ISSN(Print): 2287-5891

http://dx.doi.org/10.3745/KTCCS.2013.2.2.059

An Optimal ILP Algorithm of Memory Access Variable Storage for DSP in Embedded System

Jeong-Uk Chang† · Chi-Ho Lin††

ABSTRACT

In this paper, we proposed an optimal ILP algorithm on memory address code generation for DSP in embedded system. This paper using 0-1 ILP formulations DSP address generation units should minimize the memory variable data layout. We identify the possibility of the memory assignment of variable based on the constraints condition, and register the address code which a variable instructs in the program pointer. If the process sequence of the program is declared to the program pointer, then we apply the auto-in/decrement mode about the address code of the relevant variable. And we minimize the loads on the address registers to optimize the data layout of the variable. In this paper, in order to prove the effectiveness of the proposed algorithm, FICO Xpress-MP Modeling Tools were applied to the benchmark. The result that we apply a benchmark, an optimal memory layout of the proposed algorithm then the general declarative order memory on the address/modify register to reduce the number of loads, and reduced access to the address code. Therefor, we proved to reduce the execution time of programs.

Keywords : DSP, Embedded System, ILP, Meomory Access

임베디드 시스템에서 DSP를 위한 메모리 접근 변수 저장의 최적화 ILP 알고리즘

장 정 욱† · 인 치 호††

요 약

본 논문에서는 임베디드 시스템에서 DSP를 위한 메모리 접근 변수의 저장 방법에 대한 최적화 ILP 알고리즘을 제안하였다. 본 논문은 0-1 ILP 공식을 이용하여 DSP 주소 생성 유닛의 메모리 변수 데이터 레이아웃을 최소화한다. 제약 조건을 기반으로 변수의 메모리 할당 여부를 식별하고, 변수가 지시하는 주소코드를 프로그램 포인터에 등록한다. 프로그램의 처리 순서가 프로그램 포인터에 선언되면, 해당 변수의 주소코드에 대한 자동증감 모드를 적용한다. 주소 레지스터에 대한 로드를 최소화하여 변수의 데이터 레이아웃을 최적화한다. 본 논문에서 제안한 알고리즘의 효율성을 입증하기 위하여 FICO Xpress-MP Modeling Tools을 이용하여 벤치마크에 적용하였다. 벤치마크 적용 결과, 기존의 선언적 주문 메모리 레이아웃보다 제안한 알고리즘을 적용한 최적의 메모리 레이아웃이 주소/수정 레지스터에 대한 로드 수를 감소시켰고, 주소코드의 접근을 줄임으로써, 프로그램의 실행 시간을 단축시켰다.

키워드 : DSP, 임베디드 시스템, ILP, 메모리 접근

1. 서 론

최근에 많은 임베디드 시스템이 메모리 공간 제약으로 인하여 임베디드 어플리케이션에서 요구하는 메모리 주소 코드 공간에 대한 요구 사항을 줄이는 것이 중요한 문제점으로 대두되고 있다. 컴퓨터 및 회로 설계 수준의 발달로 인하여 한정된 영역에 추가할 수 있는 메모리도 대용량에 맞게 설계되지만, 메모리를 사용하는 응용 프로그램 크기와 복잡도의 증가는 지금까지의 임베디드 시스템을 위한 메모리 용량의 증가치를 초과한다[1-3]. 이러한 문제점을 해결하기 위하여 최근에 메모리 주소 코드 공간의 요구 사항을 줄이는 몇 가지 방법을 제안하였다[4].

첫 째, 압축 기반의 접근 방식을 채용하여 응용 프로그램 코드의 일부를 메모리에 압축된 형태로 보관하고, 필요한 경우에만 해제하여 사용하는 방법이다. 이 방법은 하드웨어

† 준 회 원 : 세명대학교 전산정보학 박사과정
†† 정 회 원 : 세명대학교 컴퓨터학부 교수
논문접수 : 2012년 8월 2일
수 정 일 : 1차 2012년 11월 30일
심사완료 : 2013년 1월 7일
* Corresponding Author : Chi-Ho Lin(ich410@semyung.ac.kr)

자원 운영 기법 및 소프트웨어 기반의 접근 방식을 모두 고려해야 하는 문제점이 존재한다. 둘 째, 메모리 주소 코드가 사용하는 공간에 따른 코드 점유율을 통제하여 유지할 수 있도록 코드 생성에 제약 조건을 설정하는 방법이다. 이 방법은 현재 DSP(Digital Signal Processor) 아키텍처를 위한 주소코드 생성 방법으로 이용되고 있다[5,6].

대부분의 DSP는 주요 실행 유닛들과 병렬로 작동할 수 있는 주소코드 생성 유닛(Address-code Generation Unit, AGU)을 포함하고 있다. AGU는 접근 데이터 구성에 따른 주소코드를 보관하는 주소 레지스터를 탑재하였고, 특정 명령어와 함께 주소 값에 대한 자동증감 연산을 병렬로 수행할 수 있다. 이러한 구성은 주소 지정에 필요한 레지스터에 걸리는 로드 수를 감소시켜 주소코드로 접근을 줄임으로써 응용 프로그램의 실행 시간을 줄이고, 시스템-온-칩의 병렬성을 증가시킨다. 하지만, 주소코드 메모리의 통제가 빈번히 발생하게 되면 응용 프로그램에서 사용되는 모든 변수에 대한 데이터 레이아웃 할당을 일일이 고려하여 계산한 후 다시 요청해야 하는 문제점이 발생한다[6].

본 논문에서는 DSP 기반의 주소코드 생성에 따른 자동증감 연산을 고려하여 응용 프로그램에서 사용되는 변수의 데이터 레이아웃을 할당할 때 발생되는 문제점을 해결하기 위하여 ILP (Integer Linear Programming) 기반의 메모리 주소코드 생성에 대한 최적의 접근 방식을 제안한다.

본 논문의 구성은 다음과 같다. 제 2장에서는 DSP의 AGU 동작 상태를 분석하고, ILP 기반의 메모리 주소코드 생성에 대한 최적의 접근 방법에 대하여 기술한다. 3장에서는 본 논문에서 제안한 ILP 알고리즘을 적용한 벤치마크 결과값을 기존의 방법과 비교 실험하여 평가한다. 끝으로 4장에서 결론을 맺도록 한다.

2. AGU 동작에 관한 ILP 기반의 접근 방법

2.1 AGU 동작에 관한 ILP 기반의 접근 방법

AGU는 주소코드 생성을 위한 기본적인 주소 레지스터와 새로운 주소코드를 로드하고 해당 내용을 수정할 수 있는 권한을 가지고 수정된 코드를 처리하는 수정 주소 레지스터로 구성되어 있다. Fig. 1은 DSP 내부에서 채용한 AGU을 나타낸 것이다.

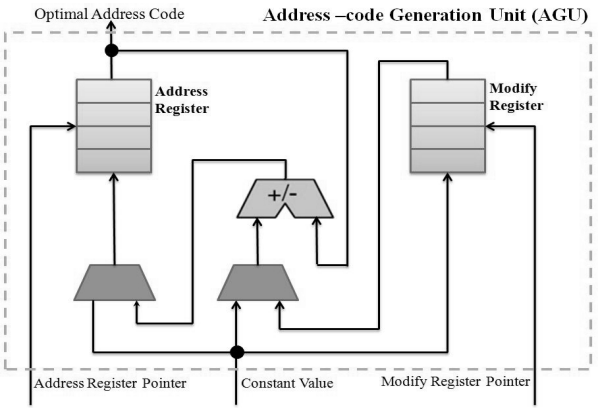
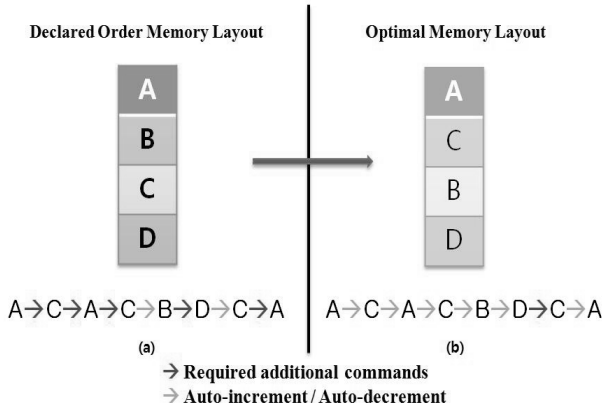


Fig. 1. The address-code generation unit(AGU) in DSP

주소 레지스터들은 주소코드에 대한 자동증감 연산이 ALU를 통해 명령어 형식으로 들어갈 수 있지만, 수정 레지스터를 포함한 주소코드 계산이 ALU 연산과 공존하기에 ALU의 기본 연산이 수행될 동안에는 지연 현상이 발생할 수 있는 문제점이 존재한다. 하지만 자동증감 연산 모드를 사용하지 않고 레지스터를 수정하여 사용하면 컴파일러가 해당 주소를 등록하여 사용할 수도 있기 때문에 다음 메모리 접근이 일어나기 전에 주소 레지스터의 내용을 수정할 지침을 고려해야 한다. 이러한 문제점을 해결하기 위해서는 실행되는 응용 프로그램의 런타임에 별도의 실행 사이클을 추가적으로 삽입해야 한다. 따라서 AGU의 효과적인 사용은 모든 주소코드의 크기 및 실행 사이클에 대한 감소효과를 가져오기 때문에, 본 논문에서는 주소코드 크기 감소에 대한 접근 방법을 ILP 공식을 이용하여 제안한다.

메모리 공간에 변수 레이아웃을 지정하는 과정에서 주소 레지스터의 자동증감 횟수의 변화에 미치는 영향을 알아보기 위하여 예제 시나리오를 구현해 보도록 한다. 하나의 응용 프로그램 안에 선언된 변수 A, B, C, D가 있다고 가정하자.



(a) Declarative order (b) Optimum order
Fig. 2. Memory layout for variables in AGU

Fig. 2에서 보는 것과 같이, 응용 프로그램의 동작 과정에서 유도한 변수에 대한 액세스 순서가 A-C-A-C-B-D-C-A라고 한다면, Fig. 2a는 선언적 주소코드 레이아웃을 나타내며, Fig. 2b는 특정 액세스 순서에 따른 최적의 레이아웃을 나타낸다. Fig. 2a와 같이 주소코드가 저장되면, 실제 주소 레지스터에 선언적 저장 순서는 다섯 번의 추가 로드에 대한 지침이 실행되고 난 후, 자동증감 모드를 사용할 수 있게 될 것이다. 하지만 Fig. 2b와 같이 변수의 순서가 저장되면, A, C, B 및 D는 다음 동일한 순서를 액세스하는 것으로 하나의 명령만 실행된 것이고, 그 이후는 자동증감 모드를 사용한다. 즉, 이 액세스 순서에 대한 접근은 주소 레지스터의 주소코드 접근 횟수와 빈도수를 감소시킴

으로써 수정 레지스터에 대한 선언적 주소코드 지정에 걸리는 로드를 감소시키는데 도움을 줄 수 있다. 예제를 통해 하나의 비트 B를 기준으로 하여 (-1, 0, 1)의 범위를 가지는 주소코드 접근 지정 방법을 사용하면 자동증감 모드의 범위는 $(-2^{B-1}, 2^{B-1})$로 정의할 수 있다.

2.2 최적의 메모리 접근을 위한 ILP 공식화

AGU의 주소 레지스터에 응용 프로그램의 변수 데이터 레이아웃을 로드하는데 필요한 접근 방법을 0-1 ILP 기반의 해법을 사용한다.

본 논문에서 제안하는 0-1 ILP 공식에 사용되는 기호들을 다음과 같이 정의한다.

- X : 메모리에 액세스하는 순서의 길이
- Y : 최적화할 수 있는 응용 프로그램 변수의 수
- R_a : 주소 레지스터의 수
- R_m : 수정 레지스터의 수
- B : 자동증감 연산 범위를 정의하는데 사용되는 비트의 수

또한, 초기 주소 레지스터에 등록되는 변수의 액세스 순서는 응용 프로그램의 코드에서 자동으로 컴파일러에 의해 추출된 결과이다.

1) 단일 주소 메모리 오프셋 할당

단일 주소 메모리 오프셋 할당 문제를 해결하기 위하여 변수 액세스 순서에 따라 주소 레지스터의 선언적 로드의 수를 최소화 시킨다. 이를 위해 하나의 주소 레지스터는 R_a=1 이라고 가정하고, 프로그램 변수에 대한 최적의 메모리 레이아웃을 결정하여 이를 증명한다.

응용 프로그램을 실행하는데 있어서 필요한 프로그램 변수가 전체 메모리에 하나만 할당되어 있다고 가정하면, 변수와 메모리 위치에 대한 0-1 변수를 정의할 수 있다. 이러한 0-1 변수를 사용함으로써, 프로그램 변수가 메모리의 특정 위치에 접근하는 것을 결정할 수 있다. 그리고 프로그램 포인터를 사용하여 주어진 액세스 순서의 단계를 나타낸다.

X는 액세스 순서의 길이를, Y는 최적화 할 수 있는 응용 프로그램의 변수의 수로 나타내면, 접근 위치에 따른 변수의 잠재적 메모리 위치 L을 지정하는 0-1 변수를 사용하여 제약 조건을 정의할 수 있다.

- $L_{v,l}$: 변수 v가 메모리 위치 l에 할당되는지의 여부를 나타낸다.

주소를 가리키는 현재의 변수가 등록되면 식별을 위해 P를 사용한다.

- $P_{v,s}$: 주소코드가 프로그램 포인터 s의 변수 v에 존재하는가의 여부를 나타낸다.

C는 프로그램 지점이 아닌 주소지 선언에 대한 로드가 발생하는지 확인하는데 사용된다.

- C_s : 주소코드에 선언된 프로그램 포인터 s의 등록 여부를 나타낸다.

R은 변수가 다른 변수의 자동증감 연산 범위에 있는지 나타낸다.

- R_{v_1,v_2} : v_1과 v_2가 서로의 자동증감 범위 내에 있는지 나타낸다.

변수가 단일 메모리에 배치될 수 있기 때문에 다음 수식 (1)과 같은 조건을 만족해야 한다.

$$\sum_{i=1}^{Y} L_{v,i} = 1, \quad \forall v \tag{1}$$

위의 수식에서 인덱스 변수 i는 가능한 모든 메모리 위치를 통해 반복적으로 적용될 수 있다. 따라서 인덱스 변수 i가 Y 위치에 있다고 가정하면, 두 개의 인접한 변수의 메모리 위치 간격을 허용하지 않는다는 조건 하에 프로그램 변수의 수가 많은 경우, 이것은 메모리 위치와 변수 간의 일대일 매핑을 의미한다. 이 가정은 변수의 개수보다 넓은 주소 공간을 가지고 있다고 볼 수 있고, 유일한 변수는 각 메모리 위치에 할당되었는가를 확인하기 위해 다음과 같은 수식 (2)를 제안한다.

$$\sum_{i=1}^{Y} L_{i,l} = 1, \quad \forall l \tag{2}$$

또한, 주소 레지스터는 한 번에 하나의 주소를 저장할 수 있기 때문에 다음과 같은 제약 조건은 모든 프로그램이 같은 조건에서 실행되는 과정 중에 만족해야 하기에 다음과 같은 수식 (3)을 제안한다.

$$\sum_{i=1}^{Y} P_{i,s} = 1, \quad \forall s \tag{3}$$

위의 수식을 기준으로 응용 프로그램이 시작되기 전, 메모리는 그 초기 값으로 *null* 포인터를 가지고 있다고 가정하면, $P_{v,0} = 0, \quad \forall s$으로 나타낼 수 있다. 이 프로그램의 메모리 변수 주소가 2^{B-1} 거리 내에 있다면, 잇따른 두 변수는 자동증감 모드를 통해 액세스 할 수 있는 조건을 만족하게 된다. 이것을 수식으로 표현하면 다음 수식 (4)와 같이 나타낼 수 있다.

$$R_{v_1,v_2} \geq L_{v_1,a} + L_{v_2,b} - 1, \forall v_1, v_2, a, b, \ |a-b| \leq 2^{B-1} \tag{4}$$

이 수식에서는 a는 첫 번째 프로그램의 변수 v_1의 주소, b는 두 번째 프로그램의 변수 v_2의 주소이다. 이 두 변수가 서로 2^{B-1} 거리 안에 있다면, 앞서 언급했듯이 자동증감 모드를 통해 액세스할 수 있다는 결론을 내릴 수 있다.

메모리 변수의 액세스가 발생하는 모든 프로그램의 특정 지점에서 발생한 비용의 합계로 DSP 전체 비용을 산출할 수 있다. 따라서 프로그램의 특정 실행 지점에서 발생한 개인 비용은 C_s 변수를 사용하여 표현한다.

프로그램에 선언적인 로드 값인 포인터 s가 존재한다면, 액세스 되고 있는 현재의 v_1에 v_2는 접근권이 없기에 자동증감이 이뤄지지 않는다. 만약 v_1이 $(P_{v_2,s})$에, v_2가 이전에 $(P_{v_2,s-1})$에 접근이 되었다면, 선언적인 주소코드는 현재 변수가 자동증가 또는 자동감소 모드로 동작되었는지에 따라 그 상태가 변할 수 있기 때문에, (R_{v_1,v_2})로 나타낼 수 있다. 이를 수식으로 표현하면 다음 수식 (5)와 같이 나타낼 수 있다.

$$C_s \geq P_{v_1,s} + P_{v_2,s-1} - R_{v_1,v_2} - 1, \quad \forall s, v_1, v_2 \tag{5}$$

주소 레지스터는 주어진 접근 순서를 따라서 액세스하는 프로그램 변수를 지정하기 때문에, 프로그램에서 최적화 할 수 있는 변수의 액세스 순서를 *Sequence(s)*로 나타내고, 프로그램 포인터 s에 액세스하는 변수 $P_{V,S}$는 강제적으로 해당 0-1 변수를 가리키도록 지정해야 한다. 이렇게 하면 액세스하는 메모리 주소코드는 주소 레지스터에서 사용할 수 있는 상태가 된다. 이를 수식으로 표현하면 다음 수식 (6)과 같이 나타낼 수 있다.

$$P_{Sequence(s),s} = 1 \quad \forall s \tag{6}$$

본 장에서 제안한 ILP 공식의 해법에 필요한 제약 조건을 요약하면 다음 수식 (7)과 같다.

$$\min \sum_{i=1}^{X} C_i \tag{7}$$

C_i는 프로그램 포인터 i에 대한 선언적 주소 로드 여부를 나타내기 때문에, 선언적인 주소 로드들의 총합계를 구할 수 있다. 이 주소 레지스터에 걸리는 로드의 총합을 최소화하는 것이 본 논문에서 제안하는 알고리즘의 핵심이다.

2) 기본 주소 메모리 오프셋 할당

기본 오프셋 할당은 1)절에서 언급한 하나의 주소 레지스터를 갖는 모델보다 확장된 개념으로 다중의 주소 레지스터로 구성된 모델이라 할 수 있다. 하지만 여러 개의 주소를 수용하기 위해서는 위에 제시된 변수와 제약 조건을 일부 개선 및 수정해야 ILP 공식에 적용시킬 수 있다. 프로그램 변수가 실행되는 동안 다른 시간에 서로 다른 주소를 레지스터에 액세스할 수 있도록 단일 주소 메모리 오프셋 할당에서 주어진 제약 조건 이외에도, 각 프로그램 변수에 액세스하는데 사용되는 추가적인 변수의 식별이 필요하다. R_a가 AGU 내부의 주소 레지스터의 수라고 가정하면, 각각의 주소가 실행되는 동안에 포인터를 등록하는 변수를 식별하기 위한 P를 수정하고, 다음과 같은 제약 조건을 정의할 수 있다.

· $P_{r,v,s}$: 주소코드가 프로그램 포인터 s의 변수 v에 r 포인터를 등록 가능한가를 나타낸다. 단일 주소메모리 오프셋 할당에서는 r 포인터 첨자가 없었다.

주소 레지스터 식별자 r은 프로그램의 특정 지점에 발생하는 선언적 주소 로드를 나타내기 위하여, 0-1 결정 변수 C에 대한 r 첨자가 추가 되어야 한다. 그리하여 프로그램 포인터에서 주소 레지스터에 의한 선언적인 로드의 여부를 판단할 수 있다.

· $C_{r,S}$: 프로그램 포인트 s에서 주소 레지스터 r에 대한 선언적인 로드의 등록 여부를 나타낸다.

결정 변수의 수정으로 인한 ILP 제약 사항도 함께 수정이 되어야 한다. 고유 변수에 대한 주소 할당은 모든 주소 레지스터에 대하여 수행되어야 하기 때문에, 이전에 제안한 수식 (3)을 다음 수식 (8)과 같이 수정한다.

$$\sum_{i=1}^{Y} P_{r,i,s} = 1, \quad \forall r, s \tag{8}$$

이 제약 조건은 각 한 번에 하나의 프로그램 변수에 사용 가능한 포인터를 주소 레지스터에 등록될 수 있도록 한다. 따라서 수식 (8)을 기준으로 응용 프로그램이 시작되기 전, 메모리는 그 초기 값으로 *null* 포인터를 가지고 있다고 가정하면, $P_{r,v,0} = 0, \quad \forall r,s$으로 나타낼 수 있다. 0 값은 초기 값을 의미하며, 최적화를 진행하기 위하여 변수들을 액세스하기 전의 주소 레지스터의 상태를 의미한다. 이러한 수정 사항은 수식 (5)에서 주어지 선언적 주소코드에 걸리는 로드의 총 횟수를 바탕으로 비용 함수를 구할 수 있다. 이를 수식으로 표현하면 다음 수식 (9)와 같이 나타낼 수 있다.

$$C_{r,s} \geq P_{r,v_1,s} + P_{r,v_2,s-1} - R_{v_1,v_2} - 1, \quad \forall r, s, v_1, v_2 \tag{9}$$

수식 (6)과 같이 변수에 액세스할 경우, 주소코드는 주소 레지스터들 중 하나에서 반드시 사용할 수 있는 상태가 되어있어야 한다. 이를 수식으로 표현하면 다음 수식 (10)과 같이 나타낼 수 있다.

$$\sum_{i=1}^{R_a} P_{i,Sequence(s),s} = 1 \quad \forall s \tag{10}$$

위에서 언급된 수식들을 종합하여 본 장에서 제안한 ILP 공식의 해법에 필요한 제약 조건을 요약하면 다음 수식 (11)과 같이 새로운 목적 함수를 나타낼 수 있다.

$$\min \sum_{i=1}^{R_a} \sum_{j=1}^{X} C_{i,j} \tag{11}$$

3) 수정 레지스터의 최적화 방법

수정 레지스터는 자동증감 주소지정 모드를 적용하여 변수 데이터의 레이아웃을 최적화한다. 기본 주소 메모리 오프셋 할당 방법 문제로 레지스터를 수정하여 적용하기 위하여 추가적인 0-1 변수를 포함한 수식을 정리하였다. 본 장에서는 변수에 대한 액세스 과정 중 수정 레지스터의 값을 나타내는 YR을 사용하여 최적화 방법을 정의할 수 있다.

· $YR_{r,S,d}$: 주소코드가 프로그램 포인터 s에서 수정 레지스터 r이 d 값을 등록 가능한가를 나타낸다.

수정 레지스터의 값을 변경하면 추가적인 로드 명령어가 요구되기 때문에 비용 산출을 고려하여 YC를 정의한다.

· $YC_{r,S}$: 주소코드가 프로그램 포인터 s에서 수정 레지스터 r에 대한 로드의 등록 여부를 나타낸다.

프로그램 포인터 시점에서, 주어진 수정 레지스터는 반드시 하나의 값을 등록할 수 있다. 이 제약 조건을 수식으로 표현하면 다음 수식 (12)와 같이 나타낼 수 있다.

$$\sum_{i=1}^{Y-1} YR_{r,s,i} = 1, \quad \forall r,s \tag{12}$$

위의 제약 조건에서 메모리 위치의 숫자와 변수의 숫자가 동일하다고 가정하면, 1에서 Y-1의 거리 범위에서 i가 반복 수행된다. 이것은 두 프로그램 변수 간의 최대 거래가 성립한다는 의미이다. 수식 (9)을 기준으로 프로그램 포인터에서 주소 레지스터의 선언적 주소 로드의 비용 함수를 구할 수 있다. 이를 수식으로 표현하면 다음 수식 (13)과 같이 나타낼 수 있다.

$$C_{r_1,s} \geq P_{r_1,v_1,s} + P_{r_1,v_2,s-1} + L_{v_1,a} + L_{v_2,b} + YR_{r_2,s,d} - 4 \ , \\ \forall_{r_1,r_2,s,v_1,v_2,a,b,d} \ , \ a \neq b, \ |a-b| \neq d, \ |a-b| \neq 1 \tag{13}$$

만약 r_1이 이전 프로그램 포인터에서 v_2의 주소를 가지고 있다면, 다음 액세스는 동일한 주소 레지스터에 의해 변수 v_1에게 있으며, 이 변수의 실제 값 a, b는 해당 수정 레지스터에 저장된 값이 d 거리 내에 없음을 의미한다. 이러한 제약 조건을 모든 위치의 조합에 대하여, 모든 거리 및 모든 수정 레지스터에 대한 프로그램 변수의 조합에 적용하여야 한다. 필요한 변수의 메모리 로드는 최소화하기 위한 비용 함수의 일부로서 자신뿐만 아니라 수정 레지스터를 위해서도 필요하다. 이러한 추가적인 로드가 발생하면 그에 따른 비용 함수의 수정도 요구된다. 수정된 비용 함수를 구하는 수식은 다음 수식 (14)와 같이 나타낼 수 있다

$$YC_{r,s} \geq YR_{r,s,i} + YR_{r,s+1,j} - 1 \ , \ \forall r,s \ , \ i \neq j \tag{14}$$

마지막으로, 수식 (11)을 기준으로 위에서 언급된 수식들을 종합하여 전체적인 추가 비용을 고려한 목적 함수를 수정하여야 한다. 이를 수식으로 표현하면 다음 수식 (15)와 같이 나타낼 수 있다.

$$\min\left\{ \sum_{i=1}^{R_a}\sum_{j=1}^{X} C_{i,j} + \sum_{i=1}^{R_m}\sum_{j=1}^{X} YC_{i,j} \right\} \tag{15}$$

위의 수식을 살펴보면, 첫 번째 부분은 주소 레지스터에 대한 선언적인 주소를 로드할 때 발생하는 비용, 두 번째 부분은 수정 레지스터의 주소를 로드할 때 발생하는 비용이다. 이 둘을 합한 결과의 최소값을 구하는 것으로 수정 레지스터에서의 최적화 해법을 구할 수 있다.

3. 실험 결과

본 논문에서 제안한 최적화 방법의 효율성을 입증하기 위하여 Intel Xeon E3-1200의 머신에서 FICO Xpress-MP Modeling Tools[7]을 이용하여 벤치마크 하였다.

Table 1. The applications used in comparison experimental

Benchmark	Explanation	Address Calculating Overhead	
		Code Size	Execution Time
FIR[8]	Radix-2 FIR Filter	23%	15%
FFT[8]	Radix-2 FFT of Complex Sequence	30%	16%
LMS[9]	Properties of IIR Filters	25%	19%
SVD[10]	SV Decomposition Algorithm	28%	15%
Qurt[11]	Root Computation of Quadratic Eqns	31%	19%
Convolution[12]	Full-length Convolution of Vectors	34%	23%

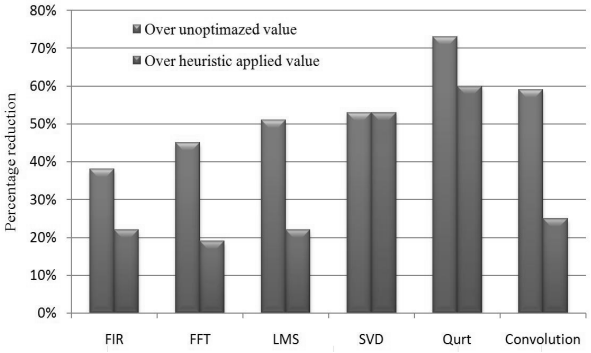
Table 1에 제시된 여섯 가지의 실제 응용 프로그램을 벤치마크 하여 그 결과값을 비교하였다.

Table 1의 첫 번째 열은 응용 프로그램의 이름을 나타내며, 두 번째 열은 간단한 프로그램의 설명을 의미한다. 세 번째 열에서의 코드 사이즈는 응용프로그램에서 선언적 주소 계산에 사용되는 주소코드가 프로그램 비트를 몇 % 차지하고 있는가에 대한 수치이고, 마지막으로 실행시간이란 전체 응용프로그램 실행시간에서 주소 계산이 차지하는 비율을 나타낸다.

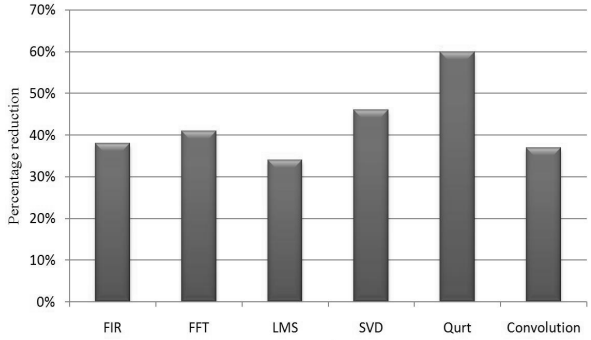
평균적으로 이러한 응용 프로그램들은 프로그램 실행 시 처리해야 될 비트가 거의 28% 정도는 선언적인 주소 계산에 사용된다. 즉, 선언적인 주소 레지스터의 로드의 수를 줄이는 것이 프로그램 실행 시간 단축에 미치는 효과가 크다고 할 수 있다. 마지막 열에는 해당 응용 프로그램 당 전체 실행 시간 대비 주소 계산에 소요되는 간접 기여도 비율을 의미한다. 비교 실험을 위하여 아래와 같이 세 가지 최적화 방법으로 실험을 수행하였다.

- 휴리스틱 방법 : Liao가 제안한 휴리스틱 접근법[13]이다. 크러스칼-왈리스의 최대 스패닝 트리 알고리즘과 비슷한 그리드 알고리즘을 사용한다. 다수의 표본간의 차이를 일괄해서 검정하려는 방법이다. 주로 선형 메모리에서 둘 이상의 근접한 프로그램 변수 사이의 표본을 추출하는데 유용하다.
- ILP 방법 : 본 논문에서 언급한 첫 번째 선형 계획법 기반의 접근법이다. 이는 두 가지 접근을 시도한다. 첫 번째로 2.2의 1)절에서 언급한 하나의 주소 레지스터에 대한 단순 주소 메모리 오프셋 할당 문제에 대한 접근을 위해서, 두 번째로 2.2의 2)절에서 언급한 여러 주소 레지스터가 프로그램 변수에 대한 액세스가 이뤄지는 글로벌 주소 메모리 오프셋 할당 문제에 대한 접근법이다.
- ILP+ 방법 : 2.2의 3)절에서 언급한 수정 레지스터에 대한 확장형 접근법이다. 프로그램 변수가 넓은 주소 공간에 걸쳐 허용되는 경우가 있으므로 프로그램 변수의 사용가능한 공간을 감축하여 주소코드의 수요를 제안하는 접근법이다. 평균적으로 얻어진 ILP 솔루션 시간은 모든 벤치마크 테스트 모델들이 2.9초에서 187.6초까지, 평균적으로 42초 정도의 접근 범위를 허용했다.

위에서 언급한 것과 같이 실험을 실행하여 그 변수 데이터 접근에 대한 평가를 한다. Fig. 3a는 선언적 주소 메모리 할당에 대한 비용 절감을 보여준다. 그래프에 알 수 있듯이, 각각 최적화되지 않은 실험값과 휴리스틱 접근 기반의 ILP 방법이 적용된 실험값의 차이 비교하면, 선언적 주소코드의 수가 최대 53%에서 최소 34%까지 비율 감소한 것을 확인할 수 있다. Fig. 3b는 세 번째 실험 방법인 ILP+ 접근법을 적용하여 수정 레지스터 사용에 따른 주소 레지스터에서 주소코드의 수의 비율이 Fig. 3a의 최적화 되지 않은 기존의 선언적 주소 메모리 할당 비율보다 감소하였음을 확인할 수 있다.



(a) The experimental results of the ILP formulation on heuristic approach



(b) The experimental results of modify register on ILP + formulation

Fig. 3. The experimental results proportion decreased declarative memory address

Fig. 4는 주소 레지스터의 개수가 1~8개로 변화되는 FFT에 대한 벤치마크 실험 결과이다. 두 번 째 ILP 실험 방법에 따라 단순 주소 메모리 오프셋부터 여러 주소 레지스터에 대한 글로벌 주소 메모리 오프셋까지의 할당 문제를 휴리스틱 접근 방법을 적용하여 실험한 결과, 최소 0%에서 53%까지 비율이 감소한 것을 확인할 수 있다. 이는 평균적으로 선언적 주소코드의 로드가 약 16%정도 감소하는 결과를 얻을 수 있었다.

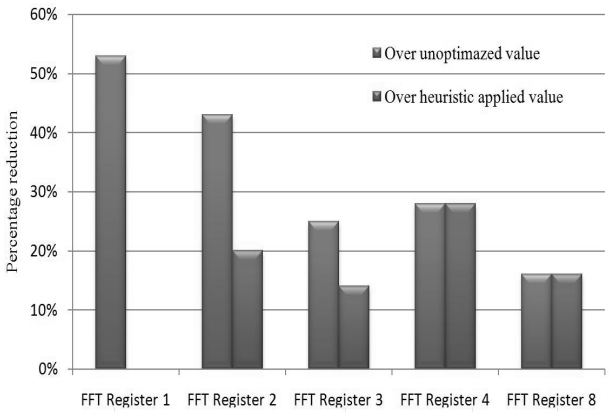


Fig. 4. The experimental results of benchmark FFT

전체적으로 Table 1의 마지막 열에 언급했던 실행 시간과 비교하면, 본 논문에서 제안한 최적화 ILP 알고리즘을 적용한 실험 결과가 평균 총 실험 시간의 약 18% 감소 효과를 확인 할 수 있었다.

4. 결 론

본 논문에서는 임베디드 시스템에서 DSP를 위한 메모리 접근 변수 저장 방법에 대한 최적화 ILP 알고리즘을 제안하였다. 본 논문에서 제안한 최적화 방법은 0-1 ILP 공식을 이용하여 DSP 주소 생성 유닛의 메모리 변수 데이터 레이아웃을 최소화하였다. 제약 조건을 기반으로 변수의 메모리 할당 여부를 식별하고, 변수가 지시하는 주소코드를 프로그램 포인터가 등록하도록 설계하였다. 프로그램의 처리 순서가 프로그램 포인터에 선언되면, 해당 변수의 주소코드에 대한 자동증감 모드를 적용하였다. 주소 레지스터에 대한 로드의 수를 최소화 하여 변수 데이터 레이아웃을 최적화 하였다.

본 논문에서 제안한 알고리즘의 효율성을 입증하기 위한 실험 방법으로는, FICO Xpress-MP Modeling Tools을 이용하여 벤치마크에 적용하였다. 휴리스틱 접근법을 기반으로 ILP 최적화 방법을 주소 레지스터와 수정 레지스터에 각각 적용한 결과, 기존의 선언적 주문 메모리 레이아웃보다 제안한 알고리즘을 적용한 최적의 메모리 레이아웃이 주소/수정 레지스터에 대한 로드 수를 감소시켜 주소코드의 접근을 줄임으로써, 응용 프로그램의 실행 시간을 평균 약 18%정도 단축시킬 수 있음을 입증하였다.

본 논문에서 제안한 알고리즘은 임베디드 시스템의 DSP를 기반으로 적용된 사항으로, 향후 다양한 예외 상황 및 상용 모델들을 대상으로 적용한 연구가 필요할 것이다.

참 고 문 헌

[1] A. Rao and S. Pande. “Storage assignment optimizations to generate compact and efficient code on embedded DSPs”, *Proceedings of the ACM SIGPLAN 1999 conference on Programming language design and implementation*, pp. 128-138, 1999.

[2] A. Darte, R. Schreiber, and G. Villard. “Lattice-based memory allocation”. *IEEE Transactions on Computers*, 54(10), pp. 1242-1257, October, 2005.

[3] B. So, M. W. Hall, and H. E. Ziegler. “Custom data layout for memory parallelism”, *Proceedings of the International Symposium on Code Generation and Optimization*, pp. 291-302, 2004.

[4] S. Leventhal, L. Yuan, N. K. Bambha, S. S. Bhattacharyya and G. Qu, “DSP address optimization using evolutionary algorithms”, *Proceedings of the 2005 workshop on Software and compilers for embedded systems*, pp.91-98, 2005.

[5] J.-Y. Lee and I.-C. Park. “Address code generation for DSP instruction-set architectures”, *ACM Transactions on Design Automation of Electronic Systems*, 8(3), pp.384-395, 2003.

[6] G. Chen and M. Kandemir. “Optimizing Address Code Generation for Array-Intensive DSP Applications”, *Proceedings of the international symposium on Code generation and optimization*, pp.141-152, 2005.

[7] FICO Xpress-MP, http://www.dashoptimization.com/pdf/Mosel1.pdf, [*Internet*], 2002.

[8] Anuj Dharia and Rosham Gummattira, “Signal Processing Examples Using the TMS320C67x Digital Signal Processing Library(DSPLIB)”, *Texas Instruments SPRA947A*, June, 2009.

[9] Properties of IIR Filters, http://cnx.org/content/m16898/latest/#uid6, [*Internet*], July, 2011.

[10] Joachims and Thorsten, “Making large scale SVM learning practical”, *MIT Press, Cambridge, USA*, Oct., 1999.

[11] David Mumford , “Varieties Defined by Quadratic Equations”, *C.I.M.E. Summer Schools*, Vol.51, pp.29-100, 2011.

[12] Ali Sazegari and Doug Clarke, “Single-channel convolution in a vector processing computer system”, *Apple Computer Inc.*, US7895252, Feb., 2011.

[13] S. Liao, S. Devadas, K. Keutzer, S. Tjiang, and A. Wang. “Storage assignment to decrease code size”. *ACM Transactions on Programming Languages and Systems*, 18(3), pp.235-253, 1996.

장 정 욱

e-mail : eugine0772@hotmail.com

2005년 세명대학교 컴퓨터과학과(이학사)

2007년 세명대학교 전산정보학(이학석사)

2007년~현 재 세명대학교 전산정보학 박사과정

관심분야 : CAD, SoC, ASIC, Embedded System, RTOS, USN

인 치 호

e-mail : ich410@semyung.ac.kr

1985년 한양대학교 공과대학 전자공학과 (공학사)

1987년 한양대학교 CAD전공(공학석사)

1996년 한양대학교 CAD전공(공학박사)

1992년~현 재 세명대학교 컴퓨터학부 교수

관심분야 : SoC CAD, ASIC 설계, CAD 알고리즘, SoC 설계, RTOS 및 내장형 시스템