Circuits and Systems, 2012, 3, 10-16
http://dx.doi.org/10.4236/cs.2012.31002 Published Online January 2012 (http://www.SciRP.org/journal/cs)

A C-Based Variable Length and Vector Pipeline Architecture Design Methodology and Its Application

Takashi Kambe[1], Nobuyuki Araki[2]
[1]Department of Electrical and Electronic Engineering, Kinki University, Higashi-Osaka, Japan
[2]Graduate School of Science and Technology, Kinki University, Higashi-Osaka, Japan
Email: tkambe@ele.kindai.ac.jp

Received November 5, 2011; revised December 9, 2011; accepted December 18, 2011

ABSTRACT

The size and performance of a System LSI depend heavily on the architecture which is chosen. As a result, the architecture design phase is one of the most important steps in the System LSI development process and is critical to the commercial success of a device. In this paper, we propose a C-based variable length and vector pipeline (VVP) architecture design methodology and apply it to the design of the output probability computation circuit for a speech recognition system. VVP processing accelerated by loop optimization, memory access methods, and application-specific circuit design was implemented to calculate the Hidden Markov Model (HMM) output probability at high speed and its performance is evaluated. It is shown that designers can explore a wide range of design choices and generate complex circuits in a short time by using a C-based pipeline architecture design method.

Keywords: Variable Length and Vector Pipeline Architecture; C-Based Design; System LSI; Speech Recognition

1. Introduction

The size and performance of a System LSI depend heavily on the architecture which is chosen. As a result, the architecture design phase is one of the most important steps in the System LSI development process and is critical to the commercial success of a device. Architectural design means determining an assignment of the circuit functions to resources and their interconnection, as well as the timing of their execution. The macroscopic figures of merit of the implementation, such as circuit area and performance, depend heavily on this step [1]. Recently, three types of architecture design style are used for System LSI design.

The first is processor based architecture design. This is based on specific computing architectures such as DSP [2], ASIP [3], heterogeneous systems on chip [4], MPSoC [5], network on chip [6], and so on. Such a method is suitable for large scale designs, but suffers the drawback that it is difficult to choose and/or change the type of processor to implement each application.

The second is architecture design based on specific architecture description languages such as LISA [7], EXPRESSION [8], and so on. Various architectures can be generated automatically using these languages, but they are restricted to pre-prepared architecture templates and limited by the language semantics.

The third is C-based architecture design. A C-based architecture design methodology offers the following advantages:

1) A C-based language with untimed semantics is suitable for large-scale architecture design.

2) A C level simulator handles bit-accurate operations and is 10 - 100 times faster than HDL simulators.

3) A High level synthesizer can compile a program (written in C) describing the untimed behavior of hardware into RTL VHDL. It can also automatically generate interface circuits for inter-process data transfer.

In this paper, we use the *Bach* system [9-12] for C-based architecture design. *Bach*'s input language, *Bach C*, is based on standard ANSI C with extensions to support explicit parallelism, communication between parallel processes, and the bit-width specification of data types and arithmetic operation. Circuits synthesized using *Bach* consist of a hierarchy of sequential threads, all running in parallel and communicating via synchronized channels and shared variables. Using the *Bach* system, designers can develop parallel algorithms, explore the architecture design space and generate complex circuits in a much shorter time than using conventional HDL based design methodologies.

In our investigation, we focused on C-based pipelining architecture design because compared to data oriented parallel processing, a pipelining architecture can achieve higher speed without increasing the circuit size. Especially we propose a C-based variable length and vector

pipelining (VVP) architecture design methodology and show how the performance can be improved by the use of loop optimizations, speeding up memory access and by creating application-specific arithmetic circuits.

The rest of this paper is organized as follows. Section 2 describes the C-based VVP architecture design methodology. In Section 3, the speech recognition algorithm is briefly explained. Section 4 describes the VVP architecture design to accelerate the output probability computation. Section 5 compares and evaluates the performance of each architecture design. Finally, Section 6 summarizes the work.

2. C-Based VVP Architecture Design Methodology

In this section, the VVP architecture design methodology and its acceleration by memory access method and application-specific arithmetic circuit design are described. And its design flow is also proposed.

2.1. Variable Length Pipelining

In *Bach* C-based design, parallelization can be applied at both the functional level and the loop structure level. The functional level pipeline processing is controlled only by the control signals among the functional modules. When the processing of one pipeline stage is complete, a signal is sent to the next stage. The next stage starts when this signal is received. Using this method, C-based functional level pipeline design permits different processing times for each stage, while conventional pipeline processing needs to maintain the same processing time for each stage. Therefore, variable length processes such as memory access conflict handling and recursion based calculations can be included in the pipeline.

Two kinds of *pragma* are used in the *Bach* system to accelerate loop calculations. The *unroll pragma* expands the specified for loop with fixed iteration number and computes as much of each step in parallel as possible. The *throughput pragma* automatically generates a loop pipelining structure with the specified throughput.

2.2. Memory Access Optimization

For many applications, memory access is often the major bottleneck. The simplest way to access large amounts of data at high speed is to store all the data in on-chip memory, but this increases chip size and is expensive. The following optimizations are applied depending on the type of data access.

1) When accessing large data multiple-times, small on-chip SRAM or registers are used to hold the most frequently accessed data and reduce the number of off-chip memory accesses.

2) When data is accessed sequentially, pipelining of the memory accesses and arithmetic calculations is effective. During internal data processing, the next data can be read into the on-chip memory. Thus only the data required for each calculation is placed in on-chip memory.

3) When accessing large off-chip memory, the memory is divided into separate *bank*s to avoid access conflicts.

4) When accessing several small off-chip memories, related data can be merged to reduce the number of memory accesses.

2.3. Application-Specific Arithmetic Circuit Design

In many cases, the processing overhead which occurs when arithmetic calculations are performed multiple times is another bottleneck in the system. The circuit size and processing time for multiplication, division and other basic functions are often especially large. In C-based design, the application-specific arithmetic hardware circuit is designed using the following procedure.

1) Select the target arithmetic block to be accelerated from the software.

2) Modify the software algorithm to facilitate implementation in hardware. For example, convert time-consuming operations, such as division, into simple and equivalent calculations.

3) Use a hardware algorithm which matches the required calculation. For example, a high order polynomial can be calculated using various shifter and adder approximations.

4) Use the *preserve pragma* of *Bach C* to ensure that the resource is not shared by other circuits.

A software implementation often uses floating-point arithmetic. However, because implementing floating-point arithmetic in hardware has a large area overhead, fixed-point arithmetic is most suitable. In addition, when minimizing the bit length of each fixed-point variable, the overflow and underflow for each calculation have to be considered. The *Bach* system's bit-accurate C level simulator enables the results of these calculations to be verified using real data with the specified bit lengths.

2.4. The C-Based VVP Architecture Design Flow

The C-based VVP architecture design flow, including memory access optimization and application-specific arithmetic circuit design, consists of the following steps.

1) The target specification such as clock frequency, processing time, and gate size of the circuit are specified.

2) The *Bach compiler* calculates the cycle number of the processing for every function in the hardware portion described by *Bach C*.

3) If (the cycle number of the *dominant stage*) <

(target cycle number), the function based variable pipeline architecture is adopted. The *dominant stage* means that its cycle number is maximum among all pipeline stages

4) Otherwise, the following four methods are applied repeatedly until achieving the target specification. The methods of which the cycle number of the *dominant stage* is reduced maximally and its circuit size is increased minimally are chosen.

(a) Vector pipeline architecture [13] is applied to the dominant stage using the plural pipelines. The number of the pipeline is given by the following equation.

$$NoP = \lfloor Cd/Tcn \rfloor \quad (1)$$

where *NoP* denotes the number of the pipeline, *Cd* denotes the cycle number of the *dominant stage*, and *Tcn* denotes the target cycle number.

(b) The *unroll* and *throughput pragma*s are applied to the for loop structure.

(c) The memory access optimization of Subsection 2.2 is applied.

(d) The application-specific arithmetic circuit design of Subsection 2.3 is applied.

The cycle number of each stage is estimated using the high-level synthesized result. In many cases, two or more methods are mixed to accelerate the architecture effectively.

3. Speech Recognition Application

We applied this C-based VVP architecture design methodology to large vocabulary continuous speech recognition using the open-source software Julius [14]. The most popular acoustic model for speech recognition is a Hidden Markov Model (HMM), where speech signals are modeled as time-sequential automata that compute output probabilities for the given speech segment (frame) and also have a probabilistic transition.

The spectrum of a fixed width speech frame is analyzed and its acoustic features are extracted. Julius extracts 25 ms frames at 10 ms intervals. The output probability is computed using the HMM algorithm and the acoustic features are extracted from the input speech frame. The acoustic features of each frame are expressed by a *p*-dimensional vector. The output probability of HMM is expressed as a Gaussian mixture distribution. Gaussian distribution calculation for state *i* and mixture *m* is given by the following expression.

$$\log b_{im}(o_t) = \omega_i - \frac{1}{2}\sum_{p=1}^{P}\frac{1}{\sigma_{imp}^2}\left(o_{tp} - \mu_{imp}\right)^2 \quad (2)$$

where o_{tp} denotes the p-dimensional input vector for frame *t*, ω_i denotes the mixture weight coefficient, μ_{imp} denotes the average vector and σ_{imp}^2 denotes the variance vector. The output probability b_i of M Gaussian mixture distributions is given by the following expression.

$$\log b_i(o_t) = \log\sum_{m=1}^{M}\exp(b_m) \quad (3)$$

Some researchers [15-18] have adopted a Simultaneous Multi-threading or Symmetric Multiprocessing architecture using a 32 bit RISC processor with some kind of co-processor to realize real-time and low-power large vocabulary speech recognition. Another researcher [19] has developed a hardware accelerator architecture for the Gaussian mixture distribution calculation to achieve real-time processing. Nevertheless, it still required a 32 bit RISC processor to implement the remaining functions. Therefore, the size of these systems is very large.

4. Pipeline Architecture Design of Output Probability Computation

This section discusses the architecture of the output probability computation circuit as an example of C-based VVP architecture design. There is a trade-off between speed and cost. In this paper, the goal of the output probability computation circuit design is to minimize the total gate count while maintaining the required performance.

4.1. VVP Architecture for the Output Probability Computation

Although the output probability can be computed in parallel for mixtures, memory access, Gaussian distribution, exponential and logarithmic (*EL*) calculations have to be processed sequentially. To accelerate the output probability computation, we applied the VVP architecture for memory access, Gaussian distribution, exponential and logarithmic calculations. Our phoneme-HMM model has 1012 states and four mixtures to recognize large vocabulary continuous speech.

Because the processing time of Gaussian mixture distribution calculation for one state and one mixture is 5731 ns from **Table 1** and the target processing time of the output probability calculation is under 10 ms, the calculation result of the Equation (1) is as follows.

$$\lfloor (5731\text{ns} \times 1012 \times 4)/10\text{ms} \rfloor = 3 \quad (4)$$

The number of pipeline is four from this result and the number of mixture in this case.

4.2. The Arithmetic Circuit Design for Output Probability Computation

By experimenting with various numbers of bits in **Table 2**, the minimum number of integer and decimal bits which have no overflow and underflow in the output probability

Table 1. The processing time(ns) of each stage in the pipelining.

No.	buffering	Gaussian	EL cal.
(A)		*5731	290
(B)		*5333	228
(C)	3391	*4835	214
(D)	1070	*4820	228
(E)		*4851	228
(F)	*3357	1344	290
(G)		*1135	228
(H)	1070	*1115	228

*: the dominant stage.

Table 2. The number of bits of fixed-point arithmetic.

No. of integer bits	Recognition rate (%)	No. of decimal bits	Recognition rate (%)
floating point	94.72	floating point	94.72
11	0.55	15	4.36
12	0.55	16	13.54
13	13.42	17	18.51
14	94.72	18	94.72
15	94.72	19	94.72
16	94.72	20	94.72

20 decimal bits / 14 integer bits

calculation were selected. It was found that when 14 integer bits and 18 decimal bits are used, the recognition rate is similar to that of the floating point arithmetic implementation in *Julius* system. In this experiment, five kinds of acoustic samples which have high recognition rate by Julius software[14] were used.

Two *EL* calculation circuits based on the continued fraction and on the *Faster Shift and Add* (FSA) algorithm [20] were also designed and their performances compared. Though the continued fraction has many divisions, *FSA* algorithm has simple calculation such as shift and addition. The processing time of the circuit for Continued fraction and FSA algorithm are 21.864 ms and 20.241 ms, respectively. The circuit area are 62,812 and 48,386 gates. The FSA algorithm is about 14% faster than the continued fraction algorithm for output probability computation.

4.3. Memory Access Optimization Methods and Loop Unrolling

To improve the memory access speed, we propose two kinds of memory access optimizations. One is to use two buffers to access the HMM parameters for each mixture. The size of each buffer is 208 bytes. HMM parameters for the next Gaussian distribution calculation is read into buffer 2(1) while the Gaussian distribution calculation circuit is accessing buffer 1(2). The other is HMM memory separation. In the pipelining, the data access to HMM RAM by Gaussian distribution calculation circuits often has conflicts. By separating HMM RAM, each Gaussian distribution calculation circuit can access HMM RAM at the same time (*memory separation*).

The 25 dimensional Gaussian distribution calculation is accelerated by utilizing *Bach* system's unroll command.

5. Design Results and Its Productivity

In this section, the design results for each architecture are compared (**Table 3**). All architectures in this section adopt the *EL* calculation based on the FSA algorithm. To evaluate the performance of the pipeline architecture, we also implemented sequential processing. **Table 3** shows the processing time of each pipeline stage in each architecture. These numbers are the average of processing time of four mixtures. In this investigation the gate level circuits are synthesized from RT level HDL using Design Compiler provided by VDEC and mapped to Hitachi 0.18 micron CMOS library cells and the clock frequency of all circuits is 100 MHz.

The following equation calculates the processing time of each architecture from that of the dominant stage.

$$Mix/Pipe \times St \times TD = TT \quad (5)$$

where *Mix* denotes the number of mixtures, *Pipe* denotes the number of pipeline stages, *St* denotes the number of states, *TD* denotes the processing time of the dominant stage in **Table 1** and *TT* denotes the processing time of the architecture in **Table 3**. For example of the archi-

Table 3. The comparison of each architecture.

arch.	mem. separ.	buffer	unroll	area (gates)	time (ms)
soft					55,300
Seq.	no	no	no	48,385	20.341
(A)	no	no	no	171,968	5.826
(B)	yes	no	no	170,813	5.396
(C)	no	yes	no	294,579	4.906
(D)	yes	yes	no	293,706	4.891
(E)	no	no	yes	316,725	5.127
(F)	no	yes	yes	547,231	4.345
(G)	yes	no	yes	439,783	1.226
(H)	yes	yes	yes	546,080	1.127

tecture (C), the result of the equation (4893 ms) is almost equal to 4906 ms in **Table 3**.

$$4/4 \times 1012 \times 4835 = 4893 \text{ ms} \quad (6)$$

The error of Equation (6) is small and it is because there are variations of the acoustic data and the convergence of *EL* calculation.

The vector pipelining circuit (A) is about four times faster than the sequential circuit. The processing time of the Gaussian distribution calculation in this architecture includes the memory access resolving time.

The results for architecture (B), (C), and (D) show the following points. The data buffering and the HMM RAM separation reduce the memory access time from the processing time of the dominant stage, but the data buffering requires additional gates for the buffers. The HMM RAM separation with data buffer implementation is best for speeding-up memory access, but the improvement is 16% or less. Because the data buffering in (C) and (F) has memory access conflicts, these processing times are larger than ones in (D) and (H).

The architectures with loop unrolling such as (F), (G), and (H) are faster than the ones without it. However, because there are memory access conflicts, architecture (E) is slower than the others. Resolving access conflicts by using the memory access methods, the architecture (H) achieves the highest performance.

These results also show that VVP processing permits different processing time of each stage such as memory access conflict and the convergence variation of *EL* calculation.

For real-time processing one frame must be handled in less than 10ms and these architectures are able to achieve real-time processing of the output probability computation and the architecture (F), (G), and (H) can use lower clock frequency to reduce their power consumption.

The architecture (H) in **Figure 1** has the following behaviors. For each mixture, the Gaussian distribution calculation circuit reads the acoustic features from registers and HMM parameters from data buffers, and computes a 25 dimensional Gaussian distribution. The result is sent to *EL* calculation circuit. The *EL* calculation circuit receives the results of *EL* calculations and Gaussian distributions from the previous mixtures and adds to the output probability of the current mixture. The result of the *EL* calculation at the final mixture is the output probability. The *Bach C* description of this architecture is shown briefly in **Figure 2**. This architecture is described by four functions and the communication among them using primitives such as *send*() and *receive*() to control the VVP process.

Table 4 shows the number of lines used in the *Bach C* description and the design time to change the architecture

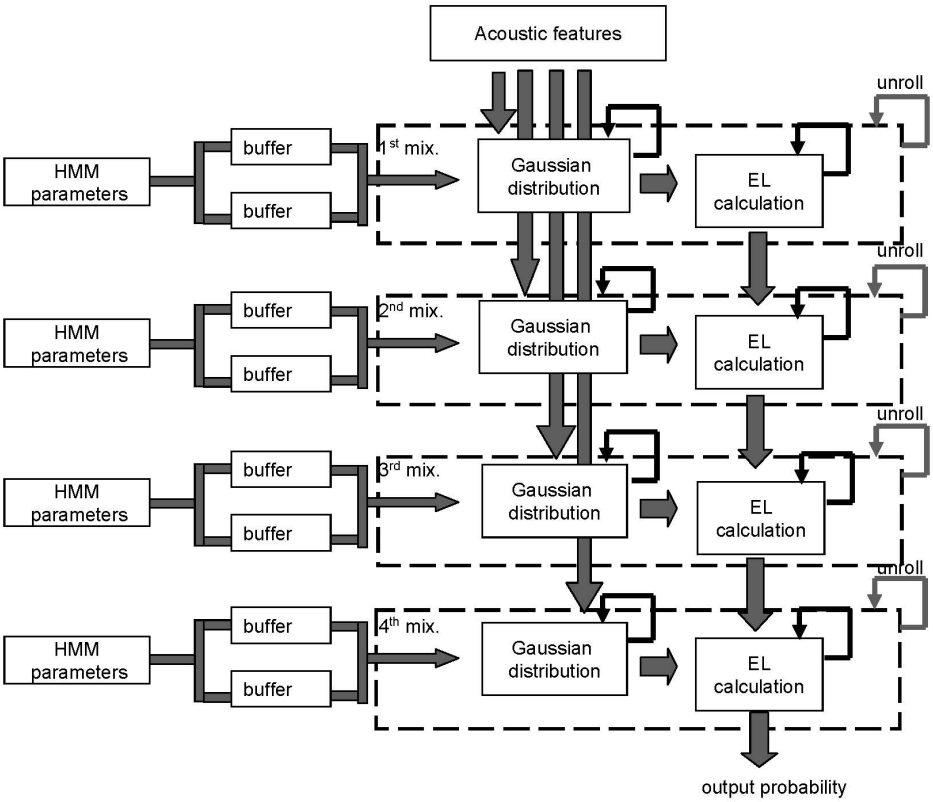

Figure 1. Block diagram of the architecture (H).

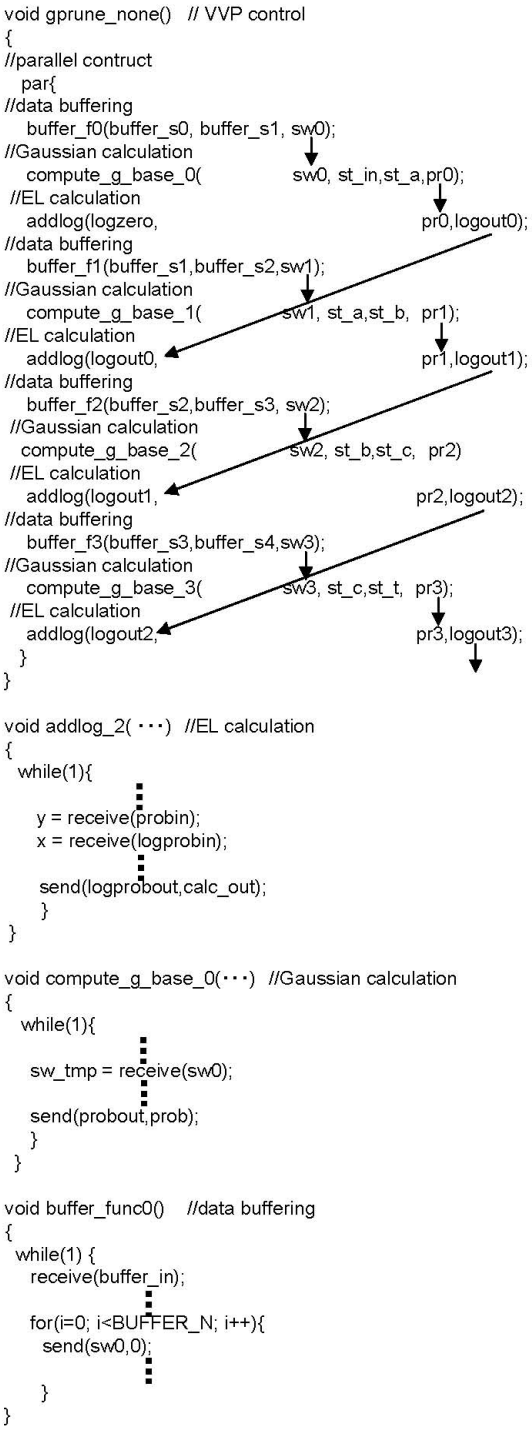

Figure 2. The Bach C description of Figure 1.

from the previous one.

6. Concluding Remarks

In this paper, we proposed a C-based VVP architecture design methodology and VVP architectures accelerated by application-specific circuit design, memory access method, and loop unrolling were implemented to calculate the Hidden Markov Model (HMM) output probability at high speed and the implementation performances were evaluated. It was demonstrated that designers can explore a wide range of design choices and generate complex circuits in a short time by using a C-based architecture design methodology.

Table 4. The number of the descriptions and the design time.

arch.	Bach C (lines)	Design time (days)
(S)	143	10
(A)	283	8
(B)	306	4
(C)	481	4
(D)	505	4
(E)	283	4
(F)	481	4
(G)	306	4
(H)	503	5

7. Acknowledgements

The authors would like to thank the *Bach* system development group in SHARP Corporation Electronic Components and Devices Development Group, for their help in hardware design using the *Bach* system. This work is supported by the VLSI Design and Education Center (VDEC), the University of Tokyo in collaboration with Synopsys Corporation.

REFERENCES

[1] G. D. Micheli, "Sythesis and Optimization of Digital circuits," McGraw-Hill, New York, 1994.

[2] F. Catthoor and H. J. De Man, "Application-Specific Architectural Methodologies for High-Throughput Digital Signal and Image Processing," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, Vol. 38, No. 2, 1990, pp. 339-349. doi:10.1109/29.103069

[3] S. Kobayashi, K. Mita, Y. Takeuchi and M. Imai, "Design Space Exploration for DSP Applications Using the ASIP Development System PEAS-III," *Proceedings of the Acoustics, Speech, and Signal Processing*, Vol. 3, 13-17 May 2002, pp. 3168-3171.

[4] H. Blume, H. Hubert, H. T. Feldkamper and T. G. Noll. "Model-Based Exploration of the Design Space for Heterogeneous Systems on Chip," *Journal of VLSI Signal Processing Systems*, Vol. 40, No. 1, 2005, pp.19-34.

[5] S. Pasricha and N. Dutt, "COSMECA: Application Specific Co-Synthesis of Memory and Communication Ar-

chitectures for MPSoC," *Proceedings of the Conference on Design, Automation and Test in Europe*, 6-10 March 2006, pp. 700-705.

[6] M. P. Vestias and H. C. Neto, "Co-Synthesis of a Configurable SoC Platform Based on a Network on Chip Architecture," *Asia and South Pacific Conference on Design Automation*, Yokohama, 24-27 January 2006, pp. 48-53. doi:10.1109/ASPDAC.2006.1594644

[7] O. Schliebusch, A. Hoffmann, A. Nohl, G. Braun and H. Meyr, "Architecture Implementation Using the Machine Description Language LISA," *Proceeding of the Asia and South Pacific Design Automation*, Bangalore, 7-11 January 2002, pp. 239-244. doi:10.1109/ASPDAC.2002.994928

[8] A. Halambi, P. Grun, V. Ganesh, A. Khare, N. Dutt and A. Nicolau, "Expression: A Language for Architec- ture Exploration through Compiler/Simulator Retarge- tability," *The Proceeding of Design, Automation and Test in Europe Conference and Exhibition*, 9-12 March 1999, pp. 485-490. doi:10.1145/307418.307549

[9] K. Okada, A. Yamada and T. Kambe: "Hardware Algorithm Optimization Using Bach C," *IEICE Transactions on Fundamentals*, Vol. E85-A, No. 4, 2002, pp. 835-841.

[10] T. Kambe, A. Yamada, K. Okada, M. Ohnishi, A. Kay, P. Boca, V. Zammit and T. Nomura, "A C-Based Synthesis System, Bach, and Its Application," *Proceeding of the Asia and South Pacific Design Automation*, 30 January-02 February 2001, pp. 151-155. doi:10.1145/370155.370309

[11] T. Kambe, H. Matsuno, Y. Miyazaki and A. Yamada, "C-Based Design of a Real Time Speech Recognition System," *Proceeding of IEEE International Symposium on Circuits and Systems*, Island of Kos, 21-24 May 2006, pp. 1751-1755. doi:10.1109/ISCAS.2006.1692944

[12] K. Jyoko, T. Ohguchi, H. Uetsu, K. Sakai, T. Ohkura and T. Kambe, "C-Based Design of a Particle Tracking System," *Proceeding of the* 13*th Workshop on Synthesis and System Integration of Mixed Information Technologies*, 2006, pp. 92-96.

[13] H. Cheng, "Vector Pipeling, Chaining and Speed on the IBM 3090 and Cray X-MP," *IEEE Computer*, Vol. 22, No. 9, September 1989, pp. 31-42, 44, 46.

[14] K. Shikano, K. Itoh, T. Kawahara, K. Takeda and M. Yamamoto, "IT TEXT Speech Recognition System," Ohomu Co., May 2001 (in Japanese).

[15] T. Anantharaman and B. Bisiani, "A Hardware Accelerator for Speech Recognition Algorithms," *Proceedings of the* 13*th Annual International Symposium on Computer Architecture*, Vol. 14, No. 2, 1986, pp. 216-223.

[16] S. Chatterjee and P. Agrawal, "Connected Speech Recognition on a Multiple Processor Pipeline," *Proceedings of International Conference on Acoustics, Speech, and Signal Processing*, Vol. 2, 23-26 May 1989, pp. 774-777. doi:10.1109/ICASSP.1989.266542

[17] H. Hon, "A Survey of Hardware Architectures Designed for Speech Recognition," Technical Report CMU-CS-91-169, August 1991.

[18] S. Kaxiras, G. Narlikar, A. Berenbaum and Z. Hu, "Comparing Power Consumption of an SMT and a CMP DSP for Mobile Phone Workloads," *Proceedings of the International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, November 2001, pp. 211-220. doi:10.1145/502217.502254

[19] B. Mathew, A. Davis and Z. Fang, "A Low-Power Accelerator for the SPHINX 3 Speech Recognition System," *Proceedings of International Conference on Compilers, Architecture and Synthesis for Embedded Systems*, 2003, pp. 210-219. doi:10.1145/951710.951739

[20] J. M. Muller, "Elementary Functions," Birkhauser, Boston, 1997.

