자바 가상 머신을 통한 ARX 쓰레드 라이브러리의 성능 측정

서양민 박정근 김기정○ 홍성수
서울대학교 전기공학부 실시간 운영체계 실험실

Performance Evaluation of ARX Thread Library in Java Virtual Machine

Yangmin Seo Jungkeun Park Ki-jung Kim Seongsoo Hong
Real-Time Operating Systems Laboratory
School of Electrical Engineering, Seoul National University, Seoul 151-742, Korea

요 약

쓰레드는 프로그램의 동시성을 표현하는데 적합하고, 프로세스 모델에 비하여 동기화나 문맥교환의 비용을 줄일 수 있어 기존의 멀티 프로세스 프로그래밍을 대체하고 있다. 운영체계에서 멀티쓰레딩 환경의 제공은 이제 필수적이며, 좋은 성능을 위해서는 운영체계의 지원이 필요하다.

ARX 실시간 운영체계는 유저 레벨 멀티쓰레딩을 지원하고 있으며 쓰레드의 성능을 높이고 유저 레벨에서 실시간 스케쥴링이 가능하도록 하기 위하여 동적 가상 쓰레드 바인딩(dynamic virtual stack binding)과 스케쥴링 이벤트 업콜(scheduling event upcall)등의 기법을 지원한다.

본 논문에서는 자바 가상 머신을 통하여 ARX 운영체계의 쓰레드 라이브러리의 성능을 측정하고 다른 운영체계의 멀티쓰레드 라이브러리와 성능 비교를 하였다. 실험결과 ARX가 제공하는 쓰레드 라이브러리가 다른 운영체계에 비해 우수한 성능을 보여줌을 확인하였다.

1. 서론

쓰레드는 프로세스에 비해 훨씬 가벼운 구조이며, 따라서 동기화나 문맥 교환같은 비용이 많이 드는 동작에 대해 훨씬 좋은 성능을 보장한다. 멀티쓰레드 프로그램은 서버/클라이언트 프로그래밍에도 적합하고 시스템의 오버헤드를 줄일 수 있는 장점이 있기 때문에 널리 쓰이는 프로그래밍 기법이다.

한편 자바는 플랫폼 독립적인 동일한 프로그래밍 환경의 제공을 목표로 Sun에 의해 개발된 프로그래밍 언어이다. 자바가 플랫폼 독립적인 환경을 제공하기 위해서는 각 플랫폼마다 자바의 바이트 코드를 분석하고 실행시키기 위한 자바 가상 머신(Java Virtual Machine)이 필요하다.

본 논문에서는 멀티쓰레드 라이브러리의 공정한 성능 평가를 위해서 플랫폼 독립적인 자바 프로그래밍 언어를 사용한다. 성능 평가를 위하여 공개용 자바 가상 머신인 Kaffe를 ARX환경에 이식하였고, 멀티쓰레드 자바 벤치마크 프로그램을 실행하여 Solaris, Windows98 운영체계 상의 쓰레드와 ARX 쓰레드의 성능을 비교하였다.

2. 연구배경

2.1 Kaffe 자바 가상 머신

본 논문에서 쓰레드 성능 평가를 위해 사용한 공개 자바 가상 머신인 Kaffe는 내부적으로 쓰레드 라이브러리를 가지고있다. 이 쓰레드 라이브러리는 운영체계와 독립적으로 제공된다. 또한 Kaffe는 옵션으로 운영체계 의존적인 쓰레드 라이브러리를 사용할 수 있도록 하고 있다. 본 논문에서 사용한 Kaffe는 ARX 실시간 운영체계의 쓰레드 라이브러리를 이용하여 컴파일하여 ARX 멀티쓰레드 환경의 성능을 멀티 쓰레디드 자바 프로그램을 통하여 측정할 수 있도록 하였다.

2.2 ARX 실시간 운영체계

ARX는 실시간 응용과 멀티 미디어 응용 프로그램을 효율적으로 수행하도록 설계된 실시간 운영체계이다[2]. ARX는 프로세스 사용량 할당 방식에 의한 프로세스 스케쥴링과 유저 레벨 쓰레드, 유저 레벨 입출력등을 지원한다. ARX의 쓰레드 라이브러리는 기존의 유저 레벨 쓰레드가 시그널 핸들링과 쓰레드 스케쥴링에 취약한 단점을 극복하기 위해 동적 가상 쓰

레드 바인딩(*Dynamic virtual thread binding*)과 스케줄링 이벤트 업콜(*Scheduling event upcall*)이라는 두 가지 방법을 제공한다.

동적 가상 쓰레드 바인딩은 커널에서 블록된 유저 레벨 쓰레드에 의해서 전체 프로세스가 선점(preempt) 되는 것을 막아준다. 여기서 가상 쓰레드(virtual thread)는 유저 레벨 쓰레드에게 커널 레벨의 가상 실행 환경을 만들어 주기 위한 것이다.

스케줄링 이벤트 업콜은 커널 이벤트를 유저에게 효율적으로 전달하는 방법으로서 유저 레벨 스케줄러가 이벤트에 대해 정확한 시점에 대응을 할 수 있도록 해준다.

2.3 Solaris

Sun은 [1]에서 LWP(Light Weight Process) 기법을 제안했다. ARX의 가상 쓰레드와 비슷하게 LWP는 유저 쓰레드에게 가상 실행 환경을 제공한다. 그러나 LWP는 가상 쓰레드와는 다르게 커널에 의해서 스케줄링 되는 오브젝트이다. Solaris는 Process scope와 System scope의 두 가지 형태의 쓰레드 라이브러리를 제공 한다. Process scope는 round-robin 스케줄링 정책에 의해 커널에 매핑된 쓰레드를 스케줄링하게 되고, System scope는 FIFO(First In First Out) 스케줄링 정책에 의해 쓰레드를 스케줄링하게 된다. 실험에 사용된 쓰레드 라이브러리는 Process scope인데 이것은 동일한 스케줄링 정책에 의한 성능측정을 위해서이다.

2.4 Windows98

Windows98에서의 성능평가를 위해 JDK 1.1.6에서 제공하는 자바 가상 머신을 사용하였 다. Kaffe같은 공개 자바 가상 머신은 JIT 컴파일이 가능하게 이식하기 어렵기 때문에 JIT 컴파일이 가능한 Sun의 자바 가상 머신을 사용하여 성능측정을 하였다. Windows용 Sun의 자바 가상 머신도 쓰레드 스케줄링 정책으로 round-robin 정책을 사용하므로 동일한 비교환경에서 성능비교를 할 수 있다.

3. 성능 측정

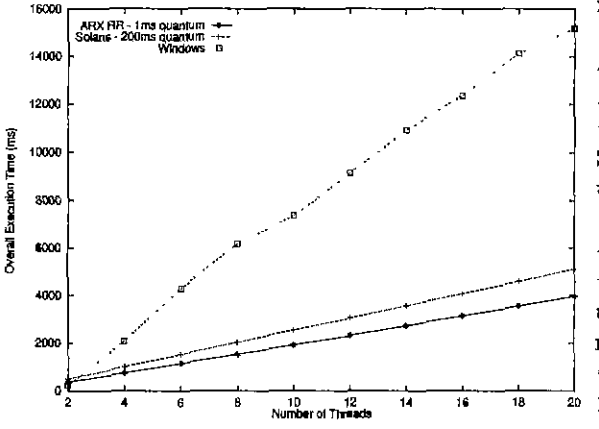
세 가지 다른 운영체계, ARX, Solaris, Windows98에서 멀티 쓰레드 자바 프로그램을 수행하였고, 쓰레드 라이브러리에서 가장 큰 오버헤드는 스케줄링시 발생하기 때문에, 각 운영체계 상에서 쓰레드 스케줄링시 발생하는 오버헤드를 비교하기 위해 수행시간을 측정하였다.

실험은 Intel 266MHz Pentium II 프로세서를 사용하는 기계에서 이루어졌다. 수행시간 측정은 50ns의 정밀도를 가지는 타이머 보드를 사용하여 이루어졌다.

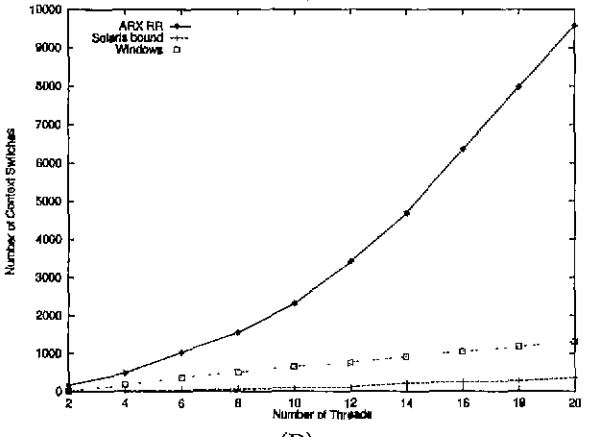
3.1 자바 벤치마크 프로그램

프로그램을 수행하게 되면 각 쓰레드가 synchronized 오브젝트에 의해 보호되는 글로벌 배열에 자신의 쓰레드 아이디를 100,000번 써 넣는 작업을 하게 된다. 첫번째 수행을 시작하는 쓰레드가 시작하면서 타이머 보드를 동작시키고, 마지막 수행을 마친 쓰레드가 타이머 보드를 멈춘다. 따라서 자바 프로그램이 수행한 총시간을 계산할 수 있다. 또 TBench.ids라는 배열에 현재 프로그램의 콘트롤을 가지고 있는 쓰레드를 기록하게함으로서 자바 프로그램의 수행 중에 발생하는 문맥교환의 횟수를 측정할 수 있다. 다음은 실험에 사용된 자바 프로그램의 소스이다.

```
public class TBench{
    public static int[] ids;
    public static int idCount = 0;
    pubic static int threadnum, activethreadnum;
    public static Object o = new Object();
    public static void main(String argv[]){

    threadnum = Integer.parseInt(argv[0]);
    activethreadnum = threadnum;
    ids = new int[threadnum*100000];
    Thread ta = new StartupThread();
    }
}

class StartupThread extends Thread{
    public StartupThread(){
        setPriority(Thread.MIN_PRIORITY+1);
    }
    public void run(){
        for(int i = 0; i < TBench.threadnum; i++){
                (new BenchmarkThread(i)).start();
        }
    }
}

class BenchmarkThread extends Thread{
    public int I;
    public BenchmarkThread(int index){
        I = index;
    }
    public void run(){
        int count = 0;
        yield();
        //start the external timer board
        if(I == 0)
            timerboard.start();
        while(count   100000){
            count++
            synchronized(TBench.o){
            TBench.ids[TBench.idCount++] = I;
            }
        }
        //stop the timer board
```



(A)



(B)

그림 1: Performance of thread scheduling in ARX, Solaris, and Windows 98: (A) aggregate execution times of all threads, and (B) corresponding numbers of context switches.

```
        if(--TBench.activethreadnum == 0)
            timerboard.end();
    }
}
```

3.2 쓰레드 스케쥴링 성능

ARX의 쓰레드 라이브러리를 이용한 Kaffe 자바 가상 머신이 ARX의 실험에 사용되었고, Solaris와 Windows98은 각각 pthread 라이브러리를 사용한 Kaffe와 JDK의 자바 가상 머신이 사용되었다. ARX의 자바 가상 머신은 Kaffe 0.10.0이 사용되었으며, JIT 모드로 동작하도록 이식되었다.

성능 측정은 쓰레드의 개수를 바꾸어 가면서 쓰레드의 총 수행에 걸리는 시간을 측정하여 이루어졌다. 쓰레드의 스케쥴링 성능만을 평가하기 위하여 자바 클래스 파일을 로드하는 시간은 제외하였고, 자바 프로그램의 모든 쓰레드는 같은 우선순위를 가지고 round-robin 스케쥴링 정책에 의해 스케쥴링 된다.

그림 1은 실험 결과를 요약한 것이다. 그림 1(A)는 세 가지 다른 운영체계 상에서 총 실행시간을 나타내고, 그림 1(B)는 문맥 교환 횟수를 나타낸다. ARX는 훨씬 더 많은 문맥교환이 발생했음에도 불구하고 Solaris와 Windows98 보다 훨씬 적은 스케쥴링 오버헤드를 가짐을 알 수 있었다.

예상과는 대조되게 Solaris에 대한 ARX의 성능향상은 근소한 차이임을 알 수 있다. 이것은 Solaris가 커널 쓰레드를 사용하지만 mutex lock/unlock primitive는 유저 레벨 라이브러리로 구현되었고, Solaris의 round-robin quantum 크기가 매우 커서 쓰레드 스케쥴링 오버헤드가 총 실행시간에 거의 영향을 주지 않기 때문이다. ARX의 quantum 크기는 1ms이고 반면에 Solaris의 quantum 크기는 200ms이다. 한편 ARX는 상당한 차이로 Windows98보다 우수한 성능을 나타냈다. Solaris와 다르게 Windows는 mutex primitive들이 시스템 콜로 만들어져 있기 때문이다. Solaris의 quantum 크기가 작아진다면 ARX가 Solaris보다 훨씬 좋은 성능을 보일 것이라고 생각한다. 이 예상은 다음과 같은 간단한 계산에 의해서 해 볼 수 있다. 20개의 쓰레드를 Solaris에서 동작시켰 을때 200ms quantum 크기로 걸린 총 실행 시간은 5109ms이었다. round-robin quantum이 1ms이라면, 거의 5000번의 추가적인 문맥교환이 발생할 것이다. Solaris 에서의 스케쥴링 오버헤드는 200 μs였기 때문에 1000ms의 추가적인 시간이 총 실행 시간 에 더해 지게 된다. 증가된 시간은 반복적으로 문맥교환의 수를 늘이게 될 것이다. 결과적으로 총 수행시간은 더 증가하게 될 것이다. 이 총 수행시간이 수렴할 때까지 계산해보면 6,374ms가 된다. 이 값은 원래 값보다 25%정도 증가한 값이다.

4. 결론

본 논문에서는 ARX에 구현된 쓰레드 라이브러리를 자바 가상 머신에 이식하여 다른 운영체계의 멀티쓰레드 환경과 성능비교를 하였다. 실험 결과에 따르면 ARX의 쓰레드는 문맥교환의 오버헤드가 작기 때문에 Solaris나 Windows98의 쓰레드보다 좋은 성능을 나타냄을 알 수 있다.

참고 문헌

[1] M.L. Powell, S.R. Kleiman, S. Barton, D. Shah, D. Stein, and M. Weeks. SunOS multi-thread architecture. In *Proceedings of 1991 USENIX Winter conference*, pages 65–79, 1991.

[2] Y. Seo, J. Park, and S. Hong. Supporting preemptive user-level threads for embedded real-time systems. Technical Report SNU-EE-TR-1998-1, School of Electrical Engineering, Seoul National University, August 1998.