Implementing a Speech Recognition Algorithm with VSIPL++

Don McCoy, Justin Voo, Stefan Seefeld, Brooks Moses
Embedded Software Division
Mentor Graphics
{don, justin, stefan, brooks}@codesourcery.com

Introduction

The VSIPL++ standard is designed to provide high programmer productivity and program portability, while maintaining high performance. We have successfully shown that it can be applied to a Synthetic Aperture Radar benchmark program and obtain high performance on a wide range of target platforms.

In this paper, we consider a different application of speech recognition, showing that the VSIPL++ API is useful beyond radar applications. We also focus on programmer productivity and source-code complexity in this paper, as there are mature open-source speech-recognition toolkits that provide a realistic basis for comparison.

The VSIPL++ API

The VSIPL++ standard [1] defines a C++ library API for developing high-performance signal and image-processing programs in a portable high-level manner. It provides an interface for multi-dimensional array data, as well as common operations (FFTs, convolutions, linear-algebraic operations and solvers, elementwise operations, and so forth) on that array data.

The high level of the functions and operations in VSIPL++ has advantages in programmer productivity, but it also is important for achieving high performance in a portable manner. When an algorithm can be expressed as a small number of compute-intensive function calls, the library implementation has freedom to implement these functions in ways that effectively use the parallel capabilities of a wide range of hardware. In previous papers [3, 4] we have demonstrated the effectiveness of this in practice, showing how Sourcery VSIPL++ can achieve high performance from the same program on both Cell/B.E. and NVIDIA CUDA hardware.

Thus, an important criterion for the applicability of VSIPL++ to a computational algorithm is how well the algorithm can be expressed in terms of large compute-intensive operations that are supported in the VSIPL++ API.

Existing Speech Recognition Implementations

To measure the effectiveness of VSIPL++ in representing speech-recognition algorithms, we need a basis for comparison. In this study, we consider two: the Hidden Markov Model Toolkit (HTK) [5], and the Probabilistic Modeling Toolkit for Matlab/Octave (pmtk3) [6].

HTK is a research implementation, written in C for high performance. As such, it represents a real-world example of production code, and illustrates both the complexity of the algorithms to be implemented and the complexity of program required to obtain high performance in a low-level language.

By contrast, ptmk3 contains reference implementations of the algorithms in Matlab, optimized for readability – and thus represent an ideal high-level expression with regards to simplicity rather than performance.

The ideal, then, would be to obtain performance comparable to the hand-optimized low-level code of HTK with the readability of the high-level ptmk3 code, and these two packages provide a good basis for comparing how close we can come to that ideal with VSIPL++.

Basics of Automated Speech Recognition

In a Hidden-Markov-Model–based automated speech recognition system, parts of speech – either whole words or phonemes, depending on the target application – are modeled as Markov processes that progress through a sequence of hidden states and produce output sound characteristics in a probabilistic manner. Thus, the recognition process involves computing for each part of speech the probability that the measured sound characteristics were generated by the corresponding Markov model.

The first step is to reduce the incoming audio stream into a discrete sequence of characteristics, ideally in a way such that the characteristics are small yet retain the data critical to distinguishing different phonemes. Typically, this is done by separating the audio stream into short windowed samples (e.g., 5ms long) and computing the mel-frequency cepstral coefficients (MFCCs) of the samples. The MFCCs are computed by taking a cosine transform of the power spectrum over a non-linear mel-scale of frequencies – thus, in effect, a spectrum of a spectrum.

More precisely, the power spectrum is first computed with a Fourier transform of the input signal, and this is binned into mel-scale frequency bins using what amounts to a sequence of bandpass filters:

$$\Phi_i = \boldsymbol{w}_i \cdot \|FFT(\boldsymbol{x})\|$$

$$\boldsymbol{\Phi} = \{\Phi_1, \Phi_2, \dots, \Phi_N\}$$

where $\boldsymbol{x}$ is the (windowed) input signal, $\boldsymbol{w}_i$ are the filter coefficients for the *i*th frequency bin in the mel scale, and $\boldsymbol{\Phi}$ is the power spectrum over the frequency bins. The MFCC coefficients $\boldsymbol{c}$ are then computed with a discrete cosine transform of the power spectrum:

$$\boldsymbol{c} = DCT[\log(\boldsymbol{\Phi})]$$

In a typical case, the overall characteristic vector for each sample might be a small number of frequency bins along with the first- and second-order differences of these bins in time.

The second step is to compute the best match between the computed sequence of characteristics and the Markov models for the parts of speech, using a Viterbi algorithm. The models each consist of a small number of states (six is a typical number), transition probabilities between these states, and probabilities that a given output characteristic vector will be produced by a given state. Thus, for a sequence of states

$X = \{x_1, x_2, \dots, x_T\}$ and a sequence of observations $O = \{o_1, o_2, \dots, o_T\}$, the joint probability that the sequence of observations is generated by state sequence X for model λ is given by

$$P(O, X|\lambda) = \pi_{x_1} b_{x_1}(o_1) a_{x_1 x_2} b_{x_2}(o_2) \cdots a_{x_{T-1} x_T} b_{x_T}(o_T)$$

where π_i are the initial probabilities of state *i*, $b_i(o)$ are the probabilities of state *i* producing observation *o*, and a_{ij} are the probabilities of transitioning from state *i* to state *j*.

The Viterbi algorithm computes the probability $P^*(O|\lambda)$ associated with the most likely sequence of hidden states. For each observed sample *t* in the sequence, the probability $\delta_t(j)$ of the most likely sequence that ends in state *j* can be computed as

$$\delta_{t+1}(j) = \max_i \left[a_{ij}\, \delta_t(i) \right] b_j(o_t)$$

This is then iterated over the observed samples up to time *T*, and the final probability for the most likely sequence is then

$$P^*(O|\lambda) = \max_i [\delta_T(i)]$$

The models are then ranked according to the P^* values, potentially with additional weighting from linguistic analysis, and the highest-ranked model determines the recognized part of speech.

Implementing Speech Recognition in VSIPL++

The MFCC algorithm fits very well into VSIPL++; the primitive operations are FFTs, dot products, and elementwise logarithms – all of which are expressed as VSIPL++ primitives. Moreover, there is an inherent data parallelism in the MFCC computations, which can also be expressed in VSIPL++. If the incoming signal is stored in a two-dimensional matrix, with each windowed sample stored in a row of the matrix, the VSIPL++ **`Fftm`** and **`vmmul`** operators can be used to apply the FFTs and dot products to all rows of the matrix in a single operation.

The Viterbi algorithm, which is the most computationally-intensive portion of the speech recognition process, fits less-well into VSIPL++. There are three levels of data parallelism – the inherent fine-grained parallelism in computing the products and maximum value, the parallelism in computing the best-path probability for each state, and the coarse-grained parallelism in computing the best-path probability for each model. Only the finest level of parallelism, in computing the $\delta_{t+1}(j)$ values, can be expressed in terms of VSIPL++ operations. Unlike with the MFCC computation, there are no operators for computing the necessary maximum values on multiple rows of a matrix simultaneously, and so the higher levels of parallelism must be expressed in necessarily-serial loops.

Thus, we anticipate that a VSIPL++ implementation of the Viterbi algorithm will provide reasonably good performance on platforms which exploit fine-grained parallelism, such as a single-core CPU, but it will not be able to effectively use platforms such as GPUs or multi-core CPUs which exploit coarse-grained parallelism.

Enhancements to VSIPL++

In the MFCC computation, we saw that the existence of VSIPL++ operators that apply the same operation to each row (or column) of a matrix are very powerful in expressing the parallelism of the computation in a way that the library can take advantage of to enable high performance. We have been experimenting with extensions to the VSIPL++ API that generalize this concept to other operators and combinations of operators, so that algorithms such as the Viterbi calculation may be expressed in a similar manner. In our presentation, we will describe these extensions and the results of applying them in this case.

Anticipated Results and Conclusions

We will compare line-count, code complexity, and performance of the C, Matlab, and VSIPL++ implementations of the speech processing algorithms, illustrating the strengths and weaknesses of the VSIPL++ API in these cases, and contrasting it to the ideal of Matlab simplicity with optimized-C performance. We will also discuss a programming model in which the bulk of the program is written in VSIPL++, and then a few small key inner loops are rewritten in low-level code to obtain maximum performance.

We expect that, in doing so, we will show that VSIPL++ is effective as an API in these real-world applications, and that it can significantly improve readability and programmer productivity.

References

[1] CodeSourcery, Inc. VSIPL++ Specification 1.02. Georgia Tech Res. Corp. 2005 [online] http://www.hpec-si.org.

[2] CodeSourcery, Inc. Sourcery VSIPL++. [online] http://go.mentor.com/vsiplxx.

[3] J. Bergmann, M. LeBlanc, D. McCoy, B. Moses and S. Seefeld. Scalable SAR with Sourcery VSIPL++ for the Cell/B.E. HPEC Workshop Proceedings. 2008. [online] http://www.ll.mit.edu/HPEC/agendas/proc08/agenda.html.

[4] D. McCoy, B. Moses, S. Seefeld, M. LeBlanc and J. Bergmann. Sourcery VSIPL++ for NVIDIA CUDA GPUs. HPEC Workshop Proceedings. 2009. [online] http://www.ll.mit.edu/HPEC/agendas/proc09/agenda.html.

[5] Hidden Model Markov Toolkit (HTK). [online] http://htk.eng.cam.ac.uk/.

[6] Probabilistic Modeling Toolkit for Matlab/Octave (pmtk3). [online] http://code.google.com/p/pmtk3/.

[7] Jurafsky, D. and Martin, J. *Speech and Language Processing, 2nd Edition.* Prentice Hall, 2008.

The authors would also like to thank Dr. Phillip De Leon of New Mexico State University for advice on speech-processing algorithms.