제26회 한국정보처리학회 추계학술발표대회 논문집 제13권 제2호 (2006. 11)

퍼베이시브 컴퓨팅을 위한 가상 기계의 실행 엔진

박지우[0], 이창환, 오세만
동국대학교 컴퓨터공학과
e-mail : {jojaryong[0], yich, smoh}@dongguk.edu

Execution Engine of Virtual Machine for Pervasive Computing

Ji-Woo Park[0], Chang-Hwan Yi, Se-Man Oh
Dept of Computer Engineering, Dongguk University

요 약

퍼베이시브 컴퓨팅 환경을 실현하기 위해서는 작고 다양한 생활 용품에 장착할 수 있는 소규모 장치와 주변의 변화를 감지할 수 있는 센서 등을 사용한다. 이 소규모 장치들은 서로 다른 프로세서와 다양한 크기의 메모리를 가지고 있기 때문에 가상 기계 기술이 도입되어야 한다. 그러나 기존의 가상 기계들은 소규모 장치에 탑재하기에는 규모가 크기 때문에 퍼베이시브 컴퓨팅 환경에 적합한 소형 가상 기계의 실행 엔진이 필요하다.

본 논문에서는 임베디드 시스템을 위한 가상 기계인 EVM의 실행 엔진을 개선하여 퍼베이시브 컴퓨팅 환경에 적합한 실행 엔진을 설계 및 구현한다. 실행에 필요한 데이터를 줄이기 위해 SIL 명령어를 축약하고 불필요한 정보 및 객체지향 개념을 가진 명령어를 제거했다. 따라서 구현된 실행 엔진은 앞으로 구축될 퍼베이시브 컴퓨팅 환경을 위한 다양한 분야에 응용할 수 있다.

1. 서론

유무선 통신 기술과 소규모 장치 개발 기술이 발전하고 개개인의 생활수준이 향상되면서 퍼베이시브 컴퓨팅(Pervasive Computing) 환경에 대한 관심이 급증하고 있다. 퍼베이시브 컴퓨팅이란 언제 어디서나 아무런 제약 없이 각종 기기들을 사용할 수 있는 환경을 말한다. 이러한 환경을 실현하기 위해서 주변의 변화를 감지할 수 있는 센서와 다양한 생활 용품에 장착 가능한 반도체 집적회로를 이용한 소규모 장치들이 개발되고 있다[1].

환경 구축에 필요한 소규모 장치들은 서로 다른 프로세서와 다양한 크기의 메모리를 가지고 있다. 따라서 가상 기계 기술을 적용하여 이종의 장치로 프로그램을 이식하더라도 프로그램의 변경 없이 실행 가능한 환경이 필요하다.

대표적인 가상 기계는 JVM과 임베디드 시스템을 위한 가상 기계 등이 있다. 그러나 근래에 관심이 집중되고 있는 퍼베이시브 컴퓨팅 환경 구축을 위한 소규모 장치는 기존의 시스템 보다 더 적은 리소스만을 사용할 수 있다. 따라서 기존의 가상 기계를 적용하기에는 많은 제약이 따른다.

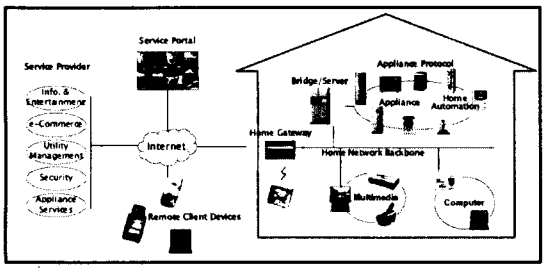
본 논문에서는 임베디드 시스템을 위한 가상 기계인 EVM(Embedded Virtual Machine)의 실행 엔진을 개선하여 퍼베이시브 컴퓨팅 환경에 적합한 실행 엔진을 설계 및 구현한다. 실행 엔진은 적은 리소스를 사용해야 한다. 따라서 실행에 필요한 리소스를 줄이기 위해 명령어를 축약하고 상수와 스트링 등을 제외한 불필요한 정보와 객체지향 개념을 가진 명령어를 제거한다.

2. 관련연구

2.1 퍼베이시브 컴퓨팅

퍼베이시브 컴퓨팅이란 컴퓨터가 자연스러운 인간

환경 속으로 들어가게 하는 것이다. 이러한 환경에서는 유무선 네트워크 접속 기능을 갖춘 컴퓨터와 네트워크와의 교신 능력을 가진 초소형 칩을 가전기기, 컵 등 대부분의 일상 사물에 사용한다. 따라서 우리는 단순히 컴퓨터와 상호작용을 하는 것이 아니라 생활 자체를 함께한다. 기계는 우리의 무의식적인 행동 속에서 세부 사항들을 처리할 수 있어야 한다[1]. 현재 이러한 기기들은 서로 다른 프로세서와 다양한 크기의 메모리를 가지고 있다. 또한, 저비용으로 소형화를 이루기 위해 제한된 리소스를 제공한다.

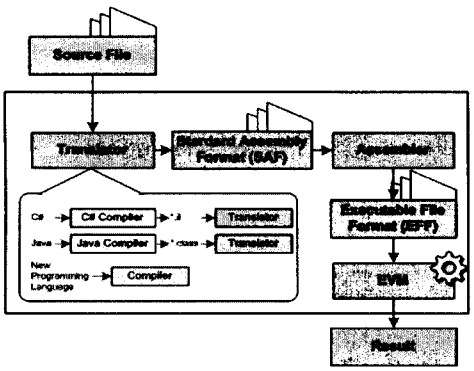


(그림 1) 퍼베이시브 컴퓨팅을 이용한 홈네트워킹

(그림 1)은 퍼베이시브 컴퓨팅을 적용하여 홈네트워크를 구성한 모습을 보여준다[2]. 가정에서 사용하는 냉장고나 전열기기, 가구 등에 초소형 칩을 장착하여 조작 없이도 적절한 기능을 수행한다.

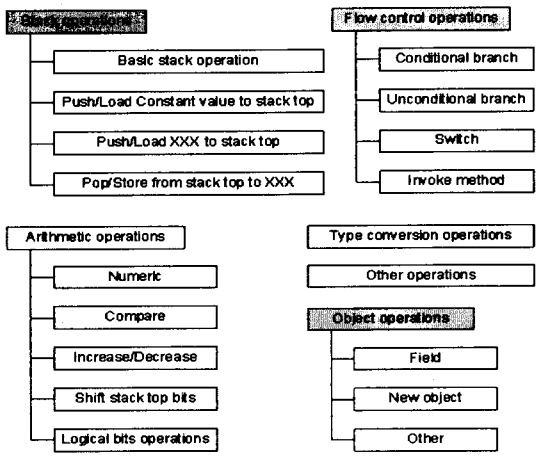
2.2 EVM

스택 기반의 가상 기계인 EVM은 PDA, 디지털 TV 등에 탑재되어 동적으로 응용프로그램을 다운로드하여 실행할 수 있다. (그림 2)는 EVM의 시스템 구성도를 나타낸 것이다[3][4][5][6].



(그림 2) EVM의 시스템 구성도

EVM의 중간 언어인 SIL(Standard Intermediate Language)은 스택 기반의 명령어 집합으로 언어 독립성과 하드웨어 및 플랫폼 독립성을 갖고 있다. SIL은 다양한 프로그래밍 언어를 수용하기 위해서 바이트코드, .NET IL 등 기존의 가상 기계 코드들의 분석을 토대로 정의되었다. 또한, 프로그램의 확장성을 위해 순차적 언어와 객체지향 언어를 모두 수용할 수 있도록 설계되었다. (그림 3)과 같이 SIL은 총 6개의 연산 카테고리로 나눌 수 있다.



(그림 3) SIL의 연산 카테고리

고급 언어로 작성된 코드는 변환기를 통해서 의사코드와 연산코드로 구성된 EVM의 어셈블리 포맷인 SAF(Standard Assembly Format)로 변환된다. SAF는 클래스 기반의 구조를 가진다. (그림 4)는 SAF 문법의 일부분을 보여준다[4].

```
<saf_program> ::= { <class_dcl> }
<class_dcl> ::= '.class' <modifiers> <class_name>
                ['.' <super_class_name>]
                  '.bgn' <class_body> '.end'
<class_body> ::= <member_dcl> { <member_dcl> }

<*_name> ::= '%ident'
<*_number> ::= '%number'
<*_op> ::= <opcode_name> [ <param_name> ]
```

(그림 4) SAF 문법의 일부분

SAF는 어셈블러를 통해 EVM 상에서 실행 가능한 EFF(Executable File Format) 파일로 변환된다.

EFF는 실행에 필요한 모든 정보를 가진 EVM의 실행 파일 포맷으로서 이진 형태의 스트림으로 구성

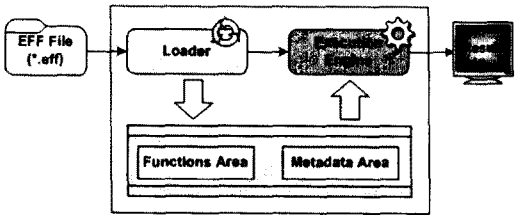
되어 있다. EFF 파일을 구성하고 있는 아이템은 헤더, 코드, 메타데이터의 3부분으로 나눌 수 있다. 헤더 부분은 EFF 파일의 버전, 프로그램의 이름, 시작점 등의 정보를 가지고 있다. 코드 부분은 실질적인 명령어들을 포함하고 있으며 메타데이터 부분은 실행시 필요한 모든 정보를 저장한다. (그림 5)는 EFF 파일의 전체 구조를 보여준다[5][6].

```
EFF_File {
        U4 magic;               // Header
        U2 majorVersion;
        U2 minorVersion;
        U4 module;
        U4 language;
        U4 entryPoint;
        U4 VMCodesCount;  // Code
        VMCodeInfo VMCode[VMCodesCount];
        U4 metadataCount;  // Metadata
        MetadataInfo Metadata[metadataCount];
}
```

(그림 5) EFF 파일 구조

3. 가상 기계의 실행 엔진

실행 엔진에 대한 설명에 앞서 EFF 파일의 실행 과정에 대해 살펴보면 (그림 6)과 같다. 로더는 EFF 파일로부터 데이터를 검증하고 함수와 메타데이터 영역에 적재한다. 실행 엔진은 함수 영역과 메타데이터 영역의 정보를 가지고 명령어들을 수행한다[7].



(그림 6) EFF 파일의 실행 과정

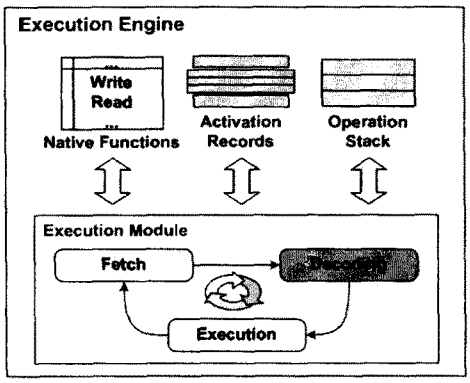
3.1 실행 엔진의 특징

실행 엔진은 퍼베이시브 컴퓨팅 환경을 위한 소규모 장치에 적합하게 적은 리소스를 사용하여 원활한 수행이 가능하도록 설계한다. 따라서 EVM에서 제공하는 명령어중 사용자 편의를 고려한 연산 코드와 불필요한 객체 연산 코드를 제거한다. 그러나 EVM과의 호환성을 위하여 프로그램은 클래스 단위로 구성한다. 또한, 실행 엔진은 최소한의 정보만으로 실행이 가능하게 설계되어 있다. 따라서 버전과 프로그램의 이름 등과 같은 불필요한 정보는 모두 제거한다. 실행시 흐름 제어와 같이 오프셋에 대한 연산이 필요한 명령어의 실행에는 오버헤드를 최소화하는 알고리즘을 적용한다. 이러한 최적화의 결과로 실행 엔진은 제한된 리소스를 제공하는 환경에서 사용할 수 있다.

3.2 시스템 구성도

실행 엔진은 (그림 7)과 같이 실행 모듈, 내장 함수, 활성 레코드, 연산자 스택으로 구성된다. 명령어는 페치, 디코딩, 실행의 수행 과정을 거친다[7][8][9].

함수 영역으로부터 명령어 코드를 가져온 후 정상적인 코드인 경우 명령어를 디코딩하게 되는데 이 명령어는 스택 연산, 산술 연산, 흐름 제어, 함수 호출 등으로 나눌 수 있다.



(그림 7) 실행 엔진의 구성도

실행 단계에서 스택 연산과 산술 연산은 연산자 스택을 이용하여 처리하며, 지역 변수에 대한 정보가 필요할 경우 활성 레코드를 참조하게 된다. 흐름 제어 명령은 프로그램 카운터를 해당 오프셋으로 설정한다. 함수 호출에 관련된 명령이 발생하면, 현재의 프로그램 카운터와 지역 변수 및 호출된 함수를 위한 파라미터 값을 활성 레코드에 저장하고 호출된 함수로의 분기가 이루어진다. 호출된 함수의 모든 명령을 수행한 후 리턴 값을 활성 레코드에 저장하며, 활성 레코드에 이미 저장된 프로그램 카운터를 이용하여 함수 호출전의 위치로 복귀한다.

명령어 수행시 추가적인 정보가 필요한 경우 메타데이터를 참조하여 실행을 하고 명령어 코드를 모두 수행할 때까지 사이클을 반복적으로 수행한다.

3.3 자료 구조

실행에 필요한 최소한의 정보는 함수 영역과 메타데이터 영역에 저장한다. 함수 영역에는 SIL 명령어들을 순차적으로 저장하고 메타데이터 영역에는 상수와 스트링 등의 데이터들을 저장한다.

실행 엔진은 명령어들을 처리하기 전에 활성 레코드와 연산자 스택을 생성한다. 활성 레코드는 지역변수와 함수 호출시 필요한 정보인 파라미터와 리턴값 그리고 프로그램 카운터(PC) 등을 저장한다. 연산자 스택은 연산시 필요한 피연산자와 연산의 결과를 저장한다.

4. 실험 및 결과

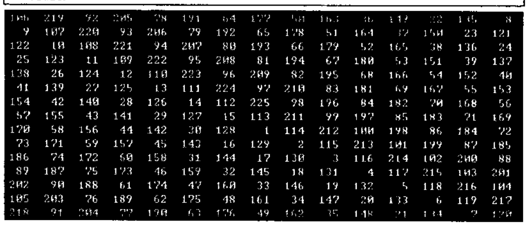
4.1 구현 및 실험 환경

실행 엔진은 MS Windows XP 환경에서 ANSI-C 호환 컴파일러인 Visual C++ 6.0을 사용하여 구현하였다.

4.2 실험 결과

실험시 사용한 다양한 프로그램 중 (그림 8)은 마방진을 구하는 코드와 실행 결과를 보여준다.

```
.class public Magic
.bgn
    .method public static void main()
    .bgn
        .locals(int matrix[225], int i, int j, int s, ...)
            ldc.i   1
            str     s
            ldc.i   15
            str     size
            ldl     size
            ldc.i   2
            div
            str     center
            ldc.i   0
              ⁝
    .end
.end
```



(그림 8) 마방진 프로그램과 실행 결과

5. 결론 및 향후 연구

본 논문에서는 임베디드 시스템을 위한 가상 기계인 EVM의 실행 엔진을 개선하여 소규모 가상 기계를 위한 실행 엔진을 설계 및 구현하였다. 실행에 필요한 데이터를 최소화하기 위해 EVM에서 제공하는 명령어중 불필요한 연산 코드와 객체 관련 코드를 제거했다. 또한, 버전과 프로그램의 이름 등과 같은 부수적인 메타데이터의 사용을 최소화했다. 따라서 구현된 실행 엔진은 앞으로 구축될 퍼베이시브 컴퓨팅 환경을 위한 다양한 분야에 응용할 수 있다.

향후 과제로 실행 엔진과 어셈블러, 디스어셈블러, 로더 등을 통합한 퍼베이시브 컴퓨팅 환경을 위한 가상 기계 플랫폼 개발에 대한 연구가 필요하다. 또한, 명령 실행 알고리즘 개선, 실행시 필요한 데이터의 최적화 등과 같은 작업을 통해 실행 엔진의 성능 향상과 내장 함수 연결 기법 개선이 필요하다.

참고문헌

[1] Mark Weiser, “The computer for the 21st century,” Scientific American, pp.94~104, 1991.

[2] Jong Hoon Chung, Daé Sung Wang, Sang Kyun Lee, Sun Mi Han, Young Hoon Roh, Min Seok Kang, “Development of LnCP based Home Network System by using high level message between heterogeneous application software,” ICCAS2004, pp.903~907, 2004.

[3] 오세만, 이양선, 고광만, “임베디드 시스템을 위한 가상기계의 설계 및 구현”, 한국멀티미디어학회 논문지, 제 8권, 제 9호, pp.1282~1291, 2005.

[4] 손윤식, 오세만, “실행 파일 포맷 생성기의 설계 및 구현”, 한국정보처리학회 추계학술발표대회 논문집, 제 11권, 제 2호, pp.623~626, 2004.

[5] 정한종, 오세만, 임베디드 가상 기계를 위한 실행 파일 포맷, 동국대학교 석사학위논문, 2004.

[6] 지정환, 오세만, EVM 파일을 위한 시각화 브라우저, 동국대학교 석사학위논문, 2004.

[7] Tim Lindholm, Frank Yellin, The Java™ Virtual Machine Specification, Second Edition, Addison Wesley, 1999.

[8] 김신덕, 박기호, 맹혜선, 강두진, 이영미, Embedded System용 JVM 개발 및 기술 분석에 관한 연구 보고서, 한국전자통신연구원, 1998.

[9] 오세만, 컴파일러 입문(개정판), 정익사, 2006.