MOBILE CPU BASED OPTIMIZATION OF FAST LIKELIHOOD COMPUTATION FOR CONTINUOUS SPEECH RECOGNITION

Kisun You, Youngjoon Lee and Wonyong Sung, Member, IEEE

School of Electrical Engineering
Seoul National University
San 56-1, Shillim-dong, Kwanak-gu, Seoul 151-744 Korea
{ksyou,yjlee}@dsp.snu.ac.kr, wysung@snu.ac.kr

ABSTRACT

We studied the real-time implementation of continuous speech recognition algorithm on a mobile CPU, Intel PXA270, platform. Especially, the optimization of fast likelihood computation, which takes the largest part in most speech recognizers, is conducted by employing SIMD (Single Instruction Multiple Data) programming and software pipelining. The overhead of exhaustive memory accesses is also minimized by placing frequently used acoustic model data at the fast internal SRAM. The number of execution cycles for the fast likelihood computation of the 1000-word vocabulary Resource Management (RM) task has been reduced by 56.42%. The resulting performance shows approximately four times faster processing speed than the real-time implementation requirement on a 520MHz Intel XScale-based system.

Index Terms— Speech recognition, Likelihood computation, Mobile platform, Scratchpad memory, Architectural optimization.

1. INTRODUCTION

Real-time implementation of a medium to large vocabulary speech recognizer on portable systems, such as PDAs or cellular phones, is much desired, but is still a challenging task because of the limited computing and memory resources of the systems. Recently, several researches on implementing continuous speech recognizer on mobile platforms have been presented. For example, the PocketSphinx project [1] showed various algorithmic level optimization results, such as fast forward Viterbi search only, beam tuning, and kd-trees, but the project only considered a relatively simple CPU architecture, Intel StrongARM, and there were not enough architecture level optimizations that are crucial for advanced mobile platforms. A multicore processor based implementation was discussed in [2], however the implementation presented is not scalable.

Recently introduced mobile CPU processors, such as Intel XScale or ARM11, incorporate quite powerful architectural enhancements such as SIMD arithmetic support and deep pipelining for higher clock frequency. For example, it is possible to process four 16-bit data at a time using one SIMD instruction. Furthermore, with the advance of system-on-a-chip (SoC) technology, memory architectures of mobile processor systems have heterogeneous configurations for the purpose of relieving insufficient memory bandwidth problem [3]. Therefore, it is very critical to utilize these features to achieve the needed performance.

Based on the available algorithmic optimization, we exploited the Intel Wireless MMX technology (IWMMX) based programming and software pipelining techniques to efficiently utilize both the data level and the instruction level parallelism. There is previous research on SIMD optimization of fast likelihood computation [4]. Our study, however, also utilize the software pipelining to increase the efficiency of a deeply pipelined architecture. Moreover, data partition algorithm which distributes the model data to the fast on-chip SRAM and the slow external SDRAM to reduce the memory access overhead is proposed. The optimization techniques demonstrated in this paper do not degrade the recognition performance.

This paper is organized as follows. Section 2 describes the baseline recognition system implemented on the target platform. The optimized scheduling of likelihood computation exploiting IWMMX is shown in Section 3. Section 4 illustrates the proposed algorithm for data partitioning. Experimental results are shown in Section 5, and finally concluding remarks are made in Section 6.

2. BASELINE SYSTEM PERFORMANCE

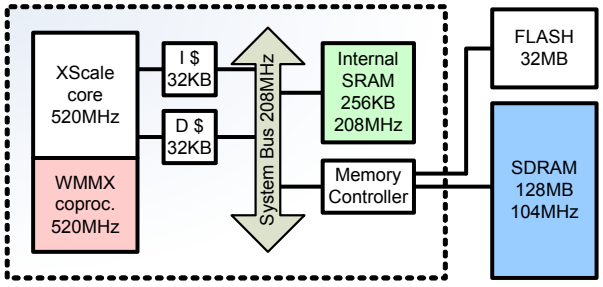

Fig. 1. Target architecture diagram

We implemented a continuous speech recognition (CSR) system on the Intel PXA27x Processor Developer's Kit. The platform is equipped with the Intel PXA270 processor running at 520MHz, 128MB mobile SDRAM at 104MHz, 32MB NOR FLASH memory for code and non-volatile data storage, and various peripherals for developing mobile prototypes. 32KB instruction and data caches and 256KB internal SRAM serve the CPU core for faster memory accesses. Figure 1 shows the overall structural diagram of the target system. Since the platform does not have a floating point unit (FPU), the implementation is based on fixed point arithmetic. The operating system employed in this research is Linux kernel version 2.6.9 provided by Intel.

We assessed the recognition performance by the evaluation set of the DARPA Resource Management (RM) corpus. The acoustic

Table 1. Baseline system performance

Function	Cycles (M)	Instr. (M)	CPI
Feature Extraction	49.90	41.65	1.20
Emission Prob.	186.02	61.96	3.00
Dynamic Programming	50.59	9.30	5.44
Pruning and Path Updates	102.45	18.18	5.64
LM Pruning	32.88	3.49	9.42
Node Updates	118.67	37.32	3.18

model is trained by 2,880 speaker independent utterances from the training set of the RM corpus. All of the training processes were performed by the HTK which is an open-source speech recognition toolkit [5]. The trained acoustic model uses context-dependent hidden Markov model (HMM). The total number of shared states by tree-clustering is 1,152 and each state has a 4-Gaussian mixture that is represented by 40x2 16-bit values, thus the total model data size is 720KB. The system has been evaluated by 300 speaker independent test utterances. We deployed the 993 word vocabulary bigram language model with language weight of 8.0. The measured word error rate (WER) of the system is 9.95%.

The decoding process consists of three major parts: feature extraction, emission probability computation, and search using the emission probability and the language model. First of all, a 30ms frame of input speech is converted to 39 dimensional MFCC (Mel-Frequency Cepstrum Coefficient) feature vector for every 10 ms. The feature incorporates with the trained acoustic model parameters to generate the emission probability of each HMM states, and the Dynamic Programming (DP) recursion expressed in Eq. 1 is performed over the Viterbi search network. Beam pruning is applied to avoid exhaustive search.

$$\psi_t(s_j) = \max_i\{\psi_{t-1}(s_i) + log(a_{ij})\} + log(b(O_t; s_j)) \quad (1)$$

a_{ij} indicates the transition probability from s_i to s_j and the $b(O_t; s_j)$ represents the emission probability. For the word level transcription, we have to keep the best predecessor word for each word hypothesis.

$$H_t(w) = \arg\max_v\{log(p(w|v)) + \psi_t(S_v)\} \quad (2)$$

$p(w|v)$ is the bigram probability from word v to word w and S_v indicates the last state of the word v.

The target processor equips a performance monitoring unit (PMU) to measure the system efficiency [6]. We can get the numbers of instruction/data cache efficiency, data/bus stall latency, branch misprediction, etc. We measured the performance of the baseline recognition system with an input sample of 3.21 second length, and the results are shown in the Table 1. Cycles (M) stands for machine cycles including pipeline stalls in millions, Instr. (M) represents dynamic instruction counts in millions, and CPI is the number of cycles per instruction.

Table 1 shows that the emission probability function which computes $b(O_t; s_j)$ consumes dominant execution time in the overall search procedure. In this table, dynamic programming executes the DP recursion in Eq. 1. Other functions are related to processing the search network characterized by the language model. The system took around 540M cycles to finish the recognition process, which is faster than real time execution speed when the processor is running at 520MHz. Exploration for performance optimization, however, is still important, because the size of work set will expand rapidly when the vocabulary size increases.

3. SIMD PROGRAMMING AND SOFTWARE PIPELINING

The target processor has a dedicated IWMMX unit for single instruction multiple data (SIMD) computation [7]. It serves as a co-processor to the XScale main pipeline. It enables 64-bit wide SIMD computations, which can deal with 2 word (32bit) data, 4 half-word (16-bit) data, or 8 byte (8bit) data at once.

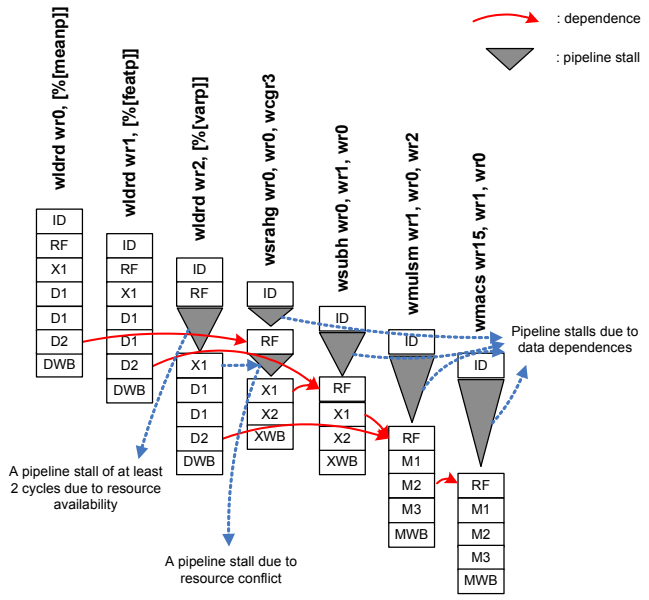

Fig. 2. Pipeline execution of the unoptimized code

The most time consuming part in the recognition system is the computation of the emission probability. It shows rather regular structure, compared to the other functions. The emission probability computation for the HMM state s is as follows:

$$\log(b_m(O_t; s)) = \max_{1\le m\le M}\{C_m - \frac{1}{2}\sum_{k=1}^{K}\frac{(x_k - \mu_{mk})^2}{\sigma_{mk}^2}\} \quad (3)$$

M indicates the number of mixtures per HMM state, and K is the feature dimension.

Since the emission probability of a state is approximated by the nearest-neighbor probability, the partial distortion elimination (PDE) technique has been adopted to take advantage of early termination [8]. The computation terminates if the partial results of Eq. 3 becomes lower than the best mixture probability at the moment. The feature component reordering (FCR) and the best mixture prediction (BMP) in [8] are also applied to leverage the chance of early termination.

In the innermost loop which computes Gaussian mixture probability, the computations are independent to each other. We can simply start the SIMD optimization by computing four components simultaneously. The pipelined operation of core computation per iteration is shown in Fig. 2.

Pipeline execution stalls due to several reasons, such as resource availability, data dependency, etc [7]. In this code, the most critical reason of the stall is memory load latency. Besides, there are pipeline stalls incurred by data dependencies. On the whole, the performance suffers from high pipeline stall counts.

To hide the long load-to-use stalls, we may use the software pipelining technique [7]. Additionally, the prefetch instruction 'pld'

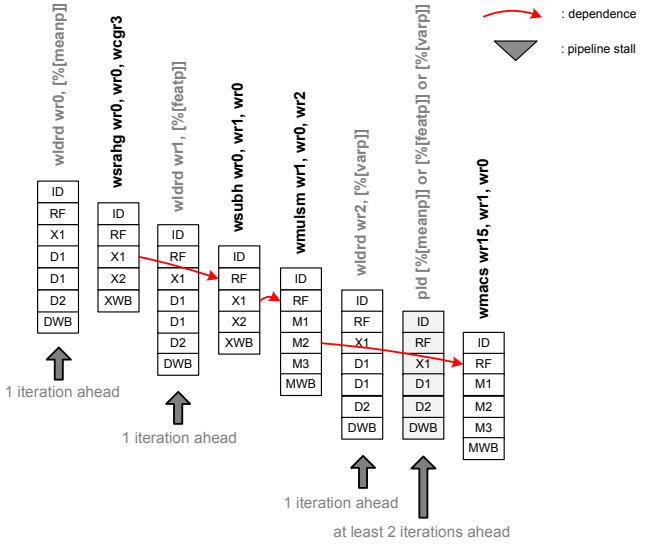

Fig. 3. Pipeline execution after software pipelining

in XScale architecture helps hide the load latencies. The ratio of data load to computation per iteration is 1:1, thus assuming that the load latency of 'wldrd' is less than a loop iteration time, we can schedule the software pipelined instructions in just 2 iterations. After the scheduling, the pipeline operates as shown in Fig. 3.

4. ACOUSTIC MODEL DATA PARTITIONING FOR EFFICIENT ON-CHIP MEMORY UTILIZATION

The Intel PXA270 Processor has a 256KB internal SRAM. Although the SRAM is located below the cache memory in the memory hierarchy as the SDRAM is, it is faster and consumes less power because it is an on-chip memory. Thus, we can reduce the execution time of the emission probability by placing some of the acoustic model data on the SRAM. This approach has been researched formerly as the scratch pad memory problem [3]. In this paper, we propose the partition algorithm that is suitable for emission probability computation.

Since the emission probability computation adopted the PDE optimization described in Section 3, not every data component of a Gaussian mixture model (GMM) is accessed. Thus, we performed a block-based SRAM assignment.

Assuming that $b(s,m)$ SRAM blocks are assigned to the SRAM for a GMM m of the state s, we can define the access time of one GMM as follows:

$$L(t,s,m) = b(s,m) \cdot L_S + (ET(t,s,m) - b(s,m)) \cdot L_{SD} \quad (4)$$

$b(s,m)$: number of SRAM blocks assigned to the mixture m of the state s
$L_{\{S,SD\}}$: access time of SRAM and SDRAM respectively
$ET(t,s,m)$: early termination point of PDE

In this case, the optimal partition can be achieved by finding the $b(s,m)$ for every mixture m and state s which minimize the following equation:

$$\sum_{t=1}^{T}\sum_{s=1}^{S}\sum_{m=1}^{M} L(t,s,m) \cdot I(t,s) \quad (5)$$

$I(t,s)$: indicator for the activeness of the state s

Assigning different number of SRAM blocks to each mixture m of the state s is prohibitive due to the induced complexity in implementation. Thus, we assumed that every mixture in the state s has the same number of SRAM blocks. We also approximated the $ET(t,s,m)$ to the expected value $\mathbf{ET}(s)$.

After the approximations, we can rewrite the total access time.

$$\sum_{s=1}^{S}\{b(s) \cdot L_S + (\mathbf{ET}(s) - b(s)) \cdot L_{SD}\} \times f(s) \quad (6)$$

$f(s)$: usage frequency of the state s ($\sum_{t=1}^{T} I(s)$)
$\mathbf{ET}(s)$: expected PDE point of the state s
Condition 1: $b(s) \leq \mathbf{ET}(s)$ for every state s
Condition 2: $\sum_{s=1}^{N_s}(s_{SIZE} \times b(s)) \leq SRAM_{SIZE}$

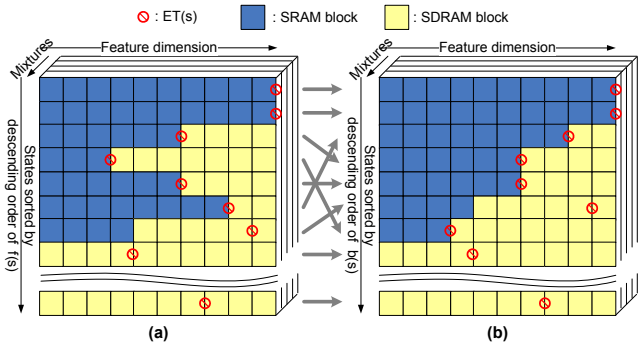

Fig. 4. Proposed model data partition method

Figure 4 shows the heuristic approach to obtain a data partition result which minimizes Eq. 6. First, all the states are sorted by descending order of $f(s)$. From the top, we assign the $b(s)$ blocks to the every mixture m of the state s until the $b(s)$ reaches the $\mathbf{ET}(s)$. The rest of the model data for the mixture m of the state s are mapped to the SDRAM. This procedure continues until all the available SRAM blocks are assigned. The assignment result is shown in Fig. 4 (a).

Since we are dealing with the case when the SRAM latency is faster than the SDRAM latency, the proposed heuristic method always leads to one of the optimal solutions for Eq. 6. If we move one SRAM block from the state with higher $f(s)$ to the state with lower $f(s)$, the Δ(total access time) is always positive, thus the total execution time will increase.

To ease the implementation overhead, the partition result is again sorted by descending order of $b(s)$. Figure 4 (b) shows the final placement of the model data.

This approach can be applied to any heterogeneous memory architecture with different costs. Since the data partition algorithm is performed in the off-line stage, it does not incur performance degradation except for the overhead of the loop distribution. The data partition algorithm may well show better performance when the speed gap between memories becomes bigger.

5. EXPERIMENTAL RESULTS

We measured the performance of the system according to the optimization steps. Table 2 shows the performance of the emission prob-

Table 2. The performance measurement of emission probability computation with SIMD optimization and software pipelining

	baseline	SIMD opt.	SW pipelining
Cycles (M)	186.02	143.69	121.55
Inst. (M)	61.96	16.12	17.11
CPI	3.00	8.91	7.10
Stall Cycles (M)	114.63	95.08	70.08
Stall Rate (%)	61.62	66.17	57.66

ability computation after SIMD optimization and software pipelining respectively. The SIMD optimization reduced the number of instructions about 74% while the execution cycles was only reduced by about 23%. This results in rather higher cycles per instruction (CPI). The IWMMX load instruction exhausts the available memory bandwidth. Then the data stall cycles increases, resulting in higher execution cycles. After applying software pipelining technique and inserting prefetch instructions, the pipeline stalls due to load-to-use latencies are reduced. The instruction count increased a little due to the insertion of prefetches. As a result of the software pipelining, the data stall rate is decreased by 12.86%, which, in turn, diminishes the core cycle count by 15.41%.

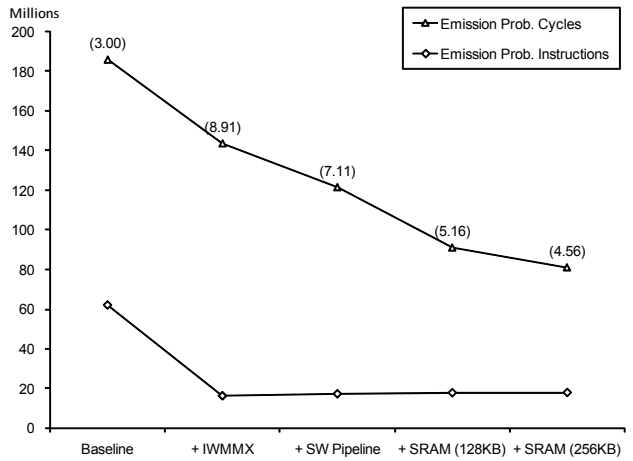

Fig. 5. The number of machine cycles and instructions according to the optimization steps. Numbers in parentheses indicate CPI.

Figure 5 shows the machine cycles and instruction counts of the emission probability computation for the optimization steps. Corresponding CPI values are shown in parentheses. Considering that cache line size of XScale is 32Bytes, we set s_{SIZE} as 32Bytes, thus $b(s)$ and $\mathbf{ET}(s)$ are between 0 and 5. We experimented with 2 configurations of $SRAM_{SIZE}$: 128KB and 256KB. Compared to the Baseline version, the execution cycle time of +SRAM (256KB) version in emission probability computation was decreased by 56.42%, which thus leads to the total execution time reduction of 22.85%. In optimization step +IWMMX, the instruction count was reduced by 73.98%, however CPI increased by a large amount due to the memory access overhead. With the SRAM optimization steps, CPI and data stall rate were decreased due to the improved memory performance, which led to the performance enhancement.

6. CONCLUDING REMARKS AND FUTURE WORKS

We have implemented a CSR system on an Intel XScale-based mobile platform. The likelihood computation was optimized by exploiting several architectural features, such as IWMMX SIMD coprocessor, data prefetch, and on-chip SRAM. Software pipelining and SIMD optimization were applied to fully grab instruction and data level parallelism. We also proposed the on-chip SRAM and external SDRAM partition algorithm which relieves the cost of intensive memory accesses. Given the algorithmic optimization, our optimization reduced the execution time of the likelihood computation by 56.42% without any performance degradation in terms of WER. As a result, the overall execution cycles is reduced by 22.85%. The optimization of the search network exploiting available parallelism will be our future work.

7. ACKNOWLEDGEMENT

This work is supported by IT R&D Project funded by Korean Ministry of Information and Communications and the Brain Korea 21 Project of Korean Ministry of Education.

8. REFERENCES

[1] D. Huggins-Daines, M. Kumar, A. Chan, A. W. Black, M. Ravishankar, and A.I. Rudnicky, "PocketSphinx: A free, real-time continuous speech recognition system for hand-held devices," *IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 1, pp. 185–188, 2006.

[2] S. Ishikawa, K. Yamabana, R. Isotani, and A. Okumura, "Parallel LVCSR algorithm for cellphone-oriented multicore processors," *IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 1, pp. 177–180, 2006.

[3] P. R. Panda, N. D. Dutt, and A. Nicolau, "On-chip vs. off-chip memory: The data partitioning problem in embedded processor-based systems," *ACM Transactions on Design Automation of Electronic Systems*, vol. 5, no. 3, pp. 682–704, July 2000.

[4] S. Kanthak, K. Schutz, and H. Ney, "Using SIMD instructions for fast likelihood calculation in LVCSR," *IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 3, pp. 1531–1534, 2000.

[5] S. Young et al., *The HTK Book Version 3.3*, 2005.

[6] Intel Corporation, *Intel XScale Core Developer's Manual*, January.

[7] Intel Corporation, *Intel Wireless MMX Technology Developer Guide*, August.

[8] B. L. Pellom, R. Sarikaya, and J. H. L. Hansen, "Fast likelihood computation techniques in nearest-neighbor based search for continuous speech recognition," *IEEE Signal Processing Letters*, vol. 8, no. 8, pp. 221–224, August 2001.