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»

НАПРАВЛЕНИЕ ПОДГОТОВКИ «09.03.04 Программная инженерия»

ОТЧЕТ

ПО ЛАБОРАТОРНОЙ РАБОТЕ №5

Название: Конвейерная обработка данных

Дисциплина: Анализ алгоритмов

Студент	ИУ7-54Б		С. Д. Параскун
	Группа	Подпись, дата	И. О. Фамилия
Преподаватель			Л. Л. Волкова
		Подпись, дата	И. О. Фамилия

Москва, 2021 г.

Содержание

Страница

Введение

Конвейер - механизм организации труда, когда производство изделия разбивается на стадии и конкретные работники закрепляются за своими стадиями (а часто и за линиями конвейера). Данный способ работы оказался эффективнее ранее используемых, в связи с чем такой термин как "конвейерная обработка"перекочевал и в программирование.

В его терминах конвейерная обработка данных - создание для определенных этапов решения задачи потоков, выполняющих свою работу на данных, полученных от предыдущей линии конвейера (другого потока). Использование многопоточности позволило существенно сократить время выполнения поставленных задач.

Целью данной лабораторной работы является получение навыков конвейерной обработки данных. Для достижения поставленной цели необходимо выполнить следующие задачи:

- изучить основы конвейерной обработки данных;
- составить схемы алгоритмов основной работы конвейера и его этапов;
- реализовать разработанные алгоритмы;
- провести сравнительный анализ конвейерной и последовательной реализаций по затрачиваемым ресурсам (время и память);
- описать и обосновать полученные результаты.

1. Аналитический раздел

В данном разделе будут представлены описания конвейерной обработки данных, а также используемых на линиях алгоритмов.

1.1 Конвейерная обработка данных

Конвейер [1] — способ организации вычислений, используемый в современных процессорах и контроллерах с целью повышения их производительности (увеличения числа инструкций, выполняемых в единицу времени — эксплуатация параллелизма на уровне инструкций), технология, используемая при разработке компьютеров и других цифровых электронных устройств.

Идея заключается в параллельном выполнении нескольких инструкций процессора. Сложные инструкции процессора представляются в виде последовательности более простых стадий. Вместо выполнения инструкций последовательно (ожидания завершения конца одной инструкции и перехода к следующей), следующая инструкция может выполняться через несколько стадий выполнения первой инструкции. Это позволяет управляющим цепям процессора получать инструкции со скоростью самой медленной стадии обработки, однако при этом намного быстрее, чем при выполнении эксклюзивной полной обработки каждой инструкции от начала до конца.

1.2 Этапы конвейерной обработки

В данной лабораторной работе работа конвейера будет осуществлена на 3 линиях. Каждая из них выполняет следующие этапы:

1. перемножение матриц (стандартный алгоритм);
2. вычисление определителя матрицы;
3. определение, является ли модуль определителя простым числом.

Данный алгоритм будет реализован с помощью многопоточности - под каждую линию конвейера будет выделено по потоку [2].

1.3 Вывод

В данном разделе были рассмотренны принципы конвейерной обработки данных, алгоритмы на линиях самого конвейера. Полученных знаний достаточно для разработки. На вход конвейеру будет подаваться односвязный список, каждый элемент которого содержит в себе следующие поля: указатели на исходные матрицы, указатель на результирующую матрицу, указатель на числовую переменную (под определитель), указатель на булевую переменную (определитель - простое число или нет) и указатель на следующий элемент списка.

Реализуемое ПО будет работать в пользовательском режиме (вывод результатов конвейерной обработки входных данных), а также в экспериментальном (проведение замеров времени выполнения алгоритмов).

2. Конструкторский раздел

В данном разделе будут спроектированы схемы алгоритмов, описаны используемые типы данных, а также произведена оценка памяти и описана структура ПО.

2.1 Схемы алгоритмов

На рисунках 2.1 - 2.2 представлены схемы алгоритмов работы конвейера.

На рисунках 2.3 - 2.5 представлены схемы алгоритмов на конкретных лентах конвейера.

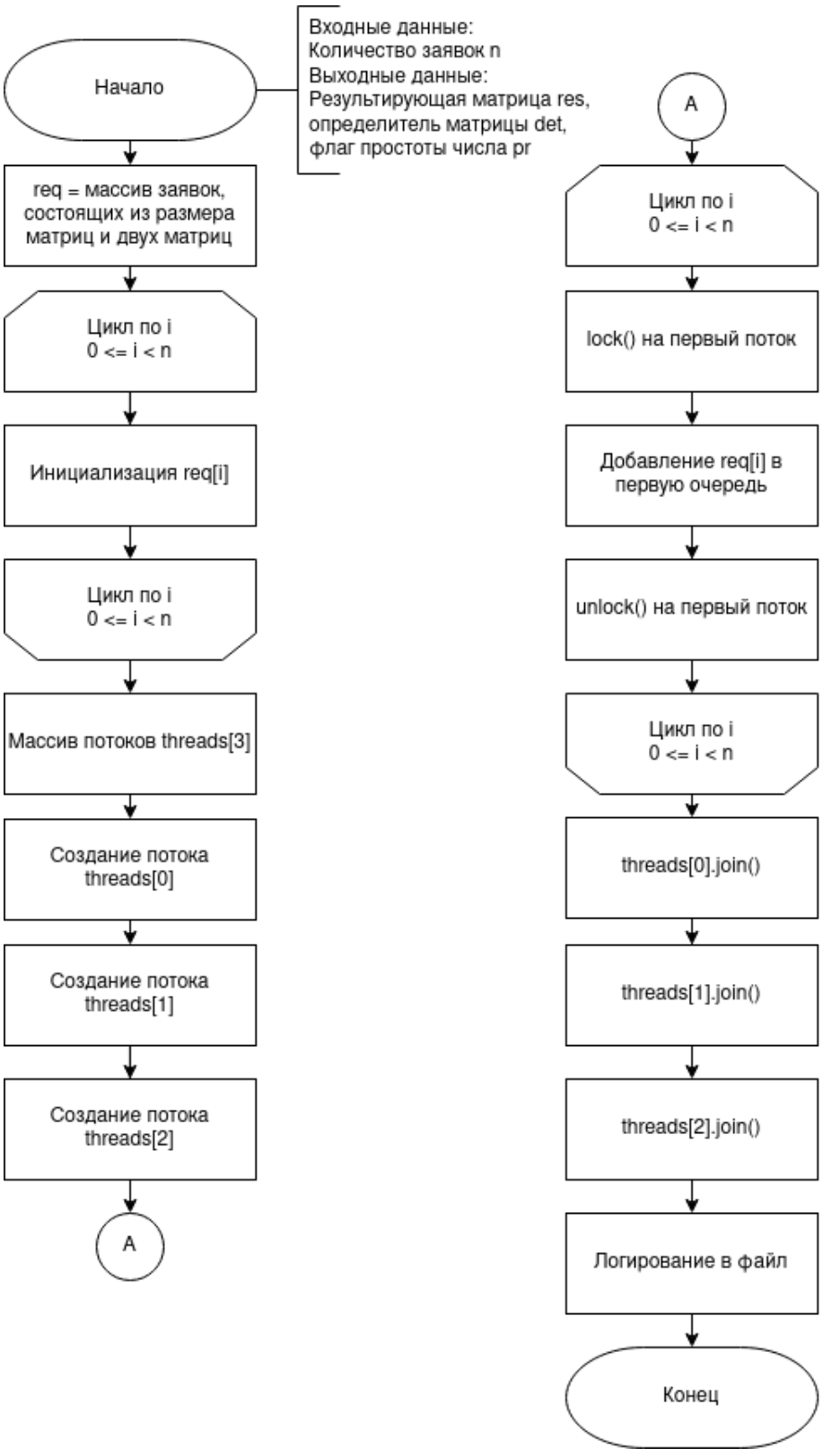

Рисунок 2.1 – Схема главного процесса конвейерной системы

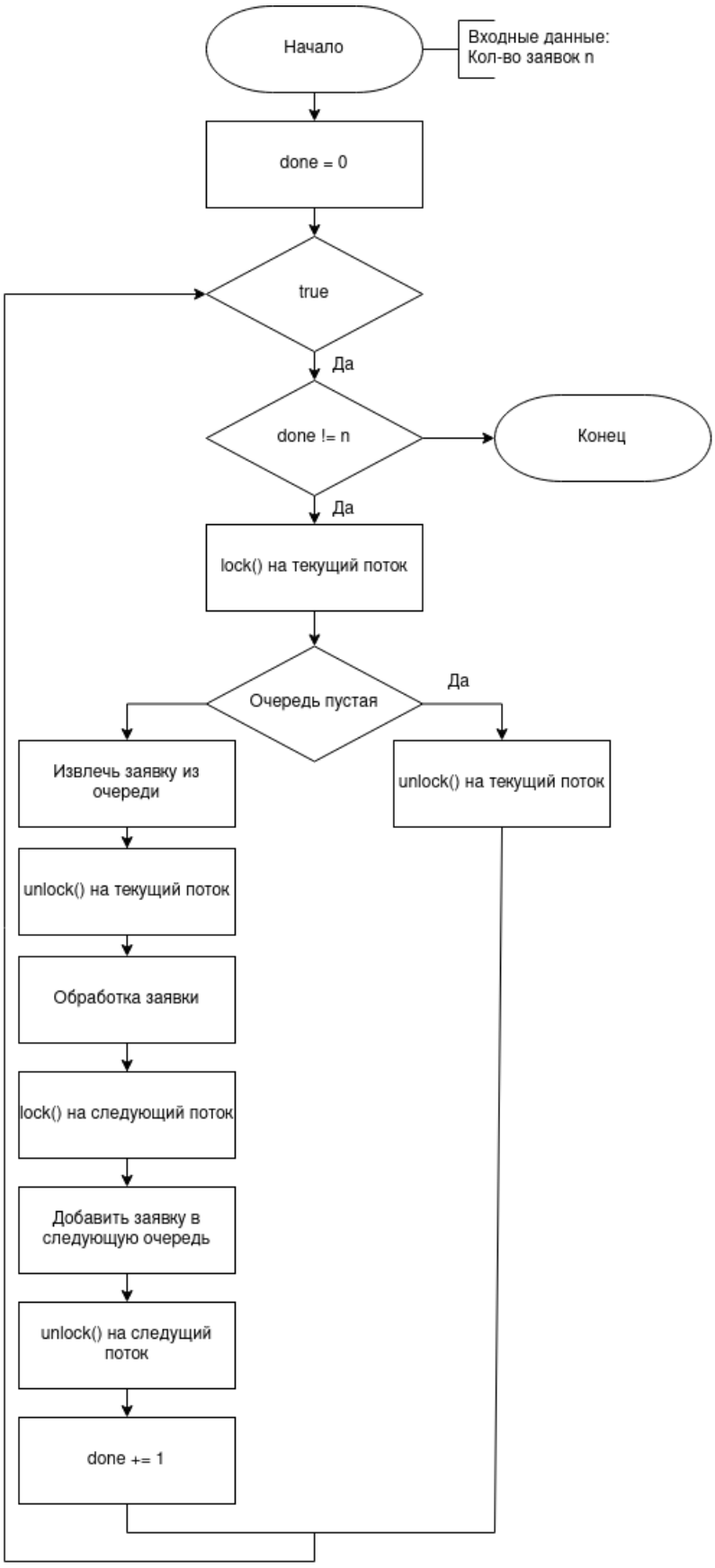

Рисунок 2.2 – Схема работы отдельных лент конвейерной системы

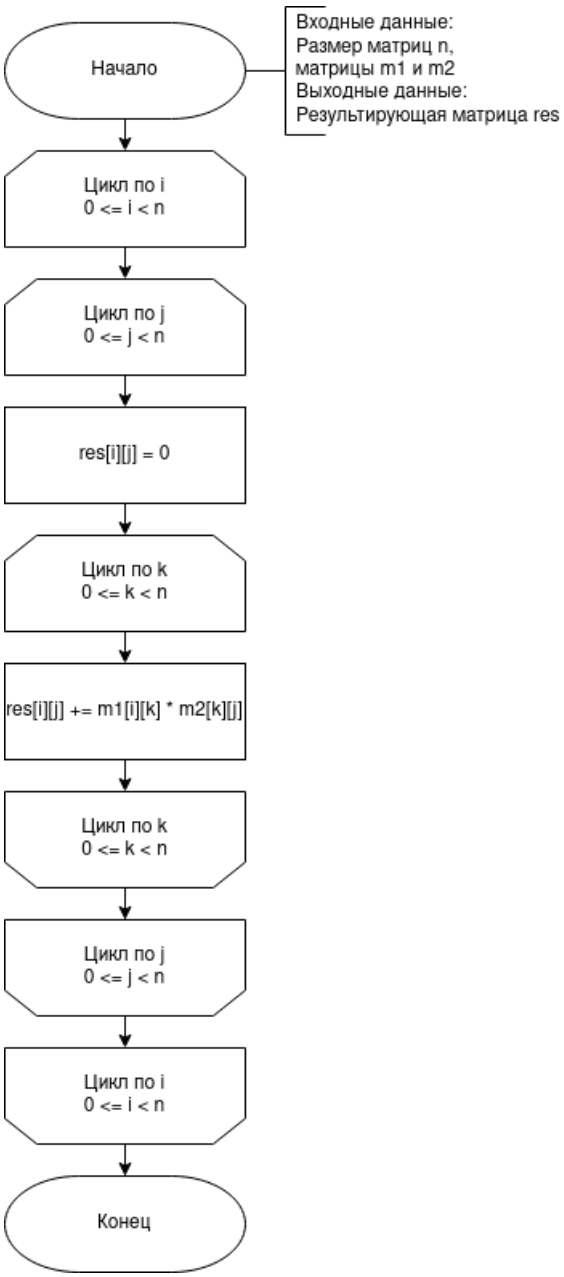

Рисунок 2.3 – Схема алгоритма умножения матриц (первая лента)

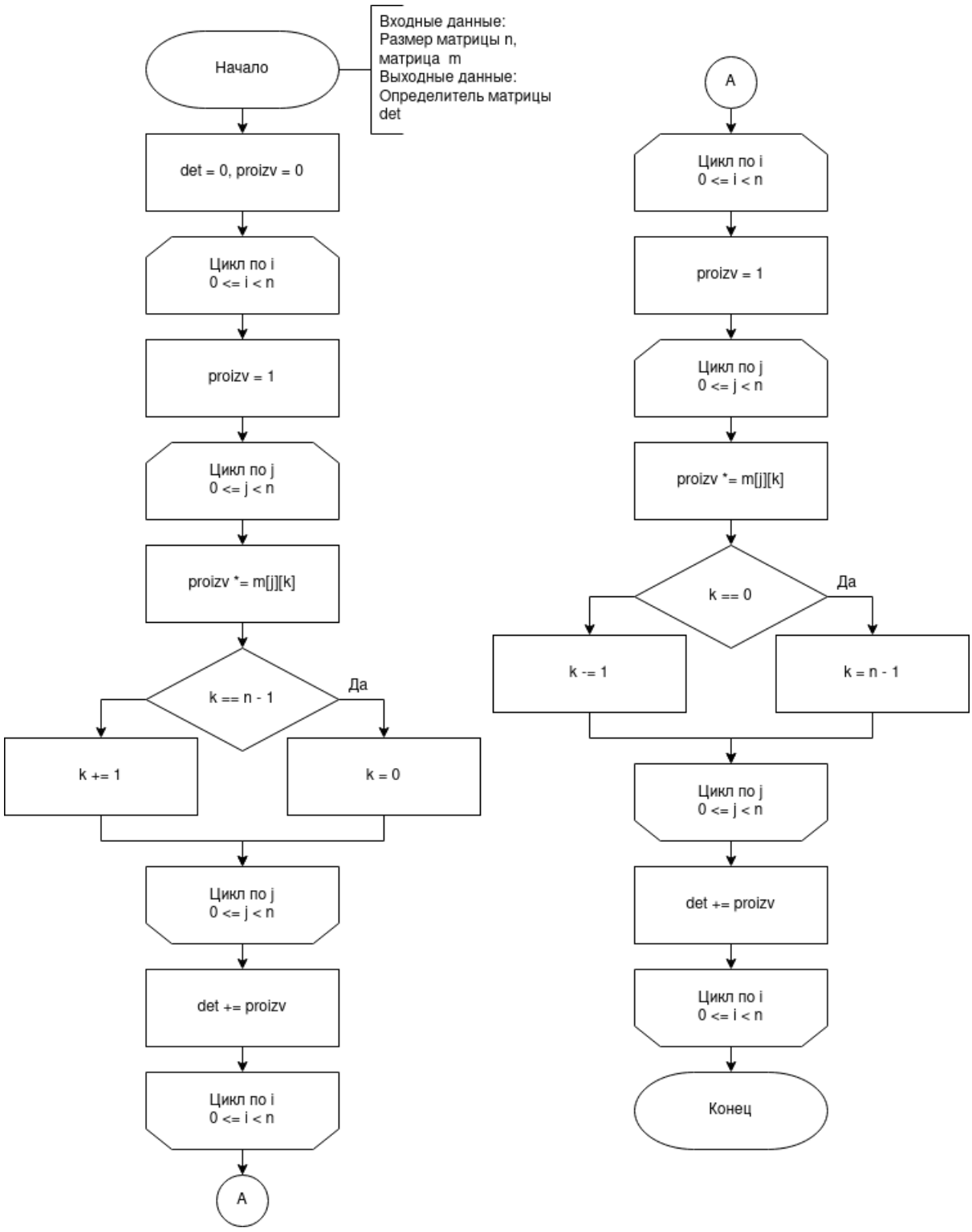

Рисунок 2.4 – Схема алгоритма вычисления определителя матрицы (вторая лента)

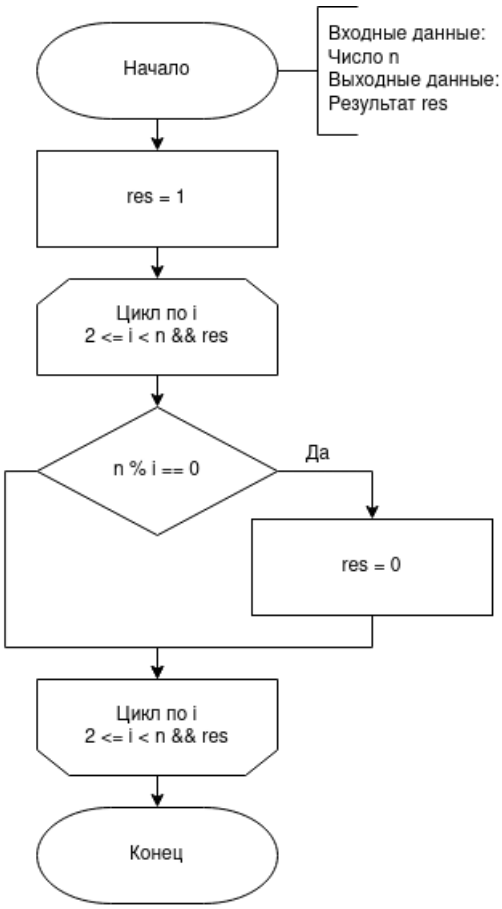

Рисунок 2.5 – Схема алгоритма определения числа на простоту (третья лента)

2.2 Используемые типы данных

При реализации алгоритмов будут использованы следующие структуры данных:

- потоки - массив типа pthread_t;
- мьютексы - элементы типа pthread_mutex_t;
- элементы очередей для каждой ленты (листинг 2.1);

Листинг 2.1 – Тип данных элементов очереди

```
1     struct flItem
2     {
3         int n;
4         int **m1;
5         int **m2;
6         int **res;
7         struct flItem *next;
8     };
9
10    struct slItem
11    {
12        int n;
13        int **m;
14        int res;
15        struct slItem *next;
16    };
17
18    struct tlItem
19    {
20        int n;
21        int res;
22        struct tlItem *next;
23    };
24
25    struct reslItem
26    {
27        int res;
28        struct reslItem *next;
29    };
```

- общие данные на весь конвейер (листинги 2.2).

Листинг 2.2 – Тип данных для общей информации по конвейеру

```
1    struct queue
2    {
3        int num;
4        struct flItem *fstart;
5        struct flItem *fend;
6        struct slItem *sstart;
7        struct slItem *send;
8        struct tlItem *tstart;
9        struct tlItem *tend;
10       struct reslItem *resstart;
11       struct reslItem *resend;
12       pthread_mutex_t m1;
13       pthread_mutex_t m2;
14       pthread_mutex_t m3;
15       pthread_mutex_t resm;
16       int startTime;
17       int fileMatrix[1000][3][2];
18   };
```

2.3 Оценка памяти

Рассмотрим затрачиваемый объем памяти для рассмотренных алгоритмов.

При последовательном вычислении память используется на:

- входные и выходные параметры функций - 3 * n * n * sizeof(int) + 3 * sizeof(int);
- вспомогательные переменные - 8 * sizeof(int).

В случае конвейерного вычисления память затрачивается также на организацию очереди, а именно:

- кол-во заявок в очереди - sizeof(int);
- элементы первой очереди - n * sizeof(flItem);
- элементы второй очереди - n * sizeof(slItem);
- элементы третьей очереди - n * sizoef(tlItem);

- элементы результирующей очереди - n * sizeof(reslItem);
- мьютексы - 4 * sizoef($pthread\_mutex\_t$);
- потоки - 3 * sizeof($pthread\_t$).

Таким образом, для конвейерного вычисления затрачивается памяти больше в среднем на n * кол-во заявок в очереди.

2.4 Структура ПО

ПО будет состоять из следующих модулей:

- главный модуль - из него будет осуществляться запуск программы и выбор соответствующего режима работы;
- модуль интерфейса - в нем будет описана реализация режимов работы программы;
- модуль, содержащий реализации алгоритмов.

2.5 Вывод

На основе полученных в аналитическом разделе знаний об алгоритмах были спроектированы схемы алгоритмов, выбраны используемые типы данных, проведена оценка затрачиваемого объема памяти, а также описана структура ПО.

3. Технологический раздел

В данном разделе будут приведены требования к ПО, средства его реализации и листинга кода алгоритмов, а также рассмотрены тестовые случаи.

3.1 Требования к ПО

Программное обеспечение должно удовлетворять следующим требованиям:

- ПО принимает количество заявок на конвейере;
- ПО возвращает целое число - 1 если определитель матрицы произведения двух исходных матриц - простое число, 0 иначе.

3.2 Средства реализации

Для реализации ПО был выбран компилируемый язык C. В качестве среды разработки - QtCreator. Оба средства были выбраны из тех соображений, что навыки работы с ними были получены в более ранних курсах.

3.3 Листинги кода

Листинги 3.1 - 3.4 демонстрируют реализацию работы конвейера.

Листинг 3.5 демонстрирует работу с очередями заявок для каждой линии конвейера.

Листинги 3.6 - 3.8 демонстрируют функции, реализуемые на конкретных лентах конвейера.

Листинг 3.1 – Функция общего запуска конвейера

```
1 void startPipeline(int n, int size)
2 {
3     int tNum = 3;
4     pthread_t threads[tNum];
```

```
5       struct queue data;
6       data.fstart = NULL;
7       data.fend = NULL;
8       data.sstart = NULL;
9       data.send = NULL;
10      data.tstart = NULL;
11      data.tend = NULL;
12      data.resstart = NULL;
13      data.resend = NULL;
14      data.num = n;
15      pthread_mutex_init(&(data.m1), NULL);
16      pthread_mutex_init(&(data.m2), NULL);
17      pthread_mutex_init(&(data.m3), NULL);
18      pthread_mutex_init(&(data.resm), NULL);
19      pthread_create(&(threads[0]), NULL, startFirst, &data);
20      pthread_create(&(threads[1]), NULL, startSecond, &data);
21      pthread_create(&(threads[2]), NULL, startThird, &data);
22
23      for (int i = 0; i < n; i++)
24      {
25          pthread_mutex_lock(&(data.m1));
26          firstPush(&data, size);
27          printf("[%d]: Start matrix:\n", i + 1);
28          outputMatrix(data.fend->n, data.fend->m1);
29          printf("\n");
30          outputMatrix(data.fend->n, data.fend->m2);
31          pthread_mutex_unlock(&(data.m1));
32      }
33      data.startTime = tick();
34
35      for (int i = 0; i < tNum; i++)
36      pthread_join(threads[i], NULL);
37
38      for (int i = 0; i < n; i++)
39      {
40          printf("[%d]: Determinator is prime? %d\n", i + 1,\
41           data.resstart->res);
42          resPop(&data);
43      }
44
45      FILE *f = fopen("texRes.txt", "w");
46      for (int i = 0; i < n; i++)
47      {
48          fprintf(f, "Tape 1 & Task %d & %d & %d\\\\\n", i + 1,\
49           data.fileMatrix[i][0][0], data.fileMatrix[i][0][1]);
50          fprintf(f, "Tape 2 & Task %d & %d & %d\\\\\n", i + 1,\
51           data.fileMatrix[i][1][0], data.fileMatrix[i][1][1]);
52          fprintf(f, "Tape 3 & Task %d & %d & %d\\\\\n", i + 1,\
```

```
53          data.fileMatrix[i][2][0], data.fileMatrix[i][2][1]);
54          fprintf(f, "\\hline\n");
55      }
56      fclose(f);
57
58      pthread_mutex_destroy(&(data.m1));
59      pthread_mutex_destroy(&(data.m2));
60      pthread_mutex_destroy(&(data.m3));
61      pthread_mutex_destroy(&(data.resm));
62  }
```

Листинг 3.2 – Первая лента конвейера

```
1  void *startFirst(void *firstData)
2  {
3      struct queue *data = (struct queue *) firstData;
4      int done = 0;
5      while (1)
6      {
7          if (done == data->num)
8              break;
9          pthread_mutex_lock(&(data->m1));
10         if (data->fstart == NULL)
11         {
12             pthread_mutex_unlock(&(data->m1));
13             continue;
14         }
15
16         data->fileMatrix[done][0][0] = tick() - data->startTime;
17         pthread_mutex_unlock(&(data->m1));
18         multMatrix(data->fstart->n, data->fstart->m1, data->fstart->m2,\
19          data->fstart->res);
20         pthread_mutex_lock(&(data->m2));
21         data->fileMatrix[done][0][1] = tick() - data->startTime;
22         secondPush(data);
23         pthread_mutex_unlock(&(data->m2));
24
25         done++;
26     }
27     return NULL;
28 }
```

Листинг 3.3 – Вторая лента конвейера

```
1  void *startSecond(void *secondData)
2  {
3      struct queue *data = (struct queue *) secondData;
4      int done = 0;
```

```
    while (1)
    {
        if (done == data->num)
            break;
        pthread_mutex_lock(&(data->m2));
        if (data->sstart == NULL)
        {
            pthread_mutex_unlock(&(data->m2));
            continue;
        }

        data->fileMatrix[done][1][0] = tick() - data->startTime;
        pthread_mutex_unlock(&(data->m2));
        data->sstart->res = determinator(data->sstart->n, data->sstart->m);
        pthread_mutex_lock(&(data->m3));
        data->fileMatrix[done][1][1] = tick() - data->startTime;
        thirdPush(data);
        pthread_mutex_unlock(&(data->m3));

        done++;
    }
    return NULL;
}
```

Листинг 3.4 – Третья лента конвейера

```
void *startThird(void *thirdData)
{
    struct queue *data = (struct queue *) thirdData;
    int done = 0;
    while (1)
    {
        if (done == data->num)
            break;
        pthread_mutex_lock(&(data->m3));
        if (data->tstart == NULL)
        {
            pthread_mutex_unlock(&(data->m3));
            continue;
        }

        data->fileMatrix[done][2][0] = tick() - data->startTime;
        pthread_mutex_unlock(&(data->m3));
        data->tstart->res = prime(data->tstart->n);
        pthread_mutex_lock(&(data->resm));
        data->fileMatrix[done][2][1] = tick() - data->startTime;
        resPush(data);
        pthread_mutex_unlock(&(data->resm));
```

```
23
24          done++;
25      }
26      return NULL;
27  }
```

Листинг 3.5 – Функции работы с очередями заявок

```
1  void firstPush(struct queue *data, int n)
2  {
3      struct flItem *new = malloc(sizeof(struct flItem));
4      new->n = n;
5      new->m1 = allocMatrix(n);
6      fillMatrix(n, new->m1);
7      new->m2 = allocMatrix(n);
8      fillMatrix(n, new->m2);
9      new->res = allocMatrix(n);
10     new->next = NULL;
11     if (data->fstart == NULL)
12     {
13         data->fstart = new;
14         data->fend = new;
15     }
16     else
17     {
18         data->fend->next = new;
19         data->fend = new;
20     }
21 }
22
23 void firstPop(struct queue *data)
24 {
25     if (data->fstart != data->fend)
26     {
27         struct flItem *buf = data->fstart;
28         data->fstart = buf->next;
29         free(buf->m1);
30         free(buf->m2);
31         free(buf);
32     }
33     else
34     {
35         free(data->fstart->m1);
36         free(data->fstart->m2);
37         free(data->fstart);
38         data->fstart = data->fend = NULL;
39     }
40 }
```

```
41
42 void secondPush(struct queue *data)
43 {
44     struct slItem *new = malloc(sizeof(struct slItem));
45     struct flItem *old = data->fstart;
46     new->n = old->n;
47     new->m = old->res;
48     if (data->sstart == NULL)
49     {
50         data->sstart = new;
51         data->send = new;
52     }
53     else
54     {
55         data->send->next = new;
56         data->send = new;
57     }
58     firstPop(data);
59 }
60
61 void secondPop(struct queue *data)
62 {
63     if (data->sstart != data->send)
64     {
65         struct slItem *buf = data->sstart;
66         data->sstart = buf->next;
67         free(buf->m);
68         free(buf);
69     }
70     else
71     {
72         free(data->sstart->m);
73         free(data->sstart);
74         data->sstart = data->send = NULL;
75     }
76 }
77
78 void thirdPush(struct queue *data)
79 {
80     struct tlItem *new = malloc(sizeof(struct tlItem));
81     struct slItem *old = data->sstart;
82     new->n = old->res;
83     if (data->tstart == NULL)
84     {
85         data->tstart = new;
86         data->tend = new;
87     }
88     else
```

```
89      {
90          data->tend->next = new;
91          data->tend = new;
92      }
93      secondPop(data);
94  }
95
96  void thirdPop(struct queue *data)
97  {
98  if (data->tstart != data->tend)
99      {
100         struct tlItem *buf = data->tstart;
101         data->tstart = buf->next;
102         free(buf);
103     }
104     else
105     {
106         free(data->tstart);
107         data->tstart = data->tend = NULL;
108     }
109 }
110
111 void resPush(struct queue *data)
112 {
113     struct reslItem *new = malloc(sizeof(struct reslItem));
114     struct tlItem *old = data->tstart;
115     new->res = old->res;
116     if (data->resstart == NULL)
117     {
118         data->resstart = new;
119         data->resend = new;
120     }
121     else
122     {
123         data->resend->next = new;
124         data->resend = new;
125     }
126     thirdPop(data);
127 }
128
129 int resPop(struct queue *data)
130 {
131     int res;
132     if (data->resstart != data->resend)
133     {
134         struct reslItem *buf = data->resstart;
135         res = buf->res;
136         data->resstart = buf->next;
```

```
137         free(buf);
138     }
139     else
140     {
141         res = data->resstart->res;
142         free(data->resstart);
143         data->resstart = data->resend = NULL;
144     }
145     return res;
146 }
```

Листинг 3.6 – Функция вычисления произведения матриц (первая лента)

```
1  void multMatrix(int n, int **m1, int **m2, int **res)
2  {
3      for (int i = 0; i < n; i++)
4      {
5          for (int j = 0; j < n; j++)
6          {
7              res[i][j] = 0;
8              for (int k = 0; k < n; k++)
9                  res[i][j] += m1[i][k] * m2[k][j];
10         }
11     }
12 }
```

Листинг 3.7 – Функция вычисления определителя матрицы (вторая лента)

```
1  int determinator(int n, int **m)
2  {
3      int sum = 0, proizv = 0;
4      for(int i= 0; i< n; i++)
5      {
6          proizv = 1;
7          for(int j= 0, k= i; j< n; j++, k= (k==n-1)?0: k+1 )
8          proizv *= m[j][k];
9          sum += proizv;
10     }
11     for(int i= 0; i< n; i++)
12     {
13         proizv= 1;
14         for(int j= 0, k= i; j< n; j++, k= (k==0)?n - 1: k-1 )
15         proizv *= m[j][k];
16         sum -= proizv;
17     }
18     return sum;
19 }
```

Листинг 3.8 – Функция проверки числа на простоту

```
int prime(int n)
{
    int res = 1;
    for (int i = 2; i < n && res; i++)
        if (n % i == 0)
            res = 0;
    return res;
}
```

3.4 Тестирование ПО

В таблице 3.1 приведены тестовые случаи для конвейерной обработки. Случаи 1-2 - результат простое число, случаи 3-4 - результат не простое число.

Таблица 3.1 – Тестовые случаи

№	Матрица 1	Матрица 2	Простое число
1	$\begin{pmatrix} 2 & 6 & 1 \\ 8 & 7 & 9 \\ 2 & 0 & 2 \end{pmatrix}$	$\begin{pmatrix} 3 & 7 & 5 \\ 9 & 2 & 2 \\ 8 & 9 & 7 \end{pmatrix}$	1
2	$\begin{pmatrix} 6 & 3 & 2 \\ 0 & 6 & 1 \\ 5 & 5 & 4 \end{pmatrix}$	$\begin{pmatrix} 7 & 6 & 5 \\ 6 & 9 & 3 \\ 7 & 4 & 5 \end{pmatrix}$	1
3	$\begin{pmatrix} 3 & 6 & 1 \\ 2 & 9 & 3 \\ 1 & 9 & 4 \end{pmatrix}$	$\begin{pmatrix} 7 & 8 & 4 \\ 5 & 0 & 3 \\ 6 & 1 & 0 \end{pmatrix}$	0
4	$\begin{pmatrix} 3 & 6 & 7 \\ 5 & 3 & 5 \\ 6 & 2 & 9 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 & 7 \\ 0 & 9 & 3 \\ 6 & 0 & 6 \end{pmatrix}$	0

3.5 Вывод

На основе схем из конструкторского раздела были написаны реализации требуемых алгоритмов, а также проведено их тестирвание.

4. Исследовательский раздел

В данном разделе будет проведен сравнительный анализ алгоритмов по времени и затрачиваемой памяти.

4.1 Технические характеристики

Тестирование выполнялось на устройстве со следующими характеристиками:

- Операционная система Ubuntu 20.04.3 LTS [3]
- Память 16 Гб
- Процессор Intel Core i5-1135G7 11th Gen, 2.40 Гц [4]

Во время проведения эксперимента устройство не было нагружено сторонними задачами, а также было подключено к блоку питания.

Замеры времени осуществлялись с помощью ассемблерной вставки, описанной в листинге 4.1.

Листинг 4.1 – Замер времени в микросекундах

```
unsigned long long tick(void)
{
    unsigned long long d;
    __asm__ __volatile__ ("rdtsc": "=A" (d));
    return d;
}
```

При проведении эксперимента была отключена оптимизация командой компилятору -o0 [5].

4.2 Оценка времени работы алгоритмов

В таблице 4.1 представлен лог выполнения программы для 10 задач и входных матриц размером 3 * 3.

Таблица 4.1 – Лог выполнения программы

Tape N	Task M	Time start	Time end
Tape 1	Task 1	78467	82983
Tape 2	Task 1	226714	235872
Tape 3	Task 1	409641	410092
Tape 1	Task 2	84450	87009
Tape 2	Task 2	248334	249506
Tape 3	Task 2	410148	410892
Tape 1	Task 3	88818	90289
Tape 2	Task 3	250799	251421
Tape 3	Task 3	411642	412268
Tape 1	Task 4	91264	92640
Tape 2	Task 4	251943	252526
Tape 3	Task 4	412664	413302
Tape 1	Task 5	94380	95808
Tape 2	Task 5	253016	253607
Tape 3	Task 5	413625	414188
Tape 1	Task 6	96975	98261
Tape 2	Task 6	254168	254650
Tape 3	Task 6	414517	414855
Tape 1	Task 7	99188	100473
Tape 2	Task 7	255138	255627
Tape 3	Task 7	415174	415826
Tape 1	Task 8	101519	102822
Tape 2	Task 8	256095	256609
Tape 3	Task 8	416141	416466
Tape 1	Task 9	103842	105246
Tape 2	Task 9	257814	258304
Tape 3	Task 9	416777	417111
Tape 1	Task 10	106273	107571
Tape 2	Task 10	258729	259300
Tape 3	Task 10	417397	417726

В таблице 4.2 представлены замеры времени обработки и простоя заявок в системе при параллельном и последовательном выполнении.

Таблица 4.2 – Анализ временных замеров

Характеристика		Парал., тики			Последоват., тики		
Линия		1	2	3	1	2	3
Простой очереди	min	1028	378	327	1200	40554	41048
	max	5421	13795	6875	14274	73787	91762
	avg	1190	1192	732	7478	42908	83574
Время заявки в системе	min	45238			41325		
	max	179982			73655		
	avg	102424			43826		

4.3 Вывод

По результатам исследования можно сделать вывод, что простой заявок в системе оказывается значительно меньшим при конвейерной реализации обработки. Однако Общее время заявок в системе при такой работе оказывается больше, это связано с занятостью лент конвейера, в отличие от последовательной реализации, когда вся заявка обрабатывается лентами до полного выхода из системы.

Заключение

В ходе выполнения данной лабораторной работы были выполнены следующие задачи:

- изучены основы конвейерной обработки данных;
- составлены схемы алгоритмов основной работы конвейера и его этапов;
- реализованны разработанные алгоритмов;
- проведен сравнительный анализ конвейерной и последовательной реализаций по затрачиваемым ресурсам (время и память);
- описаны и обоснованы полученные результаты.

В результате исследований можно прийти к выводу, что использование конвейерной обработки данных оптимальнее последовательного в тех случаях, когда выполняемые задачи намного больше по времени, чем время затрачиваемое на реализацию конвейера (работа с потоками, администрирование очередей по лентам и т.д.). По памяти же конвейерная реализация оказывается более затратной, так как выделяется память под дополнительные структуры данных, используемых для организации работы системы.

Список литературы

[1] Воеводин Вл. В. Параллельная обработка данных [Электронный ресурс]. Режим доступа: https://parallel.ru/vvv/lec1.html (дата обращения: 27.11.2021).

[2] Потоки и работа с ними [Электронный ресурс]. Режим доступа: https://docs.microsoft.com/ru-ru/dotnet/standard/threading/threads-and-threading (дата обращения: 28.10.2021).

[3] Ubuntu 20.04 LTS [Электронный ресурс]. Режим доступа: https://releases.ubuntu.com/20.04/ (дата обращения: 27.11.2021).

[4] Процессор Intel Core i5-1135G7 [Электронный ресурс]. Режим доступа: https://www.intel.ru/content/www/ru/ru/products/sku/208658/intel-core-i51135g7-processor-8m-cache-up-to-4-20-ghz/specifications.html (дата обращения: 27.11.2021).

[5] ISO/IEC 9899:1999 [Электронный ресурс]. Режим доступа: http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1124.pdf (дата обращения: 22.10.2021).