AN FPGA IMPLEMENTATION OF SPEECH RECOGNITION
WITH WEIGHTED FINITE STATE TRANSDUCERS

Jungwook Choi, Kisun You, and Wonyong Sung

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

ABSTRACT

In this paper we present a hardware architecture for large vocabulary continuous speech recognition that conducts a search over a weighted finite state transducer (WFST) network. A pipelined architecture is proposed to fully utilize the memory bandwidth. A hash table is used to manage small sized working sets efficiently. We also applied a parallelization technique that increases the traversal speed by 17%. The recognition system is fully functional on an FPGA, which runs at 100MHz. The experimental result on the Wall Street Journal 5,000 vocabulary task shows that the recognition speed of the system is 5.3× faster than real-time.

Index Terms—Speech Recognition, FPGA, Weighted Finite State Transducer

1. INTRODUCTION

Speech recognition provides many useful applications such as a user interface in mobile systems, information retrieval in video sequences, keyword marking in a lecture, and so on. Large vocabulary continuous speech recognition (LVCSR) is preferred in most of these applications. However, some of them further demand low power implementation, while others need much faster recognition speed than real-time.

Implementing LVCSR in hardware complies with those demands of speech recognition applications, because we can utilize parallel and pipelined architectures, which either reduce power with low operation frequency or speed up the recognition. Thus, there have been several prior works that propose hardware architectures for speech recognition. Some of them adopted hardware accelerators to speed up particular operations [1][2]. Hardware architectures that perform complete continuous speech recognition with the vocabulary size of 1,000~5,000 were introduced in [3][4][5][6]. A multi-FPGA implementation of 5,000-word continuous speech recognition that runs much faster than real-time was also presented in [7]. However, none of those utilized weighted finite state transducer (WFST) techniques that provide efficient and flexible recognition.

Recently, a WFST has been widely adopted as a search network in speech recognition inspired by the work in [8]. In this scheme, various information for speech recognition such as grammar, pronunciations of words, and hidden Markov model (HMM) structures can be composed and optimized into one flattened network. It enables efficient recognition, because the redundancy of the network is eliminated by finite state machine optimization techniques. A WFST-based search network is also very suitable for hardware-based recognition systems. We can easily modify the recognition system parameters or adopt new techniques without changing the underlying hardware architecture. For example, migration of the language model from bigram to trigram does not require any modification in the recognition hardware. Thus, a well-defined hardware architecture can be reused for many speech recognition applications.

In this paper we present a hardware architecture that performs efficient speech recognition over a WFST search network. A pipelined architecture for arc traversal is proposed, and a hash table based workload list that requires only small amount of management overhead is employed. We also parallelized the data path for arc processing during search to fully exploit the available memory bandwidth.

This paper is organized as follows. Section 2 explains speech recognition conducted with WFST networks. The pipelined architecture for the recognition is described in Section 3. Parallelization of the data path to speed up the recognition is presented in Section 4. Section 5 shows the performance and the hardware utilization of the implemented system. The concluding remarks are made in Section 6.

2. WFST SPEECH RECOGNITION

A WFST is a Mealy finite state machine characterized by arcs with five attributes: source state, destination state, input label, output label, and weight. Four hierarchical knowledge sources usually utilized in speech recognition can be represented by WFSTs: HMM structure $H$, context dependency $C$, pronunciation lexicon $L$, and $n$-gram grammar $G$. These WFSTs are composed and optimized into one $H \circ C \circ L \circ G$ WFST [8]. Since there is no hierarchy in the composed WFST network, the recognizer just needs to traverse over the network with no consideration of recognition system characteristics.

Fig. 1 shows a simple example of the WFST network. Since we utilize an $H$-level WFST network, input labels of the arcs indicate context dependent HMM states that correspond to the transition. The output labels, on the other hand, stand for the recognizable words,
which are stored as a history for the recognition result. There are two types of arcs. Arcs with an input label are called ‘non-epsilon arcs’ and the others with no input label are called ‘epsilon arcs.’

While a feature of input speech is extracted at every frame, the recognizer traverses the WFST network with three stages: emission probability computation, non-epsilon arc transitions, and epsilon arc transitions. If we assume that the state 3, 4, and 8 are ‘active’ in Fig. 1, the cost of the state 5 and 10 should be updated through the outgoing non-epsilon arcs by the following equation (non-epsilon arc transitions):

\[
\psi_t(s_j;w_{ij}) = \max_i \{\psi_{t-1}(s_i;w_{t-1,y}) \cdot a_{ij} \cdot b(O_t;m_k)\},
\]

where \(a_{ij}\) is a transition probability from the state \(i\) to the state \(j\), and \(b(O_t;m_k)\) represents the emission probability that corresponds to an input label \(m_k\). If there exists an output label in the arc, the word history of the state, \(w_s\), is also updated. Since many arcs in the network share the same input label, we compute the emission probability for the input labels of all outgoing non-epsilon arcs of active states before the non-epsilon arc transitions (emission probability computation).

After the non-epsilon arc transitions are performed, the cost is also propagated through epsilon arcs by the following equation (epsilon arc transitions):

\[
\psi_t(s_j;w_{ij}) = \max_i \{\psi_t(s_i;w_{t,y}) \cdot a_{ij}\},
\]

Unlike the non-epsilon arc transitions, the epsilon arc transitions have the same time index \(t\) after the transitions. Thus, if there is a chain of consecutive epsilon arcs, we must traverse all outgoing epsilon arcs from each destination state until we reach a state with no outgoing epsilon arc. For example, both the state 9 and 11 in Fig. 1 should be updated by epsilon arc transitions starting from the state 5. The multiple levels of propagation may result in very complex and irregular hardware architectures. To avoid it, we inserted lookahead epsilon arcs into the network so that every state connected by multiple expansions of epsilon arcs is reachable in only one level [9].

3. RECOGNITION SYSTEM ARCHITECTURE

3.1. Overall architecture

We used Xilinx’s Evaluation Board ML506 as a target platform [10]. It has a Virtex-5 SX50T FPGA, 256MB DDR2 SDRAM, and various peripherals. The recognition system consists of a host processor, emission probability computation unit, Viterbi search unit that performs both non-epsilon and epsilon arc transitions, a customized DRAM controller, and an external DRAM. We use the Xilinx MicroBlaze as a host processor, which controls the functional units and also extracts feature vectors from speech.

3.2. Emission probability computation unit

Since the emission probability computation fetches large size of acoustic model data from DRAM, it may exhaust the attainable memory bandwidth of the system. To reduce the amount of memory access and storage, the sub-vector quantization method is applied [11][12]. In this method, partial distances between the sub-vectors of a feature and Gaussians are computed and stored in the lookup table first. Then, the emission probability is computed by the sum of partial distances fetched in the lookup table. A parallel and pipelined architecture is utilized, which can compute up to 8 distances simultaneously.

3.3. Viterbi search unit

3.3.1. WFST data structure

A WFST network is constructed by the data structure of states and arcs. The state data consists of three elements: the address of the first outgoing arc, the number of non-epsilon arcs, and the number of epsilon arcs. Each state is represented by 64-bit. As explained in Section 2, the cost of a source state is propagated to destination states through the outgoing arcs with different input labels and weights. Since all the outgoing arcs are processed in a consecutive manner and the outgoing arcs from the same source state are stored continuously in DRAM, we just need to store the starting address of the first outgoing arcs and the number of non-epsilon and epsilon arcs to be traversed. The arc data has four elements: the address of the destination state, transition weight, an input label (the address of the corresponding emission probability), and an output label (the word identifier for the recognition result), which are packed into a 64-bit data.

3.3.2. Non-epsilon and epsilon arc transitions

The Viterbi search unit depicted in Fig. 2 processes both non-epsilon and epsilon arc transitions. The procedure of arc transitions is divided into two stages: the data loading stage and the data processing stage. In the data loading stage, active states and arcs are fetched from DRAM. Since the number of active states at each frame is at most 1~5% of the total number of states, we keep an active state list in the internal memory to contain the address of the source states to be traversed. In the pipelined execution, source state data indexed by the address in the active state list is fetched. Then, the data for outgoing non-epsilon or epsilon arcs of the source state are read from DRAM. These data are stored in the arc queue until they are processed.

The data processing stage has two major roles: updating and pruning the active states. In the beginning, one of the arcs stored in the queue is issued to calculate a propagation cost, which is either the sum of the source cost, transition weight, and emission probability in non-epsilon arc transitions as shown in Eq. (1) or the sum of the source cost and transition weight in epsilon arc transitions as shown in Eq. (2). If the cost is larger than the beam threshold, it is pruned and is not propagated to the destination state. The beam threshold is adjusted adaptively by the number of active states during search. If not pruned, the computed cost is compared with the current cost.
of the destination state, and is propagated when it is smaller than the current cost.

All stages are processed in a pipelined manner. An arc in the queue is processed at every cycle. A new arc data is fetched as soon as there is an empty slot in the queue. If there are no arcs waiting for the execution, the data loading stage issues data for the next active state. This procedure continues until all the active states in the list are issued.

3.3.3. Active state hash table

Since the cost of active states are compared and updated very often, it is beneficial for the active state list to store the cost of the states as well as the address of them. In this way, the cost of the destination can be read and updated fast, because the active state list resides in the internal memory. However, indexing of a destination state may be expensive if we need to scan the list to find the state.

To deal with this indexing issue, a simple hash table is employed. The index of each table element is the result of a modulo operation of the state index. Thus, every state has a designated entry and can be accessed by its index. If two different states fall into the same entry, which is called hash collision, a new entry is assigned to store one of them, and a link is generated to attach the newly assigned entry to the previous entry. Thus, an implemented active state hash table has a field for linked ‘indirect’ entries as well as for ‘direct’ ones.

Fig. 3 shows the pipelined execution of the arc processing with the active state hash table. In the first stage, the new cost to be propagated to the destination state is calculated from the data of an issued arc. At the same time, the existing entry that has the same hashed index with the destination state is loaded. In the second stage, the state indices of the existing entry and the new destination state are compared. If two indices are the same, which means that the entry is already assigned for the destination state, the cost of the entry will be updated to the smaller one in the third stage. If they are different, we should look up links to find a new entry for the destination state. Thus, a process stall signal is enabled to stop the pipeline in the third stage. If the write enable signals are properly controlled to avoid updating into the same entry at the same time, this collision can be solved. If the write enable signals are properly controlled to avoid updating into the same entry at the same time. Therefore, when a write-write collision occurs, the cost of the destination data for two data paths is compared, and only one of the write enable signals is enabled to update the state with the smaller cost.

5. EXPERIMENTAL RESULTS

5.1. Experimental setup

The acoustic model of the recognition system was trained by HTK [13] with the speaker independent training data in the Wall Street Journal 1 corpus. The acoustic model consists of 3,000 Gaussian mixture models, and each of them is composed of 16 Gaussians. For the evaluation, we utilized the Wall Street Journal 5,000-word continuous speech recognition task. The WFST network utilized in this paper was built using the tool described in [14]. It automatically composes and optimizes multiple WFSTs and generates the $H \circ C \circ L \circ G$ WFST network. The final network consists of 3,923,967 states and 14,390,482 arcs. The language model weight is set to 16.0. The beam threshold is adaptively adjusted to make the active state list fit into the internal memory. The word error rate (WER) of the recognition system is 9.51%.

5.2. Recognition speed

The execution time of the recognition on FPGA is summarized in Table 1. The execution time is normalized to one second of speech. For emission probability computation, the sub-vector quantization scheme is applied. Since currently the emission probability of all Gaussian mixture models are computed at every frame, the computation consumes around 50~55% of the execution time. If we compute the emission probability only for active ones, the execution time will be much decreased although this needs more complex control.
The execution time of the Viterbi search including non-epsilon and epsilon arc transitions is reduced almost 14.3% by parallel processing of arc transitions. The speed-up is limited because of large delay of non-sequential DRAM accesses caused by arbitrary activation of states. The final recognition system runs 5.3× faster than real-time. Although there are several prior works that perform 5,000-word continuous speech recognition [4][5][6][7], fair performance comparison is difficult due to the different language models, test set perplexities, and so on. Thus, we only compare the recognition speed with our previous works that utilize the same language model and the same test set. Compared to our previous works that do not employ the WFST network [5][6], the developed system executes the recognition around 3.5× and 2.9× faster, respectively.

Table 1. Execution time (ms) of the recognition normalized for one second of speech.

<table>
<thead>
<tr>
<th>Function</th>
<th>no parallel datapath</th>
<th>with parallel datapath</th>
</tr>
</thead>
<tbody>
<tr>
<td>EP computation</td>
<td>103.1</td>
<td>103.1</td>
</tr>
<tr>
<td>non-eps transition</td>
<td>99.8</td>
<td>85.5</td>
</tr>
<tr>
<td>eps transition</td>
<td>0.1</td>
<td>0.1</td>
</tr>
<tr>
<td>Total</td>
<td>203.0</td>
<td>188.7</td>
</tr>
</tbody>
</table>

5.3. Synthesis results

The FPGA synthesis results of the two speech recognition systems are shown in Table 2. The slice usage is slightly increased due to the parallelization of the data path. The total amount of internal memory implemented for the parallel version, 438.8KB, is about 2.1 times larger than the implementation without the parallelized datapath, 204.8KB. This result is due to the duplicated dual port memory for the active state hash table used in the parallelized version. Note that the maximum amount of block memory inside the target FPGA is 594KB.

Table 2. FPGA synthesis result.

<table>
<thead>
<tr>
<th>Unit</th>
<th>w/o parallel datapath</th>
<th>w/ parallel datapath</th>
</tr>
</thead>
<tbody>
<tr>
<td>Slice Register</td>
<td>20,015 (61%)</td>
<td>20,303 (62%)</td>
</tr>
<tr>
<td>Slice LUT</td>
<td>18,976 (58%)</td>
<td>20,402 (62%)</td>
</tr>
<tr>
<td>Slice LUT FF</td>
<td>29,431 (90%)</td>
<td>29,974 (91%)</td>
</tr>
<tr>
<td>RAMB18X2s</td>
<td>11 (8%)</td>
<td>3 (2%)</td>
</tr>
<tr>
<td>RAMB18X2SDPs</td>
<td>4 (3%)</td>
<td>0 (0%)</td>
</tr>
<tr>
<td>RAMB36_EXPs</td>
<td>38 (28%)</td>
<td>96 (72%)</td>
</tr>
<tr>
<td>DSP48Es</td>
<td>19 (6%)</td>
<td>19 (6%)</td>
</tr>
</tbody>
</table>

6. CONCLUDING REMARKS

We have developed a fully functional FPGA system that performs speech recognition with a WFST search network. A pipelined architecture for efficient processing of arc traversal over the WFST network has been proposed. The architecture for parallel execution of the arc processing to harness available memory bandwidth is also presented. The implemented system runs about 5.3× faster than real-time for 5,000-word continuous speech recognition. Currently, we are pursuing to apply larger recognition tasks with the vocabulary size of 20K to 50K for the implemented system.

7. ACKNOWLEDGEMENTS

This work was supported in part by the Brain Korea 21 Project and the National Research Foundation of Korea(NRF) grant funded by the Korea government(MEST) (No. 20090073770).

8. REFERENCES


