A DECODER FOR LVCSR BASED ON FIXED-POINT ARITHMETIC

Enrico Bocchieri and Doug Blewett

AT&T Labs-Research,
Florham Park, NJ 07932, USA

ABSTRACT

The increasing computational power of embedded CPU's motivates the fixed-point implementation of highly accurate large-vocabulary continuous-speech (LVCSR) algorithms, to achieve the same performance on the device as on the server. We report on methods for the fixed-point implementation of the frame-synchronous beam-search Viterbi decoder, N-grams language models, and HMM likelihood computation. This fixed-point recognizer is as accurate as our best floating-point recognizer in several LVCSR experiments on the DARPA Switchboard task and on an AT&T proprietary task, with different types of acoustic front-ends and HMM's. We also present experiments on the DARPA Resource Management task using the StrongARM-1100 206 MHz CPU, where the fixed-point implementation enables real-time performance: the floating-point recognizer, with floating-point software emulation, is 50 times slower for the same accuracy.

1. INTRODUCTION

Large-vocabulary continuous-speech recognition (LVCSR) may find wide use in consumer, military and industrial applications using embedded platforms, such as PDA's, telephone handsets, network appliances, and wearable computers. For example, a potential application is in Short Message Services that has an expected global volume in excess of 1,000 billion messages in 2005. LVCSR on embedded platforms presents a unique set of challenges [1, 2]. In particular, to lower hardware cost and power consumption, for longer battery life and miniaturization, the CPU's do not have floating-point arithmetic units. However their computational power is constantly increasing. This motivates the study of the fixed-point implementation (for operation on the device) of high-accuracy LVCSR algorithms that are traditionally implemented on the floating-point server.

Relevant studies are [3, 4], concerning HMM parameter tying, [5, 6, 7] for the state-likelihood computation in fixed-point. Other issues such as recognition of large lists, front-end implementation, memory reduction, rapid porting are treated in [8, 9, 10, 11, 12, 13]. Previous works on fixed-point decoding concern either small-vocabulary continuous-speech tasks or large-vocabulary tasks with deterministic grammars. Instead, we focus on LVCSR tasks based on word N-gram language models. Sections 2 and 3 review the decoding problem and the fixed-point arithmetic. In Section 4 we propose a systematic approach to the fixed-point representation of the parameters of the recognizer components, including frame-synchronous Viterbi beam-search, with stochastic and deterministic language models, and HMM state and state-duration likelihood computations. The fixed-point recognizer is as accurate as the floating-point recognizer in LVCSR experiments (Section 5) on the Darpa Switchboard task and on fluently spoken telephone speech from an AT&T customer care application. The design is quite general, with the same fixed-point parametrization used for different acoustic front-end features, feature transformations, and HMM's (ML and MMI trained). The test data contains up to fifty words per sentence, and even for these long utterances, the accumulation of log-likelihoods scores during fixed-point decoding is not problematic. Our target is 32-bit integer CPU's (e.g. StrongARM), but the approach may be suitable for 16-bit CPU's with 32-bit accumulators as well. We also report on real-time recognition of the DARPA Resource Management task performed on a 206 MHz StrongARM CPU.

2. LVCSR MAP DECODER

Given the observation sequence $\mathbf{O} = (\boldsymbol{o}_1, ...\boldsymbol{o}_T)$, the maximum a-posteriori decoded word sequence $\Omega = (\omega_1, ...\omega_M)$ is chosen to maximize $p(\Omega|\mathbf{O})\ p(\Omega)$. After well known steps:

$$\hat{\Omega} \;=\; \arg\max_{\Omega}\ \mathcal{F}(\Omega)\ ,\ \text{with decoding function } \mathcal{F}:$$

$$\mathcal{F} \;=\; ln(\mathcal{L}) + \alpha\, ln(\mathcal{A}) + \beta\, ln(\mathcal{D}) \quad , \quad (1)$$

$\mathcal{L}$: likelihood of language model,

$\mathcal{A}$: likelihood of the HMM states (acoustic model),

$\mathcal{D}$: likelihood of the HMM state durations,

$\alpha,\ \beta$: state and state-duration model multipliers.

2.1. Language Model And $C \circ L \circ G$ Transducer

We encode the language model probabilities by the arc costs of a $C \circ L \circ G$ transducer [14] that represents the language model, either a word N-gram or a deterministic grammar, the lexicon, with pronunciation probabilities, and the phonetic context dependencies. For generality, by $ln(\mathcal{L})$ we denote

the negative total cost of the transducer path, induced by Ω:

$$ln(\mathcal{L}) \;=\; -\sum_{arc\,\in\,path(\Omega)} cost_{arc} \qquad (2)$$

2.2. HMM State Likelihoods

The generic HMM state s is a weighted mixture of N_s Gaussians with diagonal covariances ($\boldsymbol{\sigma}$ denotes the vector of stdv's):

$$P(\boldsymbol{o}\,|s) \;=\; \sum_{i=1}^{N_s} w_{s,i}\,\mathcal{N}(\boldsymbol{o}\,;\boldsymbol{\mu}_{s,i},\boldsymbol{\sigma}_{s,i}) \;\approx\; \max_{i=1,N_s} w_{s,i}\,\mathcal{N}(\boldsymbol{o}\,;\boldsymbol{\mu}_{s,i},\boldsymbol{\sigma}_{s,i})$$

Given the sequence $S{=}(s_1,..,s_T)$ of states aligned to $\mathbf{O}$, the total state likelihood contribution to (1) is $\alpha\,ln(\mathcal{A})$:

$$\sum_{t=1}^{T}\frac{\alpha}{2}\max_{i=1,N_{s_t}}\left(2c_{s_t,i}+\sum_{j=1}^{d}\left(\left(\boldsymbol{o}_t^{\,j}-\boldsymbol{\mu}_{s_t,i}^{j}\right)\boldsymbol{\sigma}_{s_t,i}^{\,j\;-1}\right)^2\right) \qquad (3)$$

d	:	feature vector dimension
$\boldsymbol{o}_t^{\,j}, \boldsymbol{\mu}_{s,i}^{j}, \boldsymbol{\sigma}_{s,i}^{\,j}$	:	j^{th} component of $\boldsymbol{o}_t, \boldsymbol{\mu}_{s,i}, \boldsymbol{\sigma}_{s,i}$
$c_{s,i}$	:	$ln(w_{s,i}) - \sum_{j=1}^{d} ln(\boldsymbol{\sigma}_{s,i}^{\,j}\sqrt{2\pi})$

2.3. State Duration Model

We denote with $p(\delta|\psi)$ the probabilities of the duration of δ-frames for state ψ. They are estimated (assuming state independence) as gamma p.d.f's, and stored in look-up tables. Given the states $\Psi{=}(\psi_1,...\psi_\Theta)$, with durations $\Delta{=}(\delta_1,..\delta_\Theta)$, the contribution of the duration model to (1), is:

$$\beta\,ln(\mathcal{D}) = \beta\,ln(P(\Delta|\Psi)) = \sum_{\theta=1}^{\Theta}\beta\,ln(p(\delta_\theta|\psi_\theta)) \qquad (4)$$

3. FIXED-POINT ARITHMETIC

In fixed-point, a decimal number x is stored in a computer word as an integer, where the p_x least-significant bits contain the fractional part of x, and the most-significant bits the integer part. This representation is said to have $Q{-}p_x$ format. In general, the choice of p_x is a trade-off between *truncation* errors and *overflow* problems. Fixed-point arithmetic is based on integer operations. In $z = x{\pm}y$, the relation $p_x = p_y = p_z$ applies. The product $z = x \times y$ is in format $Q{-}(p_x + p_y)$. Arithmetic shifts may be used to change the Q fixed-point format , to reduce truncation errors and to avoid overflow.

3.1. Fixed-Point And Linear Quantization

The function $nearest\_integer(2^p\;x)$ gives the $Q{-}p$ fixed-point format of decimal x. Suppose that we want to quantize the range of decimal values $[a,b]$, using m bits, e.g. to the range of integers $[-2^{m-1}, 2^{m-1})$. We follow the procedure:

---

m : bits for the d quantizers of means $\boldsymbol{\mu}_{s,i}^{\,j}$, $j = 1,d$.

v : bits for the d quantizers of $\boldsymbol{\sigma}_{s,i}^{\,j\;-1}$, $j = 1,d$.

e : fixed-point $Q{-}e$ format for
- normalized error: $(\boldsymbol{o}_t^{\,j} - \boldsymbol{\mu}_{s_t,i}^{j})\,\boldsymbol{\sigma}_{s_t,i}^{\,j\;-1}$

and $Q{-}2e$ format for:
- HMM state log-likelihoods,
- Acoustic duration model log-likelihoods,
- $C \circ L \circ G$ fsm costs,
- Cumulative log-probabilities during decoding, and related parameters, such as beam threshold

---

Table 1. Fixed-point parameters m, v and e.

i. Optional. *Demean* decimal values, by subtracting $\frac{a+b}{2}$:

$$[a,b] \Rightarrow [-\frac{a+b}{2}\;,\;\frac{a+b}{2}]$$

ii. Find the largest integer p, such that

$$-2^{m-1} \;\le\; 2^p\;x \;<\; 2^{m-1}\;,\; x \;\in\; [a,b]$$

iii. Quantize $x \in [a,b]$ by $y = nearest\_integer(2^p\;x)$.

Step iii yields a fixed-point format of x with scale-invariant average truncation error, because of the choice of p in ii.

4. FIXED-POINT IMPLEMENTATION OF DECODING FUNCTION

Our fixed-point design of (1) is parametrized by e, m, and v, as summarized in Table 1. Intuitively, a crucial role in the state likelihoods (3) is played by the Mahalanobis distance,

$$\sum_{j=1}^{d}\left(\left(\boldsymbol{o}_t^{\,j}-\boldsymbol{\mu}_{s_t,i}^{j}\right)\boldsymbol{\sigma}_{s_t,i}^{\,j\;-1}\right)^2 \qquad (5)$$

and, its fixed-point format is specified as $Q{-}2e$. Since (3) accumulates (5) into the state log-likelihoods, we represent all the decoder log-likelihoods in $Q{-}2e$ format. Therefore, the HMM log-terms $2c_{s,i}$ in (3), are also $Q{-}2e$, and, the product by $\frac{\alpha}{2}$ maintains the $Q{-}2e$ format through an appropriate arithmetic shift. Similarly, the duration log-probabilities of (4) are $Q{-}2e$ fixed-point, and when multiplying by β, the $Q{-}2e$ format is maintained by arithmetic shift.

Also, the $cost_{arc}$ of (2) are $Q{-}2e$. Note that, in the fixed-point composition $C \circ L \circ G$, a simple integer addition implements the $\otimes$ operator in the *tropical* semiring, under the condition that the costs of the C, L and G are all $Q{-}2e$. Therefore the *delayed composition* feature [14] is implemented as in the floating-point decoder. This feature minimizes run-time memory and is usefull in applications such as [8].

Different approaches are suitable for the implementation of (5) in $Q{-}2e$ fixed-point. We square and accumulate terms:

$$\left(\boldsymbol{o}_t^{\,j}-\boldsymbol{\mu}_{s_t,i}^{j}\right)\boldsymbol{\sigma}_{s_t,i}^{\,j\;-1}\;,\; in\;Q{-}e\;format \qquad (6)$$

$sum\ =\ 0\ ,\ j\ =1$
$while(j \leq\ d)\ \{$
$tmp = (\boldsymbol{o}_t^j\ -\ \boldsymbol{\mu}_{s_t,i}^j)\ {\boldsymbol{\sigma}_{s_t,i}^j}^{-1}$ // Q–$(p^j + r^j)$
$tmp\ =\ tmp\ >>\ shift_j$ // change to Q–e
$sum\ =\ sum\ +\ tmp * tmp$ //sum is Q–$2e$
$j\ =\ j+1$
$\}$

Table 2. Pseudo-code for fixed-point implementation of (5)

that are computed using suitable integer representations of the HMM parameters. To account for the different dynamic ranges of the Gaussian mean components, we build a quantizer for every j^{th} ($j = 1, d$) component, as in i, ii and iii of Section 3.1, for the decimal range:

$$\left[\ \min_{\text{Gaussian } i\ ,\ \text{state } s} \boldsymbol{\mu}_{s,i}^j\ \ ,\ \ \max_{\text{Gaussian } i\ ,\ \text{state } s} \boldsymbol{\mu}_{s,i}^j\right]$$

Parameter m specifies the number of bits of the quantizers. We denote by Q–p^j the fixed-point format of $\boldsymbol{\mu}_{s,i}^j$, induced by the j^{th} quantizer that also defines the fixed-point format of $\boldsymbol{o}_t^j$, in (6). Similarly, we build another set of d quantizers, one for every ${\boldsymbol{\sigma}_{s,i}^j}^{-1}$ (the j^{th} inverse stdv component) using steps ii and iii of Section 3.1. Parameter v specifies the number of bits, output range $[0, 2^v)$, of these quantizers. We denote by Q–r^j the fixed-point format of ${\boldsymbol{\sigma}_{s,i}^j}^{-1}$, induced by its quantizer. The fixed-point format of the integer product (6) is therefore Q–$(p^j + r^j)$ that we change to Q–e with a right arithmetic shift of $shift_j = (p^j + r^j - e)$ bits, as shown in Table 2 (negative $shift_j$ implies a left shift). In practice, we can choose e, m and v, so that $shift_j \geq 0,\ j = 1, d$. It might also be worthwhile for higher computational speed, to implement (5) through multiply-add operations.

5. EXPERIMENTS

Central in our design is the fixed-point representation of (6), whose statistics are largely independent of the ASR task: after HMM maximum likelihood estimation (6) are Gaussians with zero mean and unit variance. Our hypothesys is that the decoder fixed-point parametrization does not need task-specific calibrations which we verify in recognition tests on:

SWBD: Darpa Switchboard task, tested on the 2003 real-time test set (recognition from first-pass only),

CCAPP: fluent telephone speech from a customer-care application, with word tri-gram language model (perplexity of 60), vocabulary of 7,000 words, and up to 50 words/sentence,

RM: Darpa Naval Resource Management, with word-pair grammar, speaker-independent task,

with feature types:

MFCC: mel-frequency cepstrum coefficients,

ASR System	Word Accuracy (%) Floating	Fixed
SWBD_MFCC-HDA_MMI	59.2	59.1
SWBD_MFCC-HDA_ML	56.7	56.5
SWBD_PLP-HDA_ML	55.7	55.6
CCAPP_MFCC-HDA-VTLN_MMI	80.5	80.6
CCAPP_MFCC-HDA_MMI	78.4	78.4
RM_MFCC-HDA_ML	96.4	96.4
RM_MFCC-DD_ML	95.7	95.6
RM_PLP-DD_ML	95.6	95.5

Table 3. Floating and fixed-point decoder accuracy.Fixed-point parameters: $e = 5$, $m = v = 8$

PLP: perceptual linear prediction cepstra,

and with feature transformations:

DD: cepstra with 1^{st} and 2^{nd} differentials, 39 components,

HDA: discriminative linear transformation, 60 components,

VTLN: vocal tract length normalization.

The HMM's are context-dependent triphonic models, estimated either by maximum likelihood (ML) or maximum mutual information (MMI). Training of the CCAPP and SWBD HMM's use 170 and 300 hours of audio, respectively. For example, the 1^{st} pass of the AT&T RT-03 Switchboard system, with MFCC features and discriminative transformation, and MMI-trained HMM, is denoted by SWBD_MFCC-HDA_MMI.

We want to compare the LVCSR accuracies of the fixed-point and of the floating-point recognizers, and we use a Pentium® 4 PC for this purpose. On the Pentium, fixed-point implementations may be faster than floating-point, as shown in [6] for the state likelihoods. Our target is the StrongARM CPU, and we have not optimized the fixed-point software for speed on the Pentium. However, the Pentium is convenient for measuring accuracies, because software running on the Pentium gives the same results as on the StrongARM. Table 3 shows that the accuracies of the fixed-point and the floating-point recognizers (equal beamwidth), are the same, within 0.1%, for all tasks. Means and variances can be linearly quantized to 5 bits, without significant loss of accuracy (Table 4). Additional compression may be obtained by non-linear quantization [7], at the cost of additional indirections in the computation. We wanted to quantize HMM means and variances to no more than 8 bits, to reduce the HMM memory storage to a relatively small fraction of total run-time memory use. The accuracy is unchanged (within 0.1%) for $1 < e < 7$ (Table 4). Accuracy suffers from truncation errors for $e \leq 1$, and from overflow problems for $e \geq 7$. Larger e's, would require normalization of the cumulative log-likelihoods in the Viterbi search. In any case the decoder operates correctly over a wide range of e , on the various tasks. We use 32-bit fixed point arithmetic, but the good performance for e as small as 2, suggests that the implementation is suitable for 16-bit CPU's with 32-bit accumulators.

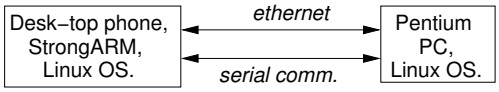
	v= 8	v= 7	v= 6	v= 5	v= 4	v= 3
m= 8	80.6	80.5	80.5	80.2	76.6	45.6
m= 7	80.5	80.6	80.4	80.2	76.5	45.7
m= 6	80.4	80.5	80.2	80.1	76.4	45.3
m= 5	80.3	80.2	80.1	80.0	76.3	45.0
m= 4	77.8	78.0	77.6	77.8	74.4	44.1
m= 3	56.4	56.6	57.0	59.9	53.3	43.5

Table 4. Word accuracy (%) as function of m and v (e= 5) of fixed-point system CCAPP_MFCC-HDA-VTLN_MMI.

e=8	e=7	e=6	e=5	e=4	e=3	e=2	e=1	e=0
69.2	79.9	80.5	80.6	80.5	80.5	80.5	80.2	75.6

Table 5. Word accuracy (%) as function of e ($m = v = 8$) of fixed-point system CCAPP_MFCC-HDA-VTLN_MMI.

We benchmarked the RM task on a StrongARM-1100, running at 206 MHz, embedded in a desk-top telephone set prototype, with 30 Mbytes of RAM, and Linux 2.6.6.



StrongARM executables are cross-compiled on the PC, with the GNU-toolchain and gcc-3.4.2. For testing ASR on the device, access to executables and to fixed-point speech feature files is through ethernet-NFS. Real-time recognition of the RM_MFCC-DD_ML fixed-point system on the StrongARM-1100 is shown in Figure 1 that plots the accuracy as a function of time (normalized by duration of input speech), for different beamwidhts (run-time memory use of 7.5 MBytes). Current generation embedded CPU's (e.g. 624 MHz XScale®) with 128 MBytes of RAM, could run in real-time more complex tasks, such as CCAPP. The fixed-point implementation is necessary for embedded ASR: we tested the floating-point decoder, cross-compiled with floating-point software emulation, and it was ≈ 50 times slower.

6. CONCLUSION

The presented fixed-point implementation of the LVCSR algorithms is as accurate as our best floating-point recognizer, in medium and large vocabulary continuous speech recognition tasks, as tested on the Pentium® 4 and StrongARM-1100 CPU's.

In addition to the more important motivation discussed in the Introduction, the algorithms are useful to prototype ASR applications in embedded systems. In fact, the decoder fixed-point parameters do not need critical task-dependent calibrations, and the LVCSR language and acoustic models, trained with the standard floating-point algorithms, can be automatically ported to the required fixed-point representation.

7. REFERENCES

[1] M.Novak, "Towards large vocabulary asr on embedded platforms," in *Proc. ICSLP'04*.

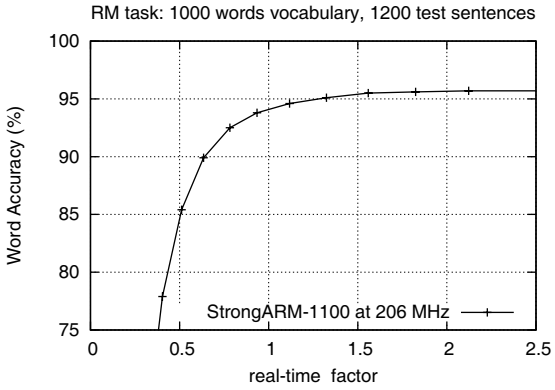


Fig. 1. Accuracy versus recognition time on the StrongARM.

[2] O.Viikki, "Asr in portable wireless devices," in *Proc. ASRU'01*, pp. 96–99.

[3] S. Sagayama and S. Takahashi, "On the use of scalar quantization for fast hmm computation," in *Proc. ICASSP'95*, pp. 213–216.

[4] E. Bocchieri and B. Mak, "Subspace distribution clustering hidden markov model," *IEEE Trans. on ASSP*, vol. 9, pp. 264–275, March 2001.

[5] M.Vasilache, "Speech recognition using hmm's with quantized paramaters," in *Proc. ICSLP 2000*, vol. 1, pp. 441–444.

[6] S. Kanthak, K.Schutz, and H.Ney, "Using simd instructions for fast likelihood calculation in lvcsr," in *Proc. ICASSP 2000*, pp. 1531–1534.

[7] J. Leppanen and I. Kiss, "Comparison of low footprint acoustic modeling techniques for embedded asr studies," in *Proc. INTERSPEECH '05*, pp. 2965–2968.

[8] M.Novak, R.Hampl, P.Krbec, and J.Sedivy, "Two-pass search startegy for large list recognition on embedded speech recognition platforms," in *Proc. ICASSP'03*.

[9] T.Kohler, C.Fugen, S.Stuker, and A.Waibel, "Rapid porting of asr systems to mobile devices," in *Proc. INTERSPEECH'05*, pp. 233–236.

[10] Y.Gong and Y. Kao, "Implementing a high accuracy speaker-independent continuous speech recognizer on a fixed-point dsp," in *Proc. ICASSP 2000*, pp. 3686–3689.

[11] Y.H. Kao and P.K. Rajasekaran, "A low cost dynamic vocabulary speech recognizer on a gpp-dsp system," in *Proc. ICASSP 2000*, pp. 3215–3218.

[12] J.Jeong, I.Han, E.Jon, and J.Kim, "Memory and computation reduction for embedded asr systems," in *Proc. ICSLP'04*.

[13] R.Rose, S.Parthasarathy, B.Gajic, A.Rosenberg, and S.Narayanan, "On the implementation of asr algorithms for hand-held wireless mobile devises," in *Proc. ICASSP'01*, 2001.

[14] M. Mohri, F. Pereira, and M. Riley, "Weighted finite-state transducers in speech recognition," *Computer, Speech and Language*, pp. 16(1):69–88, 2002.

8. ACKNOWLEDGEMENT

We thank C.Allauzen for the compilation of the FSM library with integer costs, and S.Shivappa for HMM training on PLP features.