멀티코어 환경을 위한 현대 동시성 프로그래밍

김남규* · 강영진** · 이훈재***

*동서대학교

Modern Concurrent Programming for Multicode Environment

Nam-gue Kim* · Young-Jin Kang** · HoonJae Lee***

*Dept. of Information and Communication Engineering, Dongseo University

**Dept. of Ubiquitous IT Graduate School of Dongseo University

***Dept. of Computer Engineering, Dongseo University

E-mail : knq1130@naver.com, rkddudwls55@gmail.com, hjlee@dongseo.ac.kr

요 약

멀티코어 이전의 시대에는 하드웨어의 발전을 토대로 프로그램 성능 향상에 도움을 받을 수 있었다. 그러나 하나의 코어 대한 성능 향상이 한계에 봉착하며 여러 개의 코어를 사용하는 멀티코어라는 방법이 보편화되었다. 멀티코어를 사용하기 위해 커널 레벨의 스레드를 사용하는 기존의 방법을 발전시킨 현대적 동시성 프로그래밍이 대두되었다. 현대 동시성 프로그래밍이란 경량 스레드를 사용하여 멀티코어의 장점을 최적화한다. 또 공유 가능한 데이터가 변경할 수 있는지의 유무가 동시성 코드 작성 시 주요 고려사항이 된다. 본 논문은 동시성 코드 작성 시 주요 고려사항들을 설명하고, 이러한 사항들이 현대 동시성 기술을 지원하는 언어 중 하나인 '고'에서 어떤 식으로 지원되고, 나아가 어떻게 더 훌륭한 동시성 코드를 작성할 것인지 논의한다.

ABSTRACT

The period of the previous multi-core could be helped to improve program performance, based on the development of the hardware. However, one of the core performance enhancements for this encounter limitations and become the common way of multi-core with multiple cores. Modern programming concurrency that improves the conventional method for using threads of the kernel level in order to use the multi-core come to the fore. Using modern lightweight thread concurrency programming is to optimize the benefits of multi-core. Also sharing the absence of available data that can change is a major consideration when writing concurrent code . This paper describes the key considerations when creating a discussion concurrent code, and these issues are being supported in any way in the language of one 'go' of technologies that support the modern concurrency, and even how to write better code concurrency.

키워드

multi-core, concurrency, modern language, go

Ⅰ. 서 론

멀티코어 이전의 시대에는 하드웨어의 발전을 토대로 프로그램 성능 향상에 도움을 받을 수 있었다. 그러나 하나의 코어 대한 성능 향상이 한계에 봉착하며 여러 개의 코어를 사용하는 멀티코어라는 방법이 보편화되었다. 멀티코어를 사용하기 위해 커널 레벨의 쓰레드 자체를 사용하는 기존의 방법을 발전시킨 현대적 동시성 프로그래밍 방법이 대두되었다. 현대 동시성 프로그래밍이란 경량 쓰레드를 사용하여 멀티코어의 장점을 최적화한다. 본 논문의 2장에선 동시성 프로그래

밍의 근간이 되는 이론 및 개념들을 기술한다. 3장에선 현대적 동시성 프로그래밍의 개념과 현대적 동시성 기법을 지원하는 프로그래밍 언어를 소개하고, 3장에선 2장에서 소개한 언어 중 하나인 go 언어가 어떠한 형식으로 동시성 기법을 지원하는지 프로그래밍적 관점에서 논의한다.

II. 현대 동시성 프로그래밍의 배경

2장에선 병행성과 병렬성에 대한 개념을 기술하고, 동시성 프로그래밍이 등장한 배경을 하드웨어와 소프트웨어 측면으로 나눠 소개한다.

2.1 병행성(concurrent)과 병렬성(parallel)

그림 1. 병행성과 병렬성

병행성이라 함은 동시 처리의 논리적인 개념으로 여러 개의 스레드가 겉보기엔 동시에 실행되는 것처럼 보이지만 실제로는 스레드 여러 개가 시간을 쪼개어 시분할로 처리되는 것을 뜻합니다.

병렬성은 동시 처리의 물리적인 개념으로, 작업을 여러 CPU 코어에 나눠서 동시에 처리하는 상태를 뜻합니다.

2.2 하드웨어

멀티코어라는 개념이 등장하기 이전, 프로그래머들은 하드웨어의 성능 향상으로 소프트웨어의 자동적인 속도 향상의 효과를 누렸다.

지난 30년간 CPU 클럭의 증가, 실행 최적화, 캐싱과 같은 영역에서 지속적인 향상이 있었다. 그러나 CPU 클럭이 증가하는 것은 물리적인 한계에 봉착하였고 최적화로 인한 성능 향상을 기대하는 것 역시 한계가 있다. 캐시 자체의 크기를 증가하여도 하드웨어에서 기대했던 속도 향상을 얻을 수 없다. 물론 일정부분 속도 향상에 기여한다. 결국 CPU의 성능을 효과적으로 올릴 수 있는 방법을 코어를 병렬화하여 소프트웨어 측에서 동시 또는 병렬 실행을 통하여 성능을 올려야 한다.

다시 말해서 직렬 최적화의 시대가 끝나고 병렬 최적화의 시대가 열린 것이다.

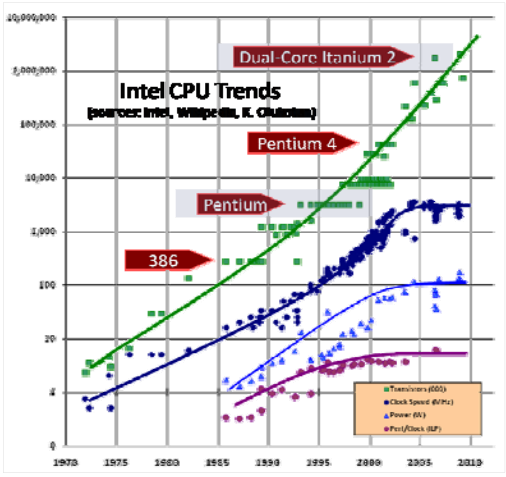

그림 2. 하드웨어 기술의 발전

2.3 소프트웨어

멀티코어를 사용할 때 고려해볼 개념은 두 가지가 있다. 첫째, 어떤 종류의 쓰레드를 사용하는지 고려해야 한다.

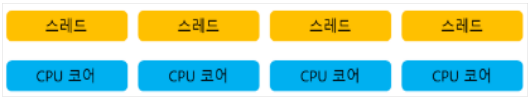

그림 3. 커널 레벨 스레드

<그림 3>에서 CPU 코어 하나 당 쓰레드 하나를 사용하는데 이는 커널 레벨에서 지원하는 스레드를 사용한다는 의미이다. 커널 레벨에서 지원하는 스레드를 사용하면 큰 부담이 감으로 코어를 적절하게 사용한다고 보기 어렵다. 예컨대, “hello world’와 같은 문자열을 출력하는 코드에 커널 레벨 쓰레드를 생성한다면 불필요한 리소스를 잡아먹을 수 있게 된다.

두 번째 고려사항은 다루는 데이터가 공유 가능하고 그 데이터를 변경할 수 있는지 여부다.

```
public synchronized void exampleMethod() {
        //code
        ...
        ..
}


private Object obj = new Object();
public void exampleMethod() {
        synchronized(obj) {
                //code
                ...
                ..
        }
}
```

그림 4. java synchronized

<그림 4>은 다루는 데이터가 공유 가능할 경우에 붙이는 키워드 “synchronized”의 경우로, 동일한 시간에 1번만 접근할 수 있는 동기화이다. 유사한 동기화 기법으로 뮤텍스, 세마포어, 이벤트 기반 동기화 기법 등이 있다.

이러한 기법들은 공유가 가능한 데이터가 변경 가능할 때 이에 대한 접근을 산발적으로 처리할 때 사용될 수 있다. 산발적이라 함은 처리 순서를 보장하지 않는다는 뜻이다. 즉, 산발적으로 처리함에 따라 순서를 보장하지 않아 실행 예측과 디버깅이 매우 어려워진다.

이러한 두 가지 정도 문제를 해결하기 위해 현대의 동시성 프로그래밍이 등장하였다.

III. 현대 동시성 프로그래밍

현대의 동시성 프로그래밍은 앞서 언급한 두 가지 고려사항에 대한 최선의 해결책을 제시한다. 3장에선 이에 관한 해결책에 관해 기술하고 어떠한 프로그래밍 언어들에서 동시성 기법을 도입하고 있는지 소개한다.

3.1 현대 동시성 프로그래밍

현대 동시성 프로그래밍에선 2장에서 언급한 고려사항인 어떤 종류의 스레드를 사용하는지, 다루는 데이터의 속성이 공유 가능 유무와 해당 데이터의 변경가능 유무에 대한 최선의 해결책을 제시한다.

첫 번째 해결책으로 적정량의 스레드, 다른 말로 경량 스레드를 생성한다. 이로 인하여 커널 레벨 쓰레드를 생성할 때의 부담을 경감할 수 있다. 두 번째 해결책을 위해 우리는 데이터의 불변성을 지향해야 한다. 스레드와 스레드간의 정보 교환시에는 따로 채널을 생성해서 관리해야 한다. <그림 5>와 같이 데이터를 공유할 필요가 있을 경우 아토믹 연산을 사용하여 순서를 유지해야하며, 되도록 채널을 사용한 데이터 교환을 권장한다.

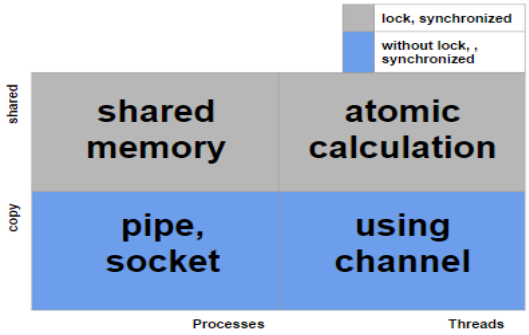

그림 5. 데이터 속성에 따른 방법 배치

추가적으로 함수적 언어들은 상수와 불변 객체를 가지는데 이는 동시성을 지원하기 위함이다.

3.2 현대 동시적 프로그래밍 언어

그림 6. 동시성을 지원하는 프로그래밍 언어

스칼라와 얼랭의 액터, 자바스크립트의 앨름, 고의 고루틴은 현대적 동시성 프로그래밍을 지원하는 도구이다.

IV. go의 동시성 프로그래밍

4장에선 고 언어에서 어떤 식으로 동시성 프로그래밍을 지원하고 있는지 실제 코드로 설명한다.

4.1 고 루틴

그림 7. 고 루틴의 경량 스레드

<그림 7>을 보면 고 루틴은 일반 스레드와는 다르게 고 루틴은 경량 스레드를 생성하여 성능을 최적화합니다.

```go
package main

import (
    "fmt"
)

func hello(n int) {

    fmt.Println(n)
}

func main() {
    for i := 0; i < 100; i++ {
        go hello(i) // go routine
    }
    fmt.Scanln()
}
```

그림 8. 고 루틴 예제

고 언어는 “go”라는 키워드를 통하여 고 루틴을 생성한다. <그림 9>의 예제는 고 루틴을 생성하여 매개변수 ‘i’ 값을 출력합니다.

그림 9. 고 루틴 예제 결과

<그림 9> 예제 코드의 결과 값을 확인해보면 순서가 산발적으로 나타나는 것을 알 수 있다.

이러한 값들은 채널을 이용하여 main 함수와 고 루틴들끼리 주고받을 수 있다.

4.2 채널

채널을 통하여 main 함수와 고 루틴이 데이터를 주고받을 수 있다. 채널을 사용하여 데이터를 주고받는 이유는 앞서 언급했던 상수, 또는 불변 객체를 사용하여 불변성을 유지해 동기화에서 발생하는 문제를 방지하기 위함이다.

<그림 10>의 예제는 고 루틴 안에서 channel에 값을 저장한 뒤 연산이 일어나면 그 값을 변수 ‘n’에 저장한다.

```
1  package main
2
3  import "fmt"
4
5  func sum(a int, b int, channel chan int) {
6      channel <- a + b
7  }
8
9  func main() {
10     channel := make(chan int)
11
12     go sum(1, 2, channel)
13
14     n := <-channel
15     fmt.Println(n)
16 }
```

그림 10. 채널 예제

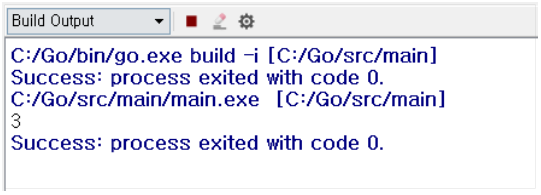

그림 11. 채널 예제 결과

<그림 11>에서 1과 2를 더해서 채널에 저장하므로 3이 결과값으로 출력된다.

고 언어에선 불가피하게 공유 데이터를 다루지 않는다면 되도록 채널을 사용한 데이터 통신을 권장한다.

V. 결 론

멀티코어를 잘 사용하는 것이 프로그램 성능 향상에 주요한 요소가 되어, 좋은 프로그래머가 되기 위해서는 그에 상응하는 동시성 코드를 만들 수 있는 능력이 필요해졌다.

여러 가지 현대 프로그래밍 언어에서 훌륭한 동시성 프로그래밍을 할 수 있는 도구들을 제공한다.

고 언어에서 고려해야할 사항을 충분히 이해한 뒤, 다른 언어들의 도구들을 사용한다면 하나의 플랫폼에서 작성한 것보다 더 낳은 동시성 코드를 작성하게 될 것이라 생각된다.

참고문헌

[1] http://www.gotw.ca/publications/concurrency-ddj.htm, The free lunch is over

[2] https://rein.kr/blog/archives/541, 암달의 법칙 깨기

[3] http://www.cnet.co.kr/view/6472, 함수형 프로그래밍이 주목받는 3가지 이유

[4] http://pyrasis.com/go.html, 가장 빨리 만나는 Go 언어