시스템 레벨 설계를 위한 소프트웨어 기능 블록의 시뮬레이션 기반 성능 예측 방법.

권성남° 오현옥 하순회
서울대학교 전기 컴퓨터 공학부
{ksn°, oho, sha}@iris.snu.ac.kr

Simulation-driven Performance Estimation of Software Function Blocks for System Level Design

Seong-nam Kwon° Hyun-ok Oh Soon-hoi Ha
The School of Electrical Engineering and Computer Science, Seoul National University

요 약

이 논문에서 우리는 각 기능 블록의 성능 분석 방법을 제안하고 어떻게 하드웨어와 소프트웨어의 합성을 위한 기능 블록의 성능을 기록한 데이터베이스를 구축하는지를 설명하겠다. 기능 블록의 성능을 예측하는 것은 초기 설계 단계에서 주어진 제약을 만족시키기 위해 어떤 기능 블록이 개선되어야 할지 결정하는 기준을 제시하기 때문에 내장형 시스템의 합성에 있어서 중요하다. 예측하는 도구로 측정에 시간이 많이 걸리지만 정확한 명령어 수준 시뮬레이터(ISS : instruction set simulator)를 사용하였다. 데이터베이스를 구축하는데 있어선 각 기능 블록을 요소(factor)라 부르는 다른 상태를 두어서 차별화하였다. 제안한 예측 방법은 개발중인 통합설계 환경에 구현되었으며 H.263 인코더에 적용하여 0.03% 이내의 오차를 얻었다.

1. 서 론

시스템 레벨 설계과정에 있어서 중요한 단계가 수많은 서로 다른 구조상에서의 성능을 예측하는 것이다. 여기서 성능은 수행 시간, 전력 소모량, 메모리 크기 등을 포함한다. 우리는 시스템의 동작이 기능 블록들의 조합으로 모델링 된다고 가정하였다. 이 기능 블록들은 주어진 구조에서 요소들을 분할하는 단위가 된다.

이 논문의 주제는 각 기능 블록의 성능을 예측하고 그것을 설계를 위한 데이터베이스에 저장하는 것이다. 여기서 얻어진 정보들은 사용할 블록의 선정과 시스템 분할의 결정에 사용될 수 있다. 또한 분할이 된 이후에 별다른 과정 없이 시스템의 성능을 예측하는데 이용된다. 단순히 각 기능 블록의 성능들을 간단한 선형식에 대입하여 시스템의 전체 성능을 예측할 수 있으며 어떤 블록을 개선하는 것이 전체 시스템 성능을 개선하는데 효과적인지도 이 정보를 통해 쉽게 알 수 있다.

우리는 각 기능 블록의 성능을 예측하는데 있어서 두 가지 요구사항을 설정하였다. 프로세서나 컴파일러 등에 대한 어떠한 가정도 없이 적용 가능한 방법이어야 된다는 것과 사용된 데이터에 따라 달라지는 성능의 변화를 고려한 방법이어야 한다는 것이 그것이다. 첫 번째 요구사항을 위해 아무런 가정 없이 대부분의 컴파일러에서 사용 가능한 기법을 사용하였으며 두 번째 요구사항을 위해 성능에 변화를 줄 수 있는 데이터의 변화를 요소(factor)라는 인자를 두어 구분할 수 있도록 하였다.

또한 측정에 있어서 각 기능 블록의 성능을 따로 측정하는 것이 아니라 전체 시스템을 한번에 측정하여 각 블록의 성능을 예측하는 방법을 사용하였다. 이렇게 함으로서 측정의 횟수를 줄일 수 있으며 각 블록간의 관계를 고려한 측정이 가능해진다. 또한 시스템에 들어가는 입력만 실제 사용되는 입력을 준다면 시스템에 사용된 기능 블록에 들어가는 모든 입력도 실제 사용되는 입력이라는 결과를 가져오므로 실제적인 결과를 얻기가 용이하다.

2. 관련된 연구들

산업적 설계에서 일반적으로 사용되는 소프트웨어의 성능 예측 방법은 명령어 수준 시뮬레이터(ISS : instruction set simulator)와 같은 시뮬레이터를 사용한 시뮬레이션이다. 이 방법은 정확하지만 느리기 때문에 설계 단계에서 반복적으로 사용되기는 부적합하다. 반면 우리의 방법은 전체 시스템의 수행 시간을 예측하는 것이 아닌 기능 블록별 수행 시간을 측정하므로 이미 데이터베이스가 구축되어 있다면 설계 단계에서 [1]과 같이 각 기능 블록의 성능을 더함으로 간단하게 성능을 예측할 수 있다.

다른 접근 방법은 컴파일된 시뮬레이션이다[2]. C 소스 코드나 어셈블리 수준에서 측정을 위한 코드를 삽입하여 측정한다. 후자의 경우는 컴파일러 최적화까지 고려가 되므로 컴파일러의 최적화를 고려할 필요가 없다. 이 경우는 프로세서의 마이크로 아키텍처에 따른 영향을 고려해야 하며, 이것이 바뀔 때마다 정보도 수정이 되어야 한다.

실시간 시스템 설계에 있어서 소프트웨어의 성능 예측은 최대 수행시간을 찾기 위한 프로그램의 정적인 분석을 통해 이루어진다. 프로그램은 기본 블록과 그것들이 연결된 컨트롤 플로우로 구성된 콘트롤 플로우 그래프(CFG)로 모델링 된다. 각 기본 블록의 비용 정보가 주어지면 가능한 경로의 가장 긴 경로를 찾을 수 있다[3][4]. 각 블록 간에 영향을 주기 때문에 가능한 경로를 찾는 것은 매우 어렵다.

우리의 접근 방법은 두 가지 다른 성능 예측 방법을 사용한다. 각 기능 블록의 성능을 측정하기 위해 시뮬레이션에 의한 방법을 사용하며 전체 시스템을 측정하기 위해선 기능 블록의 수행최대 수행시간 순서를 결정하는 분석 방법을 사용한다.

3. 제안하는 설계 공간 탐색 프레임웍

우리의 프레임웍[5]에서 시스템은 기능 블록과 그것간의 통신을 위한 연결로 모델링된다. 프레임웍 내에는 기능 블록에 대한 정보와 성능을 저장하는 데이터베이스가 존재하며 각 기능 블록들은 C 언어로 기능이 최적화되어 기술된다.

시스템을 설계하기 위해서 먼저 시스템을 블록간의 관계로 기술한다. 그 뒤 사용할 구조를 선정하고 분할 알고리즘을 수행한다. 이 과정에서 이미 저장되어있던 각 기능 블록의 성능 정보를 이용한다. 우리의 프레임웍에서 시스템은 정형화된 계산 모델을 사용하여 기술되기 때문에 정적으로 스케쥴링 정보와 각 기능 블록이 몇 번 수행되는지 등의 정보가 자동으로 생성된다. 스케쥴링 정보를 사용하여 각 기능 블록의 최대 수행 시간과 사용된 횟수를 곱한 값을 계산하여 전체 기능 블록에 대한 이 값을 합한 것이 전체 시스템의 최대 수행 시간이 된다. 이것을 사용하여 성능 예측을 하며 만약 제약 조건을 만족시키지 못한다면 다른 기능 블록을 사용하거나 분할을 다르게 하는 등 위의 과정을 반복한다.

각 기능 블록의 성능을 예측하기 위해 우리의 프레임웍 내에서 그림 1과 같은 과정을 거치게 된다. 먼저 기능 블록들을 조합하여 시스템을 명세한 뒤 생성된 코드에 타이밍을 위한 코드를 삽입한 뒤 컴파일 된다. 그 뒤 ISS를 사용하여 시뮬레이션을 한 뒤 각 블록에 대한 성능 정보를 뽑아내서 데이터베이스를 갱신하게 된다. 여기서 각 플랫폼에 맞는 컴파일러를 사용하여 컴파일된 코드를 시뮬레이션에 사용하기 때문에 이 방법은 플랫폼에 독립적으로 적용이 가능하다.

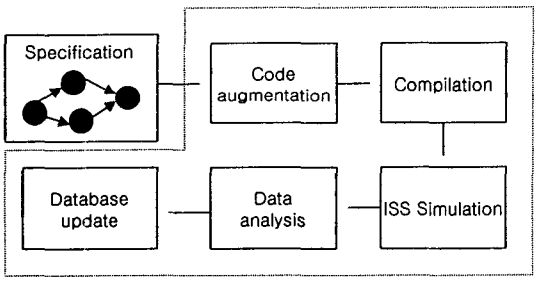

그림 1 각 기능 블록의 성능을 측정하고 데이터베이스를 구성하기 위한 흐름도.

4. 기능 블록의 성능 예측 방법

기능 블록의 성능을 측정하기 위해 제시한 우리의 접근 방법은 ISS를 기반으로 한다. 이 방법은 느리지만 정확한 측정이 가능하며 각 기능 블록의 정보를 데이터베이스에 저장하고 있다면 실제 성능을 예측할 때는 빠르고 간단하게 예측할 수 있게 된다. 우리가 방법에서 각 기능 블록의 성능은 독립적으로 측정되지 않고 전체 시스템 안에서 한번에 측정이 된다. 이 방법을 사용함으로써 시스템의 입력만 실제적인 데이터를 사용한다면 내부적으로 사용된 전체 기능 블록에 실제적인 데이터를 입력한 효과가 나며 시스템 안에서 블록간의 의존성이 자연적으로 고려된다. 그러나 이 접근 방법을 사용하기 위해선 전체 시스템의 성능을 측정한 정보에서 각 기능 블록의 성능 정보를 정확하게 분리해 내는 방법이 필요하다. 이것을 위해 간단한 기법을 사용하였다. 그림 2에 나타난 코드 삽입 방법이 바로 그것이다.

```
extern int Start_func(Block name, Factor value);
extern void End_func();
...
If(Start_func(Blcok name, Factor value)==true) {
/* the function block */
}
End_func();
...
```

그림 2 전체 시스템의 성능에서 각 기능 블록의 성능 정보를 분리하기 위한 코드 삽입 기법

이 방법의 기본적인 개념은 기능 블록을 ISS에서 내부적으로 멈춤 지점(break point)으로 잡기 위한 내용이 없는 함수로 둘러싸는 것이다. 대부분의 ISS는 특정 함수를 기점으로 프로그램 수행을 잠시 멈추는 기능을 지원하며 멈춘 지점을 기준으로 "Start_func()"과 "End_func()"과의 수행 싸이클(Cycle)수의 차나 프로그램 카운터의 차를 이용하면 간단하게 기본블록의 기본적인 성능을 얻어낼 수 있다. "Start_func()"과 "End_func()"이라는 두 함수가 그 기능을 한다. 이 두 함수는 기본적으로 아무 기능이 없다. "End_func()"은 어떠한 역할도 없이 단순히 "return"을 해주며 "Start_func()"은 무조건 1을 반환한다. 따라서 기능 함수의 앞부분에 붙은 "Start_func()"의 값이 인자로 사용된 "If"문은 무조건 실행된다.

기능 함수를 "If"문으로 둘러싼 이유는 컴파일러가 임의로 기능 블록의 내용을 옮기지 못하도록 막기 위해서이다. 만약 컴파일러가 컴파일 중 기능 블록안의 내부 내용을 다른 기능 블록안의 내용과 섞거나 "Start_func()"과 "End_func()이 아무 기능도 하지 않는 함수이므로 제거해도 된다고 판단한다면 이 두 함수를 멈춤 지점으로 사용하는 이 방법은 사용할 수 없게 된다. 그러나 "Start_func()"의 내부 명세가 "extern"을 사용하여 다른 파일에 저장되어 있기 때문에 컴파일 단계에서 컴파일러는 "Start_func()"이 어떤 값을 반환할지 알지 못하며 따라서 기능 블록의 내용을 임의로 옮기거나 두 함수를 제거하지 못하게 된다. 이 두 함수의 내부 코드는 다른 파일에 명세 되며 두 파일이 독립적으로 컴파일 된 뒤 링크 단계에서 합쳐지게 된다.

이렇게 얻어진 성능은 실제와 다를 수 있다. 그러나 실험에 의한 데이터에 의하면 그 차이가 크지 않으며 기능 블록의 크기가 상당히 크고, 기능 블록 내에선 컴파일러가 최적화를 하기 때문에 실제 오차는 그다지 크지 않다.

"Start_func()"은 기능 블록의 이름과 요소를 내부 인자로 받게 된다. "Start_func()"은 인자로 받은 기능 블록의 이름과 요소를 출력해 주며 이 출력 정보는 나중에 데이터베이스 구축을 위한 정보를 자동으로 뽑아내는데 사용된다. 즉 현재 얻어낸 성능 정보가 어떤 기능 블록에 대한 것인지 자동으로 알 수 있는 근거가 된다.

5. 데이터베이스 구축 방법

그림 3은 데이터베이스를 어떻게 구축하는지를 보여준다. 먼저 통합설계 프레임웍이 멈춤 지점을 위한 함수가 기능 블록간에 삽입되어 있는 코드를 ISS로 컴파일해서 보내면 ISS가 그 파일을 시뮬레이션한 뒤 결과 파일을 작성한다. 이 결과 파일에는 기본 블록의 이름과 요소, 그리고 성능에 대한 정보가 포함된다. 그 뒤 프레임웍에서 그 정보를 받아 데이터베이스를

갱신하게 된다.

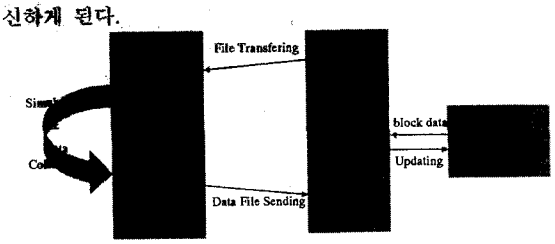

그림 3 통합설계 프레임웍과 ISS, 데이터베이스간의 관계

여기서 요소는 같은 기능을 하는 기능 블록들을 입력에 따라 구분하는 요인이 된다. 예를 들어 덧셈기의 경우 입력이 2개인 것과 3개인 것은 수행 시간에 차이가 생긴다. 또한 입력 비트 수의 차이도 마찬가지이다. 이렇듯 각 기능 블록에서 성능에 차이를 둘 수 있는 것들을 요소로 묶어서 구분한다. 이렇게 묶어서 데이터베이스에 보관할 경우 기능 블록의 이름과 요소가 같은 경우 대략적으로 성능이 비슷할 것으로 예측할 수 있다.

6. 실험 및 결과

제시한 기본 블록 성능 예측 방법을 H.263 인코더에 적용한 결과는 표1과 같다. 여기서 입력으로는 QCIF 크기의 동영상 10프레임을 사용하였으며 프로세서는 StrongARM, ISS로는 Armulator를 사용하였다.

Block name	Count	WCET	Total time	Percentage(%)
Variable length coding	990	26018	3260196	1.62
Motion Compensation	990	11356	8241654	4.10
IDCT	5940	2914	3932872	1.96
Quantization	5940	3350	19699617	9.80
DCT	5940	5059	29971695	14.92
Motion Estimation	990	382419	112034511	55.76
Etc.			23795417	11.84
Total	10		200935962	100.00

표 1 H.263 인코더에서 사용된 기능 블록의 성능측정 결과

여기서 WCET는 최대 수행 시간(Worst case execution time)을 의미하며 Count는 10프레임을 인코딩하는 동안 이 블록이 몇 번 사용되었는가를 의미한다. 시간의 단위는 CPU의 싸이클 수이다. 이 경우 Motion Estimation 블록이 55.76%의 수행시간을 차지하므로 H.263 인코더의 성능개선을 위해선 Motion Estimation 블록을 개선하던지 하드웨어로 분리하는 것이 효과적이라는 것을 알 수 있다.

시스템의 WCET는 다음의 식으로 간단하게 구할 수 있다.

$$\text{Time} = \Sigma(\text{WCET} * \text{Count})$$

H.263 인코더를 멈춤 지점을 위한 함수 삽입을 하지 않고 전체를 Armulator를 사용하여 측정한 결과는 200879320으로, 각 블록의 평균 시간을 사용하여 총 시간을 예측했을 경우 200935962로 단지 0.0282% 크게된다. 각 블록의 WCET를 사용한 경우 506649107이 되어 상당히 큰 차이를 보이게 된다. 만약 보다 근접한 성능을 알고 싶다면 각 기본 블록의 평균 성능을 이용하면 되며, 안전한 WCET를 구하길 원한다면 각 기능 블록의 WCET를 사용하면 된다.

7. 결론

우리는 기능 블록의 성능을 예측하고 그 정보를 저장하는 접근방법을 제시하였다. 또한 이 방법에 두 가지 요구사항을 정하였는데 측정을 위해 어떠한 플랫폼이나 컴파일러에 대한 가정도 없어야 된다는 것과 입력되는 데이터의 차이에 따라 달라지는 기능 블록의 성능의 차이가 반영되어야 된다는 것이 그것이다.

첫 번째 요구사항을 위해 컴파일된 코드를 ISS를 사용하여 측정하는 방법을 사용하였다. 코드엔 각 기능 블록을 구분 지을 수 있는 간단한 기법이 도입되었다. 두 번째 요구사항을 위해 데이터베이스 구축 시 각 기본블록마다 요소라는 인자를 추가하였다.

H.263 인코더에 적용하여 얻은 측정 결과를 살펴보면 매우 정확하게 결과를 얻을 수 있음을 알 수 있다. 또한 결과를 통해 어떤 블록이 전체 시스템에 주요하게 성능을 결정하는지도 알 수 있어 시스템의 성능 개선과 분할에 있어서 유용하게 사용될 수 있음을 알 수 있다.

8. 참고 문헌

[1] S. Malik, M. Martoonosi, and Y. Li. Static timing analysis of embedded software. In *Proc. Design Automation Conf.*, pages 147-152, Jun. 1997.

[2] K. Suzuki and A. Sangiovanni-Vincentelli. Efficient software performance estimation methods for hardware/software codesign. In *Proc.Design Automation Conf.*, pages 605-610, Jun. 1996.

[3] F.Stappert. Predicting pipelining and caching behaviour of hard real-time programs. 1998. C-LAB internal document, Furstenalle 11, D-333102 Paderborn, Germany.

[4] V. Zivojnovic and H. Meyr. Compiled hw/sw co-simulation. In *Proc.Design Automation Conf.*, 1996.

[5] PeaCE(Ptolemy extension as Codesign Environment) official home page http://peace.snu.ac.kr/research/peace/