공학석사 학위논문

자바 플랫폼에서 세대별 가비지 컬렉터의 효율적인 활용 방안

지도교수 윤 성 대

이 논문을 공학석사 학위논문으로 제출함.

2004년 8월

부경대학교 산업대학원

전 산 정 보 전 공

이 은 화

이은화의 공학석사 학위논문을 인준함

2004년 8월 31일

주 심 공학박사 여 정 모 

위 원 공학박사 김 창 수 

위 원 이학박사 윤 성 대 

<차례>

<그림차례>

<표차례>

An Efficient Using Method Of Generation Garbage Collector On Java Platform

Eun-Hwa Lee

Dept. of Computer and Information Graduate School of Pukyong National University

Abstract

A garbage collector does memory management automatically on Java. A garbage collector has an advantage that it can increase the productivity of software by reusing heap space which is occupied by the objects no longer used by programs. There are various methods such as reference counting, mark-sweep, copy, generation and compaction in the garbage collector.

In this paper, we have shown the performance measurement on the efficient garbage collector in the java platform using generation garbage collector. An application program having short life-cycled object and a long life-cycle one are measured. When the larger heap size is the less frequency of garbage collection and the more the execution time increase in the application program having short life-cycled object.

The larger heap size is the less frequency of garbage collection and the more the execution time decrease in the application program having long life-cycled object. Adjusted the size of young generation in the same heap size, the larger young generation size is the less frequency of garbage collection and the more the execution time increase in the application program having short life-cycled object. The larger young generation size is the less frequency of garbage collection and the more the execution time decrease in the application program having long life cycled object.

The algorithm for generation garbage collector adjusts the size of young generation, the frequency of garbage will be reduced. As a result, this can be a method to reduce the overhead by garbage collection. In this paper, we can be shown effective in the CVM of CDC which uses small memory and generation garbage collector.

1. 서 론

하드웨어 플랫폼에 독립적이고, 이식성을 보장하는 자바의 기술은 서버 급에서 임베디드 시스템 및 정보 가전 등의 다양한 종류의 소형 정보기기들이 등장하고 이러한 기기들은 모두 인터넷에 연결될 것을 예상됨에 따라 썬 마이크로 시스템즈는 한 가지 버전의 자바로서는 모든 정보 기기를 지원할 수 없다는 판단 아래 자바의 런타임 플랫폼을 컴퓨팅 환경에 따라 3가지 버전인 J2SE(Java 2 Standard Edition), J2EE(Java 2 Enterprise Edition), J2ME(Java 2 Micro Edition)로 나누었다.

J2SE는 데스크톱 PC급의 컴퓨터를 목적으로, J2EE는 엔터프라이즈 버전으로 서버급의 컴퓨터를 목적으로 J2ME는 페이저, 셀룰러폰, 스크린폰, 디지털 셋톱박스, 카 네비게이션 시스템을 포함한 광범위한 일반 소비자 기기를 목적으로 설정한 자바 런타임 환경이다[8].

J2ME는 소비자 임베디드 장비에서의 자바 프로그램을 위한 것으로 컨피규레이션(Configuration)과 프로파일(Profile)로 구성된다. J2ME는 2개의 컨피규레이션(Configuration)을 가진다.

첫째로 CLDC(Connected Limited Device Configuration)는 한정된 규격을 갖는 디바이스에서 J2ME의 구현에 필요한 가장 기본적인 라이브러리 세트와 자바 가상 머신을 정의한다. CLDC는 9.6kbytes 이하의 느린 네트워크 연결, 배터리 공급의 제한된 전원,128KB 이상의 ROM, 32KB 이상의 RAM를 갖는 디바이스를 목적으로 한 규격으로서 KVM(Kilo Virtual Machine)과 함께 160KB의 메모리와 16비트 프로세

서를 지원하는 낮은 시스템 사양을 갖는 단말기를 목적으로 한 버전이다.

둘째로 CDC(Connected Device Configuration)는 J2SE의 스트립 다운 버전이며 CLDC 클래스들이 포함되어 있다. 따라서 CDC는 CLDC의 상위에 만들어졌으며 CLDC 디바이스 용으로 개발한 애플리케이션 또한 CDC 디바이스에서 실행된다. CDC는 CVM(Card Virtual Machine)[8]과 함께 사용되며 자바 플랫폼을 위해 2MB 이상의 메모리와 32비트 프로세서를 갖고 네트워크에 연결되는 단말을 목적으로 한 자바 버전이다.

자바는 특히 메모리 관리를 가비지 콜렉터로 자동으로 관리해 준다. 가비지 콜렉터는 프로그램에 의해 더 이상 참조되지 않는 객체들을 찾아서 그 객체들이 차지하고 있던 힙(heap) 공간을 재 사용할 수 있게 함으로써 프로그래머로 하여금 메모리에 대한 부담을 전혀 가지지 않으므로 소프트웨어 생산성을 향상시킬 수 있는 장점을 가지고 있다. 그러나 실행시간 시스템은 어플리케이션이 생성한 동적 메모리 또는 객체들에 대한 수거 여부를 실행 시간에 판단해야 하기 때문에 오버헤드가 발생되는 문제점이 있다[5,6].

자동적인 가비지 콜렉터의 종류는 참조 계수, 마크-회수, 복사, 세대별, 압축 방법이 있다.

마크-회수 방법을 사용하는 자바 플랫폼으로는 Personal JAVA와 Kaffe VM, KVM이고, 세대별 방법을 사용하는 자바 플랫폼은 Hotspot, CVM 이다.

본 논문에서는 세대별 방법을 선택하고 있는 자바 플랫폼에서의 가비지 콜렉션을 특징을 고찰해서 힙의 크기에 따른 성능 측정과 동일한 힙

의 크기에서 Young generation의 크기 조정에 따른 성능 측정을 살펴서, 자바 플랫폼에서 객체의 생명 주기가 짧은 응용프로그램과 객체의 생명 주기가 긴 응용프로그램에 각 각 적용하여서 가비지 컬렉션의 회수와 실행시간의 성능을 향상시킬 수 있는 효율적인 방법을 선택할 수 있는 방법을 제시한다.

본 논문의 구성은 2장은 본 논문의 관련 연구에 대해서 소개하고, 3장은 세대별 가비지 컬렉터의 가비지 회수에 대해서 기술했으며, 4장은 성능측정 및 분석을 하고, 마지막 5장에 결론을 맺는다.

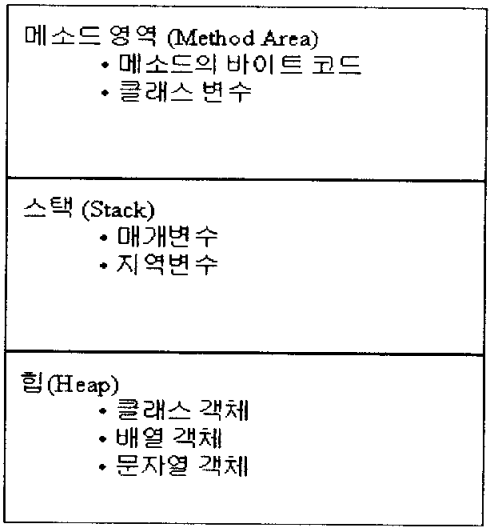
2. 관련연구

이 장에서는 관련 연구로 자바 가상 머신의 메모리 모델과 가비지 컬렉터에 대한 수거대상, 호출시기와 자동적인 가비지 컬렉터의 종류에 대해 설명한다.

2.1 자바 가상 머신의 메모리 모델

자바에서 메소드, 객체, 또는 각종 변수들을 저장하기 위한 메모리 공간을 정의하고 있는 메모리 모델을 제공해주고 있다. 자바에서 제공해 주고 있는 메모리 모델은 그림1과 같이 세 개의 논리적인 영역으로 나누어진다. 메소드 영역(method area)은 자바 프로그램을 구성하고 있는 클래스의 메소드에 대한 바이트코드 또는 클래스의 전역 변수인 클래스 변수 등이 위치한다. 스택(stack)영역은 메소드가 호출되어 수행될 때, 메소드가 갖는 매개변수, 메소드 내에 선언된 지역변수, 임시변수, 반환치 등을 저장하기 위해 스택이라는 저장공간을 이용한다. 메소드의 매개변수, 지역변수, 임시 변수, 반환치 등은 메소드가 호출되어 수행될 때만 필요하므로, 메소드 호출과 함께 이들을 저장하기 위한 메모리 공간(스택 프레임)이 할당되고, 메소드의 수행을 마치고 호출한 쪽으로 되돌아 갈 때, 더 이상 필요하지 않으므로 할당되었던 스택프레임은 다시 시스템에 반환된다. 힙(heap)영역은 자바에서 객체를 저장할 때 사용하는 메모리 공간으로 주로 연산자 new를 이용하여 생성된 객체들이 저

장되는 공간이다.



(그림1) 자바 가상머신의 메모리 모델

```
class MemoryModelTest {
        static int x=0;
        public static void main(String args[]) {
            int a=10, b=20, c;
                c = add(a, b);
        }
        static int add(int a, int b) {
            return(a + b);
        }
    }
```

(그림2) 자바 메모리 프로그램의 예

그림2의 자바프로그램에 선언된 변수 또는 객체들 각각이 위치하는 메모리 공간의 구조는 메소드 영역에 main 메소드와 sum 메소드의 바이트 코드와 x=0값이 저장되고, 스택영역에는 args 매개변수의 주소값, main 프레임의 a=10, b=20, c=0값과 sum 프레임의 a=10, b=20값이 저장되고, 힙 영역에는 args의 매개 변수값인 String객체가 저장된다[14].

2.2 가비지 컬렉터의 수거대상 및 호출시기

자바는 프로그램의 메모리를 관리한다. 자바가상머신은 메모리를 free 시키고, 메모리를 필요한 양만큼 할당시키고 하는 등의 일을 한다. 자바가상머신은 불필요한 객체 수거를 가비지 컬렉터에 의해서 이루어진다. 가비지(garbage)의 대상은 heap 영역에 할당되어 있는 객체들 중에서 더 이상 참조되지 않는 것을 수거한다. 가비지 컬렉터의 시기는 해당 객체를 가리키는 참조변수가 더 이상 존재하지 않는다고 판단될 경우 가비지 컬렉터에 의해 finalize() 메소드가 호출된다. 자바에서는 시간 알고리즘에 의하여 호출되어 수행되고, 메모리 사용에 대한 요구가 현재 남아있는 메모리의 크기보다 큰 경우 호출된다[10].

메모리 관리를 자동화하는데 따른 오버헤드가 발생하고 있는데, 이것은 가비지 컬렉터가 실행시간에 살아 있는 객체와 가비지 객체를 구분해야 하기 때문에 발생되는 것이며, 이러한 오버헤드가 사용자와 시스템간에 빈번한 상호 작용이 발생하는 환경에서는 가비지 컬렉터의 사용이 가장 큰 저해 요인이 되고 있다[9].

2.3 참조 계수(reference counting) 방법

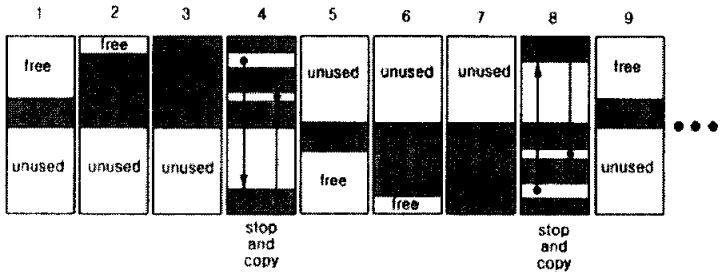
힙(heap)안에 각 객체마다 참조계수(reference counting)가 유지된다. 어떤 객체가 처음 생겨서 그 객체를 참조하는 변수에 배정하며 참조계수는 1이 되고, 다른 변수가 그 객체를 참조하면 참조계수를 1 증가시킨다. 객체의 참조가 범위를 벗어나거나, 변수에 다른 값이 배정되면 참조계수 1 감소하도록 한다. 그래서 참조계수가 0인 객체는 가비지 컬렉션의 대상 객체가 된다. 어떤 객체가 가비지 컬렉션 될 때 그 객체에서 참조하던 모든 객체의 참조계수는 1씩 감소되는 방법이다. 장점으로는 프로그램 실행 때 함께 섞여서 적은 시간에 실행되므로 오랫동안 기다리면 안 되는 실시간(realtime) 환경에 적당하다. 단점으로는 참조계수를 증가시키고 감소시키는데 오버헤드가 크고, 메모리 단편화 문제가 발생한다[4,6].

2.4 마크-회수(mark-sweep) 방법

마크-회수(mark-sweep)방법은 root로부터 시작하여 참조 트리를 순회하며 살아있는 객체와 가비지에 대해서 마크하는 마크단계와 다음 단계로 마크되지 않은 객체를 반환하여 그 객체가 차지하고 있던 힙 영역을 프로그램으로 다시 사용하는 회수단계로 이루어진다. 알고리즘 포인터 조작이 필요 없고, 작은 메모리 공간을 차지하지만 메모리 단편화가 발생한다. Personal Java와 Kaffe VM, KVM에서 사용된다[2,3,5].

2.5 복사(copying) 방법

복사(copying)방법은 메모리를 두 개로 나누어 한쪽에만 할당을 계속하고, 할당할 수 없을 때 나머지 한쪽에 살아있는 객체를 이동하고, 할당을 하는 방법이다. 그림3에서 1은 아래의 반은 사용되지 않은 영역(unused space)이다. 위의 반은 부분적으로 객체로 채워져(색칠된 부분) 있다. 2, 3 힙의 위쪽부분이 점점 객체들로 채워져 가득 차게 되는 모습을 나타내고, 4는 이 때 가비지 컬렉터는 프로그램을 멈추고 루트 노드로 부터 참조되고 있는 객체 그래프를 순회하여, 참조되고 있는 객체는 아래쪽의 힙으로 옮겨진다. 5는 가비지 컬렉션이 끝난 모습을 나타낸다. 메모리 단편화는 해결되지만 동적 메모리의 효율이 50% 이하로 떨어지는 단점이 있다[3,5,6].



(그림3) 가비지 컬렉티드 힙

2.6 세대별(generation) 방법

세대별(generation)방법은 자바 프로그램에서 만들어지는 대부분의 객체들은 매우 짧은 시간 동안만 참조되어진다는 것과 오랫동안 유효했던 객체는 계속적으로 유효할 것이라는 전제를 바탕으로 한다. 객체에 생성된 시기를 기준으로 나이라는 개념을 도입하여 young 영역과 old 영역으로 구분하여 관리된다. 객체는 young영역내의 eden영역에 할당되는데, 더 이상 객체를 할당할 수 없는 경우 살아있는 객체는 복사 알고리즘에 의해 young 영역의 survivor 영역으로 복사된다. young 객체가 old객체보다 많이 가비지 컬렉터 한다. Hotspot, CVM에 적용되고, 세대별 객체 저장 공간을 별도로 필요함으로 메모리 요구 사항이 높아진다[4,6].

2.7 압축(Compacting) 방법

현재 참조되고 있는 객체를 힙의 한 쪽 끝으로 모으는 기법으로 한쪽의 다른 끝에는 크고 연속된 빈 공간이 생긴다. 옮겨진 객체는 새로운 위치로 참조가 갱신된다. 직접 참조방법과 간접 참조 방법이 있는데 직접 참조방법은 객체 참조변수가 힙의 메모리 공간을 직접 가리키므로 객체가 옮겨질 때 객체를 가르키는 모든 참조 변수의 값을 갱신하는 단점이 있고, 간접 참조 변수 방법은 객체변수가 객체 핸들 테이블의 엔트리를 가리키게 하여, 객체 핸들은 실제 힙 메모리 주소를 가르키게 하므로 객체 핸들 값만을 변화시킴으로 더 효율적, 한 단계 더 거쳐서 참조되므로 오버헤드가 크다. 가끔 객체의 참조에 간접 참조 방법을 첨가함으로써 작업을 더 간단하게 만든다 [3].

3. 세대별 가비지 컬렉터의 가비지 회수

이 장에서는 객체들의 전형적인 생명주기와 세대별 가비지 콜렉션을 사용하는 자바힙 구조와 가비지 회수에 대해서 설명한다.

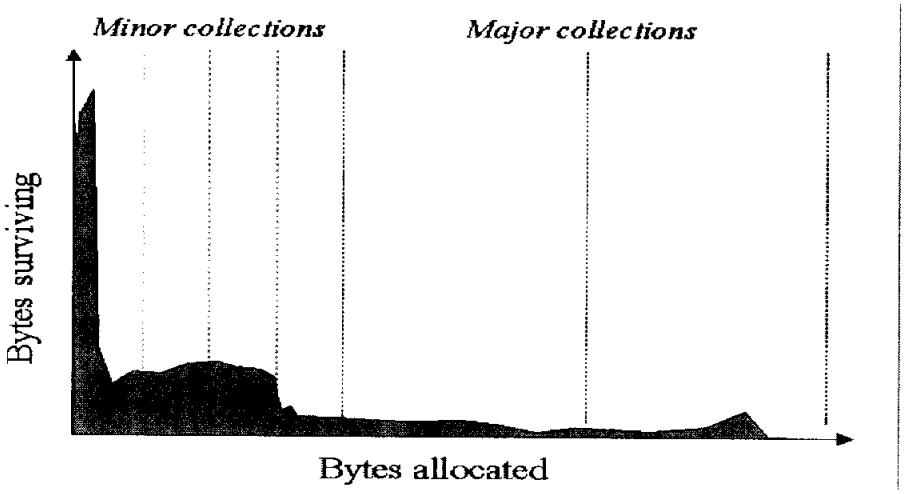
3.1 세대별 가비지 컬렉터의 자바 힙 구조

그림4에서 칠해진 영역은 객체들의 생명주기의 전형적인 분포이다. 왼쪽의 뾰족한 부분은 할당되어진 후에 간단하게 다시 선언되어진 객체들을 나타낸다

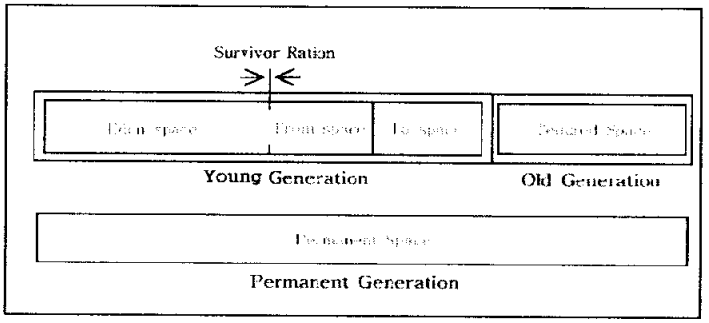
일부 객체들은 더 오랫동안 살아 있다. 그래서 분포는 오른쪽 밖으로 뻗어 있다. 물론 일반적으로 프로세스가 존재하는 동안에 살아서 초기화 된 객체들이 있다. 오른쪽으로 산등성이처럼 보이는 몇 가지 중간 계산 동안에 살아 있는 객체들이 존재한다. 일부 애플리케이션들은 매우 다른 관점의 분포들을 가진다. 그러나 놀랍게도 많은 수가 위의 일반적인 분포를 가진다. 객체의 대부분이 초기에 없어지므로 세대별 가비지 컬렉션이 만들어 져야 한다[11].

세대별 쓰레기 수집기를 사용하는 JVM의 힙 영역은 그림5과 같다. 힙 영역은 세 부분으로 나누어지는데 JVM 클래스와 메소드 객체를 저장하는 Permanent 영역, 만들어진지 좀 지난 객체들을 저장하는 Old generation 영역, 모든 새로 생성된 객체를 저장하는 Young generation 영역으로 구성된다. Young generation 영역은 new에 의해서 새로 생

성된 객체를 저장하는 Eden 영역과 Eden에 있던 객체가 Old 영역에 가기 전에 객체를 저장하는 Survivor 영역으로 구성된다[13].



(그림4) 객체 생명 주기 분포



(그림5) 힙(heap) 영역 레이아웃

세대별 가비지 컬렉터에는 두 가지의 collection이 있다. minor collection과 major collection 이다.

minor collection은 young generation에서 Old generation으로 살아있는 객체들을 이동시키고, 불필요한 객체들을 수거하는 것으로 pause time이 짧다.

major collection은 young과 old generation이 가득 차서 메모리 할당을 할 수 없을 때 수행하는 것으로 pause time이 길다[4,6].

3.2 young generation에서의 가비지 회수

young generation 가비지 컬렉터 알고리즘은 복사(copying) 알고리즘으로 처리한다. 3가지 방법이 있다.

(1) 복사 컬렉터(copying collector)

모든 응용 스레드(application thread)들은 모두 동작을 멈추고, 복사 컬렉터는 1개의 스레드만 사용하여서, 진행한다. 이것은 JVM이 컬렉션이 끝날 때까지 모든 동작을 멈추고 있기 때문에 stop-the-world collection으로 알려져 있다.

(2) 병렬 복사 컬렉터(parallel copying collector)

stop-world collection으로 다중스레드를 사용하여서 복사 컬렉터를 병렬로 진행한다. 다중 CPU 장치에서는 단일 복사 컬렉터를 사용하는 것 보다 훨씬 효과적으로 수행한다. 사용할 수 있는 CPU의 수와 같은

요인에 의해 young generation 수집을 잠재적으로 속도를 빠르게 한다.

(3)병렬 청소 수집기(parallel scavenge collector)

멀티 CPU상에서 GB(Giga Byte) 힙(heap, 10GB이상) 에 의해 조정된다. 이 알고리즘은 최소한의 지연으로 아주 높은 처리량(throught)을 제공한다. 이 알고리즘을 사용하면 나이든 세대에 대해서는 mark-compact를 사용해야 한다[1].

3.3 old generation에서의 가비지 회수

old generation가비지 컬렉터 알고리즘은 3가지의 방법이 있다.

(1) 마크-압축 컬렉터(mark-compact collector)

모든 참조할 수 없는 개체를 표시(mark)한다. 두 번째 단계에서, 참조할 수 없는 개체들은 압축된다. 이 기법은 단편화 문제를 회피하고, GC가 자주 수행되지 않을 때도 잘 동작한다

(2)병렬 컬렉터(Concurrent collector)

initial mark, Mark,Precleaning,Remark,Sweep,Reset 6단계로 이루어져 있다. 첫번째(initial-mark)와 네번째(remark) 단계는모든 스레드를 멈추는 기법이고, 나머지는 어플리케이션 스레드와 병렬로 수행된다. 다단계 가비지 컬렉션은 스레드를 멈추는 구간을 가능한 한 짧게 하여 어플리케이션이 가비지 컬렉션때 최소한으로만 지연되도록 한다.

(3)점진적 컬렉터(incremental collector)

점진적 가비지 컬렉션는 힙에 다수의 기차들로 나누어진 새 중간 영역을 생성한다. 각 기타로부터 한 번에 하나씩의 쓰레기가 회수된다. 이 기능은 더 짧고 잦은 가비지 컬렉션의 수행을 야기하지만, 전체적인 어플리케이션 성능은 떨어뜨릴 수 있다[1].

4. 성능 측정 및 분석

측정환경은 펜티엄Ⅲ, windows98, 자바가상머신으로는 JDK1.4.2를 사용했고, 4개의 응용프로그램으로는 10000개의 소수(prime)를 구하여 객체로 생성하는 프로그램과 벤치마크 프로그램인 java-olden 중 그래프의 최소 스패닝 트리(spanning tree)구하는 프로그램을 vertex를 500개를 인자로 주어 측정하고, bisort 프로그램으로 100000개 정렬을 인자로 주어 측정하고, 외판원 문제(Traveling Salesman Problem) 인자를 50000개를 주어 측정하였다. 힙의 크기를 조정하여 측정결과를 가비지 컬렉션 튜닝하는 프로그램을 통하여 디스플레이 하였다. 동일한 힙 크기 사용 시 young generation의 크기 비율을 조정하여 결과측정을 하였다.

4.1 힙 크기 가변, generation 크기 고정시의 성능 측정

힙의 크기를 변화시켜서 가비지 컬렉션의 회수와 실행시간을 측정하였다. 표1은 객체의 생명주기가 짧은 객체를 10000개 생성하였을 때 나온 결과이다. 힙의 크기가 커질수록 가비지의 컬렉션의 회수는 줄었지만, 실행시간은 증가함을 볼 수 있다.

〈표1〉 소수 구하기의 힙의 크기 변화에 따른 측정

힙크기	가비지 콜렉션 회수	실행시간(ms)
1M	5	8950
2M	5	9000
4M	5	9170
8M	5	9330
10M	4	9720
16M	2	9780
32M	1	9940
64M	0	12300

표2는 vertex 500개로 최소 스패닝(spanning)트리를 구하는 프로그램에서 힙의 크기 변화에 따른 측정 결과이다. 객체의 생명주기가 앞의 예제보다는 긴 객체 생성에 대한 예제이다. 힙의 크기가 클수록 가비지 컬렉션 회수가 감소했으며, 실행시간도 감소함을 볼 수 있다. 작은 크기의 힙에서는 살아있는 객체의 수가 증가함으로써 pause time을 많이 차지하는 major collection(괄호속 숫자)가 많이 발생함으로써 실행 시간이 증가하였다.

표3은 100000개를 bisort하는 프로그램에서 힙의 크기 변화에 따른 측정 결과이다. 먼저 생성된 객체와의 참조가 발생함으로써 객체의 생명주기가 긴 예제이다. 힙의 크기가 클수록 가비지 컬렉션 회수가 감소했으며, 실행시간도 감소함을 볼 수 있다.

<표2> 최소 스패닝트리의 힙의 크기변화에 따른 측정

힙 크기	가비지 콜렉션 회수 (major collection)	실행시간(ms)
1M	23(4)	2360
2M	20(3)	1650
4M	19(2)	1600
8M	19(1)	1370
10M	15(0)	880
16M	9(0)	820
32M	4(0)	800
64M	2(0)	770

<표3> bisort의 힙의 크기변화에 따른 측정

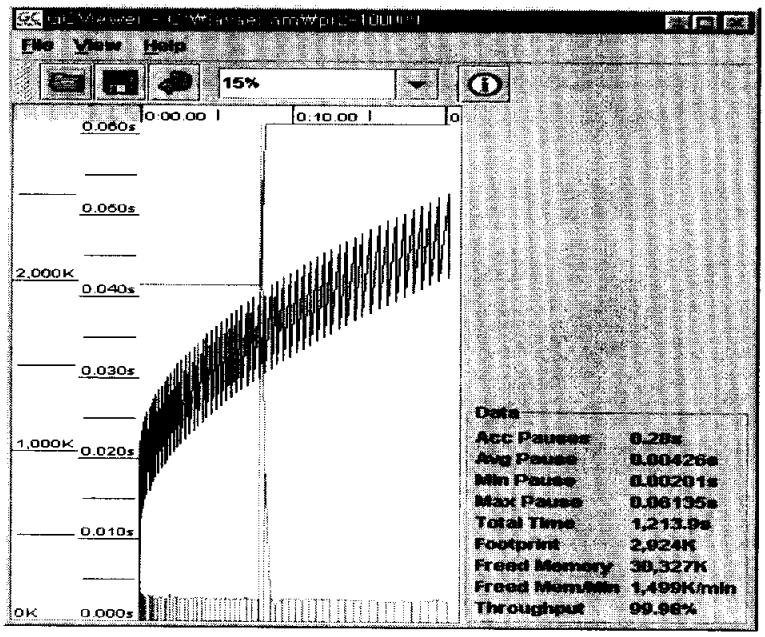
힙 크기	가비지 콜렉션 회수 (major collection)	실행시간(ms)
1M	5(3)	1200
2M	3(1)	1100
4M	3(0)	990
8M	3(0)	940
10M	2(0)	930
16M	1(0)	930
32M	0(0)	880
64M	0(0)	820

<표4> 외판원문제의 힙의 크기변화에 따른 측정

힙 크기	가비지 콜렉션 회수 (major collection)	실행시간(ms)
1M	19(3)	5440
2M	18(2)	5330
4M	16(0)	5160
8M	16(0)	5160
10M	13(0)	5110
16M	7(0)	5050
32M	3(0)	5050
64M	2(0)	4890

표4는 외판원 문제(Traveling Salesman Problem) 인자를 50000개를 주어 힙의 크기 변화에 따른 측정 결과이다. 표2,3 과 같이 힙의 크기가 클수록 가비지 컬렉션 회수가 감소했으며, 실행시간도 감소함을 볼 수 있다.

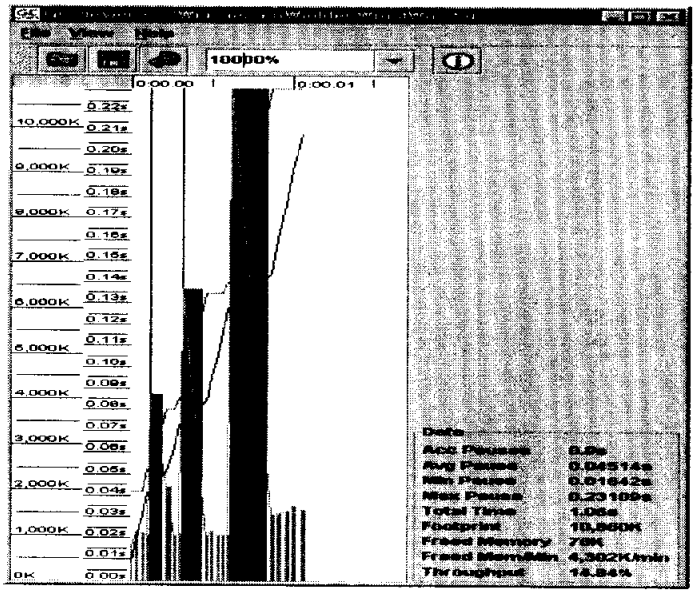
그림6은 객체 100000개를 생성하고, 힙의 크기를 2M 사용시 가비지 컬렉션 튜닝 프로그램으로 결과를 디스플레이한 결과이다. 1열은 힙의 소비량,2열은 garbage collection time, 3열은 응용프로그램 실행시간, 하단의 수직선은 minor collection을 나타내고, 중간에 가장 긴 수직선은 major collection을 나타내고, 리플은 사용되는 힙을 나타낸다. 수십 번의 minor collection 발생하였고, 한 번의 major collection이 발생한 이후로 사용 가능한 힙의 크기가 증가됨을 볼 수 있다. 가비지 컬렉션의 throughput은 99.98%로 높음을 볼 수 있다.



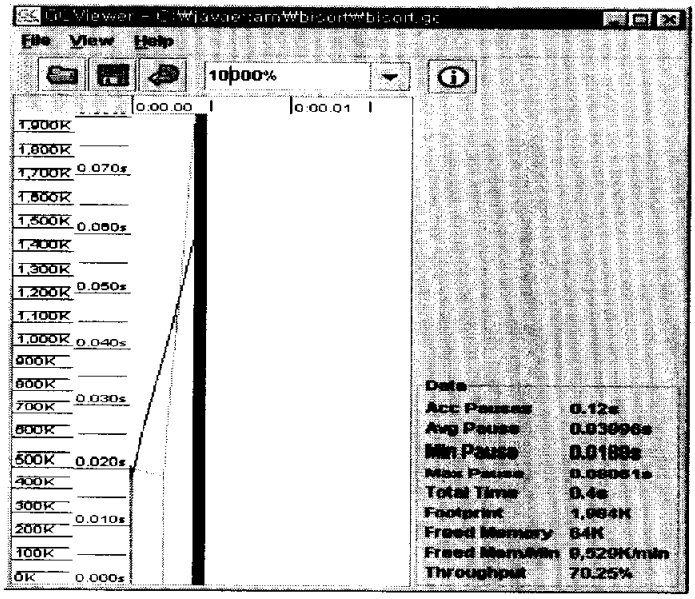
(그림6) 소수 구하기의 가비지 컬렉션 튜닝

그림7은 최소 스패닝 트리에 대한 힙의 크기를 2M vertex 500개 생성으로 가비지 컬렉션 튜닝을 나타낸 결과이다. 20번의 minor collection과 3번의 major collection이 발생함을 볼수 있으나 가비지 컬렉션의 throughput은 14.84%로 낮음을 볼 수 있다.

그림8은 bisort로 힙의 크기 2M로 100000개를 정렬한 가비지 컬렉션 튜닝을 나타낸 결과이다. 2번의 minor collection과 1번의 major collection이 발생함을 볼수 있고 가비지 컬렉션의 throughput은 70.25%를 볼 수 있다.

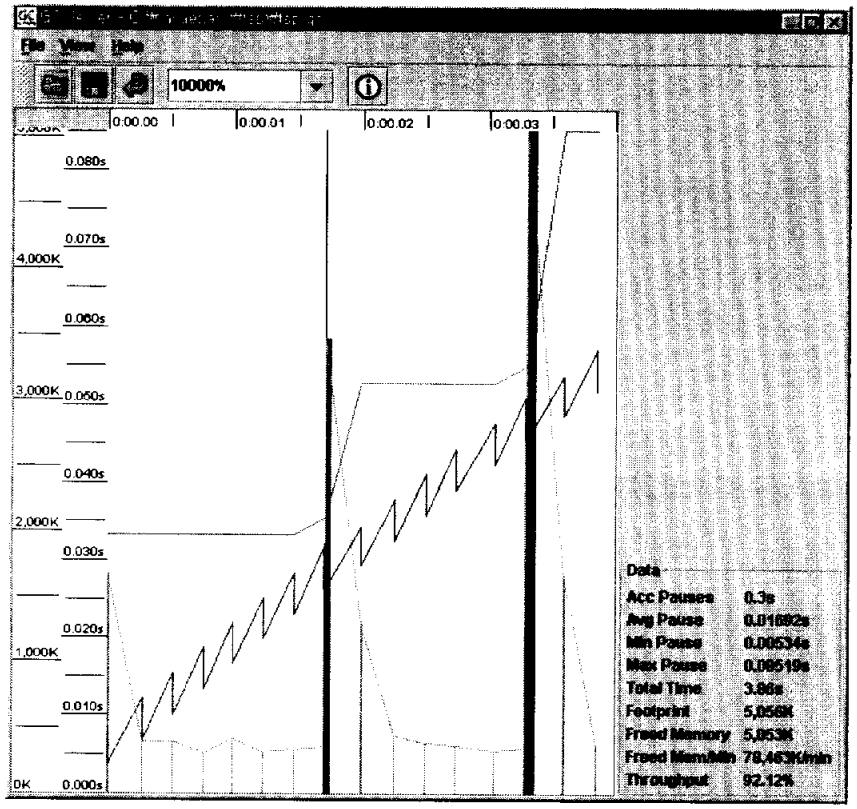


(그림7) 최소 스패닝트리의 가비지 컬렉션 튜닝



(그림8) bisort의 가비지 컬렉션 튜닝

그림9는 외판원문제(TSP)로 힙의 크기 2M로 50000개의 인자로 가비지 컬렉션 튜닝을 나타낸 결과이다. 16번의 minor collection과 2번의 major collection이 발생함을 볼 수 있고 가비지 컬렉션의 throughput은 92.12%를 볼 수 있다.



(그림9) 외판원 문제의 가비지 컬렉션 튜닝

4.2 힙크기 고정, generation 크기 가변시의 성능 측정

힙의 크기를 고정시키고,young generation 영역의 비율을 다르게 하여서 측정한 결과이다. 표5,표6,표7,표8에서 힙의 크기를 1M,2M,4M,8M로 고정하고, 각 young generation 영역을 힙에서의 차지하는 비율로 조정한 결과이다. 표5는 객체의 생명주기가 짧은 객체를 10000개 생성한 결과이다. 힙의 크기가
1M일 때에는 가비지 컬렉션의 회수는 young generation 영역 조정하지 않을 때와 결과는 동일하나, 실행시간이 더 걸렸다. 힙의 크기가 2-8M일 때 young generation 크기가 클 때에는 가비지 컬렉션 회수를 줄이고, 실행시간이 많이 걸렸다. 전체적으로 young generation을 조정하지 않을 때 보다 훨씬 가비지 컬렉션 회수는 감소하였으나 실행 시간은 많이 걸렸다.

표6은 vertex 500개로 최소 스패닝(spanning)트리를 구하는 프로그램으로, 동일한 힙 크기일 때, young generation 크기가 클수록 가비지 컬렉션의 회수가 감소하고, 실행 시간도 감소함을 보였다. 전체적으로 young generation을 조정하지 않을 때 보다 훨씬 가비지 컬렉션 회수와 실행 시간 면에서 성능이 향상되었다.

<표5> 소수 구하기의 young generation 비율 조정

힙 크기	Young /heap	가비지 콜렉션 회수 (major collection)	실행시간(ms)
1M	1 / 2	5	9450
	1 / 3	5	9390
	1 / 4	5	9230
	1 / 5	5	9120
2M	1 / 2	3	9830
	1 / 3	5	9440
	1 / 4	5	9340
	1 / 5	5	9180
4M	1 / 2	1	12240
	1 / 3	2	10550
	1 / 4	3	10440
	1 / 5	4	10320
8M	1 / 2	0	15660
	1 / 3	1	12350
	1 / 4	1	11260
	1 / 5	2	10380

<표6> 최소 스패닝트리의 young generation 비율 조정

힙 크기	Young /heap	가비지 콜렉션 회수 (major collection)	실행시간(ms)
1M	1 / 2	7(3)	1210
	1 / 3	10(3)	1320
	1 / 4	12(4)	1600
	1 / 5	13(5)	1970
2M	1 / 2	7(3)	1590
	1 / 3	10(3)	1380
	1 / 4	11(3)	1370
	1 / 5	13(3)	1480
4M	1 / 2	4(2)	1320
	1 / 3	7(2)	1380
	1 / 4	9(2)	1480
	1 / 5	11(2)	1540
8M	1 / 2	3(1)	1040
	1 / 3	4(1)	1090
	1 / 4	6(1)	1160
	1 / 5	7(1)	1210

표7은 100000개를 bisort하는 프로그램에서 힙의 크기 변화에 따른 측정 결과이다. 동일한 힙 크기일 때, young generation 크기가 클수록 가비지 컬렉션의 회수가 감소하고, 실행 시간도 감소함을 보였다. 하지만 young generation 크기가 전체 힙의 1/2일 때가 가장 실행 시간이 적었고, 나머지 실행시간은 거의 비슷하거나 동일함으로써 차이가 거의 없었다. 전체적으로 young generation을 조정하지 않을 때 보다 가비지 컬렉션 회수와 실행 시간면에서 성능이 향상되었다.

<표7> bisort의 young generation 비율 조정

힙 크기	Young /heap	가비지 콜렉션 회수 (major collection)	실행시간(ms)
1M	1 / 2	3(2)	990
	1 / 3	5(3)	1100
	1 / 4	5(3)	1100
	1 / 5	5(3)	1100
2M	1 / 2	1(0)	990
	1 / 3	4(1)	1050
	1 / 4	4(1)	1050
	1 / 5	4(1)	1100
4M	1 / 2	1(0)	980
	1 / 3	1(0)	990
	1 / 4	1(0)	990
	1 / 5	2(0)	1040
8M	1 / 2	0(0)	880
	1 / 3	0(0)	940
	1 / 4	1(1)	990
	1 / 5	1(1)	990

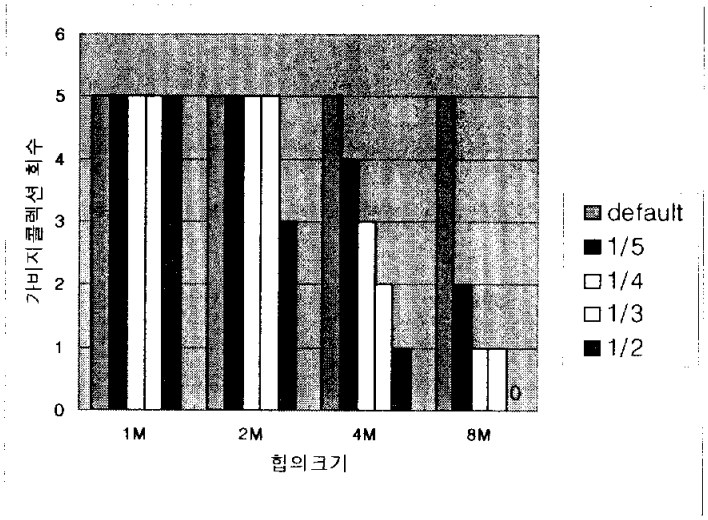
표8은 외판원문제 프로그램에서 힙의 크기 변화에 따른 측정 결과이다. 동일한 힙 크기일 때, young generation 크기가 클수록 가비지 컬렉션의 회수는 감소하고, 실행 시간은 증가함 을 보였다. 전체적으로 young generation을 조정하지 않을 때 보다 훨씬 가비지 컬렉션 회수와 실행 시간 면에서 성능이 향상되었다.

<표8> 외판원문제의 young generation 비율 조정

힙 크기	Young /heap	가비지 콜렉션 회수 (major collection)	실행시간(ms)
1M	1 / 2	10(3)	5220
	1 / 3	13(3)	5170
	1 / 4	16(3)	5160
	1 / 5	17(3)	5160
2M	1 / 2	8(2)	5170
	1 / 3	13(2)	5160
	1 / 4	14(2)	5110
	1 / 5	16(2)	5110
4M	1 / 2	5(1)	5160
	1 / 3	8(1)	5060
	1 / 4	9(0)	5000
	1 / 5	13(0)	4950
8M	1 / 2	2(0)	5050
	1 / 3	3(0)	5000
	1 / 4	5(1)	4950
	1 / 5	6(0)	4940

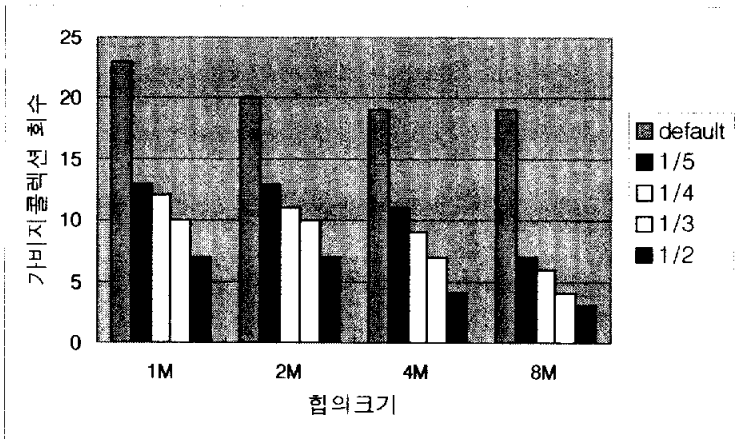
4.3 세대별 가비지 컬렉터의 성능 분석

그림10은 소수 구하기 프로그램을 동일한 힙 크기일 때 기본(default)과 young generation크기를 조정하였을 때 가비지 컬렉션의 회수를 그래프로 나타낸 것이다. 힙의 크기가 1M, 2M에서는 차이가 나지 않지만 4M, 8M에서는 회수의 감소를 볼 수 있다.



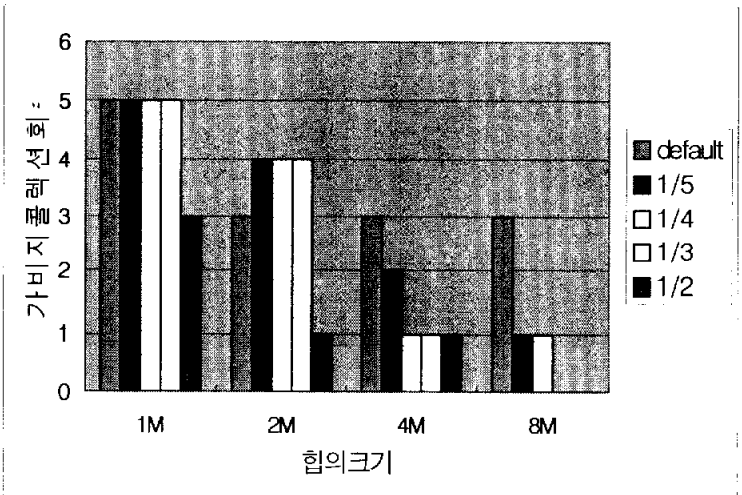
(그림10) 소수 구하기의 가비지 컬렉션 회수 비교

그림11은 최소 스패닝 트리 구하는 프로그램을 동일한 힙 크기일 때 기본(default)과 young generation크기를 조정하였을 때 가비지 컬렉션의 회수를 그래프로 나타낸 것이다. 힙의 크기가 클수록 현저하게 회수의 감소를 볼 수 있다.



(그림11) 최소 스패닝트리의 가비지 컬렉션 회수 비교

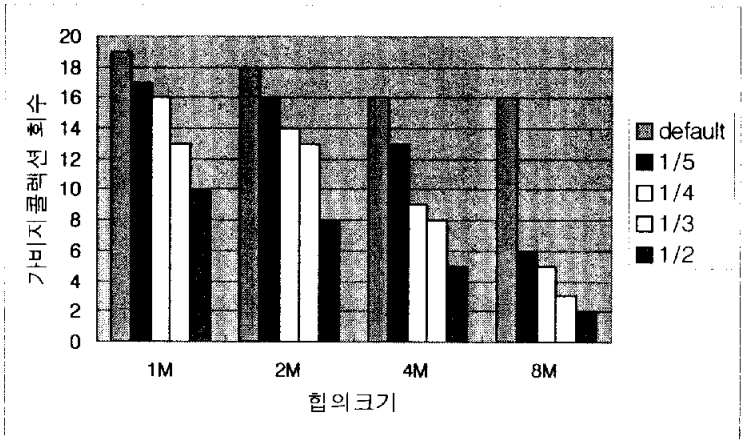
그림12는 100000개를 bisort하는 프로그램을 동일한 힙 크기일 때 기본(default)과 young generation크기를 조정하였을 때 가비지 컬렉션의 회수를 그래프로 나타낸 것이다. 힙의 크기가 클수록 현저하게 회수의 감소를 볼 수 있다.



(그림12) bisort의 가비지 컬렉션 회수 비교

그림13은 외판원문제 구하는 프로그램을 동일한 힙 크기일 때 기본(default)과 young generation크기를 조정하였을 때 가비지 컬렉션의 회수를 그래프로 나타낸 것이다. 힙의 크기가 클수록 현저하게 회수의 감소를 볼 수 있다.

전체적으로 young generation의 크기를 조정하여서 사용하는 것이 가비지 켈렉션의 회수를 줄임으로써 가비지 컬렉션으로 인한 오버헤드를 줄이는 방안이다.



(그림13) 외판원 문제의 가비지 컬렉션 회수 비교

5. 결론

자바는 메모리 관리를 가비지 컬렉터가 자동으로 관리를 해준다. 가비지 컬렉터는 프로그램에 의해 더 이상 참조되지 않는 객체들이 차지하고 있던 heap공간을 재사용 할 수 있게 함으로써 소프트웨어의 생산성을 향상시킬 수 있는 장점을 가지고 있다.

대부분의 객체가 초기에 소멸되므로 능률적인 가비지 컬렉션 방법으로 세대별 가비지 컬렉션을 사용하는 것이다. 특히 객체의 생명 주기가 짧은 응용프로그램에서는 힙의 크기가 커질수록 가비지의 컬렉션의 회수는 줄지만 실행시간이 증가되었다. 동일한 힙에서 young generation 크기를 조정하지 않을 때 보다는 훨씬 가비지 컬렉션 회수는 감소하였으나 실행 시간은 많이 걸렸다.

메모리의 최대 사용량이 높고, 객체의 생명이 긴 응용프로그램에서는 힙의 크기에 따라 가비지의 컬렉션의 회수도 줄고, 실행시간도 줄었다. 동일한 힙에서 young generation을 조정하지 않을 때 보다 조정할 때가 훨씬 가비지 컬렉션 회수와 실행 시간 면에서 성능이 향상되었다.

전체적으로 young generation의 크기를 조정하여서 사용하는 것이 가비지 컬렉션의 회수를 줄임으로써 가비지 컬렉션으로 인한 오버헤드를 줄이는 방안이다.

적은 메모리를 사용하고, 세대별 쓰레기 수집기(generation garbage collector) 알고리즘을 사용하는 CDC(Connected Device Configuration)의 CVM(C Virtual Machine)에서도 본 논문에서 제안한 방법을 사용하

는 것이 효율적이다.

향후 연구로는 자동화된 메모리 관리하에 명시적으로 객체를 수거하여 가비지 컬렉션 회수와 실행 시간면에서 성능 향상을 고찰하고자 한다.

참고문헌

[1] An excellent article on 1.4.1 garbage collection "Improving Java Application Performance and Scalability by Reducing Garbage Collection Times and Sizing Memory Using JDK 1.4.1," Nagendra Nagarajayya and J. Steven Mayer (Sun Microsystems, November 2002)

[2] A. Petit-Bianco, No Silver Bullet Garbage Collection for Java in Embedded Systems, Cygnus Solution, August 1998.

[3] R. Jones, and R. Lins, Garbage Collection Algorithms for Automatic Dynamic Memory Management, John Wiley & Sons, 1996

[4] Sun Microsystems, The Hotspot Virtual Machine Technical White Paper, Sun Microsystems,2001

[5] Bill Venners, "Inside the JAVA2 Virtual Machine Second edition",pp.356-384,1999

[6] Richards Jones, "Garbage Colletion Algorithm for Automatic Dynamic Memory Management" pp.75-182,1999

[7] P Wilson,Uniprocessor Garbage Collection Techniques, International Workshop on Memory Management, Number 637 Lecture Notes in Computer Science, pp. 1-42, St. Malo, France, September 1992. Springer-Verlag.

[8] 박준석, 김명규, 한동원, "임베디드 시스템의 자바 가속 기술", 전자통신동향분석,제18권 제2호,2003.

[9] 배수강, 이승룡, 전태웅, "자바 실행시간 환경에서 명시적인 동적 메모리 관리 기법", 정보과학회 논문지,제 30권 제 1 호, pp.58-71, 2003.

[10] http://sookmyung.ac.kr/~u0011616

[11] http://www.monosun.com/doc/gc.html,Java HotSpot VM Options

[12] http://www.javaperformancetuning.com,JavaPerformance tunning 2002.

[13] http://www.borlandforum.com

[14] http://www.jabook.net

감사의 글

대학 졸업후 늦었다고 생각한 시기에 시작한 대학원 생활은 새롭고, 소중한 많은 경험을 주는 귀중한 시간들이었습니다. 소중한 경험을 얻을 수 있도록 해 주신 많은 분들께 감사의 말씀 드립니다.

먼저 부족한 점이 많고, 나약한 저에게 힘들어 할 때마다 할 수 있다는 용기과 따뜻한 배려를 베풀어 주신 존경하는 지도교수님 윤성대 교수님께 진심으로 감사의 말씀을 드립니다. 논문 심사를 위해 바쁘신 와중에도 세심한 조언과 배려를 해주신 여정모 교수님, 많은 격려와 가르침을 주신 김창수 교수님께 감사의 말씀을 드립니다.

항상 연구실에 가면 따뜻하게 맞이 해주시고, 많은 조언과 격려를 해주신 박현호 선배님, 바쁘신 와중에도 부족한 부분을 항상 시원하게 매꾸어 주신 황순환 선배님, 세세하게 도움의 말씀을 해주신 박월선 선배님, 항상 정중한 말씀으로 도움을 주신 박상일 선배님, 항상 웃는 얼굴로 많은 도움을 주신 오윤주 선생님, 웃으면서 많이 챙겨주신 김진현씨, 같이 논문 준비하면서 많이 도움주신 박성련 선생님, 많이 챙겨주신 한지영 선생님, 같이 학회 논문 준비하면서 많은 도움과 격려해 주신 윤종찬 선생님, 웃는 얼굴인 박해준 선생님, 항상 열렬하게 격려해 주신 이경미 선생님, 천소영 선생님, 즐거운 연구실 생활을 할 수 있게 해준 유은아 선생님, 너무 다정해서 자매같은 이미나 선생님, 서정애 선생님, 정귀녀 선생님, 열심히 공부하는 학부생과 연구실의 모든 식구들에게 감사드립니다.

항상 걱정해주고, 많이 챙겨주신 부산대학교 김미경 선생님께 감사드립니다.

그리고 대학원 생활을 할 수 있도록 육아, 집안일을 해주시느라 고생하신 시어머니, 힘들 때 항상 따뜻하게 격려해 주신 부모님, 어려운 일을 겪으면서도 많은 격려를 해준 언니, 멀리서 격려해 준 오빠, 묵묵히 나의 부탁을 들어준 남동생 내외에게 감사드립니다.

끝으로 대학원 생활동안 불평하지 않고, 적극적으로 지원해 주고, 많은 사랑을 준 나의 든든한 배우자인 정동인씨, 많이 챙겨주지도 못해서 미안한데 건강하게 자라주고, 항상 웃는 얼굴로 엄마를 응원해준 나의 사랑하는 아들 정진욱, 딸 정예원에게 감사와 사랑한다는 말을 전합니다.