Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и технологий
Кафедра компьютерных систем и программных технологий

Отчет по лабораторной работе

Дисциплина: **Параллельные вычисления**

Тема: **Создание многопоточных программ на языке Java с использованием Java Threads и MPI**

Выполнил студент гр. 13541/3 _____ Попсуйко М.Ю.
(подпись)

Преподаватель _____ Стручков И.В.
(подпись)

Санкт-Петербург
2018

Содержание

1 Постановка задачи

1.1 Задание

Вычислить пересечение трех множеств.

1.2 Программа работы

1. Для алгоритма из полученного задания написать последовательную программу на языке C или C++, реализующую этот алгоритм;

2. Для созданной последовательной программы необходимо написать 3-5 тестов, которые покрывают основные варианты функционирования программы. Для создания тестов можно воспользоваться механизмом Unit-тестов среды NetBeans, или описать входные тестовые данные в файлах. При использовании NetBeans необходимо в свойствах проекта установить ключ компилятора -pthread;

3. Проанализировать полученный алгоритм, выделить части, которые могут быть распараллелены, разработать структуру параллельной программы. Определить количество используемых потоков, а также правила и используемые объекты синхронизации;

4. Согласовать разработанную структуру и детали реализации параллельной программы с преподавателем;

5. Написать код параллельной программы и проверить ее корректность на созданном ранее наборе тестов. При необходимости найти и исправить ошибки;

6. Провести эксперименты для оценки времени выполнения последовательной и параллельной программ. Проанализировать полученные результаты;

7. Сделать общие выводы по результатам проделанной работы:

    - Различия между способами проектирования последовательной и параллельной реализаций алгоритма;
    - Возможные способы выделения параллельно выполняющихся частей;
    - Возможные правила синхронизации потоков;
    - Сравнение времени выполнения последовательной и параллельной программ;
    - Принципиальные ограничения повышения эффективности параллельной реализации по сравнению с последовательной.

2 Сведения о системе

Таблица 2.1: Сведения о системе

Процессор	Intel(R) Core(TM) i7-3630QM CPU @ 2.40GHz, 2401 МГц
Ядер	4
Логических потоков	8
Установленная память (ОЗУ)	8.00 ГБ

3 Структура проекта

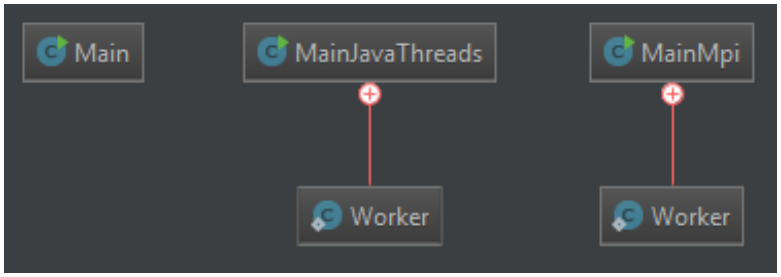

Рисунок 3.1. Структура проекта

Main - класс, решающий задачу вычисления пересечения трех множеств в одном потоке. В данном классе используются следующие методы:

- **generateSet** - метод для генерации множеств;
- **intersection** - метод для поиска пересечений трех множеств.

MainJavaThreads - класс, решающий задачу вычисления пересечения трех множеств в нескольких потоках. Для этого используются вспомогательный класс Worker, который наследуется от класса Threads.

В данном классе используются следующие методы:

- **generateSet** - метод для генерации множеств;
- **createWorkers** - метод создание списка потоков (workers);
- **startWorkers** - запуск всех потоков;
- **joinWorkers** - ожидание завершения всех потоков;
- **mergeResults** - получение результатов из каждого потока;

- **intersection** - поиск пересечений для определенного интервала подмножества множества A.

MainMPI - класс, решающий задачи вычисления пересечения трех множеств в нескольких процессах с использованием MPI. Вспомогательный класс Worker используется для упрощения коммуникаций между процессами.

В данном классе используются следующие методы:

- **generateSet** - метод для генерации множеств;
- **intersection** - поиск пересечений для определенного интервала подмножества множества A.
- **createWorkers** - метод для создание списка объектов класса Worker и передачи в каждый объект необходимых параметров: множества A, B, C, индексы начала и конца проверяемого подмножества, интерфейс коммуникации между процессами, индекс процесса, в котором будет происходить поиск пересечений;
- **startWorkers** - метод, который инициирует для каждого Worker'а отправку данных в определенный процесс для поиска пересечения;
- **receiveResults** - метод, который инициирует для каждого Worker'а получение результата из определенного процесса и сливает их вместе.

4 Алгоритм решения

4.1 В одном потоке

Алгоритм работы следующий: проходим по элементам первого множества и проверяем содержится ли очередной элемент первого множества во множествах B и C. Если да, то добавляем его.

4.2 С использованием Java Threads

1. Алгоритм работы начинается с создания заданного количества потоков;
2. Затем первое множество делится на количество получившихся потоков. Таким образом мы визуально разделяем первое множество на равное количеству потокам множества и запоминаем границы этих подмножеств;
3. Далее каждому потоку передаются три множества и границы получившихся подмножеств;
4. Запускаем все потоки;

5. Каждый поток ищет пересечение для своей части первого множества;
6. Ждем завершения потоков;
7. Получаем результаты из каждого потока и сливаем все вместе.

4.3 С использованием MPI

1. В нулевом (родительском) процессе происходит генерация множеств;
2. Затем первое множество делится на заданное количество вспомогательных процессов;
3. Далее каждому процессу передаются три множества, границы получившихся подмножеств, интерфейс для передачи сообщений и индекс соответствующего вспомогательного процесса;
4. Отправляем сообщения во вспомогательные процессы;
5. Каждый вспомогательный процесс получает сообщения, ищет пересечение своей части первого множества и отправляет результат в нулевой (родительский) процесс;
6. В родительском процессе получаем результаты из каждого вспомогательного процесса и сливаем все вместе.

5 Тестирование

Среднее время выполнения было посчитано исходя из 50 запусков для каждого варианта. Количество элементов в массиве 100000.

Таблица 5.1: Среднее время выполнения для разных вариантов запуска

Количество потоков/процессов	Java Threads, мс	MPI, мс
1	3855,04	3897,30
2	2503,92	2209,72
4	1502,02	1634,64
8	1147,20	3057,22

Таблица 5.2: Дисперсия для разных вариантов запуска

Количество потоков/процессов	Java Threads	MPI
1	271,38	32,83
2	243,04	43,35
4	342,30	89,10
8	120,57	126,06

Далее представлены графики зависимости времени выполнения от количества потоков для Java Threads и MPI.

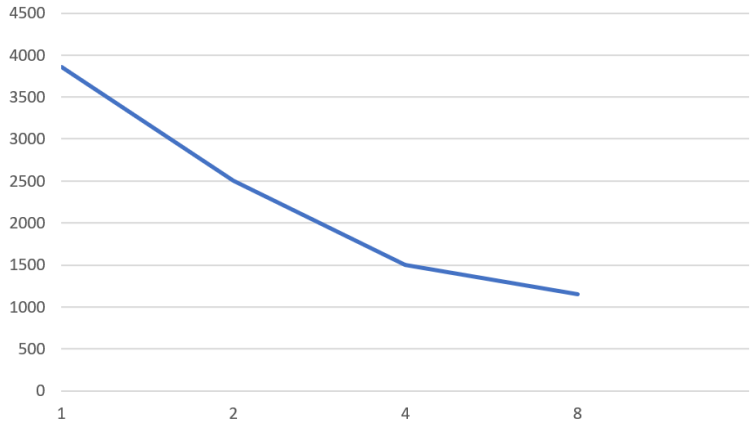

Рисунок 5.1. Математическое ожидание для Java Threads

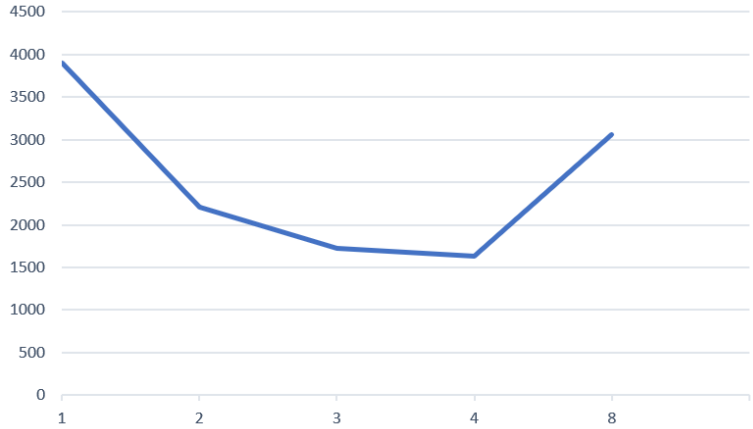

Рисунок 5.2. Математическое ожидание для MPI

6 Выводы

Были рассмотрены способы написания параллельных приложений при помощи Java Threads и MPI на примере задачи поиска пересечения трех множеств.

Выяснено, что при увеличении числа потоков/процессов время выполнения программы удается значительно снизить. Однако, при дальнейшем увеличении количества потоков или процессов время выполнения увеличивается. Это происходит из-за многих факторов, например, время уходит на создание процессов, на выделение памяти на создание массивов. Стоит отметить, что в компьютере установлен 4-х ядерных процессор, с 8-ью логическими потоками. При запуске программы с восемью вспомогательными процессами, еще один процесс выделяется, как родительный, что в сумме дает 9 процессов. Это также может стать причиной увеличения времени выполнения.

В результате дисперсия при использовании Java Threads оказалась выше, чем при использовании MPI.