임베디드 자바가상기계에서 외부 단편화 최소화를 위한 메모리 할당†

김성수°, 양희재
경성대학교 컴퓨터공학과
sskim@conet.ks.ac.kr, hjyang@star.ks.ac.kr

A Memory Allocation Strategy for Minimizing External Fragmentation in Embedded Java Virtual Machine

Sungsu Kim°, Heejae Yang
Dept. of Computer Engineering, Kyungsung University

요 약

자바가상기계의 메모리 할당에서 서로 다른 크기의 메모리 할당과 해제는 힙 영역과 자바 스택 영역에 심각한 외부 단편화를 발생 시킨다. 자바가상기계에서 외부 단편화는 가비지 콜렉션의 발생을 증가 시키고 메모리를 할당하기 위한 메모리 접근이 증가되는 고비용의 동작이 발생하므로 소규모 메모리에서 동작하는 임베디드 자바가상기계에서 성능저하가 발생하게 된다. 본 논문에서는 임베디드 자바가상기계에서 외부 단편화를 최소화 하고 메모리를 효율적으로 관리하기 위한 한 가지 방안으로 고정크기 메모리 할당 방법에 대한 연구이다. 고정크기 메모리 할당 기법은 자바가상기계의 힙 영역에 가장 큰 객체의 크기를 기준으로 할당하고, 자바 스택 영역에 가장 큰 스택 프레임을 기준으로 할당 하도록 하여, 힙 영역과 자바 스택 영역에 외부 단편화를 최소화 하도록 하는 메모리 할당 정책이다. 고정 크기 메모리 할당은 내부 단편화에 따른 메모리 낭비가 발생될 수 있지만, 외부 단편화는 최소화되기 때문에 가비지 콜렉션 발생 횟수를 감소시킬 수 있으며, 회수된 메모리 공간을 재구성 하는 고비용을 제거할 수 있다. 또한 할당 해제된 영역들은 Free-List로 연결되어 메모리 할당을 위한 메모리 접근을 최소화 시키는 장점을 가진다.

1. 서 론

자바가상기계에서 메모리 할당은 클래스의 인스턴스 생성과 메소드 호출시 발생하게 된다. 즉 클래스의 인스턴스가 생성될 때마다 그 인스턴스를 위한 메모리가 힙 영역에 할당되어 지고, 그 인스턴스가 더 이상 사용되지 않으면 할당된 메모리는 가비지 콜렉터에 의해 해제되어 진다. 그리고 메소드가 호출될 때마다 스택 프레임이 자바 스택영역에 할당되어 지고, 메소드에서 복귀하게 되면 할당된 메모리는 해제되어진다[1]. 이런 자바가상기계의 메모리 할당과 해제에서 메모리 관리의 효율적 지원을 위한 메모리 할당 및 해제 알고리즘은 특히 제한된 메모리를 사용하는 임베디드 자바가상기계에서 필수적이다.

자바가상기계에서 서로 다른 크기의 메모리 할당과 해제가 반복되어지면 심각한 외부 단편화 현상이 발생하게 된다. 이런 외부 단편화는 메모리 할당과 해제를 어렵게 하기 때문에 특히 소규모 메모리를 사용하는 임베디드 자바가상기계에 성능저하가 발생할 수 있다[2]. 따라서 본 논문은 임베디드 자바가상기계에서 외부 단편화 현상을 최소화 하기 위한 메모리 관리 방법으로 고정크기 메모리 할당에 대해 연구하였다. 고정크기 메모리 할당은 자바가상기계의 힙 영역에 가장 큰 객체의 크기를 기준으로 할당하고, 자바 스택 영역에 가장 큰 스택 프레임을 기준으로 할당 하도록 하여, 힙 영역과 자바 스택 영역에 외부 단편화를 최소화 하도록 하는 메모리 할당 정책이다.

본 논문에서 연구한 고정 크기 메모리 할당은 내부 단편화에 따른 메모리 낭비가 발생될 수 있지만, 외부 단편화는 최소화되기 때문에 가비지 콜렉션 발생 횟수를 감소시킬 수 있으며, 회수된 메모리 공간을 재구성 하는 고비용을 제거할 수 있고, 또한 할당 해제된 영역들은 Free-List로 연결되어 메모리 할당을 위한 메모리 접근을 최소화 시키는 장점을 가진다.

본 논문의 구성은 다음과 같다. 2절에서는 본 논문의 관련 연구 분야에 대해 소개를 하고, 3절에는 가변 크기 메모리 할당에 대해 간략히 기술 했으며, 4절에는 고정 크기 메모리 할당에 대해 기술 했다. 5절에서는 고정 크기 메모리 할당의 실험 및 성능평가를 했고 마지막 6절에는 결론을 맺는다.

2. 관련연구

고정 크기 메모리 할당을 적용한 자바가상기계로 simpleRTJ와 Jamaica가 있다. simpleRTJ[3, 4]는 제한된 자원을 사용하는 임베디드 시스템에서 자바가상기계의 메모리 절감과 클래스 내부 정보를 효율적으로 접근하기 위해 기존의 클래스 파일을 클래스 이미지 파일로 변환하여 사용한다. 클래스 이미지 파일은 기본적으로 정적 클래스 적재의 특성을 가지고 있기 때문에 자바가상기계가 사용하는 클래스 정보들을 모두 포함하고 있으며, 자바가상기계의 실제 동작에 필요 없는 정보들은 제외되어 있어 클래스 파일보다 상대적으로 소량의 크기를 가진다. 또한 클래스 이미지 파일 내부에는 자바가상기계에서 생성하게 되는 가장 큰 객체 및 스택 프레임의 크기를 명시하고 있어 자바가상기계에서 고정 크기 메모리 할당이 가능하다. 즉 simpleRTJ는 클래스 이미지 파일에 명시된 객체와 스택 프레임의 고정 크기로 메모리를 할당하여 외부 단편화를 제거할 수 있다. 그러나 simpleRTJ는 할당 가능한 메모리 영역들이 Free-List로 연결되어 있지 않기 때문에 메모리 할당을 위한 메모리 접근이 증가되어 동작의 고비용이 발생할 수 있는 단점을 가진다.

† 이 논문은 2003년도 정보통신부 지원 정보통신기초기술연구지원사업에 의해 연구되었음.

Jamaica[2, 5]는 힙 영역의 외부 단편화를 제거하기 위해 메모리 영역을 16Byte 크기의 고정된 블럭으로 나누고 있으며, 각 블럭에 객체의 정보를 저장하게 된다. 만약 객체의 크기가 고정된 블럭보다 크다면, 새로운 블럭을 할당하여 연결 리스트로 연결하게 되고, 그 블럭에 나머지 객체의 정보를 저장하게 된다. 이렇게 할당된 객체는 할당해제 후에도 외부 단편화가 발생하지 않는 장점을 가지지만 객체의 크기가 블럭보다 클 경우 완전한 객체의 정보를 얻기 위해, 연결 리스트로 연결되어 있는 블럭들을 접근해야 하므로 객체정보의 접근효율이 떨어지는 단점을 가진다.

3. 가변 크기 메모리 할당 및 해제

가변 크기 메모리 할당을 적용한 임베디드 자바가상기계인 KVM[6, 7]에서는 객체와 스택 프레임을 동일한 메모리 영역에 할당하고 있다.

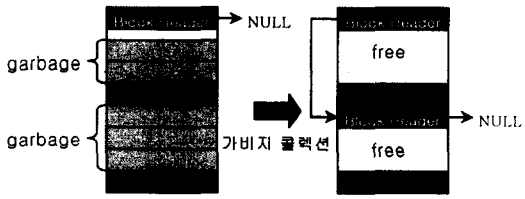

그림 1 가변 크기 메모리 할당 구조

KVM은 그림 1에서와 같이 할당 가능한 블럭 단위로 메모리 영역을 표현하고, 가비지 콜렉터에 의해 서로 크기가 다른 다수개의 블럭들이 연결 리스트로 연결되어 진다. 각 블럭의 크기는 해제 되는 메모리 크기가 서로 다르기 때문에 가비지 콜렉션이 발생할수록 크기가 다른 블럭들로 분할되어지고 그럴수록 심각한 외부 단편화가 발생하게 된다. 즉 가비지 콜렉션의 발생에 따른 고비용의 동작이 발생하게 된다. 또한 KVM에서는 스택 프레임이 가변크기를 가지면서 객체와 같은 영역에 할당되어 지기 때문에 메소드 복귀로 인해 메모리가 해제 되더라도 그 영역은 즉시 사용하지 못 하고, 가비지 콜렉터에 의해 새로운 블럭들로 재구성 되어야만 사용 가능하다. 그 결과 가비지 콜렉션 발생빈도가 높아지고 메모리 사용 효율 또한 떨어지게 되는 단점이 발생하게 된다.

4. 고정 크기 메모리 할당 및 해제

simpleRTJ에서와 같이 고정 크기 메모리 할당을 위해 클래스의 인스턴스 크기와 스택 프레임의 가장 큰 크기를 이미지 파일 변환기[8, 9]를 통해 클래스 이미지 파일 내부에 명시 한다.

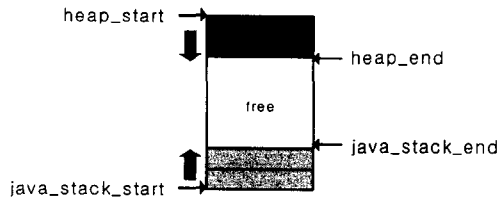

그림 2 고정 크기 메모리 할당 구조

고정 크기 메모리 할당은 객체의 최대 크기와 스택 프레임의 최대 크기가 서로 다르기 때문에 메모리 외부 단편화를 피하기 위해 힙 영역과 자바 스택 영역을 그림 2와 같이 서로 구분한다. 즉 객체는 메모리의 상위부터 하위로 할당하고 스택 프레임은 메모리의 하위에서 상위로 할당하게 된다.

각 영역은 heap_end와 java_stack_end로 서로의 경계를 나타내고 free 영역에 객체와 스택 프레임이 할당되면 마지막 할당된 위치로 그 경계가 이동하게 된다. 만약 heap_end와 java_stack_end가 서로 교차되어 그 경계를 침범하면 가비지 콜렉션과 함께 heap_end 와 java_stack_end의 위치 값을 이동 시켜 free 영역을 확보하게 된다.

4.1 힙 영역의 고정크기 할당 및 해제

힙 영역에 생성될 인스턴스 크기는 필드의 개수에 의해 결정되고, 각 인스턴스가 가변적인 필드의 개수를 가지므로 힙 영역에 할당되는 인스턴스의 크기는 서로 다르다. 즉 고정 크기 메모리 할당을 위해서는 가장 많은 필드를 가지는 인스턴스를 기준으로 힙 영역에 할당하게 된다. 고정 크기로 메모리를 할당하게 되면 외부 단편화에 따른 메모리 낭비는 발생하지 않지만 내부 단편화에 의한 메모리 낭비가 발생될 수 있다. 그렇지만 임베디드 자바가상기계인 CLDC와 simpleRTJ의 핵심 API 클래스의 조사결과 필드의 개수가 0개에서 6개 까지의 필드를 가지면서 분포도 또한 넓게 퍼져 있었기 때문에 내부 단편화에 따른 메모리 낭비는 그리 크지 않다. 특히 0개의 필드를 가지는 예외/오류 클래스들은 정상상태에서 일어나지 않으므로 메모리 사용에는 영향을 미치지 않는다[10, 11].

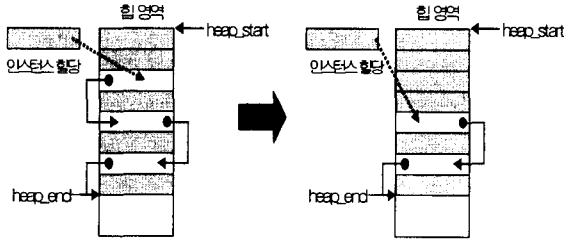

그림 3 힙 영역의 고정 크기 메모리 할당

힙 영역에서 고정 크기 메모리 할당의 가장 큰 특징은 할당될 인스턴스와 해제될 인스턴스의 크기가 서로 같기 때문에 외부 단편화가 최소화 된다는 것이다. 그 결과 그림 3에서와 같이 가비지 콜렉터는 할당 해제된 영역들을 Free-List 로 서로 연결 하면 되기 때문에 메모리를 재구성해야 하는 고비용 동작을 수행하지 않아도 된다. 또한 할당 가능한 영역들을 Free-List로 연결되어 있기 때문에 메모리를 할당을 위한 메모리 접근의 횟수를 최소화 하는 역할을 수행한다.

4.2 자바 스택 영역의 고정 크기 할당 및 해제

메소드가 호출될 때마다 스택 프레임은 자바 스택 영역에 할당된다. 스택 프레임은 지역 변수의 개수와 오퍼랜드 스택 영역의 크기 등에 따라 가변크기를 가진다. 하지만 고정 크기 메모리 할당에서는 가장 큰 스택 프레임을 기준으로 자바 스택 영역에 고정 크기로 할당하기 때문에 외부 단편화를 제거할 수 있다. 이 경우에도 힙 영역에서와 동일하게 내부 단편화에 의한 메모리 낭비가 발생하게 되지만 고정 크기 메모리 할당에서 자바 스택 영역의 스택

프레임은 메소드 호출 종료시 해제되어 즉시 재사용이 가능하기 때문에 그림 4와 같이 스택 프레임 리스트를 독립적으로 가지고 있는 쓰레드의 수가 많지 않고 스택 프레임 리스트가 그리 길지 않다면 내부 단편화에 따른 메모리 손실은 크기 않다.

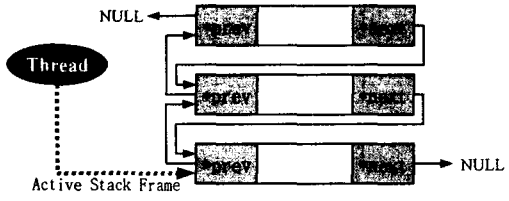

그림 4 스택 프레임 리스트의 구조

고정 크기 메모리 할당의 자바 스택 영역 에서 메소드 호출 종료로 해제된 메모리 영역은 고정 크기를 가지며 메모리 접근의 횟수를 최소화 하기 위해 Free-List로 연결하게 된다. 즉 메소드 호출 종료로 할당 해제된 영역들은 언제든지 사용가능 하면서 메모리의 접근을 최소화 시킨다. 또한 메모리 사용 효율이 증가하기 때문에 가비지 콜렉션 발생 빈도를 줄여 자바가상기계의 고비용 동작을 줄일 수 있게 된다.

5. 실험 및 성능평가

본 절에서는 시뮬레이션을 통해 고정 크기 메모리 할당과 가변 크기 메모리 할당의 성능을 측정해 보았다. 본 실험을 위한 시뮬레이터는 64KB 힙 메모리를 가지는 KVM에서와 같이 힙과 자바 스택을 위해 64KB의 메모리 공간을 가진다.

메모리에 할당되는 객체와 스택 프레임의 크기는 CLDC[8]와 simpleRTJ[9]의 클래스 파일 형식 분석에 의해 가변 크기 메모리 할당에서 객체는 4Byte~60Byte, 스택 프레임은 12Byte~80Byte의 가변 크기를 가지고 고정 크기 메모리 할당에서 객체는 60Byte, 스택 프레임은 80Byte의 고정크기를 가진다. 객체와 스택 프레임의 생성 비율은 객체가 30%, 스택 프레임이 70% 비율로 생성한다.

시뮬레이션은 임베디드 자바가상기계에서 쓰레드의 개수를 최대 30개로 한정 한다고 가정 후 스레드의 개수가 증가함에 따라 고정 크기 메모리 할당과 가변 크기 메모리 할당에서 가비지 콜렉션의 발생 횟수와 메모리 평균 탐색 빈도를 측정하였다.

표 1 시뮬레이션 결과

	가변 크기 메모리 할당		고정 크기 메모리 할당	
쓰레드 개수	가비지 콜렉션 발생 횟수	메모리 평균탐색빈도	가비지 콜렉션 발생 횟수	메모리 평균탐색빈도
1	131	227	94	1
2	265	221	195	1
4	543	209	421	1
8	1,149	191	985	1
16	2,537	180	2,716	1
20	3,352	170	3,321	1
25	4,703	179	5,158	1
30	6,027	187	8,086	1

표 1의 시뮬레이션 결과에서 가변 크기 메모리 할당은 쓰레드 개수가 16개 이하 일 때 외부 단편화에 따른 가비지 콜렉션 발생 횟수가 고정 크기 메모리 할당보다 증가함을 보인다. 하지만 쓰레드 개수가 증가함에 따라 인스턴스의 할당과 메소드의 호출이 증가 하기 때문에 쓰레드 개수가 16개 이상 부터 고정 크기 메모리 할당은 내부 단편화에 의해서 가비지 콜렉션 발생이 가변 크기 메모리 할당보다 증가함을 보인다.

고정 크기 메모리 할당은 외부 단편화가 발생하지 않는다는 특징으로 표 1에서와 같이 메모리 영역을 Free-List로 구축시 메모리 접근 빈도는 쓰레드 증가에 상관없이 일정함을 알 수 있다. 그러나 가변 크기 메모리 할당은 Free-List를 구축하더라도 외부 단편화 영향에 의해 메모리 평균 탐색 빈도가 일정하지 않음을 알 수 있다.

6. 결 론

고정 크기 메모리 할당에서 가비지 콜렉션은 메모리의 재구성을 최소화 하고 있어 고비용의 동작을 수행하지 않는다. 또한 Free-List로 인한 메모리 접근의 최적화는 임베디드 자바가상기계의 실시간성을 보장해 줄 수도 있다. 하지만 쓰레드의 개수가 증가되면서 객체와 메소드 호출이 빈번하게 발생할 때는 내부 단편화에 의한 메모리 손실이 증가하여 가비지 콜렉션 발생이 높아진다는 단점을 가지므로 상대적으로 적은 개수의 쓰레드를 구동시키는 임베디드 자바가상기계에서 높은 효율을 발생 시킨다. 향후 연구로 고정 크기 메모리 할당을 실제로 자바가상기계에 적용하여 그 성능과 함께 실시간성을 어느 정도 보장하는지를 정량적으로 분석해 볼 것이다.

참 고 문 헌

[1] J. Engel, *Programming for the Java Virtual Machine*, Addison-Wesley, 1999.

[2] Fridtjof Siebert, "Eliminating external fragmentation in a non-moving garbage collector for Java", *ACM Compilers Architectures and Synthesis for Embedded Systems (CASES2000)*, 2000

[3] 양회재, "simpleRTJ 자바가상기계에서 클래스 파일의 메모리상 배치", *한국정보과학회 추계학술대회*, 2002. 10.

[4] RTJ Computing, *simpleRTJ: a small footfrint Java VM for embedded and consumer device*, http://www.rtjcom.com

[5] Fridtjof Siebert, "Hard Real-Time Garbage-Collection in the Jamaica Virtual Machine", *IEEE Sixth International Conference on Real-Time Computing Systems and Applications(RTSS'99)*, 1999

[6] Sun microsystem, Connected, *Limited Device Configuration specification Version 1.0a*, 2000

[7] Sun microsystem, Java 2 Platform Micro Edition(J2ME) Technology for Creating Mobile Devices White Paper, 2001

[8] 지정훈, 양회재, "내장형 자바가상기계를 위한 클래스 파일 변환기의 설계 및 구현", *한국정보과학회 춘계학술대회*, 2003. 4.

[9] 김성수, 김세영 양회재, "내장형 자바가상기계를 위한 클래스 이미지 파일의 분석과 비교", *한국정보과학회 춘계학술대회*, 2003. 4.

[10] 양회재, "CLDC 클래스 파일 형식에 대한 분석", 한국*정보처리학회 추계학술대회*, 2003. 10

[11] 양회재, "simpleRTJ 클래스 파일의 형식 분석", *한국해양정보통신학회 학술대회*, 6권, 2호, pp.373-377, 2002. 11.