제 19 회 한국정보처리학회 춘계학술발표대회 논문집 제 10 권 제 1 호 (2003. 5)

자바가상머신에서 다차원 배열의 효율적인 접근과 성능 개선

이지현*, 원희선**, 문경덕**, 김영국*
*충남대 컴퓨터 과학과
** 한국전자통신연구원 정보가전연구부
e-mail : {jhlee, ykim}@cs.cnu.ac.kr/{ hswon,kdmoon}@etri.re.kr

Efficient Access and Performance Improvement of Multi-Dimensional Array in Java Virtual Machine

Ji-Hyun Lee*, Hee-sun Won**, Kyung-Doek Moon**, Young-Kuk Kim*
* Dept. of Computer Science, Chungnam National University
** Information Appliance Technical Department, ETRI

요 약

자바는 플랫폼에 독립적인 실행을 위해서 바이트코드를 사용하며, 자바가상머신에서 바이트코드를 해석하여 실행시키는 인터프리터 방식을 취하고 있으므로 느린 실행 시간을 갖는다. 이와 같은 느린 실행 시간에 영향을 주는 요인 중의 하나로써 다차원 배열(Multi-dimension Array)에 접근(Acess)할 때 비효율적인 방법으로 실행되는 것을 개선할 필요가 있다. 자바가상머신에서 오브젝트와 배열은 레퍼런스에 의해서 접근 되어지고, 특히 다차원 배열은 배열 안에 또 다른 배열이 들어있는 자료구조를 취하기 때문에 다차원 배열의 크기가 커질수록 하나의 원소에 접근할 때 발생되는 레퍼런스(Reference)의 횟수가 많아질수록 성능 저하를 유발한다. 본 논문에서는 다차원 배열에 효율적으로 접근하기 위한 새로운 배열의 자료 구조를 제시하고, 다차원 배열에 접근하는 여러 개의 바이트코드 대신 이런 자료구조에 접근할 수 있는 새로운 바이트코드를 정의 및 구현하였다. 그리고 이를 실제 클래스파일에 적용하고, 간단한 성능 평가를 통해서 성능 개선 효과를 확인할 수 있다.

1. 서론

자바는 어떠한 프로세서 혹은 하드웨어에서도 동작할 수 있는 높은 이식성을 바탕으로 광범위하게 사용되고 있다. 이것은 자바로 작성된 프로그램인 경우에 플랫폼에 독립적으로 실행되는 바이트코드를 만들어 내기 때문이다. 자바 바이트코드는 플랫폼에 독립적으로 실행될 수 있는 장점이 있지만, 스택을 기반으로 하는 가상기계 코드이고 인터프리팅을 통해 실행되는 특징으로 실행 속도가 현저히 느린 단점이 있다.

이런 느린 실행 시간에 영향을 줄 수 있는 것으로 다차원 배열에 접근할 때 비효율적인 방법으로 실행되는 것을 들 수 있다. 자바에서는 다차원 배열을 지원하지 않으며, 다차원 배열은 배열의 배열로서 표현된다. 즉, 배열 안에 또 다른 배열이 들어 있는 자료구조를 취한다. 이와 같은 다차원 배열의 자료 구조로 인하여 다차원 배열의 원소에 접근할 때 여러 번의 레퍼런스를 통해서 이루어진다. 또한 다차원 배열에 접근하는 특별한 명령어가 존재하지 않으므로 여러 개의 바이트코드가 인터프리트 되어 실행된다. 이런 바이트코드가 실행되면서 각각의 배열의 원소에 접근하기 위한 레퍼런스가 이루어지며, 스택 연산을 위한 적재(Load)와 저장(Store)이 수행된다. 따라서 다차원 배열인 경우에 배열의 차원이 크면 클수록 배열에 접근하는 횟수가 많아질수록 자바가상머신의 성능 저하에 영향을 미치게 된다.

본 논문에서는 이런 성능상의 문제점을 개선하기 위해서 다차원 배열을 생성할 때 새로운 자료구조를 제안하며, 이런 자료구조를 통하여 다차원 배열에 접근할 때 여러 개의 바이트코드 대신에 새롭게 정의된 하나의 바이트코드를 사용한다. 그리고 이를 클래스파일에 실제로 적용하고, 간단한 성능 분석을 통하여

성능개선 효과를 확인한다.

본 논문의 구성은 다음과 같다. 2 장에서는 자바 바이트코드에 대해서 알아보고, 3 장에서는 자바의 다차원 배열에 대해서 알아 본 후 본 논문에서 제시한 다차원 배열의 자료구조와 새롭게 정의된 바이트코드에 대해서 알아본다. 4 장에서는 실제적으로 구현을 통하여 어느 정도의 성능 향상이 이루어지는지 분석하며, 5 장에서는 결론과 향후 연구 방향에 대해서 기술한다.

2. 관련연구

2.1 자바 바이트코드

자바 소스 프로그램은 자바 컴파일러를 통하여 클래스 파일로 생성된다. 클래스 파일인 바이트코드들은 8 비트 단위로 읽혀져 해당 플랫폼에 알맞은 형태로 번역된다. 자바 바이트코드는 인터프리트되기 쉬운 장점을 갖고 있지만 자바가상머신이 스택 기반 이므로 값을 재사용할 수 없기 때문에 스택과 로컬 변수에 대한 불필요한 적재와 저장에 관한 바이트코드가 많이 사용되는 단점을 가지고 있다.[1][6]

자바가상머신은 200 개의 바이트코드로 구성되어 있으며 예약(reserved)되어있는 3 개의 바이트코드가 있다. 아직 정의되어 있지 않은 바이트코드는 새롭게 정의되어 추가할 수 있다. 썬사에서는 인터프리터 되어진 바이트코드의 성능 향상을 위해 25 개의 "_quick" 바이트코드를 자바가상머신에 내부적으로 추가하여 정의하였다. [2]

3. 효율적인 다차원 배열의 접근 방법

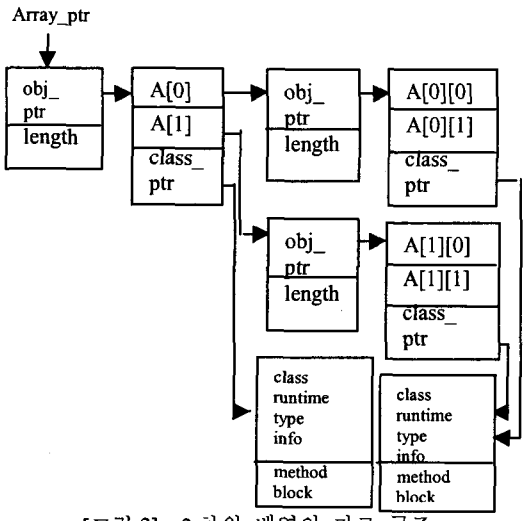
3.1 자바의 다차원 배열

배열이란 같은 타입(Type)의 데이터를 여러 개 저장하기 위한 메모리 공간을 말한다. C/C++ 와는 달리 자바에서는 배열을 하나의 객체로써 처리하며, 자바가상머신에서 오브젝트와 배열은 레퍼런스에 의해서 접근되어진다. 자바에서 배열을 생성하기 위해서는 먼저 선언을 하고 new 연산자를 이용한다. 자바의 new 연산자는 배열을 위한 자료 구조를 실행 시간에 동적으로 생성한다.

특히 자바에서는 다차원 배열을 지원하지 않는다. 다차원 배열은 배열의 배열이라고도 한다. 즉, 다차원 배열은 배열 안에 또 다른 배열이 들어있는 자료구조를 취하기 때문이다. 이차원 배열을 생성하고 배열의 원소에 값을 저장하고 적재하는 프로그램의 예를 [그림 1] 에서 보여주고 있으며, [그림 2]은 이차원 배열이 생성될 때의 자료구조를 보여주고 있다.[3][4]

```
Class test {
   Public static void main(String args[]) {
        Int i = 2; int j = 3;
        Int[][] a = new int[5][5]; // 다차원 배열 생성
        Int b;
        a[i][j] = 1; // 다차원 배열에 값을 저장
        b = a[i][j]; // 다차원 배열에서 적재
}}//test
```

[그림 1] 2 차원 배열의 자바 프로그램



[그림 2] 2 차원 배열의 자료 구조

위의 [그림 2] 에서 2 차원 배열의 원소에 접근하기 위해서 4 번의 레퍼런스를 필요로 한다. 따라서 다차원 배열의 크기가 커질수록 하나의 원소에 접근할 때 발생되는 레퍼런스의 횟수가 많아질수록 자바가상머신의 성능 저하를 유발한다. [5]

아래의 [그림 3]은 위의 [그림 1]의 프로그램이 컴파일되어 수행되는 바이트코드를 보여주고 있다.

PC	ByteCode	
0	5	// iconst_2
1	60	// istore_1
2	6	// iconst_3
3	61	// istore_2
4	8	// iconst_5
5	8	// iconst_5
6	197	// multianewarray
10	58	// astore
12	**25**	**// aload**
13	**27**	**// iload_1**
14	**50**	**// aaload**
15	**28**	**// iload_2**
16	4	// iconst_1
17	79	// iastore
19 ~ 24	**< PC 15-20 의 바이트코드와 동일 >**	
25	46	// iaload
26	54	// istore
27	177	// return

[그림 3] 2 차원 배열을 수행하는 바이트코드

[그림 3]에서 알 수 있듯이 다차원 배열의 원소의 값을 적재하고 저장하는 특별한 바이트코드가 존재하지 않으므로 다차원 배열에 접근하기 위해서는 여러 개의 바이트코드가 인터프리트되어 스택 연산을 통하여 실행된다. 이런 바이트코드의 실행을 통하여 각각

의 배열의 원소에 접근하기 위한 레퍼런스가 이루어지며, 스택 연산을 위한 적재와 저장이 실행된다. 따라서 다차원 배열인 경우에 배열의 차원이 크고, 배열에 접근하는 횟수가 많을수록 자바가상머신의 성능 저하에 영향을 미치게 된다.

3.2 개선된 다차원 배열의 접근 방법

본 논문에서는 다차원 배열의 접근에 관한 성능 개선을 위해서 새로운 배열 자료구조를 제안하고, 이런 자료구조를 통하여 다차원 배열에 접근하는 maload 라는 바이트코드를 정의하였다.

먼저 이를 위해 실행 될 자바 프로그램을 컴파일해서 생성 되어진 클래스 파일을 읽어 들여 다차원 배열의 원소에 접근하려는 바이트코드가 있는지 찾는다. 이런 경우가 발견되면 기존의 바이트코드 대신 새롭게 정의된 바이트코드인 maload 와 오퍼랜드로 변경해 준다. 이때, 배열의 인덱스가 로컬인 경우에만 변경해 준다. 아래 [그림 4]는 다차원 배열의 원소에 접근하는 경우인지 판별하기 위한 알고리즘이다. 그리고 새로 정의된 바이트코드의 실행을 위해 자바가상머신에 추가된 바이트코드에 대한 기능을 정의해주어야 한다.

1. 로컬 변수로부터 배열의 레퍼런스가 적재되는 바이트코드를 찾는다. 이런 바이트코드로는 aload_1 부터 aload_3 그리고 aload 가 있다.
2. 1 번 다음에 오는 바이트코드로써 배열의 인덱스 값을 가져오기 위해서 로컬 변수에 접근하는 바이트코드(iload 그리고 iload_0 ~ iload_3)가 있는지 찾는다.
3. 1 번과 2 번 다음에 오는 바이트코드로써 배열의 레퍼런스를 적재하기 위한 aaload 바이트코드가 오는지 찾는다.
4. 1 번과 2 번 그리고 3 번 다음에 오는 바이트코드로써 접근하려는 배열의 인덱스 값을 가져오기 위해서 로컬 변수에 접근하는 바이트코드가 있는지 찾는다.
5. 4 번까지의 조건을 모두 만족하면 2 차원 배열에 접근하려는 바이트코드이므로 maload 바이트코드와 오퍼랜드로 변경해준다.
6. 만약 4 번까지 만족을 하고 3-4 번을 반복하는 경우는 2 차원 이상인 경우이다. 따라서 계속 3-4 번의 바이트코드를 수행하는지 검사하고, 반복이 끝나면 maload 바이트코드와 오퍼랜드로 변경해 준다.

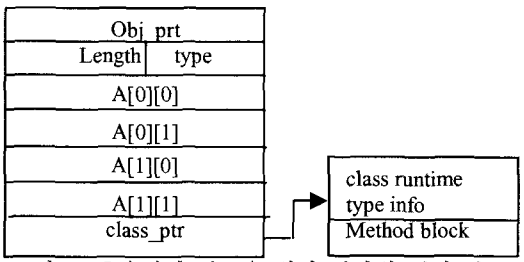
[그림 4] 다차원 배열을 발견하는 알고리즘

이와 같이 새롭게 정의한 maload 바이트코드와 오퍼랜드는 [그림 5]와 같다. 구현상의 이유로 2 차원, 3 차원 그리고 4 차원 이상 배열의 원소에 접근하는 3 개의 바이트코드를 정의하였다. [그림 5]는 4 차원 이상인 경우의 바이트코드의 정의를 보여주고 있다.

- 바이트코드 : maload
- 오퍼랜드 : index, dimension, local_index1, index2..
  - index : 배열에 저장되어 있는 로컬 인덱스
  - *dimension* : 배열의 차원
  - local_index1 : 접근하려는 배열의 원소의 로컬변수의 인덱스

[그림 5] maload 의 바이트코드

위와 같은 새로운 바이트코드를 적용하기 위해 기존의 다차원 배열의 자료구조보다 좀 더 효율적인 자료구조를 제안한다. 자바에서 배열은 오브젝트의 레퍼런스를 통해서 접근 되어지므로 기존의 방식으로 배열을 생성하면 레퍼런스의 횟수가 증가할 뿐만 아니라 이를 수행하기 위한 바이트코드의 수도 많아진다. 따라서 다차원 배열에서 값을 적재하거나 저장할 때 레퍼런스의 횟수를 줄이기 위해서 [그림 6]과 같은 방식으로 이차원 배열을 생성한다.



[그림 6] 효율적인 접근을 위한 이차원 배열 자료구조

이와 같은 다차원 배열의 새로운 자료구조와 추가된 바이트코드가 실제적으로 클래스 파일에 적용되는 것을 알아보기 위해 [그림 3]의 바이트코드를 변경한 것이 [그림 7]이다. [그림 7]에서 multianewarray 바이트코드를 실행 할 때 자바가상머신 내부에서 본 논문에서 제시한 [그림 6]과 같은 구조로 이차원 배열을 생성하며, 배열의 원소에 접근할 때 새로 정의된 바이트코드로 변경된 것을 보여주고 있다.

PC	ByteCode	
0	5	// iconst_2
1	60	// istore_1
2	6	// iconst_3
3	61	// istore_2
4	8	// iconst_5
5	8	// iconst_5
6	197	// multianewarray
10	58	// astore
12	**230**	**// maload 3 1 1**
16	4	// iconst_1
17	79	// iastore
18	**230**	**// maload 3 1 1**
25	46	// iaload
26	54	// istore
27	177	// return

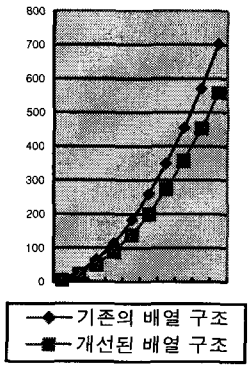
[그림 7] 개선된 바이트코드

위의 [그림 7]에서 보듯이 새로운 배열의 자료구조를 통해서 추가된 하나의 바이트코드를 통해 배열의 원소에 접근하므로 레퍼런스의 횟수를 줄일 수 있을 뿐만 아니라 바이트코드의 길이도 줄일 수 있다. 따라서 효율적인 다차원 배열의 접근을 통하여 자바가상머신의 성능 개선 할 수 있다.

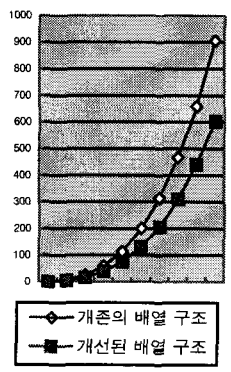
4. 구현 및 성능 평가

실제적인 바이트코드의 구현과 성능 분석을 위해서 테스트 기반은 레드햇 리눅스 7.1 에서 인터프리터 방식으로 수행되는 자바가상머신 kaffe 1.0.6 을 사용 하였다. 테스트 프로그램은 다차원 배열에 많이 접근하는 것을 대상으로 하였으며, 성능 측정은 자바가상머신에서 실행될 때 리눅스 시간으로 측정하였다.

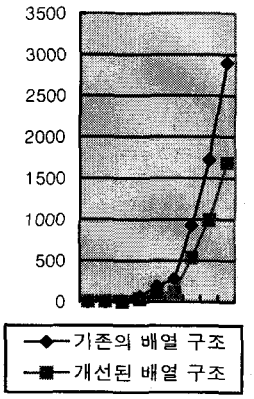
본 논문에서 제시한 방법으로 어느 정도의 성능 개선이 이루어졌는지 측정하기 위하여 다차원 배열에 접근하는 접근 횟수를 증가시키면서 기존의 실행 방법과 새로운 방법 간의 프로그램 실행 시간을 비교하였다. [그림 8]부터 [그림 11]은 각각의 2 차원, 3 차원, 4 차원, 5 차원 배열에서 접근 횟수를 증가시키면서 실행 시간을 측정한 결과이며, 측정한 결과 전체적으로 대략 20 ~ 30 % 정도 성능이 개선 되어지는 것을 볼 수 있었다.



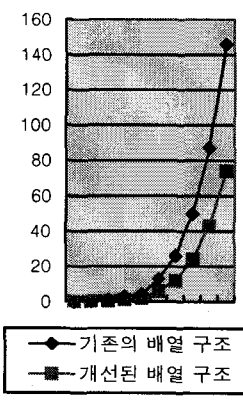
[그림 8] 2 차원 배열 구조



[그림 9] 3 차원 배열 구조



[그림 10] 4 차원 배열 구조



[그림 11] 5 차원 배열 구조

5. 결론 및 향후 연구 방안

본 논문에서는 다차원 배열에 접근하는 경우 비효율적으로 실행되는 점을 개선하기 위한 방법을 제안하였다. 다차원 배열의 자료구조를 새로 제안하였으며, 이런 자료구조를 통하여 다차원 배열을 한번에 접근할 수 있는 바이트코드를 정의하였다. 이의 적용을 위해 실제적인 구현을 통한 성능 평가를 하였으며, 성능 평가를 통하여 기존의 방식보다 20-30% 정도 성능이 개선된 것을 볼 수 있었다. 또한 자바가상머신 내부에서 변경함으로써 비교적 간단한 방법으로 자바가상머신의 성능을 개선할 수 있었다.

향후 연구 과제로는 본 논문에서 제시한 성능 개선 방법 이외에 비효율적으로 수행되는 부분들을 발견해서 자바가상머신의 성능을 개선 할 수 있는 연구가 필요하다.

참고문헌

[1] Tim Lindholm and Frank Yellin, The Java Virtual Machine, Addison Wesley, 1997

[2] Bill Venners, Inside the Java Virtual Machine, McGraw-Hill, 1999

[3] James Gosling, The Java Language Specification, Addison-Wesley, 1996

[4] Joshua Engel, Programming for the Java Virtual Macine, Addision-Wesley, 1999

[5] Safia Belkada, Mitsugu Suzuki, Efficient access for multi-dimension arrays by Java compiler, 電子情報通信研究報告 sofeware science, SS97-46, pp1~8, 1998-01

[6] 황순명, 조창오, 오세만, 자바 바이트코드 최적화기의 설계, 한국정보처리학회 98 추계 학술발표 논문, 5 권 2 호, pp1264~1267, 1998