고급소프트웨어실습

기말 리포트

(CSE 4152)

Due: 2021년, 12/15일, 23:59분

학번 : 20172141

반 번호 : 2반

이름 : 김미소

**1. React와 같이 웹환경에서 사용하는 개발 툴로 Spring framework가 있으며, React는 front end를 Spring framework는 back end를 개발하는 데 주로 사용된다. React와 Spring framework를 대체할 수 있는 도구로 최근 관심을 받고 있는 것들로는 A와 B가 있다. 적절한 A와 B를 찾고 React와 A, 그리고 Spring framework와 B의 장단점을 아래의 테이블에 정리하시오.**

|  |  |  |
| --- | --- | --- |
|  | 장점 | 단점 |
| React | 1. React web application은 여러 개의 컴포넌트로 구성되어 있는데, 이 컴포넌트들은 재사용이 가능하므로 개발이나 유지보수하기 편하다.  2. HTML DOM을 대체하여 virtual DOM을 사용하는데, virtual DOM은 전체 DOM을 리로딩하는 것 대신에 각 컴포넌트의 DOM을 변경하여 사용하므로 더 빠르다.  3. 서버 사이드 렌더링과 클라이언트 사이드 렌더링을 지원하여 초기 렌더링 딜레이를 줄일 수 있다.  4. 라이브러리이기 때문에 다른 프레임워크와 혼용하여 사용할 수 있다. | 1. IE8 이하의 버전은 지원하지 않는다.  2. View 이외의 기능은 다른 라이브러리를 이용하거나 직접 구현하여야 한다.  3. 리액트가 개발/개선되는 속도가 빠르기 때문에 개발자들이 계속해서 새로운 부분에 대하여 update하기 버거우며 document가 update되는 속도보다 리액트 개발 속도가 빨라 적절한 document 제공이 어렵다. |
| Angular | 1. MVC 구조를 사용하기 때문에 앱 로직을 UI와 분리할 수 있다.  2. 단방향 또는 양방향의 선택적 데이터 바인딩을 지원한다.  3. 컴포넌트 기반으로 개발하므로 기능에 따라 코드를 분리하고, 재사용할 수 있다.  4. 화면을 생성한 이후에 필요한 데이터만 Ajax 요청을 통해 서버에서 내려 받기 때문에 페이지 전환 속도가 빠르다.  5. 프론트엔드 웹 개발을 위한 종합 프레임워크이므로 많은 모듈을 가지고 있다. | 1. 초기 로딩속도가 느리다.  2. 검색엔진의 접근성이 떨어진다.  3. 복잡한 시스템을 위해 디자인된 것이기 때문에 작은 app을 개발하기에는 부적절할 수 있다. |

|  |  |  |
| --- | --- | --- |
|  | 장점 | 단점 |
| Spring framework | 1. POJO(Plain Old Java Object)를 지원하므로 코드가 간결하고 테스트하기 용이하다.  2. Dependency Injection을 지원하여 재사용성을 높이고 코드 수정이 용이하다.  3. IoC(Inversion of Control)가 적용되어 낮은 결합도를 유지함으로써 유지 보수가 편리하다.  3. 빠르게 시작/종료할 수 있고 최적화하여 실행하기 때문에 성능이 좋다.  4. AOP(Aspect Oriented Programming)이 지원되어 비즈니스 메소드마다 반복되는 공통 로직을 분리할 수 있다. | 1. 복잡하고 많은 전문지식을 필요로 하기 때문에 배우기 어렵다.  2. Spring은 많은 옵션을 개발자에게 제공하는데, 어떤 옵션을 사용하는 것이 좋을지 결정하기 다소 혼란스러우며 잘못된 결정을 할 경우에 상당한 딜레이를 발생시킬 수 있다.  3. 많은 기능을 가지고 있기 때문에 무겁다. |
| django | 1. 독립적인 app 단위로 구성되어 있어 로직을 분리하여 사용할 수있다. 즉 유지보수가 쉽고 재사용하기 좋다.  2. 컴퓨팅 기능이 뛰어나 머신러닝 라이브러리와 호환된다.  3. 파이썬으로 작성되어 있으므로 다양한 운영체제에서 작동할 수 있다. 즉 portable하다.  4. 발생하는 늘어난 트래픽에 대응해 크기를 변경할 수 있는 확장성을 갖고 있다.  5. 어떠한 클라이언트측 프레임워크와도 협업할 수 있고, 다양한 형식(HTML, RSS 피드, JSON, XML 등)으로 컨텐츠를 전송할 수 있다. | 1. 작업을 정의하고 수행하는 특정한 방법이 존재하기 때문에 개발자가 직접 파일 구조를 사용할 수 없다.  2. 서버의 처리와 시간이 걸리는 코드를 제공하므로 기능과 요구사항이 적은 프로젝트에는 적합하지 않다.  3. 개별 프로세스가 여러 요청을 동시에 처리할 수 없다. |

**2. 웹에서 AMD사의 pdf 문서 “Software Optimization Guide for AMD Family 15h Processors (Revision 3.08, January 2014)” 문서를 다운로드 하여 Chapter 3. C and C++ Source-Level Optimizations에 기술되어 있는 코드 최적화 방법 중에서 Loop Unrolling을 제외한 3개를 선택하여 1500자 분량으로 명료하게 설명하라. 특히 source-level code optimization에 대한 개념이 없는 컴퓨터공학 전공 2학년에게 설명하는 수준에서 이해하기 쉽게 기술할 것. (3번 문제의 1500자 분량 예시 참조)**

첫번째 방법은 Arrange Boolean Operands for Quick Expression Evaluation이다. 이 방법은 If문에서 AND(&&) 연산자나 OR(||)연산자를 사용할 때, 빠르게 조건의 참/거짓을 확인하여 if문 실행 여부를 판단할 수 있도록 하는 방법이다. AND(&&) 연산자를 사용할 때는, 첫번째 조건부터 마지막 조건까지 false가 될 확률이 높은 순서대로 나열하고 OR(||) 연산자를 사용할 때는, 첫번째 조건부터 마지막 조건까지 true가 될 확률이 높은 순서대로 나열한다. 왜냐하면 AND의 경우, 조건 중 하나라도 false가 되는 조건이 존재하면 결과 값은 항상 false이고, OR의 경우에는 조건 중 하나라도 true가 되는 조건이 존재하면 결과 값이 항상 true이기 때문이다. 따라서 AND 연산자를 사용하는 경우에는, 첫번째 조건부터 false가 될 가능성이 높은 것을 배치하면 false가 충족되는 조건 이후의 조건들에 대하여 참/거짓을 판별할 필요가 없다. 그리고 OR 연산자를 사용하는 경우에는 첫번째 조건부터 true가 될 가능성이 높은 것을 배치하면 true가 충족되는 조건 이후의 조건들에 대하여 참/거짓을 판별할 필요가 없다. 즉 컴파일러가 evaluation을 덜 하기 때문에 코드가 최적화될 수 있다.

두번째 방법은 Unnecessary Store-to-Load Dependencies이다. Store-to-load dependency는 데이터가 메모리에 저장되었다가 곧바로 메모리로부터 데이터를 다시 읽어올 때 존재한다. 이러한 dependency를 줄이고 데이터를 내부의 레지스터에 저장 해놓고 사용하면 코드를 더 빨리 실행할 수 있다. 그 이유는 레지스터에 접근하여 데이터를 가져오는 속도가 메모리에 직접 접근하여 데이터에 가져오는 속도보다 빠르기 때문이다. 따라서 메모리에 접근을 자주 할수록 CPU의 데이터 처리 속도가 느려질 것이기 때문에 데이터를 레지스터에 저장해두고 사용하는 것이 유리하다.

for(i=1;i<len;i++){

x[i] = x[i-1] + y[i];

}

이 경우를 생각하여 보자. 이 경우, x[i]에 데이터를 쓰는 시간, x[i-1] 값을 메모리로부터 가져오는 시간, y[i] 값을 메모리로부터 가져오는 시간이 발생한다. 즉 3번 메모리에 접근한다.

t = x[0]’

for(i=1;i<len;i++){

t = t+y[i];

x[i] = t;

}

접근 시간을 줄이기 위하여 임의의 변수 t = x[0]으로 선언하여 사용하면, t는 레지스터를 사용하기 때문에 매번 x[i-1]을 메모리로부터 불러오는 시간을 없앨 수 있다. 이는 for loop을 수행하는 횟수가 커질수록 차이가 더 많이 나게 될 것이다.

세번째 방법은 Generic Loop Hoisting이다. 이 방법은 for loop에서 반복되는 불변의 연산을 밖으로 빼내어 코드 수행시간을 줄일 수 있는 방법이다. 반복되는 불변의 연산이란 for loop에 영향을 받지 않아 값이 변하지 않으면서 매번 반복되는 연산이다. For loop에 영향을 받지 않는다면 for loop 밖으로 꺼내 한 번만 연산하고 for loop 내부에서 가져다 쓰기만 하면 연산 수행 시간을 줄일 수 있다.

for (i...) {

if (CONSTANT0) {

DoWork0(i); // Does not affect CONSTANT0.

}

else {

DoWork1(i); // Does not affect CONSTANT0.

}

}

이 경우를 생각해보면 CONSTANT0은 아무런 영향을 받지 않는 불변 값인데 for loop이 실행될 때마다 if문을 거치게 되어 매번 비교 연산 시간이 든다.

if (CONSTANT0) {

for (i...) {

DoWork0(i);

}

}

else {

for (i...) {

DoWork1(i);

}

}

따라서 if 문을 for loop 밖으로 꺼내면 if문에 대한 조건 비교는 한 번만 수행할 수 있고 매번 for loop 안에서 if 문을 거치지 않아도 되어 수행시간이 줄어든다. 이렇게 반복되는 불변 값을 반복문 밖으로 빼내어 코드를 최적화할 수 있다.

**3. CUDA 시스템에서 호스트에서 디바이스로 커널을 호출할 때 그리드 안의 쓰레드들이 SIMT 형태로 병렬 처리되는 과정에 대하여 1500자 분량으로 명료하게 기술하라. 이를 위하여 kernel launch, grid, thread block, thread, GPU scheduler, warp, SIMT, zero-cost context switching, memory latency hiding 등의 용어를 포함하며, 반드시 자신의 언어로 CUDA에 대하여 잘 모르는 컴퓨터공학 전공 2학년에게 설명하는 수준에서 이해하기 쉽게 기술할 것. (제출 내용에 대하여 copy-check를 진행할 예정임)**

동일한 계산을 서로 다른 데이터에 대하여 반복할 때, 이 연산을 여러 개의 프로세서에 대하여 병렬적으로 수행시킨다면 수행 시간을 크게 줄일 수 있다. GPU는 SM(Streaming Multiprocessor)들로 구성되어 있는데, 각 SM은 Processing Block들로 나누어져 있고 각 Processing Block들은 CUDA 코어들로 나누어져 있다. 이 CUDA 코어가 병렬적인 연산을 수행하는 하나의 프로세서 역할을 한다. CPU의 프로세서를 호스트, GPU의 프로세서를 디바이스라고 한다.

호스트 쪽에서 메모리 공간을 할당하는 cudaMalloc과 메모리 데이터를 복사하는 cudaMemcpy와 같은 CUDA runtime 함수들을 사용하여 디바이스 메모리 공간에 호스트 메모리를 적절히 할당하여 데이터를 주고받을 수 있다. 그리고 blockDim과 gridDim을 지정하고 계산 커널(kernel)을 호출한다. 디바이스에서는 여러 개의 grid가 존재할 수 있고 grid 안에는 여러 개의 block이 존재한다. blockDim과 gridDim을 지정하는 것은 몇 개의 grid와 block이 존재할 것인지 지정해주는 것이다. 계산 kernel은 각 thread에서 수행하는 한 개의 프로그램을 의미한다. 프로그래머가 한 개의 task가 한 개의 core에서 수행되는 계산에 집중하여 작성한 프로그램이다.

CUDA core가 task를 처리하는 방법은 task를 여러 개로 나누어 병렬적으로 처리하는 것이다. 처리해야할 task를 동일한 task 개수를 갖는 블록 단위로 나눌 수 있는데 이를 thread block이라고 한다. 이 thread block들은 GPU를 구성하는 SM으로 할당된다. 하나의 thread block은 동일한 SM 안에서 처리된다. SM이 실행할 수 있는 thread block의 개수는 SM이 실행할 수 있는 총 thread의 개수, SM의 레지스터 개수, SM의 shared memory 개수에 따라 달라진다. GPU scheduler는 이 조건들을 따져 thread block들을 scheduling한다.

Thread block 내의 thread는 한꺼번에 수행되는 것이 아니라 32개의 thread를 갖는 warp 단위로 계산 커널을 통해 계산을 수행한다. 한 warp 내의 32개 쓰레드들은 동시에 SIMD 방식으로 처리가 되는데 어떤 warp가 먼저 계산이 되는지는 GPU scheduler가 스케줄링한다. 이러한 방식을 사용하여 얻을 수 있는 장점은 시간 소요가 큰 작업을 하는 동안 작업이 끝날 때까지 마냥 기다리는 것이 아니라 다른 warp로 switching하여 연산을 진행할 수 있다. 이를 memory latency hiding이라고 한다. 프로그래머 입장에서는 어떤 warp가 먼저 계산되는지 알 수 없지만 CUDA 코어에 할당되어 커널 프로그램이 수행될 때, thread가 어떤 task에 대응되었는지 알 필요가 있다. 이를 위하여 blockDim.x, blockIdx.x, 그리고 threadIdx.x를 사용하는데, 이를 통해 task에 대하여 indexing 할 수 있다. 이를 통해 계산 커널을 가지고 계산을 할 때, 각 thread가 몇 번째 task에 대응되는지 파악하여 적절한 연산을 수행할 수 있다.

CUDA 메모리 계층 구조에 따르면 thread는 자신만의 register와 local memory를 가지며, 한 block 내에 존재하는 thread끼리는 서로 공유하기 위한 shared memory가 있고, grid는 데이터 공유를 위한 global memory가 존재한다. Register와 shared memory는 on-chip memory로서 접근 속도가 off-chip memory인 global memory보다 빠르기 때문에 shared memory를 이용한 병렬 처리 프로그램을 계산 kernel에 작성하면 더욱 빠르게 연산을 수행할 수 있다.

이러한 방식으로 GPU를 통해 task에 대하여 병렬적으로 처리할 수 있고, SIMD 방식이 multithreading을 통해 이루어지므로 SIMT라고 한다.