VLIW 기반 고성능 DSP에서의 SAD 알고리즘 최적화 스케쥴링

Optimal Scheduling of SAD Algorithm on VLIW-Based High Performance DSP

유희재, 정선태, 정수환
숭실대학교 정보통신전자공학부

Hui-Jae Yu(rovhr@erc.ssu.ac.kr), Sun-Tae Chung(cst@ssu.ac.kr),
Souhwan Jung(souhwanj@ssu.ac.kr)

요약

SAD(Sum of Absolute Difference) 알고리즘은 동영상 인코더에서 가장 많은 시간이 소용되는 것으로 잘 알려진 움직임 추정에서 가장 자주 계산이 수행되는 알고리즘으로, 동영상 인코딩 수행시간을 줄이기 위해서 우선적으로 최적화 구현되어야 하는 알고리즘이다. 본 논문에서는 VLIW 기반 고성능 DSP 프로세서에서의 조건 분기를 갖는 SAD 알고리즘의 최적 스케쥴링 구현 방법을 제안한다. 제안 방법은 먼저 조건 분기를 갖는 중첩 루프를 VLIW 구조가 제공하는 ILP(Instruction Level Parallelism) 능력을 잘 활용할 수 있도록 충분한 크기의 루프 몸체를 가지며 또한 빨리 루프를 탈출 할 수 있는 조건 분기를 갖는 단일 루프로 변환한 후에, 모듈로 스케쥴링 기법을 적용하여 VLIW 기반 프로세서에서 최적화 스케쥴링 구현을 한다. 제안된 구현 방안을 TMS320C6713에서 구현하고, 코드 크기 및 수행 시간에 대한 성능 분석을 하였다. 구현된 최적화 SAD 루틴은 코드 크기도 크지 않아 임베디드 응용에 적합하며, 이 SAD 구현을 사용한 H.263 인코더가 그렇지 않은 H.263 인코더보다 훨씬 좋은 성능을 보임을 실험을 통해 확인하였다.

■ 중심어 : | SAD | VLIW | 소프트웨어 파이프라이닝 | 움직임 추정 | 동영상 코덱 |

Abstract

SAD (Sum of Absolute Difference) algorithm is the most frequently executing routine in motion estimation, which is the most demanding process in motion picture encoding. To enhance the performance of motion picture encoding on a VLIW processor, an optimal implementation of SAD algorithm on VLIW processor should be accomplished. In this paper, we propose an implementation of optimal scheduling of SAD algorithm with conditional branch on a VLIW-based high performance DSP. We first transform the nested loop with conditional branch of SAD algorithm into a single loop with conditional branch which has a large enough loop body to utilize fully the ILP capability of VLIW DSP and has a conditional branch to make the escape from loop to be achieved as soon as possible. And then we apply a modulo scheduling technique to the tranformed single loop. We test the proposed implementation on TMS320C6713, and analyze the code size and performance with respect to processing time. Through experiments, it is shown that the SAD implementation proposed in this paper has small code size appropriate for embedded applications, and the H.263 encoder with the proposed SAD implementation performs better than other H.263 encoder with other SAD implementations.

■ keyword : | SAD Algorithm | VLIW | DSP | Software Pipelining | Block Matching | Motion Picture Coding |

* 본 연구는 숭실대학교 교내연구비 지원으로 수행되었습니다.

접수번호 : #071107-001
접수일자 : 2007년 11월 07일
심사완료일 : 2007년 12월 03일
교신저자 : 정선태, e-mail : cst@ssu.ac.kr

I. 서 론

최근 들어 동영상 폰, 스마트폰, 디지털 카메라, 임베디드 웹카메라 서버, PDA 등 차세대 PC 제품 임베디드 환경에서 멀티미디어 스트림 처리를 수행하여야 하는 멀티미디어 응용이 증가하고 있다[1][2].

임베디드 프로세서의 경우에, 요구되는 멀티미디어 처리 능력의 증대를 지원하기 위한 병렬처리 지원 구조로는 VLIW(Very Long Instruction Word)의 채택이 대세이다[1,3,4]. 멀티미디어 스트림 처리 응용에 많이 쓰이는 고성능 프로세서인 TMS320C6x의 경우에도 VLIW 구조에 기반하고 있다. 그런데, 워크스테이션에서 개발된 동영상 코덱을 VLIW 구조 프로세서에 그대로 이식하는 경우, VLIW 구조가 지원하는 ILP (Instruction Level Parallelism)를 효율적으로 이용하지 못하기 때문에, 제대로 된 성능을 얻기 힘들다.

대부분의 멀티미디어 스트림 처리를 필요로 하는 멀티미디어 응용은 동영상 인코딩 / 디코딩 작업을 필요로 한다. 움직임 추정은 동영상 코딩에서 가장 중요한 과정의 하나이다[5]. 움직임 추정을 위해서는 보통 수백번 이상의 SAD(Sum of Absolute Difference) 연산이 필요하다. 따라서 움직임 추정 과정이 빠르게 수행되기 위해서는 이 SAD 알고리즘의 효율적 구현은 필수적이다. SAD 알고리즘은 보통 C언어에서 루프 탈출을 위한 조건 분기를 갖는 중첩 루프로 프로그래밍 된다. 단일 루프에 대한 소프트웨어 파이프라이닝 기법의 최적 스케줄링에 대한 연구는 많이 수행되었다[4][18]. 그러나 중첩 루프, 특히 조건 분기를 갖는 중첩 루프에 대한 소프트웨어 파이프라이닝의 최적화 스케줄링에 관한 연구는 드물다[6-8].

본 논문에서는 조건 분기를 갖는 중첩 루프로 구현되는 SAD 알고리즘을 VLIW 기반 고성능 DSP에서 소프트웨어 파이프라이닝 최적 스케줄링으로 구현하는 방법에 대해 제안한다. 제안 방법은 먼저 조건 분기를 갖는 중첩 루프를 VLIW 구조가 제공하는 ILP (Instruction Level Parallelism) 능력을 최대로 잘 활용할 수 있도록 충분한 크기의 루프 몸체를 가지며 가능한 빨리 루프를 탈출할 수 있도록 하는 조건 분기를 갖는 단일 루프로 변환한다. 이후, 변환된 단일 루프에 대해 모듈로 스케줄링 기법을 적용하여 최적화 스케줄링을 구현한다. 제안한 SAD 알고리즘의 최적화 스케줄링 구현 방안을 텍사스 인스트루먼트 사의 VLIW 기반 고성능 DSP인 TMS320C6713에 구현하고 코드 크기 및 수행 시간에 대한 성능 분석을 수행하였다. 성능 분석을 위해 사용한 동영상 코덱은 UBC의 H.263[9] 인코더이다. 분석 결과, 코드 크기가 작아 임베디드 응용에 적절하였다. 또한, QCIF (176×144) 크기의 3장의 원시(raw) 영상 프레임을 1장 인트라모드 인코딩 및 2장 인터모드 인코딩으로 인코딩할 때, 본 논문에서 제안한 SAD 구현을 사용한 H.263 인코더가 워크스테이션에서 구현된 SAD 알고리즘을 사용한 H.263 인코더보다 수행 소요 사이클 수에서 1.97 배 개선되었음을 확인할 수 있었다. H.263 또는 MPEG 의 경우, 인트라모드의 경우보다 인터모드 코딩이 훨씬 더 많으므로, 이러한 성능 개선은 더 많은 프레임을 인코딩할수록 더욱 향상된다.

블록 매칭 알고리즘의 구현에 대한 연구는 많으나, 대부분은 프로세서 구조를 고려한 구현이 아니고, 알고리즘 자체의 효율적인 구현에 대한 연구이다[10][11]. 또한, H.263/MPEG 의 최적 구현 및 성능 분석에 대해 많은 연구가 수행되었으나[2][12-15], 전체 SW 설계 또는 SIMD 및 VLIW 프로세서에서의 성능 분석이고, VLIW 프로세서에서의 특정 알고리즘 의 최적화에 대한 연구는 드물다[16].

텍사스 인스트루먼트 사의 DSP 이미지 라이브러리[17]는 SAD 알고리즘의 최적화 스케줄링된 어셈블리 구현을 제공하고 있다. 그러나 이는 조건 분기가 없는 SAD 알고리즘이어서 이를 사용하는 H.263 인코딩은 본 논문에서 제안한 SAD 구현을 사용하는 H.263 인코딩 보다 수행 속도에 대한 성능이 떨어진다.

본 논문의 구성은 다음과 같다. 제2절에서는 본 논문 내용의 기술적 배경에 대해 간략히 소개된다. 제3절에서는 본 논문에서 연구한 SAD 알고리즘의 VLIW DSP에서의 최적화 스케줄링 구현 방법이 기술되며, 제4절에서는 실험 결과 및 분석이 기술된다. 마지막으로 제5절에 결론이 주어진다.

II. 기술적 배경

1. 동영상 코덱, 움직임 추정 및 블록 매칭[5]

H.263, H.264, MPEG 등의 동영상 코덱과 JPEG 등의 정지 영상 압축 코덱과의 기본적인 차이는 동영상 압축 코덱에서는 영상 프레임들 간에 존재하는 시간적 영상 정보 중복(temporal visual information redundancy)을 제거하고 압축하므로 써 압축률을 크게 개선시킨다는 점이다. 시간적 영상 정보 중복의 제거는 움직임 추정(Motion Estimation) 및 움직임 보상(Motion Compensation)을 통해 이루어진다. 움직임 보상은 압축하고자 하는 현 프레임의 매크로블록에 대해 가장 잘 매칭 되는 매크로블록을 참조 프레임의 탐색 영역에서 찾는 블록 매칭 작업 수행으로 이루어진다.

블록 매칭 작업은 SAD(Sum of Absolute Difference) 알고리즘을 이용하는데, 현 매크로블록(16x16 픽셀)과 이전 인코딩된 프레임에서의 탐색 영역에서 같은 크기(16x16)의 매크로블록과 해당 픽셀 쌍의 차이의 절대값을 구하고 이를 다 합친 값을 해당 매크로블록 매칭의 SAD 라 하며, 이 SAD가 가장 작은 값의 매크로블록을 탐색영역에서의 최선 매칭 블록으로 판정하는 것으로 수행된다. 참고로 시간적 영상 정보 중복을 제거하고 인코딩할 때 이를 인터모드 인코딩이라 하며, 그렇지 않은 경우는 인트라모드 인코딩이라 한다.

2. VLIW(Very Long Instruction Word) 프로세서

VLIW 프로세서에서는 소프트웨어적으로(컴파일러에 의해) 명령어간 데이터 의존성이 점검되고 병렬로 수행될 수 있는 인스트럭션 워드 패킷 구성이 이루어진다. 명령 수행 시에는 인스트럭션 워드 패킷이 명령어 페치, 명령어 해석 파이프라인 단계까지 파이프라이닝으로 동작되고 실행단계에서 인스트럭션 패켓의 구성 명령어들이 각각 다른 기능블록으로 이슈 되어 명령어 실행이 병렬로 실행되는 구조를 갖는다. 하드웨어적으로 명령어 스케줄링을 하는 수퍼스칼라 프로세서에 비해 VLIW 프로세서는 하드웨어가 수퍼스칼라 프로세서보다 간단하므로, 크기 제약이 주어지는 임베디드 프로세서의 주요 ILP 프로세서 구조로 자리 잡고 있다[1][3][4][18].

3. 소프트웨어 스케줄링, 루프 펼침(loop unrolling) 및 소프트웨어 파이프라이닝 (software pipelining)

스케줄링의 제약 조건으로는 프로세서 내 자원의 유한(레지스터 및 기능블록 개수의 유한), 명령어간의 데이터 의존성, 기능블록 수행 지연(functional unit latency), 조건 분기, 선결 제약(recurrence constraint) 등이 있다. 선결 제약은 루프의 한 반복에서의 한 연산이 이전 루프 반복에서의 같은 연산에 의존될 때의 제약을 말한다. 스케줄링의 기본적인 목표는 이러한 제약 조건하에서 가급적 많은 명령어들이 병렬로 처리되도록 명령어 순서를 재조정하고 그룹핑 하는 것이다[3][4][18]. 프로그램의 루프에 적용되는 스케줄링 기법에는 루프 펼침(loop unrolling), 소프트웨어 파이프라이닝(software pipelining) 등이 잘 알려져 있다[3][4][18].

루프 펼침은 단순히 루프 몸체(body)를 여러 차례 복사하여 루프 몸체에 추가함으로써 루프 몸체의 크기를 증가시키는 것을 말한다. 루프 펼침은 루프 반복에 의한 분기 및 오버헤드 명령을 줄인다. 또한, 루프 몸체가 증가하여 좀 더 많은 명령어 단계 병렬성을 스케줄러가 고려할 수 있기 때문에 명령어 단계 병렬성이 증가하게 된다. 소프트웨어 파이프라이닝은 소프트웨어 파이프라인된 코드의 각 반복(iteration)이 원래 루프에서의 각기 다른 반복으로부터 선택된 명령으로부터 만들어 지도록 루프를 재구성하여 루프와 기능블록들이 효율적으로 스케줄링 되도록 하는 소프트웨어 최적화 기법이다.

4. 소프트웨어 파이프라이닝 스케줄링

4.1 개요

소프트웨어 파이프라인된 루프는 3단계를 거친다. 소프트웨어 파이프라인된 코드에서 파이프라인이 가능한 가득 채워져 반복되는 단계를 커널이라고 하며, 커널로 진입하는 코드 단계 부분을 프롤로그, 커널에서 나오는 코드 부분 단계를 에필로그라 한다.

커널의 길이는 연속적인 커널 반복 시작 사이간의 간격(interval)을 의미하는 데, 이를 시작 간격(initiation interval)(보통 II 라 표기)이라 하며, II가 적을수록 소프트웨어 파이프라인의 명령 단계 병렬성이 높아지므로 프로그램 수행 시간이 짧아진다. 따라서 소프트웨어 파이프라이닝 스케줄링 알고리즘 목적은 제약 조건 아래서 II를 최소화하도록 스케줄링을 하여 성능을 최대화하는 것이다. 최소 II 는 MII라 보통 표기되는 데, MII를 구하는 것은 NP-complete[18]로 잘 알려져 있다. 따라서 MII 계산에 아래의 모듈로 스케줄링에서와 같이 휴리스틱 접근이 필요하다.

4.2 모듈로 스케줄링 (modulo scheduling) [4][19]

모듈로 스케줄링 기법은 소프트웨어 파이프라인 루프의 모든 반복이 같은 스케줄(모듈로 스케줄)을 사용하며, 일정 비율(Single II)로 연속적인 반복이 수행되도록 함으로써, 루프의 반복 간에 있는 ILP를 개발하는 스케줄링 기법으로 소프트웨어 파이프라이닝 스케줄링 기법가운데 가장 많이 사용된다. 기본적으로 다음과 같은 절차로 이루어진다.

① 처음 II 값을 다음과 같이 초기화 한다.
$II = Max(ResMII, RecMII)$
② 데이터 의존 그래프(Data Dependence Graph) 와 MRT(Modulo Resource reservation Table) 에 기반을 두어 II 을 만족하는 모듈로 스케줄을 탐색
③ ②를 만족하는 스케줄이 있으면 종료, 아니면 II 값을 하나 증가시키고 ②를 다시 수행

여기서 ResMII는 자원 제약 조건하에서의 최소 II 을 RecMII는 선결 제약 조건 하에서의 최소 II 을 나타낸다. ResII와 RecMII를 결정하는 방법은 [19]를 참조하라. 데이터 의존 그래프는 명령어와 데이터의 흐름과 명령어들 사이의 의존 관계를 보여주는 그래프이며, 순환예약테이블 {c mod II } 에 해당하는 타임 슬롯에서 사이클 c 에서의 자원 활용을 기록하고 점검하는 테이블을 말한다[4][19].

5. TMS320C67x

5.1 TMS320C67x 구조[20]

TMS320C6x는 텍사스 인스트루먼트(TI) 사의 VLIW 기반 고성능 DSP로서, 한 사이클에 최대 8개의 명령을 병렬 처리할 수 있는 8개의 기능 블록을 가지며 16개 또는 32개 32비트 레지스터를 갖는다 (TMS320C6x 중 C64x는 32개, 그 밖의 C6x 시리즈는 16개). TMS320C67x는 TMS320C6x 계열의 칩으로, 부동소수점을 지원한다. TMS320C67x에 대한 보다 자세한 내용은 [20]을 참조하라.

5.2 TMS320C6x 명령어 세트 구조[21]

TMS320C6000시리즈 (6713 포함)의 명령어 세트는 RISC 컴퓨터의 Load-Store 구조를 갖는다. 즉, load 및 store 명령어 외에는 메모리 접근(읽기/쓰기)이 허용되지 않는다. 또한 각 명령어는 8개의 기능블록 (L, S, D, M 기능 블록이 각 2개씩)중 지정된 기능블록에서만 수행된다.

파이프라인 지연 시간(delay slot)은 명령어의 소스 피연산자(오퍼랜드)가 읽혀진 후, 명령어의 결과가 읽혀질 수 있도록 되는 데 걸리는 CPU 클록 사이클 수를 말한다. TMS320C6x 명령어 세트 대부분을 차지하는 1 사이클 명령어의 경우, 지연 시간은 0 사이클이며, 16×16 곱셈 명령은 1 사이클, 분기 명령은 5 사이클, load 명령은 4 사이클 등의 지연 시간을 갖는다.

다음은 TMS320C6x 어셈블리 명령어 예이다.

```
LDBU .D1 *A4++, A7 ;
||[A1] ADD .S2 B13, A8, B12;//A1 의 값이 0 이 아니면 실행
```

TMS DSP 어셈블리 프로그래밍 시에 상기 예와 같이 동시에 병렬로 수행할 수 있는 지의 여부 표시(||), 조건 실행([A1]), 해당 명령어를 수행하는 기능블록(.D1, S2) 명시, 또한 사용될 레지스터 지정(A4, A7, B13, A8, B12), 파이프라인 지연 시간 등을 고려하여야 하므로 프로그래밍이 매우 복잡한 편이어서, 일반 사용자로서는 사용하기가 쉽지 않다. 따라서 TI 사 에서는 이러한 복잡한 사항 가운데, 병행 수행 여부 (||), 해당 명령어 수행 기능블록, 사용될 레지스터 구체적 지정, 각 명령어의 파이프라인 지연 시간 등을 모두 다 명시

할 필요 없는 선형 어셈블리(linear assembly)를 지원하고, 어셈블리 최적화기(assembly optimizer)가 이 선형 어셈블리 코드로부터 어셈블리 코드를 생성할 수 있도록 지원하고 있다.

5.3 TMS320C6x에서 자원 제약 조건 및 메모리 뱅크 충돌

TMS320C6x에서 자원 제약 조건은 [21]을 참조하라. TMS320C6x에서는 또한 메모리 뱅크 충돌을 고려하여야 한다[21]. 대부분의 C6x 프로세서들은 인터리브드 메모리 뱅크를 사용하고 있으며, 뱅크 메모리가 단일 포트(single port) 메모리로 구성되기 때문에 주어진 사이클에서 같은 뱅크를 접근하는 경우는 1 메모리 사이클 정지(stall)를 초래한다.

III. TMS320C67x 에서 SAD 알고리즘 최적화 스케줄링 구현

1. SAD 알고리즘의 스케줄링 분석

보통 구현되는 매크로블록간 SAD 알고리즘 C 코드는 다음과 같다 (이 코드는 UBC H.263 인코더[9]에서 사용되었다).

```
int SAD_Macroblock (unsigned char *ii,
 unsigned char *act_block, int h_length, int Min_FRAME)
{
 // ii ; 탐색 영역 메모리 포인터
 //act_block ; 현재 매크로블록 메모리 포인터
 // h_length ; 탐색 영역의 너비 크기 ;
              본 논문의 H.263 에서는 31~46
 // INT_MAX   ; unsigned 정수의 최대 값(2,147,483,647)
 // Min_FRAME ; 이전에 수행된 SAD 계산에서의 최소 값
               초기 값은 INT_MAX
 int i,j;
 int sad = 0;
 unsigned char *kk;
 kk = act_block;
 i = 16;
 while (i--)
 {
    for (j=0; j<16; j++)    // 내부 루프
      sad+= abs(*(ii+j)-*(kk+j)); //내부 루프
    ii += h_length;
    kk += 16;
   if (sad > Min_FRAME) //조건 분기
     return INT_MAX;
 }
 return sad;
}
```

그림 1. 매크로블록간 SAD 알고리즘의 C 구현

[그림 1]의 코드에서 보면, 매크로블록간 SAD 계산 루틴은 조건 분기를 갖는 중첩 루프([그림 1]에서 이탤릭체 굵은 글씨 부분)로 구현되는 데, 중첩 루프의 소프트웨어 파이프라이닝 최적화를 위해 여러 접근 방법을 고려할 수 있다.

첫째, 내부 루프를 최적 소프트웨어 파이프라이닝 스케줄링하고 이를 외부 루프에 적용하는 방법이다. 이 경우 외부 루프 매 반복에서 내부 루프의 프롤로그, 커널, 에필로그 단계가 파이프라인 채움과 비움을 반복하기 때문에 그 효율이 좋지 않다. 현재, TMS320C6x 컴파일러가 [그림 1]의 코드에 대해 이와 같은 소프트웨어 파이프라이닝 스케줄링을 한다.

둘째, 내부 루프를 모두 펼친 후에 외부 루프 하나에 대해 기존에 개발된 싱글 루프 소프트웨어 최적화 기법을 적용하는 것이다. 이 경우, 내부 루프를 모두 펼치기 때문에 코드량이 많이 증가하게 된다. [16]에서는 이 방법에 의해 최적화 스케줄링을 수행하였다. 그러나, 전체 코드량이 많아 임베디드 시스템에서의 구현에 불리하며, 또한 루프 몸체가 커져 레지스터의 재활용 등에 대해서도 고려해야 하므로, 효율적인 최적화 스케줄링이 어려워진다.

셋째, 중첩 루프를 다차원 루프 소프트웨어 파이프라이닝 스케줄 최적화 문제로 해결하는 방법이다[8]. 이 방법은 엄밀한 분석 기반 위에서 스케줄링이 가능하나, 현재까지 연구된 결과는 루프 안에 조건 분기가 없는 다중 루프의 경우를 주로 다루고 있어, 루프 안에 조건 분기를 갖는 블록 매칭 알고리즘의 경우에 손쉽게 적용할 수는 없다.

본 논문에서는 이상의 중첩 루프 소프트웨어 파이프라이닝 스케줄링 방법들을 개선한 다음의 스케줄링 방법을 제안하다.

VLIW 프로세서의 병렬 처리 능력 (TMS320C6x의 경우, 최대 8개 명령어 동시 수행 가능)과 빠른 루프 탈출 조건 분기 실현을 고려하여, 조건 분기를 갖는 중첩 루프를 단일 루프로 변환하고 단일 루프에서 개발된 모듈로 스케줄링 기법을 이용한다.

이렇게 변환된 단일 루프의 경우, 루프 몸체가 내부 루프 전체를 펼친 것보다는 코드량이 작아 임베디드 구

현에 유리하고, 또한 VLIW 프로세서의 병렬 처리 능력이 충분히 활용할 수 있도록 스케줄링이 가능하여 소프트웨어 파이프라이닝의 커널 길이가 감소되고, 또한 빠른 루프 탈 분기 조건을 실현하기 때문에 SAD 알고리즘의 평균적 수행 시간이 적게 소요되어 성능의 개선이 가능하다. TI 의 DSP 이미지 라이브러리 [17]에서 구현된 SAD 알고리즘도 VLIW 병렬 처리 능력이 최대가 되도록 구현되었다. 하지만, 이 구현은 조건 분기가 없는 SAD 알고리즘의 구현이어서, 본 논문의 구현과는 다르며 또한 조건 분기에 의한 루프 탈출 시간의 감소가 없어 평균적 SAD 알고리즘 소요 시간이 커서, 루프 탈출 조건 분기를 갖는 SAD 알고리즘 구현보다는 성능이 떨어진다.

2. 단일 루프 변환 소프트웨어 파이프라이닝 스케줄링

2.1 단일 루프로의 변환

SAD 수행은 기본적으로 한 쌍의 영상 픽셀 값(현재 매크로블록의 한 영상 픽셀 값과 그와 비교되어야 할 참조 매크로블록의 영상 픽셀 값)을 읽어 들이고(LDBU), 두 영상 픽셀 값의 차이 구한 후(SUB)에 이 값의 절대값을 구하고(ABS), 이 값을 이전 값에 더하는(ADD) 과정의 반복으로 이루어진다. 따라서 이러한 SAD 수행 명령어들이 VLIW 프로세서의 ILP 능력에 따라 가능한 한 많이 병렬로 수행되어야 한다. 또한, 루프 탈출의 조건 분기가 가능하므로, 가능한 한 빨리 루프를 탈출할 수 있도록 루프 탈출 조건을 빨리 점검하도록 할 필요가 있다. 따라서 루프 탈출 조건 분기를 갖는 중첩 루프의 SAD 알고리즘을 VLIW 프로세서에서 최적화 스케줄링 되도록 하는 데 적합한 단일 루프 SAD 알고리즘으로 변형하는 데 고려해야 할 기본적인 방침은 다음의 2가지이다.

① SAD 수행 명령어들을 VLIW 프로세서의 ILP능력(TMS320C6x의 경우, 최대 8개 명령을 동시에 수행 가능)을 최대로 이용할 수 있도록 한다.
② 루프를 가급적이면 빨리 탈출할 수 있도록 한다.

이러한 2가지 기본 방침에 기반하여 TMS320C6x 프로세서에서의 구현을 염두에 둔 SAD 단일 루프 C 코드 구현은 [그림 2]와 같다.

[그림 2]의 단일 루프 변환은 위 2가지 방침에 기반한 구현을 위해, 짝수 픽셀 2개 쌍, 홀수 픽셀 2개 쌍 등 4개의 픽셀 쌍에 대해 SAD 수행 명령들을 최대한 병렬로 수행하고 이를 합친 후에 결과가 이전 매크로블록의 SAD 연산 값인 MIN_FRAME보다 크거나 같으면 SAD 알고리즘 루프를 탈출하도록 하고, 그렇지 않으면 루프를 반복하게 하도록 설계한 C 코드이다.

```
int SAD_Macroblock (unsigned char *ii,
 unsigned char *act_block, int h_length, int Min_FRAME) {
  int i, l=0;
  int sad = 0, sad_e=0, sad_o=0;
  unsigned char *kk;
  kk = act_block;
  for (i=0 ; i < 64 ; i+=1)
  {
    sad_e =
        abs (*(ii+l) - *kk) + abs (*(ii+2+l) - *(kk+2));
    sad_o =
        abs (*(ii+1+l)- *(kk+l)) + abs (*(ii+3+l) - *(kk+3));
    sad = sad_e+sad_o;
    if (sad >=Min_FRAME)
        return sad ;
    ii+=4;
    kk+=4;
    if ((i+1)%4 == 0)      //(1)
        l += h_length-16;     //(1)
    }
  return sad;
}
```

그림 2. 단일 루프로 변환된 SAD 알고리즘의 C 구현

2.2 TMS320C67x에서의 SAD 알고리즘의 최적화 소프트웨어 파이프라이닝 스케줄링

[그림 2]의 굵은 글씨 부분의 C 코드 에 대해 본 논문에서 설계한 TI 선형 어셈블리 코드는 다음과 같다.

```
ZERO     j, sad_e, sad_o, sad
MVK      0x40, i
MVK      0x2000, j_init
SUB      h_length,  16, A_p

loop:
      LDBU  *A_srcImg[1],   odd_sp1
      LDBU  *B_srcImg[1],   odd_sp3
      LDBU  *A_refImg[1],   odd_rp1
      LDBU  *B_refImg[1],   odd_rp3
```

```
      LDBU  *A_srcImg++[4], even_sp0
      LDBU  *B_srcImg++[4], even_sp2
      LDBU  *A_refImg++[4], even_rp0
      LDBU  *B_refImg++[4], even_rp2

      SUB   even_sp0,  even_rp0,  even_d0
      SUB   even_sp2,  even_rp2,  even_d2
      SUB   odd_sp1,   odd_rp1,   odd_d1
      SUB   odd_sp3,   odd_rp3,   odd_d3

      ABS   even_d0,   even_a0
      ABS   even_d2,   even_a2
      ABS   odd_d1,    odd_a1
      ABS   odd_d3,    odd_a3

      ADD   sad_e,  even_a0, sad_e
      ADD   sad_o,  odd_a1,  sad_o
      ADD   sad_e,  even_a2, sad_e
      ADD   sad_o,  odd_a3,  sad_o

      ADD   sad_e, sad_o,  sad

          CMPLTU  sad, Min_FRAME, temp
[temp] B      Get_Out   //(B.1)
       SUB    i,   1,   i

[temp] MPY    i,  0,  i     //  (A.1)
       MPY    j,  2,  j     //  (A.2)
[!j]   ADD     A_refImg, A_p, A_refImg
       ADD     A_refImg, 2,   B_refImg

[!j]   MPY     j_init,   1,  j   // (A.3)

[i]    B     loop      //(B.2)

Get_Out:
   .return          sad
```

그림 3. 그림 2의 굵은 글씨체 부분 C 코드에 대한 TMS320C67x에서의 선형 어셈블리 코드

[그림 3]의 선형 어셈블리 코드의 기본 아이디어의 하나는 TMS320C67x에서의 한 사이클에서 가능한 한 8개 기능 블록 모두가 충분히 활용될 수 있도록 [그림 2]의 코드부분 (1)에 대해 남는 M 기능 블록을 사용하도록 하게 한 것이다([그림 3]에서 (A.1), (A.2), (A.3) 부분)

이제, [그림 3]의 선형 어셈블리 명령어들에 대해 소프트웨어 파이프라이닝 스케줄링 분석을 수행한다.

2.4 절에서 설명하였듯이 소프트웨어 파이프라이닝 스케줄 기법 중 가장 많이 유용하게 쓰이는 것이 모듈로 스케줄링이다. 본 논문에서도 모듈로 스케줄링 기법 [19]을 사용하여 소프트웨어 파이프라이닝 스케줄링 분석을 한다.

가. 소요 레지스터에 따른 자원 제약

[그림 3]의 선형 어셈블리 코드에서 사용한 심볼릭 레지스터들이 모두 각기 다른 레지스터를 사용한다고 할 때, 총 18개의 레지스터가 필요하다. TMS320C67x에서는 레지스터의 총 개수가 32개이므로, 소요 레지스터 사용에 따른 자원 제약은 없다.

나. MII 계산

먼저, 선결 제약은 없으므로($RecMII = 0$), 자원 제약 조건만을 고려하면 된다. 또한, 레지스터에 따른 자원 제약은 없으므로, 기능블록에 따른 자원 제약 조건만을 고려하여 ResMII를 계산하면 된다(2.4 절 참조). [그림 3]의 선형 어셈블리 코드에서, 소프트웨어 파이프라인의 커널에서 필요로 하는 기능 블록의 수는 다음과 같다. 8개의 LDBU 명령어 수행에 필요한 D 기능블록 8개, 4개의 ABS 명령어 수행에 필요로 하는 L 기능블록 4개, 7개의 ADD 명령어 및 5개의 SUB 명령어 수행에 필요한 L 또는 S 또는 D 기능블록 12개, 3개의 MPY 명령어 수행에 필요한 M 기능 블록 3개, 1개의 CMPGT 명령어를 수행하는 데 필요한 L 기능 블록 1개, 2개의 B 명령어 수행에 필요한 S 기능 블록 수는 2개이다.

소프트웨어 파이프라이닝 커널에서 각 기능 블록을 사용하는 명령어들을 수행하는 데 필요한 최소 사이클은 다음과 같이 계산된다.

$$\left\lceil \frac{\text{필요한 해당블록의 총갯수}}{\text{한 사이클에서 사용가능한 해당기능블록 수}} \right\rceil$$

따라서 D 기능 블록을 사용하는 명령어들을 수행하는데 필요한 최소 사이클은 $\lceil 8/2 \rceil = 4$, L 또는 S 기능 블록을 사용하는 명령어들을 수행하는 데 필요한 최소 사이클은 $\lceil 19/4 \rceil = 5$, M 기능 블록을 사용하는 명령어들을 수행하는 데 필요한 최소 사이클은 $\lceil 3/2 \rceil = 2$, 따라서 $ResMII = 5$이며 $II = \lceil Max(ResMII RecMII) \rceil = 5$ 이다. 이제, $II = 5$ 즉, 커널 길이(또는 시작 간격)가 5인 모듈로 스케줄링이 제약 조건(명령어간 의존 제약, 명령어들의 지연시간 제약, 동시 사용 가능 블록 제약, 메모리 뱅크

충돌 제약 등) 아래에서 가능한지에 대해, 의존 그래프와 MRT을 이용하여 점검할 필요가 있다.

다. 2개의 분기 명령과 최적 소프트웨어 파이프라이닝 스케줄링

그런데, 커널 길이를 II = 5 로 구현하려면 무엇보다도 분기(Branch) 명령어의 지연 시간 문제를 해결해야만 한다. [그림 3]의 SAD 어셈블리 루틴은 2개의 조건 분기를 갖는다. 분기 명령어 하나는 루프를 반복하도록 하는 분기 명령어 (B.2) 이고, 다른 하나가 루프를 탈출하는 분기 명령어 (B.1) 이다. 이 경우, II = 5 에서는 2개의 분기 명령어 간격이 4 사이클을 넘을 수 없는 데, TMS320C6x의 경우, 분기 (B) 명령어의 지연시간은 5 사이클이므로 별다른 조처를 취하지 않으면 루프를 탈출하는 분기 명령어가 실행되기 전에 루프 반복 분기 명령어도 파이프라인에 적재되게 된다. 따라서 루프 탈출 분기 명령어 실행되더라도, 이미 파이프라인에 들어가 있는 루프 반복 분기 명령도 결국 실행되어 다시 루프로 복귀하게 되어 루프를 영원히 빠져나갈 수 없는 경우가 발생한다.

본 논문에서는 이러한 문제를 조건 분기를 이용하여 해결하였다. 즉, 루프를 빠져 나가는 분기 명령이 실행되는 경우에는 이후 루프 반복 분기 명령이 실행되지 못하도록 하는 것이다. 루프 반복 분기 명령은 감소하는 루프 카운트 가 0이 되는 경우에 수행되지 않아 파이프라인에 적재되지 못한다. 따라서 루프 탈출 분기 명령이 실행되는 경우에 루프 카운트의 값이 0이 되도록 세팅하면([그림 3]의 (A.1) 부분), 루프 반복 분기 명령어는 적재되지 않으므로 루프로 실행되지 못하며 따라서 다시 루프로 돌아가는 것을 막을 수 있다. 루프 카운트의 값을 0으로 세팅하는 것은 남는 M 유니트를 이용하도록 MPY 를 이용하였다.

라. II = 5 모듈로 스케줄링

상기 가), 나), 다) 의 고려 아래에서, [그림 3]의 코드에 대해 의존 그래프와 MRT 을 이용하여 II = 5 을 만족하는 모듈로 스케줄링이 가능함을 어렵지 않게 보일 수 있다. (의존 그래프 분석, MRT 분석 및 이렇게 하여 구한 최적 소프트웨어 파이프라이닝 모듈로 스케줄링 결과의 최종코드는 지면 관계상 생략한다).

IV. 실험 및 결과 분석

1. 실험 환경

본 논문에서는 Texas Instruments 의 6713DSK 보드를 사용하였다. 6713DSK 평가보드는 호스트에서의 통합 개발환경인 Code Composer Studio를 지원한다. Code Compose Studio는 Assembler, C Compiler, Linker, Profiler, Debugger, Deassembler 등을 지원한다. 평가보드의 6713 DSP는 225MHz 로 동작하며, 외장 메모리로 사용되는 8 MB 의 SDRAM은 100MHz로 동작한다. Flash memory 512KBytes를 지원하며, 테스트를 위한 I/O를 지원한다.

L2 메모리는 224KB, 내부 메모리는 32KB L2 캐쉬로 구성하였으며, heap 사이즈는 7MB, 스택 사이즈는 64KB로 세팅하였다.

Host는 윈도우즈 워크스테이션이다. Host가 한 일은 코드의 크로스 컴파일링, 타켓 6713 DSK 로의 실행파일 다운로드, 디버깅, 프로파일링 등 이다.

사용한 H.263 Encoder는 TMN 8.0 에 기반을 둔 UBC(University of British Columbia) H.263 version 2(H.263+) 구현[16]으로 C 소스 코드의 크기는 17,000 라인(530KBytes)이다. 실험에서는 기본 모드(baseline) 만을 테스트하였고 선택한 영상 포맷은 QCIF이며, 테스트 영상은 테스트 영상으로 많이 쓰이는 foreman (foreman.qcif)이다. 또한, 움직임 벡터 추정에는 전 탐색(fast full search) 알고리즘을 선택하였으며, 역 DCT 는 빠른 IDCT(FastIDCT) 알고리즘을 사용하였다. 또한, 컴파일 시에 -o2 옵션(소프트웨어 파이프라이닝 지원)으로 컴파일 하였다.

2. 실험 결과

2.1 SAD 알고리즘의 코드 크기 및 최악 수행 소요 시간 비교

[표 1]은 본 논문에서 구현한 SAD 알고리즘의 코드

크기 및 수행 소요 클록 사이클을 다른 3가지 구현과 비교한 실험 결과이다. 다른 3가지 구현은 다음과 같다. 1) UBC SAD ; UBC H.263 인코더에 구현된 SAD 알고리즘 (그림 1)으로 -o2 옵션으로 컴파일 한 결과, 2) TI 라이브러리 SAD; TI DSP 이미지 라이브러리에서 구현한 SAD 어셈블리 코드, 3) [16] SAD ; 논문 [16]에서 구현한 SAD 어셈블리 코드

표 1. 본 논문 구현 SAD와 다른 SAD 구현간의 SAD 알고리즘 의 코드 크기 및 수행 소요 클록 사이클 수 비교

SAD 구현	코드 크기 (Bytes)	최악 소요 클록 사이클 수
UBC SAD (-o2 옵션 사용)	324	739
TI 라이브러리 SAD	240	258
[16] SAD	808	275
본 논문 구현 SAD	252	319

본 논문 구현 SAD 의 코드 크기가 TI 라이브러리 SAD 보다 크며, 최악 소요 사이클 수가 많은 이유는 TI 라이브러리 SAD 는 조건 분기가 없는 SAD C 알고리즘을 구현했기 때문이다. TI SAD 라이브러리 SAD는 최악 소요 사이클 수나 최소 소요 사이클 수나 같다. 그러나 본 논문 구현 SAD 는 조건 분기가 있기 때문에 현재에 계산되는 SAD 의 값이 이전 매크로블록들의 SAD 계산까지의 최소 SAD 값보다 크면, 더 이상 SAD 계산을 하지 않고 SAD 루틴을 빠져 나오기 때문에, 평균적 SAD 수행 소요 사이클 수는 TI 라이브러리 SAD 보다 적게 된다. 따라서 최대 900번 SAD 알고리즘을 수행해야 하는 블록 매칭 계산에 있어서, 인트라모드 인코딩이 더 빠르게 수행된다[표 2].

논문 [16]에서는 본 논문과 같이 조건 분기가 있는 SAD 알고리즘을 구현하였다. 그런데 [16]의 경우, 중첩 루프에서 내부 루프를 모두 펼쳐서 단일 루프를 만들었기 때문에 코드 크기는 매우 크다. 한편, 변형된 단일 루프의 카운트는 16 이므로, 본 논문의 구현의 단일 루프의 카운트 64 보다는 작아 루프 반복에 따른 오버헤드가 적기 때문에 최악 소요 사이클 수는 본 논문의 구현보다 적다. 그러나 같은 조건 분기를 구현한 SAD 이더라도, 논문 [16]의 경우는 16개 픽셀 쌍에 대해, SAD 를 계산하고 루프 탈출 조건을 점검하는 데 반해, 본 논문 구현은 4개 픽셀쌍에 대해 SAD 를 계산하고 루프 탈출 조건을 계산하므로, 평균적 SAD 수행 소요 사이클 수는 적게 된다[표 2].

2.2 구현 SAD 알고리즘의 성능 분석

본 논문에서 구현한 SAD 알고리즘의 성능을 분석하기 위해, 6713DSK에 구현된 4 가지 종류 H.263 인코더에 대해 QCIF 크기의 3개의 원시(raw) 이미지 프레임을 한 개는 인트라 모드로, 다른 2개는 인터모드로 인코딩하여 성능을 비교하였다. 구현된 4 가지 종류의 H.263 인코더는 다음과 같다.

① 기존 UBC H.263 인코더,
② TI 라이브러리 SAD 구현을 사용한 H.263 인코더
③ 논문 [16]의 SAD 구현을 사용한 H.263 인코더
④ 본 논문 SAD 구현을 사용한 H.263 인코더

4가지 인코더 모두 o2 옵션으로 컴파일 하였다. 또한, 사용한 QCIF 원시(raw) 비디오 시퀀스는 테스팅 용으로 자주 인용되는 foreman.qcif 이다.

다음 [표 2]는 4 종류 H.263 의 성능 비교 실험의 결과이다.

표 2. 구현된 H.263 인코더들의 성능 비교

H.263 인코더	SNR			총 소요 사이클 수
	Y	Cr	Cb	
UBC 인코더	31.25	38.78	38.82	150,925,893
TI 인코더	31.55	38.58	39.15	124,580,125
[16] 인코더	31.34	38.85	39.06	109,273,707
본 논문 구현 SAD 사용 인코더	30.90	38.25	38.58	76,484,977

[표 2]에서의 SNR 값은 2개 프레임에 대한 인터모드 인코딩에서의 SNR 값의 평균을 나타낸다.

[표 2]에서 보면, 본 논문에서 구현한 SAD 루틴을 사용한 H.263 인코더가 기존 UBC H.263 인코더보다, QCIF 인트라모드 1 프레임, 인터모드 2 프레임 등 총 3 프레임 인코딩 시에 비슷한 SNR 값을 가지고, 1.97

배 빠르게 수행되었음을 볼 수 있다. 또한 다른 SAD 구현을 사용한 H.263 인코딩에 비해서도 성능이 개선되었음을 알 수 있다. H.263 및 MPEG 등에서 인터모드 인코딩 횟수가 인트라모드 인코딩 횟수보다 훨씬 많으므로 이러한 성능 개선은 인코딩하는 프레임수가 많을수록 상당한 시간 절약을 가져옴을 알 수 있다.

앞에서 지적하였듯이 본 논문에서 제안한 SAD 최적화 어셈블리 루틴이 TI DSP Image Library에서 제공한 SAD 어셈블리 루틴과 논문 [16]에서 구현한 SAD 어셈블리 루틴들 보다 최악 소요 사이클 수가 크지만, 이들을 사용한 H.263 인코더보다 본 논문에서 제안한 SAD 최적화 루틴을 사용한 H.263 의 성능이 우수한 이유는 본 논문에서 제안한 SAD 어셈블리 루틴이 평균 소요 사이클이 적게 걸리기 때문이다.

V. 결 론

본 논문에서는 VLIW 구조 프로세서에서의 동영상 인코더에서 가장 많은 시간이 소용되는 것으로 잘 알려진 움직임 추정의 SAD 알고리즘을 VLIW 구조 프로세서에서의 최적화 구현에 대해 연구하였다.

SAD 알고리즘은 보통 조건 분기를 갖는 중첩 루프로 프로그래밍 된다. VLIW 프로세서에서 루프는 소프트웨어 파이프라이닝 기법으로 최적 구현된다. 본 논문에서는 조건 분기를 갖는 중첩 루프를 VLIW 구조가 제공하는 ILP(Instruction Level Parallelism) 능력을 잘 활용할 수 있도록 충분한 크기의 루프 몸체를 가지며 또한 빨리 루프를 탈출 할 수 있는 조건 분기를 갖는 단일 루프로 변환한 후에, 모듈로 스케줄링 기법을 적용하여 VLIW 기반 프로세서에서 최적화 스케줄링 구현을 하는 방안을 제안하였다. 제안된 SAD 구현 방안을 TMS320C6713에서 구현하고, 코드 크기 및 수행 시간에 대한 성능 분석을 하였다. 분석 결과, 구현된 최적화 SAD 루틴은 코드 크기도 크지 않아 임베디드 응용에 적합하며, 이 SAD 구현을 사용한 H.263 인코더가 그렇지 않은 H.263 인코더보다 훨씬 좋은 성능을 보임을 실험을 통해 확인하였다.

본 논문의 연구 결과는 VLIW 기반 프로세서에 이식되거나 설계되는 S/W 알고리즘의 최적화 방안에 도움을 줄 것으로 기대된다.

참 고 문 헌

[1] P. G. Paulin, et al., "Embedded Software in Real-Time Signal Processing Systems: Application and Architecture Trends," Proc. of IEEE, Vol.85, No.3, pp.419-435, Mar. 1997.

[2] M. Budagavi et. al., "Wireless MPEG-4 Video Communication on DSP chips," IEEE Signal Processing Magazine, pp.36-53, Jan. 2000.

[3] J. Hennessy and D. Patterson, *Computer ArchitectureA Quantitative Approach, 3rd ed.*, MK Pub. 2003.

[4] J. A. Fisher, P. Farabosch, and C. Young, *Embedded Computing A VLIW Approach to Architecture, Compilers, and Tools,* Morgan Kaufmann, 2004.

[5] I. E. G. Richardson, *Video CODEC Design Developing image and video compression systems*, John Wiley & Sons, 2002.

[6] K. Muthukumar and G. Doshi, "Software pipelining of nested loops," In R. Wilhelm, editor, CC 2001, LNCS 2027. Springer-Verlag, Berlin Heidelberg, pp.165-181. 2001.

[7] R. Scales, *Nested loop optimization on the TMS320C6x. Application Report SPRA519*, Texas Instruments, Feb. 1999.

[8] Q. Zhuge, Z. Shao, and E. H.-M. Sha, "Optimization of Nest-Loop Software Pipelining, " http://www.utdallas.edu/~edsha

[9] B. Erol, F. Kossentini and H. Alnuweiri, "Implementation of a fast H.263+ encoder/decoder," Proc. IEEE Asilomar Conf. Alsilomar Conf. on Signals, Systems and Computers,

Vol.1, pp.462-466, Nov. 1998.

[10] G. Gupta and C. Chakrabarti, "Architectures for hierarchical and other block matching algorithms," IEEE Trans. on Circuits and Systems for Video Technology, Vol.5, No.6, pp.477-489, Dec. 1995.

[11] W. Hwang, Y. Lu, and Y. Zeng, "Fast block-matching algorithm for video coding," Letters Vol.33, Issue 10, pp.833-835, May 1997.

[12] D. Talla, et al., "Evaluating signal processing and multimedia applications on SIMD, VLIW and Superscalararchitectures," Proc. of 2000 Int'l Conf. on Computer Design, pp.163-172, Sept. 2000.

[13] S. M. Akramullah, I. Ahmad, and M. L. Liou, "Optimization of H.263 video encoding using a single processor computer: performance tradeoffs and benchmarking," IEEE Trans. on Circuits and Systems for Video Technology, Vol.11, Issue 8, pp.901-915, Aug. 2001.

[14] H. Miyazawa, *H.263 Encoder: TMS320C6000 Implementation*, Application Report SPRA721, Texas Instruments, Dec. 2000.

[15] O. Lehtoranta, T. Hamalainen, and J. Saarinen, "Real-time H.263 encoding of QCIF-images on TMS320C6201 fixed point DSP," Proc. of IEEE International Symposium on Circuits and Systems, Vol.1, pp.28-31 pp.583-586, 2000.

[16] S. Bangerjee, et al., "VLIW DSP vs. Superscalar Implementation of a Baseline H.263 Video Encoder," 34th IEEE Alsilomar Conf. on Signals, Systems and Computers, Vol.2, pp.1665-1669, 2000.

[17] http://focus.ti.com/docs/toolsw/folders/print/sprc093.html

[18] M. S. Lam, "Software pipelining: An effective scheduling technique for VLIW machines," in Proc. ACM SIGPLAN 1988 Conf. on Programming Language Design and Implementation, pp.318-328, Jun. 1988.

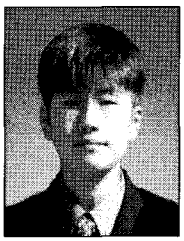
[19] B. R. Rau, "Iterative Modulo Scheduling," International Journal of Parallel Processing, Vol. 24, No.1, Feb. 1996.

[20] *TMS320C6713 Datasheet,* No. SPRU186D, Texas Instruments, May, 2003.

[21] *TMS320C6000 CPU and Instruction Set Reference Guide,* No. SPRU189F, Texas Instruments, 2000.

저 자 소 개

유 희 재(Hui-Jae Yu) 준회원



- 2006년 2월 : 성결대학교 정보통신공학과 학사
- 2006년 8월 ~ 현재 : 숭실대학교 대학원 석사 재학

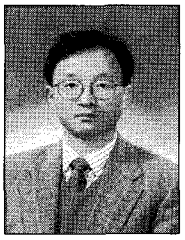
<관심분야> : 임베디드 컴퓨팅

정 선 태(Sun-Tae Chung) 정회원

- 1990년 12월 : 미국 미시간대학교(앤아버) 전자 및 컴퓨터 박사
- 1991년 ~ 현재 : 숭실대학교 정보통신전자공학부 교수

<관심분야> : 임베디드 컴퓨팅, 생체인식, 컴퓨터 비젼, 영상 감시

정 수 환(Souhwan Jung) 정회원



- 1996년 6월 : University of Washington 박사
- 1997년 ~ 현재 : 숭실대학교 정보통신전자공학부 부교수

<관심분야> : 이동인터넷 보안, VoIP 보안, RFID/USN 보안