패턴 매칭 기법을 이용한 GVM SAL 코드 최적화 (GVM SAL Code Optimization using Pattern Matching Technique)

김은경, 오세만
동국대학교 컴퓨터공학과 컴퓨터공학전공
{plusek, smoh}@dongguk.edu

요약

GVM(General Virtual Machine)은 무선 단말기 상에서 다운로드 솔루션을 통해 동적인 응용프로그램을 효과적으로 실행할 수 있는 가상기계(Virtual Machine) 플랫폼이다. GVM은 시스템 리소스의 제한이 큰 무선 단말기 상에서 SGS 파일을 다운로드 받아 실행되는 시스템이므로, 성능의 저하 없이 실행되기 위해서는 보다 효율적인 컴파일러의 코드 생성이 요구된다.

본 논문은 GVM의 SAL 코드에 대한 최적화를 수행하는 코드 최적화기 시스템을 설계하고 구현하였다. 존재하는 최적화 방법론에 관한 연구를 통하여 SAL 코드의 특성을 고려한 알고리즘을 제안하고, 이를 바탕으로 패턴 테이블을 이용한 패턴 매칭 기법을 사용하였다. 구현된 시스템의 최적화율은 코드 크기 면에서 약 22% 정도의 감소를 이루었고 실행 속도 면에서도 평균적으로 약 8% 정도 향상되었다. 이렇게 함으로써 네트워크상의 전송 속도뿐만 아니라 가상 기계에서도 좀 더 빠르게 실행할 수 있다.

1. 서론

GVM(General Virtual Machine)은 무선 단말기 상에서 동적인 응용프로그램을 실행할 수 있게 순수 국내 기술로 개발된 가상 기계(Virtual Machine) 플랫폼이다. 가상 기계를 이용한 응용프로그램의 실행은 플랫폼 독립적인 실행이 가능하며 또한 효과적인 다운로드 솔루션을 통한 동적인 실행이 가능하다.

GVM은 중간 형태의 SAL(Sinji Assembly Language) 코드를 생성하고, SAL 어셈블러를 통해 단말기용 게임 플러그인에서 동작하는 바이너리 형태의 SGS(Sinji Game Script) 파일로 변환된 후 다운로드 받아 실행되는 시스템이다.

가상기계가 갖춰야 할 가장 기본이 되는 필요충분조건은 단연 '저용량'과 '고속'에 있다. 모바일 디바이스의 제한된 메모리와 저속 CPU 환경에서 원활히 수행되기 위해서는 가상기계가 끊임없이 경량화

되고 고속 동작을 보장해야만 한다.

GVM 또한 가상기계의 가장 큰 단점으로 제기돼 왔던 처리 속도 문제를 해결할 필요가 있다. 또한 고용량 데이터베이스를 고속으로 처리해야 하는 모바일 멀티미디어 서비스를 위해서는 모바일 디바이스의 메모리 부담을 줄이는 일이 요구된다.

일반적으로 컴파일러는 원시 프로그램에 대한 컴파일과정 중 최적화 단계에서 프로그램의 실행 속도를 개선시키고 코드 크기를 줄일 수 있는 다양한 최적화 기법을 수행한다.

GVM에는 보다 빠른 사운드 및 이미지 처리를 위한 최적화 코드들이 설계되어 있다. 이를 이용하여 SAL 코드의 최적화를 수행하면 보다 효율적인 코드를 생성해낼 수 있다. 최적화된 SAL 코드들은 시스템에서 적은 네트워크 로드를 가지고 실행될 수 있도록 하며, 실행 시간을 개선하고 전체 SGS 파일의 크기를 줄이게 된다.

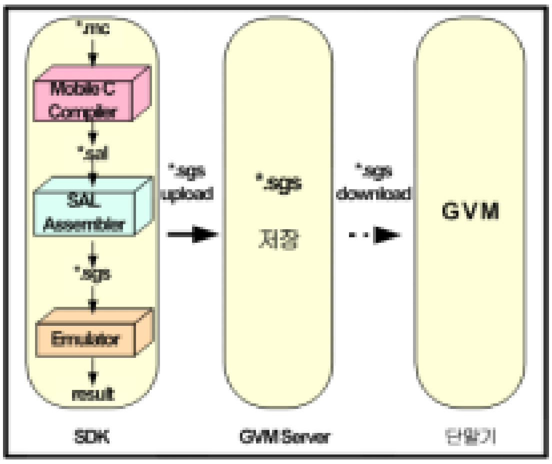
본 논문은 GVM의 SAL 코드에 대해 최적화를 수행하는 코드 최적화기 시스템을 설계하고 구현하였다. 존재하는 최적화 방법론에 관한 연구를 통하여 SAL 코드의 특성을 고려한 알고리즘을 제안하였다. 또한 이를 바탕으로 패턴 테이블을 작성하고 패턴 매칭 기법을 사용하여 최적화를 수행하였다.

본 논문의 구성은 2장에서는 본 연구의 기반이 되는 GVM, SAL 그리고 최적화 기법에 대한 관련 연구를 소개한다. 3장에서는 본 연구에서 설계하고 구현한 SAL 코드 최적화 시스템에 대해서 기술한다. 4장에서는 최적화 알고리즘의 성능을 평가하며 마지막으로 5장에서는 결론과 향후 연구 과제에 대해 기술한다.

2. 관련연구

2.1. GVM과 SAL 코드

GVM 컨텐츠는 표준 C언어를 기반으로 모바일 환경에 최적화된 언어인 Mobile C로 개발된다[3]. Mobile C 소스 파일(*.mc)은 Mobile C 컴파일러를 통해 중간 형태의 코드(*.sal)를 생성하고, SAL 어셈블러를 통해 GVM에서 실행될 수 있는 코드(*.sgs)로 변환된다. 이렇게 작성된 모바일 응용 프로그램은 서버에 업로드 시킨 후 GVM이 내장된 단말기로 다운로드 되어 실행하게 된다. [그림 1]은 이와 같은 과정을 나타내고 있다.



[그림 2] GVM 애플리케이션 실행 모델

SAL은 GVM의 어셈블리 언어에 해당된다. SAL은 실제 실행이 되는 연산 코드(operation code)와 이에 대한 선언과 데이터를 지칭하는 의사 코드(pseudo code)로 분류된다. SAL의 연산 코드는 218개이며 크게 5개의 카테고리로 나눌

수 있다. 각 카테고리는 스택 관련 연산, 산술/논리 연산, 배정(assignment) 연산, 제어 연산, 간접 연산이다[3].

모든 연산 코드는 1바이트 길이를 갖는다. 코드 크기를 줄이기 위해 명령어 길이는 연산 코드에 따라 가변적이다. 코드 최적화를 위해 다양한 스택 연산 코드가 정의되어 있다(예: pushx, pushi, pushic, pushz, pushc, pushw, popx, popi, popic, 등). 그리고 스택을 사용하지 않고 연산하기 위해 다양한 배정 연산 코드가 정의돼 있다(예: setx, seti, setn, setz, setc 등). 또한 내장 API를 연산 코드화해 연산 코드의 패치 시간을 줄이도록 설계되었다.

2.2. 코드 최적화 방법론

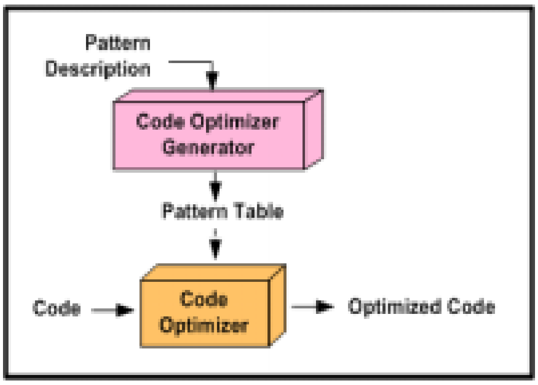
코드 최적화란 주어진 입력 프로그램과 의미적으로 동등하면서 보다 효율적인 코드로 바꾸는 과정을 말한다.

최적화 과정은 개선 대상과 관찰의 범위에 따라 크게 두 가지로 분류된다. 첫째, 프로그램의 기본 블록(basic block)내에서 최적화 기법을 적용하는 지역 최적화(local optimization)와 둘째, 기본 블록들 간에 최적화 기법을 적용하는 전역 최적화(global optimization)로 나눌 수 있다.

또한 프로그램 실행에 있어서 대부분의 시간을 차지하는 루프(loop)에 대해 코드의 개선 방법을 적용하는 루프 최적화가 있다. 그리고, 전단부에서 생성된 중간 언어에 대해 최적화를 수행하는 기계 독립적인 최적화(machine independent optimization)와 목적기계의 특성에 따라 목적 코드에 대해 수행되는 기계 종속적인 최적화(machine independent optimization)로 나눌 수 있다[1].

2.3. 코드 최적화 시스템

코드 최적화 과정을 자동화한다는 것은 기계 독립적인 코드 최적화 알고리즘(Code Optimization Algorithm)과 코드 최적화 알고리즘이 사용하는 테이블을 패턴 묘사(pattern description)로부터 자동적으로 유도하는 것을 의미한다. [그림 2]는 이와 같은 과정을 보여주고 있다.



[그림 3] 코드 최적화 시스템

코드 최적화기 시스템은 크게 패턴 묘사, 코드 최적화기 생성기, 패턴 테이블 그리고 코드 최적화기로 구성되며, 입력 코드에 대한 의미적으로 동등하면서 보다 효율적인 최적화 코드를 생성해 낸다.

패턴 묘사는 (패턴부:대치부)의 쌍으로 구성되는데 패턴부는 최적화기에 의해 인식될 수 있는 최적화되지 않은 명령어의 열을 나타내며 대치부는 패턴에 대해 최적화된 명령어로 변환되는 부분을 나타낸다. 코드 최적화기 생성기는 패턴 묘사의 내용을 파싱하고 (패턴:대치)쌍으로 구성된 입력을 받아들여 패턴 테이블을 출력한다.

패턴 테이블은 코드 최적화기가 효율적으로 이용할 수 있도록 패턴 묘사를 바이트 배열 형태로 갖고 있는 테이블이다. 코드 최적화기는 구성된 패턴 테이블을 참조하여 입력으로 들어오는 코드에 대하여 패턴 매칭 기법을 통해 최적화된 코드로 생성한다.

3. SAL 코드의 최적화 시스템 구현

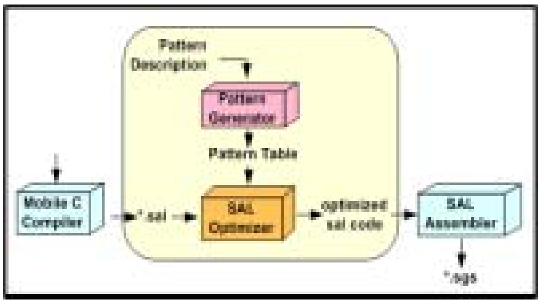
3.1. 시스템 개요

GVM 위에서 실행되는 SAL 코드는 기계 독립적인 특징을 가지고 있으므로 SAL 코드의 패턴을 분석하여 좀 더 코드의 크기를 줄이면서, 실행 효율을 높일 수 있다.

본 절에서는 하나의 SAL 코드로부터 의미는 동등하면서 보다 효율적인 SAL 코드를 생성하는 SAL 최적화기의 시스템 구성 모델을 제시한다. SAL 최적화기 시스템은 크게 패턴 생성기, SAL 코드 최적화기의 두 부분으로 구성되어 있다.

SAL 코드의 특성을 고려하여 작성된 패턴 묘사를 통해 패턴테이블을 생성한다. SAL 코드 최적화기는 전단부에서 생성된 SAL 코드를 입력받아 패턴테이블을 이용한 패턴 매칭 기법을 통해 최적화된 SAL 코드를 생성한다. [그림 3]은 이와 같은 과정을 나타내고 있다.

3.2. 패턴 테이블



[그림 3] SAL 코드 최적화 시스템 구성도

패턴 테이블은 입력된 SAL 코드 명령어 집합을 향상된 SAL 코드로 교체하기 위한 수단으로써 사용한다. 패턴 테이블을 생성하기 위하여 패턴 생성기의 입력으로 패턴 정의 언어가 입력으로 사용되고 그 결과로 SAL 코드 최적화기 모듈의 입력으로 사용되는 패턴 테이블이 생성된다. 이 패턴 정의 언어는 SAL 코드 명령어의 연속적인 SAL 코드 열에서 찾아낸 패턴을 나열한 것으로 EBNF 형태로 정의한다.

하나의 패턴 형태는 패턴(pattern)과 대치(replacement)로 구성되며 구분자는 '::=' 이다. 패턴부는 최적화기에 의해 인식될 수 있는 최적화되지 않은 SAL 명령어의 열을 나타내며 대치부는 패턴에 대해 최적화된 SAL 명령어로 변환되는 부분을 나타낸다. 코드 최적화기에서는 패턴과 매칭이 되는 부분을 찾아내어 대치부분으로 대치해나간다.

[표 1]은 최적화에 사용될 SAL 코드 패턴 테이블을 보여주고 있다.

[표 1] SAL 코드 패턴 테이블

```
[pushc|pushw]c1 [pushc|pushw]c2 [add|
sub| mult| div| mod]
  ::= [pushc|pushw] [c1+c2| c1-c2|
c1*c2| c1/c2|   c1%c2]
  ......
```

```
pushc c ldrz A add ::= ldric A c
ldric A c ixa ::= pushic A c
pushz B ldrz A ixa ::= pushi A B
  ......
pushc c popz A ::= zsetc A c
pushz B popz A ::= zset A B
pushic B c popz A ::= zsetn A B c
  ......
$$1 nop ujp $$2 ::= nop ujp $$2
ujp $$1 $$1 nop ::= $$1 nop
  ......
```

패턴에 반영된 최적화 내용은 크게 일반적인 최적화 기법과 SAL 코드의 특징적 요소를 반영하는 SAL 코드 의존적인 기법으로 분류할 수 있다. 일반적인 최적화 기법은 기존 연구를 통해 이론적으로 정립되고, 다른 최적화 시스템에 의해서 적용되고 있는 기법들을 의미한다. 본 시스템에 적용된 기법으로는 실행에 불필요한 코드의 제거, 중복된 코드의 제거, 상수들로 구성된 연산들에 대한 상수 폴딩, 제어 관련 최적화 등이 있다.

그리고 SAL 코드 의존적인 최적화 내용으로는 스택 관련 최적화, 배정 관련 최적화 등을 수행하였다. 스택 관련 최적화는 SAL 코드가 스택 기반 언어인 것을 고려하여 효율적인 스택 운영을 위한 것이다. 배정 관련 최적화는 보다 빠른 사운드 및 이미지 처리를 위하여 설계된 다양한 특수 코드들을 이용한 것이다. 이 배정 연산 코드들은 여러 개로 구성되어 있는 배정 표현을 하나의 코드로 변경하는 등의 최적화를 수행하게 된다.

3.3. 패턴 생성기

패턴 생성기는 패턴 묘사 테이블의 내용을 파싱하고 (패턴::=대치)쌍으로 구성된 입력을 받아들여 SAL 코드 최적화기가 최적화된 SAL 코드 생성 시에 필요한 정보를 테이블 형태로 출력한다.

테이블을 생성하기 위해서는 먼저 입력으로 들어오는 패턴 묘사에 대한 구문 분석(syntax analysis)과 의미 분석(lexical analysis)을 수행한 후 테이블 정보를 파일(table.h, table.c)에 직접 생성하기도 하고 배열에 임시적으로 저장하기도 한다. 생성된 임시 정보는 테이블에 추가적인 정보를 생성하는데 이용된다. 패턴 탐색은 SAL 코드 패턴 문법으로부터 패턴을 찾아낸다.

패턴 생성기의 출력인 table.h에는 실제 SAL 코드 최적화기에서 사용되는 배열의 크기를 결정하는데 필요한 상수를 정의하고 있다. 또 다른 출력인 table.c 파일은 코드 생성기에 의해 참조되는 정보를 배열 구조로 저장하고 있다.

3.4. SAL 코드 최적화기

SAL 코드 최적화기는 SAL 코드를 입력으로 받아 패턴 생성기에 의해 생성된 테이블 정보를 참조하여 실질적으로 최적화된 SAL 코드를 생성하는 부분이다.

SAL 코드 최적화에 적용되는 최적화 알고리즘은 패턴을 탐색하는 부분과 기존 최적화 알고리즘을 적용하는 부분으로 구성된다. SAL 최적화기의 목적은 기존 최적화 방법인 핍홀 최적화에 의하여 Mobile C 컴파일러가 일관성 있게 생성한 코드들을 조작하여 좀 더 작은 크기의 SAL 코드 집합과 좀 더 실행 효율이 높은 코드들로 바꾸어 생성하는 데 있다. 또한 SAL 최적화기는 SAL 코드에 대응하는 최적의 패턴을 찾기 위해 패턴테이블을 이용한 패턴 매칭 기법을 적용하여

동등한 의미의 보다 효율적인 코드를 생성해 낸다.

SAL 코드 최적화기가 재귀적인 호출 방식으로 최적화된 SAL 코드를 생성하는 알고리즘은 [표 2]와 같다.

[표 2] SAL 코드 최적화 알고리즘

[단계 1] 첫 명령어를 기준으로 최적화기의 패턴 테이블에서 가장 긴 연산자 패턴부터 검색

[단계 2] 명령어의 피연산자들과 패턴 테이블에서의 피연산자가 대응되는지를 검사

[단계 3] 찾아진 코드 규칙의 하나 이상의 패턴 중 최소의 비용을 갖는 패턴을 선택

[단계 4] 제약 조건이 만족되는지 검사하여 일치하면 패턴의 대치 명령어로 대치

[단계 5] 대치가 수행된 후 매칭 과정은 다시 대치된 첫 문장으로 제어를 옮겨 리스캔(rescan)을 반복하며 더 이상 기본 블록 내에서 최적화가 이루어지지 않을 때까지 반복 수행

결론적으로 SAL 코드 최적화기는 현재 상태에서 도달될 수 있는 각 상태로의 이동을 찾아보고 그 중에서 가장 최소 비용을 갖는 하나를 선택한다. 이렇게 SAL 코드 최적화 단계를 수행한 SGS 파일은 부분적으로 SGS 파일의 최적화를 가져온다. 최적화된 SAL 코드는 SGS 파일로 변환하는 동안 변수명을 축약하여 SGS 파일을 생성한다. 이로써 전체 SGS 파일의 크기를 줄이고, 실행될 때 수행 속도 면에서 좀 더 빠른 실행 속도를 가지게 된다.

또한 테이블을 이용함으로써 융통성이 뛰어나며 스택 기계에 기반을 둔 SAL 코드로부터 최적화된 SAL 코드를 생성할 수 있다.

4. 실험결과

본 논문에서 구현한 SAL 최적화기의 성능을 평가하기 위한 실험은 실행속도와 SAL 코드 크기의 변화를 비교 측정하였다. 실험 환경으로는 Intel Pentium-Ⅲ 1GHz, RAM 256MB, Windows XP Professional, GVM SDK version 1.52를 사용하였다.

실험대상 데이터는 VM의 특징을 살펴보기 위한 단순한 프로그램들과 본 논문에서 패턴 작성 시 중점 둔 부분을 테스트하기 위한 이미지/사운드 등 멀티미디어 기반의 복합적인 프로그램들을 선정해 사용하였다. 여러 기능 중 단순 연산 기능, 사각형 그리기, 원 그리기 테스트, 멀티미디어, 부가 기능 등 다양한 기능 테스트 등으로 나눠 진행했다.

[표 3] 최적화 전、후 비교 테이블

(*.sgs)	Code Size [byte]			Execution Time [msec.]		
	전	후	경감율	전	후	경감율
arithmetic	221	181	18 %	28.2	27	4 %
perfect	288	269	7 %	47	44.3	6 %
bubble	486	345	29 %	119	114	4 %
prime	248	233	6 %	32	31	3 %
eightqueens	512	396	23 %	123	112	9 %
matmul	412	329	21 %	103	95.1	8 %
rectangle	199	167	16 %	24.6	23.1	6 %
circle	263	208	21 %	41.2	37.4	9 %
sound	387	271	30 %	98	90.1	8 %
assigntest	569	370	35 %	131	117	11 %
multitest	633	437	31 %	137	120	13 %

[표 3]은 각각 Mobile C 컴파일러를 통해 생성된 SAL 코드와 SAL 최적화기를 통해 생성된 최적화된 SAL 코드를 입력으로 받아 생성된 SGS 파일을 각각 비교 분석한 것이다. SGS 코드 크기의 경감과 실행 속도의 경감을 통한 비교 실험을 하였다.

테스트 기본 자료는 타사의 VM BMT (벤치마크테스트) 자료에서 응용한 로직을 이용했다.

이 테스트는 연산측면에서의 VM 성능과 효율적인 화면 리플래시 처리 능력, 즉 이미지 성능 테스트와 연산과 이미지가 결합된 형태의 성능 테스트 등을 목적으로 수행했다. 또한 이미지/사운드 등 멀티미디어 기반의 복합 테스트를 함으로써 본 최적화기의 성능을 증명하였다. 특히, 사각형, 원 그리기, 멀티미디어 기능 등 본 최적화기의 패턴 테이블에서 중점을 둔 배열의 배정 관련 최적화 테스트에서는 두드러지게 향상되었다.

실험 결과 분석에서 실험에 이용되었던 최적화된 SGS 파일들은 코드 크기 면에서 약 20 ~ 30% 경감의 결과를 보여주었다. 실행 속도 면에서도 각 단위 예제에 대한 100회 반복 실험을 통하여 약 9 ~ 10% 정도의 향상을 나타내었다.

5. 결론 및 향후 연구

본 논문에서는 기존의 최적화 기법들의 분석을 기반으로 하여, GVM을 위한 중간 언어인 SAL의 특성을 고려한 최적화를 수행하였다.

SAL 최적화기는 패턴 매칭 기법을 이용해 SAL 코드에 대한 각 명령어의 특성을 고려하여 최적화를 수행하였다. 이와 같은 최적화는 패턴 묘사에 의존적이며 패턴 묘사에 기술된 최적화 내용은 널시퀀스 코드의 제거, 연산 강도 경감, 중복된 코드의 제거 등과 같은 최적화를 진행하였다. 특히 SAL 코드 의존적인 배정, 스택 연산에 관련된 최적화를 통하여 게임 등 멀티미디어 기반 프로그램의 실행을 고려한 이미지/사운드 복합 테스트에서 결과적으로 SGS 파일의 크기를 줄이고 좀 더 빠른 실행속도를 이끌어내었다. 이렇게 함으로써 네트워크상의 전송 속도뿐만 아니라 가상 기계에서도 좀 더 빠르게 실행할 수 있게 되었다.

본 논문의 향후 연구 방향은 개선될 수 있는 패턴에 대한 지속적인 연구가 필요하다. 또한 SAL 코드 최적화기에 사용되는 코드 최적화 알고리즘을 보다 효율적으로 보완하여, 양질의 최적화된 코드를 생성하는데 있다.

참고문헌

[1] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman, Compilers Principles, Techniques, and Tools, Addison-Wesley, 1985.

[2] B. J. Mckenzie, “Fast Peephole Optimization Techniques”, Software Practice and Experience, Vol. 19, No. 2, pp.1151-1162, Dec., 1989.

[3] Sinjisoft, GVM & GNEX Developer Guide Homepage, http://www.gnexclub.com

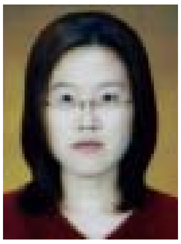
[4] W. M. Mckeeman, “Peephole Optimization”, CACM, Vol.8, No.8, pp.443 ~ 444, 1970.

[5] Has Van Staveren, Dept. of

Mathematics and Computer Science Vrije Universiteit Amsterdam, The Netherlands, "Internal Documentation on the peephole optimizer from the Amsterdam Compiler Kit", 1991.

[6] 김정숙, 고광만, 오세만, “트리 패턴 매칭을 이용한 핍홀 최적화기의 설계 및 구현”, 한국정보과학회 프로그래밍 언어 연구회 논문지, 제8권 2호, pp. 5 ~ 15, 1994.

[7] 이야리, 오세만, 자바 바이트코드를 위한 코드 최적화기의 자동적 구성, 동국대학교 박사학위 논문, 2001.



김 은 경
2003년 ~ 현재 동국대학교 대학원 컴퓨터공학과 프로그래밍언어연구실 석사과정
관심분야 : 컴파일러, 프로그래밍 언어, 모바일 컴퓨팅.

오 세 만
1993년 ~ 1998년 동국대학교 컴퓨터공학과 대학원 학과장
1985년 ~ 현재 동국대학교 컴퓨터공학과 교수
관심분야 : 컴파일러, 프로그래밍 언어, 모바일 컴퓨팅, 임베디드 시스템, 유비쿼터스.