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 № 4
по дисциплине «Анализ алгоритмов»

Тема Параллельные вычисления на основе нативных потоков

Студент Вавилова В. Л.

Группа ИУ7-54Б

Преподаватели Волкова Л. Л., Строганов Ю. В.

Москва, 2024

СОДЕРЖАНИЕ

ВВЕДЕНИЕ **3**

1 Входные и выходные данные **4**

2 Преобразование входных данных в выходные **5**

3 Тестирование **9**

4 Примеры работы программы **10**

5 Описание исследования **12**

ЗАКЛЮЧЕНИЕ **15**

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ **16**

ВВЕДЕНИЕ

Цель работы — проведение исследования организации параллельных вычислений на основе нативных потоков.

Задачи работы:

— исследование предметной области;

— разработка алгоритма параллельной обработки данных;

— реализация ПО (программного обеспечения) на основе нативных потоков;

— исследование зависимости производительности ПО от количества потоков.

Параллелизм может быть 2 видов: конечный и массовый. Конечный означает исполнение параллельно отдельных инструкций/команд/строк кода, массовый — исполнение параллельно итераций цикла. В данном случае реализован конечный параллелизм. [1]

1 Входные и выходные данные

Входные данные:

— адрес главной страницы ресурса;

— максимальное количество страниц, подлежащих обработке;

— режим работы программы (последовательный или параллельный);

— при параллельном режиме — максимальное количество потоков.

Выходные данные:

— директория с файлами, содержащими скачанные страницы в формате HTML;

— структура данных содержащая уникальные ссылки на обработанные страницы.

2 Преобразование входных данных в выходные

В листинге 2.1 представлен метод для последовательной обработки страниц.

```
1  private static void SequentialDischarge(string start_address, int
       max_pages)
2  {
3      while (address_queue.TryDequeue(out string current_address) &&
           visited_addresses.Count < max_pages)
4      {
5          if (!visited_addresses.ContainsKey(current_address))
6              ProcessPage(current_address);
7      }
8  }
```

Листинг 2.1 — Метод SequentialDischarge для последовательной обработки страниц

В листинге 2.2 представлен метод для параллельной обработки страниц.

```
1  private static void ParallelDischarge(string start_address, int
       max_pages, int max_threads)
2  {
3      var threads = new List<Thread>();
4      var activeThreads = max_threads;
5
6      for (int i = 0; i < max_threads; i++)
7      {
8          var thread = new Thread(() =>
9          {
10             while (true)
11             {
12                 if (visited_addresses.Count >= max_pages)
13                 {
14                     break;
15                 }
16
17                 if (address_queue.TryDequeue(out string current_address))
18                 {
19                     if (!visited_addresses.ContainsKey(current_address))
20                     {
21                         visited_addresses.TryAdd(current_address, true);
22                         ProcessPage(current_address);
23                     }
24                 }
```

```
                else
                    Thread.Sleep(10);
            }

            Interlocked.Decrement(ref activeThreads);
        });

        thread.Start();
        threads.Add(thread);
    }

    foreach (var thread in threads)
        thread.Join();
}
```

Листинг 2.2 — Метод ParallelDischarge для параллельной обработки страниц

В листинге 2.3 представлен метод для загрузки страницы.

```
private static async Task<string> FetchPageAsync(string address)
{
    try
    {
        return await httpClient.GetStringAsync(address);
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Error loading the page {address}: {ex.Message}");
        return string.Empty;
    }
}
```

Листинг 2.3 — Метод FetchPageAsync для загрузки страницы

В листинге 2.4 представлен метод для обработки страницы.

```
private static void ProcessPage(string address)
{
    try
    {
        string page_content =
            FetchPageAsync(address).GetAwaiter().GetResult();

        SavePageToFile(address, page_content);

        var links = ExtractLinks(page_content);
```

```
10          foreach (var link in links)
11          {
12              if (!visited_addresses.ContainsKey(link))
13                  address_queue.Enqueue(link);
14          }
15
16          visited_addresses[address] = true;
17      }
18      catch (Exception ex)
19      {
20          Console.WriteLine($"Error address processing {address}: 
                {ex.Message}");
21      }
22  }
```

Листинг 2.4 — Метод ProcessPage для обработки страницы

В листинге 2.5 представлен метод для сохранения страницы в файл.

```
1   private static void SavePageToFile(string address, string content)
2   {
3       lock (lockObj)
4       {
5           string fileName = $"page_{visited_addresses.Count + 1}.html";
6           File.WriteAllText(fileName, content);
7       }
8   }
```

Листинг 2.5 — Метод SavePageToFile для сохранения страницы в файл

В листинге 2.5 представлен метод для извлечения ссылок из содержимого страницы.

```
1   private static IEnumerable<string> ExtractLinks(string page_content)
2   {
3       var links = new List<string>();
4       var matches = Regex.Matches(page_content, @"href=""(/[^""]+)""");
5       foreach (Match match in matches)
6       {
7           if (match.Groups.Count > 1)
8           {
9               string link = match.Groups[1].Value;
10              link = "https://maguro-tuna.ru" + link;
11              if (link.Contains("/recipes/"))
12                  links.Add(link);
13          }
14      }
```

```
15      return links;
16  }
```

Листинг 2.6 — Метод ExtractLinks для извлечения ссылок из содержимого страницы

Для обеспечения монопольного доступа к общей структуре данных используется синхронизация потоков. В частности, использованы `ConcurrentDictionary` и `ConcurrentQueue`.

ConcurrentQueue<T> — потокобезопасная коллекция, реализующая принцип «первый пришёл, первый вышел» (FIFO). Она является заменой для небезопасной в многопоточном окружении коллекции Queue<T>. [2]

ConcurrentDictionary<TKey, TValue> — потокобезопасная версия Dictionary<TKey, TValue>. Она позволяет безопасно работать с данными в многопоточном окружении, поддерживая операции добавления, обновления и получения элементов. [2]

3 Тестирование

Для тестирования программы была проведена серия тестов. Программа была запущена с различным количеством потоков и различным количеством станиц. Тестирование прошло успешно.

4 Примеры работы программы

На рисунке 4.1 представлен сайт данного варианта — `https://maguro-tuna.ru/recipes/`.

На рисунке 4.2 представлена работа ПО при заданных входных параметрах: количество страниц — 10, режим работы — 2 (параллельный), количество потоков — 4.

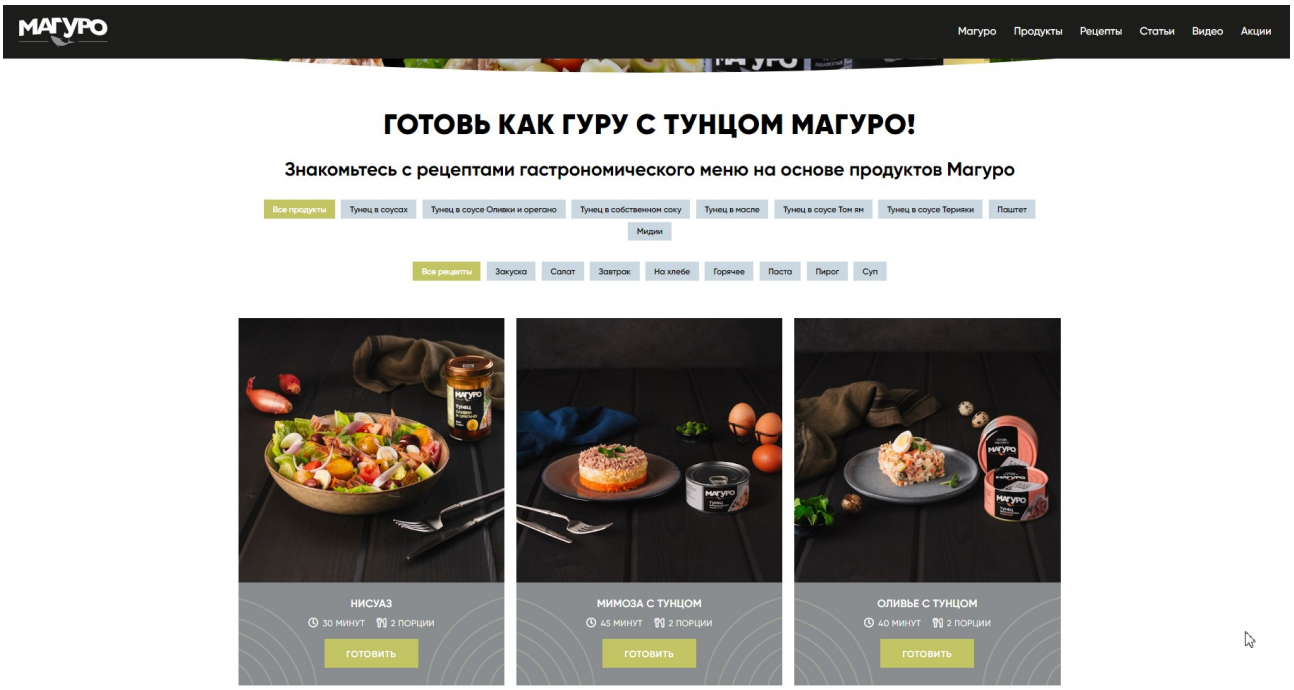

Рисунок 4.1 — Обрабатываемый сайт.

```
Адрес главной страницы: https://maguro-tuna.ru/recipes/

Введите максимальное количество страниц для возможной обработки:
10
Выберите режим:
 1. Последовательный
 2. Параллельный
2
Введите максимальное количество потоков:
4
Thread ID: 13, Address: https://maguro-tuna.ru/recipes/
Thread ID: 13, Address: https://maguro-tuna.ru/recipes/nisuaz/
Thread ID: 16, Address: https://maguro-tuna.ru/recipes/utrenniy-sendvich-s-tuntsom-/
Thread ID: 15, Address: https://maguro-tuna.ru/recipes/olive-s-tuntsom-/
Thread ID: 13, Address: https://maguro-tuna.ru/recipes/smyerrebrod-s-tuntsom-maguro-v-skandinavskom-stile/
Thread ID: 14, Address: https://maguro-tuna.ru/recipes/mimoza-s-tuntsom-/
Thread ID: 16, Address: https://maguro-tuna.ru/recipes/spring-rolly-s-tuntsom-v-souse-tom-yam/
Thread ID: 15, Address: https://maguro-tuna.ru/recipes/avokado-farshirovannyy-tuntsom-olivki-i-oregano-v-italyanskom-stile-/
Thread ID: 13, Address: https://maguro-tuna.ru/recipes/sendvich-na-aysberge-bezglyutenovyy-format-sredizemnomorskoy-kukhni-/
Thread ID: 14, Address: https://maguro-tuna.ru/recipes/lenivye-rolly-filadelfiya-v-aziatskom-stile/
Time: 0,4893147

Process finished with exit code 0.
```

Рисунок 4.2 — Пример работы программы.

5 Описание исследования

Целью исследования является выявление оптимального числа потоков для максимальной производительности на многопроцессорной системе с четырьмя ядрами.

Замеры работы программы на разном количестве потоков и разном количестве страниц представлены в таблице 5.1.

Таблица 5.1 — Среднее время выполнения и процентное ускорение работы программы

Кол-во потоков	Кол-во страниц	Среднее время выполнения (с)	Ускорение (%)
1 поток	1	0,10	100
	11	0,66	100
	21	1,27	100
	31	2,20	100
	41	2,84	100
	51	3,06	100
2 потока	1	0,06	167
	11	1,41	47
	21	1,29	98
	31	0,97	227
	41	1,31	217
	51	1,54	199
4 потока	1	0,05	200
	11	1,16	57
	21	1,23	103
	31	1,14	193
	41	1,24	229
	51	1,09	281
8 потоков	1	0,05	200
	11	1,18	56
	21	1,17	109
	31	1,17	188

Кол-во потоков	Кол-во страниц	Среднее время выполнения (с)	Ускорение (%)
	41	1,19	239
	51	1,46	210
16 потоков	1	0,06	167
	11	1,35	49
	21	1,26	101
	31	1,25	176
	41	1,40	203
	51	1,24	247

Результаты вычисления среднего ускорения работы программы при разном количестве потоков представлены в таблице 5.2.

Таблица 5.2 — Среднее ускорение работы программы в зависимости от количества потоков

Кол-во потоков	Среднее ускорение (%)
1 поток	100.00
2 потока	159.17
4 потока	177.17
8 потоков	167.00
16 потоков	157.17

Общая формула

Ускорение вычисляется по формуле:

$$\text{Ускорение } (\%) = \left(\frac{T_{\text{1 поток}}}{T_{\text{n потоков}}} \right) \times 100 \tag{5.1}$$

где:

— $T_{\text{1 поток}}$ — среднее время выполнения в последовательном режиме (1 поток),

— $T_{\text{n потоков}}$ — среднее время выполнения в параллельном режиме (n потоков).

Пример расчёта

Для 2 потоков и 1 страницы:

$$\text{Ускорение (\%)} = \frac{0.10}{0.06} \times 100 = 166.67\%. \tag{5.2}$$

По результатам исследования построены графики времени выполнения программы и ускорения работы в зависимости от количества потоков 5.1.

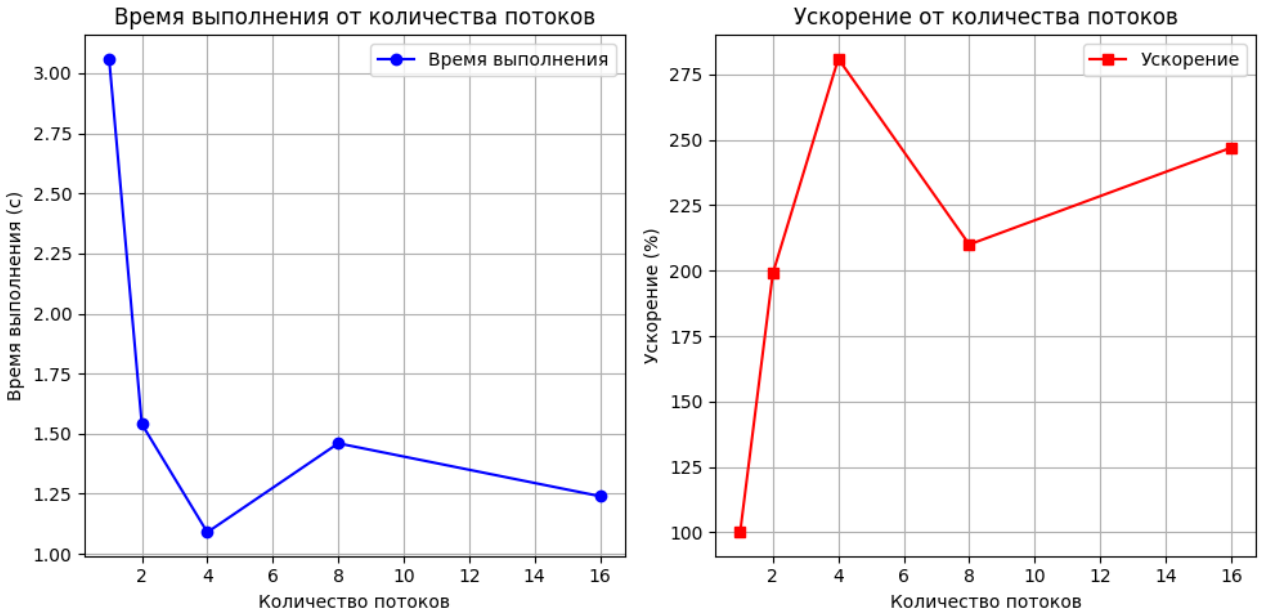

Рисунок 5.1 — Графики времени выполнения программы и ускорения работы в зависимости от количества потоков.

Технические характеристики устройства:

— процессор: Intel(R) Core(TM) i5-10300H с тактовой частотой 2.5 Ггц;

— ядра: 4;

— логических процессоров: 8.

Вывод: при количестве потоков, равных количеству ядер процессора (4 ядра) производительность наибольшая.

ЗАКЛЮЧЕНИЕ

Цель работы достигнута. Разработано программное обеспечение, которое позволяет осуществлять параллельную выгрузку данных с интернет-ресурсов с использованием нативных потоков. Проведено исследование зависимости производительности программы от количества потоков и количества страниц. Полученные результаты показывают, что использование потоков значительно ускоряет выполнение задачи до предела в 4 потока (количество ядер).

Выполнены следующие задачи:

— исследована предметная область;
— разработан и реализован алгоритм параллельной обработки данных;
— реализовано ПО на основе нативных потоков;
— проведены тесты производительности.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Волкова Л. Л. Конспекты лекций и семинаров курса «Анализ алгоритмов». 2024.

2. Alvin Ashcraft. *Parallel Programming and Concurrency with C# 10 And .NET 6*. — 2022. — С. 45–47. (дата обращения 18.11.2024)