**고급소프트웨어실습**

**기말 리포트**

**(CSE 4152)**

**Due: 2021년, 12/15일, 23:59분**

**학번 : 20171646**

**반 번호 : 4반**

**이름 : 박태윤**

**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. 가상 DOM을 사용한다. 가상 DOM을 먼저 업데이트 하고 실제 바뀐 부분만 실제 DOM에서 바꾼 뒤 다시 렌더링을 하는 방식으로 DOM의 조작을 최소화하여 빠른 처리를 기대할 수 있다.  2. template이나 model과 같은 것들을 Component라는 단 하나의 단위로 관리를 해 유지 및 보수를 용이하게 할 수 있다.  3. 페이지의 내용을 브라우저에서 그리는 클라이언트 사이드 렌더링과 서버에서 페이지를 다 그려 브라우저로 넘겨주는 서버 사이드 렌더링 두 가지 방식 모두 가능하다. | 1. 인터넷 익스플로러 8(IE8)이하의 버전에서는 지원되지 않는다.  2. 사용자에게 보이는 부분인 view이외의 기능들은 라이브러리를 학습해 사용하거나 직접 구현해야 하는 단점이 존재한다.  3. 서버사이드 렌더링(SSR)방식으로 구현을 했을 시 불필요한 부분까지 렌더링 하는 overhead가 발생하기 때문에 느려질 수 있다는 단점이 존재한다. |
| Vuejs | 1. 데이터 제어 로직과 MVVM패턴으로 화면의 요소를 제어하는 코드를 분리하여 코드를 더 직관적으로 볼 수 있고 유지 및 보수 측면에서 용이하다는 장점이 존재한다.  2. single file component로 개발하는 방식이기 때문에 js나 jquery로 개발을 했던 개발자 등에게 익숙한 러닝 커브가 낮다는 장점이 존재한다.  3. 화면에서 작은 단위로 분리하여 구현을 하기 때문에 이에 대한 re-use측면에서 효율적이며 구현이 빠르다는 장점이 존재한다. | 1. Template에 기반한 개발을 하는 툴이기 때문에 run-time에러가 나오기 쉽고 테스트하기가 어려우며 재구조화하기 어렵기 때문에 규모가 큰 application을 단체로 개발할 때 불편하다는 단점이 존재한다.  2. react에 비해 좁은 생태계를 가지고 있어 이에 관련해 찾아볼 수 있는 자료 등이 한정되어 있다는 단점이 존재한다. |

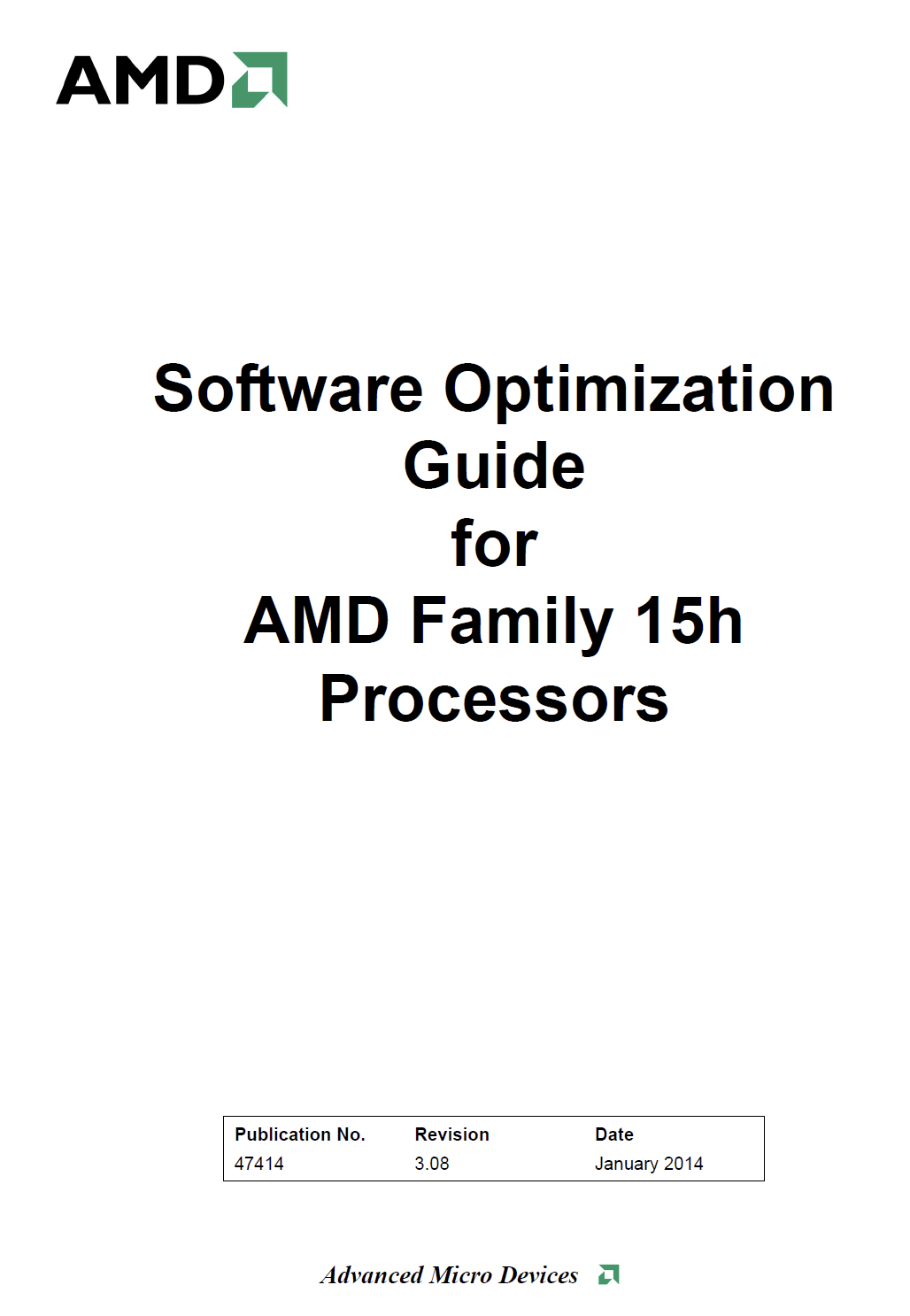
|  |  |  |
| --- | --- | --- |
|  | 장점 | 단점 |
| Spring framework | 1. 기존에 존재하는 라이브러리들을 스프링에서 감싸는 정도로 사용을 할 수 있기 때문에 많은 라이브러리가 지원되며 별도로 분리해서 사용하기도 용이한 확장성이 높다는 장점이 존재한다.  2. 메소드마다 반복되는 로직 중 공통되는 부분을 분리하는 관점지향 프로그래밍(AOP)방식을 사용하여 반복되는 코드를 줄일 수 있는 장점이 존재한다.  3. 대한민국의 공공기관에서 웹 서비스를 개발할 시 이 스프링 프레임워크를 표준 프레임워크로 채택하고 있기 때문에 국내 한정으로 수요가 크다.  4. 객체지향의 원리를 따르면서 필요에 따라 쉽게 재활용될 수 있는 오브젝트인 POJO(Plain Old Java Object)를 베이스로 하여 상속 및 인터페이스가 필요 없는 단순한 객체인 POJO를 통해 원하는 business logic만 넣을 수 있도록 enterprise application개발을 쉽고 간단하게 할 수 있다. | 1. 스프링 프레임워크 자체의 형식에 맞춰져 있기 때문에 자유롭고 유연한 개발에 한계가 존재한다.  2. 내부에 많은 기능들이 존재하지만 그만큼 용량이 큰 무겁고 이를 습득하기 위해 많은 시간을 투자해야 한다는 단점이 존재한다. |
| Node.js | 1. spring에 비해 하나의 쓰레드로 클라이언트에 대한 작업을 수행하며 I/O작업을 비 동기적이고 blocking없이 수행할 수 있다.  2. event에 기반한 비동기방식으로 구현이 되므로 서버에 대한 부담이 적다.  3. node package manager를 통해 다양한 모듈을 제공하여 효율성을 높일 수 있다는 장점이 존재한다. | 1. 단일 쓰레드 방식이기 때문에 CPU사용이 많은, 작업량 자체가 많이 걸리는 서비스에는 어울리지 않는다.  2. 비동기 방식으로 처리를 하기 위해 콜백 함수를 연속해서 사용하게 되면 call back hell에 빠질 수 있다. |

**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 있은 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)**

**C에서는 배열에 접근하기 위해 ‘[]’를 사용하거나 포인터를 사용하는 두 가지 방법을 제공한다. 포인터는 메모리의 주소를 참조를 할 수 있도록 하는데, 여기서 pointer alias가 발생할 수 있다. pointer alias는 두 개 이상의 포인터 변수가 메모리에서 같은 위치를 가리키고 있을 때를 의미하는데, 예를 들어 \*a와 \*b라는 같은 타입의 두 포인터 변수가 존재하고 이를 모두 1씩 increment하는 코드를 만들었다면 각각 다른 영역에 1씩 증가된 값을 저장하는 것이 아닌 같은 영역에 1씩 2번 증가시키는, 총 2가 증가된 값을 갖게 될 수 있다는 것이다. 물론 코드 작성자가 이를 유도했을 수도 있다. 어쨌든 고급 언어로 작성된 코드를 low level 언어로 바꿔주는 컴파일러는 실제로 두 포인터가 aliasing하고 있는지 아닌지를 모르기 때문에, low level 언어에서 위와 같은 두 가지 포인터에 대한 코드를 컴파일 하려고 할 때, 한 포인터를 먼저 로드하고 작업을 수행한 뒤 다른 포인터를 로드하고 작업을 수행한다. 하지만 이는 두 영역에 대해 load를 바로 두 번 하는 방식보다 메모리 접근에 대한 지연의 영향을 커지게 만든다. 따라서 aliasing이 된 경우를 의도하는 코드가 아니라면 포인터로 해당 연산을 처리하는 것 보다 pointer alias가 확실하게 일어나지 않는 ‘[]’를 사용해 이를 처리하는 것이 최적화 면에서 좋은 코드이다.**

**C/C++에서 조건문을 사용할 때 and를 나타내는 ‘&&’연산자나 or를 나타내는 ‘||’연산자를 많이 사용하게 된다. 예를 들면 다음과 같이 사용할 수 있다. if(a == 1 && b == 2), if(c == 3 || d == 4). 이 때 ‘&&’연산자를 사용한 조건문의 예시에서 만약 실제 a에는 1이, b에는 3이 assign되어 있으면 a == 1이 true이므로 b == 2까지 수행한 다음에 이것이 false임을 알게 되고 조건문 안의 코드를 수행하지 않는다. 만약 해당 조건문을 if(b == 2 && a == 1)로 바꿔본다고 가정하자. 이 때는 b가 3으로 2가 아니기 때문에 굳이 a == 1을 수행하지 않고도 조건문을 종료하게 된다. and연산자에서는 교환법칙이 성립하기 때문에 두 조건문은 같은 조건문이라고 볼 수 있으며 if(b == 2 && a == 1)경우가 불필요한 비교 연산인 a == 1을 수행하지 않기 때문에 좀 더 최적화가 된 코드라고 볼 수 있다. or의 경우에서도 마찬가지로, c에 4, d에도 4가 assign되어 있으면 기존의 예시인 if(c == 3 || d == 4)보다 if(d == 4 || c == 3)으로 바꿔주는 것이 d == 4비교연산만 수행하고 조건문 안의 코드를 수행시킬 수 있기 때문에 최적화가 좀 더 잘 된 코드라고 할 수 있다. 따라서 조건문 안에서 ‘&&’연산을 수행하는 경우 좀 더 false가 많이 나올 만한 조건을 앞에 두는 것이 좋고 반대로 ‘||’연산을 수행하는 경우 좀 더 true가 많이 나올 만한 조건을 앞에 두는 것이 코드 최적화 면에서 좋다고 볼 수 있다.**

**C / C++에서 signed 과 unsigned는 각각 양수 값 만을 가지는 타입과 음수와 양수 값을 모두 가지는 타입을 나타낸다. 가령 unsigned int는 0 ~ 4,294,967,295를, int는 -2,147,483,648 ~ 2,147,483,647을 나타낸다. 이 때, integer to float연산에서는 AMD64 아키텍처에서 부호가 있는 정수를 부동 소수점으로 변환하는 명령어를 제공하지만 부호가 없는 정수를 부동 소수점으로 변환하는 명령어는 제공되지 않기 때문에 이러한 경우에서는 signed integer를 사용하는 것이 low level 명령어의 개수를 줄일 수 있어 최적화 면에서 더 좋다. 반대로 음수 값이 나올 수 없는 배열의 인덱스나, 루프의 카운터를 i = 0 ~ 100등과 같이 음수 값으로는 돌리지 않는 경우는 unsigned integer를 사용하는 것이 더 좋다.**

****

**3. CUDA 시스템에서 호스트에서 디바이스로 커널을 호출할 때 그리드 안의 쓰레드들이SIMT 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)**

**CUDA를 이용하면 CPU가 아닌 GPU를 이용해서 수행하고자 하는 연산을 처리할 수 있다. GPU를 이용해 연산을 처리하는 경우는 어떤 비슷한 내용의 계산이 반복되는 경우에 연산 처리 속도가 빠르다는 장점을 지닌다. 단, GPU에서 연산을 처리하기 위해서는 CPU에서 계산을 할 때처럼 기존의 C/C++코드대로 function을 호출하는 것이 아닌 GPU에서 실행되는 함수인 kernel를 구동(launch)하는 kernel launch해주는 코드가 필요하다. SIMT는 Single Instruction Multi Thread의 약자로, 하드웨어가 여러 개의 쓰레드들을 그룹화하여instruction을 여러 쓰레드들이 병렬적으로 동시에 처리를 하는 방법을 나타낸다. GPU를 이용한 연산은 kernel launch를 통해 수행하고 여기서는 인자로 grid의 크기와 thread block의 크기를 받는다. grid는 GPU를 이용한 연산에서 virtual한 가장 큰 단위이고 이는 여러 개의 thread block으로 나뉜다. 각 thread block은 또 여러 개의 thread로 나뉜다. 이 thread들이 SIMT방식에서 하나의 instruction을 담당하게 된다. thread block은 또 여러 개의 warp들로 나뉘는데, warp라는 단위는 NVIDIA GPU에서 총 32개의 thread를 가지고 한 warp내의 thread들은 같은 instruction을 수행하며 이 instruction들은 동시에 수행이 된다. 그리고 실제 GPU를 보면 여러 개의 Streaming Multiprocessor라는 것으로 구성이 되는데, 한 Streaming Multiprocessor는 여러 processing block들로 나뉘고 이 processing block은 앞서 언급한 한 warp가 실행되는 단위라고 볼 수 있다. GPU scheduler가 어떤 쓰레드 블록이 어떤 Streaming Multiprocessor내에서 수행이 될 지 스케줄링을 해주면 그 속에서 warp단위로 processing block에서 연산을 처리하는 과정이 수행되는데, 만약 어떤 warp가 해당 연산에 필요한 데이터를 메모리에서 가져오는 등의 이유로 메모리에 접근하는 과정을 수행하여cycle을 많이 잡아먹게 되는 memory latency가 발생하게 된다면 zero-cost context switching을 통해 다른 warp가 processing block에 배정되어 스케줄링을 통해 조금 더 효율적인 연산 처리를 할 수 있도록 한다. 한 processing block에 배정되어 수행이 되는 warp를 바꾸는 작업을 처리하는 과정이 비용이 적다는 것을 의미하는 zero-cost라 할 수 있는 이유는 warp안의 thread마다 자신만의 레지스터를 가져 레지스터 개수가 적은 CPU에서 어떠한 쓰레드가 수행되는지 flow가 변경되는 context switch처럼 해당 thread의 레지스터 내용을 메모리에 store하고 수행하고자 하는 thread의 레지스터 내용을 메모리에서 load해오는 과정에서 발생하는 추가적인 cycle없이 context switch가 일어날 수 있기 때문이다. 그리고 이와 같이 적은 cost로 context switching을 수행하여 processing block에서 수행되는 warp중 memory에 접근하여 latency를 발생시키는 warp를 후순위로 scheduling함으로써 메모리 접근에 대한 대기 시간을 숨기는 것을 memory latency hiding이라고 한다. 따라서 GPU를 이용하면 이러한 방식으로 반복되는 계산을 warp단위로 병렬적으로 동시에 처리할 수 있기에 특정 연산에 대해 걸리는 시간을 CPU에서 처리하는 것 보다 줄일 수 있다.**