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»

ОТЧЕТ

по Лабораторной работе №5

по курсу «Анализ Алгоритмов»

на тему: «Конвейерная обработка данных»

Студент группы ИУ7-56Б _____ (Подпись, дата) Мансуров В. М. (Фамилия И.О.)

Преподаватель _____ (Подпись, дата) Волкова Л. Л. (Фамилия И.О.)

Преподаватель _____ (Подпись, дата) Строганов Ю. В.. (Фамилия И.О.)

Москва — 2023 г.

Содержание

Введение

Использование параллельной обработки открывает новые способы для ускорения работы программ. Конвейерная обработка является одним из примеров, где использование принципов параллельности помогает ускорить обработку данных. Суть та же, что и при работе реальных конвейерных лент — материал (данное) поступает на обработку, после окончания обработки материал передается на место следующего обработчика, при этом предыдущий обработчик не ждет полного цикла обработки материала, а получает новый материал и работает с ним.

Целью данной лабораторной работы является описание параллельных конвейерных вычислений.

Для поставленной цели необходимо выполнить следующие задачи.

1) Описать организацию конвейерной обработки данных.

2) Описать алгоритмы обработки данных, которые будут использоваться в текущей лабораторной работе.

3) Реализовать программу, реализующую конвейер с количеством лент не менее трех в однопоточной и многопоточной среде.

4) Сравнить и проанализировать реализации алгоритмов по затраченному времени.

1 Аналитическая часть

В данном разделе рассмотрена информация, касающаяся основ конвейерной обработки данных.

1.1 Конвейерная обработка данных

Конвейер [1] (англ. *conway*) — организация вычислений, при которой увеличивается количество выполняемых инструкций за единицу времени за счет использования принципов параллельности.

Конвейеризация (или конвейерная обработка) в общем случае основана на разделении подлежащей исполнению функции на более мелкие части, называемые ступенями, и выделении для каждой из них отдельного блока аппаратуры. Так, обработку любой машинной команды можно разделить на несколько этапов (несколько ступеней), организовав передачу данных от одного этапа к следующему. При этом конвейерную обработку можно использовать для совмещения этапов выполнения разных команд. Производительность при этом возрастает, благодаря тому, что одновременно на различных ступенях конвейера выполняется несколько команд. Конвейерная обработка такого рода широко применяется во всех современных быстродействующих процессорах.

Конвейеризация позволяет увеличить пропускную способность процессора (количество команд, завершающихся в единицу времени), но она не сокращает время выполнения отдельной команды. В действительности она даже несколько увеличивает время выполнения каждой команды из-за накладных расходов, связанных с хранением промежуточных результатов. Однако увеличение пропускной способности означает, что программа будет выполняться быстрее по сравнению с простой, не конвейерной схемой.

1.2 Разреженный строчный формат матрицы

Во многих областях человеческой деятельности информацию часто представляют в форме матриц. Матрица — это регулярный числовой массив. Разреженная матрица — матрица, в которой большинство элементов равны нулю.

Разреженный строчный формат (сокр. РСФ) [2] — это одна из наиболее широко используемых схем хранения разреженных матриц. Эта схема предъявляет минимальные требования к памяти и в то же время оказывается очень удобной для нескольких важных операций над разреженными матрицами: сложения, умножения, перестановок строк и столбцов, транспонирования, решения линейных систем с разреженными матрицами коэффициентов как прямыми, так и итерационными методами и т. д. Значения ненулевых элементов матрицы и соответствующие столбцовые индексы хранятся в этой схеме по строкам в двух массивах; назовем их соответственно AN и JA. Используется также массив указателей (скажем, AI, еще обозначающийся как NR), отмечающих позиции массивов AN и JA, с которых начинается описание очередной строки. Дополнительная компонента в AI содержит указатель первой свободной позиции в JA и AN. Пример представления матрицы РСФ на рисунке 1.1.

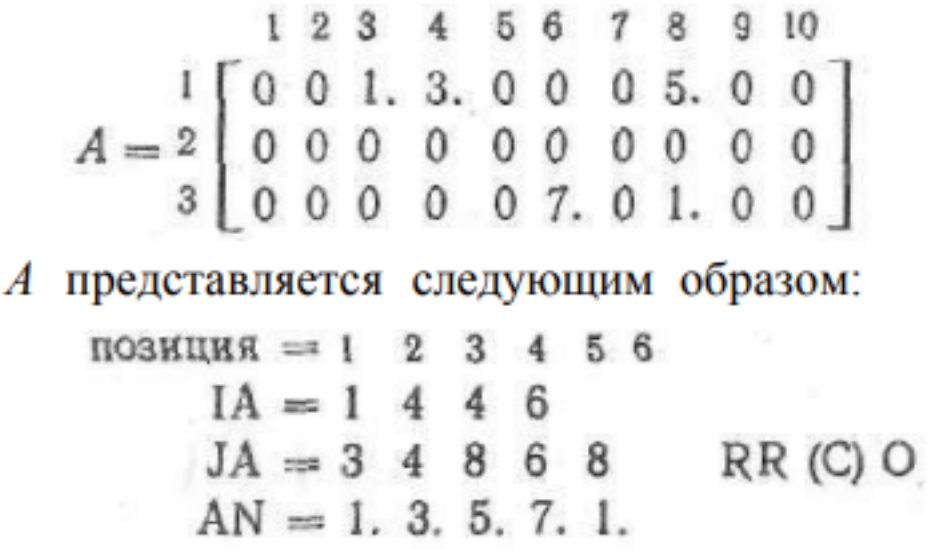

Рисунок 1.1 – Пример представления матрицы в РСФ

В общем случае описание i-й строки хранится в позициях с $IA(i)$ до $IA(i+1) - 1$ массивов JA и AN, за исключением равенства $IA(i+1) = IA(i)$, означающего, что r-я строка пуста. Если матрица имеет n строк, то IA содержит $n+1$ позиций.

1.3 Сложение РСФ матриц

Для сложения матриц в РСФ необходимо пройти через матрицу указателей AI поэлементно через две матрицы, выделяя интервалы между соседними элементами AI по AN и JA, то есть $AI(i)$ и $AI(i+1)$. Если встретится элементы с одинаковыми строками и столбцами, то производится сложение. Затем происходит повторные проходы по двум матрицам по отдельности для того чтобы добавить в результирующую матрицу остаточные значение [2].

1.4 Описание алгоритмов

В качестве операций, выполняющихся на конвейере, взяты следующие:

1) создание двух матриц в РСФ;

2) сложение двух созданных ранее матриц в РСФ;

3) распаковка результата РСФ в классическое матричное представление.

Ленты конвейера (обработчики) будут передавать друг другу заявки. Первый этап, или обработчик, будет формировать заявку, которая будет передаваться от этапа к этапу

Заявка будет содержать:

— две матрицы в РСФ;

— результат сложения двух матриц в РСФ;

— матрица в классическом представлении для распаковки;

— временные отметки начала и конца выполнения стадии обработки заявки.

Вывод

В данном разделе было рассмотрено понятие конвейерной обработки, а также выбраны этапы для обработки матрицы, которые будут обрабатывать ленты конвейера. Также рассмотрена разреженная матрицы и операция сложения таких матриц.

Программа будет получать на вход количество задач, размеры матрицы и количество ненулевых элементов, а также выбор алгоритма — линейный или конвейерный.

Реализуемое программное обеспечение будет давать возможность получить журнал программы для установленного числа задач при линейной и конвейерной обработке. Также будет возможность провести тестирование по времени для разного количества задач и разных размеров самих матриц.

2 Конструкторская часть

В данном разделе будут представлены схемы последовательной и параллельной работы стадий конвейера

2.1 Требования к программному обеспечению

К программе предъявлены ряд требований:

— наличие меню для выбора запускаемого режима работы конвейера — последовательного или параллельного — или выхода из программ;

— предоставление интерфейса для ввода линейного размера обрабатываемых матриц и числа заявок;

— работа с массивами и «нативными» потоками;

— формирование файла с логом работы конвейера, логирование событий обработки должно происходить после окончания работы, собственно, конвейера.

2.2 Разработка алгоритмов

На рисунке 2.1 представлен последовательный алгоритм работы стадий конвейера.

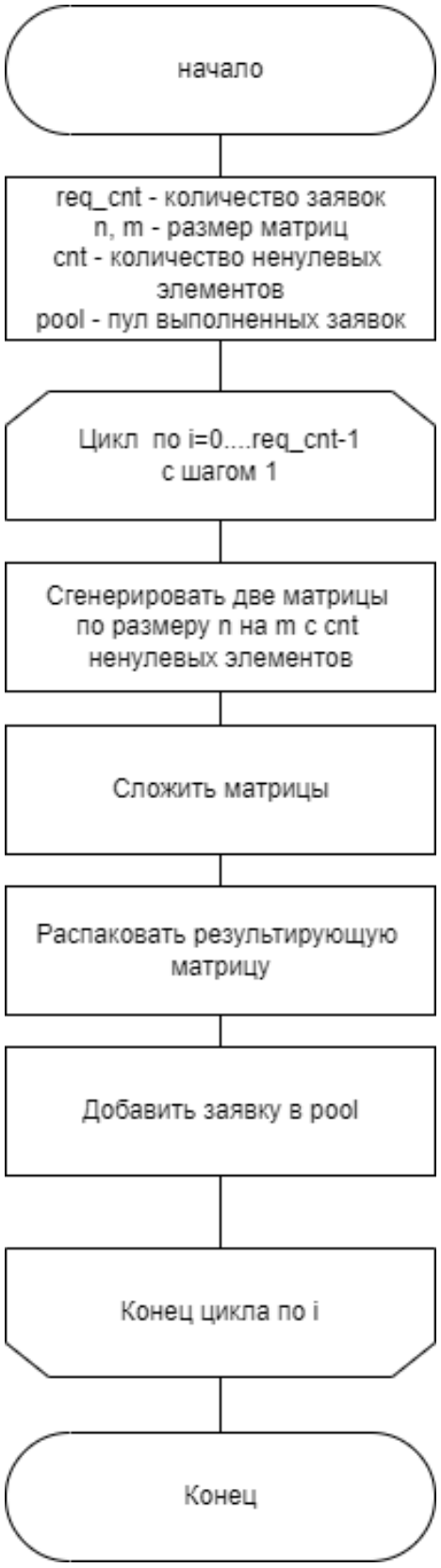

Рисунок 2.1 – Схема алгоритма последовательной конвейерной обработки

Параллельная работа будет реализована с помощью добавления 3-х вспомогательных потоков, где каждый поток отвечает за свою стадию обработки. Вспомогательному потоку в числе аргументов в качестве структуры будут переданы:

— две матрицы в РСФ;

— результирующая матрица сложения двух матриц в РСФ;

— матрица в классическом представлении для распаковки;

— временные отметки начала и конца выполнения стадии обработки заявки.

На рисунке 2.2 представлена схема главного потока при параллельной работе стадий конвейера.

Рисунок 2.2 – Схема параллельной конвейерной обработки

На рисунках 2.3–2.5 представлены схемы алгоритмов каждого из обработчиков (потоков) при параллельной работе.

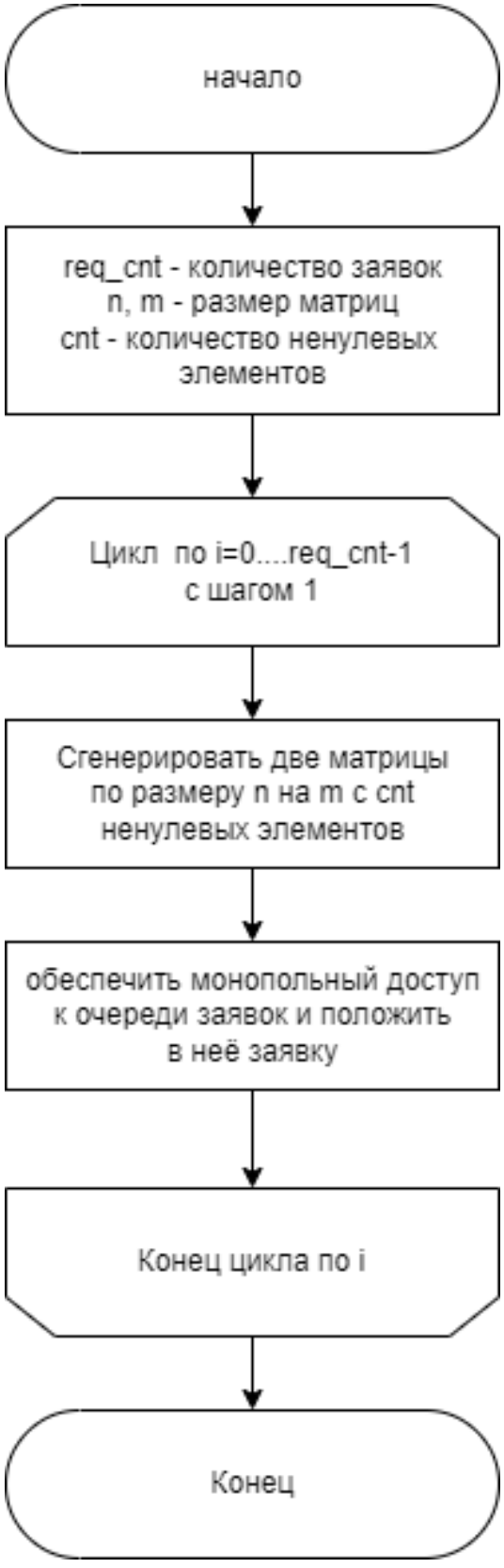

Рисунок 2.3 – Схема алгоритма потока 1

Рисунок 2.4 – Схема алгоритма потока 2

Рисунок 2.5 – Схема алгоритма потока 3

Вывод

В данном разделе были представлены схемы последовательной и параллельной работы стадий конвейера.

3 Технологическая часть

В данном разделе рассмотрены средства реализации, а также представлены листинги реализаций алгоритма расчета термовой частота для всех термов из выборки документов.

3.1 Средства реализации

В качестве языка программирования для реализации данной лабораторной работы был выбран язык $++C$ [3]. Данный выбор обусловлен наличием у языка встроенной модулем измерения процессорного времени [4] и соответствием с выдвинутыми требованиям:

— работа с «зеленными» потоками предоставляется классом *thread* [5];

— работа с мьютексами предоставляется классом *mutex* [6];

— работа с очередями предоставляется классом *queue* [7].

.

3.2 Сведения о модулях программы

Данная программа разбита на следующие модули:

— `main.cpp` — файл, содержащий точку входа в программу. В нем происходит общение с пользователем и вызов алгоритмов;

— `mtr_op.cpp` — файл содержит функции операций над матрицей и матрицей в РСФ;

— `read_size.cpp` — файл содержит функции чтения данных;

— `conveyor.cpp` — файл содержит функции конвейерной обработки;

— `meassure.cpp` — файл содержит функции, замеряющее процессорное время работы реализаций алгоритмов.

3.3 Реализация алгоритмов

На листинге 3.1 представлена последовательная реализация конвейерной обработки. На листингах 3.2 – ?? представлены параллельная реализация конвейерной обработки и реализация каждой стадии в отдельном вспомогательном потоке. На листингах 3.6 – 3.9 представлены реализации алгоритмов генерации матрицы РСФ, суммы двух матриц РСФ и распаковка результирующей матрицы РСФ в классическое представление матрицы.

Листинг 3.1 – Реализация последовательной конвейерной обработки

```
1  void linear() {
2      int req_cnt = get_request_number();
3      int n = get_matrix_n();
4      int m = get_matrix_m();
5      int cnt = get_matrix_num();
6
7      std::vector<request_t *> pool(req_cnt);
8
9      for (int i = 0; i < req_cnt; i++) {
10         request_t *r = new request_t();
11
12         clock_gettime(CLOCK_REALTIME, &r->p1_start);
13         pack_data(n, m, cnt, r);
14         clock_gettime(CLOCK_REALTIME, &r->p1_end);
15
16         clock_gettime(CLOCK_REALTIME, &r->p2_start);
17         r->mtr_c = sum_mtr_csr(r->mtr_a, r->mtr_b);
18         clock_gettime(CLOCK_REALTIME, &r->p2_end);
19
20         clock_gettime(CLOCK_REALTIME, &r->p3_start);
21         r->result = decomprass(r->mtr_c);
22         clock_gettime(CLOCK_REALTIME, &r->p3_end);
23
24         pool[i] = r;
25     }
26     print_pool(pool, "linear.txt");
27     for (size_t i = 0; i < pool.size(); ++i)
28         delete pool[i];
29 }
```

Листинг 3.2 – Реализация основного потока для конвейерной обработки, создающий вспомогтальены потоки

```
1  void parallel() {
2      int req_cnt = get_request_number();
3      int n = get_matrix_n();
4      int m = get_matrix_m();
5      int cnt = get_matrix_num();
6
7      std::vector<request_t *> pool(req_cnt);
8      std::queue<request_t *> q1;
9      std::queue<request_t *> q2;
10     std::queue<request_t *> q3;
11
12     std::thread t_1(thread_1, req_cnt, n, m, cnt, std::ref(q1));
13     std::thread t_2(thread_2, req_cnt, std::ref(q1),
           std::ref(q2));
14     std::thread t_3(thread_3, req_cnt, std::ref(q2),
           std::ref(pool));
15
16     t_1.join();
17     t_2.join();
18     t_3.join();
19
20     print_pool(pool, "parallel.txt");
21     for (size_t i = 0; i < pool.size(); ++i)
22         delete pool[i];
23 }
```

Листинг 3.3 – Реализация вспомогательного потока, отвечающий за создание матриц РСФ

```
1  void thread_1(size_t req_cnt, size_t n, size_t m, size_t cnt,
       std::queue<request_t *> &q1) {
2      for (int i = 0; i < req_cnt; i++)
3      {
4          request_t *r = new request_t();
5          clock_gettime(CLOCK_REALTIME, &r->p1_start);
6          pack_data(n, m, cnt, r);
7
8          mutex_q1.lock();
9          clock_gettime(CLOCK_REALTIME, &r->p1_end);
10         q1.push(r);
11         mutex_q1.unlock();
12     }
13 }
```

Листинг 3.4 – Реализация вспомогательного потока, отвечающий за сложение матриц РСФ

```
1  void thread_2(int req_cnt, std::queue<request_t *> &q1,
       std::queue<request_t *> &q2) {
2      for (int i = 0; i < req_cnt; i++)
3      {
4          while (q1.empty());
5          mutex_q1.lock();
6          request_t *r = q1.front();
7          q1.pop();
8          mutex_q1.unlock();
9
10         clock_gettime(CLOCK_REALTIME, &r->p2_start);
11         r->mtr_c = sum_mtr_csr(r->mtr_a, r->mtr_b);
12
13         mutex_q2.lock();
14         clock_gettime(CLOCK_REALTIME, &r->p2_end);
15         q2.push(r);
16         mutex_q2.unlock();
17     }
18 }
```

Листинг 3.5 – Реализация вспомогательного потока, отвечающий за распаковку матрицы (Часть 1)

```
1  void thread_3(int req_cnt, std::queue<request_t *> &q2,
       std::vector<request_t *> &pool) {
2      for (int i = 0; i < req_cnt; i++)
3      {
4          while (q2.empty());
5          mutex_q2.lock();
6          request_t *r = q2.front();
7          q2.pop();
8          mutex_q2.unlock();
9
10         clock_gettime(CLOCK_REALTIME, &r->p3_start);
11         r->result = decomprass(r->mtr_c);
12         clock_gettime(CLOCK_REALTIME, &r->p3_end);
13         pool[i] = r;
14     }
15 }
```

Листинг 3.6 – Реализация алгоритма генерации данных для матрицы РСФ

```
1  matrix_csr_t sum_mtr_csr(matrix_csr_t &a, matrix_csr_t &b) {
2      matrix_csr_t c;
3      c.n = a.n;
4      c.m = b.n;
5
6      int val = 0;
7      for (int i = 0; i < a.nr.size() - 1; i++) {
8          if (i == 0)
9              c.nr.push_back(0);
10         else
11             c.nr.push_back(c.nr[i - 1] + val);
12         val = 0;
13
14         int ka = a.nr[i];
15         int kb = b.nr[i];
16         for (; ka < a.nr[i + 1] && kb < b.nr[i + 1];) {
17             if (a.ja[ka] < b.ja[kb]) {
18                 c.ja.push_back(a.ja[ka]);
19                 c.an.push_back(a.an[ka++]);
```

Листинг 3.7 – Реализация алгоритма генерации данных для матрицы РСФ (Часть 2)

```
            } else if (a.ja[ka] > b.ja[kb]) {
                c.ja.push_back(b.ja[kb]);
                c.an.push_back(b.an[kb++]);
            } else {
                c.ja.push_back(a.ja[ka]);
                c.an.push_back(a.an[ka++] + b.an[kb++]);
            }
            val++;
        }

        for (; ka < a.nr[i + 1]; ka++) {
            c.ja.push_back(a.ja[ka]);
            c.an.push_back(a.an[ka]);
            val++;
        }
        for (; kb < b.nr[i + 1]; kb++) {
            c.ja.push_back(b.ja[kb]);
            c.an.push_back(b.an[kb]);
            val++;
        }
    }
    c.nr.push_back(c.nr[c.nr.size() - 1] + (c.an.size() -
       c.nr[c.nr.size() - 1]));
    return c;
}
```

Листинг 3.8 – Реализация алгоритма генерации данных для матрицы в РСФ

```
1  void generate_mtr_csr(matrix_csr_t &a, size_t n, size_t m,
       size_t num_cnt) {
2      int cnt = 0, prev = 0;
3      a.n = n;
4      a.m = m;
5
6      for (size_t i = 0; i < n; i++) {
7          srand(clock() % 1000000);
8          int val = 0;
9          for (size_t j = 0; cnt < num_cnt && j < m; j++) {
10             if ((rand() % 15) - 5 > 0) {
11                 a.an.push_back(rand() % 10 + 1);
12                 a.ja.push_back(j);
13                 cnt++;
14                 val++;
15             }
16         }
17
18         if (i == 0)
19             a.nr.push_back(0);
20         else if (val == 0) {
21             if (prev == 0)
22                 a.nr.push_back(a.nr[i - 1]);
23             else
24                 a.nr.push_back(a.nr[i - 1] + prev);
25         }
26         else
27             a.nr.push_back(a.nr[i - 1] + prev);
28         prev = val;
29     }
30     a.nr.push_back(a.nr[n - 1] + (a.an.size() - a.nr[n - 1]));
31 }
```

Листинг 3.9 – Реализация алгоритма распоковки матрицы РСФ

```
matrix_t decomprass(matrix_csr_t &a) {
    matrix_t mtr;
    mtr.m = a.m;
    mtr.n = a.n;

    for (int i = 0; i < a.n; i++ )
    {
        mtr.buff.emplace_back();
        for (int j = 0; j < a.m; j++ )
            mtr.buff.back().push_back(0);
    }

    int mtr_i = 0;
    for (int i = 0; i < a.n; i++) {
        for (int j = a.nr[i]; j < a.nr[i + 1]; j++)
            mtr.buff[mtr_i][a.ja[j]] = a.an[j];
        mtr_i++;
    }

    return mtr;
}
```

Вывод

В данном разделе была приведена информация о выбранных средствах для разработки алгоритмов. Для реализации алгоритмов был выбран язык C++. Были представлены листинги для каждой из реализаций работы конвейера и функциональные тесты для этапов конвейера

4 Исследовательская часть

В данном разделе приведены технические характеристики устройства, на котором проводилось измерение времени работы программного обеспечения, а также результаты замеров времени.

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялись замеры по времени представлены далее.

— Процессор: Intel(R) Core(TM) i5-10300H CPU 2.50 ГГц [8].

— Количество ядер: 4 физических и 8 логических ядер.

— Оперативная память: 16 ГБайт.

— Операционная система: Windows 11 Pro 64-разрядная система [9].

При замерах времени ноутбук был включен в сеть электропитания и был нагружен только системными приложениями.

4.2 Демонстрация работы программы

На рисунке 4.1 представлен пример результата работы программы. Пользователь, указывая соответствующие пункты меню, запускает последовательную обработку заявок, затем параллельное исполнение конвейера, затем выходит из программы. Лог программы при этом на экран не выводится — он записывается в файл. На рисунке 4.2 представлен пример лог–файла.

```
Вас приветствует измеритель времени конвейерной обработки разреженных матриц!!!



Выберите необходимую задачу и введите её номер для запуска.
1 - запустить последовательную обработку матриц
2 - запустить конвейерную обработку матриц
3 - замеры времени реализаций
0 - выход

Ваш выбор: 1
Введите количество заявок (больше 0): 10
Введите количество строк n (больше 0): 10
Введите количество столбцов m (больше 0): 10
Введите количество элементов (больше 0): 40
```

Рисунок 4.1 – Пример работы программы

```
Request 0 start creating: 0 ns
Request 0 end creating: 34900 ns
Request 0 start sum: 34900 ns
Request 0 end sum: 49800 ns
Request 0 start unpack: 49900 ns
Request 0 end unpack: 77500 ns
Request 1 start creating: 3656900 ns
Request 1 end creating: 3705800 ns
Request 1 start sum: 3705900 ns
Request 1 end sum: 3717000 ns
Request 1 start unpack: 3717000 ns
Request 1 end unpack: 3742200 ns
Request 2 start creating: 6480900 ns
Request 2 end creating: 6530600 ns
Request 2 start sum: 6530600 ns
Request 2 end sum: 6541800 ns
Request 2 start unpack: 6541900 ns
Request 2 end unpack: 6568200 ns
Request 3 start creating: 9318600 ns
Request 3 end creating: 9364100 ns
Request 3 start sum: 9364100 ns
Request 3 end sum: 9374700 ns
Request 3 start unpack: 9374700 ns
Request 3 end unpack: 9409200 ns
Request 4 start creating: 12138500 ns
Request 4 end creating: 12183400 ns
Request 4 start sum: 12183400 ns
Request 4 end sum: 12193900 ns
Request 4 start unpack: 12194000 ns
Request 4 end unpack: 12222400 ns
Request 5 start creating: 14956800 ns
Request 5 start sum: 14982900 ns
Request 5 end creating: 14982900 ns
Request 5 end sum: 14993300 ns
Request 5 start unpack: 14993300 ns
Request 5 end unpack: 15022000 ns
Request 6 start creating: 17914100 ns
Request 6 end creating: 17947300 ns
Request 6 start sum: 17947300 ns
```

Рисунок 4.2 – Пример файла с логом работы конвейера

4.3 Временные характеристики

Для замеров времени использовалась функция получения значения системных часов *clock_gettime*() [4]. Функция применялась два раза — в начале и в конце измерения времени, значения полученных временных меток вычитались друг из друга для получения времени выполнения программы. Замеры проводились по 100 раз для набора заявок от 10 до 100 штук с шагом 10.

В таблице 4.1 представлены замеры времени выполнения двух реализаций конвейерной обработки в зависимости от количества заявок.

Таблица 4.1 – Результаты нагрузочного тестирования (в мкс)

Кол-во заявок	**Время, мкс**	
	Последовательный	**Параллельный**
10	275.14	441.83
20	571.80	632.01
30	811.11	755.36
40	1069.60	1022.27
50	1325.81	1168.49
60	1585.55	1447.42
70	1853.32	1614.98
80	2112.91	1821.58
90	2374.95	1993.78
100	2644.65	2176.60

На рисунке 4.3 приведен график результатов замеров для различных значений количества.

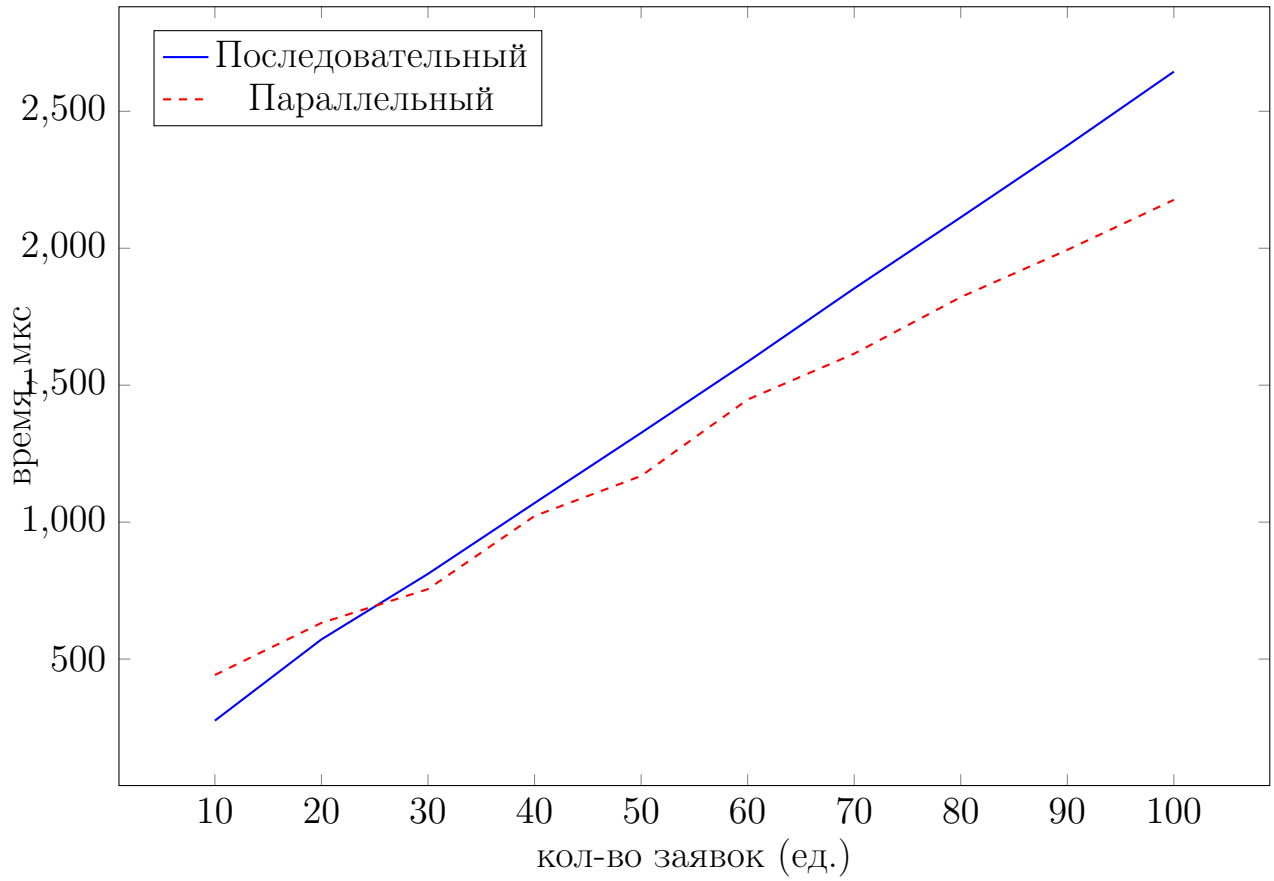

Рисунок 4.3 – Результаты замеров времени работы реализации конвейерной обработки

4.4 Вывод

В результате эксперимента было получено, что использование конвейерной обработки лучше по времени линейной реализации на 100 заявках примерно в 1.5 раза. В силу линейности графиков на рисунке 4.3 можно сказать, что на достаточно большом количестве заявок выигрыш параллельной обработки над последовательной во времени в абсолютных единицах будет увеличиваться.

Заключение

В ходе выполнения лабораторной работы было выявлено, что в результате использования параллельной реализации конвейерной обработки примерно в 1.5 раза лучше работает, чем последовательная конвейерная обработка.

Цель, поставленная в начале работы, была достигнута. Кроме того были достигнуты все поставленные задачи.

1) Описана организация конвейерной обработки данных.

2) Описаны алгоритмы обработки данных, которые будут использоваться в текущей лабораторной работе.

3) Реализованы программу, реализующую конвейер с количеством лент не менее трех в однопоточной и многопоточной среде.

4) Сравнены и проанализированы реализации алгоритмов по затраченному времени.

Список использованных источников

1 Конвейерная обработка данных [Электронный ресурс]. — Режим доступа: https://studref.com/636041/ekonomika/konveyernaya_obrabotka_dannyh (дата обращения: 28.01.2023).

2 С. Писсанецки. Технология разреженных матриц. — М.: Издательство «МИР», 1998. — С. 410.

3 C++ language [Электронный ресурс]. — Режим доступа: https://en.cppreference.com/ (дата обращения: 28.01.2023).

4 *clock_gettime* function [Электронный ресурс]. — Режим доступа: https://man7.org/linux/man-pages/man3/clock_gettime.3.html (дата обращения: 28.01.2023).

5 Класс thread [Электронный ресурс]. — Режим доступа: https://learn.microsoft.com/ru-ru/cpp/standard-library/thread-class (дата обращения: 28.01.2023).

6 Класс mutex [Электронный ресурс]. — Режим доступа: https://learn.microsoft.com/ru-ru/cpp/standard-library/mutex-class-stl (дата обращения: 28.01.2023).

7 Класс queue [Электронный ресурс]. — Режим доступа: https://learn.microsoft.com/ru-ru/cpp/standard-library/queue-class (дата обращения: 28.01.2023).

8 Intel [Электронный ресурс]. — Режим доступа: https://ark.intel.com/content/www/ru/ru/ark/products/201839/intel-core-i510300h-processor-8m-cache-up-to-4-50-ghz.html (дата обращения: 25.09.2022).

9 Windows 10 Pro 2h21 64-bit [Электронный ресурс]. — Режим доступа: https://www.microsoft.com/ru-ru/software-download/windows10 (дата обращения: 25.09.2022).