Министерство науки и высшего образования 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»

Лабораторная работа №5

По предмету: «Анализ алгоритмов»

Тема: Конвейерные вычисления

Студент: Гасанзаде М.А.,
Группа: ИУ7-56Б

Москва, 2019 г.

Оглавление

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.

Цель работы: изучение метода конвейерных вычислений.

Задачи работы:

1) изучение основ конвейерной обработки данных;

2) получение практических навыков конвейерных вычислений;

3) сравнительный анализ конвейерной и традиционной обработки данных по затрачиваемым ресурсам;

4) экспериментальное подтверждение различий во временной эффективности реализаций вычислений при помощи разработанного программного обеспечения на материале замеров процессорного времени выполнения;

6) описание и обоснование полученных результатов в отчете о выполненной лабораторной работе, выполненного как расчётно-пояснительная записка к работе.

1. Аналитическая часть

В данной части дано теоретическое описание метода и указание области его применения.

1.1. Описание метода

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

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

1.2. Применение метода

Методы активно применяются в [2]:

1. многих задачах линейной алгебры и задачах решения систем уравнений в частных производных;
2. обработке машинных команд процессором.

2. Конструкторская часть

Вдохновившись курсом ОС, я решил перенести передачу по передачу сообщений между процессами по трубе, на многопоточную реализацию.

Моделируется передача данных между потоками, полезная работа не производится.

2.1 Разработка алгоритма

Функциональная модель процесса передачи сообщения в нотации IDEF0 представлена на рис. 1.

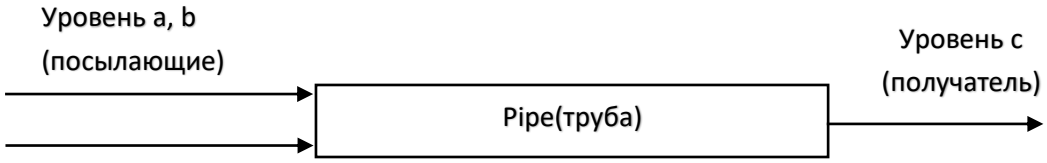

Рисунок 1 – Функциональная модель процесса передачи сообщений

Схема алгоритма представлена на рис. 2.

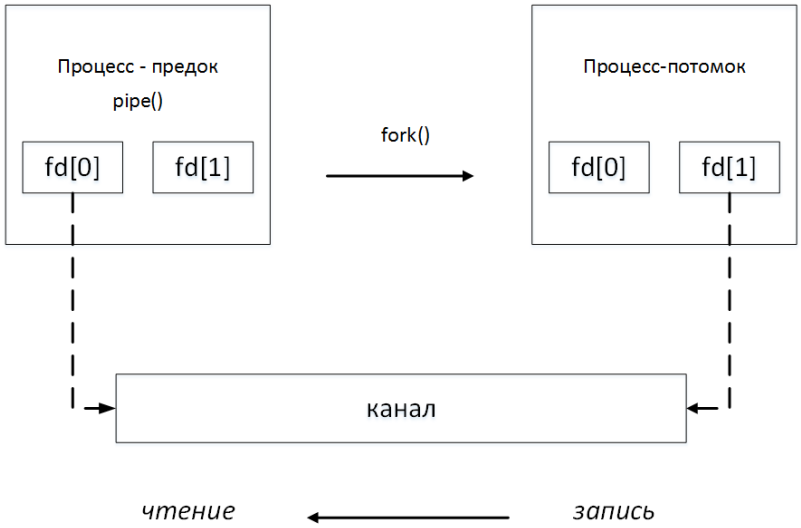

Рисунок 2 – Схема алгоритма сообщения через трубу

2.2 Вывод

Не смотря на сложность реализации многопоточной реализации, она даст выигрыш, за счёт параллельной обработки ресурсов.

3. Технологическая часть

В данной части приведены используемые технические средства, а также примеры тестирования и листинг программы.

3.1 Требования к программному обеспечению

Программа должна корректно осуществлять передачу/приём между процессами. Требуется также обеспечить возможность замера времени работы алгоритма.

3.2 Средства реализации

Выбран язык программирования Python3 за высокую скорость выполнения программ и широкий выбор инструментов для параллельных вычислений. Работа с потоками осуществляется посредством класса `asyncio`. Замер времени проводится функцией `time`.

3.3 *Листинг кода*

Исходный код реализации программы приведен в листингах 1, 2.

Листинг 1 – Создание loggerов, загрузка необходимых библиотек

```
import asyncio
import logging
import time
from concurrent.futures import ThreadPoolExecutor

logging.basicConfig(format="[%(thread)-5d]%(asctime)s:
%(message)s")
logger = logging.getLogger('async')
logger.setLevel(logging.INFO)

executor = ThreadPoolExecutor(max_workers=13)  # thread pool
loop = asyncio.get_event_loop()  # event loop
```

Листинг 2 – Создание трубы и связи между процессами и создание процесса

```
def cpu_bound_op(exec_time, *data):
    logger.info("Running cpu-bound op on {} for {} seconds".format(data,
exec_time))
    time.sleep(exec_time)
    return sum(data)


async def process_pipeline(data):
    # just pass the data along to level_a and return the results
    results = await level_a(data)  # Waiting for the level a
    return results


async def level_a(data):
    level_b_inputs = data, 2 * data, 4 * data
    results = await asyncio.gather(*[level_b(val) for val in level_b_inputs])  #
aggregate results from the level b

    result = await loop.run_in_executor(executor, cpu_bound_op, 3, *results)
    return result
```

3.4 Описание тестирования: Тестирование проводится по методу чёрного ящика [3]. Требуется проверить корректность работы на случаях с различной степенью загруженности трубы.

3.4 Вывод

Текущая реализация на языке Python3 позволяет корректно передавать сообщения, а также производить замеры времени.

4. Экспериментальная часть

В этой части приведены примеры интерфейса, входные данные тестирования, результаты замера времени и их анализ.

4.1. Примеры работы

На рис. 3, 4 приведены изображения внешнего вида интерфейса программы во время его работы.

Рисунок 3 – Пример работы программы при значении 2&4 и 3&5 данных(элементов в процессе)

```
[12280]2019-12-20 03:10:16,081: Running cpu-bound op on (12,) for 1 seconds
[11392]2019-12-20 03:10:16,082: Running cpu-bound op on (4,) for 1 seconds
[376  ]2019-12-20 03:10:16,083: Running cpu-bound op on (20,) for 1 seconds
[10112]2019-12-20 03:10:16,084: Running cpu-bound op on (40,) for 1 seconds
[13096]2019-12-20 03:10:16,084: Running cpu-bound op on (24,) for 1 seconds
[12284]2019-12-20 03:10:16,085: Running cpu-bound op on (8,) for 1 seconds
[6064 ]2019-12-20 03:10:16,085: Running cpu-bound op on (10,) for 1 seconds
[12756]2019-12-20 03:10:16,086: Running cpu-bound op on (6,) for 1 seconds
[7804 ]2019-12-20 03:10:16,087: Running cpu-bound op on (2,) for 1 seconds
[12280]2019-12-20 03:10:17,096: Running cpu-bound op on (4, 12, 20) for 2 seconds
[11392]2019-12-20 03:10:17,108: Running cpu-bound op on (8, 24, 40) for 2 seconds
[376  ]2019-12-20 03:10:17,123: Running cpu-bound op on (2, 6, 10) for 2 seconds
[9308 ]2019-12-20 03:10:19,139: Running cpu-bound op on (18, 36, 72) for 3 seconds
[8180 ]2019-12-20 03:10:22,207: Completed (126) in 6.068879127502441 seconds and 6.0690916 cpu-time
[8180 ]2019-12-20 03:10:22,212: Running cpu-bound op on (2, 4, 8)
[8180 ]2019-12-20 03:10:22,215: Running cpu-bound op on (2, 6, 10)
[8180 ]2019-12-20 03:10:22,218: Running cpu-bound op on (4, 12, 20)
[8180 ]2019-12-20 03:10:22,222: Running cpu-bound op on (8, 24, 40)
[8180 ]2019-12-20 03:10:40,225: Running cpu-bound op on 18
[8180 ]2019-12-20 03:10:40,229: Completed (126) in 18.017069578170776 seconds and 18.017839000000002 cpu-time
>>>
```

Рисунок 4 –Пример работы программы при значении 20&40 и 30&50 данных(элементов в процессе)

```
[11984]2019-12-20 03:20:18,233: Running cpu-bound op on (24,) for 1 seconds
[4264 ]2019-12-20 03:20:18,233: Running cpu-bound op on (8,) for 1 seconds
[11816]2019-12-20 03:20:18,233: Running cpu-bound op on (40,) for 1 seconds
[9340 ]2019-12-20 03:20:18,233: Running cpu-bound op on (20,) for 1 seconds
[7476 ]2019-12-20 03:20:18,233: Running cpu-bound op on (12,) for 1 seconds
[12160]2019-12-20 03:20:18,249: Running cpu-bound op on (4,) for 1 seconds
[11944]2019-12-20 03:20:18,249: Running cpu-bound op on (10,) for 1 seconds
[11980]2019-12-20 03:20:18,249: Running cpu-bound op on (6,) for 1 seconds
[268  ]2019-12-20 03:20:18,249: Running cpu-bound op on (2,) for 1 seconds
[11816]2019-12-20 03:20:19,266: Running cpu-bound op on (8, 24, 40) for 2 seconds
[11944]2019-12-20 03:20:19,282: Running cpu-bound op on (4, 12, 20) for 2 seconds
[11980]2019-12-20 03:20:19,297: Running cpu-bound op on (2, 6, 10) for 2 seconds
[12508]2019-12-20 03:20:21,304: Running cpu-bound op on (18, 36, 72) for 3 seconds
[12076]2019-12-20 03:20:24,323: Completed (126) in 6.090068101882935 seconds and 6.0787995 cpu-time
[12076]2019-12-20 03:20:24,339: Running cpu-bound op on (2, 40, 80)
[12076]2019-12-20 03:20:24,339: Running cpu-bound op on (2, 60, 100)
[12076]2019-12-20 03:20:24,339: Running cpu-bound op on (40, 1200, 2000)
[12076]2019-12-20 03:20:24,354: Running cpu-bound op on (80, 2400, 4000)
[12076]2019-12-20 03:20:42,368: Running cpu-bound op on 162
[12076]2019-12-20 03:20:42,430: Completed (9882) in 18.091418266296387 seconds and 18.0903418 cpu-time
>>>
```

4.2. Результаты тестирования

В Таблице 1 представлены результаты тестирования.

Таблица 1.

Результаты тестирования по методу черного ящика

Информативная нагрузка проц.	Традиционная реализация	Конвейерная реализация
2&4 3&5	Передача успешна	Передача успешна
20&40 30&50	Передача успешна	Передача успешна

Тесты пройдены успешно.

4.3. Постановка эксперимента по замеру времени

*З*амер времени проводился 5 раз, результат одного эксперимента рассчитывался как среднее значение результатов проведенных испытаний с одинаковыми входными данными. Замерялись традиционная реализация и многопоточная реализация.

4.4. Сравнительный анализ на материале экспериментальных данных

На рис. 5 представлен результат замеров времени.

Где min: 1&1 1&1

Medium: 2&4 3&5

Medium2: 20&40 30&50

High: 50&50 50&50

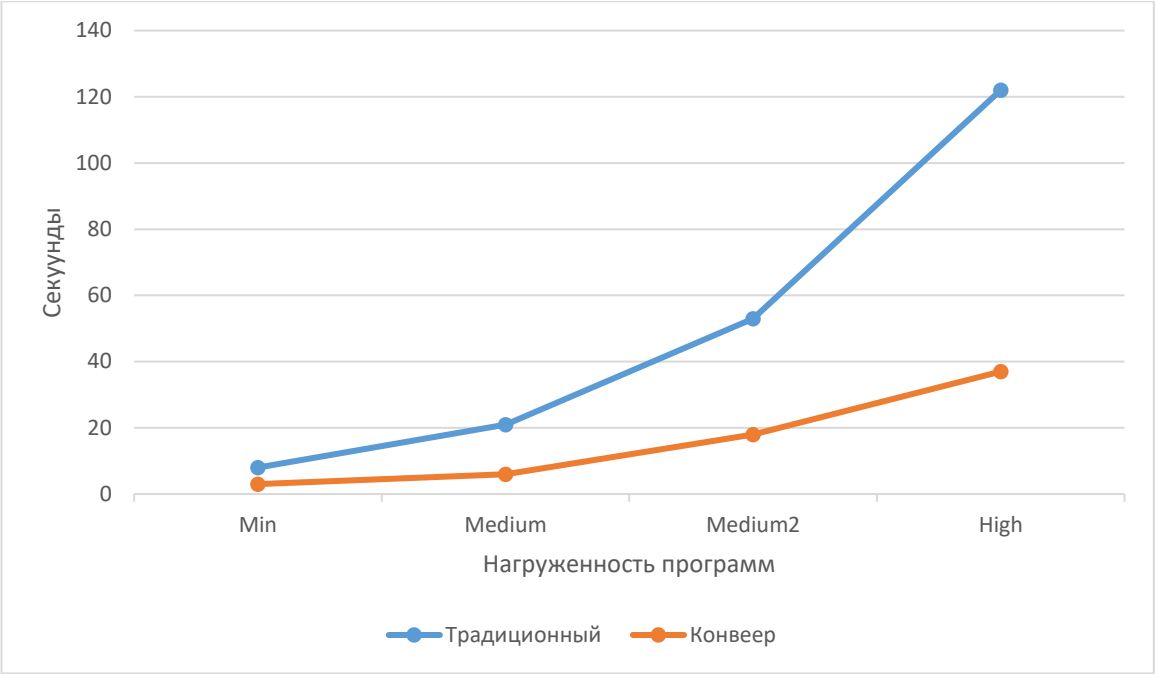

Рисунок 5 – Время передачи сообщений традиционной и конвейерной реализациями

4.5 Вывод

По результатам эксперимента подтверждено, что время исполнения конвейерной реализации значительно меньше времени исполнения традиционной реализации. Для High загруженности контейнерная реализация примерно в 3.29 раза быстрее.

Заключение

В рамках данной работы успешно изучены основы контейнерных вычислений. Применен метод динамического программирования для контейнерной реализации для передачи сообщений между процессами. 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. Конвейерные вычисления // STUDYLIB URL: https://studylib.ru/doc/4736512/konvejernye-vychisleniya (дата обращения: 19.12.2019).
2. Конвейеризация вычислений // studref URL: https://studref.com/313869/informatika/konveyerizatsiya_vychisleniy (дата обращения: 19.12.2019).
3. Kara kutu test technologisi URL:https://www.mobilhanem.com/kara-kutu-test-teknigi-ve-uygulanmasi/ (дата обращения 19.12.2019)