논문 2012-49SP-2-19

시간적 근접성 향상을 통한 효율적인 SVM 기반 음성/음악 분류기의 구현 방법

(Efficient Implementation of SVM-Based Speech/Music Classifier by Utilizing Temporal Locality)

임 정 수*, 장 준 혁**

(Chungsoo Lim and Joon-Hyuk Chang)

요 약

서포트벡터머신 (support vector machine)을 이용한 음성/음악 분류기는 높은 분류 정확도로 주목받고 있으나 많은 계산 량과 저장 공간을 요구하므로 특히 임베디드 시스템과 같이 자원이 제한 적인 경우에는 효율적인 구현이 필수적이다. 특히, 서포트벡터 (support vector)의 차원과 개수에 의해 결정되는 서포트벡터의 저장 공간의 크기는 일반적으로 임베디드 프로세서의 캐시 (cache)의 크기보다 훨씬 크므로 캐시에 존재하지 않는 서포트벡터를 메인 메모리로부터 읽어야 하는 경우가 많다. 메모리에서 데이터를 가져오는 데는 캐시나 레지스터와 비교했을 때 상대적으로 긴 시간과 많은 에너지가 소비되어 분류기의 실행 시간과 에너지 소비를 증가시키는 요인이 된다. 본 논문에서는 분류기의 데이터 접근 양식을 보다 시간적 근접성을 가지게 변환하여 일단 프로세서 칩으로 불려진 데이터를 최대한 활용함으로써 메모리의 접근 횟수를 줄여 전체적인 서포트벡터의 실행 시간의 단축시키는 기법을 제안한다. 실험을 통해 메모리로의 접근 회수의 감소와 이에 따른 실행시간 그리고 에너지 소비의 감소를 확인하였다.

Abstract

Support vector machines (SVMs) are well known for their pattern recognition capability, but proper care should be taken to alleviate their inherent implementation cost resulting from high computational intensity and memory requirement, especially in embedded systems where only limited resources are available. Since the memory requirement determined by the dimensionality and the number of support vectors is generally too high for a cache in embedded systems to accomodate, frequent accesses to the main memory occur inevitably whenever the cache is not able to provide requested data to the processor. These frequent accesses to the main memory result in overall performance degradation and increased energy consumption because a memory access typically takes longer and consumes more energy than a cache access or a register access. In this paper, we propose a technique that reduces the number of main memory accesses by optimizing the data access pattern of the SVM-based classifier in such a way that the temporal locality of the accesses increases, fully utilizing data loaded into the processor chip. With experiments, we confirm the enhancement made by the proposed technique in terms of the number of memory accesses, overall execution time, and energy consumption.

Keywords: Support Vector Machine (SVM), Selectable Mode Vocoder (SMV), Speech/Muisc Classification Algorithm, Embedded System, Memory, Cache

* 정회원, 목포대학교 정보산업연구소
(Research Institute of Information Science and Engineering, Mokpo National University)
** 정회원, 한양대학교 융합전자공학부
(Dep. of Electronic Engineering, Hanyang University)
※ 본 연구는 지식경제부 및 한국산업기술평가관리원의 IT핵심기술개발사업의 일환으로 수행하였음. [2009-S-036-1, 고성능 가상머신 규격 및 기술 개발]
※ 이 논문은 2011년도 정부(교육과학기술부)의 재원으로 한국연구재단의 대학중점연구소 지원사업으로 수행된 연구임 (2011-0022980)
접수일자: 2011년7월14일, 수정완료일: 2011년12월9일

Ⅰ. 서 론

근래에 무선통신기술을 기반으로 하는 멀티미디어 서비스의 비약적인 성장으로 제한된 주파수 대역의 효율적인 사용이 중요한 과제로 연구되어지고 있다. 제한된 통신망의 효율적인 이용을 위하여 가변 전송률을 가지는 다양한 음성 코덱이 개발 되었는데[1, 2], 음성신호의 유형에 따라 적절한 전송률을 할당하기 위해서는 음성신호의 유형을 정확히 분별하는 작업이 선행되어야 한다. 특히 음성과 음악의 분류는 각기 할당되어야 하는 전송률의 차이가 크기 때문에 매우 중요하고 따라서 활발히 연구되어지고 있는데[3, 4], 최근에 서포트벡터머신 (SVM)을 이용한 분류기가 분류의 정확도로 인해 주목받고 있다[5, 6]. 그러나 SVM은 일반적으로 많은 계산량과 서포트벡터를 저장하기 위한 넓은 메모리 공간을 요구한다. 특히 SVM 기반 분류기를 임베디드 시스템에서 구현하기에는 임베디드 시스템의 제한된 프로세서의 성능이나 메모리의 크기로 인해 효율적인 구현이 절실히 요구되어 진다.

대부분의 프로세서는 데이터와 코드를 독립적인 메모리에서 불러들여 사용하는 Von Neumann 구조[7]를 가지고 있다. 프로세서와 메모리의 구조를 그림 1에 나타내었다. 캐시와 레지스터파일 (register file)은 프로세서 칩 (processor chip) 안에 위치한다. 프로세서는 속도를 높이는 방향으로 개발이 되어져 왔고, 메모리는 용량을 늘이는 방향으로 발전되어져 왔기 때문에, 원래 컸었던 둘 사이의 속도의 격차가 점점 더 커지게 되었다. 이 속도의 차이 때문에 메모리보다 접근속도는 빠르지만 용량이 적은 캐시 (cache)가 도입되어 메모리와 프로세서 사이에서 메모리의 역할을 일부 감당하게 되었다. 즉 캐시가 프로세서 파이프라인에서 필요한 데이터를 가지고 있다면, 메모리 접근 없이 그 데이터는 레지스터파일로 옮겨져 프로세서 파이프라인에서 연산에 사용되어 진다. 프로세서와 메모리 사이의 속도 격차가 커질수록 둘 사이에서 중개의 역할을 하는 캐시의 중요성이 높아지게 되고, 또한 캐시의 활용 정도에 따라서 성능의 차이가 커지게 되었다.

일반적으로 학습된 서포트벡터머신은 많은 수의 서포트벡터를 포함하고 또한 벡터의 차원도 높은 경우가 많아서, 일반적인 임베디드 프로세서의 온칩 캐시 (on-chip cache)의 크기 (8KB ~ 64KB)로는 모든 서포트벡터를 캐시에 저장할 수 없다. 따라서 서포트벡터

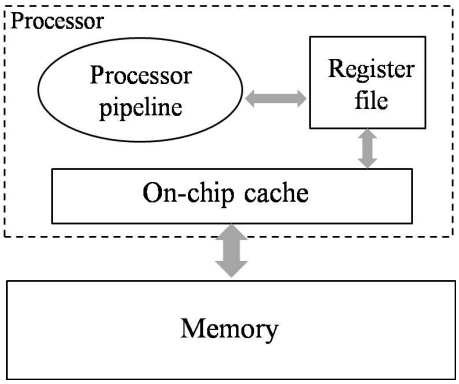


그림 1 프로세서와 메모리의 구조.
Fig. 1. Processor and memory structure.

중 일부만이 캐시에 저장되어지고 계속 메모리를 접근하여 데이터를 캐시로 이동시켜야 한다. 메모리에의 접근은 레지스터파일이나 캐시에 접근하는 것 보다 실행시간과 에너지소비의 관점에서 불리하므로 최대한 줄여야 한다. 그러나 서포트벡터머신의 분류과정을 살펴보면 하나의 입력벡터에 대해 모든 서포트벡터들이 순차적으로 읽혀서 연산에 사용되고 다음 입력벡터에 대해서 다시 처음부터 읽혀진다. 그러므로 캐시가 모든 서포트벡터를 저장할 만큼 크지 않다면, 각 서포트벡터는 메모리로부터 캐시로 불려진 후 캐시나 레지스터파일에 저장되어 있는 동안 하나의 입력벡터를 위해서만 사용되어진다. 따라서 서포트머신의 데이터 접근 양식 (data reference pattern)은 매우 비효율적이고 개선의 여지가 있다고 볼 수 있다.

본 논문에서는 일단 프로세서칩 (processor chip)으로 읽혀진 데이터를 효율적으로 사용하기 위하여 데이터 접근의 시간적 근접성 (temporal locality)[8] 를 증가시키는 방향으로 소스 코드를 변환하는 방법을 제안한다. 즉 하나의 서포트벡터가 캐시나 레지스터 파일에 저장되는 동안 하나의 입력벡터가 아니라 여러개의 입력벡터를 위해 쓰이게 되어서 메모리 접근 횟수를 줄이게 된다. 적은 메모리 접근 횟수를 가지고 같은 일을 한다면 그것은 각 메모리 접근의 효율성을 높였다는 의미로도 생각될 수 있다.

본 논문은 다음과 같이 구성된다. Ⅱ장에서는 서포트벡터머신기반 음성/음악분류기에 대해서 설명하고 그의 데이터 접근 양식을 분석한다. Ⅲ장에서는 Ⅱ장의 분석을 바탕으로 보다 효율적인 데이터 접근 방법을 제안한다. Ⅳ장에서는 실험 결과를 보이고 Ⅴ장에서 본 논문을 끝맺는다.

Ⅱ. SVM기반 음성/음악 분류기의 데이터 접근 양식 분석

이번 장에서는 본 논문의 주제가 되는 SVM기반의 음성/음악 분류기의 데이터 접근 양식을 분석해본다. 먼저 분류의 기초가 되는 판별식은 다음과 같다.

$$f(\boldsymbol{X}) = \sum_{i=1}^{M} \alpha_i^* z_i K(\boldsymbol{X_i^*}, \boldsymbol{X}) + b^* \quad (1)$$

$\boldsymbol{X_i^*}$는 학습에 의해 구해진 M개의 서포트 벡터 (support vector) 중 i번째 벡터이다. 최적화 바이어스 (optimization bias) b^*와 라그랑제 승수 (*Lagrange* Multiplier) α^*는 학습에 의해 구해지는 quadratic programming problem의 해이다. 커널함수로 radial basis function (RBF)를 사용한다면 $K(\boldsymbol{X_i^*}, \boldsymbol{X})$는 다음과 같이 표현된다.

$$K(\boldsymbol{X_i^*}, \boldsymbol{X}) = \exp\left(-\frac{\| \boldsymbol{X_i^*} - \boldsymbol{X} \|^2}{\sigma^2}\right) \quad (2)$$

σ는 RBF의 폭과 관계된 파라미터이고, RBF를 커널 함수로 사용하는 이유는 RBF는 입력신호가 선형분류가 가능하지 않은 경우 선호되는 커널함수이기 때문이다. 식 (1)과 (2)로 구성되는 판별식에 대응하는 의사코드 (pseudo code)는 그림 2와 같다.

NUM$_{in}$: the number of input vectors
NUM$_{sv}$: the number of support vectors
NUM$_{feature}$: the number of features in a support/input vector
SV[ ]: each element of this structure contains a support vector and its corresponding alpha
IN[ ]: each element of this structure is an input vector
KF: kernel function output
F: decision function output

```
for (i=0;i<NUM_in;i=i+1) {
    for (j=0;j<NUM_sv;j=j+1) {
        for (k=0;k<NUM_feature;k=k+1) {
            dist+=(SV[j].feature[k]-IN[i].feature[k])^2
        }
        KF=exp(-1/σ^2 * dist)
        F+=SV[j].alpha*KF
    }
    F=F+bias
}
```

그림 2. SVM기반 분류기의 의사코드.
Fig. 2 Pseudo code for SVM-based classifier.

NUM$_{in}$, *NUM*$_{sv}$, *NUM*$_{feature}$는 각기 입력벡터의 개수, 서포트벡터의 개수, 그리고 서포트벡터와 입력벡터의 차원을 나타낸다. *SV*는 서포트벡터를 저장하고 있는 구조체로 차원이 *NUM*$_{feature}$인 서포트벡터와 그에 해당하는 라그랑제 승수로 구성되어진다. *IN*은 입력벡터를 저장하고 있는 구조체이다. 각 구조체 속의 벡터는 어레이로 저장되는데 그 이름은 *feature*이다. *dist*는 식 (2)에서 $\| \boldsymbol{X_i^*} - \boldsymbol{X} \|^2$에 해당하는 값이고, *KF*는 식 (2)로 표현되어지는 커널함수의 결과 값이고, *F*는 식 (1)의 좌변의 $f(\boldsymbol{X})$이다. 의사코드에는 세 개의 루프가 있는데, 첫 번째 루프는 입력벡터를 순차적으로 판별식에 대입하는 역할을 하고, 두 번째 루프는 각 입력벡터에 대해 모든 서포트벡터를 차례대로 대응시키고, 마지막 루프는 하나의 입력벡터와 하나의 서포트벡터의 벡터연산을 수행한다. 본 장의 분석 대상이 되는 서포트벡터는 각 입력벡터에 대해 벡터의 요소 단위로 순차적으로 불려서 연산에 사용된다. 다시 말하면, 하나의 입력벡터에 대하여 첫 서포트벡터의 첫 요소부터 마지막 서포트벡터의 마지막 요소까지 차례대로 읽혀지고, 다음 입력벡터에 대해서도 같은 순서로 읽혀진다. 이런 엑세스 패턴이 캐시와 레지스터파일을 어떻게 활용하는지 알아보기 위해 간단한 캐시구조와 서포트벡터, 2개의 입력벡터 *in1*, *in2*를 가지고 캐시와 레지스터파일에 저장되는 내용이 어떻게 변하는지 그림 3에 나타내었다. 이 그림을 얻기 위한 가정으로 *NUM*$_{in}$, *NUM*$_{sv}$, *NUM*$_{feature}$는 각각 2, 8, 7로 정해졌고, 서포트벡터의 메모리 엑세스만 고려하였다. 또한 캐시는 네 개의 32바이트 블럭으로 구성되어있다고 가정하였고, 서포트벡터는 *SV1*부터 *SV8*까지 연속적인 주소를 가지고 있다고 가정하였다. 즉 서포트벡터와 각 벡터의 요소들의 주소가 위의 가정하에 그림3 (a)에 표시된 것 처럼 정해졌다. 하나의 서포트벡터는 float 형태 (4바이트)의 요소 8개 (7개 요소를 가진 벡터 한개와 라그랑제 승수)로 구성되어 있으므로 32바이트인 캐시블럭에 완전히 저장되어진다.

그림에서 알 수 있듯이 SV1과 SV5는 모두 캐시블럭 0에 저장되어진다. 이렇게 각 캐시블럭에는 두 개의 서포트벡터가 대응된다. 첫 입력벡터 in1의 분류를 위해

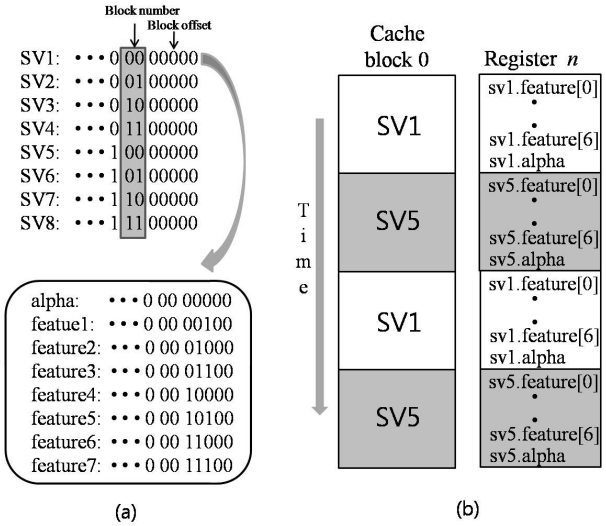


그림 3. (a) 서포트벡터의 주소 (b) 캐시블럭 0과 레지스터 n의 내용변화.

Fig. 3 (a) Support vectors' addresses (b) Contents of cache block 0 and register n.

k 루프의 SV1의 첫 요소를 읽는 명령어가 실행되면, 먼저 캐시에서 해당 데이터를 찾는다. 그러나 캐시의 블록 0는 그 값을 가지고 있지 않으므로 메모리에서 SV1 전체를 불러들여 캐시블럭 0에 저장하고 벡터의 첫 요소인 *feature[0]*를 레지스터에 저장하게 된다. *k* 루프가 진행되면서 *SV1*의 모든 요소가 하나씩 순서대로 레지스터 *n*에 저장되고 *k* 루프가 끝나고 *j* 루프가 다음 반복(iteration)을 시작하기 전에 *SV1*의 라그랑제 승수가 레지스터로 불려진다. 새로운 *j* 루프의 반복이 시작되면, *SV1*의 경우와 같은 양식으로 *SV5*가 캐시블럭 0에 *SV1*를 대체하고 저장되며 레지스터 파일에 *SV5*의 첫 요소부터 라그랑제 승수까지 순차적으로 불려 들여진다. 그리고 *in2*에 대해서도 *SV1*과 *SV5*가 순차적으로 캐시와 레지스터파일로 불려 들여진다. 두 개의 입력벡터 *in1*과 *in2*를 분류하는데 *SV1*과 *SV5*를 읽는데 발생한 메모리 접근과 캐시 접근의 횟수를 세어보면 메모리는 4번 접근됐고 캐시는 32번 접근되었다.

데이터 접근의 공간적 근접성 (spatial locality)[8]이란 현재 어느 한 데이터가 접근되었다면 그 데이터와 인접한 다른 데이터가 근 미래에 접근이 된다는 성질을 말한다. 시간적 근접성 (temporal locality)[8]이란 현재 어느 한 데이터가 접근되었다면 그 데이터가 근 미래에 다시 접근되는 성질을 뜻한다. 캐시는 이러한 성질이 존재할 때 메모리로의 접근을 효과적으로 줄일 수 있다. 이 정의를 바탕으로 위의 예에서 보여진 데이터 접근을 살펴보면 공간적 근접성이 많이 존재하고 또한 캐시에 의해서 잘 활용되어지고 있음을 볼 수 있다. 반면에 시간적 근접성은 존재하지만 캐시에 의해서 전혀 활용되어지고 있지 못하고 있음을 볼 수 있다. 즉 캐시가 활용할 수 있을 만큼 시간적 근접성이 없다고 볼 수도 있다. 이를 해결하기 위하여 다음 장에서는 캐시가 시간적 근접성을 잘 활용할 수 있도록 메모리 접근 양식을 변화시켜, 메모리 접근의 수를 줄이는 기법을 소개한다.

Ⅲ. 시간적 근접성을 높이기 위한 기법

이번 장에서는 Ⅱ장에서 밝혀진 SVM기반 분류기의 일반적인 성향인 낮은 시간의 근접성을 향상시키는 방법을 제안한다. 캐시의 관점에서시간의 근접성이란 한 번 메모리로부터 읽혀져 캐시에 저장된 데이터가 다른 데이터에 의해 대체되기 전에 얼마나 프로세서 코어에 의해 사용되는가를 나타낸다. 그러므로 시간의 근접성을 높이기 위해서는 한번 캐시에 저장된 데이터를 여러 번 사용해야 한다. 그림 3에서 *SV1*을 살펴보면, *SV1*은 *in1*과 *in2* 모두를 위해 사용되므로 *in1*을 위해 캐시에 저장되었을 때 *in2*를 위해서도 사용한다면 시간적 근접성을 높일 수 있다. 즉 그림2의 의사코드에서는 한 번에 하나의 입력벡터를 분류하는 형태지만 한 번에 여러 개의 입력벡터를 동시에 분류하는 형태로 바꾼다면 시간의 근접성을 높일 수 있을 것이다. 이 관찰을 바탕으로 그림 2의 의사코드를 변형시키면 그림 4의 의사코드가 된다.

그림 4의 의사코드는 두 개의 입력벡터를 동시에 분류하기 위한 코드로 더 많은 입력벡터를 동시에 분류하기 위한 코드도 같은 방법으로 쉽게 얻을 수 있다. 두 개의 입력벡터를 동시에 처리하기 위하여 중간 값을 저장하는 변수들 (*dist2*, *KF2*)가 추가되었고 마지막 분류 값을 저장하는 변수 *F2*도 추가되었다. 가장 안쪽의 루프에서 입력벡터들은 각각 *i*와 *i*+1로 접근되고 입력벡터를 차례대로 대입시키는 가장 바깥쪽의 루프는 *i*를 각 반복 (iteration)마다 동시에 처리되는 입력벡터의 개수만큼 증가시킨다.

이 시간적 근접성이 개선된 코드를 실행시키면 그림 3에 사용됐던 것과 동일한 가정 하에 캐시 블록 0와레지스터 *n*의 내용의 변화는 그림 5와 같다.

그림 3 (b)의 경우는 입력벡터 *in1*과 *in2*에 대하여

```
NUM_in: the number of input vectors
NUM_sv: the number of support vectors
NUM_feature: the number of features in a support/input vector
SV[ ]: each element of this structure contains a support vector and
its corresponding alpha
IN[ ]: each element of this structure is an input vector
KF1, KF2: kernel function output
F1, F2: decision function output

for (i=0;i<NUM_in;i=i+2) {
    for (j=0;j<NUM_sv;j=j+1) {
        for (k=0;k<NUM_feature;k=k+1) {
            dist1+=(SV[j].feature[k]-IN[i].feature[k])^2
            dist2+=(SV[j].feature[k]-IN[i+1].feature[k])^2
        }
        KF1=exp(-1/σ^2 * dist1)
        KF2=exp(-1/σ^2 * dist2)
        F1+=SV[j].alpha*KF1
        F2+=SV[j].alpha*KF2
    }
    F1=F1+bias
    F2=F2+bias
}
```

그림 4. 시간적 근접성을 높인 SVM기반 분류기의 의사 코드.

Fig. 4 Pseudo code for the enhanced SVM-based classifier that generates more temporal locality.

*SV1*과 *SV5*를 읽는데 메모리는 4번 접근했고 캐시는 32번 접근된 반면, 그림 5의 경우는 메모리 접근이 2번, 캐시 접근이16번 일어난다. 즉 메모리 접근의 횟수가 줄고 캐시 접근의 개수도 줄어들게 되어 실행시간과 에너지 소비를 줄이는 효과가 있다. 다음 장에서는 실험을 통하여 제안된 기법을 검증해 본다.

Ⅳ. 실 험

1. 실험 설정

본 논문에서 사용된 음성/음악 데이터를 구성하기 위하여 음성 데이터베이스로 8kHz로 샘플링 된 약 6 sec 정도의 깨끗한 음성으로 326명의 남자와 138명의 여자 화자에 의해서 화자마다 10개의 파일이 발음된 TIMIT 데이터베이스가 사용되었다[9]. 음악 데이터베이스는 CD로부터 다섯 가지 장르의 음악을 모바일 폰을 통해서

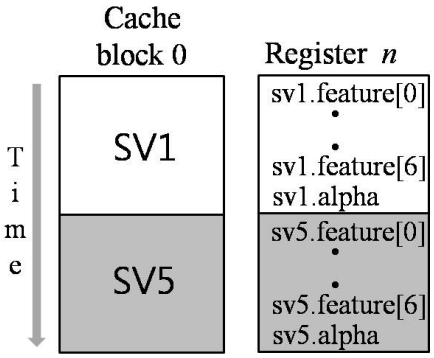


그림 5. 시간적 근접성을 높인 SVM기반 분류기에 의한 캐시블럭 0과 레지스터 n의 내용변화..

Fig. 5 Contents of cache block 0 and register n when the enhanced SVM-based classifier is used.

녹음하였고, 8kHz로 다운 샘플링 하여 사용하였으며, 각기 약 5분 정도의 길이를 가진다. 학습을 위해서 음성과 음악 데이터베이스에서 20분 정도의 데이터를 선별하여 사용하였으며, 테스트를 위해서는 학습에 사용되지 않은 데이터로 각 장르별로 2 개씩 모두 10개의 파일을 만들었다. 한 개의 파일에는 음성구간 20개 그리고 음악구간 20개가 포함되어 있으며 각 구간은 5초의 길이를 가진다. 음성과 음악구간은 서로 교대로 나타나게 구성되었다.

실험에 사용된 특징벡터로는 SMV[1]에서 추출되어지는 6개의 파라미터로 구성해 사용하였다[5]. SVM을 학습시키고 테스트 할 때 기본으로 사용되는 커널 width 파라미터 $1/\sigma^2$를 구하기 위해 여러 값을 시도하였고 그 중 가장 좋은 분류성능과 가장 적은 수의 support vector를 필요로 하는 0.01이 사용되었다.

SVM기반 음성/음악 분류기의 분류성능, 실행시간, 그리고 에너지소비를 구하기 위하여 사이클 (cycle)단위로 프로세서의 행동을 모델한 시뮬레이터인 sim-

표 1. 프로세서 시뮬레이터 파라미터.

Table 1. Parameters for the processor simulator.

ISA	ARMv4T
Clock Freq	800Mhz
Pipeline	5 stages, in-order, dual issue
Branch	Dynamic prediction (bimodal)
Cache	64KB separate Inst/Data cache,
	4way, 32B line, 1 cycle latency
Memory	64bit bus, 100 cycle latency, 1 port
FPU	1 adder (2 cycle, pipelined)
	1 Multiplier (4 cycle, pipelined)

panalyzer[10] 가 사용되었다. 이 시뮬레이터는 프로세서의 아키텍쳐 레벨의 모델링 뿐 아니라 프로세서의 각 부분에서의 정적 그리고 동적인 에너지 소비도 예측할 수 있는 에너지 모델도 포함하고 있다. SVM기반 음성/음악 분류기 코드를 ARM instruction set architecture (ISA)로 컴파일 하기위하여 gcc-2.95 기반의 교차컴파일러 (cross-compiler)를 사용하였고 시뮬레이션 되는 임베디드 프로세서의 구성은 표 1에 정리
하였다.

2. 실험 결과

제안된 알고리즘을 검증하기 위하여 알고리즘을 적용한 경우와 적용하지 않은 경우의 서포트벡터를 읽는
명령에 대한 캐시미스의 비율 (cache miss ratio), 실행된 총 명령어 (instruction)의 개수, 실행시간, 그리고 에너지 소비를 비교해 보았다. 먼저 표 2에 서포트벡터를 읽는 명령어에 대한 캐시 접근 수 (the number of cache accesses), 캐시미스의 개수 (the number of cache misses), 그리고 캐시미스비율 (cache miss ratio)를 정리하였다.

표의 1행은 동시에 분류되는 입력벡터의 수를 타나내고, 2행, 3행, 4행은 각각 하나의 입력벡터를 분류하는데 일어난 평균 캐시 접근 수, 평균 캐시미스의 개수, 그리고 평균 캐시미스의 비율 이다. 여기서 캐시미스의 개수는 메모리 접근의 개수와 같다.

제안된 기법을 사용하면 메모리로부터 캐시로 이동된서포트벡터를 여러 입력벡터에 사용할 수 있으므로 캐시 접근 수와 캐시미스의 개수는 줄어들게 된다. 그러나 캐시미스의 비율은 일정하게 유지되는 것을 볼 수 있는데, 이것은 서포트벡터의 첫 요소를 읽는 명령어에만 캐시미스가 발생하고 그 서포트벡터의 나머지 부분에는 캐시미스가 발생하지 않는 현상은 동시에 분류되는 입력벡터의 수와는 관계없이 동일하기 때문이다.

동시에 분류되는 입력벡터의 수를 변화시키며 하나의 입력벡터를 처리하는데 실행된 평균 명령어의 개수, 평균 실행시간, 그리고 평균 에너지 소비를 그림 6에 나타내었다. 분류의 정확도는 제안된 기법의 사용 여부에 관계없이 동일하므로 포함시키지 않았고 세가지 지표 모두 기법이 사용되지 않은 경우, 즉 동시에 처리되는 입력벡터의 수가 1인 경우에 정규화 되었다.

명령어의 개수와 에너지 소비량은 동시에 분류되는 입력벡터의 개수가 4개일 때 최소가 되고 개수가 늘어

표 2. 서포트벡터를 읽기위한 캐시 접근 통계.
Table 2. Statistics of the accesses to the cache for reading support vectors.

	1	2	3	4	5	6
access	88K	44K	30K	22K	18K	15K
miss	1.3K	6.4K	4.4K	3.2K	2.6K	2.2K
ratio	0.15	0.15	0.15	0.15	0.15	0.15

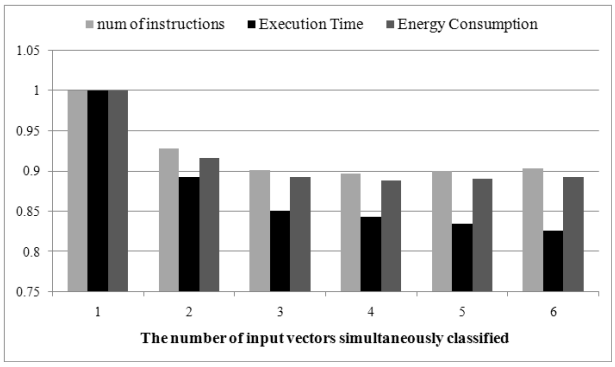


그림 6. 제안된 기법에 의한 명령어의 개수, 실행시간, 에너지소비의 변화.
Fig. 6 Effectiveness of the proposed technique in terms of the number of executed instructions, execution time, and energy consumption.

날수록 조금씩 증가한다. 그림 2의 의사코드와 그림 4의 의사코드를 비교해 보면, 동시에 분류되는 입력벡터의 개수가 많아질수록 명령어의 개수가 점점 증가하는 것을 알 수 있다. 반면에, 가장 바깥 루프의 반복회수는 동시에 분류되는 입력벡터의 개수가 많아질 수록 감소한다. 그러므로 동시에 처리되는 입력벡터의 수가 5일 때 이 두 가지의 균형이 명령어의 개수가 많아지는 쪽으로 기울었다고 볼 수 있다. 실행되는 명령어의 개수가 증가함에 따라 에너지 소비도 조금 증가한 것으로 보인다. 실행시간은 표 2에서 알 수 있듯이 동시에 분류되는 입력벡터의 수가 증가할수록 메모리 접근의 횟수가 계속 줄어드는 추세이므로 명령어의 증가에 영향을 덜 받는 것으로 보인다.

V. 결 론

본 논문에서는 SVM기반 음성/음악 분류기의 임베디드 시스템으로의 구현을 목표로, 먼저 SVM기반 분류기의 메모리 접근의 성향을 분석하여 시간적 근접성이 부족함을 밝혀내었다. 이를 바탕으로 메모리 접근의 시간적 근접성을 증가시켜 메모리 접근횟수를 줄이고, 실행시간과 에너지소비를 줄이는 기법을 제안하였다. 제

안된 기법을 검증한 결과, SVM기반 분류기의 실행시간과 에너지소비를 효과적으로 감소시킬 수 있음을 보였다.

앞으로의 연구과제로는 SVM기반 분류기의 공간적 근접성을 분석하고 이를 활용하는 효과적인 기법을 연구하는 것이다.

감사의 글

본 연구는 지식경제부 및 한국산업기술평가관리원의 IT핵심기술개발사업의 일환으로 수행하였음. [2009-S-036-1, 고성능 가상머신 규격 및 기술 개발]

또한 이 논문은 2011년도 정부(교육과학기술부)의 재원으로 한국연구재단의 대학중점연구소 지원사업으로 수행된 연구임 (2011-0022980)

참 고 문 헌

[1] 3GPP2 Spec., "Source-controlled variable-rate multimedia wideband speech codec (VMR-WB), service option 62 and 63 for spread spectrum systems," *3GPP2-C.S0052-A*, vol. 1.0, April 2005.

[2] Y. Gao, E. Shlomot, A. Benyassine, J. Hyssen, Huan-yu Su, and C. Murgia, "The SMV algorithm selected by TIA and 3GPP2 for CDMA appications," in *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 2, pp. 709-712, May 2001.

[3] J. Saunders, "Real-time discrimination of broadcast speech/music," in *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 2, pp. 7-10, May 1996.

[4] A. Bugatti, A. Flammini, and P. Migliorati, "Audio classification in speech and music: a comparison between statistical and a neural approach," *EURASIP Journal on Appliled Signal Processing*, vol. 2002, pp. 372-378, April 2002.

[5] S. -K. Kim and J. -H. Chang, "Speech/music classification enhancement for 3GPP2 SMV codec based on support vector machine," *IEICE Trans. Fundamentals of Electronics, Communications and Computer Sciences*, Vol. E92-A, no. 2, February 2009.

[6] S. -K. Kim and J. -H. Chang, "Discriminative weight training for support vector machine-based speech/music classification in 3GPP2 SMV codec," *IEICE Trans. Fundamentals of Electronics, Communications and Computer Sciences*, vol. E93-A, no. 1, pp. 316-319, January 2010.

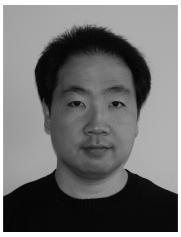
[7] J. Bsckus, "Can programming be liberated from the von Neumann style?: a functional style and its algebra of programs," *Communications of the ACM*, vol. 21, pp. 613-641, August 1978.

[8] J. L. Hennessy and D. A. Patterson, "Computer architecture: a quantitative approach," *Morgan Kaufmann Publishers Inc., San Francisco, CA*, 1990.

[9] W. M. Fisher, G. R. Doddington and K. M. Goudie-Marshall, "The DARPA speech recognition research database: Specifications and status," in *Proc. DARPA Workshop Speech Recognition*, pp. 93-99, February 1986.

[10] T. Austin, T. Mudge, and D. Grunwald, Sim -panalyzer. http://www.eecs.umich.edu/~panalyzer/.

저 자 소 개



임 정 수(정회원)

1996년 인하학교 전기공학 학사
2004년 메릴랜드주립대학 전자공학 석사.
2009년 노스캐롤라이나주립대학 컴퓨터공학 박사
2010년 인하대학교 박사후 연구원
2011년 목포대학교 연구교수
<주관심분야 : 컴퓨터 구조, 임베디드 시스템, 신호처리, 인공지능>

장 준 혁(정회원)

1998년 경북대학교 전자공학과 학사.
2000년 서울대학교 전기공학부 석사.
2004년 서울대학교 전기컴퓨터 공학부 박사.
2000년～2005년 (주)넷더스 연구소장
2004년～2005년 캘리포니아 주립대학, 산타바바라(UCSB) 박사후연구원
2005년 한국과학기술연구원(KIST) 연구원
2005년～2011년 인하대학교 전자공학부 조교수
2011년～현재 한양대학교 융합전자공학부 부교수
<주관심분야 : 음성신호처리, 오디오 신호처리, 통신 신호처리, 휴먼/컴퓨터 인터페이스>