임베디드 자바 가상머신을 위한 가비지 컬렉터 개발

차창일*, 김형준*, 황규정*, 김상욱*, 이상윤**, 원희선**
*한양대학교 정보통신학부
**한국전자통신연구원
e-mail: cha8680@korea.com

Development of a Garbage Collector for an Embedded Java Virtual Machine

Chang-Il Cha*, Hyung-Jun Kim*, Gyu-Jeong Hwang*,
Sang-Wook Kim*, Sang-Yun Lee**, Hui-Seon One**
*Division of Information and Communications, Hanyang University
**Electronics and Telecommunications Research Institute

요 약

자바 언어는 그 객체지향성, 안전성, 유연성으로 인하여 현재 가장 널리 쓰이는 프로그래밍 언어의 하나가 되었으며, 자바 가상머신이 제공해주는 가비지 컬렉터로 인하여 프로그래머는 메모리 관리에 관한 많은 고민이 줄어들었다. 임베디드 환경에서 역시 자바는 강세를 나타내고 있으며 임베디드 환경의 특성을 반영한 가상 머신과 가비지 컬렉션 기법이 요구되고 있다. 본 논문에서는 힙이라고 불리는 메모리 영역을 크게 젊은 세대와 늙은 세대의 두 부분으로 나누어서 관리하며 각 세대는 그 특성과 요구사항에 적합하도록 각기 다른 기법을 적용한 가비지 컬렉터를 제안한다. 더불어 효과적인 가비지의 식별을 위한 쓰기 장벽과 2중 필터링 기법을 제안하고 있으며, 일반적인 방법으로 회수가 불가능한 순환적 구조의 가비지를 검출하여 회수하기 위한 이중 검사 기법을 제안한다. 제안하는 기법은 임베디드 환경의 요구사항인 객체의 빠른 할당, 동작의 실시간성, 모든 가비지의 회수, 단편화 제거, 높은 지역성 등을 모두 만족한다.

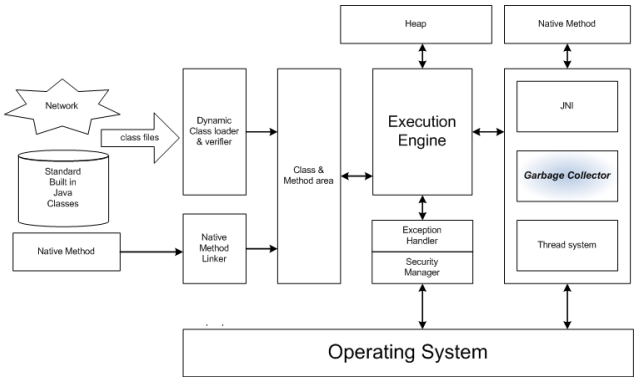
1. 서론

자바(Java) 언어는 그 객체지향성, 안전성, 유연성으로 인하여 현재 가장 널리 쓰이는 프로그래밍 언어의 하나이다. 이러한 자바 언어로 작성된 응용 프로그램이 동작하기 위한 자바 플랫폼은 타깃 시스템의 메모리, CPU 성능, 전력, 응용 프로그램에 적합한 환경 구성을 위해 컨피규레이션(configuration)을 정의한다[SUN05]. 컨피규레이션은 자바 가상머신(Java virtual machine)과 코어 API를 정의하고 있으며 J2EE/J2SE/J2ME로 구분된다. 이 중 J2ME(Java2 platform, micro edition)는 임베디드 환경을 위한 자바 기술로서 네트워크로 연결된 임베디드 혹은 모바일 기기에서 사용된다.

자바 가상머신이란 컴파일된 자바 바이트코드와 실제로 프로그램의 명령어를 실행하는 마이크로프로세서 또는 하드웨어 플랫폼 간에 인터페이스 역할을 담당하는 소프트웨어를 의미한다. 자바 가상머신이 가지는 중요한 모듈 중 하나인 가비지 컬렉터(garbage collector)는 응용 프로그램이 더 이상 사용하지 않는 메모리상의 객체를 찾아서 회수하는 역할을 담당한다. 이 결과, 가비지 컬렉터는 프로그래머에게 메모리 관리에 관한 많은 고민을 줄여주었으며, 응용 프로그램이 보다 안정적으로 동작할 수 있게 해준다. 현재, 한국전자통신연구원에서는 임베디드 자바 가상머신을 개발하고 있으며 그 구조는 그림 1과 같다. 본 논문에서는 임베디드 자바 가상머신을 위한 효과적인 가비지 컬렉터를 제안한다.

제안하는 기법은 힙(heap)이라 불리는 주어진 메모리 영역을 크게 젊은 세대(young generation)와 늙은 세대(old generation)의 두 부분으로 나누어서 관리하는 세대 가비지 컬렉션(generational garbage collection)을 기반으로 한다[Lieb83, Unga84]. 객체는 상대적으로 크기가 작은 젊은 세대에 할당되어 대부분이 젊은 세대에서의 가비지 컬렉션을 통하여 소멸되고, 여전히 사용 중인 객체만이 늙은 세대로 이동하게 된다. 젊은 세대와 늙은 세대는 각 세대의 특성을 반영하기위해 서로 다른 가비지 컬렉션 기법을 사용하게 된다. 젊은 세대에서는 객체의 할당과 소멸이 빈번하게 발생하므로, 본 연구에서는 가비지 컬렉션의 동작에 부하가 적은 세미 스페이스 복사 컬렉터(semi-space copying collector)를 채택한다[Chen70]. 반면, 젊은 세대와 비교하여 그 크기가 큰 늙은 세대에서는 실시간성을 보장하기 위해 보다 작은 영역으로 분할하여 관리하는 점진적인 가비지 컬렉션 기법(incremental garbage collection)을 채택한다. 이와 같이 힙을 여러 개의 작은 영역으로 분할하여 가비지 컬렉션을 수행하기 위해서는 먼저 해당 영역에서 여전히 사용 중인 객체들을 효과적으로 식별하는 방법이 필요하다. 본 논문에서는 이러한 효과적인 객체 식별 문제를 해결하기 위해 임베디드 환경에 적합한 효과적인 쓰기 장벽(write barrier)을 제안한다[Zorn90]. 또한, 일반적인 방법으로 회수가 불가능한 분할된 다수의 영역에 넓게 걸쳐서 존재하는 순환적 구조(cycle structure)의 가비지를 검출하고 회수

하는 기법을 제안한다. 제안하는 가비지 컬렉션 기법은 임베디드 기기의 물리적 제약사항으로 인해 요구되는 빠른 할당, 동작의 실시간성, 모든 가비지의 회수, 단편화 제거, 높은 지역성 등을 모두 만족한다.



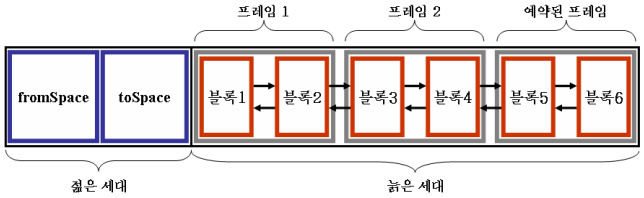
(그림 1) 임베디드 자바 가상머신.

2. 제안하는 가비지 컬렉터

2.1 가비지 컬렉션 구조와 알고리즘

본 논문에서 제안하는 가비지 컬렉터는 그림 2에 나타난 바와 같이 힙을 젊은 세대와 늙은 세대의 비대칭적인 두 개의 세대로 분할하여 관리하는 세대 가비지 컬렉션에 기반하고 있다[Lieb83, Unga84]. 이러한 세대 가비지 컬렉션은 대다수의 객체가 할당된 후 짧은 시간 동안만 사용된다는 특성으로 인해 많은 수의 가비지 객체가 젊은 세대에서 회수된다[Barr93, Blac04, Zorn89]. 따라서 상대적으로 크기가 작은 젊은 세대에서의 가비지 컬렉션이 보다 빈번하게 발생하여 소수의 객체만이 늙은 세대로 이동하므로 가비지 컬렉션에 의한 지연시간이 적고 전체 프로그램의 실행시간이 감소한다는 장점을 갖는다[Jone96].

본 연구에서는 젊은 세대 가비지 컬렉션 기법으로 구현이 쉽고 동작에 부하가 적은 세미 스페이스 복사 가비지 컬렉션을 채택한다[Chen70]. 반면, 늙은 세대는 영역의 크기가 매우 크기 때문에 전 영역에 걸쳐 가비지 컬렉터를 수행하면 지연 시간이 길어 실시간 요구사항을 만족시킬 수 없다. 따라서 본 연구에서는 늙은 세대의 영역을 분할하여 복사 가비지 컬렉션을 부분적으로 수행하되, 이를 점진적으로 수행함으로써 실시간 요구사항을 충족시키고 더불어 복사 가비지 컬렉터의 장점을 모두 수용할 수 있는 기법을 제안한다.



(그림 2) 힙의 구조.

그림 2에 나타난 바와 같이 늙은 세대는 균등한 크기의 블록이라는 단위로 분할되어 있고, 이 블록들은 이중 연결 리스트로 상호 연결되어 있다. 블록은 객체 할당의 단위가 된다. 이 블록들을 여러 개 묶어서 프레임을 형성하는데 이 프레임은 가비지 컬렉션의 단위가 된다. 힙이 초기화되는 시점에서 블록의 크기와 프레임의 크기는 환경 변수를 통해 결정할 수 있고, 이로써 가비지 컬렉터의 수행으로 인한 지연 시간의 조절이 가능하다. 각 블록은 할당이 발생한 순서대로 나이를 가진다. 전체 프레임 중 한 프레임은 객체 할당을 위하여 사용하지 않고 가비지 컬렉션 수행을 위한 예비 공간으로 보존된다.

늙은 세대의 가비지 컬렉션은 젊은 세대로부터 일정한 나이에 도달한 객체를 늙은 세대로 이동할 때 그 공간이 부족하면 수행된다. 컬렉션의 대상으로 선정되는 프레임은 가장 나이가 많은 프레임이다. 그 이유는 나이가 가장 많은 프레임이 가장 많은 가비지 객체를 포함하고 있을 가능성이 높기 때문이다. 가비지 컬렉터는 대상이 되는 프레임에서 루트 셋(root set)으로부터 도달 가능한 객체를 미리 예약된 빈 프레임으로 복사시킨다. 루트 셋은 수행 중인 프로그램에서 참조하고 있는 살아있는 객체들에 대한 참조 값들의 집합이며, 일반적으로 스택(stack)이나 전역 변수 내에 저장된다. 모든 도달 가능한 객체들을 옮긴 후 프레임은 비워지고 다음 가비지 컬렉션을 위한 예비 공간으로 사용된다. 복사가 완료된 프레임의 블록의 나이는 가장 어린 나이로 설정되어 젊은 세대로부터의 다음 할당을 기다린다. 한 프레임의 가비지 컬렉션이 완료되면 멈춰있던 응용 프로그램의 수행이 재개된다.

2.2 논의 사항

제안하는 기법에서 젊은 세대와 늙은 세대 모두에서 객체들은 연속적으로 할당된다. 이는 객체의 할당 위치 선정을 위한 별도의 연산이 없어 빠른 할당이 가능하다. 함께 할당된 객체들은 함께 엑세스될 가능성이 높으므로, 이러한 객체의 연속적인 할당은 참조의 지역성을 높인다[Blac04]. 젊은 세대에서는 단편화 현상이 없이 객체가 할당되나, 늙은 세대에서는 할당 요청된 객체의 크기가 현재 할당 중인 블록의 남은 여유 공간보다 큰 경우에는 새로운 블록에 할당된다. 따라서 블록 간에 부분적인 단편화 현상이 발생하나 그 크기는 크지 않다. 가장 중요한 실시간 요구 조건에 대하여 젊은 세대는 그 크기가 크지 않고, 또한 많은 수의 객체가 젊은 세대 내에서 소멸되기 때문에 가비지 컬렉션의 지연시간이 실시간성을 만족시킨다. 상대적으로 크기가 큰 늙은 세대는 보다 작은 프레임 단위로 분할하여 가비지 컬렉션을 수행하기 때문에 실시간성을 보장할 수 있다. 이 때, 블록의 크기와 프레임의 크기는 힙이 초기화되는 시점에서 환경 변수를 통해 결정한다. 또한, 가장 오래된 프레임부터 가비지 컬렉션을 수행하기 때문에 수행 대상인 프레임 내에서 많은 객체가 소멸하며, 이동 되는 객체의 수가 적기 때문에 가비지 컬렉션의 수행 속도가 빠르다.

3. 쓰기 장벽

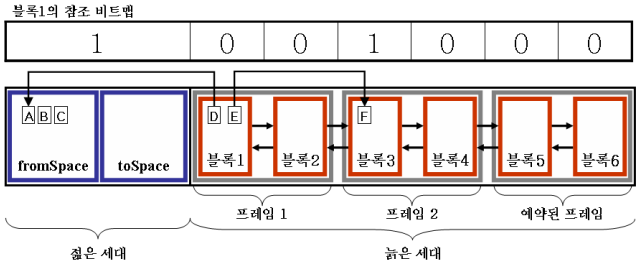
3.1 정의

쓰기 장벽이란 메모리상의 특정 영역에 쓰기 연산을 수행할 때, 이를 감지하여 특정 메모리 관리 연산을 수행시키는 일련의 메커니즘이다[Zorn90]. 쓰기 장벽이 필요한 이유는 본 연구에서 제안하는 가비지 컬렉션이 동작할 때 힙의 전체 영역을 대상으로 하지 않고 젊은 세대와 늙은 세대의 한 프레임이라는 일부 영역만을 대상으로 하기 때문이다. 즉, 힙의 한 영역만을 가비지 컬렉션하기 위해서는 그 부분에 속해있는 모든 도달 가능한 객체들만을 정확히 식별해낼 수 있어야 한다.

3.2 해결 방법

본 연구에서는 가비지 컬렉션 시 도달 가능한 객체를 빠르게 검출하기 위해서 서로 다른 세대나 서로 다른 블록 내 객체들 간

의 참조가 발생하면 쓰기 장벽을 이용하여 이러한 정보를 프로그램이 실행하는 동안 별도의 영역에 기록해둔다. 이 후 가비지 컬렉터는 이 별도의 영역만을 검사하여 가비지 컬렉션을 하고자 하는 대상 영역 내 모든 도달 가능한 객체를 식별할 수 있다.



(그림 3) 쓰기 장벽과 객체 참조 비트맵

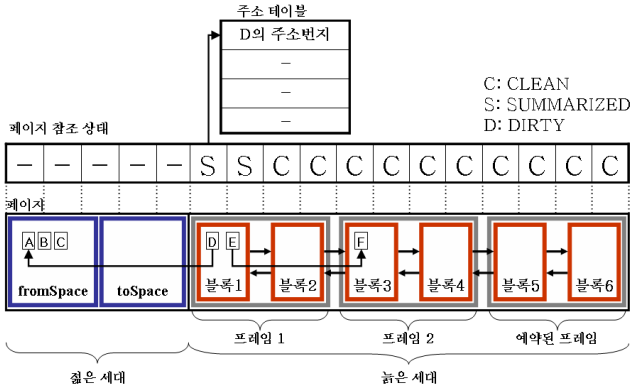
제안하는 기법은 한 영역의 도달 가능한 객체를 식별하기 위해 이 단계 필터링을 사용한다. 제 1차 필터는 가비지 컬렉션의 대상이 각 블록에 대하여 그 블록내의 객체들이 참조하는 블록들을 식별한다. 제 1차 필터링을 위하여 각 블록의 헤더는 그림 3과 같이 참조 비트맵을 가진다. 비트맵 내의 각 비트는 힙 내의 블록과 1대1 대응되며, 따라서 각 블록 내의 비트맵은 블록 수만큼의 비트수를 갖는다. 한 블록A내의 한 객체가 다른 세대 혹은 다른 블록B 내의 객체를 참조하게 되면 블록A의 참조 비트맵은 참조가 되는 세대 또는 블록B와 대응되는 비트를 1로 설정한다. 가비지 컬렉터는 블록 헤더의 참조 비트맵을 검사함으로서 현재의 블록을 참조하는 객체를 가지는 모든 블록들을 검출해낼 수 있다. 그림 3의 예에서, 블록1은 블록3과 젊은 세대 내의 객체들을 참조함을 볼 수 있다. 참조 비트맵은 한 블록A에서 다른 블록B로의 참조가 발생하면, 블록A내의 비트맵에서 블록B에 해당하는 비트가 즉시 1로 설정된다. 그러나 블록A의 참조 비트맵의 초기화는 블록A를 대상으로 가비지 컬렉션이 수행될 때 이루어진다.

제 1차 필터에 의해 추적해야 할 블록이 결정되더라도 하나의 블록에는 많은 수의 객체가 존재할 수 있으므로 이 객체들을 모두 추적하는 것은 역시 상당한 시간이 소요된다. 따라서 블록 내에서 다른 블록을 참조하는 객체를 구체적으로 식별하기 위해 제안된 기법에서는 그림 4와 같은 제 2차 필터를 둔다. 제 2차 필터는 힙을 페이지라는 물리적 단위로 다시 세분화한 후 각 페이지마다 참조 상태를 저장하여 참조 상태에 따라 객체 추적을 달리하는 방법이다[1]. 각 페이지의 참조 상태 정보는 CLEAN, DIRTY, SUMMARIZE 중의 하나이다. 참조 상태가 CLEAN 이면 그 페이지 내에서는 다른 세대 혹은 블록을 참조하는 객체가 전혀 없음을 뜻하며, 그 페이지는 가비지 컬렉션 시 내부의 객체들을 추적하지 않아도 된다. DIRTY 이면 해당 페이지 내에 다른 세대 또는 블록을 참조하는 객체가 많음을 뜻하며 해당 페이지의 모든 객체들에 대하여 참조를 추적해야 한다. 참조 상태가 SUMMARIZE 이면 다른 세대 또는 블록으로의 참조가 있지만 특정 수 이하임을 뜻한다. 즉, 다른 세대 또는 블록을 참조하는 객체의 수가 많지 않으므로, 해당 페이지의 객체를 전부 추적하지 않도록 다른 세대 또는 블록을 참조하는 객체의 주소를 별도의 주소 테이블에 저장한다. 따라서 참조 상태가 SUMMARIZED 이면 이 주소 테이블에 기록된 객체만을 추적해보면 된다.

가비지 컬렉터는 제 1차 필터를 통해 해당 세대 또는 프레임이 어떤 블록에서 참조 되는지를 검사한 후, 참조가 있는 블록에 대해서는 제 2차 필터를 통해 정확히 어떤 객체가 도달 가능한지 식별할 수 있다.

3.3 논의 사항

제안하는 기법은 가비지 컬렉션의 대상이 되는 블록의 도달 가능한 객체를 찾기 위하여 전체 힙에 존재하는 객체 추적의 부하를 응용 프로그램의 실행 중에 분산시킴으로서 가비지 컬렉션의 지연 시간을 줄인다. 이 단계 필터링에 필요한 정보를 저장하기 위한 추가적인 공간이 소요된다. 제 1차 필터를 위해서는 (전체 블록의 수+1)2/8 바이트가 필요하며, 제 2차 필터를 위해서는 (페이지의 수 * 페이지 참조 상태 정보의 크기) + (SUMMARIZED 페이지를 위한 주소 테이블의 크기) 만큼의 공간이 필요하다. 제안하는 기법은 메모리가 제한적인 임베디드 환경을 대상으로 하므로 전체 블록과 페이지의 수가 많지 않다. 따라서 추가 요구 공간은 제한적인 메모리 내에서도 충분히 수용할 수 있을 정도로 작다.



(그림 4) 쓰기 장벽과 페이지 참조 상태.

4. 순환적 구조의 가비지에 대한 검출과 회수

4.1 정의

순환적 구조(cycle structure)의 가비지란 참조를 통해 순환적 구조를 가지는 객체들이 루트 셋으로부터의 참조가 없어지면서 가비지가 되는 것을 의미한다[Jone96]. 여기서 순환적 구조란 한 객체에서 그 객체가 참조하는 다른 객체를 재귀적으로 추적해가면 자기 자신으로 돌아오는 구조를 말한다.

이러한 순환적 구조의 가비지가 두 개 이상의 여러 프레임에 걸쳐서 존재할 경우 이러한 가비지는 회수가 불가능해진다. 그 이유는 제안하는 기법에서는 가비지 컬렉터가 하나의 영역만을 대상으로 하며, 도달 가능한 객체의 식별은 앞 장에서 설명한 쓰기 장벽을 통한 기록에 의존하기 때문이다. 즉, 가비지 컬렉션 대상인 프레임의 외부 블록 내의 가비지로부터 참조되어지는 객체는 여전히 도달가능하다고 판단되어 살아있는 객체로 간주되는 문제가 발생한다.

4.2 해결 방법

순환적 구조의 가비지는 힙의 어느 공간에서든 발생할 수 있다. 그러나 젊은 세대에까지 걸쳐서 존재하는 순환적 구조의 가비지는 젊은 세대에서의 가비지 컬렉션을 통해 모두 늙은 세대로 이동된다. 따라서 늙은 세대에서의 가비지 컬렉션에서 순환적 구조의 가비지를 검출하고 회수하여야 한다.

* 제 2차 필터는 CVM[SUN05a]을 위한 쓰기 장벽 기법으로서 채택하고 있다.

제안하는 기법에서는 먼저 순환적 구조를 가지는 가비지를 식별하기 위해 모든 프레임에 대한 가비지 컬렉션이 완료된 시점에서 주기적으로 전체 힙 영역을 대상으로 모든 도달 가능한 객체들을 추적한다. 객체를 추적하면서 도달 가능한 객체는 그 객체가 가지는 헤더 내 하나의 비트에 표기를 한다.

이제 첫 번째 프레임에 대한 가비지 컬렉션이 실행되면 가비지 컬렉터는 쓰기 장벽을 통해 기록된 정보로부터 도달 가능한 객체를 식별하는데, 이 때 그 객체의 헤더를 조사하여 표기가 되어 있는지를 검사한다.

만약 한 객체가 다른 블록의 객체로부터 참조되어지지만 헤더의 표기가 없다면 순환적 구조의 가비지에 해당되므로 예약된 프레임으로 이동이 되지 않는다. 이러한 이중 검사를 통해 도달 가능하다고 식별된 객체는 예약된 프레임으로 이동하면서 헤더의 표기를 초기화 한다.

4.3 논의 사항

순환적 구조의 가비지가 힙 상에 오래도록 존재하는 것은 메모리의 낭비를 유발시킬 뿐만 아니라, 할당 공간 확보를 위한 가비지 컬렉션을 보다 자주 수행시킴으로서 시간적 부하도 유발시킨다. 이러한 순환적 구조의 가비지를 검출하기 위하여 제안하는 기법도 수행상의 부하가 존재한다.

그러나 전체 힙을 대상으로 가비지 컬렉션을 하는 것이 아니라 단지 도달 가능한 객체들만을 식별하는 과정만 수행하므로 그 부하는 상대적으로 작으며, 따라서 제안된 가비지 컬렉터의 실시간성의 보장을 저해하지는 않는다.

5. 결론

본 논문에서는 임베디드 환경의 제약 사항을 고려한 가비지 컬렉터를 제안한다. 본 논문에서 제안하는 가비지 컬렉션 기법은 힙을 크게 젊은 세대와 늙은 세대의 두 부분으로 나누어서 관리하는 세대 가비지 컬렉션에 기반 하고 있으며, 젊은 세대에 세미스페이스 가비지 컬렉터를, 늙은 세대에 점진적인 가비지 컬렉터라는 각기 다른 기법의 적용한다. 제안하는 기법은 연속적인 할당으로 인하여 빠른 할당과 객체 참조의 지역성을 만족하며, 젊은 세대의 크기 및 블록의 크기와 하나의 프레임이 가지는 블록 수를 변경함으로서 지연 시간이 조절가능하고 따라서 실시간성을 보장한다. 또한, 도달 가능한 객체 식별을 위한 효과적인 쓰기 장벽과 이 단계 필터링 기법을 제안하고 있으며, 순환적 구조의 가비지를 검출하여 회수하기 위한 이중 검사 기법을 제안한다. 이로 인하여 모든 가비지의 정확한 회수가 가능하다. 향후 연구로는 제안하는 기법의 최적의 효율을 위한 각종 파라미터의 설정에 대한 실험을 수행하고, 제안하는 기법과 기존의 기법들에 대한 성능 평가를 수행할 예정이다.

감사의 글

본 연구는 정보통신부 및 정보통신연구진흥원의 대학 IT연구센터 지원사업의 부분적인 지원을 받았음. (IITA-2005-C1090-0502-0009)

참고문헌

[Barr93] D. A. Barrett and B. G. Zorn, Using lifetime predictors to improve memory allocation performance, In *Proceedings of SIGPLAN Conference on Programming Languages Design and Implementation(PLDI),* Vol. 24, No. 7, pp. 187-196, June 1993.

[Blac04] S. M. Blackburn, P. Cheng, and K. S. McKinley, Myths and reality: The performance impact of garbage collection, In *Proceedings of International Conference on Measurement and Modeling of Computer Systems,* pp. 25-36, June 2004.

[Chen70] C. J. Cheney, A non-recursive list compacting algorithm, *Communications of the ACM,* Vol. 13, No. 11, pp. 677-678, Nov. 1970.

[Chen02] G. Chen et al., Tuning garbage collection in an embedded Java environment, In *Proceedings of International Symposium on High-Performance Computer Architecture*, pp. 92-103, Feb. 2002.

[SUN05] Sun Microsystems, *Java2 Platform, Micro Edition, Connected Device Configuration(CDC),* http://java.sun.com/products/cdc/index.jsp 2005.

[SUN05a] Sun Microsystems, *Connected Device Configuration(CDC) HotSpot Implementation,* http://java.sun.com/j2me/docs/cdc_hotspotds.pdf 2005.

[Jone96] R. E. Jones and R. Lines, Garbage collection: Algorithms for automatic dynamic memory management, Wiley, Chichester, July 1996.

[Lieb83] H. Lieberman and C. E. Hewitt, A real-time garbage collector based on the lifetimes of objects, *Communications of the ACM,* Vol. 26, No. 6, pp. 419-429, 1983.

[Unga84] D. M. Ungar, Generation scavenging: A non-disruptive high performance storage reclamation algorithm," *ACM SIGPLAN Notices,* Vol. 19, No. 5, pp. 157-167, Apr. 1984.

[Zorn89] B. G. Zorn, Comparative performance evaluation of garbage collection algorithms, *PhD thesis, University of California at Berkeley,* Technical Report UCB/CSD 89/544, Mar. 1989.

[Zorn90] B. Zorn, Barrier methods for garbage *collection, University of Colorado,* Technical Report CU-CS-494-90, Nov. 1990.