

The Analysis and Design of Architecture Systems for Speech Recognition on Modern Handheld-Computing Devices

Andreas Hagen

Center for Spoken Language
Research, University of Colorado at
Boulder

001 303 735-5227

andreash@colorado.edu

Daniel A. Connors

Department of Electrical and
Computer Engineering, University of
Colorado at Boulder

001 303 735-7199

dconnors@colorado.edu

Bryan L. Pellom

Center for Spoken Language
Research, University of Colorado at
Boulder

001 303 735-5382

bryan.pellom@colorado.edu

ABSTRACT

Growing demand for high performance in embedded systems is creating new opportunities to use speech recognition systems traditionally executed only on high performance systems. In several ways, the needs of embedded computing differ from those of more traditional general-purpose systems. Embedded systems have more stringent constraints on cost and power consumption that lead to design bottlenecks for many computationally-intensive applications. This paper characterizes the speech recognition process on hand-held mobile devices and evaluates the use of modern architecture features and compiler techniques for performing real-time speech recognition. We evaluate the University of Colorado Sonic speech recognition software on the IMPACT architectural simulator and compiler framework. Experimental results show that by using a strategic set of compiler optimization, a 500MHz processor with moderate levels of instruction-level parallelism and cache resources can meet the real-time computing and power constraints of an advanced speech recognition application.

Categories and Subject Descriptors

C.3 [Special-Purpose and Application-Based Systems]: – *Real-time and embedded systems.*

General Terms

Measurement, Performance, Design

Keywords

Speech Recognition, Embedded Systems, Performance

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

CODES+ISSS'03, October 1–3, 2003, Newport Beach, California, USA.

Copyright 2003 ACM 1-58113-742-7/03/0010...\$5.00.

This research was partially sponsored by Avaya Inc. via research award W0711119CWS and by NSF under award NSF/IERI 1R01HD44276-01.

1. INTRODUCTION

In the recent years PDAs have become more and more powerful and affordable at the same time. It has become possible to run powerful applications such as multimedia on handheld devices. Nevertheless, PDAs have a unique interface system that differs from other general-purpose systems. Typing on PDAs is more error prone than on normal sized keyboards because of their size, also PDAs are used in situations where typing might be hard because of influences like shaking while traveling. Therefore it is very desirable to look for alternatives to typing as a user input modality on PDAs. A very natural way of controlling and giving commands to a PDA is speech since human beings are used to this kind of interaction with each other. In order to assure a comfortable speech command and control behavior a few conditions have to be given. The recognition error rate should be very small and the processing of the speech input should be possible in real time such that no delays are conceivable for the user.

The first requirement of low error rates is a task that is sufficiently solved for command and control tasks on state of the art speech recognition systems for desktop computers. By command and control task we mean a recognition task consisting between a 500 and 2000 word vocabulary and input of keywords, phrases, or small finite-state recognition grammars. The error rates for such a task are low enough for enabling a comfortable use (especially if the system is adapted to the user).

The second requirement is the necessity for real-time processing of the input speech. This requires processing of the audio as it is recorded by the PDA with recognized text strings available almost immediately after the user finishes speaking. Real-time factors less than 1.0 are desirable while factors greater than 1.0 imply that the software cannot keep up with processing the input speech. A real-time factor of 2.2, for example, would lead to processing of 5 seconds of audio during a span of 11 seconds. This would result in an unacceptable delay of 6 seconds between the time that the user finishes speaking until the time that the text string was available to the application.

In this paper we focus on system design issues for the second requirement, the real-time processing of speech on PDAs. We simulated 500-2000 word vocabulary speech recognition tasks in which each word is equally probable. Such vocabulary sizes seem

to be a sufficiently large vocabulary for a set of commands that could be used to control a PDA. The recognition system works in a speaker independent manner, this means the acoustic model is trained ahead of the actual recognition and the user does not need to train the system for his or her voice.

In order to be useful the word error rate for such a vocabulary size should be no higher than 10% (this means about 1 in 10 words needs to be repeated). This accuracy is achievable with state of the art speech recognizers. A recognition task consisting of isolated words spoken by kids is reported in [1], where the word error rate is less than 1% for a 200-word vocabulary for 10th grade students. This task is very similar to the command and control setup mentioned earlier. Of course a noisy environment would influence the results in a negative way.

The question to be answered in this paper is if PDA architectures are fast enough to recognize the spoken utterances in real-time and what code optimizations and hardware changes could be applied to improve the performance further.

2. RELATED WORK

Speech Recognition is a very active area of research. In addition, two main problems, accuracy and running time have been analyzed in speech recognizers executing on modern general purpose hardware. Agaram et al. [2,11] have analyzed the amount of instruction level parallelism (ILP) in the SPHINX speech recognizer of CMU [3], the influence of cache sizes and other hardware specifications on the instruction per cycle (IPC) rate was simulated. They also give an estimate of the instruction throughput needed to process speech in real-time on the Alpha ISA. In [4] Vlaovic and Uhlig report cache and translation look-aside buffer (TLB) hit ratios. The cache-hit ratios are evaluated for different cache sizes. As expected the number of cache misses decreases with increasing cache size. The same behavior can be seen for different TLB sizes. The authors used the Dragon Systems NaturallySpeaking speech recognition system on an X86 ISA simulator.

In our work we evaluate the performance of the University of Colorado Sonic speech recognizer on a PDA-like architecture. Using the IMPACT architectural simulator and compiler framework [5], we provide estimates on the needed clock rate of several processors which would be required in order to process the command-and-control vocabulary in real-time. We also analyze Sonic's performance dependent on different data and instruction cache sizes. At an even higher level, the Wattch [14] power analysis tool provides accurate power estimations based on the architectural description of the simulated microprocessor.

3. THE SONIC SPEECH RECOGNIZER

3.1 Overview

The Sonic speech recognition system was developed at the University of Colorado [6,7]. The recognizer is based on decision-tree state-clustered triphone hidden Markov (CDHMM) models. In such an architecture, the basic building blocks of sounds, also known as phonemes, are characterized by 3 Markov states. The three states are used to model the statistical distribution at the beginning, middle, and end of the sound unit. Each state is modeled by a mixture of multivariate Gaussian

densities. Transitions between states are modeled using a gamma probability density function.

During recognition, features are extracted every 10 ms (100 feature vectors per second) from the audio signal. The features are represented as a floating-point vector containing 12 Mel Frequency Cepstral Coefficients (MFCCs), energy, and the first and second differences of these parameters. The features encode the short-time spectrum of the audio signal. The resulting feature vector stream has a dimension of 39. Feature computation is extremely efficient consuming less than 1% of the CPU in a typical speech recognizer.

The search network in Sonic is represented using a reentrant static tree-lexicon [8]. Here, phonetic word pronunciations are expanded into sequences of context-dependent HMM states and inserted into a tree-based search structure for efficiency. In this framework, words that share similar initial phoneme sequences also share the same root node in the lexical search tree.

The recognizer implements a two-pass search strategy. The first pass consists of a frame-synchronous, beam-pruned Viterbi token-passing search [9]. Tokens are used to maintain the state of the recognizer. They encode paths (consisting of unique word sequences) that the recognizer is considering given the current feature vector of input. Tokens propagate through the tree-lexicon while consuming input frames until a leaf-node in the tree is reached. At that point, the identity of a word is known. The word-exiting token can then propagate into the root nodes of the tree allowing for continuous speech recognition.

At the transition between words, a language model probability is applied. Most state-of-the-art speech recognition systems constrain the sequence of allowable words using a fixed grammar or by using a statistical n-gram language model. N-gram language models condition the probability of transitioning to a word based on the previous N-1 words. Trigram (N=3) and bigram (N=2) language models are typical for most continuous speech recognition tasks.

The first recognition pass creates a lattice of word ends. During the second pass, the resulting word-lattice is converted into a word-graph. The word-graph is searched for the most-likely word string.

CPU requirements for the second-pass search are often negligible compared to the first-pass search.

3.2 Efficiency Issues

Speech recognition is an extremely intense task from a computational standpoint. Search within the Sonic recognizer is dominated primarily by two operations: token passing and Gaussian evaluation. Token passing refers to the maintenance of the paths (word sequences) and their relative probabilities. Gaussian evaluation is required to compute the probability of observing an input feature vector within an HMM state during search through the lexical tree.

For small to medium vocabulary tasks such as command-and-control, calculation of the Gaussian likelihoods consumes much of the CPU resources of the speech recognizer. Larger input vocabularies require more path history maintenance resulting in large CPU efforts for both token passing and Gaussian

evaluation. Because our task is command and control, let's further consider the issue of Gaussian computation.

Each HMM state within Sonic is represented by a mixture of M Gaussian densities. Typical systems have between 6 and 32 Gaussian densities per HMM state. Each Gaussian density is represented by a weight (w_m), mean vector (μ_m), and covariance matrix (Σ_m). For efficiency and issues related to data sparseness, the covariance matrix is assumed to be diagonal resulting in the following likelihood calculation,

$$b(\vec{x}_t) = \sum_{m=1}^M \frac{w_m}{(2\pi)^{D/2} \sqrt{\prod_{d=1}^D \sigma_m^2[d]}} \exp\left\{-\frac{1}{2} \sum_{d=1}^D \frac{(x_t[d] - \mu_m[d])^2}{\sigma_m^2[d]}\right\}$$

where $b(\vec{x}_t)$ represents the probability of observing the t th frame of input (x_t). Note that the feature dimension D is 39 for most traditional speech recognition systems. We can see that evaluation of a single HMM state consisting of 6 Gaussians would require 480 floating point multiplies, 234 subtractions and 6 $\exp(\cdot)$ operations.

The likelihood function can be computed with almost no loss using a nearest-neighbor approximation and by performing the calculations for search in the log-domain. The resulting equation for the log-likelihood becomes,

$$\log b(\vec{x}_t) \approx \arg \max_m \left\{ C_m - \frac{1}{2} \sum_{d=1}^D \frac{(x_t[d] - \mu_m[d])^2}{\sigma_m^2[d]} \right\}$$

where C_m is a constant that can be pre-computed prior to runtime. This implementation requires less floating-point operations in practice since the above equation can be partially computed. That is, while searching for the value of m that maximizes the equation, we can drop out of the summation loop over the D feature elements as soon as the partial sum falls below that of the best scoring mixture component. This approximation when combined with several straight-forward optimizations can improve the speed of the Gaussian computation within a speech recognizer by almost 30% with no loss in performance for most practical implementations [10].

In summary, command-and-control speech recognition tasks require many floating-point operations. This is directly related to the fact that Gaussian densities are evaluated during the recognition search process. In perspective, the recognizer receives 100 feature vectors input for one second of audio. For each frame, it is not uncommon to evaluate several hundred HMM states (each consisting of multivariate mixture Gaussians). Some speech recognition architectures, such as the older CMU Sphinx-II system, deal with this issue by using integer arithmetic and semi-continuous Gaussian densities. These improvements often come at a loss in system accuracy [8].

4. THE IMPACT COMPILER

An important requirement for evaluating Sonic's performance on different processor systems is to either have each system available in hardware or to be able to simulate Sonic's execution on a hardware simulator. A very powerful tool for doing this is the IMPACT simulator. IMPACT processes Sonic's source code and traverses various stages of optimization.

All compiler optimizations are done based on profile information. First, function inlining, which represents a compiler technique for replacing a function call with the respective function body. Next, compiler technology generates analysis of interprocedural memory dependence information maintained throughout the compilation process. After analysis, frequently executed code traces are identified using execution profiles. These traces, called Superblocks[15] are similar to extended basic blocks, except code duplication creates a single entry multiple exit region with more potential for ILP optimization. Finally, IMPACT is able to generate scheduled code, thereby adapting low-level code to a certain degree to a specific target architecture.

Furthermore IMPACT is able to simulate optimized code execution on different target architectures, these can be specified in sufficient detail in order to well approximate various architectural specifications. Some of the adjustable parameters (only a small subset) are the processor's issuing type, in our case superscalar, the issue width, the number of different functional units, cache sizes, ports, and associativities. IMPACT and detailed information about its functionality can be found in [5].

5. INTEL'S XSCALE ARCHITECTURE

Intel's XScale architecture is a special processor architecture for high performance embedded systems and is based on the ARM architecture specifications. The PXA250 processor is a highly integrated system with an XScale core. It is a 32-bit RISC microarchitecture especially designed for good performance and integration as well as low power consumption. It serves as the reference for the family of embedded systems in this work.

The PXA250 has clock frequencies from 200 to 400MHz and a 2KB mini data cache as well as an equally sized mini instruction cache. The standard caches that could be compared to L2 caches (on desktop machines) are 32KB, again one such sized for data and one for instruction caching. The processor's core represents a superscalar machine with 4 functional units. The functional units are: a memory unit, a shift and permute unit, a multiply and permute unit, and a general execution unit.

More details on the functional units are given in [11]. The branch target buffer (BTB) of the PXA250 holds 128 entries and the branch predictor is 2 bit predictor. More information on the PXA250 can be found in [12] and [13].

6. EXPERIMENTAL SETUP

In order to simulate a command and control task on handheld devices two main steps have to be taken. The first one is the setup of an appropriate speech recognition task of the necessary complexity. The other one is the setup of the hardware simulator for the target hardware and some slightly modified hardware setups.

6.1 The Recognition Task

Children's speech represents an interesting field of research because of applications in the fields of education, learning devices, and toys.

For the command and control recognition task we use the Sonic speech recognizer described in detail earlier. The vocabulary for the task consists of 500 words. The speech recognition acoustic

model is trained from 1781 students from grades 1 through 10. Data from 1118 children were obtained from the OGI Kids Speech Corpus [1] while data from the remaining 663 children were collected from the Boulder Valley School district in Boulder Colorado. Data was collected from each student using a head-mounted microphone and a desktop PC. Each subject contributed 20 training word examples on average.

The models do not have to be adapted to the current user, but of course an additional adaptation step would further improve the recognition rate that is already certainly higher than 90% [1]. 10% or less word error rate seems to be sufficient for a comfortable use on a PDA. In order to evaluate the performance of this command and control task the input speech file is a 16KHz sampled instance of the spoken word 'QUIT'.

The length of the sound sample is 2.2 seconds. On a 1.7GHz Pentium 4 desktop machine the real-time factor for its processing is 0.57. In average the real-time factor for 29 different words on this machine is about 0.46 and the standard deviation results in 0.06. So the speech sample used for this task has a real-time factor slightly above average but still in the range that can be seen as representative for the set of input sound samples. This means if a system is able to process the test sound sample in real-time or even faster, it will be able to process a majority of all possible words (commands) in real-time.

6.2 Target Hardware Specifications

The simulation setup for the command and control task has to be very close to the specifications of a state of the art handheld device to give useful results and predictions. For this work we consider the following 4-wide machine:

- 2 integer units
- 2 floating point units
- 2 memory units
- 1 branch unit

Only four units out of the given seven can be active at any time. This is very similar to the PXA250 processor, which has a general execution unit and therefore about the same amount of scheduling-freedom.

Further specifications are:

- 2KB data cache
- 2KB instruction cache
- 64KB combined data and instruction cache
- 128 entry BTB w/ 2 bit branch predictor

The IMPACT simulator is adjusted to these parameters and therefore the simulation of the recognition task results in similar cycle counts as the execution of this task would require on a handheld device. This will allow us to analyze the clock frequency rates required to run the described command and control task in real-time on state of the art as well as future or extended PDAs.

7. PERFORMANCE EVALUATIONS

We compiled Sonic's code under three levels of optimization for three different architectural setups, a 2-wide issue machine, a limited 4-wide machine (described earlier, similar to the XScale

architecture) and an unlimited 4-wide machine (4 floating point units, 4 integer units, 4 memory units and 1 branch unit). The three different kinds of optimized code were architecture specific scheduled O-code, which is similar to what GNU's gcc compiler would generate, superblock optimized code (S-code) which is also scheduled for the target architecture. As mentioned all codes were scheduled for each target architecture. Profiling information was also used for improved optimization, and the use of inlining/interprocedural analysis improved the base code performance by 10%.

One of the goals in this paper is to determine the lower bounds in the clock frequencies of modern handheld devices under the constraint of real-time speech recognition for the introduced command and control task. In order to do this analysis the initialization phase of Sonic needs to be subtracted from the overall cycle count of Sonic's execution cycle count, since the speech recognizer would be started once at the beginning and would not be shut down in between the recognition of multiple commands. Our simulations showed that the percentage used for Sonic's initialization compared to the overall command recognition (initialization plus recognition of one word) only takes 1.3%. Even though this means only a small impact we still took it into consideration. We found that S-code had the best performance and that the application of hyperblock formation (use of predicated execution support in the architecture) did not help but rather worsened the results.

7.1 Clock Rate Estimation

To compute the clock frequencies the cycle counts from the IMPACT simulator are normalized such that the fraction counting for the initialization part is ignored. Since the length of the input speech file is 2.2 seconds the computation of the clock frequency is simply the normalized cycle count divided by the duration of the speech input (2.2s).

7.2 Simulation Results

For the 4-wide restricted machine (comparable to the PXA250) the O-code requires a clock frequency of at least 452 MHz for the task to run in real time. The S-code improves the required clock frequency by 33 MHz to 419 MHz. This behavior can be observed for all test architectures and leads to the conclusion that superblock- optimization is beneficial. Interprocedural memory alias analysis can be safely left aside for the Sonic speech recognizer that shares many techniques with the Sphinx speech recognizer from CMU.

The estimated clock frequencies show that the command and control task for a 500-word vocabulary is almost executable on a current handheld device in real-time. The PXA250 is delivered with clock frequencies from 200 to 400 MHz. The gap of about 20 MHz is very likely to be closed in the near future. It might also be closed by an additional speaker adaptation step that would speed up the execution.

In the next section we will also show that an architectural change in cache sizes is able to close this gap already.

7.3 Cache Evaluation

We simulated different data and instruction cache sizes in order to evaluate how these changes reflect in Sonic's performance and to what extent they are beneficial. Figure 1 shows the number of

cycles relative to the data cache size on the unrestricted 4-wide machine.

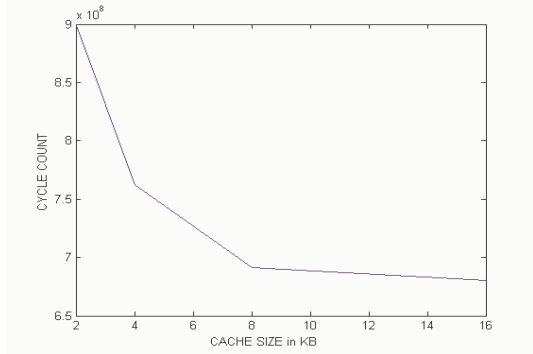


Figure 1: Cycle counts relative to data cache size

The simulation results show that a 16KB mini cache would yield in performance benefits of over 32%, an 8KB cache improves the performance by about 30%.

A 30% performance benefit due to an increased mini cache matches about a 125MHz increase in clock frequency, given the minimum required clock frequency proposed above and the fact that the whole cache and memory unit would be clocked faster by the same ratio of 30%. Therefore the consequences would be that the command and control task of 500 words would already run on a PXA250 with increased mini cache (8KB). Increasing the instruction mini cache did not result in a remarkable performance benefit that would justify the costs for expensive cache modules.

Although 500 words seem sufficient to control a PDA at this time, it might be possible that speech recognition draws increased interest in the near future, especially if the systems evolve such that more complicated commands, even small sentences are possible as control input. To give a rough estimate of the performance increase required for larger tasks, we simulated a 1000-word and a 1500-word command and control task (Table 2).

# of words	# of cycles	cycle increase
500	926278753	1.00
1000	1195492858	1.29
1500	1481844128	1.60
2000	1672314581	1.81

Table 2: Cycle counts and increases (based on the 500-word task) for variously sized command and control tasks

It turns out that the 1000-word task requires about 1.3 times the performance of the 500-word task, and the 1500-word task requires about 60% more performance compared to the 500-word vocabulary task and finally the 2000-word task needs 80% more resources than the 500-word task to run in real-time. The curve of cycle increases is expected to become more and more flat.

These tasks are expected to be executable in real-time on a PDA in the next one or two years, given the performance increase observed in the embedded systems field in the recent years.

8. POWER SIMULATION

As the issues and concerns facing power consumption grow to include high performance microprocessors as well as the pervasive hand-held and wireless computing market, novel approaches to power conservation at the architectural level are being sought. At the same time, ubiquitous computing is becoming more prevalent, and thus the performance demands made upon these devices are increasing, leading toward more complex architectures.

Although much work on energy reduction has taken place in the circuit and device technology domains [16], there has been an increasing emphasis on designing for power efficiency at the architectural level. This implies that the desired energy and power goals must be targeted early in the design cycle and that the system microarchitecture must work in concert with advances in circuit technology to reduce power demands.

To evaluate power consumption of speech recognition, we integrated the Watch power analysis tool [14] with the IMPACT architecture simulator. Figure 2 illustrates the power consumption for the architecture studied through the duration of the sample input. During initialization, the memory system has the peak power of nearly 800mWatts, but averages 380mW, only going slightly above 400mW for the ending duration of the Sonic application.

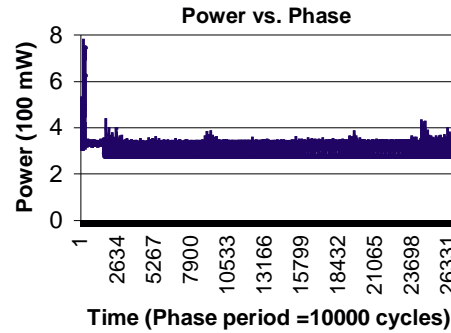


Figure 2: Power consumption timeline for Sonic

With the studied architecture, the breakdown of architecture elements and power consumption were:

- Clock (43.3%)
- Floating-point ALUs (26.7%)
- Register file/result bus (10.2%)
- Integer ALU (7.5%)
- Mini Dcache (5.5%)
- Mini Icache (3.6%)
- Level2 Cache (3%)

By using low-power techniques at clock distribution and clock-gating techniques for the floating-point unit, it may be possible to

design a low-power system specifically designed for speech recognition. Generally all speech recognition applications require intense floating-point computations, so it is not straight forward to reduce the power spent in this unit. We are currently looking at techniques to evaluate the number of speech sentences that can be recognized and the impact on power consumption and battery lifetime.

9. SUMMARY AND FUTURE WORK

In this paper the main goal was to explore real-time speech recognition for command and control tasks on handheld devices. It was shown that real-time speech recognition is very close to being possible for optimized code. The missing 20MHz in clock frequency estimated for the PXA250 running Sonic could be filled by a larger cache size of 8KB. For non-superscalar optimized code about 50 MHz are missing for the introduced command and control task to run in real-time, this means superblock optimization is very beneficial in this application.

We gave a prediction of the required performance increases that are necessary for larger command and control tasks to still run in real-time. It could be shown that the 500-word task is at the edge of real-time executable recognition tasks on architectures similar to the PXA250.

The power simulations give an overview of the processors' by unit power consumptions. It can be seen that the dominating power consumption comes from the floating-point computations that are executed during the Gaussian mixture densities computation.

The field of speech recognition on handheld devices is an active field of research far from being sufficiently explored. Some ideas for future work that were left aside in this work are the following. We were using a language model with equal likelihood for all words, this is certainly not true for command and control tasks. Different probabilities for different commands are likely to improve performance. Another performance improving step that could be applied is speaker adaptation that also improves the recognition rate further.

Another interesting direction to further explore in the field of speech recognition systems for PDAs is a fixed-point implementation. Sonic up to this point is based on floating-point operations and all other state of the art speech recognition systems are, too. Developing a fixed-point speech recognizer is definitely an interesting project for the future.

The data used for this task are from children's speech corpus. The speech of children has a higher variability than adult speech and might therefore influence the performance in a slightly negative way. It would be interesting to use data directly recorded on PDAs. For this data the quality would be lower and as a consequence the performance is expected to drop (because more extensive search has to be done). Unfortunately there is no PDA-recorded data available as a standardized corpus, which would be of great benefit to the research community.

10. REFERENCES

[1] Khaldoun Shobaki, John-Paul Hosom, and Ronald Cole. The OGI Kids' Speech Corpus and Recognizers. In Proceedings

of the International Conference on Spoken Language Processing (ICSLP), Beijing, China, October 2000.

- [2] K. Agaram, S. Keckler, and D. Burger. A Characterization of Speech Recognition on Modern Computer Systems. In 4th IEEE Workshop on Workload Characterization, December 2001.
- [3] Kai-Fu Lee, Hsiao-Wuen Hon, and Raj Reddy. An Overview of the SPHINX Speech Recognition System. IEEE Transactions on Acoustics, Speech and Signal Processing, 38:35--44, 1990.
- [4] S. Vlaovic, R. Uhlig. Performance of Natural I/O Applications. 2nd Workshop on Workload Characterization. October 1999.
- [5] <http://www.crhc.uiuc.edu/Impact> (link checked on 04/14/03)
- [6] Pellom, B., Hacıoglu, K., Recent Improvements in the Sonic ASR System for Noisy Speech: the SPINE task. Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP), Hong Kong, April 2003.
- [7] Pellom, B., Sonic: The University of Colorado Continuous Speech Recognition System. Technical Report TR-CSLR-2001-01, University of Colorado, March 2001.
- [8] M. Ravishankar, Efficient Algorithms for Speech Recognition. PhD Thesis, Carnegie Mellon University, 1996.
- [9] S.J. Young, N.H. Russell, J.H.S. Thornton, Token Passing: A Simple Conceptual Model for Connected Speech Recognition Systems. Cambridge University Engineering Department Technical Report CUED/F-INFENG/TR.38, 1989.
- [10] Bryan Pellom, Ruhi Sarikaya, John Hansen, Fast Likelihood Computation in Nearest-Neighbor Based Search for Continuous Speech Recognition. In IEEE Signal Processing Letters, vol. 8, no. 8, pp. 221-224, August 2001.
- [11] Nigel Paver. Intel XScale Micro-Architecture with Wireless MMX technology. www.intel.com/pca/developernetwork, Volume 4, Summer 2002.
- [12] The Intel PXA250 Applications Processor. White Paper, Intel, February 2002.
- [13] PXA250 and PXA210 Application Processors Developer's Manual, Intel, 2002.
- [14] David Brooks, Vivek Tiwari, and Margaret Martonosi, *Watch: A Framework for Architectural-Level Power Analysis and Optimizations*. 27th International Symposium on Computer Architecture (ISCA), Vancouver, British Columbia, June 2000.
- [15] Hwu et al. The Superblock: An Effective Structure for {VLIW} and Superscalar Compilation. Center for Reliable and High-Performance Computing. University of Illinois, Urbana, IL. February, 1992.
- [16] A. P. Chandrakasan, S. Sheng, and R. W. Brodersen. *Low-power CMOS digital design*. IEEE Journal of Solid-State Circuits, vol. 27, no. 4, pp. 473-484, April 1992