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»

Отчёт

по лабораторной работе № 5

Название: Конвейер

Дисциплина: Анализ алгоритмов

Студент ИУ7-54Б _____ Д.Ю. Расколотов

(Группа) (Подпись, дата) (И.О. Фамилия)

Преподователь _____ Л.Л. Волкова

(Подпись, дата) (И.О. Фамилия)

Москва, 2020

Оглавление

Введение **2**

1 Аналитическая часть **3**

1.1 Общие сведения о конвейерной обработке 3

1.2 Параллельное программирование 3

1.2.1 Организация взаимодействия параллельных потоков 4

1.3 Вывод 4

2 Конструкторская часть **5**

2.1 Организация обработки 5

2.2 Генератор задач 6

2.3 Вывод 7

3 Технологическая часть **8**

3.1 Выбор ЯП 8

3.2 Листинг программы 9

3.3 Вывод 12

4 Исследовательская часть **13**

4.1 Анализ выбранных сортировок 13

4.2 Тестирование 13

4.3 Вывод 15

Заключение **16**

Список литературы **16**

Введение

Цель работы: получить навык организации конвейерных вычислений в параллельном режиме.

Задачи данной лабораторной работы:

1. спроектировать ПО, реализующего конвейерную обработку;
2. описать реализацию ПО;
3. провести тестирование ПО.

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 [1].

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.

Каждое звено конвейера выполняет следующие действия:

1. Получить данные,
2. Обработать данные,
3. Послать данные следующим звеньям.

1.2 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.

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.

1.2.1 Организация взаимодействия параллельных потоков

Потоки исполняются в общем адресном пространстве параллельной программы. Как результат, взаимодействие параллельных потоков можно организовать через использование общих данных, являющихся доступными для всех потоков. Наиболее простая ситуация состоит в использовании общих данных только для чтения. В случае же, когда общие данные могут изменяться несколькими потоками, необходимы специальные усилия для организации правильного взаимодействия.

1.3 Вывод

В данном разделе были рассмотрены основы конвейерной обработки, технология параллельного программирования и организация взаимодействия параллельных потоков.

2 Конструкторская часть

Требования к вводу: На вход в очередь №1 подаются задачи из генератора задач.

Требования к программе при параллельной обработке:

- объекты должны последовательно проходить конвейеры в заданном подядке;
- конвейеры должны работать каждый в своем потоке;
- конвейер должен завершать свою работу при выполнении всех задач;
- до завершения работы конвейер должен ожидать поступления новых элементов.

2.1 Организация обработки

У каждой линии конвейера есть очередь элементов. Когда линия еще активна, но элементов в очереди нет, линия уходит в режим ожидания. По прошествию заданного времени линия проверяет не появились ли новые элементы в очереди. Если очередь не пустая, то нужно получить и обработать элемент, передать его следующей линии, если такая существует.

На рисунке 2.1 представлена подробная схема организации конвейерной обработки.

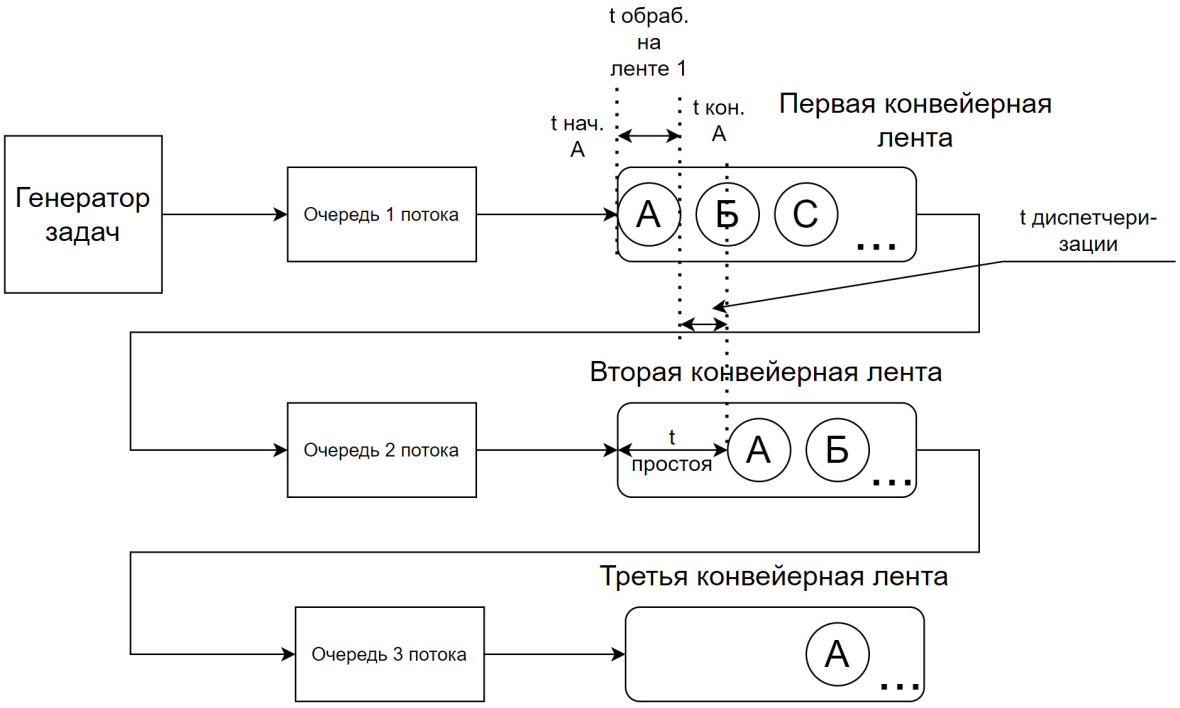

Рисунок 2.1 – Схема конвейерной обработки

2.2 Генератор задач

В качестве задач, реализуемых на конвейере, я решил выбрать - сортировки. Данные, которые они будут сортировать - неизменны. То есть на вход каждой сортировки будет подаваться неотсортированный константный массив состоящий из 10000 элементов. Выбор сортировок осуществлялся, исходя из практически идентичного времени работы каждой сортировки.

Выбранные алгоритмы сортировок:

- сортировка Шейкером;
- сортировка Пузырьком;
- сортировка Пузырьком с флагом;
- сортировка прямыми включениями;
- сортировка прямым выбором.

2.3 Вывод

В данном разделе была рассмотрена схема организации конвейерной обработки. А так же выбраны соразмерные задачи для реализации конвейерной ленты.

3 Технологическая часть

Замеры времени были произведены на: Intel(R) Core(TM) i7-10510U, 4 ядра, 8 логических процессоров.

3.1 Выбор ЯП

В качестве языка программирования был выбран C++, так как этот язык поддерживает управление потоками на уровне ОС(незеленые потоки). Средой разработки Visual Studio Code. Время работы алгоритмов было замерено с помощью класса <windows.h>. Многопоточное программирование было реализовано с помощью класса Thread.

3.2 Листинг программы

Листинг 3.1 – Структура объекта

```
typedef struct Log
{
    int tape;
    int sort;
    char status;
    SYSTEMTIME time;
} Log;
```

Листинг 3.2 – Основная функция

```
void MainFunc()
{
    generate_massive(a, b, c, len_mas);

    for (auto i = 0; i < 5; ++i)
    que1.push((i + 1));

    Thread threads;
    threads.push_back(thread(way1));
    threads.push_back(thread(way2));
    threads.push_back(thread(way3));

    for (auto &th : threads)
    th.join();
}
```

Листинг 3.3 – Лента №1

```
void way1()
{
    while (true)
    {
```

```
            if (!que1.empty())
            {
                int sortN = que1.front();

                m.lock();
                GetSystemTime(&t);
                res.push_back({1, sortN, 'S', t});
                m.unlock();

                CurrentSort(sortN, a, len_mas);

                que1.pop();

                m.lock();
                GetSystemTime(&t);
                res.push_back({1, sortN, 'F', t});
                m.unlock();

                que2.push(sortN);

                if (sortN == 5)
                return;
            }
        }
    }
```

Листинг 3.4 – Лента №2

```
    void way2()
    {
        while (true)
        {
            if (!que2.empty())
            {
```

```
                int sortN = que2.front();

                m.lock();
                GetSystemTime(&t);
                res.push_back({2, sortN, 'S', t});
                m.unlock();

                CurrentSort(sortN, b, len_mas);

                que2.pop();

                m.lock();
                GetSystemTime(&t);
                res.push_back({2, sortN, 'F', t});
                m.unlock();

                que3.push(sortN);

                if (sortN == 5)
                return;
            }
        }
    }
```

Листинг 3.5 – Лента №3

```
    void way3()
    {
        while (true)
        {
            if (!que3.empty())
            {
                int sortN = que3.front();

```

```
                m.lock();
                GetSystemTime(&t);
                res.push_back({3, sortN, 'S', t});
                m.unlock();

                CurrentSort(sortN, c, len_mas);

                que3.pop();

                m.lock();
                GetSystemTime(&t);
                res.push_back({3, sortN, 'F', t});
                m.unlock();

                if (sortN == 5)
                return;
            }
        }
    }
```

3.3 Вывод

В данном разделе была рассмотрена структура ПО и основной листинг кода программы.

4 | Исследовательская часть

4.1 Анализ выбранных сортировок

Был проведен замер времени работы каждого из алгоритмов сортировок. Первый эксперимент производился на неотсортировнном массиве, состоящем из 10000 элементов.

В таблице 4.1 представлены замеры времени выполнения сортировок.

Табл. 4.1 Замер времени сортировок

№	Сортировка	Время,с.
1	Шейкером:	0.755
2	Пузырьком:	0.802
3	Пузырьком с флагом:	0.954
4	Прямым включением:	0.602
5	Прямым выбором:	0.685

4.2 Тестирование

Обозначения:

- № - номер операции;
- Belt - номер ленты;
- Task - номер задачи (см. табл. 4.1);
- State - состояние обработки (S - начало, F - конец);
- Time - время события в формате ч:м:с:мс.

В таблице 4.2 представлены замеры времени выполнения конвейерной ленты.

Табл. 4.2 Тестирование программы

№	Belt	Task	State	Time
1	1	1	S	17:29:26:430
2	1	1	F	17:29:33:637
3	1	2	S	17:29:33:637
4	2	1	S	17:29:33:637
5	1	2	F	17:29:39:334
6	1	3	S	17:29:39:334
7	2	1	F	17:29:41:640
8	2	2	S	17:29:41:640
9	3	1	S	17:29:41:640
10	1	3	F	17:29:43:369
11	1	4	S	17:29:43:369
12	1	4	F	17:29:44:657
13	1	5	S	17:29:44:657
14	1	5	F	17:29:45:472
15	2	2	F	17:29:46:610
16	2	3	S	17:29:46:610
17	3	1	F	17:29:48:348
18	3	2	S	17:29:48:348
19	2	3	F	17:29:50:623
20	2	4	S	17:29:50:623
21	2	4	F	17:29:52:330
22	2	5	S	17:29:52:330
23	2	5	F	17:29:52:680
24	3	2	F	17:29:53:674
25	3	3	S	17:29:53:674
26	3	3	F	17:29:56:995
27	3	4	S	17:29:56:995
28	3	4	F	17:29:58:344
29	3	5	S	17:29:58:344
30	3	5	F	17:29:59:670

Основываясь на данных, приведённых в таблице можно посчитать следующие величины:

Величины:

- макс. время простоя = 15,16 сек.;
- полное время работы конвейера = 33,24 сек.;
- мин. время выполнения сортировки Шейкера (Task 1) = 6,708 сек.;
- макс. время выполнения сортировки Шейкера (Task 1) = 8,003 сек.;
- среднее время выполнения сортировки Шейкера (Task 1) = 7,306 сек.;
- макс. время ожидания в очереди №2 = 6,858 сек.
  мин. время « 0.001 сек.;
- макс. время ожидания в очереди №3 = 5,664 сек.
  мин. время « 0.001 сек.;
- среднее время ожидания в очередях = 5,124 сек..

4.3 Вывод

По результатам исследования конвейерную обработку нет смысла применять для задач, занимающих мало времени, т.к. в этом случее большая часть времени потратится на ожидание доступа к переменной, дополнительных проверок. Тестирование показало, что конвейерная обработка реализована правильно.

Заключение

В ходе лабораторной работы я изучил возможности применения параллельных вычислений и конвейерной обработки и использовал такой подход на практике.

Был проведен эксперимент с соразмерными задачами, который показал что если первый конвейер тормозит работу, то общее время работы системы линейно повышается от задержки первого конвейера. Также этот эксперимент показал, что конвейерную обработку нет смысла применять для задач, занимающих мало времени, т.к. в этом случее большая часть времени потратится на ожидание доступа к переменной, дополнительных проверок.

Конвейерная обработка позволяет сильно ускорить программу, если требуется обработать набор из однотипных данных, причем алгоритм обработки должен быть разбиваем на стадии. Однако от конвейерной обработки не будет смысла, если одна из стадий намного более трудоемкая, чем остальные, так как производительность всей программы будет упираться в производительность этой самой стадии, и разницы между обычной обработкой и конвейерной не будет, только добавятся накладные вычисления, связанные с диспетчеризацией потоков. В таком случае можно либо разбить трудоемкую стадию на набор менее трудоемких, либо выбрать другой алгоритм, либо отказаться от конвейерной обработки.

Цель достигнута и все задачи выполнены.

Литература

[1] Меднов В.П., Бондаренко Е.П. Транспортные, распределительные и рабочие конвейеры. М., 1970.

[2] Конвейерное производство[Электронный ресурс] - режим доступа https:/ë/dic.academic.ru/dic.nsf/ruwiki/1526795

[3] Конвейерный метод производства Генри Форда[Электронный ресурс] - режим доступа https://popecon.ru/305-konveiernyi-metod-proizvodstva-genri-forda.html

[4] Константин Баркалов, Владимир Воеводин, Виктор Гергель. Intel Parallel Programming [Электронный ресурс], - режим доступа https://www.intuit.ru/studies/courses/4447/983/lecture/14925

[5] Константин Баркалов, Владимир Воеводин, Виктор Гергель. Intel Parallel Programming [Электронный ресурс], - режим доступа https://www.intuit.ru/studies/courses/4447/983/lecture/14925

[6] Алгоритмы сортировки и поиска - режим доступа https://prog-cpp.ru/algorithm-sort/