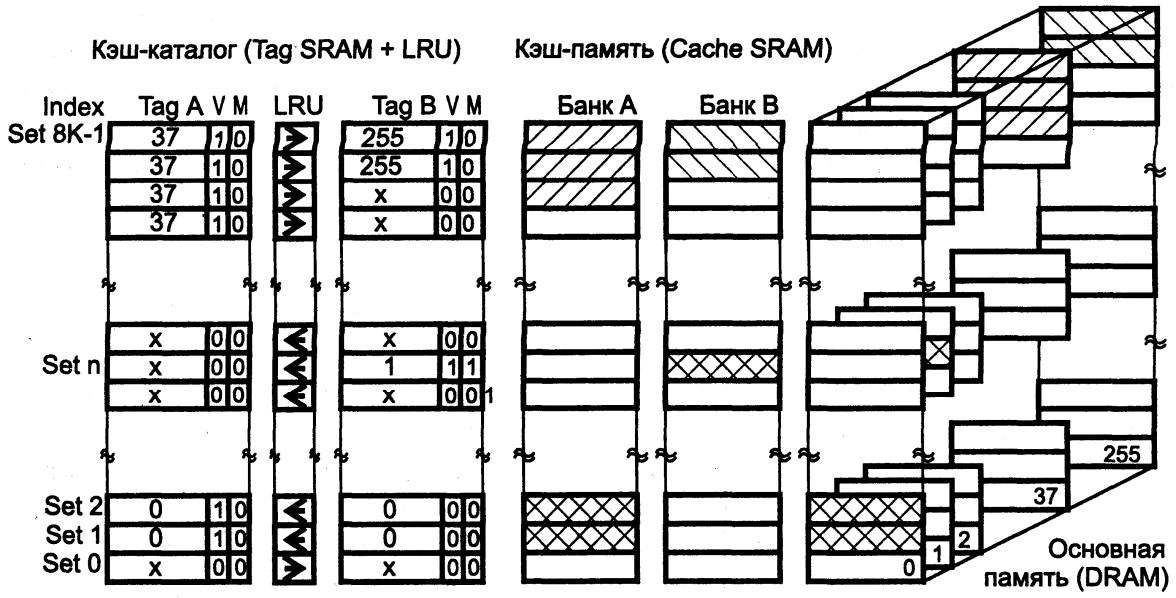

Рисунок 1.3 – Наборно-ассоциативный кэш

Далее разберем методы предвыборки данных и инструкций в кэш.

1.2 Предвыборка данных и команд в кэш

В данном подразделе приводится общее описание предвыборки команд и данных, а также описываются различные методы аппаратной и программной предвыборки.

1.2.1 Общие принципы предвыборки данных и кода в кэш

Как указывалось ранее предвыборка – это загрузка данных (инструкций) из основной памяти кэш заблаговременно (до того, как они будут использоваться) [14]. Это позволяет избежать промаха кэша при обращении к этим данным и тем самым оптимизирует работу программы. Иллюстрация данного эффекта представлена на рисунке 1.4.

В первую очередь предвыборку делят на аппаратную и программную. При аппаратной предвыборке процессор с помощью своих собственных алгоритмов осуществляет загрузку данных в кэш. Обычно выделяют пространственные и временные алгоритмы предвыборки. Пример процессора с блоком предвыборки представлен на рисунке 1.5.

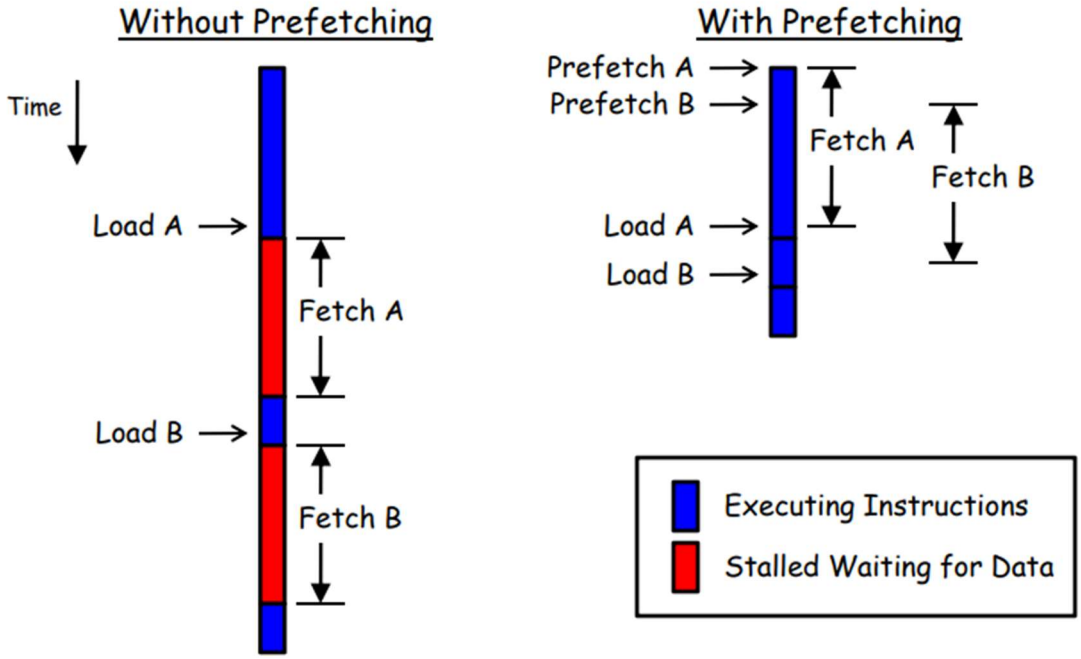

Рисунок 1.4. – Принцип оптимизации при предвыборке данных

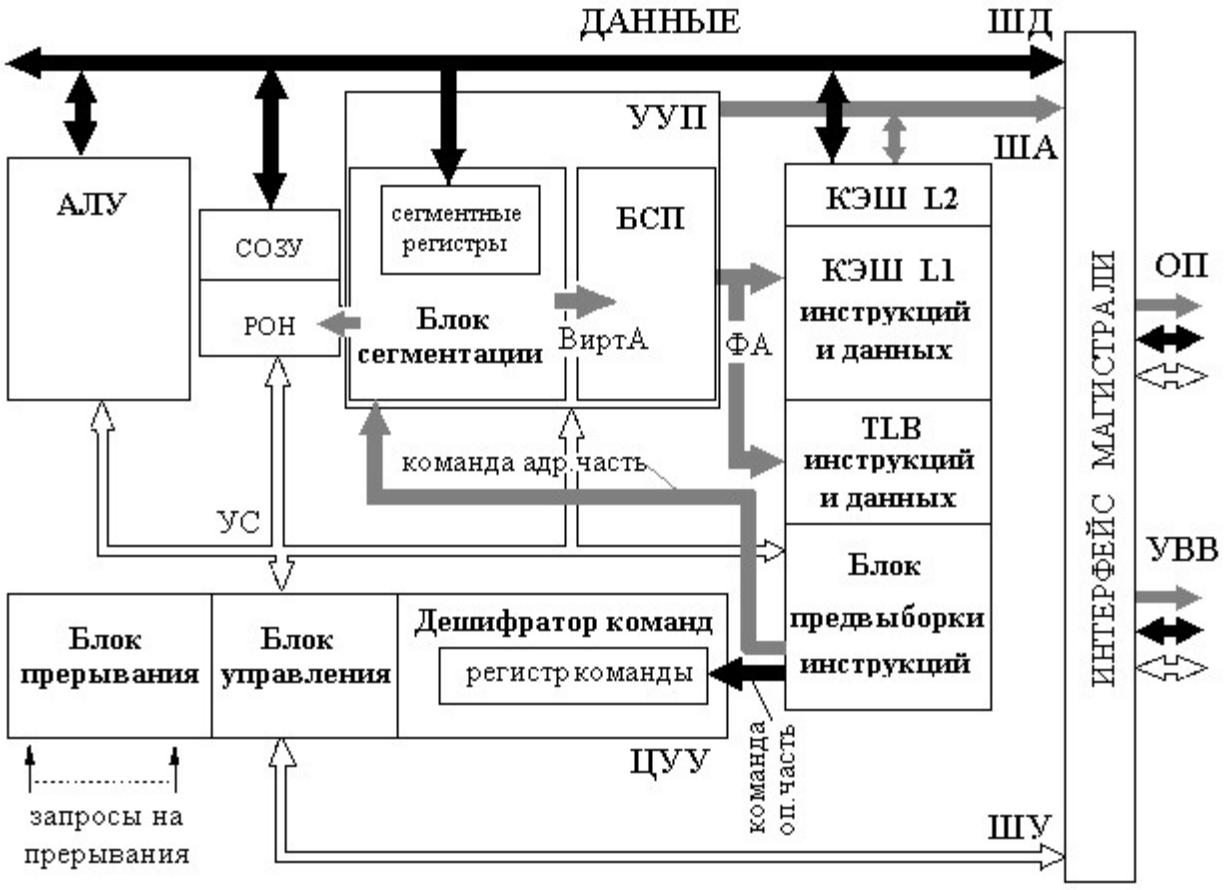

Рисунок 1.5 – Структура процессора с блоком предвыборки

При программной предвыборке данных процессор осуществляет загрузку данных в кэш в соответствии с командами управления

кэшированием. В программу явно вставляются команды предвыборки при написании программы или при ее компиляции.

Также предвыборку обычно подразделяют на предвыборку данных и предвыборку инструкций. При предвыборке инструкций процессор предугадывает, какая инструкция будет выполняться дальше и заранее загружает ее в кэш (стоит заметить, что часто у данных и инструкций бывает отдельный кэш). Для предсказания инструкций существует множество алгоритмов и чаще всего они уже реализованы на уровне процессора (т.е. обычно это аппаратная предвыборка) [14].

Предвыборка данных считается более сложной, так как паттерны обращения к данным труднее определить и предсказать, чем паттерны выполнения инструкций. Из-за этого чаще ее выполняют на программном уровне (кроме легко предсказываемых паттернов в циклах).

1.2.2 Методы аппаратной предвыборки

Один из самых часто встречающихся методов аппаратной предвыборки – потоковый буфер (Stream Buffer) [15]. Пример представлен на рисунке 1.6.

Принцип его действия очень прост. При возникновении промаха кэша, несколько последующих адресов (зависит от размера буфера) загружаются в потоковый буфер. Далее если процессор начинает читать адреса в этой последовательности, они загружаются из буфера. Тем самым получается избежать часть промахов кэша.

Этот метод обычно применяется для предвыборки инструкций, так как чаще всего они выполняются последовательно. В некоторых случаях он подходит и для предвыборки данных, если они также читаются последовательно.

Следующим методом является шаговая предвыборка. Данный метод основан на нахождении паттерна в шагах между доступами к памяти. Если шаг постоянный, то все довольно просто – достаточно рассчитать этот шаг и

совершать предвыборку с отступом на него. Но если шаг постоянно меняется, то найти паттерн в нем значительно сложнее [16]. Часть процессоров имеет алгоритмы для этого, а в иных случаях это можно сделать на программном уровне.

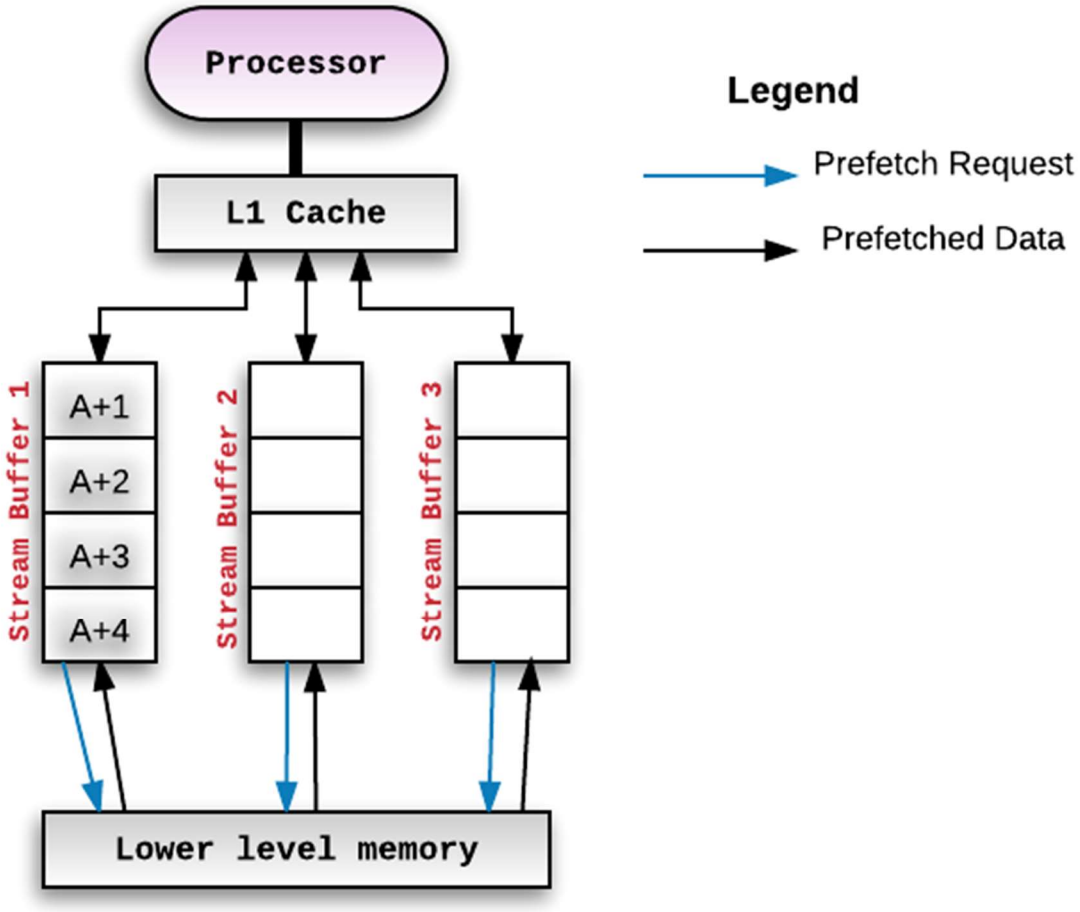

Рисунок 1.6 – Схема работы потокового буфера

Также часто используются временные методы предвыборки данных. Процессор определяет, что какая-то последовательность доступов к памяти часто повторяется и на основе этого паттерна осуществляет предвыборку.

1.2.3 Методы программной предвыборки

Первый метод программной предвыборки – самостоятельная вставка команд предвыборки разработчиком при написании программы. Обычно это осуществляется с помощью вставки директив компилятора в определенные

места программного кода. Например, в языках C/C++ существует встроенный объект компилятора GCC (intrinsic function) __builtin_prefetch [17]. При вставке этой директивы в код компилятор вставит команду предвыборки в сгенерированный им машинный код.

Второй метод программной предвыборки – предвыборка, осуществляемая компилятором. В этом случае компилятор сам (с помощью известных алгоритмов) определяет места в коде, куда можно вставить инструкции предвыборки. Этот способ наиболее удобен, так как во многих случаях программист не может выполнить анализ сложных паттернов самостоятельно. Также если код очень большой, вставка директив, указывающих компилятору выполнить prefetch может занять очень много времени, которое программист мог бы потратить на более полезные занятия.

1.3 Выводы по разделу

В данном разделе были описаны и проанализированы структура кэш-памяти, а также методы предвыборки данных в нее. В результате можно сделать следующие выводы:

1) Так как кэш работает куда быстрее основной памяти, избегание промахов позволяет ускорить работу программы.

2) Правильно выполненная предвыборка данных в кэш позволяет минимизировать количество промахов кэша.

3) Аппаратная предвыборка реализована на уровне процессоров – ее мы не рассматриваем.

4) Программная предвыборка может быть выполнена на компиляторном уровне и позволяет «отлавливать» более сложные паттерны без дополнительного оверхеда.

Таким образом, для реализации программной предвыборки данных нам следует изменять программу при ее компиляции. Для этого в следующем разделе будет рассмотрена структура LLVM.

2 Описание структуры LLVM

Данный раздел посвящен описанию структуры LLVM. В нем приведены общие сведения об LLVM, а также важные особенности LLVM IR и принципы создания LLVM проходов и плагинов Clang.

2.1 Общее описание LLVM

Проект LLVM представляет собой набор технологий для создания компиляторов и связанных с ними утилит [18]. Основным подпроектом в LLVM является ядро LLVM, которое предоставляет мультиплатформенную систему оптимизации кода и поддерживает генерацию кода для большинства популярных (и некоторых непопулярных) процессоров. Библиотеки LLVM построены на основе промежуточного представления кода, называемого LLVM IR. Ядро LLVM также можно использовать для создания собственного языка и/или компилятора. Схема инфраструктуры LLVM представлена на рисунке 2.1.

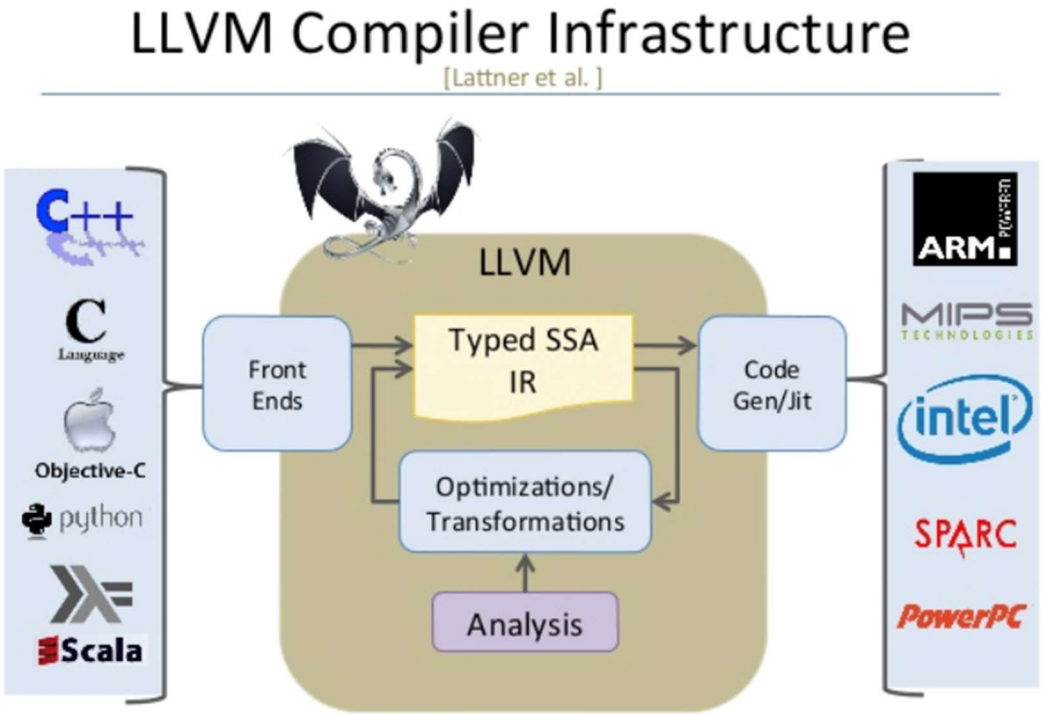

Рисунок 2.1 – Инфраструктура LLVM

Другим важным подпроектом LLVM является Clang – компилятор для языков C/C++/Objective C, который построен на основе ядра LLVM. Помимо высокой скорости компиляции, Clang предоставляет достаточно обширный API для разработки оптимизаций (в виде Clang плагинов) на уровне программного кода и множество инструментов для автоматического анализа кода (например, такие утилиты как clang-format, clang-tidy и другие) [18].

Помимо этих двух подпроектов в LLVM входит множество других полезных утилит – отладчик LLDB, компоновщик LLD, утилита оптимизации бинарного кода BOLT и другие.

2.2 Описание LLVM IR

LLVM IR – это промежуточное представление кода программы, используемое в компиляторе LLVM. Оно является абстрактным низкоуровневым языком, который позволяет описывать программу в виде последовательности инструкций, не зависящих от конкретной аппаратной архитектуры. Из него компилятор может сгенерировать машинный код для различных платформ (x86, ARM, PowerPC и т.п.) [18].

Преимуществом использования LLVM IR является то, что он обеспечивает высокую степень переносимости кода между различными платформами и аппаратными средствами. Кроме того, благодаря этому, LLVM IR позволяет создавать универсальные оптимизации кода для разных аппаратных платформ и разных языков программирования, которые могут компилироваться в LLVM IR

Программа на LLVM IR состоит из модулей. В каждом модуле содержится набор функций, глобальных переменных, типов и метаданных. Функции содержат множество базовых блоков, каждый из который состоит из набора инструкций. Иерархическая структура программы в LLVM IR представлена на рисунке 2.2.

Каждая инструкция LLVM IR выполняет определенную операцию, такую как арифметические вычисления, работу с памятью или передачу управления другому блоку кода.

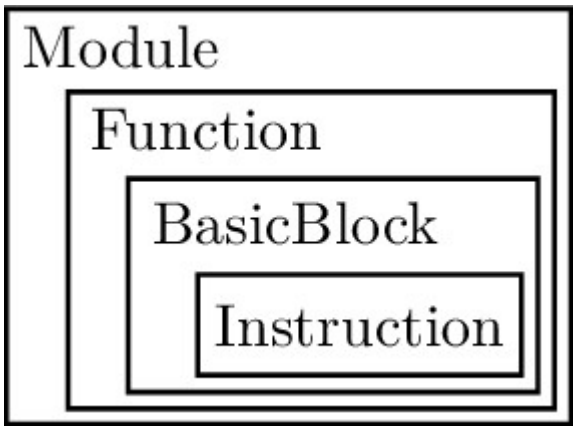

Рисунок 2.2 – Иерархия программы в LLVM IR

Обычно программа, скомпилированная в LLVM IR, представляется в виде текстового файла формата «ll», который содержит код IR или в виде биткода в формате «bc». Чтобы выполнять оптимизации над IR кодом, применяются LLVM проходы [18].

2.3 Описание LLVM проходов

LLVM проход (Pass) – это программный модуль, который принимает на вход LLVM IR и производит некоторое преобразование этого кода. LLVM проход может быть использован для выполнения широкого спектра задач, таких как оптимизация кода, анализ производительности, сбор статистики и многое другое. Каждый проход может выполнять одну или несколько задач.

Пользователи LLVM могут использовать готовые проходы из стандартной библиотеки LLVM или написать свой собственный, если проход с нужной функциональностью, не присутствует в LLVM [18].

Для применения LLVM прохода есть несколько способов. Первый из них – скомпилировать программу в формат LLVM IR с помощью специального аргумента компилятора (emit-llvm). После этого можно запустить проход на этом коде, используя инструменты LLVM (например,

opt), передав при этом аргумент, соответствующий этому проходу. Второй способ – это передать соответствующий аргумент в компилятор clang. Другой вариант – запустить этот проход внутри другого прохода программно. Например, это удобно, если для какой-нибудь трансформации требуется предварительно провести анализ.

В LLVM на данный момент есть два способа создания собственного прохода на языке C++. Первый из них (с Legacy PassManager) начинается с объявления нового подкласса llvm::Pass. Этот класс определяет тип прохода (например, ModulePass или FunctionPass) и содержит функции для обработки LLVM IR. При втором способе все проходы наследуются от CRTP mix-in класса PassInfoMixin<PassT>. При этом для определения типа прохода используется определенный аргумент в основной функции модуля run. Например, для функционала, аналогичного FunctionPass, требуется передать в нее аргумент типа Function [18].

Для того, чтобы использовать собственный проход в утилитах llvm (например, opt), требуется скомпилировать его в динамическую библиотеку (формата so) и загрузить при запуске соответствующей утилиты с помощью специального аргумента. При данном условии проход можно будет вызвать аналогично стандартным проходам LLVM. Созданный проход при соблюдении некоторых условий также может быть добавлен в стандартную библиотеку LLVM. Инструкции для этого есть на официальном сайте LLVM.

2.3 Описание Clang плагинов

Clang плагин – это программное расширение для компилятора Clang, которое позволяет пользователю модифицировать поведение компилятора и добавлять новые функциональные возможности. Они представляют собой динамические библиотеки, подключаемые во время компиляции [18].

Для создания Clang плагина на языке C++ используется базовый класс "PluginASTAction", который обеспечивает интеграцию плагина с

компилятором Clang. Обработка параметров осуществляется с помощью функции ParseArgs. Далее для обработки AST-дерева используется класс RecursiveAstVisitor.

Для использования Clang плагинов необходимо указать их при вызове компилятора Clang с помощью параметра «-load» и пути к файлу плагина. Вызвать его можно с помощью аргумента «-plugin PluginName» Кроме того, можно передать параметры плагину с помощью опций ключа «-plugin-arg-<plugin-name>». Например, чтобы запустить плагин «my_plugin.so» и передать ему параметр «param1», нужно выполнить следующую команду:

```
clang -Xclang -load -Xclang my_plugin.so -Xclang -plugin-arg-my_plugin -Xclang param1 file.c
```

В отличие от прохода LLVM плагин Clang анализирует код высокоуровневого языка программирования (в виде AST дерева) и позволяет изменять не только функционал, но и формат этого кода. Отсюда же вытекают и недостатки: привязанность к определенному языку, а также более сложная структура кода, чем на простом IR.

2.4 Выводы по разделу

В данном разделе был проведен анализ структуры проекта LLVM и описаны важные его компоненты, такие как LLVM IR, Clang и проходы LLVM. Можно сделать следующие выводы:

1) Инфраструктура LLVM позволяет модифицировать поведение компилятора различными способами, а то есть с ее помощью можно проводить оптимизации.

2) Есть два основных способа написания оптимизаций с помощью LLVM – проходы LLVM и плагины Clang.

3) LLVM проходы более универсальны, но плагины Clang позволяют модифицировать оригинальный код напрямую.

Таким образом, инфраструктуру LLVM можно использовать для реализации средств оптимизации программы на основе программной предвыборки данных. В соответствии с приведенными выше особенностями решено выбрать реализацию в виде LLVM прохода, так как она наиболее универсальна в применении.

3 Описание алгоритмов программной предвыборки данных

Данный раздел посвящен описанию алгоритмов программной предвыборки данных и их особенностей

3.1 Алгоритмы предвыборки данных для массивов

В данном подразделе описываются алгоритмы программной предвыборки данных и их особенности.

3.1.1 Алгоритм при прямом доступе по индексу

Первый алгоритм, который будет рассмотрен, используется для массивов при прямой итерации по нему с помощью цикла. Для удобства он будет рассмотрен на примере [19]. Алгоритм состоит из следующих основных шагов:

1) Для каждой ссылки определяются доступы к памяти, которые скорее всего вызовут промах кэша и, следовательно, должны быть предварительно загружены в него.

2) Изолируются предполагаемые промахи кэша через разделение цикла. Это позволяет избежать накладных расходов, связанных с добавлением условных операторов в тела циклов.

3) Вставляются команды предвыборки для каждого возможного промаха.

Далее опишем каждый шаг алгоритма более подробно.

Для того, чтобы определить промахи, для начало нужно провести анализ повторений доступов, которые попадают в одну кэш-линию и определить тип этого повторения. Для этого рассмотрим пример на рисунке 3.1.

Здесь доступ A[i][j] имеет пространственную повторяемость во внутреннем цикле. Так как итерации внутреннего цикла несколько раз подряд обращаются к одной и той же кэш-линии. Доступ B[j+1][0] имеет временную

повторяемость во внешнем цикле, так как каждая из итераций внешнего цикла обращается к одним и тем же адресам. Соответственно доступы B[j][0] и B[j+1][0] имеют групповую повторяемость, так как один из них постоянно обращается к адресам, к которым перед этим обратился другой из них (в предыдущей итерации).

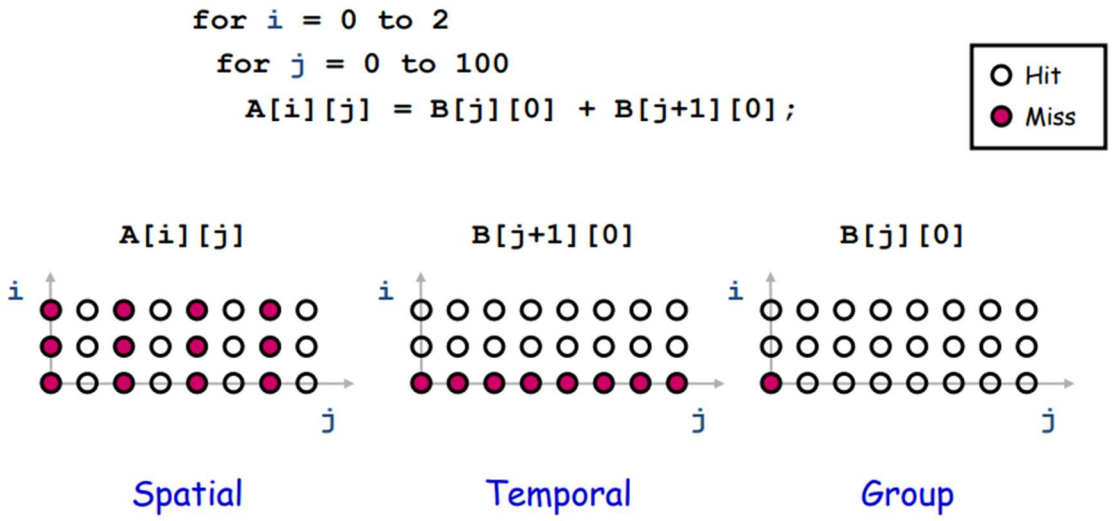

Рисунок 3.1 – Пример определения повторяемости

Далее нужно понять позволяют ли эти повторения говорить о локальности данных. Рассмотрим пример на рисунке 3.2.

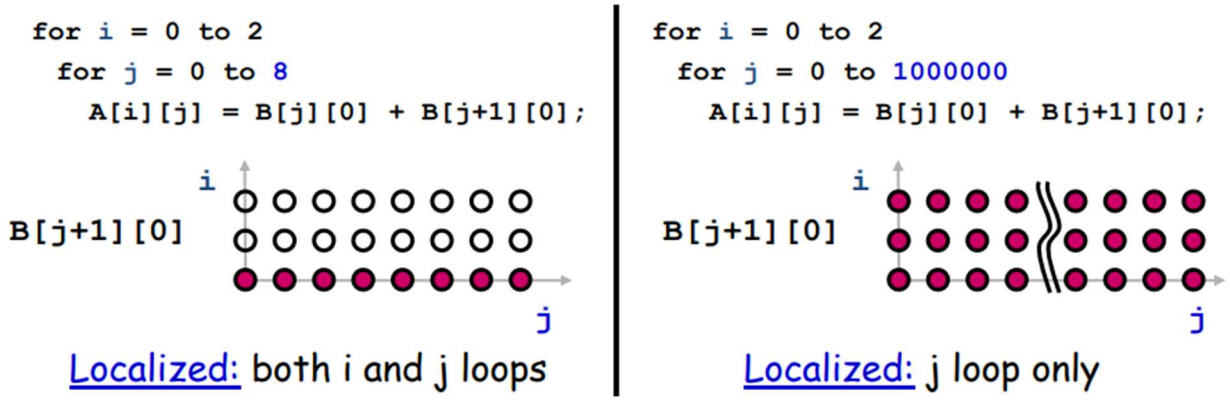

Рисунок 3.2 – Определение локальности

На рисунке видно, что если внутренний цикл слишком большой или слишком маленький, то временная повторяемость может потерять локальность. Почему так происходит? Ответ прост – для того, чтобы сохранялась локальность, требуется, чтобы внутренний цикл использовал

столько данных, сколько может поместиться в кэш-памяти. Если же он перейдет за эту границу, то данные в кэше начнут перезаписываться и локальность потеряется.

Далее определяем места промахов. Очевидно, что для массива A, по которому цикл проходит последовательно, промахи будут зависеть от размера кэш-линии (например, можно рассчитать итерации с промахами как j % s = 0, где s – это число элементов, помещающихся в кэш-линию) [19]. Для массива B промахи будут на первой итерации (но только для одного из двух доступов), то есть при i = 0.

Теперь, когда мы определили места промахов требуется разделить циклы таким образом, чтобы сделать предвыборку наиболее эффективно и вставить команды предвыборки. Сначала требуется понять, насколько k итераций вперед нужно делать предвыборку. Это зависит от задержки памяти и обычно получается экспериментально. При этом цикл при вставке команд предвыборки делится на три части. Первая называется прологом и делает предвыборку на первые k элементов. Вторая называется устойчивым состоянием и проводит следующие n - k (где n – общее число итераций) итераций цикла, предзагружая элементы на k итераций вперед. Третья называется эпилогом и проводит оставшиеся k итераций. Пример представлен на рисунке 3.3.

Original Loop

```
for (i = 0; i<100; i++)
   a[i] = 0;
```

Software Pipelined Loop
(5 iterations ahead)

```
for (i = 0; i<5; i++)        /* Prolog */
   prefetch(&a[i]);

for (i = 0; i<95; i++) {     /* Steady State*/
   prefetch(&a[i+5]);
   a[i] = 0;
}

for (i = 95; i<100; i++)     /* Epilog */
   a[i] = 0;
```

Рисунок 3.3 – Пример разделения цикла для прямого доступа

Далее рассмотрим, как именно нужно разделять циклы для разных типов локальности. В случае с временной повторяемостью (как для ранее упомянутого массива B) проще всего выделить первую итерацию внешнего цикла в отдельный блок кода, а дальше сделать предвыборку по принципу, описанному в предыдущем абзаце [19].

В случае с пространственной повторяемостью (массив A) цикл можно развернуть, изменив шаг цикла на число равное размеру кэш-линии. На рисунке 3.4 представлен пример вставки команд предвыборки.

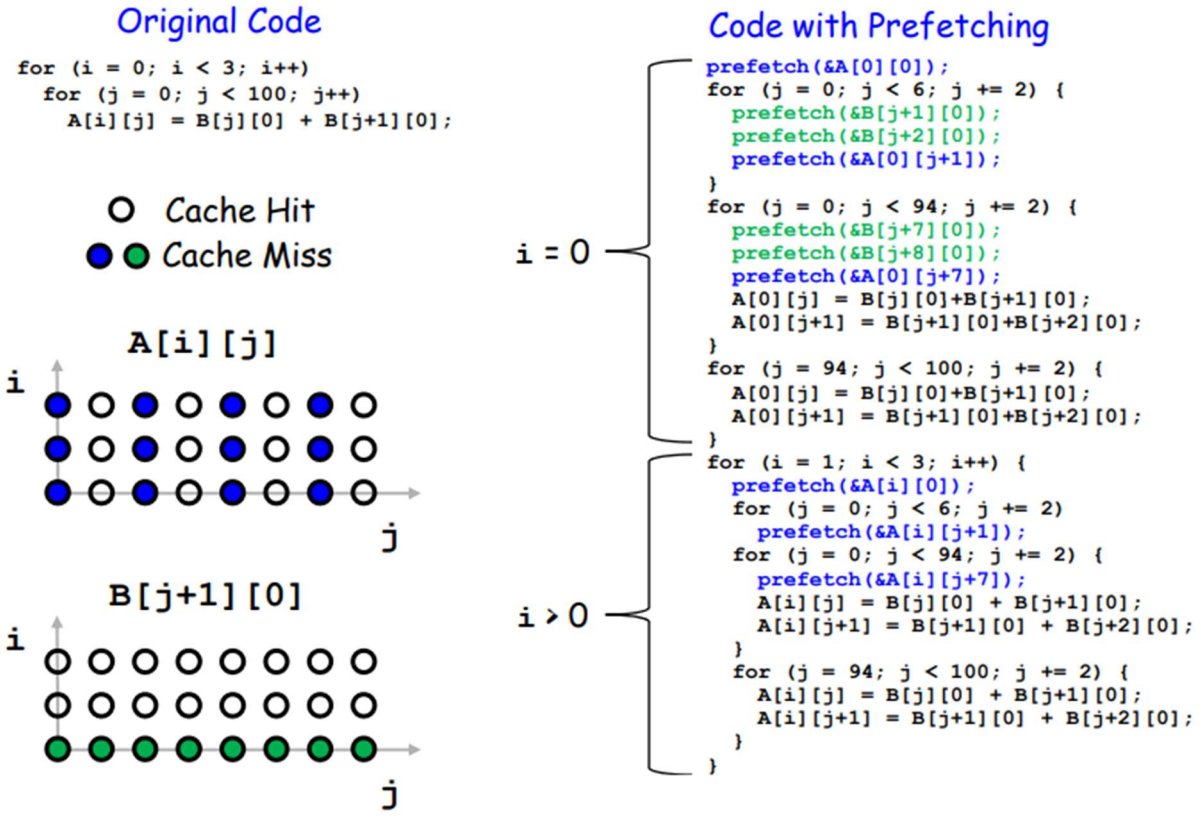

Рисунок 3.4 – Пример вставки команд предвыборки

Для групповой повторяемости обычно не требуется дополнительных действий, так как промахов в ней очень мало, но в данном примере можно дополнительно предзагрузить B[0][0].

Достоинство данного алгоритма в его простоте. К недостаткам можно отнести то, что алгоритм генерирует довольно много дополнительного кода и что некоторые из этих паттернов уже определяются процессором, то есть он будет проводить предвыборку этих данных аппаратно и дополнительная

программная предвыборка может не сочетаться с предвыборками, сделанными процессором.

3.1.2 Алгоритм при непрямом доступе к элементам массива

Под непрямым доступом будет понимать ситуацию, когда индексы, использующиеся для доступа к основному массиву данных, хранятся в другом массиве (например, доступ к массиву A происходит через элементы массива B A[B[i]]). Часто это применяется для разреженных матриц. В таком случае алгоритм предвыборки данных будет отличаться [19].

Для начала нужно установить два важных факта. Во-первых, предсказать доступа, при которых происходят промахи сложно. Причиной этому служит то, что индексы вычисляются не напрямую. Во-вторых, нужно предзагружать не только элементы основного массива, но и элементы массива индексов. Это важно, так как в противном случае команда предвыборки может сама вызвать промах, и оптимизация будет бессмысленной.

Итак, исходя из этих двух замечаний, рассмотрим алгоритм. Разделять циклы нужно по такому же принципу, что было описано до этого, но теперь у нас имеется массив индексов, который нужно предзагружать заранее до предзагрузки основного массива, то есть задержку памяти для индексов надо брать в два раза больше. Поэтому теперь требуется 2 цикла-пролога, один предзагружает первые k элементов массива индексов, а второй k соответствующих элементов из основного массива, а также последующие k элементов массива индексов. Далее в цикле устойчивого состояния мы делаем n - 2*k итераций, предзагружая индексы на 2*k итераций вперед, а элементы основного массива по индексу на k операций вперед. В конце помещается два цикла-эпилога, которые загружают оставшиеся 2*k элементов. Пример представлен на рисунке 3.5 [19].

Достоинство данного алгоритма, как и предыдущего в простоте. Также можно отметить, что данный паттерн уже не так легко предсказывается

процессором, так как элементы могут быть расположены достаточно разрозненно. Недостатком служат повышенные накладные расходы, связанные с расчетом индекса.

Original Loop

```
for (i = 0; i<100; i++)
   sum += A[index[i]];
```

Software Pipelined Loop
(5 iterations ahead)

```
for (i = 0; i<5; i++)        /* Prolog 1 */
   prefetch(&index[i]);

for (i = 0; i<5; i++) {      /* Prolog 2 */
   prefetch(&index[i+5]);
   prefetch(&A[index[i]]);
}
for (i = 0; i<90; i++) {  /* Steady State*/
   prefetch(&index[i+10]);
   prefetch(&A[index[i+5]]);
   sum += A[index[i]];
}
for (i = 90; i<95; i++) {  /* Epilog 1 */
   prefetch(&A[index[i+5]]);
   sum += A[index[i]];
}
for (i = 95; i<100; i++)     /* Epilog 2 */
   sum += A[index[i]];
```

Рисунок 3.5 – Пример разделения цикла для непрямого доступа

Далее рассмотрим алгоритмы, которые применяются для различных рекурсивных структур данных.

3.2 Алгоритмы предвыборки для рекурсивных структур данных

В данном подразделе описываются алгоритмы предвыборки данных для рекурсивных структур данных, то есть списков, деревьев, графов и т.д.

Основная сложность алгоритмов для рекурсивных структур данных состоит в том, что для предвыборки будущих элементов требуется обратиться по указателю к предшествующему. Из-за этого оптимизация при предвыборке на больше, чем один элемент вперед, не дает желаемых результатов. Пример смотрите на рисунках 3.6-3.7. В связи с этим существует ряд алгоритмов, которые решают эту проблему тем или иным образом [19].

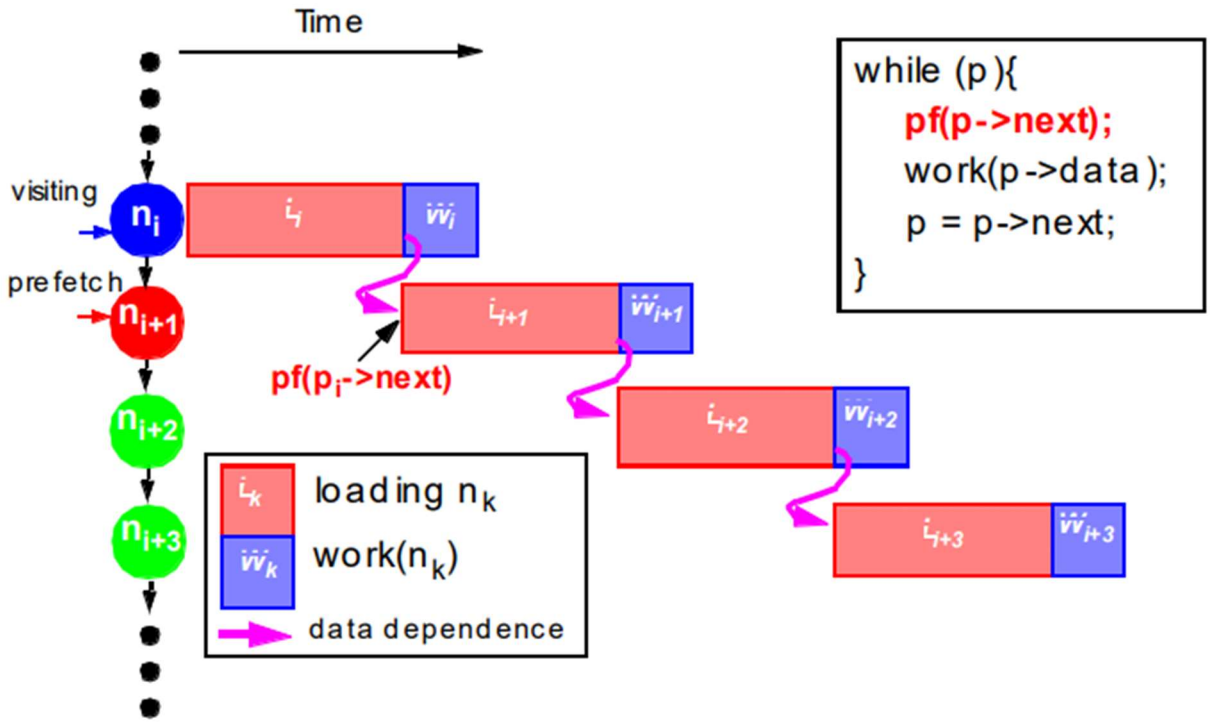

Рисунок 3.6 – Предвыборка на один узел вперед

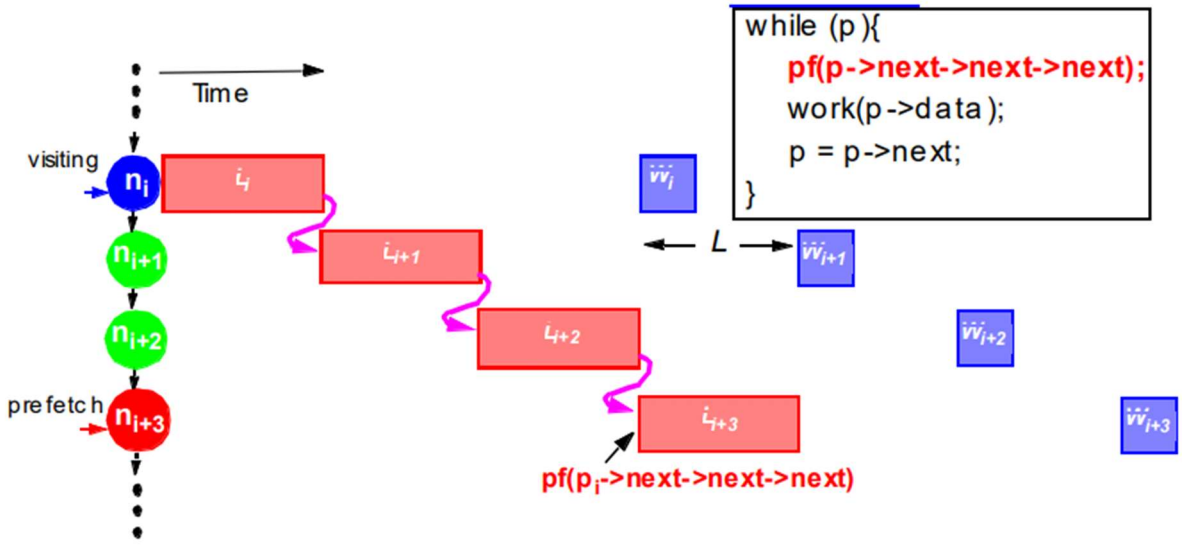

Рисунок 3.7 – Предвыборка на три узла вперед

Далее мы рассмотрим различные алгоритмы для рекурсивных структур данных.

3.2.1 Жадный алгоритм

Жадный алгоритм предвыборки данных для рекурсивных структур данных основывается на том, чтобы предзагружать все соседние элементы при

прохождении каждого из узлов [19]. При этом нельзя точно сказать, когда именно (через сколько итераций) будут посещены эти узлы, особенно в графах и деревьях. Пример представлен на рисунке 3.8.

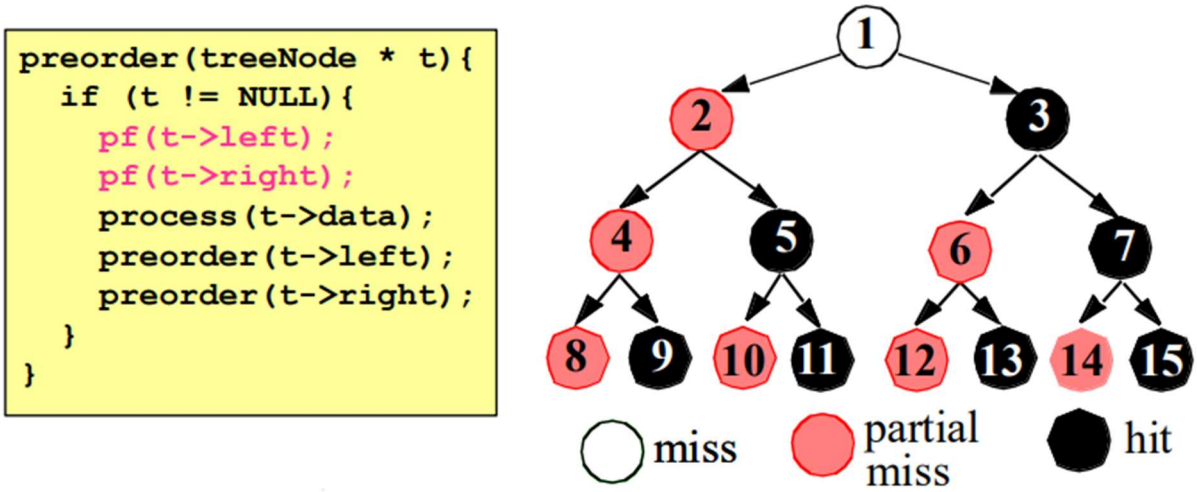

Рисунок 3.8 – Схема жадного алгоритма

Главное достоинство алгоритма в его простоте. На практике он также себя показывает довольно хорошо и реализован в компиляторе SUIF. Недостаток этого алгоритма, как было сказано ранее, в том, что нельзя предсказать, через сколько итераций будут посещены узлы.

3.2.2 Алгоритм обратного указателя (history-pointer)

Данный алгоритм заключается в том, что при первом обходе структуры данных, в каждый из узлов добавляются дополнительные указатели, которые будут указывать на несколько узлов вперед. Для этого во время обхода несколько последних узлов помещаются в очередь и при выходе из нее к узлу добавляется указатель на текущий элемент [19]. Схема алгоритма представлена на рисунке 3.9.

Достоинство алгоритма по сравнению с жадным в том, что он позволяет достаточно точно предзагружать узлы на требуемое количество итераций вперёд. Но существенные недостатки этого алгоритма заключаются в необходимости дополнительного обхода структуры данных и в необходимости добавления дополнительных указателей.

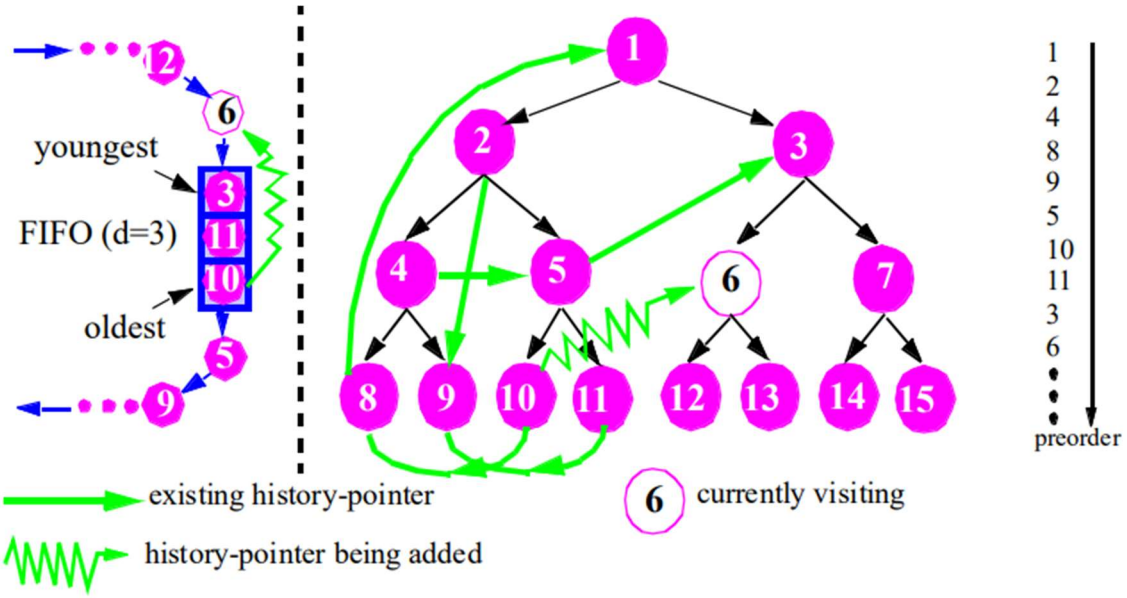

Рисунок 3.9 – Схема алгоритма обратного указателя

Также этот алгоритм более сложен в реализации.

3.2.3 Алгоритм линеаризации данных (data-linearization)

Данный алгоритм заключается в том, что узлы, которые при обходе находятся рядом, помещаются в смежных блоках памяти. Тем самым можно легко вычислить узел на любом расстоянии от текущего [19]. Схема алгоритма представлена на рисунке 3.10.

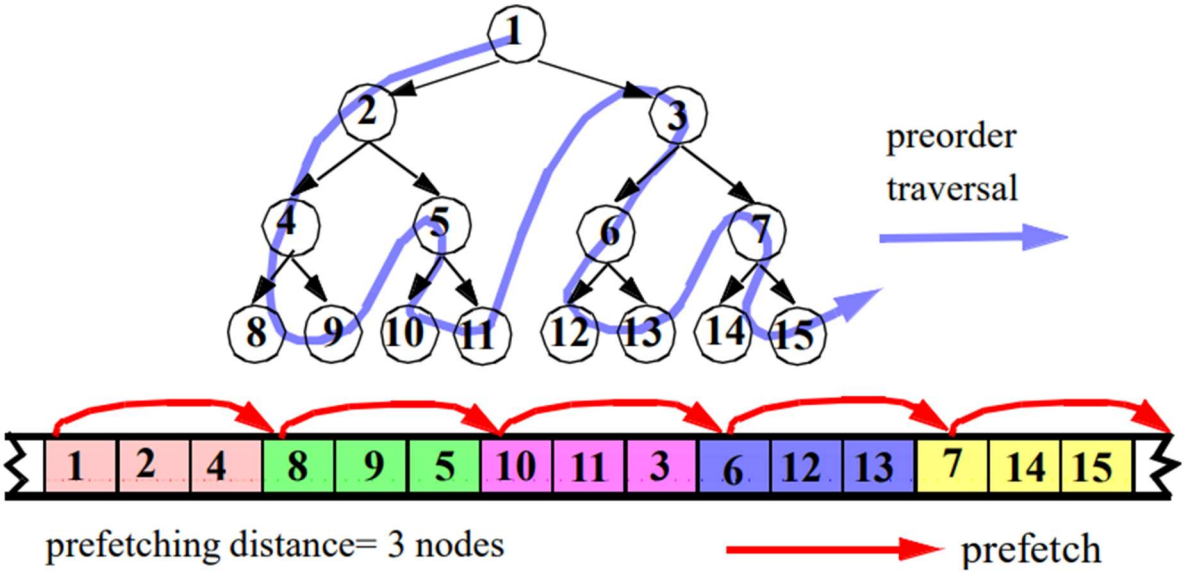

Рисунок 3.10 – Схема алгоритма линеаризации данных

Достоинство алгоритма по сравнению с алгоритмом обратного указателя в том, что у этого алгоритма нет дополнительных накладных расходов на указатели. Однако есть и существенные недостатки. Первый из них заключается в том, что структура данных обязательно должна иметь известный порядок обхода узлов. Второй заключается в том, что требуется, чтобы узлы были расположены в смежных блоках памяти, то есть нужно заранее предусмотреть аллокатор, который это сделает или перераспределить узлы в памяти.

3.3 Выводы по разделу

В данном разделе был проведен анализ алгоритмов программной предвыборки данных. В результате можно сделать следующие выводы:

1) Предвыборка данных может быть применена как к массивам, так и к рекурсивным структурам данных.

2) Алгоритмы для массивов более простые, но и более точные, чем алгоритмы для рекурсивных структур данных.

3) Предвыборку непрямых доступов к памяти сложно выполнить на аппаратном уровне, а следовательно, лучше выполнять на программном.

Таким образом, можно выбрать алгоритм для реализации. Исходя из сделанных ранее выводов лучше начать с реализации алгоритма для массивов с непрямыми доступами к памяти. Он более простой, но достаточно эффективный.

4 Реализация и тестирование алгоритма предвыборки данных

В данном разделе приведено описание реализации и тестирования алгоритма программной предвыборки данных.

4.1 Описание реализации алгоритма предвыборки

Реализация оптимизации выполнена в виде LLVM прохода. Исходный код представлен в приложении А.

Реализация основана на проходе, созданном Sam Ainsworth и Timothy M. Jones [20].

В ее основе лежит следующий алгоритм:

1) Алгоритм проходит по циклам и находит индуктивные переменные.

2) Алгоритм определяет, что в этом цикле присутствует непрямой доступ к элементам массива.

3) Алгоритм проверяет возможно ли провести трансформацию (достаточен ли размер массива индексов для предвыборки).

4) Алгоритм вычисляет задержку предвыборки по формуле.

5) Алгоритм вставляет команды предвыборки в виде LLVM Intrinsic.

Формула для вычисления задержки [20]:

$$k = \frac{(t - (l - 1))}{t},$$

где c – микроархитектурная константа, вычисленная экспериментально (взято значение 64, так как оно оптимально для большинства архитектур), t – число доступов к памяти, l – число доступов памяти в данном префетче (то есть, если это предвыборка для основного массива, то их будет 2, а если для массива индексов, то будет 1).

4.2 Тестирование алгоритма

Алгоритм был протестирован на бенчмарке randacc [20]. Эта программа выполняет множество случайных непрямых доступов к огромной таблице данных. Исходный код программы представлен в приложении А.

Эксперимент проводился при компиляции с уровнем оптимизации O1 в среде WSL2, процессор – AMD Ryzen 7 5800X. Процесс эксперимента состоял из следующих этапов. Исходный код программы randacc компилировался в LLVM IR со следующими параметрами:

```
clang -S -emit-llvm -O1 -Xclang -disable-O0-optnone
-Xclang -no-opaque-pointers randacc.c
```

Затем к этому файлу применялась оптимизация с помощью утилиты opt.

```
opt -S -load-pass-plugin=../libPrefetchIndirect.so
-passes=prefetch-indirect randacc.ll -debug > randopt.ll
```

Затем получившийся файл и оригинальный LLVM IR (файлы представлены в приложении Б) компилировались в исполняемые файлы.

```
clang randopt.ll -o randopt
clang randacc.ll -o randacc
```

Затем время выполнения обеих программ сравнивалось с помощью утилиты time. Для точности измерение проводилось 5 раз, результатом взяли среднее. Результаты представлены в таблице 4.1. Время измеряется в секундах.

Таблица 4.1 – Результаты работы программы

no prefetch	6.698	6.660	6.674	6.703	6.627
prefetch	6.286	6.275	6.254	6.255	6.256

Среднее значение для программы до применения оптимизации – 6.672 секунд, а после – 6.265 секунд. Соответственно ускорение программы – 1.065, т.е. результат работает примерно на 6.5% быстрее.

4.3 Выводы по разделу

В данном разделе были описаны реализация алгоритма предвыборки и тестирование этой реализации на бенчмарке randacc. В результате можно сделать следующие выводы:

1) Оптимизация была успешно реализована и выполняет поставленную задачу (предвыборка данных).

2) Получен положительный результат (ускорение) на одной тестовой программе.

Таким образом, можно сделать вывод, что предвыборка данных действительно может помочь оптимизировать программу. Однако нужно провести больше экспериментов на различных бенчмарках и различных платформах, чтобы выяснить, насколько эффективна эта оптимизация на самом деле.

5 Оценка и защита результатов интеллектуальной деятельности

Данный раздел посвящен защите результатов интеллектуальной деятельности, полученных в ВКР. В разделе приводится описание результата интеллектуальной деятельности, оценка его рыночной стоимости, а также раскрытие нормативно-правовых актов, позволяющих защитить объект исследования.

5.1 Описание результата интеллектуальной деятельности

Целью выпускной квалификационной работы является анализ методов предвыборки данных в кэш и тестирование различных методов.

В результате выполнения ВКР был разработан программный комплекс для анализа оптимизации при использовании предвыборки данных в кэш, то есть программа для электронных вычислительных машин. Согласно Гражданскому кодексу РФ, программы для ЭВМ являются интеллектуальной собственностью [1]. Таким образом эту программу можно принять в качестве объекта исследования.

5.2 Оценка рыночной стоимости программы

В данном подразделе оценивается рыночная стоимость программы. В начале приводится описание подходов к оценке, а затем описывается сам процесс оценки рыночной стоимости.

5.2.1 Выбор подхода к оценке

Для оценки рыночной стоимости используются три основных подхода: доходный, сравнительный и затратный [1]. Рассмотрим каждый из этих подходов отдельно.

Доходный подход основан на дисконтировании денежных потоков. При его использовании нужно определить период, в течении которого объект

может приносить доход. Рекомендуется в том случае, когда можно достоверно прогнозировать будущие доходы [1].

Затратный подход основан на определении затрат на производство объекта (включая расходы на оплату труда, материалы и т.п.). Рекомендуется применять при недостаточности данных для остальных подходов в случае, если объект оценки создан самим правообладателем [1].

Сравнительный поход основан на сопоставлении объекта с его аналогами. Рекомендуется применять данный метод, если имеется достаточная информация об объектах-аналогах (цены, характеристики) [1].

В некоторых случаях можно применять 2 или 3 метода сразу, а затем привести результаты к одной стоимости (например, используя среднее арифметическое значение) [1].

Для данной работы целесообразно выбрать затратный подход, так как данных для двух других подходов недостаточно, но возможно определить затраты на создание программы.

5.2.2 Применение выбранного подхода

Для оценки по затратному подходу требуется рассчитать затраты на разработку программы.

Сначала рассчитаем затраты на заработную плату. Исполнителем в данном случае является разработчик программы. Для оценки основной заработной платы используем сайт HH (HeadHunter). При анализе рынка труда выявлено, что средняя месячная заработная плата разработчика составила 125000 рублей [2].

Для определения продолжительности работы возьмем количества фактически затраченных дней. Рассчитаем дневную ставку, взяв в качестве числа рабочих дней в месяце 21 день:

$$\mathrm{C} = \frac{125000}{21} = 5952{,}38 \text{ руб./день}$$

В качестве времени работы возьмем T = 20 дней.

Теперь можно рассчитать расходы на основную заработную плату исполнителей:

$$З_{осн.з/пл} = С * Т = 5952{,}38 * 20 = 119047{,}6 \text{ руб.}$$

Для расчёта дополнительной заработной платы требуется расходы на основную заработную плату умножить на норматив (8.3%).

$$З_{доп.з/пл} = З_{осн.з/пл} * \frac{Н_{доп}}{100} = 119047{,}6 \frac{8{,}3}{100} = 9880{,}95 \text{ руб.}$$

Для расчёта социальных отчислений сумма расходов на основную и дополнительную заработную плату умножается на норматив (30%).

$$З_{соц} = (З_{осн.з/пл} + З_{доп.з/пл}) * \frac{Н_{соц}}{100} =$$

$$= (119047{,}6 + 9880{,}95) \frac{30}{100} = 38678{,}57 \text{ руб.}$$

Суммы затрат на основную и дополнительную заработную плату, а также на социальные отчисления приведены в таблице 5.1.

Таблица 5.1 – Расходы на заработную плату

№ п/п	Вид затрат	Сумма, руб.
1	Основная заработная плата	119047,6
2	Дополнительная заработная плата	9880,95
3	Отчисления на социальные нужды	38678,57

Итоговая сумма затрат на заработную плату составляет 167607,12 рублей.

Далее рассчитаем накладные расходы. В данном случае их можно рассчитать по нормативу 20% от суммарной заработной платы.

$$З_{накл} = (119047{,}6 + 9880{,}95) \frac{20}{100} = 25785{,}71 \text{ руб.}$$

Итоговая сумма накладных расходов составляет 25785,71 рубль.

Далее рассчитаем расходы на электроэнергию и Интернет.

Для расчёта использованной электроэнергии возьмем средний показатель потребления мощности ПК и монитора – 200 Вт, а среднюю потребляемую мощность лампы – 50 Вт.

То есть за 20 рабочих дней по 8 часов работы суммарно будет потрачено 0.25 кВт * 8 ч * 20 = 40 кВт*ч. При этом тариф на электроэнергию в Санкт-Петербурге составляет 6.51 рублей за кВт*ч [3]. Расчёт затрат на электроэнергию приведён в таблице 4.2.

Для расчёта затрат на Интернет возьмем месячный тариф интернет-провайдера Skynet. Он составляет 600 рублей в месяц [4]. Так как продолжительность разработки составляла меньше месяца достаточно рассчитать стоимость интернета за один месяц. Расчёт затрат на интернет приведён в таблице 5.2.

Таблица 5.2 – Расходы на интернет и электроэнергию

№ п/п	Вид затрат	Сумма, руб.
1	Электроэнергия	40*6.51 = 260.4
2	Интернет	600*1 = 600

Итоговая сумма затрат на интернет и электроэнергию составляет 860.4 рублей.

Далее необходимо учесть юридические издержки. Для этого рассчитаем расходы на государственную регистрацию программы.

Для регистрации воспользуемся услугами патентного бюро Онлайн Патент [5]. Данные о расходах на регистрацию представлены в таблице 5.3.

Таблица 5.3 – Расходы на юридические издержки

№ п/п	Вид затрат	Сумма, руб.
1	Стоимость регистрации программы	19900
2	Государственная пошлина за регистрацию	3000

Итоговая сумма затрат на юридические издержки составляет 22900 рублей.

Теперь необходимо вычислить сумму затрат на производство объекта оценки. Расчёты приведены в таблице 5.4.

Таблица 5.4 – Калькуляция затрат на разработку программы

№ п/п	Наименование статьи	Сумма, руб.
1	Заработная плата	167607,12
2	Накладные расходы	25785,71
3	Интернет и электроэнергия	860.4
4	Юридические издержки	22900
ИТОГО затрат		217153,23

Сумма затрат по всем статьям получилась 217153,23 рублей. Получившаяся величина является рыночной стоимостью результата интеллектуальной деятельности.

5.3 Правовая защита результатов интеллектуальной деятельности

В данном подразделе проводится анализ правовой защиты объекта исследования.

В начале приводится перечень и описание нормативно-правовых актов, которые обеспечивают правовую защиту этого объекта на территории РФ, а затем описываются объемы и сроки правовой защиты.

5.3.1 Нормативные акты, обеспечивающие правовую защиту

Основные источники права интеллектуальной собственности в РФ: Конституция РФ, Гражданский кодекс РФ, федеральные законы, подзаконные нормативно-правовые акты, а также международные соглашения.

Рассмотрим каждый из них по отдельности.

В Конституции Российской Федерации приводятся следующие статьи:

1. Ст. 44 Конституции Российской Федерации – «Каждому гарантируется свобода литературного, художественного, научного, технического и других видов творчества, преподавания. Интеллектуальная собственность охраняется законом» [6].

2. П. 4 ст. 15 Конституции Российской Федерации: «Общепризнанные принципы и нормы международного права и международные договоры Российской Федерации являются составной частью ее правовой системы. Если международным договором Российской Федерации установлены иные правила, чем предусмотренные законом, то применяются правила международного договора» [6].

В Гражданском кодексе Российской Федерации приводится следующая статья:

Статья 1270 – «Исключительное право на произведение» [7]. В данной статье определяются права автора и правообладателя программы. В ней перечисляются действия, которые являются использованием программы (с целью извлечения прибыли и без неё).

Уголовный кодекс Российской Федерации предусматривает наказание за нарушения прав на интеллектуальную собственность. Размеры наказаний указаны в статьях 146 («Нарушение авторских и смежных прав») и 147 («Нарушение изобретательских и патентных прав») УК РФ [8].

Также права на интеллектуальную собственность защищаются приказом Министерства экономического развития РФ от 29 сентября 2016 г. N 616 – «Об утверждении Административного регламента предоставления Федеральной службой по интеллектуальной собственности государственной услуги по аттестации и регистрации патентных поверенных Российской Федерации, выдаче патентным поверенным свидетельств».

Помимо этого, существует перечень международных договоров, которые являются источниками права на интеллектуальную собственность. Приведем некоторые из них:

- Конвенция об учреждении Всемирной организации интеллектуальной собственности (1967);
- Женевская конвенция об авторских правах (1952);
- Международная конвенция об охране прав исполнителей, изготовителей фонограмм и вещательных организаций (1961);
- Соглашение по торговым аспектам прав интеллектуальной собственности (ТРИПС/TRIPS) (1994);
- Договор ВОИС по авторскому праву (Вместе с “Согласованными заявлениями в отношении Договора ВОИС по авторскому праву”) (1996).

Но стоит отметить, что если международный договор противоречит Конституции РФ, то он не подлежит исполнению на территории Российской Федерации (ст. 79 Конституции РФ).

5.3.2 Объем и срок и правовой защиты объекта исследования

В статье 1261 Гражданского кодекса сказано: «Программа для ЭВМ – это представленная в объективной форме совокупность данных и команд, которая предназначена для функционирования ЭВМ и прочих компьютерных устройств, чтобы получить определенный результат, включая подготовительные материалы, полученные в ходе разработки программы для ЭВМ и порождаемые ею аудиовизуальные отображения» [7].

Авторские права на любые программы для ЭВМ (в т. ч. на программные комплексы), которые могут быть выражены на любом языке и в любой форме, включая исходный текст и объектный код, охраняются так же, как авторские права на произведения литературы.

На программу для ЭВМ распространяются как интеллектуальные права (в том числе исключительное право, являющееся имущественным правом), так и личные неимущественные права.

В случае необходимости защиты авторского права, можно зарегистрировать программу для ЭВМ в Федеральной службе по

интеллектуальной собственности, патентам и товарным знакам РФ (ст. 1262 ГК РФ) [7].

В соответствии со статьей 1281 ГК РФ исключительное право на произведение действует на протяжении жизни автора и 70 лет, считая с 1 января года, следующего за годом смерти автора.

5.4 Выводы по разделу

В данном разделе была произведена оценка результата интеллектуальной деятельности (программы для ЭВМ), а также были описаны законные методы защиты права собственности на этот результат.

Для оценки стоимости программы был использован затратный подход. В результате получилась сумма 217153,23 рублей. Большую часть затрат составили затраты на оплату труда разработчика.

Для защиты интеллектуальной собственности было рассмотрено множество нормативно-правовых документов Российской Федерации, а также международных договоров. В результате можно сделать вывод, что основной способ защиты авторского права – это регистрация программы в Роспатенте.

ЗАКЛЮЧЕНИЕ

В результате выполнения выпускной квалификационной работы был разработан проход LLVM, который реализует алгоритм предвыборки данных в кэш для непрямого доступа к элементам массива.

Разработанная программа получает на вход LLVM IR и оптимизирует его, вставляя prefetch директивы. Программа работает в инфраструктуре LLVM.

В ходе работы были решены все задачи, которые были поставлены. Был проведен анализ методов предвыборки данных и кода, рассмотрены различные типы этих методов и сделан вывод о том, что предвыборка данных позволяет ускорить работу программы за счет оптимизации работы кэш-памяти и уменьшения количество промахов кэша.

Также была изучена архитектура LLVM. Этот проект позволяет создавать свои компиляторные оптимизации, полученные знания были применены в ходе реализации алгоритма.

После изучения различных алгоритмов программной предвыборки данных были сделаны выводы о том, что у каждого из них есть свои преимущества и недостатки, а также подходящая область применения. В итоге для реализации был выбран алгоритм предвыборки данных для массивов с непрямым доступом к элементам. Этот алгоритм легко реализуется программно, но аппаратно такие паттерны отлавливать тяжело, поэтому он имеет больший потенциал, чем другие алгоритмы.

В результате был реализован этот алгоритм и протестирован с использованием бенчмарка randacc. Получено ускорение 1.065. На точность данного значения могут влиять внешние факторы: фоновые программы, сбои в работе системы, поэтому для более полной оценки алгоритма нужно провести эксперименты над большим числом бенчмарков и на различных платформах.

Помимо проведения большего числа экспериментов в качестве развития данной работы следует заняться реализацией других алгоритмов, например, алгоритмов для рекурсивных структур данных. В будущем можно рассмотреть возможность добавления реализованных алгоритмов в стандартную библиотеку LLVM.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Магомедов М.Н., Чигирь М.В. Оценка и защита результатов интеллектуальной деятельности: учебно-методическое пособие по выполнению дополнительного раздела выпускных квалификационных работ магистров. – СПб.: Изд-во СПБГЭТУ «ЛЭТИ», 2019. – 26 с.
2. Headhunter // Работа разработчиком ПО в Санкт-Петербурге. – URL: https://spb.hh.ru/vacancies/razrabotchik_po (дата обращения 10.05.2023)
3. Официальный сайт администрации Санкт-Петербурга // Тарифы 2023 года. – URL: https://www.gov.spb.ru/helper/tarif/tarify-2023/ (дата обращения: 10.05.2023)
4. Skynet // Интернет 100 Мбит/с – подключить в СПб. – URL: https://skynet.ru/internet/tarify/tarif-internet-t-100/ (дата обращения: 12.05.2023)
5. Онлайн Патент // Сколько стоит зарегистрировать программное обеспечение. – URL: https://onlinepatent.ru/faq/evmdb/price-software/ (дата обращения 12.05.2023)
6. Конституция Российской Федерации: принята всенародным голосованием 12 декабря 1993 г. с изменениями, одобренными в ходе общероссийского голосования 01 июля 2020 г. // Официальный интернет–портал правовой информации. – URL: http://www.pravo.gov.ru (дата обращения: 12.05.2023).
7. Гражданский кодекс Российской Федерации (часть четвертая): от 18.12.2006 № 230-ФЗ (ред. от 05.12.2022) // Собрание законодательства РФ. – 25.12.2006. – № 52 (1 ч.). – Ст. 5496.
8. Уголовный кодекс Российской Федерации: от 13.06.1996 № 63-ФЗ (ред. от 07.04.2020) // Собрание законодательства РФ. – 17.06.1996. – № 25. – Ст. 2954.
9. Генератор продаж // Скорость загрузки сайта: удобство пользователей и ваша прибыль. – URL: https://sales-generator.ru/blog/skorost-zagruzki-sayta/ (дата обращения: 12.05.2023)

10. Google Search Central // #MobileMadness. – URL: https://developers.google.com/search/blog/2015/04/mobilemadness-campaign-to-help-you-go (дата обращения: 12.05.2023)

11. Контент-платформа Pandia // Иерархическая структура памяти. – URL: https://pandia.ru/text/78/108/8078.php (дата обращения: 12.05.2023)

12. TechTarget // What is cache memory. – URL: https://www.techtarget.com/searchstorage/definition/cache-memory (дата обращения: 12.05.2023)

13. Basics of Cache memory. – URL: https://www.cs.umd.edu/~meesh/411/CA-online/chapter/basics-of-cache-memory/index.html (дата обращения: 12.05.2023)

14. When is prefetching useful. – URL: https://www.usenix.org/legacy/events/fast07/tech/full_papers/gill/gill_html/node3.html (дата обращения: 12.05.2023)

15. Improving Direct-Mapped Cache Performance by the Addition of a Small Fully-Associative Cache and Prefetch buffers. – URL: https://citeseerx.ist.psu.edu/doc/10.1.1.37.6114 (дата обращения: 12.05.2023)

16. Marius Grannaes, Magnus Jahre, Lasse Natvig. Storage Efficient Hardware Prefetching using Delta-Correlating Prediction Tables // Journal of Instruction-Level Parallelism. – 2011. – N 13. – C.1-16.

17. GCC docs. – URL: https://gcc.gnu.org/onlinedocs/gcc/Target-Builtins.html (дата обращения: 12.05.2023)

18. LLVM Documentation. – URL: https://llvm.org/docs/LangRef.html (дата обращения: 12.05.2023)

19. Compiler algorithms for prefetching data. – URL: http://www.cs.cmu.edu/afs/cs/academic/class/15745-s12/public/lectures/L27-Data-Prefetching-1up.pdf (дата обращения: 12.05.2023)

20. Software prefetching for indirect memory accesses. – URL: https://ctuning.org/ae/resources/paper-with-distinguished-ck-artifact-and-ae-appendix-cgo2017.pdf (дата обращения: 12.05.2023)

ПРИЛОЖЕНИЕ А

Исходный код программы на C++

LLVM-Pass, реализующий алгоритм предвыборки данных.

```
#include "PrefetchIndirect.h"
#ifdef NO_STRIDES
#define C_CONSTANT (32)
#else
#define C_CONSTANT (64)
#endif
#ifndef IGNORE_SIZE
#define IGNORE_SIZE 0
#endif
/// New PM implementation
PreservedAnalyses PrefetchIndirect::run(Function &F, FunctionAnalysisManager &FAM) {
    LoopInfo &LI = FAM.getResult<LoopAnalysis>(F);
    Module *M = F.getParent();
    bool modified = false;
    SmallVector<Instruction*,4> Loads;
    SmallVector<Instruction*,4> Phis;
    SmallVector<int,4> Offsets;
    SmallVector<int,4> MaxOffsets;
    std::vector<SmallVector<Instruction*,8>> Insts;
    for(auto &BB : F) {
    for (auto &I : BB) {
        if (LoadInst *i = dyn_cast<LoadInst>(&I)) {
        if(LI.getLoopFor(&BB)) {
            SmallVector<Instruction*,8> Instrz;
            Instrz.push_back(i);
            Instruction * phi = nullptr;
            if(depthFirstSearch(i,LI,phi,Instrz,  Loads, Phis, Insts)) {
              int loads = 0;
              for(auto &z : Instrz) {
              if(dyn_cast<LoadInst>(z)) {
                loads++;
              }
              }

            if(loads < 2) {
```

```
            LLVM_DEBUG(dbgs()<<"stride\n");    //don't remove the stride cases yet though. Only
remove them once we know it's not in a sequence with an indirect.
    #ifdef NO_STRIDES
            //add a continue in here to avoid generating strided prefetches. Make sure to reduce
the value of C accordingly!
            continue;
    #endif
        }

          LLVM_DEBUG(dbgs() << "Can prefetch " << *i << " from PhiNode " << *phi << "\n");
          LLVM_DEBUG(dbgs() << "need to change \n");
          for (auto &z : Instrz) {
            LLVM_DEBUG(dbgs() << *z << "\n");
          }

          Loads.push_back(i);
          Insts.push_back(Instrz);
          Phis.push_back(phi);
          Offsets.push_back(0);
          MaxOffsets.push_back(1);

       } else {
         LLVM_DEBUG(dbgs() << "Can't prefetch load" << *i << "\n");
       }

        }
      }
    }
   }

    for(uint64_t x=0; x<Loads.size(); x++) {
    ValueMap<Instruction*, Value*> Transforms;
    bool ignore = true;
    Loop* L = LI.getLoopFor(Phis[x]->getParent());
    for(uint64_t y=x+1; y< Loads.size(); y++) {
        bool subset = true;
        for(auto &in : Insts[x]) {
          if(std::find(Insts[y].begin(),Insts[y].end(),in) == Insts[y].end()) subset = false;
        }
        if(subset) {
          MaxOffsets[x]++;
          Offsets[y]++;
```

```
          ignore=false;
        }
    }
    int loads = 0;
    LoadInst* firstLoad = NULL;
    for(auto &z : Insts[x]) {
        if(dyn_cast<LoadInst>(z)) {
        if(!firstLoad) firstLoad = dyn_cast<LoadInst>(z);
          loads++;
        }
    }
    //loads limited to two on second case, to avoid needing to check bound validity on later
loads.
    if((getCanonicalishSizeVariable(L)) == nullptr) {
        if(!getArrayOrAllocSize(firstLoad, M) || loads >2)
        continue;
    }
    if(loads < 2 && ignore) {
        LLVM_DEBUG(dbgs() << "Ignoring" << *(Loads[x]) << "\n");
        continue; //remove strides with no dependent indirects.
    }
    IRBuilder<> Builder(Loads[x]);
    bool tryToPushDown = (LI.getLoopFor(Loads[x]->getParent()) != LI.getLoopFor(Phis[x]-
>getParent()));
    if(tryToPushDown) LLVM_DEBUG(dbgs() << "Trying to push down!\n");
    //reverse list.
    SmallVector<Instruction*,8> newInsts;
    for(auto q = Insts[x].end()-1; q > Insts[x].begin()-1; q--) newInsts.push_back(*q);
    for(auto &z : newInsts) {
        if(Transforms.count(z)) continue;
        if(z == Phis[x]) {
        Instruction* n;
        bool weird = false;
        Loop* L = LI.getLoopFor(Phis[x]->getParent());
        int offset = (C_CONSTANT*MaxOffsets[x])/(MaxOffsets[x]+Offsets[x]);
        if(z == getCanonicalishInductionVariable(L)) {
            n = dyn_cast<Instruction>(Builder.CreateAdd(Phis[x],Phis[x]->getType()-
>isIntegerTy(64) ? ConstantInt::get(Type::getInt64Ty(M->getContext()),offset) :
ConstantInt::get(Type::getInt32Ty(M->getContext()),offset)));
    #ifdef BROKEN
        n = dyn_cast<Instruction>(Builder.CreateAnd(n,1));
    #endif
```

```
        }
        else if (z == getWeirdCanonicalishInductionVariable(L)) {
            //This covers code where a pointer is incremented, instead of a canonical induction
variable
            n = getWeirdCanonicalishInductionVariableGep(L)->clone();
            Builder.Insert(n);
            n->setOperand (n->getNumOperands ()-1, ConstantInt::get(Type::getInt64Ty(M-
>getContext()),offset));
            weird = true;
            bool changed = true;
            while(LI.getLoopFor(Phis[x]->getParent()) != LI.getLoopFor(n->getParent()) &&
changed){
              Loop* ol = LI.getLoopFor(n->getParent());
              makeLoopInvariantSpec(n,changed,LI.getLoopFor(n->getParent()));
              if(ol && ol == LI.getLoopFor(n->getParent())) break;
            }
         }
        assert(L);
        assert(n);
        Value* size = getCanonicalishSizeVariable(L);
        if(!size) {
            size = getArrayOrAllocSize(firstLoad, M);
        }
        assert(size);
        std::string type_str;
        llvm::raw_string_ostream rso(type_str);
        size->getType()->print(rso);
        rso.flush();
        /* assert(size->getType()->isIntegerTy() || IGNORE_SIZE); */
        if(loads< 2 || !size || !size->getType()->isIntegerTy() || IGNORE_SIZE) {
            Transforms.insert(std::pair<Instruction*,Instruction*>(z,n));
            continue;
        }
        Instruction* mod;
        if(weird) {
            //This covers code where a pointer is incremented, instead of a canonical induction
variable.
            Instruction* endcast =
dyn_cast<Instruction>(Builder.CreatePtrToInt(size,Type::getInt64Ty(M->getContext())));
            Instruction* startcast =
dyn_cast<Instruction>(Builder.CreatePtrToInt(getWeirdCanonicalishInductionVariableFirst(L),Type::g
etInt64Ty(M->getContext())));
```

```
            Instruction* valcast =
dyn_cast<Instruction>(Builder.CreatePtrToInt(n,Type::getInt64Ty(M->getContext())));
            Instruction* sub1 = dyn_cast<Instruction>(Builder.CreateSub(valcast,startcast));
            Instruction* sub2 = dyn_cast<Instruction>(Builder.CreateSub(endcast,startcast));
            Value* cmp = Builder.CreateICmp(CmpInst::ICMP_SLT,sub1,sub2);
            Instruction* rem = dyn_cast<Instruction>(Builder.CreateSelect(cmp,sub1,sub2));
            Instruction* add = dyn_cast<Instruction>(Builder.CreateAdd(rem,startcast));
            mod = dyn_cast<Instruction>(Builder.CreateIntToPtr(add,n->getType()));
        }
        else if(size->getType() != n->getType()) {
            Instruction* cast = CastInst::CreateIntegerCast(size,n->getType(),true);
            assert(cast);
            Builder.Insert(cast);
            Value* sub = Builder.CreateSub(cast,ConstantInt::get(Type::getInt64Ty(M-
>getContext()),1));
            Value* cmp = Builder.CreateICmp(CmpInst::ICMP_SLT,sub,n);
            mod = dyn_cast<Instruction>(Builder.CreateSelect(cmp,sub,n));
        } else {
            Value* sub = Builder.CreateSub(size,ConstantInt::get(n->getType(),1));
            Value* cmp = Builder.CreateICmp(CmpInst::ICMP_SLT,sub,n);
            mod = dyn_cast<Instruction>(Builder.CreateSelect(cmp,sub,n));
        }
        bool changed = true;
        while(LI.getLoopFor(Phis[x]->getParent()) != LI.getLoopFor(mod->getParent()) && changed) {
            Loop* ol = LI.getLoopFor(mod->getParent());
            makeLoopInvariantSpec(mod,changed,LI.getLoopFor(mod->getParent()));
            if(ol && ol == LI.getLoopFor(mod->getParent())) break;
        }
        Transforms.insert(std::pair<Instruction*,Instruction*>(z,mod));
        modified = true;
        } else if (z == Loads[x]) {
        assert(Loads[x]->getOperand(0));
        Instruction* oldGep = dyn_cast<Instruction>(Loads[x]->getOperand(0));
        assert(oldGep);
        assert(Transforms.lookup(oldGep));
        Instruction* gep = dyn_cast<Instruction>(Transforms.lookup(oldGep));
        assert(gep);
        modified = true;
        Instruction* cast = dyn_cast<Instruction>(Builder.CreateBitCast (gep,
Type::getInt8PtrTy(M->getContext())));
```

```
        bool changed = true;
        while(LI.getLoopFor(Phis[x]->getParent()) != LI.getLoopFor(cast->getParent()) && changed)
{
            Loop* ol = LI.getLoopFor(cast->getParent());
            makeLoopInvariantSpec(cast,changed,ol);
            if(ol && ol == LI.getLoopFor(cast->getParent())) break;
        }
        Value* ar[] = {
            cast,
            ConstantInt::get(Type::getInt32Ty(M->getContext()),0),
            ConstantInt::get(Type::getInt32Ty(M->getContext()),3),
            ConstantInt::get(Type::getInt32Ty(M->getContext()),1)
        };
        ArrayRef<Type *> art = {
            Type::getInt8PtrTy(M->getContext())
        };
        Function *fun = Intrinsic::getDeclaration(M, Intrinsic::prefetch, art);

        assert(fun);

        CallInst* call = CallInst::Create(fun,ar);

        call->insertBefore(cast->getParent()->getTerminator());




        } else if(PHINode * pn = dyn_cast<PHINode>(z)) {

        Value* v = getOddPhiFirst(LI.getLoopFor(pn->getParent()),pn);
        //assert(v);
        if(v) {
            if(Instruction* ins = dyn_cast<Instruction>(v)) v = Transforms.lookup(ins);
            Transforms.insert(std::pair<Instruction*,Value*>(z,v));
        } else {
            Transforms.insert(std::pair<Instruction*,Value*>(z,z));
        }
        }
        else {
```

```
        Instruction* n = z->clone();

        Use* u = n->getOperandList();
        int64_t size = n->getNumOperands();
        for(int64_t x = 0; x<size; x++) {
            Value* v = u[x].get();
            assert(v);
            if(Instruction* t = dyn_cast<Instruction>(v)) {
        if(Transforms.count(t)) {
            n->setOperand(x,Transforms.lookup(t));
        }
            }
        }

        n->insertBefore(Loads[x]);

        bool changed = true;
        while(changed && LI.getLoopFor(Phis[x]->getParent()) != LI.getLoopFor(n->getParent())) {
            changed = false;

            makeLoopInvariantSpec(n,changed,LI.getLoopFor(n->getParent()));
            if(changed) LLVM_DEBUG(dbgs()<< "moved loop" << *n << "\n");
            else LLVM_DEBUG(dbgs()<< "not moved loop" << *n << "\n");

        }

        Transforms.insert(std::pair<Instruction*,Instruction*>(z,n));
        modified = true;
        }

    }


        }
    if (modified)
        return PreservedAnalyses::none();

    return PreservedAnalyses::all();
}

//} // namespace
```

Остальной код можно посмотреть в репозитории github:

https://github.com/valvenya/prefetch-thesis/

ПРИЛОЖЕНИЕ Б

Пример LLVM IR до и после оптимизации

До оптимизации

```
; ModuleID = 'randacc.c'
source_filename = "randacc.c"
target datalayout = "e-m:e-p270:32:32-p271:32:32-p272:64:64-i64:64-f80:128-n8:16:32:64-S128"
target triple = "x86_64-unknown-linux-gnu"

; Function Attrs: nofree nosync nounwind readnone uwtable
define dso_local i64 @HPCC_starts(i64 noundef %n) local_unnamed_addr #0 {
entry:
  %m2 = alloca [64 x i64], align 16
  %0 = bitcast [64 x i64]* %m2 to i8*
  call void @llvm.lifetime.start.p0i8(i64 512, i8* nonnull %0) #7
  %n.lobit = lshr i64 %n, 63
  %smax = tail call i64 @llvm.smax.i64(i64 %n, i64 0)
  %1 = add i64 %n.lobit, %n
  %2 = sub i64 %smax, %1
  %3 = udiv i64 %2, 1317624576693539401
  %4 = add nuw nsw i64 %n.lobit, %3
  %5 = mul i64 %4, 1317624576693539401
  %6 = add i64 %5, %n
  %7 = add i64 %6, 1317624576693539400
  %smin = tail call i64 @llvm.smin.i64(i64 %6, i64 1317624576693539401)
  %8 = sub i64 %7, %smin
  %.fr = freeze i64 %8
  %9 = urem i64 %.fr, 1317624576693539401
  %.neg = sub i64 %9, %.fr
  %10 = add i64 %.neg, %6
  %cmp5 = icmp eq i64 %10, 0
  br i1 %cmp5, label %cleanup, label %for.body

for.body:                                         ; preds = %entry, %for.body
  %indvars.iv = phi i64 [ %indvars.iv.next, %for.body ], [ 0, %entry ]
  %temp.086 = phi i64 [ %xor11, %for.body ], [ 1, %entry ]
  %arrayidx = getelementptr inbounds [64 x i64], [64 x i64]* %m2, i64 0, i64 %indvars.iv
  store i64 %temp.086, i64* %arrayidx, align 8, !tbaa !5
  %shl = shl i64 %temp.086, 1
  %cmp7 = icmp slt i64 %temp.086, 0
```

```
  %cond = select i1 %cmp7, i64 7, i64 0
  %xor = xor i64 %cond, %shl
  %shl8 = shl i64 %xor, 1
  %cmp9 = icmp slt i64 %xor, 0
  %cond10 = select i1 %cmp9, i64 7, i64 0
  %xor11 = xor i64 %cond10, %shl8
  %indvars.iv.next = add nuw nsw i64 %indvars.iv, 1
  %exitcond.not = icmp eq i64 %indvars.iv.next, 64
  br i1 %exitcond.not, label %for.body14, label %for.body, !llvm.loop !9

for.body14:                                       ; preds = %for.body, %for.inc17
  %i.187 = phi i32 [ %dec, %for.inc17 ], [ 62, %for.body ]
  %sh_prom = zext i32 %i.187 to i64
  %11 = shl i64 1, %sh_prom
  %12 = and i64 %11, %10
  %tobool.not = icmp eq i64 %12, 0
  br i1 %tobool.not, label %for.inc17, label %for.end18

for.inc17:                                        ; preds = %for.body14
  %dec = add nsw i32 %i.187, -1
  %cmp13.not = icmp eq i32 %i.187, 0
  br i1 %cmp13.not, label %for.end18, label %for.body14, !llvm.loop !12

for.end18:                                        ; preds = %for.body14, %for.inc17
  %i.1.lcssa = phi i32 [ %i.187, %for.body14 ], [ -1, %for.inc17 ]
  %cmp2090 = icmp sgt i32 %i.1.lcssa, 0
  br i1 %cmp2090, label %for.cond22.preheader.preheader, label %cleanup

for.cond22.preheader.preheader:                   ; preds = %for.end18
  %13 = zext i32 %i.1.lcssa to i64
  br label %for.cond22.preheader

for.cond22.preheader:                             ; preds = %for.cond22.preheader.preheader,
%for.end36
  %indvars.iv98 = phi i64 [ %13, %for.cond22.preheader.preheader ], [ %indvars.iv.next99,
%for.end36 ]
  %ran.092 = phi i64 [ 2, %for.cond22.preheader.preheader ], [ %ran.1, %for.end36 ]
  br label %for.body24

for.body24:                                       ; preds = %for.cond22.preheader, %for.inc34
  %indvars.iv94 = phi i64 [ 0, %for.cond22.preheader ], [ %indvars.iv.next95, %for.inc34 ]
  %temp.189 = phi i64 [ 0, %for.cond22.preheader ], [ %temp.2, %for.inc34 ]
```

```
  %14 = shl i64 1, %indvars.iv94
  %15 = and i64 %14, %ran.092
  %tobool28.not = icmp eq i64 %15, 0
  br i1 %tobool28.not, label %for.inc34, label %if.then29

if.then29:                                        ; preds = %for.body24
  %arrayidx31 = getelementptr inbounds [64 x i64], [64 x i64]* %m2, i64 0, i64 %indvars.iv94
  %16 = load i64, i64* %arrayidx31, align 8, !tbaa !5
  %xor32 = xor i64 %16, %temp.189
  br label %for.inc34

for.inc34:                                        ; preds = %for.body24, %if.then29
  %temp.2 = phi i64 [ %xor32, %if.then29 ], [ %temp.189, %for.body24 ]
  %indvars.iv.next95 = add nuw nsw i64 %indvars.iv94, 1
  %exitcond97.not = icmp eq i64 %indvars.iv.next95, 64
  br i1 %exitcond97.not, label %for.end36, label %for.body24, !llvm.loop !13

for.end36:                                        ; preds = %for.inc34
  %indvars.iv.next99 = add nsw i64 %indvars.iv98, -1
  %sh_prom38 = and i64 %indvars.iv.next99, 4294967295
  %17 = shl i64 1, %sh_prom38
  %18 = and i64 %17, %10
  %tobool41.not = icmp eq i64 %18, 0
  %shl43 = shl i64 %temp.2, 1
  %cmp44 = icmp slt i64 %temp.2, 0
  %cond45 = select i1 %cmp44, i64 7, i64 0
  %xor46 = xor i64 %cond45, %shl43
  %ran.1 = select i1 %tobool41.not, i64 %temp.2, i64 %xor46
  %cmp20 = icmp sgt i64 %indvars.iv98, 1
  br i1 %cmp20, label %for.cond22.preheader, label %cleanup, !llvm.loop !14

cleanup:                                          ; preds = %for.end36, %for.end18, %entry
  %retval.0 = phi i64 [ 1, %entry ], [ 2, %for.end18 ], [ %ran.1, %for.end36 ]
  call void @llvm.lifetime.end.p0i8(i64 512, i8* nonnull %0) #7
  ret i64 %retval.0
}

; Function Attrs: argmemonly mustprogress nocallback nofree nosync nounwind willreturn
declare void @llvm.lifetime.start.p0i8(i64 immarg, i8* nocapture) #1

; Function Attrs: argmemonly mustprogress nocallback nofree nosync nounwind willreturn
declare void @llvm.lifetime.end.p0i8(i64 immarg, i8* nocapture) #1
```

```
; Function Attrs: nounwind uwtable
define dso_local i32 @main(i32 noundef %argc, i8** nocapture noundef readonly %argv)
local_unnamed_addr #2 {
entry:
  %ran.i = alloca [128 x i64], align 16
  %arrayidx = getelementptr inbounds i8*, i8** %argv, i64 1
  %0 = load i8*, i8** %arrayidx, align 8, !tbaa !15
  %call.i = tail call i64 @strtol(i8* nocapture noundef nonnull %0, i8** noundef null, i32 noundef
10) #7
  %conv.i = trunc i64 %call.i to i32
  %conv = sitofp i32 %conv.i to double
  %div = fmul double %conv, 1.250000e-01
  br label %for.cond

for.cond:                                         ; preds = %for.cond, %entry
  %totalMem.0.in = phi double [ %div, %entry ], [ %totalMem.0, %for.cond ]
  %TableSize.0 = phi i64 [ 1, %entry ], [ %shl, %for.cond ]
  %totalMem.0 = fmul double %totalMem.0.in, 5.000000e-01
  %cmp = fcmp ult double %totalMem.0, 1.000000e+00
  %shl = shl i64 %TableSize.0, 1
  br i1 %cmp, label %for.end, label %for.cond, !llvm.loop !17

for.end:                                          ; preds = %for.cond
  %mul3 = shl i64 %TableSize.0, 3
  %call4 = tail call noalias i8* @malloc(i64 noundef %mul3) #8
  %1 = bitcast i8* %call4 to i64*
  %tobool.not = icmp eq i8* %call4, null
  br i1 %tobool.not, label %cleanup, label %for.cond5.preheader

for.cond5.preheader:                              ; preds = %for.end
  %cmp637.not = icmp eq i64 %TableSize.0, 0
  br i1 %cmp637.not, label %for.end12, label %for.body8

for.body8:                                        ; preds = %for.cond5.preheader, %for.body8
  %i.038 = phi i64 [ %inc11, %for.body8 ], [ 0, %for.cond5.preheader ]
  %arrayidx9 = getelementptr inbounds i64, i64* %1, i64 %i.038
  store i64 %i.038, i64* %arrayidx9, align 8, !tbaa !5
  %inc11 = add nuw i64 %i.038, 1
  %exitcond.not = icmp eq i64 %inc11, %TableSize.0
  br i1 %exitcond.not, label %for.end12, label %for.body8, !llvm.loop !18
```

```
for.end12:                                        ; preds = %for.body8, %for.cond5.preheader
  %2 = bitcast [128 x i64]* %ran.i to i8*
  call void @llvm.lifetime.start.p0i8(i64 1024, i8* nonnull %2) #7
  %3 = lshr i64 %TableSize.0, 5
  %div48.i = and i64 %3, 144115188075855871
  br label %for.body.i

for.cond2.preheader.i:                            ; preds = %for.body.i
  %4 = and i64 %TableSize.0, 4611686018427387872
  %cmp551.not.i = icmp eq i64 %4, 0
  br i1 %cmp551.not.i, label %RandomAccessUpdate.exit, label %for.cond8.preheader.lr.ph.i

for.cond8.preheader.lr.ph.i:                      ; preds = %for.cond2.preheader.i
  %sub.i = add i64 %TableSize.0, -1
  br label %for.cond8.preheader.i

for.body.i:                                       ; preds = %for.body.i, %for.end12
  %indvars.iv.i = phi i64 [ 0, %for.end12 ], [ %indvars.iv.next.i, %for.body.i ]
  %mul1.i = mul nuw i64 %indvars.iv.i, %div48.i
  %call.i36 = tail call i64 @HPCC_starts(i64 noundef %mul1.i)
  %arrayidx.i = getelementptr inbounds [128 x i64], [128 x i64]* %ran.i, i64 0, i64 %indvars.iv.i
  store i64 %call.i36, i64* %arrayidx.i, align 8, !tbaa !5
  %indvars.iv.next.i = add nuw nsw i64 %indvars.iv.i, 1
  %exitcond.not.i = icmp eq i64 %indvars.iv.next.i, 128
  br i1 %exitcond.not.i, label %for.cond2.preheader.i, label %for.body.i, !llvm.loop !19

for.cond8.preheader.i:                            ; preds = %for.inc29.i,
%for.cond8.preheader.lr.ph.i
  %i.052.i = phi i64 [ 0, %for.cond8.preheader.lr.ph.i ], [ %inc30.i, %for.inc29.i ]
  br label %for.body11.i

for.body11.i:                                     ; preds = %for.body11.i, %for.cond8.preheader.i
  %indvars.iv54.i = phi i64 [ 0, %for.cond8.preheader.i ], [ %indvars.iv.next55.i, %for.body11.i ]
  %arrayidx13.i = getelementptr inbounds [128 x i64], [128 x i64]* %ran.i, i64 0, i64
%indvars.iv54.i
  %5 = load i64, i64* %arrayidx13.i, align 8, !tbaa !5
  %shl.i = shl i64 %5, 1
  %cmp16.i = icmp slt i64 %5, 0
  %cond.i = select i1 %cmp16.i, i64 7, i64 0
  %xor.i = xor i64 %cond.i, %shl.i
  store i64 %xor.i, i64* %arrayidx13.i, align 8, !tbaa !5
  %and.i = and i64 %xor.i, %sub.i
```

```
  %arrayidx24.i = getelementptr inbounds i64, i64* %1, i64 %and.i
  %6 = load i64, i64* %arrayidx24.i, align 8, !tbaa !5
  %xor25.i = xor i64 %xor.i, %6
  store i64 %xor25.i, i64* %arrayidx24.i, align 8, !tbaa !5
  %indvars.iv.next55.i = add nuw nsw i64 %indvars.iv54.i, 1
  %exitcond57.not.i = icmp eq i64 %indvars.iv.next55.i, 128
  br i1 %exitcond57.not.i, label %for.inc29.i, label %for.body11.i, !llvm.loop !20

for.inc29.i:                                      ; preds = %for.body11.i
  %inc30.i = add nuw nsw i64 %i.052.i, 1
  %exitcond58.not.i = icmp eq i64 %inc30.i, %div48.i
  br i1 %exitcond58.not.i, label %RandomAccessUpdate.exit, label %for.cond8.preheader.i,
!llvm.loop !21

RandomAccessUpdate.exit:                          ; preds = %for.inc29.i, %for.cond2.preheader.i
  call void @llvm.lifetime.end.p0i8(i64 1024, i8* nonnull %2) #7
  tail call void @free(i8* noundef %call4) #7
  br label %cleanup

cleanup:                                          ; preds = %for.end, %RandomAccessUpdate.exit
  %retval.0 = phi i32 [ 0, %RandomAccessUpdate.exit ], [ 1, %for.end ]
  ret i32 %retval.0
}

; Function Attrs: inaccessiblememonly mustprogress nofree nounwind willreturn
allockind("alloc,uninitialized") allocsize(0)
declare noalias noundef i8* @malloc(i64 noundef) local_unnamed_addr #3

; Function Attrs: inaccessiblemem_or_argmemonly mustprogress nounwind willreturn allockind("free")
declare void @free(i8* allocptr nocapture noundef) local_unnamed_addr #4

; Function Attrs: mustprogress nofree nounwind willreturn
declare i64 @strtol(i8* noundef readonly, i8** nocapture noundef, i32 noundef) local_unnamed_addr
#5

; Function Attrs: nocallback nofree nosync nounwind readnone speculatable willreturn
declare i64 @llvm.smax.i64(i64, i64) #6

; Function Attrs: nocallback nofree nosync nounwind readnone speculatable willreturn
declare i64 @llvm.smin.i64(i64, i64) #6
```

```
attributes #0 = { nofree nosync nounwind readnone uwtable "frame-pointer"="none" "min-legal-vector-width"="0" "no-trapping-math"="true" "stack-protector-buffer-size"="8" "target-cpu"="x86-64" "target-features"="+cx8,+fxsr,+mmx,+sse,+sse2,+x87" "tune-cpu"="generic" }
attributes #1 = { argmemonly mustprogress nocallback nofree nosync nounwind willreturn }
attributes #2 = { nounwind uwtable "frame-pointer"="none" "min-legal-vector-width"="0" "no-trapping-math"="true" "stack-protector-buffer-size"="8" "target-cpu"="x86-64" "target-features"="+cx8,+fxsr,+mmx,+sse,+sse2,+x87" "tune-cpu"="generic" }
attributes #3 = { inaccessiblememonly mustprogress nofree nounwind willreturn allockind("alloc,uninitialized") allocsize(0) "alloc-family"="malloc" "frame-pointer"="none" "no-trapping-math"="true" "stack-protector-buffer-size"="8" "target-cpu"="x86-64" "target-features"="+cx8,+fxsr,+mmx,+sse,+sse2,+x87" "tune-cpu"="generic" }
attributes #4 = { inaccessiblemem_or_argmemonly mustprogress nounwind willreturn allockind("free") "alloc-family"="malloc" "frame-pointer"="none" "no-trapping-math"="true" "stack-protector-buffer-size"="8" "target-cpu"="x86-64" "target-features"="+cx8,+fxsr,+mmx,+sse,+sse2,+x87" "tune-cpu"="generic" }
attributes #5 = { mustprogress nofree nounwind willreturn "frame-pointer"="none" "no-trapping-math"="true" "stack-protector-buffer-size"="8" "target-cpu"="x86-64" "target-features"="+cx8,+fxsr,+mmx,+sse,+sse2,+x87" "tune-cpu"="generic" }
attributes #6 = { nocallback nofree nosync nounwind readnone speculatable willreturn }
attributes #7 = { nounwind }
attributes #8 = { nounwind allocsize(0) }

!llvm.module.flags = !{!0, !1, !2, !3}
!llvm.ident = !{!4}

!0 = !{i32 1, !"wchar_size", i32 4}
!1 = !{i32 7, !"PIC Level", i32 2}
!2 = !{i32 7, !"PIE Level", i32 2}
!3 = !{i32 7, !"uwtable", i32 2}
!4 = !{!"clang version 15.0.6 (https://github.com/llvm/llvm-project.git 088f33605d8a61ff519c580a71b1dd57d16a03f8)"}
!5 = !{!6, !6, i64 0}
!6 = !{!"long", !7, i64 0}
!7 = !{!"omnipotent char", !8, i64 0}
!8 = !{!"Simple C/C++ TBAA"}
!9 = distinct !{!9, !10, !11}
!10 = !{!"llvm.loop.mustprogress"}
!11 = !{!"llvm.loop.unroll.disable"}
!12 = distinct !{!12, !10, !11}
!13 = distinct !{!13, !10, !11}
!14 = distinct !{!14, !10, !11}
!15 = !{!16, !16, i64 0}
```

```
!16 = !{!"any pointer", !7, i64 0}
!17 = distinct !{!17, !10, !11}
!18 = distinct !{!18, !10, !11}
!19 = distinct !{!19, !10, !11}
!20 = distinct !{!20, !10, !11}
!21 = distinct !{!21, !10, !11}
```

После оптимизации

```
; ModuleID = 'randacc.ll'
source_filename = "randacc.c"
target datalayout = "e-m:e-p270:32:32-p271:32:32-p272:64:64-i64:64-f80:128-n8:16:32:64-S128"
target triple = "x86_64-unknown-linux-gnu"

; Function Attrs: nofree nosync nounwind readnone uwtable
define dso_local i64 @HPCC_starts(i64 noundef %n) local_unnamed_addr #0 {
entry:
  %m2 = alloca [64 x i64], align 16
  %0 = bitcast [64 x i64]* %m2 to i8*
  call void @llvm.lifetime.start.p0i8(i64 512, i8* nonnull %0) #8
  %n.lobit = lshr i64 %n, 63
  %smax = tail call i64 @llvm.smax.i64(i64 %n, i64 0)
  %1 = add i64 %n.lobit, %n
  %2 = sub i64 %smax, %1
  %3 = udiv i64 %2, 1317624576693539401
  %4 = add nuw nsw i64 %n.lobit, %3
  %5 = mul i64 %4, 1317624576693539401
  %6 = add i64 %5, %n
  %7 = add i64 %6, 1317624576693539400
  %smin = tail call i64 @llvm.smin.i64(i64 %6, i64 1317624576693539401)
  %8 = sub i64 %7, %smin
  %.fr = freeze i64 %8
  %9 = urem i64 %.fr, 1317624576693539401
  %.neg = sub i64 %9, %.fr
  %10 = add i64 %.neg, %6
  %cmp5 = icmp eq i64 %10, 0
  br i1 %cmp5, label %cleanup, label %for.body

for.body:                                         ; preds = %for.body, %entry
  %indvars.iv = phi i64 [ %indvars.iv.next, %for.body ], [ 0, %entry ]
  %temp.086 = phi i64 [ %xor11, %for.body ], [ 1, %entry ]
  %arrayidx = getelementptr inbounds [64 x i64], [64 x i64]* %m2, i64 0, i64 %indvars.iv
```

```
  store i64 %temp.086, i64* %arrayidx, align 8, !tbaa !5
  %shl = shl i64 %temp.086, 1
  %cmp7 = icmp slt i64 %temp.086, 0
  %cond = select i1 %cmp7, i64 7, i64 0
  %xor = xor i64 %cond, %shl
  %shl8 = shl i64 %xor, 1
  %cmp9 = icmp slt i64 %xor, 0
  %cond10 = select i1 %cmp9, i64 7, i64 0
  %xor11 = xor i64 %cond10, %shl8
  %indvars.iv.next = add nuw nsw i64 %indvars.iv, 1
  %exitcond.not = icmp eq i64 %indvars.iv.next, 64
  br i1 %exitcond.not, label %for.body14, label %for.body, !llvm.loop !9

for.body14:                                       ; preds = %for.inc17, %for.body
  %i.187 = phi i32 [ %dec, %for.inc17 ], [ 62, %for.body ]
  %sh_prom = zext i32 %i.187 to i64
  %11 = shl i64 1, %sh_prom
  %12 = and i64 %11, %10
  %tobool.not = icmp eq i64 %12, 0
  br i1 %tobool.not, label %for.inc17, label %for.end18

for.inc17:                                        ; preds = %for.body14
  %dec = add nsw i32 %i.187, -1
  %cmp13.not = icmp eq i32 %i.187, 0
  br i1 %cmp13.not, label %for.end18, label %for.body14, !llvm.loop !12

for.end18:                                        ; preds = %for.inc17, %for.body14
  %i.1.lcssa = phi i32 [ %i.187, %for.body14 ], [ -1, %for.inc17 ]
  %cmp2090 = icmp sgt i32 %i.1.lcssa, 0
  br i1 %cmp2090, label %for.cond22.preheader.preheader, label %cleanup

for.cond22.preheader.preheader:                   ; preds = %for.end18
  %13 = zext i32 %i.1.lcssa to i64
  br label %for.cond22.preheader

for.cond22.preheader:                             ; preds = %for.end36,
%for.cond22.preheader.preheader
  %indvars.iv98 = phi i64 [ %13, %for.cond22.preheader.preheader ], [ %indvars.iv.next99,
%for.end36 ]
  %ran.092 = phi i64 [ 2, %for.cond22.preheader.preheader ], [ %ran.1, %for.end36 ]
  br label %for.body24
```

```
for.body24:                                       ; preds = %for.inc34, %for.cond22.preheader
  %indvars.iv94 = phi i64 [ 0, %for.cond22.preheader ], [ %indvars.iv.next95, %for.inc34 ]
  %temp.189 = phi i64 [ 0, %for.cond22.preheader ], [ %temp.2, %for.inc34 ]
  %14 = shl i64 1, %indvars.iv94
  %15 = and i64 %14, %ran.092
  %tobool28.not = icmp eq i64 %15, 0
  br i1 %tobool28.not, label %for.inc34, label %if.then29

if.then29:                                        ; preds = %for.body24
  %arrayidx31 = getelementptr inbounds [64 x i64], [64 x i64]* %m2, i64 0, i64 %indvars.iv94
  %16 = load i64, i64* %arrayidx31, align 8, !tbaa !5
  %xor32 = xor i64 %16, %temp.189
  br label %for.inc34

for.inc34:                                        ; preds = %if.then29, %for.body24
  %temp.2 = phi i64 [ %xor32, %if.then29 ], [ %temp.189, %for.body24 ]
  %indvars.iv.next95 = add nuw nsw i64 %indvars.iv94, 1
  %exitcond97.not = icmp eq i64 %indvars.iv.next95, 64
  br i1 %exitcond97.not, label %for.end36, label %for.body24, !llvm.loop !13

for.end36:                                        ; preds = %for.inc34
  %indvars.iv.next99 = add nsw i64 %indvars.iv98, -1
  %sh_prom38 = and i64 %indvars.iv.next99, 4294967295
  %17 = shl i64 1, %sh_prom38
  %18 = and i64 %17, %10
  %tobool41.not = icmp eq i64 %18, 0
  %shl43 = shl i64 %temp.2, 1
  %cmp44 = icmp slt i64 %temp.2, 0
  %cond45 = select i1 %cmp44, i64 7, i64 0
  %xor46 = xor i64 %cond45, %shl43
  %ran.1 = select i1 %tobool41.not, i64 %temp.2, i64 %xor46
  %cmp20 = icmp sgt i64 %indvars.iv98, 1
  br i1 %cmp20, label %for.cond22.preheader, label %cleanup, !llvm.loop !14

cleanup:                                          ; preds = %for.end36, %for.end18, %entry
  %retval.0 = phi i64 [ 1, %entry ], [ 2, %for.end18 ], [ %ran.1, %for.end36 ]
  call void @llvm.lifetime.end.p0i8(i64 512, i8* nonnull %0) #8
  ret i64 %retval.0
}

; Function Attrs: argmemonly nocallback nofree nosync nounwind willreturn
declare void @llvm.lifetime.start.p0i8(i64 immarg, i8* nocapture) #1
```

```
; Function Attrs: argmemonly nocallback nofree nosync nounwind willreturn
declare void @llvm.lifetime.end.p0i8(i64 immarg, i8* nocapture) #1

; Function Attrs: nounwind uwtable
define dso_local i32 @main(i32 noundef %argc, i8** nocapture noundef readonly %argv)
local_unnamed_addr #2 {
entry:
  %ran.i = alloca [128 x i64], align 16
  %arrayidx = getelementptr inbounds i8*, i8** %argv, i64 1
  %0 = load i8*, i8** %arrayidx, align 8, !tbaa !15
  %call.i = tail call i64 @strtol(i8* nocapture noundef nonnull %0, i8** noundef null, i32 noundef
10) #8
  %conv.i = trunc i64 %call.i to i32
  %conv = sitofp i32 %conv.i to double
  %div = fmul double %conv, 1.250000e-01
  br label %for.cond

for.cond:                                         ; preds = %for.cond, %entry
  %totalMem.0.in = phi double [ %div, %entry ], [ %totalMem.0, %for.cond ]
  %TableSize.0 = phi i64 [ 1, %entry ], [ %shl, %for.cond ]
  %totalMem.0 = fmul double %totalMem.0.in, 5.000000e-01
  %cmp = fcmp ult double %totalMem.0, 1.000000e+00
  %shl = shl i64 %TableSize.0, 1
  br i1 %cmp, label %for.end, label %for.cond, !llvm.loop !17

for.end:                                          ; preds = %for.cond
  %mul3 = shl i64 %TableSize.0, 3
  %call4 = tail call noalias i8* @malloc(i64 noundef %mul3) #9
  %1 = bitcast i8* %call4 to i64*
  %tobool.not = icmp eq i8* %call4, null
  br i1 %tobool.not, label %cleanup, label %for.cond5.preheader

for.cond5.preheader:                              ; preds = %for.end
  %cmp637.not = icmp eq i64 %TableSize.0, 0
  br i1 %cmp637.not, label %for.end12, label %for.body8

for.body8:                                        ; preds = %for.body8, %for.cond5.preheader
  %i.038 = phi i64 [ %inc11, %for.body8 ], [ 0, %for.cond5.preheader ]
  %arrayidx9 = getelementptr inbounds i64, i64* %1, i64 %i.038
  store i64 %i.038, i64* %arrayidx9, align 8, !tbaa !5
  %inc11 = add nuw i64 %i.038, 1
```

```
  %exitcond.not = icmp eq i64 %inc11, %TableSize.0
  br i1 %exitcond.not, label %for.end12, label %for.body8, !llvm.loop !18

for.end12:                                        ; preds = %for.body8, %for.cond5.preheader
  %2 = bitcast [128 x i64]* %ran.i to i8*
  call void @llvm.lifetime.start.p0i8(i64 1024, i8* nonnull %2) #8
  %3 = lshr i64 %TableSize.0, 5
  %div48.i = and i64 %3, 144115188075855871
  br label %for.body.i

for.cond2.preheader.i:                            ; preds = %for.body.i
  %4 = and i64 %TableSize.0, 4611686018427387872
  %cmp551.not.i = icmp eq i64 %4, 0
  br i1 %cmp551.not.i, label %RandomAccessUpdate.exit, label %for.cond8.preheader.lr.ph.i

for.cond8.preheader.lr.ph.i:                      ; preds = %for.cond2.preheader.i
  %sub.i = add i64 %TableSize.0, -1
  br label %for.cond8.preheader.i

for.body.i:                                       ; preds = %for.body.i, %for.end12
  %indvars.iv.i = phi i64 [ 0, %for.end12 ], [ %indvars.iv.next.i, %for.body.i ]
  %mul1.i = mul nuw i64 %indvars.iv.i, %div48.i
  %call.i36 = tail call i64 @HPCC_starts(i64 noundef %mul1.i)
  %arrayidx.i = getelementptr inbounds [128 x i64], [128 x i64]* %ran.i, i64 0, i64 %indvars.iv.i
  store i64 %call.i36, i64* %arrayidx.i, align 8, !tbaa !5
  %indvars.iv.next.i = add nuw nsw i64 %indvars.iv.i, 1
  %exitcond.not.i = icmp eq i64 %indvars.iv.next.i, 128
  br i1 %exitcond.not.i, label %for.cond2.preheader.i, label %for.body.i, !llvm.loop !19

for.cond8.preheader.i:                            ; preds = %for.inc29.i,
%for.cond8.preheader.lr.ph.i
  %i.052.i = phi i64 [ 0, %for.cond8.preheader.lr.ph.i ], [ %inc30.i, %for.inc29.i ]
  br label %for.body11.i

for.body11.i:                                     ; preds = %for.body11.i, %for.cond8.preheader.i
  %indvars.iv54.i = phi i64 [ 0, %for.cond8.preheader.i ], [ %indvars.iv.next55.i, %for.body11.i ]
  %arrayidx13.i = getelementptr inbounds [128 x i64], [128 x i64]* %ran.i, i64 0, i64
%indvars.iv54.i
  %5 = add i64 %indvars.iv54.i, 64
  %6 = getelementptr inbounds [128 x i64], [128 x i64]* %ran.i, i64 0, i64 %5
  %7 = bitcast i64* %6 to i8*
  %8 = load i64, i64* %arrayidx13.i, align 8, !tbaa !5
```

```
  %shl.i = shl i64 %8, 1
  %cmp16.i = icmp slt i64 %8, 0
  %cond.i = select i1 %cmp16.i, i64 7, i64 0
  %xor.i = xor i64 %cond.i, %shl.i
  store i64 %xor.i, i64* %arrayidx13.i, align 8, !tbaa !5
  %and.i = and i64 %xor.i, %sub.i
  %arrayidx24.i = getelementptr inbounds i64, i64* %1, i64 %and.i
  %9 = add i64 %indvars.iv54.i, 32
  %10 = icmp slt i64 127, %9
  %11 = select i1 %10, i64 127, i64 %9
  %12 = getelementptr inbounds [128 x i64], [128 x i64]* %ran.i, i64 0, i64 %11
  %13 = load i64, i64* %12, align 8, !tbaa !5
  %14 = shl i64 %13, 1
  %15 = icmp slt i64 %13, 0
  %16 = select i1 %15, i64 7, i64 0
  %17 = xor i64 %16, %14
  %18 = and i64 %17, %sub.i
  %19 = getelementptr inbounds i64, i64* %1, i64 %18
  %20 = bitcast i64* %19 to i8*
  %21 = load i64, i64* %arrayidx24.i, align 8, !tbaa !5
  %xor25.i = xor i64 %xor.i, %21
  store i64 %xor25.i, i64* %arrayidx24.i, align 8, !tbaa !5
  %indvars.iv.next55.i = add nuw nsw i64 %indvars.iv54.i, 1
  %exitcond57.not.i = icmp eq i64 %indvars.iv.next55.i, 128
  call void @llvm.prefetch.p0i8(i8* %7, i32 0, i32 3, i32 1)
  call void @llvm.prefetch.p0i8(i8* %20, i32 0, i32 3, i32 1)
  br i1 %exitcond57.not.i, label %for.inc29.i, label %for.body11.i, !llvm.loop !20

for.inc29.i:                                      ; preds = %for.body11.i
  %inc30.i = add nuw nsw i64 %i.052.i, 1
  %exitcond58.not.i = icmp eq i64 %inc30.i, %div48.i
  br i1 %exitcond58.not.i, label %RandomAccessUpdate.exit, label %for.cond8.preheader.i,
!llvm.loop !21

RandomAccessUpdate.exit:                          ; preds = %for.inc29.i, %for.cond2.preheader.i
  call void @llvm.lifetime.end.p0i8(i64 1024, i8* nonnull %2) #8
  tail call void @free(i8* noundef %call4) #8
  br label %cleanup

cleanup:                                          ; preds = %RandomAccessUpdate.exit, %for.end
  %retval.0 = phi i32 [ 0, %RandomAccessUpdate.exit ], [ 1, %for.end ]
  ret i32 %retval.0
```

```
}

; Function Attrs: inaccessiblememonly mustprogress nofree nounwind willreturn
allockind("alloc,uninitialized") allocsize(0)
declare noalias noundef i8* @malloc(i64 noundef) local_unnamed_addr #3

; Function Attrs: inaccessiblemem_or_argmemonly mustprogress nounwind willreturn allockind("free")
declare void @free(i8* allocptr nocapture noundef) local_unnamed_addr #4

; Function Attrs: mustprogress nofree nounwind willreturn
declare i64 @strtol(i8* noundef readonly, i8** nocapture noundef, i32 noundef) local_unnamed_addr
#5

; Function Attrs: nocallback nofree nosync nounwind readnone speculatable willreturn
declare i64 @llvm.smax.i64(i64, i64) #6

; Function Attrs: nocallback nofree nosync nounwind readnone speculatable willreturn
declare i64 @llvm.smin.i64(i64, i64) #6

; Function Attrs: inaccessiblemem_or_argmemonly nocallback nofree nosync nounwind willreturn
declare void @llvm.prefetch.p0i8(i8* nocapture readonly, i32 immarg, i32 immarg, i32) #7

attributes #0 = { nofree nosync nounwind readnone uwtable "frame-pointer"="none" "min-legal-
vector-width"="0" "no-trapping-math"="true" "stack-protector-buffer-size"="8" "target-cpu"="x86-
64" "target-features"="+cx8,+fxsr,+mmx,+sse,+sse2,+x87" "tune-cpu"="generic" }
attributes #1 = { argmemonly nocallback nofree nosync nounwind willreturn }
attributes #2 = { nounwind uwtable "frame-pointer"="none" "min-legal-vector-width"="0" "no-
trapping-math"="true" "stack-protector-buffer-size"="8" "target-cpu"="x86-64" "target-
features"="+cx8,+fxsr,+mmx,+sse,+sse2,+x87" "tune-cpu"="generic" }
attributes #3 = { inaccessiblememonly mustprogress nofree nounwind willreturn
allockind("alloc,uninitialized") allocsize(0) "alloc-family"="malloc" "frame-pointer"="none" "no-
trapping-math"="true" "stack-protector-buffer-size"="8" "target-cpu"="x86-64" "target-
features"="+cx8,+fxsr,+mmx,+sse,+sse2,+x87" "tune-cpu"="generic" }
attributes #4 = { inaccessiblemem_or_argmemonly mustprogress nounwind willreturn allockind("free")
"alloc-family"="malloc" "frame-pointer"="none" "no-trapping-math"="true" "stack-protector-buffer-
size"="8" "target-cpu"="x86-64" "target-features"="+cx8,+fxsr,+mmx,+sse,+sse2,+x87" "tune-
cpu"="generic" }
attributes #5 = { mustprogress nofree nounwind willreturn "frame-pointer"="none" "no-trapping-
math"="true" "stack-protector-buffer-size"="8" "target-cpu"="x86-64" "target-
features"="+cx8,+fxsr,+mmx,+sse,+sse2,+x87" "tune-cpu"="generic" }
attributes #6 = { nocallback nofree nosync nounwind readnone speculatable willreturn }
attributes #7 = { inaccessiblemem_or_argmemonly nocallback nofree nosync nounwind willreturn }
```

```
attributes #8 = { nounwind }
attributes #9 = { nounwind allocsize(0) }

!llvm.module.flags = !{!0, !1, !2, !3}
!llvm.ident = !{!4}

!0 = !{i32 1, !"wchar_size", i32 4}
!1 = !{i32 7, !"PIC Level", i32 2}
!2 = !{i32 7, !"PIE Level", i32 2}
!3 = !{i32 7, !"uwtable", i32 2}
!4 = !{!"clang version 15.0.6 (https://github.com/llvm/llvm-project.git
088f33605d8a61ff519c580a71b1dd57d16a03f8)"}
!5 = !{!6, !6, i64 0}
!6 = !{!"long", !7, i64 0}
!7 = !{!"omnipotent char", !8, i64 0}
!8 = !{!"Simple C/C++ TBAA"}
!9 = distinct !{!9, !10, !11}
!10 = !{!"llvm.loop.mustprogress"}
!11 = !{!"llvm.loop.unroll.disable"}
!12 = distinct !{!12, !10, !11}
!13 = distinct !{!13, !10, !11}
!14 = distinct !{!14, !10, !11}
!15 = !{!16, !16, i64 0}
!16 = !{!"any pointer", !7, i64 0}
!17 = distinct !{!17, !10, !11}
!18 = distinct !{!18, !10, !11}
!19 = distinct !{!19, !10, !11}
!20 = distinct !{!20, !10, !11}
!21 = distinct !{!21, !10, !11}
```