LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

UCRL-JRNL-230195

Speech recognition systems on the Cell Broadband Engine

Y. Liu, H. Jones, S. Vaidya, M. Perrone, B. Tydlitat, A. Nanda

April 23, 2007

IBM Journal of Research and Development

Disclaimer

This document was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor the University of California nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or the University of California. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or the University of California, and shall not be used for advertising or product endorsement purposes.

Speech recognition systems on the Cell Broadband Engine

Y. Liu, H. Jones, S. Vaidya, M. Perrone, B. Tydlitát, A. Nanda

In this paper we describe our design, implementation, and first results of a prototype connected-phoneme-based speech recognition system on the Cell Broadband Engine™ (Cell/B.E.). Automatic speech recognition decodes speech samples into plain text (other representations are possible) and must process samples at real-time rates. Fortunately, the computational tasks involved in this pipeline are highly data-parallel and can receive significant hardware acceleration from vector-streaming architectures such as the Cell/B.E. Identifying and exploiting these parallelism opportunities is challenging, but also critical to improving system performance. We observed, from our initial performance timings, that a single Cell/B.E. processor can recognize speech from thousands of simultaneous voice channels in real time—a channel density that is orders-of-magnitude greater than the capacity of existing software speech recognizers based on CPUs (central processing units). This result emphasizes the potential for Cell/B.E.-based speech recognition and will likely lead to the future development of production speech systems using Cell/B.E. clusters.

Introduction

Speech recognition has already been successfully integrated into many application areas and commercial products. Consider, for example, the Honda Acura** TL navigational system that responds to verbal queries, the Palm OS** 5 Voice Command recognition software for personal digital assistants (PDAs), the Motorola Bluetooth** Car Kit that includes voice recognition and automatic dial, or the Genesta speech-controlled portable computer. These products together demonstrate that speech recognition at interactive rates is viable even within the limited processing capabilities and resources of portable and embedded devices. However, many other applications require speech processing beyond interactive rates. Speech recognition systems in telephony applications for automatic call centers represent the largest segment of the speech processing market; these centers receive and must process thousands of telephone conversations. Similarly, in areas of data mining, such as intelligence and surveillance, there is also a growing interest in applying speech recognition to both online compressed speech channels and repositories of archival speech.

These systems must process many channels of speech at real-time rates and are generally constructed from clusters of processors based on commodity central processing units (CPUs). The number of nodes in such a cluster scales commensurately with the amount of speech traffic the system is expected to process. Using the current generation of processors, each node can manage roughly 20 to 30 speech channels in real time, and cluster sizes range from tens to thousands of nodes. System performance can also be scaled by incorporating more powerful processors. This is perhaps a more viable approach since recent trends show that streaming vector architectures, such as the Cell Broadband Engine** (Cell/B.E.), exhibit a better cost-performance ratio than traditional computer architectures for a variety of data-parallel applications. Implementing a speech system on the Cell/B.E., however, requires more effort than simply porting legacy source codes and then expecting automatic hardware acceleration to result only from compiler optimizations and special hand-tuned math libraries. Individual algorithms must be profiled and reformulated to explicitly expose areas of data parallelism amenable for a streaming and vector implementation. This is the approach we took in designing a prototype speech recognition engine on Cell/B.E. The results we observed were very surprising and encouraging: Our system performs roughly two orders of magnitude faster than existing speech systems.

Cell/B.E.

The Cell/B.E. is a new streaming heterogeneous multiprocessor architecture jointly designed by Sony, Toshiba, and IBM. This architecture is heterogeneous in the sense that it combines a general purpose IBM PowerPC* processing element (PPE) with several special purpose vector processing cores, called *synergistic processing elements* (SPEs). Each core executes on an independent instruction stream. Cell/B.E. also supports data streaming by providing explicit user management over the data communication via DMA memory transfers between the PPE main memory and the local store memories of the SPEs. Memory transactions can be interleaved with instruction execution, allowing their transfer latencies to be partially or completely concealed to improve pipeline efficiency.

This design provides Cell/B.E. with several interesting advantages over traditional processors. Many data-parallel tasks can be structured to expose single-instruction multiple-data (SIMD) parallelism, predictable memory access patterns, and data-independent processing. These parallel tasks generally execute much faster on the SPE processors than on the PPE processor. SIMD computations map directly to vector instructions, predictable memory access patterns allow prefetching of data elements, and data-independent processing enables a simplification of the vector execution pipeline (no need for complex branch-prediction strategies). Furthermore, whereas traditional processors employ caches to exploit data coherency, Cell/B.E. allows users to directly program the memory hierarchy and implement their own application-specific data caching policies. Streaming applications with completely predictable memory access benefit the most from user-managed caches and, when implemented correctly, can experience 100% cache hit performance. For further information on the Cell/B.E. architecture and its programming models, please refer to References [1] and [2].

Speech processing

Early analysts segmented speech signals into small windowed intervals and annotated them by phonemes (linguistically distinct speech sounds). This classification is possible because a speech signal looks roughly like a sequence of stationary waveforms. Analysts look at the waveforms and spectrogram plots and distinguish phonemes by examining their spectral characteristics (e.g., format frequencies) (**Figure 1**). Today, this analysis is completely automated by digital signal processing and pattern-matching algorithms.

Speech recognition systems generally consist of three components: feature extraction, pattern matching, and model training. These components work together to recognize the information being communicated by verbal speech. In a real-time system, the first two components require special optimization since this system has the special constraint that speech channels must be processed at line rates using a fixed amount of memory. Optimizations for model training are less important since the models only need to be trained once prior to any recognition activity. Efforts to optimize this step, however, are still worthwhile because training is an iterative process and can be very computationally expensive. We limit our discussion here to only the feature extraction and pattern-matching components.

Feature extraction

The feature extraction front-end takes a windowed speech frame from the speech audio waveform and from it derives a compact feature vector representation that captures important spectral and temporal properties. The most common features used by speech systems are the mel frequency cepstral coefficients (MFCC). MFCC features are based on the Fourier spectrum of the audio signal, mapped to a nonlinear frequency scale that roughly corresponds to the human perception of sounds. The first and second derivatives of this spectrum are also considered to measure the rate at which sounds change. The mean energy is subtracted and the variance is normalized to remove the channel transfer function.

There are 12 stages of processing:

1. Window frame extraction.
2. Mean subtraction.
3. Energy computation.
4. Preemphasis filtering.
5. Hamming window filtering.
6. Spectrum computation (using fast Fourier transform).
7. Mel frequency scale mapping.
8. Cepstrum computation.

9. Decorrelation (using discrete Fourier transform).
10. Cepstral filtering.
11. Cepstrum energy normalization.
12. First and second order derivatives.

The first processing stage starts with a windowed frame of 200 samples (25 ms of audio at 8 kHz) and the final result is a 39-component feature vector (12 MFCC, 1 energy, 13 first derivative, 13 second derivative). This processing is uniformly applied to overlapping frames (10 ms of overlap) in the speech signal to produce a sequence of MFCC feature vectors.

Pattern matching

Under this representation, new speech samples can be compared with reference samples by discovering and quantifying common patterns in their feature vector sequences. This is a test for similarity rather than equality, since speech samples are not expected to match exactly. Matches are scored using hidden Markov models (HMMs), which statistically summarize patterns over a reference set. The purpose of the pattern-matching component is to then evaluate or decode new speech samples against a set of HMMs.

HMMs

An HMM [3–5] models a stochastic temporal process with parameters that are not directly observable (hence *hidden*), but can only be inferred from the set of observation sequences that it generates (here, the observations are MFCC feature vectors). HMMs are graphically represented by a set of nodes and directed edges. The nodes represent states and edges represent transitions between states. Observation sequences are generated by paths between the start node and the end nodes. These are special states that do not generate observations. All other states generate observations whenever they are visited according to their probability density functions (PDFs). State transitions also occur probabilistically.

An HMM learns patterns over reference examples by assigning state PDFs and transition probabilities that maximize the probabilities of their sequence output, while also accommodating the variability of individual feature sequences. For example, constructing an HMM to recognize the word “one” requires several verbal samples of this word by different speakers. The pronunciation of this word could vary from speaker to speaker, and even the same speaker cannot exactly reproduce the same sounds twice. However, these pronunciations share common spectral and temporal patterns that are captured by the HMM through selectively strengthening paths and feature distributions in the network during the training process. Although HMMs cannot be explicitly trained using negative examples, discrimination is possible by comparing probabilities across all other models.

An HMM is scored against a new speech sample by evaluating paths through the HMM network. Multiple paths could generate the same feature sequence, so the likelihood for an HMM matching the feature sequence is given by the total of all possible path probabilities. This requires an exhaustive search through the network which can be efficiently computed using the Viterbi algorithm, explained in the next section.

Viterbi algorithm

A direct search of all paths in the network is not feasible computationally, so the Viterbi algorithm applies recursion to cache the intermediate path probabilities. This recursion can be efficiently implemented using dynamic programming. For each feature vector frame f_i, the algorithm examines each HMM state s_j and computes its *emission probability* $p(f = f_i \mid s_j)$ by evaluating the feature vector against the PDF of state s_j. All possible transitions into this state are then examined. Probabilities from previous states that transition into this one are multiplied with their respective transitional probabilities $p(s = s_j \mid s_k)$ and then summed together. This result is multiplied with the emission probability of state s_j to give the *total probability* $L_{i,j}$ of all intermediate paths between the start state and the current state that generate feature vector frames up to f_i. The initial conditions are set such that path computations begin at the start state for the first feature vector frame. The recurrence is given by

$$L_{i,j} = p(f = f_i \mid s_j)\sum_k L_{i-1,k}\, p(s = s_j \mid s_k), \tag{1a}$$

with the initial values set to

$$L_{0,0} = 1, \quad L_{0,j} = 0 \mid \forall j > 0 . \tag{1b}$$

Since the system obeys stochastic constraints, all path probabilities sum to unity. This means that probabilities of individual paths can be quite small. Therefore, it is useful to express these probabilities on a log scale. However, it is very expensive to add two numbers together in the log scale, i.e., computing log(a + b) directly from log(a) and log(b). To simplify matters, the maximal path is generally a good approximation to the summation of all possible paths. Using this approximation, Equations (1a) and (1b) can be approximated by

$$L'_{i,j} = \log\left(p(f = f_i \mid s_j)\right) + \max_k \left(L'_{i-1,k} + \log\left(p(s = s_j \mid s_k)\right)\right), \tag{2a}$$

with the initial conditions

$$L'_{0,0} = 0, \quad L'_{0,j} = -\infty \mid \forall j > 0\,. \tag{2b}$$

The goal of this computation is to evaluate $L_{\mathrm{m,n}}$, the probability that the feature sequence was generated by a path through the HMM. The approximation for the term $L'_{\mathrm{m,n}}$ is called the Viterbi probability and is computed recursively using Equations (2a) and (2b). Strictly speaking, $L'_{\mathrm{m,n}}$ is not a probability, but rather a likelihood. This likelihood value is sufficient for recognizing speech from samples that contain exactly one word unit (called the isolated digit recognition problem). However, in most practical recognition systems, the speech channels contain multiple words, and decoding from these channels (called the connected digit recognition problem) requires an additional trace-back step after computing $L'_{\mathrm{m,n}}$ to recover the maximal path through the HMM network and to identify the actual sequence of decoded words encountered along this path. Supporting this trace-back step requires that bookkeeping information, such as back-pointers and model labels, be maintained along with intermediate path likelihoods during the recursion. The Viterbi algorithm decodes HMMs against isolated digits, but recognizing connected digits requires searching hypothetical paths that pass through multiple HMM models. This search can also be organized efficiently using dynamic programming to extend the basic Viterbi algorithm. Such an approach, called the level-building algorithm (LBA), is discussed in the next section.

Speech system design

Constructing a speech recognition system requires modeling at two levels. At the highest level, the vocabulary of words and the language grammar that govern their syntactic use must be first decided. **Figure 2** is an example of a simple task of recognizing sequences of numeric digits. The vocabulary consists of the numbers “one” through “nine” and “oh”. The grammar allows any arrangement of digits in the sequence. A special silence model is also included to account for periods of silence (or background noise) between each number utterance. The set of numbers along with the silence model are all modeled by HMMs. The type of HMM (e.g., number of states and the allowable transitions between them) most commonly used in speech recognition is called the left-right HMM model. Here, the number of states roughly corresponds to the duration of the utterance, and the states are connected and arranged sequentially so that transitions occur only monotonically from left to right; that is, each state allows only self-transitions and forward-transitions. The PDF for each HMM state is generally modeled by a set of Gaussian functions over the feature vectors. This representation is called the Gaussian mixture model (GMM). The parameters of a GMM includes the Gaussian means and covariances (the feature vectors are decorrelated such the covariance matrices are diagonal) and weights for each Gaussian. Gaussian functions are commonly shared across multiple GMMs to reduce the model complexity, a technique called Gaussian parameter tying.

Decoding isolated digits amounts to evaluating the Viterbi probability of a speech sample against several HMM word models and selecting the best. Decoding connected digits is more challenging because the speech sample contains several words, and the word boundaries are unknown. The LBA solves this problem by evaluating multiple hypothetical intervals within the speech sample. The computation is organized into levels where each level corresponds to a single digit decode. The process begins by initializing all HMMs to decode starting at the first speech frame. The location of the ending frame for the first digit is unknown, so each HMM evaluates all speech frames thereafter as a potential candidate for the last frame of the first digit. In practice though, only a small interval past the first speech frame is searched since the word utterance is not expected to span the entire speech sample. The second level then evaluates each of the ending frames from the first level as a possible starting frame for the second digit, and this process proceeds so on until all speech frames are evaluated. During the course of the decode process, word transition probabilities (e.g., bigrams or trigrams) can be applied to enforce a local syntax. Back pointers are also kept to support the trace-back step, in which we work backwards from the last speech frame to recover all the word-level transitions that were made.

The algorithm just described has an exponential computational complexity, but captures the idea of decoding connected digits. In real-time systems, the amount of processing must be directly proportional to the size of the speech sample, and the amount of storage must be constant. Therefore, the decoding must occur synchronously with each speech frame, and only a small word transition history can be kept. For details about this approach, please refer to the frame-synchronous level building (FSLB) algorithm by Lee and Rabiner [6].

System implementation

Our speech recognition system on Cell/B.E. is implemented by three SPE kernel programs: `spe_extract`, `spe_computeobs`, and `spe_viterbi`. The PPE processor is responsible for initializing and loading data into the SPE kernels, invoking the SPE kernels, and final scoring. In the future, we will implement a scheduler on the PPE to analyze and distribute load across the SPE processors. The feature extraction frontend is implemented by `spe_extract` while the decoder is factored into two SPE programs: `spe_computeobs` and `spe_viterbi`. Our system processes a speech channel by calling each of the SPE kernels in sequence. Intermediate data is streamed between the SPE local store and PPE memory during successive SPE calls. The final scoring lattice from `spe_viterbi` is traversed by an FSLB implementation on the PPE to perform a trace-back step and recover the decoded text.

spe_extract

The design of the feature extraction is based on the pipeline from the Mississippi State Institute for Signal and Information Processing (MS ISIP) speech recognition toolkit [7], with the stages listed in the section on feature extraction. All of these steps are implemented within the resources of a single SPE program. The mean subtraction and energy computation across the speech window requires the summation of elements in the window. The computation for this sum is vectorized by laying out data elements as an array of 128-bit (4-component) SIMD vectors, and then performing the sum across the vectors. Elements in the resulting array are then combined together by dot product with a ones vector. The spectrum computation step is considered the pivot or core of the pipeline as it is the algorithm with the highest computational cost. Fortunately the Cell/B.E. software development kit library contains an extremely efficient fast Fourier transform (FFT) algorithm [8] which we judiciously apply. We profiled the FFT performance and determined that it completes eight FFTs in 3,800 cycles, which roughly accounts for 69 Gflops of computation and represents 34% efficiency on the Cell/B.E. Data vectorization occurs along the axis of a speech window frame; each block of four sequential data elements in the window are processed concurrently using vector instructions. However, the FFT routine expects a complex signal input in a format that interleaves real and imaginary components. Accommodating this data layout incurs only a small performance penalty to perform data interleaving and deinterleaving when moving data in and out of the FFT routine. Many of these stages require precomputed look-up tables. For example, FFT requires a table of twiddle factors to be precomputed for one of its parameters. Likewise, the discrete cosine transform (DCT) step, which decorrelates the MFCC vectors (to allow diagonal covariance matrices), and the various filtering operations also take advantage of precomputed factors. Using table lookups helps in both computation and accuracy as constant data terms can be computed only once and in higher precision. The first and second MFCC derivatives are computed by central differencing. Supporting this computation requires a short queue of MFCC frames to be maintained.

spe_computeobs

Evaluating the emission probability of generating a particular feature vector at a particular HMM state is independent of the network search, and this computation can be easily factored from the main decode. This is actually necessary for recognition tasks of higher complexity since the parameters for the GMMs occupy a significant amount of memory. Currently, we use four Gaussians per mixture (1 GMM per state) with 57 total HMM states. These parameters use roughly 71 KB of memory, which is almost a quarter of the SPE local store memory. Future implementations may require the parameter to be streamed into the local store. The end result of this kernel computation is a two-dimensional table (feature vector frames by HMM states) of emission probabilities. Since probabilities are expressed in log space, vectorizing this computation is very straightforward:

$$\log\left(\frac{1}{(2\pi)^{N/2}|\Sigma|^{1/2}}\exp\left(-\frac{1}{2}(x-\mu)^{\mathrm{T}}\Sigma^{-1}(x-\mu)\right)\right)=\log\left(\frac{1}{(2\pi)^{N/2}|\Sigma|^{1/2}}\right)-\frac{1}{2}(x-\mu)^{\mathrm{T}}\Sigma^{-1}(x-\mu)\,. \qquad (3)$$

The first term on the right-hand side of Equation 3 can be precomputed, and the second term can be computed by a dot product, since $\sum^{-1}$ is diagonal. The dot-product computation is vectorized by first multiplying the three vectors together componentwise using the 128-bit SIMD registers and then aggregating the result into a single value by summing across the vector, four components at a time. Evaluating the emission probabilities is the most arithmetic-intensive step and represents the bottleneck of our system.

spe_viterbi

The decoding process occurs frame-synchronously by permitting the predecessor state for a model start state to come from the end state of any HMM model (as allowed by the language grammar). This computation is vectorized along the HMM state axis by concatenating HMM models together and setting transitional probabilities across model boundaries to zero. The recurrence relations given in Equations (2a) and (2b) are processed using vector instructions for each block of four states. The layout for this computation is shown in **Figure 3**. Shaded groups of four elements in each column are stored in 128-bit registers and operated on by SIMD vector instructions. Evaluating the path probabilities requires only two columns of data to be stored at any given time. In addition to path probabilities, additional bookkeeping information is kept to support the trace-back process. Since all HMM models in our experiment are strictly first order left-right, data access to previous state probabilities is aligned by shifting the state column down by one state. Decoding proceeds by seeding the start states of each model with an initial probability and then streaming the emission probability table in and streaming the intermediate path probabilities out to main memory.

Results

To profile the performance of our recognizer, we set up a simple experiment to perform speaker-independent speech recognition of phonemes from a digit vocabulary based on the TIDIGITS corpus. This vocabulary includes the utterances “zero”, “one”, …, “nine”, and “oh” (as a substitution for “zero”), (meaning zero) by speakers of different gender and dialects, which are altogether modeled by 19 phonemes (each a three-state HMM with four Gaussians per HMM state). We used the MS ISIP speech decoder to provide model training and establish the baseline decoder performance. The platforms we used for testing are shown in **Table 1**.

The performance is measured by timing the latency to process a single-channel speech sample on a single SPE, and then extrapolating to the total number of physical SPEs available. This helps to estimate peak system performance under perfect load balancing and task scheduling. Speech recognition performance for these platforms is summarized in **Table 1**. Units are measured in real-time channels (RTCs), where 1 RTC = 1 s of audio/1 s of processing time.

On both the Cell/B.E. and software platforms, recognition accuracy was scored at 99%, which is to be expected for such a simple recognition task. Since recognition accuracy depends only on the training and language modeling, the performance of our prototype speech recognition engine on Cell/B.E. will extend to production systems because the SPE kernel programs were designed to scale with model and language complexity.

Conclusions

We have implemented and demonstrated a prototype speech recognition engine that is capable of processing approximately a thousand speech channels on a single Cell/B.E. processor. The kernel computations are designed to be highly scalable, and we expect this performance result to generalize well to commercial speech systems. We attribute the performance gains in our system mainly to the raw computational power and memory management of the Cell/B.E. We harness these resources by carefully choosing data layouts and reformulating algorithms to expose data parallelism and streaming opportunities.

Although the performance we measured pertains to only a simple digit recognition problem with a small vocabulary, the relative performance between CPU and Cell/B.E. systems is important to note. Speech recognition systems that incorporate very large vocabularies, complex grammar, and detailed GMMs decode channels at rates far below real time. Implementing these systems on Cell/B.E. allows not only the channel density to be scaled, but also the task complexity and recognition quality.

Future work

Having implemented the core algorithms in a basic speech recognition system, we identify three areas in which we can focus our future development efforts: language modeling, compressed speech, and speech activity detection.

Language modeling

Toward the longer-range goal of developing a production speech system, we plan to apply tools from the HTK 3.3 framework [9] to train HMM and language models to recognize speech from more complex and challenging sources. The simple experiment we conducted for this study did not include any language modeling; any digit can follow any other digit, and we made no attempt to construct actual words from the sequence of decoded phonemes. HTK is a collection of software utilities and tools to train, decode, and evaluate HMM models. We plan to start with the TIMIT corpus [10], which contains conversations from a finite dictionary and strictly follows a language grammar. After constructing and training the appropriate models, we will integrate them into our existing Cell/B.E. speech recognition system.

Compressed speech

The amount of speech traffic being transmitted over digital networks (e.g., Voice Over Internet Protocol) is rapidly outpacing our existing ability to efficiently process it. To communicate over a digital network, speech samples are first encoded by a lossy compression protocol. This compression step allows speech to be represented using a very low (fixed) bit rate, which increases the channel density given a target bandwidth, but also introduces significant noise and degradation to the original audio signals. Qualifying and quantifying how recognition accuracy is affected by compression artifacts is an interesting area of study. Furthermore, recent techniques have been proposed to derive MFCC features from the speech-encoding parameters and use the encoding parameters directly as a feature set. We expect to test both approaches and compare their results with a third approach, which is to compress and decompress audio samples (to artificially add compression noise) and apply speech recognition to establish a baseline. After establishing the best approach to computing features on compressed speech, we will integrate it into our existing feature-extraction pipeline on Cell/B.E.

Speech activity detection

Speech channels often contain long periods of with no speech. Removing these segments will not only help cull computation, but also improve recognition performance since speaker normalization is intended to be performed over voice activity. Identifying and annotating intervals of speech activity in voice channels is a binary classification problem; we are trying to classify speech from background. Therefore, models for both will be required. We plan to investigate approaches using linear classifiers, such as support vector machines, single-state HMMs, GMMs, or a hybrid combination of these approaches. We plan to integrate the best result in our Cell/B.E. speech-recognition pipeline.

This work was performed under the auspices of the U.S. Department of Energy by University of California, Lawrence Livermore National Laboratory under Contract W-7405-Eng-48.

[*] Trademark, service mark, or registered trademark of International Business Machines Corporation in the United States, other countries, or both.

[**] Trademark, service mark, or registered trademark of Honda Motor Company, Ltd., Bluetooth SIG, Inc., Sony Computer Entertainment, Inc., or Intel Corporation in the United States, other countries, or both.

References

1. J. A. Kahle, M. N. Day, H. P. Hofstee, C. R. Johns, T. R. Maurer, and D. Shippy, “Introduction to the Cell Microprocessor,” *IBM J. Res. & Dev*. 49, No. 4/5, 589–604 (2005).

2. A. Nanda, J. Moulic, R. Hanson, G. Goldrian, M. Day, B. D’Amora, S. Kesavarapu, and T. Sanuki, “Cell Blades: Building Blocks for Scalable, Real Time, Interactive and Digital Media Servers,” *IBM J. Res. & Dev*. **51**, No. 5, ___–___ (2007, this issue).

3. J. Picone, “Continuous Speech Recognition Using Hidden Markov Models,” *IEEE ASSP Magazine* 7, No. 3, 26–41 (1990).

4. J. W. Picone, “Signal Modeling Techniques for Speech Recognition,” *Proceedings of the IEEE* **81**, No. 9, 1215–1247 (1993).

5. L. R. Rabiner, “A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition,” *Proceedings of the IEEE* **77**, No. 2, 257–286 (1989).

6. C.-H. Lee and L. Rabiner, “A Network-Based Frame-Synchronous Level Building Algorithm for Connected Word Recognition,” *Proceedings of the International*

Conference on Acoustics, Speech, and Signal Processing, New York, NY, 1998, pp. 410–413.

7. A. Ganapathiraju, N. Deshmukh, J. Hamaker, V. Mantha, Y. Wu, X. Zhang, J. Zhao, and J. Picone, "ISIP Public Domain LVCSR System," *Proceedings of the Speech Transcription Workshop*, Linthicum Heights, MD, 1999; see *http://scholar.google.com/scholar?hl=en&lr=&q=cache:0xINGex23gEJ:www.isip.msstate.edu/publications/conferences/dod_lvcsr/1999/asr/doc/paper_v2.pdf+author:%22Ganapathiraju%22+intitle:%22ISIP+Public+Domain+LVCSR+System%22+.*

8. A. C. Chow, G. C. Fossum, and D. A. Brokenshire, "A Programming Example: Large FFT on the Cell Broadband Engine," *Proceedings of the Global Signal Processing Expo and Conference*, Santa Clara, CA, 2005; see *http://www-306.ibm.com/chips/techlib/techlib.nsf/techdocs/0AA2394A505EF0FB872570AB005BF0F1/$file/GSPx_FFT_paper_legal_0115.pdf.*

9. What is HTK?, University of Cambridge, Cambridge, U.K.; see *http://htk.eng.cam.ac.uk/.*

10. W. M. Fisher, G. R. Doddington, and K. M. Goudie-Marshall, "The DARPA Speech Recognition Research Database: Specifications and Status," *Proceedings of DARPA Workshop on Speech Recognition*, 1986, pp. 93–99.

Received 3-15-07; accepted for publication April 3, 2007

Yang Liu *Lawrence Livermore National Laboratory, 7000 East Avenue, Livermore, California 94550 (liu24@llnl.gov).* Dr. Liu is a computer scientist at the Lawrence Livermore National Laboratory (LLNL). He received a Ph.D. degree in computer science from the University of California, Davis, in 2004. His research interests are in computer graphics, scientific visualization, high-performance computing, and bioinformatics. Dr. Liu has directed his recent efforts at LLNL to applying hardware acceleration to data and computation-intensive algorithms using commodity computer processors.

Holger Jones *Lawrence Livermore National Laboratory, 7000 East Avenue, Livermore, California 94550 (holgerjones@llnl.gov)*. Mr. Jones is a projects team leader and senior developer at LLNL, with experience in signal processing, systems programming, distributed computing, control systems engineering, and scientific visualization. He received an M.S. degree in electrical and computer engineering from the University of California, Davis, in 2002.

Sheila Vaidya *Lawrence Livermore National Laboratory, 7000 East Avenue, Livermore, California 94550 (vaidya1@llnl.gov)*. Dr. Vaidya is an embedded computing and data processing solutions program leader at LLNL. She has a background in high-performance computing, information technology, and digital imaging, and extensive experience in microelectronics systems and technology, semiconductor devices, integrated-circuit design and fabrication, and chip-manufacturing infrastructure. She received a Ph.D. degree in materials science and solid state physics from the State University of New York, Stony Brook, in 1979. She has over 100 scientific publications and 14 patents. Dr. Vaidya is currently responsible for developing embedded computing and data processing solutions for national security applications at LLNL.

Michael Perrone *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (mpp@us.ibm.com).* Dr. Perrone is an IBM Master Inventor and the manager of the Cell/B.E. Solutions department, which has the mission of identifying and optimizing high-affinity workloads for the Cell/B.E. and other multicore processors. Current projects include high-performance computing workloads, seismic imaging, network intrusion detection, digital content creation, rich media mining, image analysis, speech recognition and bioinformatics. He received a Ph.D. degree in physics from Brown University. Dr. Perrone's research includes algorithmic optimization for the Cell/B.E. processor, parallel computing, and statistical machine learning.

Bořivoj Tydlitát *IBM Czech Republic, Voice Technologies and Systems, V Parku 2294/4, 148 00 Praha 4, Czech Republic (borivoj_tydlitat@cz.ibm.com).* Mr. Tydlitát received an M.S. degree in computer engineering from the Czech Technical University, Prague. He has worked at the IBM Thomas J. Watson Research Center on multiple projects related to speech recognition and natural language understanding. Mr. Tydlitát is currently a member of the IBM Research team in Prague, working on the development of embedded IBM ViaVoice* speech recognition software and participating in speech technology research.

Ashwini Nanda *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (ashwini@us.ibm.com).* Dr. Nanda is a Research Staff Member currently leading research and technology strategy on the Cell/B.E. processor-based systems, clusters, and their applications. He is also the chief architect of the Cell/B.E. processor blades prototype and roadmap in IBM. Dr. Nanda has been a co-general chair of the International Symposium on High Performance Computer Architecture (HPCA-7), served on the editorial board of *IEEE Transactions on Parallel and Distributed Systems*, and co-edited a special issue of *IEEE Computer* magazine. He holds ten patents and has published over 40 papers on computer system architecture, design, and performance.

* * * * *

Table 1 Speech recognition performance.

Platform	*Processor*	*Real-time channels (RTCs)*
SIM	4.0 GHz Cell/B.E. simulator	1,759
Cell/.B.E.	2.4 GHz Cell/B.E. hardware	1,216
Sony PlayStation** 3 (PS3)	3.2 GHz Playstation 3 (6 SPEs)	526
Central processing unit (CPU)	3.2 GHz Intel Pentium** 4	10

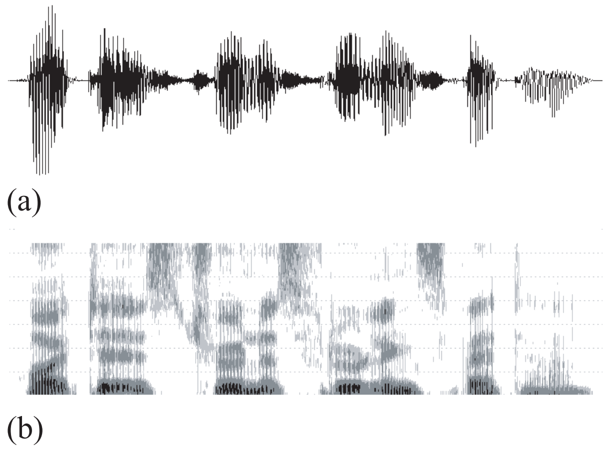

Figure 1
(a) Oscillogram and (b) spectrogram for, "Heute ist schönes Frühlingswetter." (It's nice weather today.)

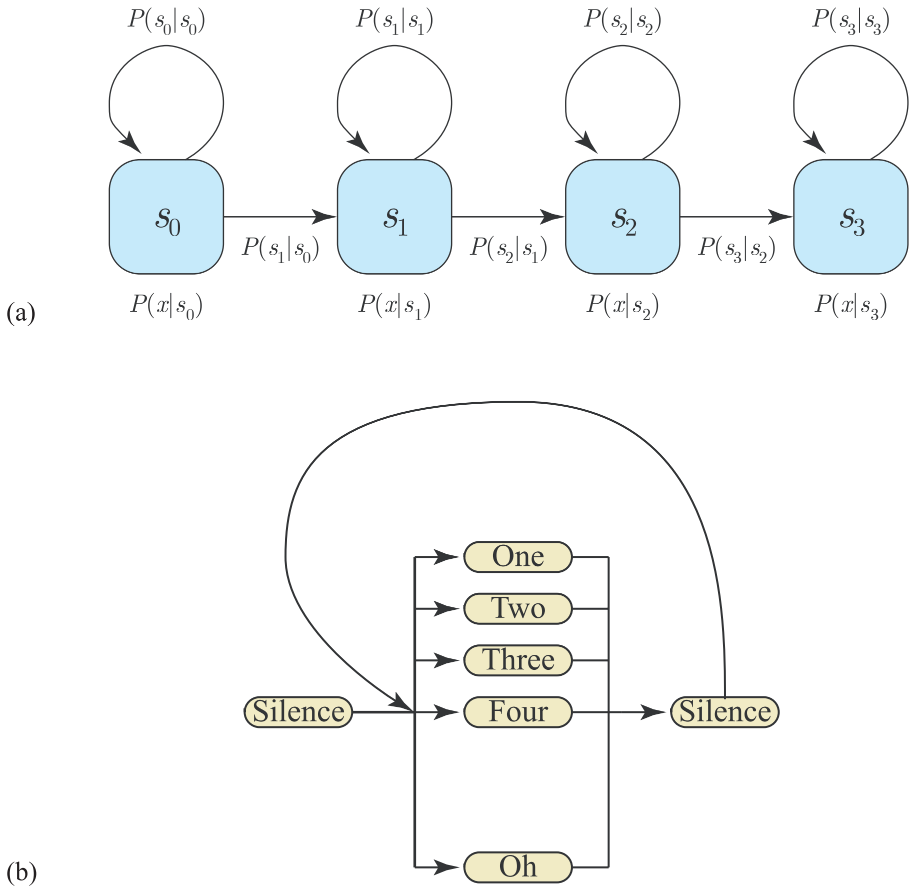

Figure 2
(a) First order left-right HMM model and (b) HMM system for recognizing connected digits.

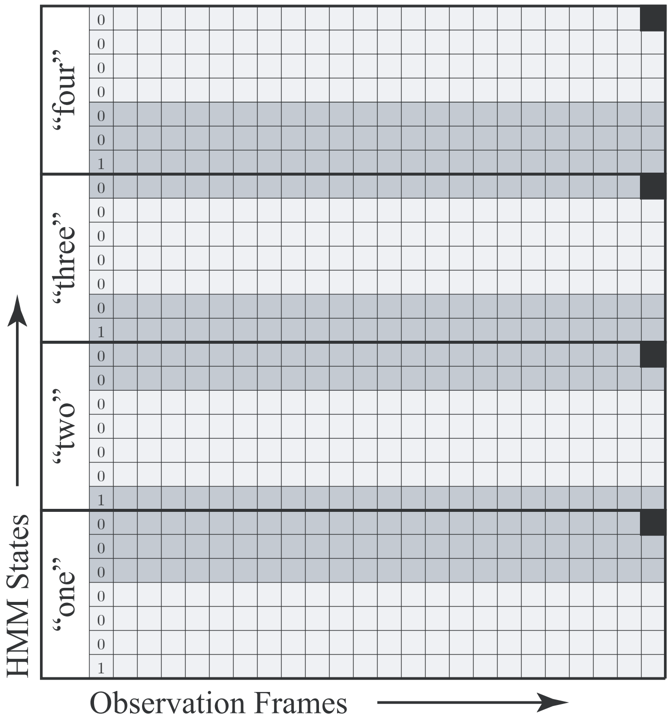

Figure 3
SIMD vectorization of the Viterbi algorithm.