Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«Пермский национальный** **исследовательский политехнический университет»**

Факультет: Прикладной математики и механики

Кафедра: Вычислительной математики, механики и биомеханики

Направление: 09.04.02 Информационные технологии и системная инженерия

Профиль: «Информационные технологии и системная инженерия»

**Лабораторные работы**

**по дисциплине: «Параллельное программирование»**

**Выполнил**

студент гр. ИТСИ-24-1м

**Юнусов Василь Равилевич**

**Принял** Преподаватель кафедры ВММБ

**Истомин Денис Андреевич**

Пермь 2025

Лабораторная работа №1

В ходе работы были рассмотрены два программы, выполняющие поэлементное перемножение двух массивов. Первая программа решает эту задачу последовательно, вторая - при помощи sse.

Результаты выполнения программ

|  |
| --- |
| $ ./seq.sh 1000000000  real 0m6.923s  user 0m6.927s  sys 0m0.000s  ```  ```  $ ./sse.sh 1000000000  real 0m1.561s  user 0m1.567s  sys 0m0.000s |

Разница во времени выполнения: 6.923 / 1.561 = 4.434977578475336

Эта разница достигается за счёт того, что в случае с sse.c используются SSE инструкции, которые позволяют выполнять 1 вычислительную операцию по отношению к нескольким значениям за счет того, что они загружены в один регистр.

Лабораторная работа №2

В ходе работы были рассмотрены два программы, выполняющие вычисление квадратного корня различных чисел. Первая программа решает эту задачу последовательно, вторая - параллельно при помощи pthreads.

Результаты выполнения программ на 8 потоках

|  |
| --- |
| $ ./seq.sh 8  MAIN: starting thread 0  Thread #0 started  Thread #0 acquired mutex  Thread #0, counter: 1  Thread #0 released mutex  Thread #0 finished  MAIN: starting thread 1  Thread #1 started  Thread #1 acquired mutex  Thread #1, counter: 2  Thread #1 released mutex  Thread #1 finished  MAIN: starting thread 2  Thread #2 started  Thread #2 acquired mutex  Thread #2, counter: 3  Thread #2 released mutex  Thread #2 finished  MAIN: starting thread 3  Thread #3 started  Thread #3 acquired mutex  Thread #3, counter: 4  Thread #3 released mutex  Thread #3 finished  MAIN: starting thread 4  Thread #4 started  Thread #4 acquired mutex  Thread #4, counter: 5  Thread #4 released mutex  Thread #4 finished  MAIN: starting thread 5  Thread #5 started  Thread #5 acquired mutex  Thread #5, counter: 6  Thread #5 released mutex  Thread #5 finished  MAIN: starting thread 6  Thread #6 started  Thread #6 acquired mutex  Thread #6, counter: 7  Thread #6 released mutex  Thread #6 finished  MAIN: starting thread 7  Thread #7 started  Thread #7 acquired mutex  Thread #7, counter: 8  Thread #7 released mutex  Thread #7 finished  real 0m1.328s  user 0m1.297s  sys 0m0.000s |

|  |
| --- |
| $ ./pt.sh 8  MAIN: starting thread 0  MAIN: starting thread 1  Thread #0 started  Thread #0 acquired mutex  Thread #0, counter: 1  Thread #0 released mutex  Thread #1 started  Thread #1 acquired mutex  Thread #1, counter: 2  Thread #1 released mutex  MAIN: starting thread 2  MAIN: starting thread 3  Thread #2 started  Thread #2 acquired mutex  Thread #3 started  Thread #2, counter: 3  Thread #2 released mutex  MAIN: starting thread 4  Thread #3 acquired mutex  Thread #3, counter: 4  Thread #3 released mutex  MAIN: starting thread 5  Thread #4 started  Thread #4 acquired mutex  Thread #4, counter: 5  Thread #4 released mutex  Thread #5 started  MAIN: starting thread 6  MAIN: starting thread 7  Thread #6 started  Thread #7 started  Thread #5 acquired mutex  Thread #5, counter: 6  Thread #5 released mutex  Thread #6 acquired mutex  Thread #6, counter: 7  Thread #6 released mutex  Thread #7 acquired mutex  Thread #7, counter: 8  Thread #7 released mutex  Thread #2 finished  Thread #4 finished  Thread #1 finished  Thread #5 finished  Thread #0 finished  Thread #3 finished  Thread #7 finished  Thread #6 finished  real 0m0.297s  user 0m2.271s  sys 0m0.010s |

Разница во времени выполнения: 1.328 / 0.297 = 4.471380471380471

Как уже сказано выше, во втором случае задачи выполняются параллельно и не дожидаются выполнения предыдущих. За счет этого и была получена такая разница во времени выполнения.

Лабораторная работа №3

В ходе работы была рассмотрена программа, выполняющая вычисление квадратного корня различных чисел. Программа решает эту задачу параллельно при помощи OpenMP.

Результаты выполнения программ на 8 потоках

|  |
| --- |
| OpenMP threads: 1  OpenMP threads: 8  OpenMP threads: 8  OpenMP threads: 8  OpenMP threads: 8  OpenMP threads: 8  OpenMP threads: 8  OpenMP threads: 8  OpenMP threads: 8  real 0m0.307s  user 0m2.399s  sys 0m0.000s |

В данном случае была получена сравнительная с pthreads (0.307s~0.297s) производительность, тк по сути ничего не поменялось, кроме того, что распараллеливанием кода занимается сам компилятор. Задача всё так же выполняется параллельно в 8 потоках

Лабораторная работа №4

В ходе работы была рассмотрена программа, которая в многопоточном режиме работает с различными реализациями Map - HashMap, Hashtable, SynchronizedMap, ConcurrentHashMap.

Результаты выполнения программ на 8 потоках

|  |
| --- |
| Collections:  java.util.HashMap...done.  java.util.Hashtable...done.  java.util.Collections$SynchronizedMap...done.  java.util.concurrent.ConcurrentHashMap...done.  Execution times:  HashMap: 0.010 s,  HashTable: 0.010 s,  SyncMap: 0.010 s,  ConcurrentHashMap: 0.015 s.  hashMap: {0=40860, 1=35981, 2=50000, 3=49994, 4=49260}  hashTable: {4=50000, 3=50000, 2=50000, 1=50000, 0=50000}  syncMap: {0=50000, 1=50000, 2=50000, 3=50000, 4=50000}  cHashMap: {0=50000, 1=50000, 2=50000, 3=50000, 4=50000} |

Каждый доступ к Map - инкремент хранящегося значения на 1. Он реализовывался 50 потоками по 1000 итераций => конечным значением должно быть 50000. Как можно заметить, в первом случае (HashMap) мы получили не тот результат, который ожидался. Результат остальных реализаций соответствует ожиданиям. Это связано с тем, что первая реализация не потокобезопасна. Остальные же реализации обеспечивают потокобезопасность, но реализуют её по-разному.

Лабораторная работа №5

В ходе работы была предоставлена собтвенная реализация считающего семафора.

Результаты выполнения программ на 8 потоках и семафором с максимум 2 входами

|  |
| --- |
| -------------------  Regular semaphore:  -------------------  Поток pool-1-thread-2 пытается захватить семафор  Поток pool-1-thread-8 пытается захватить семафор  Поток pool-1-thread-6 пытается захватить семафор  Поток pool-1-thread-3 пытается захватить семафор  Поток pool-1-thread-4 пытается захватить семафор  Поток pool-1-thread-5 пытается захватить семафор  Поток pool-1-thread-7 пытается захватить семафор  Поток pool-1-thread-1 пытается захватить семафор  Поток pool-1-thread-8 захватил семафор, доступно входов: 0  Поток pool-1-thread-2 захватил семафор, доступно входов: 1  Поток pool-1-thread-2 выполняет работу в критической секции  Поток pool-1-thread-8 выполняет работу в критической секции  Поток pool-1-thread-2 пытается отпустить семафор  Поток pool-1-thread-8 пытается отпустить семафор  Поток pool-1-thread-8 отпустил семафор, доступно входов: 2  Поток pool-1-thread-2 отпустил семафор, доступно входов: 1  Поток pool-1-thread-5 захватил семафор, доступно входов: 0  Поток pool-1-thread-5 выполняет работу в критической секции  Поток pool-1-thread-6 захватил семафор, доступно входов: 1  Поток pool-1-thread-6 выполняет работу в критической секции  Поток pool-1-thread-6 пытается отпустить семафор  Поток pool-1-thread-5 пытается отпустить семафор  Поток pool-1-thread-1 захватил семафор, доступно входов: 0  Поток pool-1-thread-5 отпустил семафор, доступно входов: 1  Поток pool-1-thread-6 отпустил семафор, доступно входов: 1  Поток pool-1-thread-1 выполняет работу в критической секции  Поток pool-1-thread-3 захватил семафор, доступно входов: 0  Поток pool-1-thread-3 выполняет работу в критической секции  Поток pool-1-thread-1 пытается отпустить семафор  Поток pool-1-thread-3 пытается отпустить семафор  Поток pool-1-thread-1 отпустил семафор, доступно входов: 1  Поток pool-1-thread-4 захватил семафор, доступно входов: 0  Поток pool-1-thread-3 отпустил семафор, доступно входов: 1  Поток pool-1-thread-7 захватил семафор, доступно входов: 0  Поток pool-1-thread-4 выполняет работу в критической секции  Поток pool-1-thread-7 выполняет работу в критической секции  Поток pool-1-thread-7 пытается отпустить семафор  Поток pool-1-thread-4 пытается отпустить семафор  Поток pool-1-thread-7 отпустил семафор, доступно входов: 1  Поток pool-1-thread-4 отпустил семафор, доступно входов: 2  --------------  My semaphore:  --------------  [MySemaphore] Поток pool-2-thread-1 пытается захватить семафор, доступно входов: 2  [MySemaphore] Поток pool-2-thread-1 захватил семафор, доступно входов: 1  [MySemaphore] Поток pool-2-thread-2 пытается захватить семафор, доступно входов: 1  Поток pool-2-thread-1 выполняет работу в критической секции  [MySemaphore] Поток pool-2-thread-2 захватил семафор, доступно входов: 0  Поток pool-2-thread-2 выполняет работу в критической секции  [MySemaphore] Поток pool-2-thread-4 пытается захватить семафор, доступно входов: 0  [MySemaphore] Поток pool-2-thread-4 ожидает освобождения семафора, доступно входов: 0  [MySemaphore] Поток pool-2-thread-6 пытается захватить семафор, доступно входов: 0  [MySemaphore] Поток pool-2-thread-6 ожидает освобождения семафора, доступно входов: 0  [MySemaphore] Поток pool-2-thread-3 пытается захватить семафор, доступно входов: 0  [MySemaphore] Поток pool-2-thread-3 ожидает освобождения семафора, доступно входов: 0  [MySemaphore] Поток pool-2-thread-5 пытается захватить семафор, доступно входов: 0  [MySemaphore] Поток pool-2-thread-5 ожидает освобождения семафора, доступно входов: 0  [MySemaphore] Поток pool-2-thread-7 пытается захватить семафор, доступно входов: 0  [MySemaphore] Поток pool-2-thread-7 ожидает освобождения семафора, доступно входов: 0  [MySemaphore] Поток pool-2-thread-8 пытается захватить семафор, доступно входов: 0  [MySemaphore] Поток pool-2-thread-8 ожидает освобождения семафора, доступно входов: 0  [MySemaphore] Поток pool-2-thread-1 отпустил семафор, доступно входов: 1  [MySemaphore] Поток pool-2-thread-2 отпустил семафор, доступно входов: 2  [MySemaphore] Поток pool-2-thread-4 захватил семафор, доступно входов: 1  [MySemaphore] Поток pool-2-thread-6 захватил семафор, доступно входов: 0  Поток pool-2-thread-4 выполняет работу в критической секции  Поток pool-2-thread-6 выполняет работу в критической секции  [MySemaphore] Поток pool-2-thread-4 отпустил семафор, доступно входов: 1  [MySemaphore] Поток pool-2-thread-3 захватил семафор, доступно входов: 0  Поток pool-2-thread-3 выполняет работу в критической секции  [MySemaphore] Поток pool-2-thread-6 отпустил семафор, доступно входов: 1  [MySemaphore] Поток pool-2-thread-5 захватил семафор, доступно входов: 0  Поток pool-2-thread-5 выполняет работу в критической секции  [MySemaphore] Поток pool-2-thread-3 отпустил семафор, доступно входов: 1  [MySemaphore] Поток pool-2-thread-7 захватил семафор, доступно входов: 0  Поток pool-2-thread-7 выполняет работу в критической секции  [MySemaphore] Поток pool-2-thread-5 отпустил семафор, доступно входов: 1  [MySemaphore] Поток pool-2-thread-8 захватил семафор, доступно входов: 0  Поток pool-2-thread-8 выполняет работу в критической секции  [MySemaphore] Поток pool-2-thread-7 отпустил семафор, доступно входов: 1  [MySemaphore] Поток pool-2-thread-8 отпустил семафор, доступно входов: 2  Process finished with exit code 0 |

Как можно заметить, количество доступных входов для семафора всегда было в пределах [0, 2], следовательно, собственная реализация семафора работает корректно.

Лабораторная работа №6

В ходе работы была рассмотрено клиент-серверное приложение примитивного чата через socket.

Результат работы сервера

|  |
| --- |
| $ java Server.java  Try to run server on 8080 port  Server is running and waiting for connections...  New client connected: Socket[addr=/127.0.0.1,port=36642,localport=8080]  User User 1 connected.  New client connected: Socket[addr=/127.0.0.1,port=56068,localport=8080]  User User 2 connected.  Hello from server!  [Server]: Hello from server!  [User 1]: Hello from user 1  [User 2]: Hello from user 2 |

|  |
| --- |
| $ java Client.java  Connected to the chat server!  Enter your username:  User 1  Welcome to the chat, User 1!  Type Your Message  [Server]: Hello from server!  Hello from user 1  [User 1]: Hello from user 1  [User 2]: Hello from user 2 |

|  |
| --- |
| $ java Client.java  Connected to the chat server!  Enter your username:  User 2  Welcome to the chat, User 2!  Type Your Message  [Server]: Hello from server!  [User 1]: Hello from user 1  Hello from user 2  [User 2]: Hello from user 2 |

Как можно заметить, приложение работает корректно, передавая сообщения всем участникам чата.

Лабораторная работа №7

В ходе работы была изучена библиотека mappedbus, а также были запущены тестовые примеры

Были запущены два ObjectWriter и один ObjectReader. ObjectWriter записывали сообщения в MemoryMappedFile, а ObjectReader считывал их и выводил в консоль.

Результат работы ObjectWriter 1

|  |
| --- |
| $ java -cp mappedbus\*.jar io.mappedbus.sample.object.ObjectWriter 0 |

Результат работы ObjectWriter 2

|  |
| --- |
| $ java -cp mappedbus\*.jar io.mappedbus.sample.object.ObjectWriter 0 |

Результат работы ObjectReader

|  |
| --- |
| $ java -cp mappedbus\*.jar io.mappedbus.sample.object.ObjectReader  Read: PriceUpdate [source=1, price=0, quantity=0], hasRecovered=false  Read: PriceUpdate [source=1, price=2, quantity=4], hasRecovered=false  Read: PriceUpdate [source=1, price=4, quantity=8], hasRecovered=false  Read: PriceUpdate [source=1, price=6, quantity=12], hasRecovered=false  Read: PriceUpdate [source=1, price=8, quantity=16], hasRecovered=false  Read: PriceUpdate [source=1, price=38, quantity=76], hasRecovered=false  Read: PriceUpdate [source=1, price=40, quantity=80], hasRecovered=false  Read: PriceUpdate [source=0, price=0, quantity=0], hasRecovered=false  Read: PriceUpdate [source=0, price=2, quantity=4], hasRecovered=false  Read: PriceUpdate [source=0, price=4, quantity=8], hasRecovered=false  Read: PriceUpdate [source=0, price=6, quantity=12], hasRecovered=false  Read: PriceUpdate [source=0, price=8, quantity=16], hasRecovered=false  Read: PriceUpdate [source=0, price=10, quantity=20], hasRecovered=false |

Как можно заметить, ObjectReader считывает сообщения из двух разных источников.