DSP 67x 기반 음성인식 시스템의 가우시안 확률 계산 최적화 구현

최태일*, 김태윤**, 고한석***
고려대학교 전자컴퓨터 공학과

Optimization of Gaussian Mixture Computation of ASR on DSP 67x.

Taeil Choi*, Taeyun Kim**, Hanseok Ko***
Dept. Electronics and Computer Engineering,Korea University
*tichoi@ispl.korea.ac.kr, **tykim@ispl.korea.ac.kr, ***hsko@korea.ac.

요약

본 논문은 HMM 기반 임베디드 음성인식 시스템 구현에 관한 몇 가지 주제들을 설명한다. 임베디드 환경은 한정된 자원을 가지고 있고 그러한 가운데 타당한 인식률과 향상된 인식 속도를 얻기 위해서 몇 가지 방법들을 이 논문에서 설명한다.

구현 환경은 DSP6711 기반에서 이루어졌다. 가우시안 mixture 계산 루틴을 부동소수점 연산에서 고정소수점 연산 및 software pipelining 을 적용하였다.

고정소수점 변환 전과 후 비슷한 인식률을 얻었고 고정소수점 변환과 software pipelining 적용 후 연산 속도의 향상을 얻었다.

1. 서론

고정소수점 변환은 저전력을 요구하는 휴대용 기기들을 구현하기 위한 필수적인 대세이다. 고정소수점 변환은 계산 량 부하를 최소화 할 수 있고, 또한 작은 크기의 데이터가 동시에 접근하게 됨으로 서 자원이 제한된 임베디드 환경에서 효과적인 연산 처리가 가능해질 수 있다. 또한 DSP 구조를 통하여 software 적인 pipelining 개념을 이용한 병렬처리를 통하여 시스템 성능을 높일 수 있다. [5]

1.1. 고정소수점 변환

임베디드 환경에서 궁극적인 음성인식 시스템은 실시간으로 인식 과정을 수행하는 시스템이다. 따라서 빠른 인식 속도를 얻는 것이 중요하다. HMM 알고리즘을 기반한 음성 인식 시스템에서 인식 속도에 가장 많은 영향을 미치는 부분은 확률 스코어 (likelihood score) 계산 부분이다.

보통 고정소수점 연산은 부동 소수점 연산보다 더 빠른 속도를 보장한다. 따라서 이러한 확률(likelihood) 계산 루틴을 고정소수점 수학 연산으로 변환하여 계산을 빠르게 하는 것이 성능 향상의 한가지 방법이다. [2] 그러나 고정소수점 변환은 계산상의 정확성 감소를 초래하게 된다. 그러므로 납득할만한 인식률을 유지하면서 고정소수점 변환을 하는 것이 주요하다.

1.2. Software pipelining

Software pipelining[5]은 어떤 루프를 구성하는 명령어들을 조절하여 그 루프를 병렬적으로 수행하도록 하는 방법을 말한다. 이 방법은 고정소수점 변환을 통해서 그 효율성이 더욱 증대될 수 있다. 예를 들어 32bit 부동 소수점 연산이 16bit 고정소수점 연산으로 변한다면 두 개의 16bit 데이터가 병렬로 동시에 접근될 수 있을 뿐만 아니라 하나의 데이터에 대한 처리 속도도 부동 소수점 연산보다 빨라지게 된다. 따라서 각 루프당 명령어 수와 각 명령어 당 처리

속도가 줄게 되고 각각의 반복 루틴(iteration) 간격 이 줄게 되어 연산 속도가 빨라지게 된다.

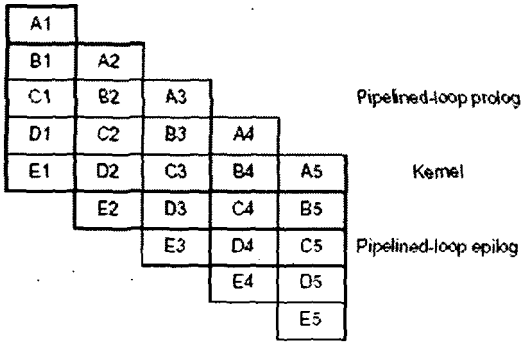

그림 1. 병렬 처리의 개념

위의 그림은 명령어 의 병렬 수행에 관한 개념을 나타내고 있다. 1 부터 5 까지의 숫자들은 루프 주기를 나타내고, A~E 까지는 각각의 루프를 이루는 명령어들을 나타낸다. 특정한(A~E) 명령어들을 수행하는 프로세서들이 독립적으로 존재한다면 하나의 명령어가 수행될 때 다른 명령어가 동시에 수행될 수 있다는 것을 보여주고 있다.

2. 구현

2.1. DSP 구조

그림 2. 는 C67x 계열의 DSP 구조를 보여주고 있다.[4] 두 개의 register file A, B 가 존재하고 각 register file 들은 16 의 register 로 이루어져 있다. 그리고 각 register file 에는 독립적이면서 서로 다른 기능을 수행하는 functional unit 이 4 개씩 연결되어있고 register file A 와 B 간의 접근을 지원하기 위해 cross path 가 하나씩 서로 연결되어 있다.

기본적인 cross-compiler는 compile과정에서 음성 인식 시스템 처리를 위한 각 명령어들에 대해서 위에서 말한 자원들을 자동적으로 할당하게 된다.

명령어 집합에서 사용되는 총 8개의 functional unit과 2개의 cross path들은 명령어 scheduling 하는데 있어서 같은 cycle에서 각각의 명령어에 서로 중첩되어 사용되지 않도록 충분히 고려해서 할당해 주어야 한다.

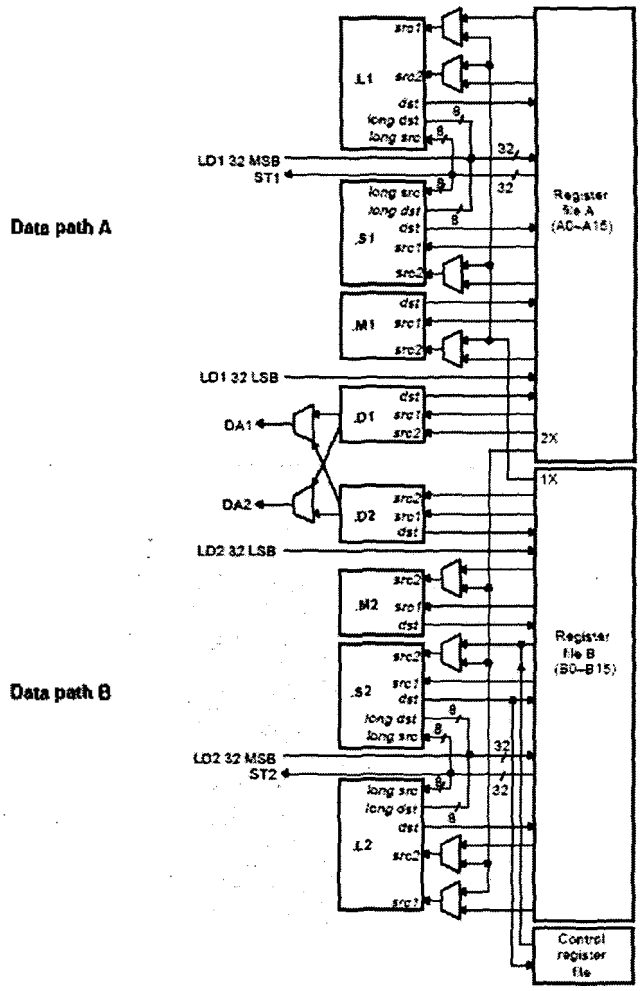

그림 2. 기본적인 67x DSP 구조

2.2. 메모리 의존성(Memory dependency) [5]

본 임베디드 음성 인식 시스템은 continuous HMM 을 기반으로 하고 있으며 Tied-Mixture 를 이용하여 각 스테이트(state)의 출력 확률 값이 나오게 되어있다. 여기서 mixture component 는 multivariate gaussian model 이다. 인식 시스템의 계산 부하를 줄이기 위하여 multivariate gaussian mixture model 을 Single value 로 변환하면 다음과 같은 log 확률(likelihood) 값을 계산할 수 있게 된다. [1]

$$\ln N(o;\mu,\Sigma) = G + \frac{1}{2}(o-\mu)^T \Sigma^{-1}(o-\mu) \qquad (1)$$

Gaussian mixture model 기반 실시간 음성 인식 시스템에서는 위의 계산 루틴이 프레임 단위로 반복해서 일어나게 되고 따라서 인식 속도에 큰 영향을 미치게 된다.

그림 3.은 mixture 확률(likelihood) 값을 얻기 위한 루틴 내부의 메모리 의존성(memory dependency) 그래프 이다. 16bit 고정소수점 변환을 통해 한 cycle 에 2 개의 16bit 데이터(1 WORD) 메모리 로딩(loading)을 시작할 수 있다. 따라서 수식에서 보여지듯 13 차 특징이 입력인 경우 총 7 번의 루프가 돌아가게 된다. 다시 말해서 그림의 데이터 흐름이 7 번 반복된다.

그림 3.에서 화살표 방향의 메모리는 화살표 시작 메모리에 의존(dependent)한다는 의미이다. 따라서 화살표 방향의 메모리 처리(memory processing)은 표시된 숫자만큼의 cycle 수가 걸리고 그 다음에서야 결과값이 나오게 된다. 그리고 다음 명령어가 그 결과값을 이용하여 실행될 수 있다. 따라서 그림 3.의 데이터 흐름은 총 13 cycle 이 걸리게 된다.

DSP 67x는 이론적으로 한 cycle에 종첩이 되지 않는 명령어를 총 8개까지 동시에 수행을 할 수 있다. 이러한 DSP의 병렬 성을 이용하여 그림 3.이 나타내는 처음 하나의 반복 루틴(iteration)이 끝나기 전에 다시 새로운 반복 루틴(iteration)들을 시작할 수 있다. 결국 software pipelining 의 목표는 이러한 다중 반복 루틴(iteration)을 병렬적으로 처리하는 것이다.

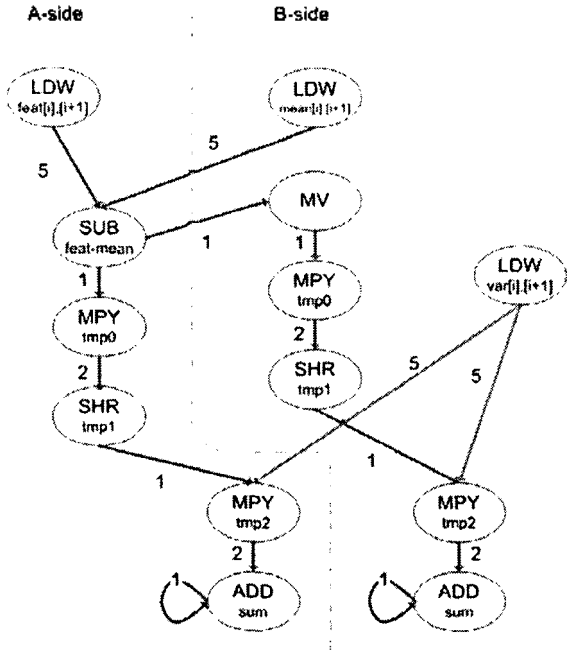

그림 3. 메모리 의존(memory dependency) 그래프

표 1.은 그림 3.의 데이터 흐름에서 각각의 명령어들에 대한 자원 할당을 보여주고 있다. 표 2.는 그림 3.의 데이터 흐름과 표 1.에서 명령어들에 할당된 자원을 고려하여 A-side 부분의 명령어들을 scheduling 한 것이다. 이 것은 몇 개의 명령어들로 이루어진 반복 루틴(iteration)을 어떻게 병렬처리 할 것인지를 보여주고 있다. 1~12 까지는 cycle 수를 나타낸다.

표 1. DSP 67x 에서의 자원 할당

A-side		B-side	
Unit	명령어	Unit	명령어
.M1	MPY	.M2	MPY, MPY
.S1	SHR	.S2	SHR, MV
.D1	LDW	.D2	LDW
.L1,.S1,.D1	SUB, ADD	.L2,.S2,.D2	ADD
1X	SUB, MPY	2X	MV

표 2. 명령어 scheduling (A-side)

	1	2	3	4	5	6
.D1	LDW1		LDW2		LDW3	
.M1						
.L1						
.S1						SUB1
	7	8	9	10	11	12
.D1	LDW4					
.M1	MPY1		MPY2	MPY'1	MPY3	MPY'2
.L1						ADD1
.S1		SUB2	SHR1	SUB3	SHR2	SUB4

(각 functional unit 과 명령어는 참고문헌[4] 참조)

3. 실험 및 결과

실험에서는 본 연구실에서 개발한 C 언어 기반 2000 단어 급 고립단어 음성인식 시스템을 이용하였다.

시스템은 음성 처리/인식을 위하여 모듈화 되어있고 본 실험에서는 실시간 임베디드 환경에서 가우시안 mixture 성분 추출 모듈의 성능 향상에 초점을 맞추어 실험이 진행되었다.

3.1 고정소수점 변환의 영향

인식률은 120 개의 고립단어 샘플의 인식 테스트를 하였다. 표 3. 은 부동소수점 연산과 고정소수점 연산의 탐색 공간 변화에 따른 인식률을 나타내는 표이다. 탐색 공간은 워드 네트워크(word network)에 존재하여 발산하는 토큰의 개수를 말한다.

표 3. 부동소수점 연산과 고정소수점 연산의 인식률 비교

탐색 공간 (MAXBEAM)	100	150	200	250
부동소수점 연산의 인식률(%)	90.0	92.5	93.3	95.0
고정소수점 연산의 인식률(%)	89.2	90.0	90.8	95.0

3.2 고정소수점 변환과 software pipelining 의 영향

표 4. 는 부동소수점 연산 속도와 고정소수점 변환 후 연산속도 그리고 software pipelining 후의 연산 속도를 비교하고 있다. 연산 속도의 비교는 하나의 단어가 인식되는 과정에서 각 프레임당 연산 루틴의 평균 계산 cycle 수 및 시간을 비교하였다. software pipelining 은 고정소수점 변환 후 적용하였다.

표 4. 부동소수점 연산과 고정소수점 연산의 연산 속도 비교 (1 cycle = 약 6.7 ns)

	평균 Cycle/frame	평균 (μ sec.)
부동소수점 연산 루틴	101039	676.96
고정소수점 연산 루틴	67811	454.33
고정소수점 & Software pipelining	41336	276.95

4. 결론

위의 인식 실험과 연산 속도 실험에서 보여지듯이 고정소수점 변화에 의한 인식률의 감소는 최대 0.8%로서 무시할만한 수준이다. 고정소수점 변환 후 연산 속도는 약 33% 의 증가를 보였고 그 후 software pipelining 을 적용하면 고정소수점 변환 후의 연산 처리 속도와 비교하여 약 39% 의 연산처리 속도의 증가를 보였다. 결과적으로 고정소수점 변환 후 software pipelining 을 적용하면 부동소수점 연산보다 약 59%의 연산 속도 성능 향상을 보였다.

참고문헌

[1] L. R. Rabiner and B. H. Juang, Fundamentals of Speech Recognition, 1st ed., ser. Prentice Hall Signal Processing Series.

[2] Jean-Claude Junqua, Panasonic Technologies, Inc., U.S.A, Robust Speech Recognition in Embedded Systems And PC Applications, Kluwer Academic Publishers

[3] Yifan Gong and Yu-Hung Kao, Implementing a High Speed Accuracy Speaker -independent Continuous Speech recognizer on Fixed-point DSP, Texas Instruments Incorporated

[4] TMS320C6000 CPU and Instruction Set (sprz168e) Texas Instruments, August 2002. [Online]Available:http://focus.ti.com/lit/ug/spru189f/spru189f.pdf

[5] TMS320C6000 Programmer's Guide (SPRU198G), Texas Instruments, August 2002. [Online]Available:http://focus.ti.com/lit/ug/spru198g/spru198g.pdf

[6] Implementing Speech-Recognition Algorithms on the TMS320C2xx Platform Application Report, Texas Instruments 1998