Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Отчет по лабораторной работе № 5

По курсу "Анализ Алгоритмов"

Конвейерные вычисления

Студент:

Турсунов Жасурбек Рустамович

Группа: ИУ7-56Б

Преподователи:

Волкова Лилия Леонидовна
Строганов Юрий Владимирович

Москва, 2020 г.

Содержание

Введение

Параллельные вычисления используют для увеличения скорости исполнения программ. Ведь пока нет возможности сделать один очень быстрый процессор, который можно было бы сравнить с современными параллельными компьютерами. Конвейерная обработка данных является популярным приемом при работе с параллельными машинами.

В ходе лабораторной предстоит выполнить следующие задачи:

1. изучение основ конвейерной обработки данных;
2. получение практических навыков конвейерных вычислений;
3. экспериментальное подтверждение различий во временной эффективности реализаций при помощи разработанного программного обеспечения на материале замеров процессорного времени выполнения;
4. описание и обоснование полученных результатов в отчете о выполненной лабораторной работе.

1 Аналитическая часть

В данной части будут рассмотрены главные принципы конвейерной обработки и параллельных вычислений.

1.1 Общие сведения о конвейерной обработке

Конвейер - машина непрерывного транспорта, предназначенная для перемещения сыпучих, кусковых или штучных грузов.

Конвейерное производство - система поточной организации производства на основе конвейера, при которой оно разделено на простейшие короткие операции, а перемещение деталей осуществляется автоматически. Это такая организация выполнения операций над объектами, при которой весь процесс воздействия разделяется на последовательность стадий с целью повышения производительности путём одновременного независимого выполнения операций над несколькими объектами, проходящими различные стадии. Конвейером также называют средство продвижения объектов между стадиями при такой организации.[2]

Появилось в 1914 году на производстве Модели-Т на заводе Генри Форда и произвело революцию сначала в автомобилестроениии, а потом и во всей промышленности.

1.2 Параллельный алгоритм классического умножения матриц

Чтобы улучшить алгоритм, следует распараллелить ту часть алгоритма, которая содержит 3 вложенных цикла. Вычисление результата для каждой строки не зависит от результата выполнения умножения для других строк. Поэтому можно распараллелить часть кода, где происходят эти действия. Каждый поток будет выполнять вычисления определенных строк результирующей матрицы.

1.3 Параллельное программирование

При использовании многопроцессорных вычислительных систем с общей памятью обычно предполагается, что имеющиеся в составе системы процессоры обладают равной производительностью, являются равноправными при доступе к общей памяти, и время доступа к памяти является одинаковым (при одновременном доступе нескольких процессоров к одному и тому же элементу памяти очередность и синхронизация доступа обеспечивается на аппаратном уровне). Многопроцессорные системы подобного типа обычно именуются симметричными мультипроцессорами (symmetric multiprocessors, SMP).

Перечисленному выше набору предположений удовлетворяют также активно развиваемые в последнее время многоядерные процессоры, в которых каждое ядро представляет практически независимо функциони рующее вычислительное устройство. Для общности излагаемого учебного материала для упоминания одновременно и мультипроцессоров и много ядерных процессоров для обозначения одного вычислительного устройства.

Обычный подход при организации вычислений для многопроцессорных вычислительных систем с общей памятью – создание новых параллельных методов на основе обычных последовательных программ, в которых или автоматически компилятором, или непосредственно программистом выделяются участки независимых друг от друга вычислений. Возможности автоматического анализа программ для порождения параллельных вычислений достаточно ограничены, и второй подход является преобладающим. При этом для разработки параллельных программ могут применяться как новые алгоритмические языки, ориентированные на параллельное программирование, так и уже имеющиеся языки, расширенные некоторым набором операторов для параллельных вычислений.

Широко используемый подход состоит и в применении тех или иных библиотек, обеспечивающих определенный программный интерфейс (API) для разработки параллельных программ. В рамках такого подхода наиболее известны Windows Thread API. Однако первый способ применим только для ОС семейства Microsoft Windows, а второй вариант API является достаточно трудоемким для использования и имеет низкоуровневый характер. [3]

1.4 Организация взаимодействия параллельных потоков

Потоки исполняются в общем адресном пространстве параллельной программы. Как результат, взаимодействия параллельных потоков можно организовать через использование общих данных, являющихся доступными для всех потоков. Наиболее простая ситуация состоит в использовании общих данных только для чтения.

1.5 Описание метода

Конвейеризация - это техника, в результате которой задача разбивается на некоторое число подзадач, которые выполняются последовательно. Каждая подзадача выполняется на своем логическом устройстве. Все логические устройства соединяются последовательно таким образом, что выход i-ой ступени связан с выходом (i + 1)-ой ступени, все ступени работают одновременно. Множество ступеней называется конвейерном.

Выиграш во времени достигается при выполнении нескольких задач за счет параллельной работы ступеней, вовлекая на каждом такте новую задачу. Но в бесконвейерном подходе предсказать намного сложнее, и она может значительно различаться при разных данных.

1.6 Вывод

Были рассмотрены основы конвейерной обработки, технология параллельного программирования и организация взаимодействия параллельных потоков.

2 Конструкторская часть

Требования к вводу: Количество конвейеров должно быть больше 0.

Требования к программе при параллельной обработке:

1. объекты должны последовательно проходить конвейеры в заданном порядке;
2. конвейеры должны работать каждый в своем потоке;
3. конвейер должен завершать свою работу при поступлении специального элемента.

2.1 Организация обработки данных

На вход алгоритм получает одно число. Начальная функция пробрасывает число на уровень A. На уровне A, с помощью входного числа генерируются соответствующие числа для уровня B и пробрасываются в него. При этом уровень A начинает ожидать результата от уровня B. После его получения, числа суммируются в результат. Уровень B аналогичен уровню A - пробрасывает результат в C. Уровень C отправляет числа в функцию суммированию и возвращает результат в уровень B. На выходе получаем сумму чисел, дошедших до уровня C. Все действия выполняются асинхронно. Функция суммирования работает в 9 потоков.[1]

2.2 Вывод

В данном разделе была рассмотрена организация работы конвейерной обработки данных. Несмотря на сложность реализации многопоточной реализации, она даст выиграш, за счет параллельной обработки ресурсов.

3 Технологическая часть

В данном разделе будут рассмотрены требования к программному обеспечению, средства реализации и представлен листинг кода.

3.1 Требования к программному обеспечению

1. программа должна корректно осуществлять передачу/прием между конвейерами;

2. программа должна обеспечить возможность замера времени работы алгоритма.

3.2 Средства реализации

В данной работе используется язык программирования Python, за высокую скорость выполнения программ и широкий выбор инструментов для параллельных вычислений. Проект выполнен в среде разработки Visual Studio Code. Многопоточное программирование было реализовано с помощью ThreadPoolExecutor. [4]

3.3 Листинг кода

В данном пункте представлен листинг кода, а именно:

- алгоритм конвейерной обработки данных.

На листинге 1 представлен код трехуровневого конвейера.

```
    async def level_a(data):
        lvl_b_data = data, 2 * data, 4 * data
        results = await asyncio.gather(*[level_b(val) for val in lvl_b_data])
        result = await loop.run_in_executor(executor, cpu_bound_op, 3, *results)
        return result


    async def level_b(data):
        lvl_c_data = data, 3 * data, 5 * data
        results = await asyncio.gather(*[level_c(val) for val in lvl_c_data])
        result = await loop.run_in_executor(executor, cpu_bound_op, 2, *results)
        return result


    async def level_c(data):
        result = await loop.run_in_executor(executor, cpu_bound_op, 1, data)
        return result

```

Листинг 1: Конвейерный алгоритм

3.4 Вывод

В данном разделе была представлена структура ПО и листинги кода программы.

4 Исследовательская часть

В данном разделе будет проведен эксперимент и сравнительный анализ.

4.1 Системные характеристики

Характеристики компьютера на котором проводился замер времени сортировки массива:

1. операционная система - Windows 10;
2. процессор - Intel(R) Core(TM) i7-10510U CPU @1.80GHz 2.30GHz;
3. оперативная память - 16 ГБ;
4. количество ядер - 4;
5. количество логических процессов - 8.

4.2 Постановка эксперимента

В рамках данного проекта были проведены эксперименты, описанные ниже:

1. сравнение и анализ времени работы конвейера для разного количества потоков;
2. сравнение и анализ времени работы программы при асинхронном и не асинхронном программировании.

4.3 Сравнительный анализ на основе замеров времени работы программы

Был проведен замер времени работы каждого потока.

На рисунке 1 показаны результаты первого эксперимента, суть которого заключается в анализе зависимости числа потоков от времени работы конвейра при одном и том же числовом значении. Ниже приведена полученная диаграмма:

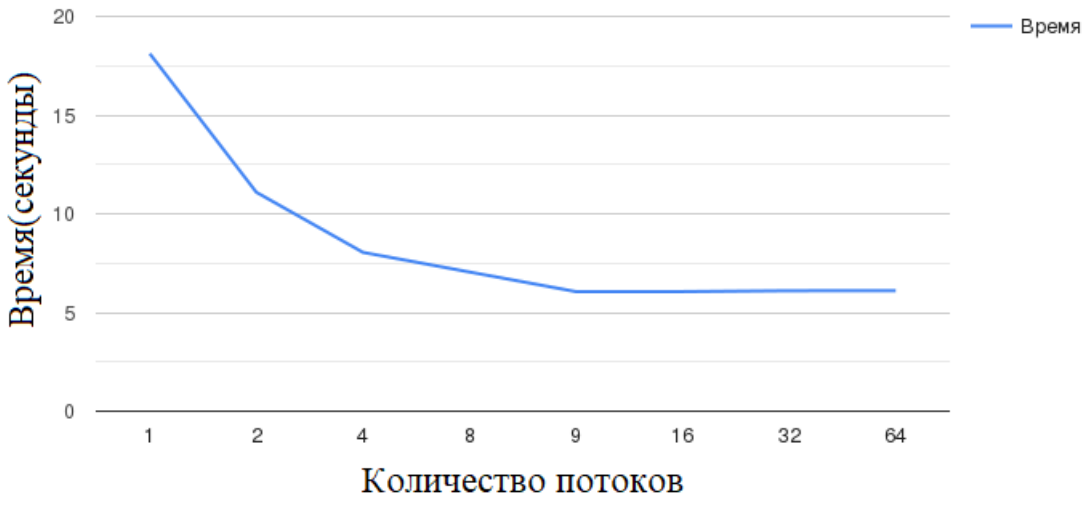

Рис. 1: Сравнение времени работы конвейера при разном количестве потоков

На рисунке 2 показаны результаты первого эксперимента, суть которого заключается в анализе зависимости числа потоков от времени работы процессора при одном и том же числовом значении. Ниже приведена полученная диаграмма:

Рис. 2: Сравнение времени работы процессора при работе конвейера при разном количестве потоков.

Третий эксперимент был проведен, для того чтобы сделать анализ зависимости времени работы при асинхронном и не асинхронном программировании. В таблице 1 показаны результаты этого эксперимента.

Таблица 1: Результаты временного замери при асинхронном и не асинхронном программировании.

Входное число	Асинхронно	Не асинхронно
2	6.052519 | 0.015625	18.024558 | 0.015625
100	6.051918 | 0.015625	18.010213 | 0.015625
10 000	6.055719 | 0.031250	18.009668 | 0.015600
10 000 000	6.061437 | 0.031250	18.003867 | 0.015534

4.4 Вывод

По проведенному анализу из первого эксперимента можно сделать вывод, что наилучшее время работы было достигнуто, когда количество потоков равно 9. Это не случайный результат, потому что на уровне C выполняется 9 вычислений. Следовательно у каждого потока была своя задача и так как у нас не было лишних или недостающих потоков и был достигнут такой результат. Второй эксперимент был отчасти связан с первым. Единственным отличием было замер времени. Здесь он проводился у самого процессора. Сделав небольшой анализ, можно понять, что лучший результат был достигнут когда количество потоков было равным 9. Третий эксперимент был проведен чтобы увидеть значимую разницу при асинхронном и не асинхронном программировании. Асинхронный метод с конвейерной реализацией показал время в три раза быстрее чем при не асинхронном методе, что не скажешь про время работы самого процессора. Лучший показатель был достигнкт при не асинхронном методе и без конвейера.

Заключение

В рамках данной работы успешно изучены основы контейнерных вычислений. Применен метод асинхронного и не асинхронного программирования. Проведен сравнительный анализ контейнерной и традиционной реализаций. Подтверждены экспериментально различия во временной эффективности реализаций при помощи разработанного программного обеспечения на материале замеров времени выполнения в зависимости от загруженности процесса. Дано описание и обоснование полученных результатов.

Экспериментально получено, что реализация с контейнерами работает более чем три раза быстрее, нежели традиционная.

Список литературы

[1] David Beazly. *Developing a computational pipeline using the asyncio module in Python3[ЭЛ. РЕСУРС] Режим доступа:* URL: http://deklund.com/blog/2015/11/22/developing-a-computational-pipeline-using-the-asyncio-module-in-python-3. (дата обращения: 27.11.2020).

[2] Wikipedia. *Конвейерное производство [ЭЛ. РЕСУРС] Режим доступа:* URL: https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BD%D0%B2%D0%B5%D0%B9%D0%B5%D1%80%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B8%D0%B7%D0%B2%D0%BE%D0%B4%D1%81%D1%82%D0%B2%D0%BE#. (дата обращения: 27.11.2020).

[3] Интуит. *Введение в технологии параллельного программирования[ЭЛ. РЕСУРС] Режим доступа:* URL: https://intuit.ru/studies/courses/4447/983/lecture/14925. (дата обращения: 22.11.2020).

[4] Гвидо ван Россум. *Python documentation[ЭЛ. РЕСУРС] Режим доступа:* URL: https://docs.python.org/3/library/concurrent.futures.html. (дата обращения: 22.11.2020).