|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **주차** | 2 주차 | **기간** | 2025.09.08  ~ 2025.09.14 | **지도교수** | (서명) |
| 이번주 한일 요약 | 1. 멀티코어 프로그래밍 학습  2. 셰이더 프로그래밍 학습  3. 서버 외부 강의 수강  4. 데이터베이스 학습  5. 매주 팀원 정기 회의 | | | | |

<상세 수행내용>

1. 서버 외부 강의를 통해 Client – Server 연동 실습

서버와 클라이언트간 상호 연결 및 통신 실습

클라이언트가 보낸 내용을 서버에게 띄워 확인하는 작업 수행

2. 셰이더 프로그래밍 학습

Vectex, Fragment Shader 작업 수행

-> 삼각형을 그린 후 정점의 데이터를 Shader에게 처리할 수 있게 프로그래밍

-> Color을 입혀 삼각형 각 정점에 RGBA값을 넣은 후 Fragment Shader에서 처리

Vectex의 데이터가 바뀌지 않더라도 형태를 바꾸거나 특정 부분만 그리는 방식을 학습함

-> Fragment Shader에서 조건문을 추가하여 조건에 부합한 부분만 그리고 나머지는 dispatch로 지우는 방식을 함, 정점의 데이터가 바뀌지 않더라도 삼각형에서 다른 원하는 부분만 그릴 수 있었음

3. 멀티코어 프로그래밍 학습

Data Race를 없애기 위한 대책으로 Lock과 Unlock을 사용

-> mutex 객체를 전역 변수로 선언해 사용

-> Lock과 Lock을 사용하지 않은 프로그램 속도 비교, 쓰레드의 개수를 늘려가면서 측정하였을 때, 동일한 결과 값을 보장할 수 있었지만 Lock을 안 썼을 때보다 20배가 넘게 느린 성능을 보여주었음

-> 결과적으로 병렬성이 감소했고 그로 인한 성능 향상도 감소됨, lock 자체에도 오버헤드가 발생하기도 함

Lock을 쓰지 않기 위해 std::atomic과 volatile을 사용

-> atomic연산은 컴파일러가 실행 순서를 절대 바꾸지 않고 프로그래밍 순서를 유지함

-> atomic은 뜻 그대로 원자이기에 atomic으로 선언한 변수인 sum의 sum +=2 와 sum = sum + 2는 전혀 다른 뜻임

-> 결과적으로 lock을 썼을 때마다 성능 향상이 일어났지만 단일 쓰레드에 비해서 느림

하지만 lock과 atomic연산 개수를 줄이면서 Data Race가 적도록 할 수 있으나 완전히 없앨 순 없음

-> Cache Trashing 해결을 위해 alignas을 선언하여 사용, 쓰레드마다 서로 다른 cache line 사용하게 조치

4. 데이터베이스 학습

데이터베이스 시스템의 구성 요소, 메타 데이터(시스템 카탈로그)를 활용하여 형식을 확인 후 데이터 베이스에 접근하는 방식 학습

파일 시스템의 장점과 단점, 데이터의 형식이 바뀌어도 그에 대한 의존도를 낮추는 방법에 대해 학습함

-> 메타 데이터를 생성하고 이 곳에 형식을 저장 후, 찾아가 해당 위치 데이터베이스에 데이터를 저장하는 방식, 그리고 찾고자 하는 형식을 찾은 후 데이터에 접근하는 것임을 인지함, 유지보수에 용이함

5. 매주 정기 팀원 회의

매주 토요일 오후 9시, 정기적으로 팀원과 만나 회의 진행

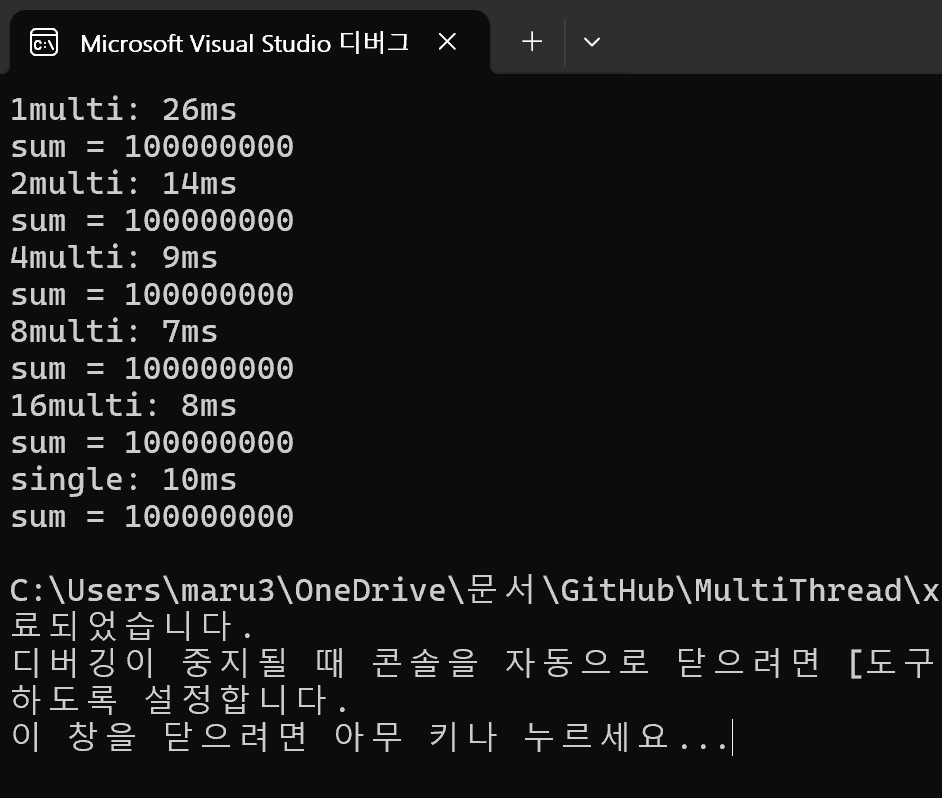
각자가 이번주에 공부한 내용 및 기획서 공유

Lock을 사용하지 않고 Data Race를 최소화하기 위해 alignas와 volatile을 사용해 구조체 형태로 선언하고 배열 형태로 선언하여 각 쓰레드가 자기 번호의 영역만을 접근하여 데이터를 쓰는 방식으로 방법을 바꿔보았다.

그러나 여전히 싱글 쓰레드에 비해 일부는 느린 경향을 보였다. 그리고 lock을 사용하지 않았기에 컴파일러가 volatile을 사용했음에도 불구하고 나머지 코드는 최적화하여 순서를 바꾸거나 아예 안 돌릴 가능성도 있었다.

이를 해결하기 위해 상호 배제를 유지하고 lock과 volatile을 사용한 상태로 Cache Thrashing이 이루어지지 않게 조치를 해야 한다.

이에 대한 조치로 Bakery algorithm이라는 알고리즘이 있었다. 이를 학습하여 적용하면 lock을 사용하면서 상호 배제 상태를 유지할 수 있을 것 같다.



|  |  |  |  |
| --- | --- | --- | --- |
| **문제점 정리** | 1. alignas volatile 변수를 선언하고 서로 다른 cache line을 사용했음에도 온전한 성능 개선이 이루어지지 않음 2. 일부는 싱글 스레드보다 느린 현상 발생 | | |
| **해결방안** | 1. Data Race을 최소화하고 Cache Trashing이 없는 상태로 Lock을 사용하기 위해 Bakery algorithm 구현 | | |
| **다음주차** | 3주차 | **다음기간** | 2025.09.15 ~ 2025.09.21 |
| **다음주 할일** | 1. 멀티코어 프로그래밍 학습  2. 셰이더 프로그래밍 학습  3. 에코 서버 테스트  4. 매주 팀원 정기 회의 | | |
| **지도 교수**  **Comment** | 무엇이 부족한지 생각하고 그에 맞게 계획 수렴 | | |