임베디드 디바이스에서 음성 인식 알고리듬 구현을 위한 부동 소수점 연산의 고정 소수점 연산 변환 기법

*윤성락, 유창동
한국과학기술원 전자전산학과 전기 및 전자 공학 전공
e-mail : *yunsungrack@kaist.ac.kr, cdyoo@ee.kaist.ac.kr*

Automatic Floating-Point to Fixed-Point Conversion for Speech Recognition in Embedded Device

*Sungrack Yun, Chang D. Yoo
Department of Electrical Engineering
Korea Advanced Institute of Science and Technology

Abstract

This paper proposes an automatic conversion method from floating-point value computations to fixed-point value computations for implementing automatic speech recognition (ASR) algorithms in embedded device.

I. 서론

Fixed-point digital signal processors are required for the minimization of cost and power consumption and the maximization of working speed in implementing a signal processing algorithm. Most signal processing algorithms are developed with floating-point data types, but they should be converted into the fixed-point architectures to satisfy the cost, power consumption and working speed constraints of embedded systems. The automatic speech recognition (ASR) algorithm is also specified with floating-point data types. The proposed method uses integer variables to automatically convert the floating-point value computations into fixed-point value computations for speech recognition algorithms.

II. 본론

In speech recognition algorithm, the likelihood of an utterance is used which is defined as

$$l(x_t) = \sum_q \pi_{q_0} \prod_{t=1}^{T} a_{q_{t-1}q_t} b_{q_t}(x_t) \qquad (1)$$

where q_t is the state at time t, π_q is the initial probability of state q and $a_{q_{t-1}q_t}$ is the transition probability from state q_{t-1} and q_t, $b_{q_t}(x_t)$ is the state output probability of the feature vector x_t and T is the number of frames of the feature vector. The feature vector is extracted from an utterance by some feature extraction methods such as mel frequency cepstral coefficient (MFCC) and

linear prediction coefficient (LPC).

The feature extraction methods are specified with floating-point data types. To convert the floating-point data types into fixed-point data types, we propose the following structure named by INTEX.

INTEX	f/precision

Thus, an floating-point value is assigned into the integer variable f by multiplying a proper precision value. For example, 12.67893 is multiplied by a precision value 1000 and then assigned into f with the value 12678.

In equation (1), the multiplication of transition probability and state output probability are performed for T frames. These iterative multiplications cause $l(x_t)$ to be very low value since a probability is less than 1 and larger than 0. If $l(x_t)$ becomes too small, an underflow may occur. Thus, INTEX cannot be used in this case and another data structure for computing very low floating-point values without an underflow is required.

We propose a floating point data structure with extended range (FLOATEX) to compute very low floating-point values by using the following format.

FLOATEX	$1.\text{fraction} \times 2^{\text{exponent}}$

The fraction and exponent are 32-bit integer. Thus, it can present a value with order of $10^{-6.468e+8}$. With this precision, any underflow which occurs in speech recognition algorithm can be prevented.

Ⅲ. 구현

The data structures are implemented in C++. The operations associated with the floating-point data such as addition, multiplication and division are automatically converted into fixed-point operations by using the operator overloading capability of C++. Thus, they does not need any other manual conversions which are time-consuming tasks and tend to produce an error.

Ⅳ. 결론 및 향후 연구 방향

The performances were evaluated on an embedded devices by computing random 100,000 additions, multiplications and divisions. In table 1 and 2, the results are shown. The INTEX data structure is suitable for the floating point value with 10^{-3} precision and it is faster than the FLOATEX. The FLOATEX is slower than the INTEX but it supports the floating point value with $10^{-6.468e+8}$ precision. Computing floating point values in embedded device with both data structures were faster than the double data type.

		Time (sec.)
double	addition(subtraction)	4.5
	multiplication	4.48
	division	8.52
INTEX	addition(subtraction)	0.27
	multiplication	0.28
	division	0.47

Table 1. Performances of double and INTEX

		Time (sec.)
double	addition(subtraction)	4.5
	multiplication	4.48
	division	8.52
FLOATEX	addition(subtraction)	0.46
	multiplication	0.35
	division	0.71

Table 2. Performances of double and FLOATEX

참고문헌

[1] D. Menard and D. Chillet "Automatic Floating point to Fixed point Conversion for DSP Code Generation," CASES 2002.