МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по научно-исследовательской работе

Тема: Оптимизация доступа к кэш-памяти в вычислительных системах с иерархической структурой.

Студент гр. 7304 _____ Грибов А.В.

Руководитель _____ Пазников А.А.

Санкт-Петербург

2022

ЗАДАНИЕ

НА НАУЧНО-ИССЛЕДОВАТЕЛЬСУКЮ РАБОТУ

Студент Грибов А.В.

Группа 7304

Тема НИР: Оптимизация доступа к кэш-памяти в вычислительных системах с иерархической структурой.

Задание на практику: Оптимизировать доступ к кэш-памяти в вычислительных системах с иерархической структурой.

Сроки прохождения НИР: 01.09.2022 – 27.12.2022

Дата сдачи отчета: 27.12.2022

Дата защиты отчета: 27.12.2022

Студент _____ Грибов А.В.

Руководитель _____ Пазников А.А.

АННОТАЦИЯ

В данной работе были рассмотрены готовые варианты оптимизации кода. В них использовалась LLVM в качестве проекта программной инфраструктуры для создания компиляторов и сопутствующих им утилит. Так же в качестве методов оптимизации кода был изучен MPI для возможности обработки задач на нескольких потоках. На основе второго метода были разработаны программы, выполняющиеся в несколько потоков.

SUMMARY

In this paper, we considered off-the-shelf options for code optimization. They used LLVM as a software infrastructure project to create compilers and associated utilities. MPI was also explored as a method for code optimization to handle tasks on multiple threads. Based on the second method, programs running in multiple threads were developed.

СОДЕРЖАНИЕ

	Введение	4
1.	Постановка задачи	5
2.	Результат работы в осеннем семестре	0
3.	План работы на весенний семестр	0
	Заключение	0
	Список использованных источников	0

ВВЕДЕНИЕ

В данной работе были рассмотрены возможные методы получения информации о коде, который предстоит оптимизировать, а также инструмент оптимизации.

Цель работы - Оптимизировать доступ к кэш-памяти в вычислительных системах с иерархической структурой.

1. ПОСТАНОВКА ЗАДАЧИ

Актуальность: на реальных машинах теоретически сложные алгоритмы могут выполняться быстрее чем теоретически простые, например, хороший алгоритм, использующий оперативную память, будет хуже, чем плохой, но с возможностью сохранения всех данных в кэш памяти.

Проблема: для большинства программ использующих хорошие алгоритмы возможна оптимизация на кэш уровне, но не все компиляторы позволяют её обеспечить, поскольку под разные алгоритмы требуются различные оптимизации.

Цель: оптимизировать доступ к кэш-памяти в вычислительных системах с иерархической структурой.

Задачи, поставленные для достижения данной цели:

1. Выбор инструментов и технологий для реализации алгоритма для выбора наиболее подходящих оптимизаций;
2. Реализация алгоритма для выбора наиболее подходящих оптимизаций;
3. Оценка эффективности полученной реализации алгоритма для выбора наиболее подходящих оптимизаций

2. РЕЗУЛЬТАТЫ РАБОТЫ В ОСЕННЕМ СЕМЕСТРЕ

В качестве основного инструмента для выполнения поставленной в весеннем семестре задачи использовался LLVM. Это написанный на C++ проект для создания компиляторов, в связи с чем можно использовать его для создания собственных компиляторов, которые будут изменять код по заданным настройкам. Основная работа в весеннем семестре заключалась в изучении различных функций LLVM и их работы на тестовом коде[1], пример результата работы кода для LLVM представлен на Рисунке 1. На нём можно увидеть вывод названий функций, которые имеются в тестовой программе. Одну такую настройку для LLVM с последующим запуском будем называть проходом[2].

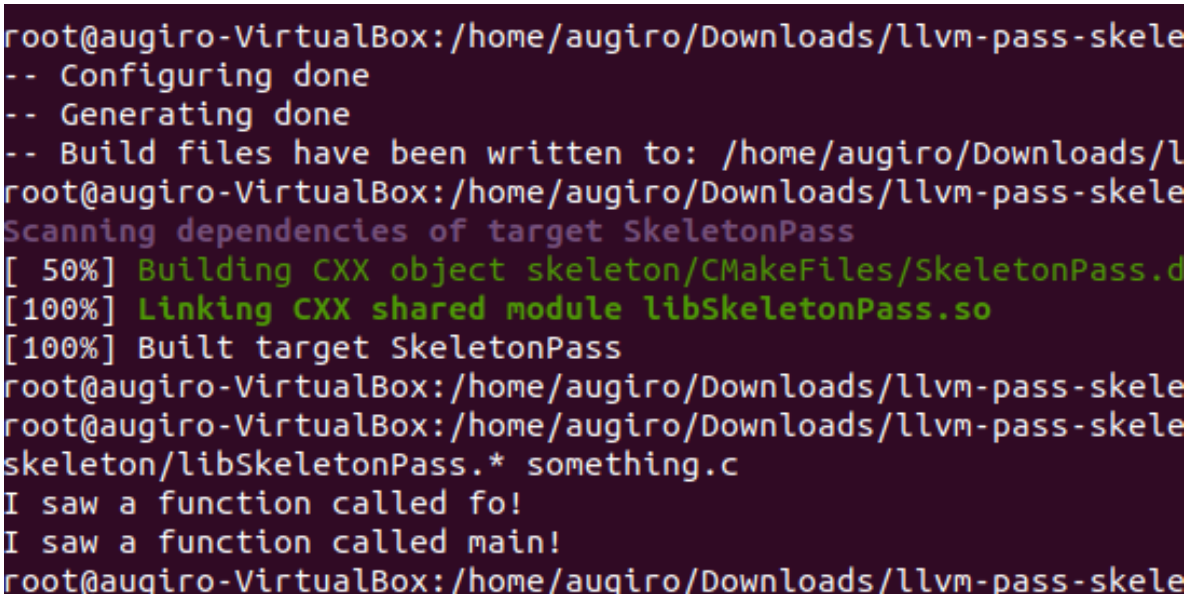

Рис. 1 – Устройство временной и пространственной локальности в памяти

Всего за весенний семестр было выполнено несколько различных проходов. В самом начале они просто позволяли посмотреть на тот или иной параметр в тестовом коде, к примеру количество GetElementPtr[3] в программе. Далее использовались проходы, которые изменяли какие-либо параметры тестовой программы. В частности, программа, которая меняла сложение на умножение, т.е. функции сложения в тестовой программе при её компиляции с помощью LLVM заменялись на умножение.

В осеннем семестре были изучены готовые решения оптимизации с использованием LLVM, например, трансформации разделения и слияния циклов. Зачастую оптимизации циклов не производятся в стандартных компиляторах, но именно эти оптимизации могут позволить сильно повлиять на производительность программы. Данное решение описывает множество различных оптимизаций циклов, но само исследование сводится к методу оптимизации посредством разделения и слияния циклов, поскольку они не так распространены и зачастую имеют дополнительные условия и проблемы. Было рассмотрено и развёртывание циклов, оно неплохо подходит для последующей оптимизации при помощи многопоточности. Изменение кода в данном случае заключается в увеличении шага итерации в цикле в K раз, таким образом тело цикла увеличивается в K раз, в случае если это независимые задачи их можно разбить по разным потокам для более быстрой обработки. Но при этом стоит учитывать, что интервал разбиения напрямую может влиять на скорость работы программы. Разделение циклов подразумевает за собой деление одного цикла на несколько производных. Это повышает локальность данных, в результате чего кэш-память быстрее их обрабатывает. Так же это позволяет запустить обработку производных циклов в разные потоки. Однако зависимые строки в теле изначального цикла разбить таким образом нельзя. Слияние циклов - это трансформация кода обратная разделению, зачастую она наоборот может ухудшить скорость работы кода, но в некоторых моментах ускоряет его. Оптимизации в таком случае будут идти с возможностью распараллеливания по инструкциям, как в случае с развёртыванием. Основная проблема таких оптимизаций заключается в проверке их необходимостей, порой она может занимать сильно больше сэкономленного времени, что делает её неэффективной.

Так же был изучен интерфейс передачи сообщений MPI(Message Passing Interface)[5]. Он позволяет достаточно легко организовать параллельные вычисления. На Рисунке 2 можно увидеть разбиение программы на несколько процессов[1]. Данный интерфейс даёт возможность одновременно

происходящим вычислениям общаться посредством сообщений. Само разбиение на процессы зависит от реализации программы. Проще всего его использовать, когда имеется несколько независимых однотипных задач, которые можно считать одновременно. Таким образом задача обработки программы при помощи MPI зачастую сводится к разбиению изначальных данных на независимые задачи. Если для этого необходимо каким-либо образом обработать изначальные данные, то в таких случаях, как правило, первый поток обрабатывает их, а затем общается с остальными, указывая им диапазон данных для работы. После запуска параллельно взаимодействующих процессов MPI[6] использует для общения коммуникаторы, сообщения и теги. Коммуникатор - это объект через который общается определённая группа процессов. Процессы передают сообщения, являющиеся набором данных, а тег - это целое неотрицательное число. В MPI передача данных в сообщениях является надёжной, но не обязательно при отправке одним процессом сообщения второй его получит. Результат составленной программы[1] можно увидеть на Рисунке 3, она в несколько процессов определяет числа в заданном промежутке, являющиеся простыми. Для этого первый процесс пересылает остальным введённое значение, пока те находятся в ожидании, затем каждый из процессов берёт свой интервал чисел и начинает его обрабатывать.

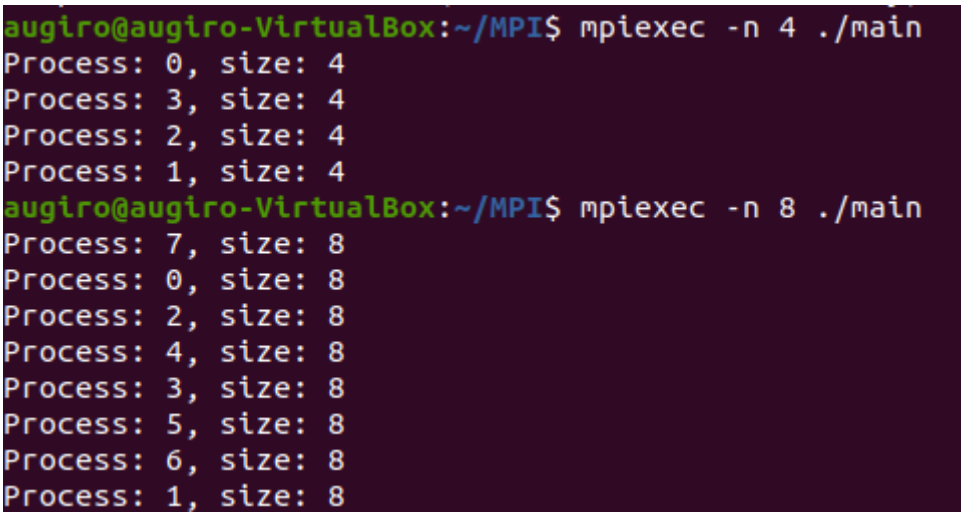

Рис. 2 – Разбиение на процессы с помощью MPI

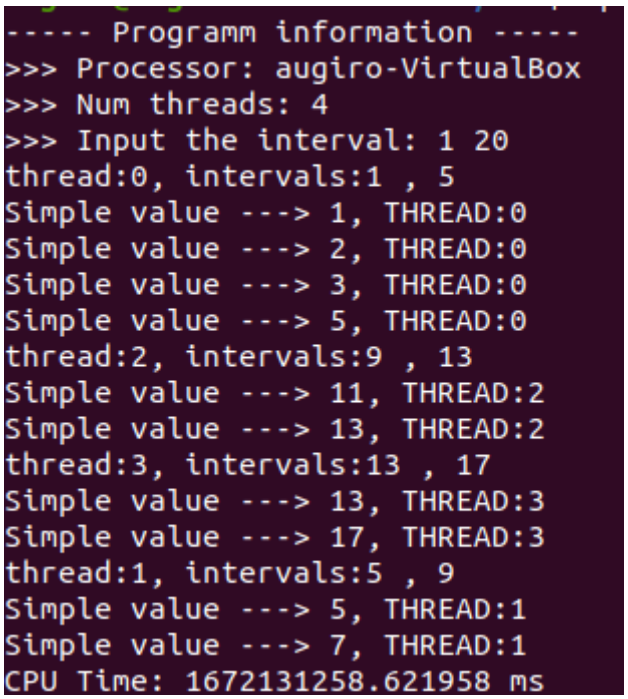

Рис. 3 – Результат работы многопоточной программы определяющей простые числа

3. ПЛАН РАБОТЫ НА ВЕСЕННИЙ СЕМЕСТР

В план работы на осенний семестр входит продолжение работы над проектом по шагам, которые были определены в прошлом весеннем семестре, а именно реализация алгоритма для наиболее подходящих оптимизаций и оценка эффективности данного алгоритма.

ЗАКЛЮЧЕНИЕ

В ходе работы над НИР в осеннем семестре были рассмотрены реализации оптимизаций с применением LLVM, а также было разработано несколько программ, которые могут работать в многопоточном режиме при помощи интерфейса MPI. В осеннем семестре планируется продолжение разработки алгоритма для оптимизаций согласно плану, который составлен в весеннем семестре прошлого учебного года.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Репозиторий разработанного алгоритма /
   URL: https://github.com/Augiro747/llvm-passww
2. Writing an LLVM Pass /
   URL:https://llvm.org/docs/WritingAnLLVMPass.html
3. The Often Misunderstood GEP Instruction /
   URL: https://llvm.org/docs/GetElementPtr.html
4. Machine code layout optimizations. /
   URL: https://easyperf.net/blog/2019/03/27/Machine-code-layout-optimizatoins
5. Message Passing Interface /
   URL: https://hpc.nmsu.edu/discovery/mpi/introduction/
6. MPICH
   URL: https://www.mpich.org/