A Design System for Special Purpose Processors Based on Architectures for Distributed Processing

Katsuhiko Shirai and Jin Hiwatashi
School of Science and Engineering, Waseda University, Tokyo, Japan

Abstract

In recent years, although design systems using high-level synthesis are becoming practicable, yet they are not powerful enough to design the digital signal processing (DSP) applications which are needed in real-time processing. In this paper, we introduce the system synthesis based on the assumption that a behavioral description is mapped to an architecture for distributed processing, and design several DSP algorithms which include the Forward algorithm for HMM (Hidden Markov Model) speech recognition. We also evaluate execution time and hardware resources of those applications, and indicate that the new design methodology is suitable for those DSP algorithms.

1 Introduction

As the advance of VLSI technology enables to realize a variety of VLSIs, the applications specific integrated circuits (ASICs), which execute a particular algorithm at a high speed, are increasing. Since the size of VLSIs increases, the system design becomes more complicated and more diversified, and the problems of the long-term project and the increasing cost are getting worse. As VLSIs apply to a variety of fields in the future, the demands for special purpose processors will increase. For satisfying those demands, design systems that VLSIs can be designed with simply and quickly are needed. In a background of this sort, we have been proposing and researching the design system for special purpose processors, named SYARDS, whose input is a behavioral description written in a high-level language[1]-[3].

Although there is a variety of architectures suitable for an algorithm, the research on our design system SYARDS using an algorithm description makes it clear that the structure design is needed at the initial stage of system design so as to find an optimal solution. As for the design of LD-CELP (Low-Delay Code Excited Linear Prediction), it is confirmed that the system structure affects the performance of the whole system seriously, and that the structural design is essential to implement large-scale systems. Therefore, in consideration of the structural design on a behavioral description, it is a prudent solution to try to obtain a high-performance system through manual partitioning.

On the other hand, the importance of the design environment using high-level synthesis with VHDL and the hardware/software codesign[4]-[7] are indicated in the field of system synthesis. In this paper, we propose the distributed synthesis, which is the new design method on the assumption that a input description is synthesized into the architecture for distributed processing, and evaluate a variety of DSP algorithms through the application of this design method.

2 Design Methodology

2.1 Design Methodology using Architecture for Distributed Processing

Although an uni-processor is adopted as a target architecture in a conventional high-level synthesis system except for special situations, the processors designed with the synthesis method may not satisfy the constrains on their performance, when the application is real-time processing, e.g. image and speech processing. In this paper, we propose the distributed synthesis, in which the architecture for distributed processing is adopted as a target architecture and the input description is assumed to be synthesized into the tight coupled multi-processor.

If the distributed processing architecture were synthesized through particular optimizations in a conventional high-level synthesis, the distributed synthesis would not be necessary. However because high-level synthesis systems practically do not perform such optimizations, the importance of the distributed synthesis increases in consideration of DSP applications.

2.2 Introduction of Parallel Statement

The strategies, which exploit the parallelism of an algorithm description, can be classified roughly into two types. The one is the strategy extracting the fine-grain parallelism automatically, the other is the strategy extracting the coarse-grain parallelism manually. Although the strategy extracting the fine-grain parallelism is adopted by many high-level synthesis systems as a principal optimization, the strategy extracting the coarse-grain parallelism is not exploited effectively, and

the parallel statements (e.g. $pbegin \sim pend$) are rarely prepared.

Therefore we introduce the parallel statements in the distributed synthesis, and intend to exploit the coarse-grain parallelism as much as possible. The mechanisms of the parallel programming[8], such as shared memory, mutual exclusion, and barrier synchronization, are provided in the form of function calls. The newly introduced functions are depicted in Figure 1, and the operations of those functions are explained as follows.

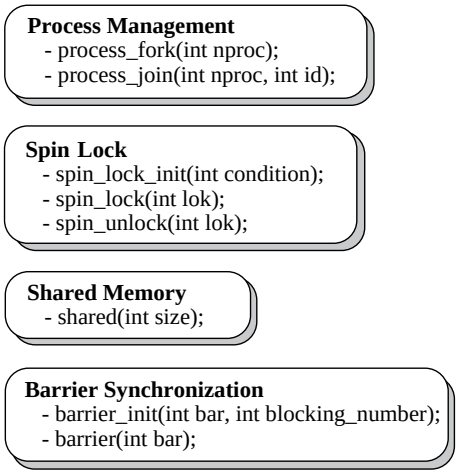

Figure 1: Newly introduced functions

When a parent processor calls the function $process_fork(nproc)$, $nproc - 1$ child processors begin to work. Each processor can detect the code sequences to execute on account of the processor identifier $process_id$ particular to it. When a processor calls the procedure $process_join()$, a child processor terminates and a parent processor waits until all children processors terminate.

The spin lock is the mechanism which protects a critical section, and only one processor can execute the critical section concurrently. The function $spin_lock_init()$ initializes the spin lock mechanism, and the procedure $spin_lock()$ and $spin_unlock()$ protects and releases the critical section respectively. The procedure $shared()$ declares the variable which can be shared by all processors.

The barrier synchronization synchronizes all processors so that a particular processor may not execute code sequences ahead. The function $barrier_init()$ initializes the barrier synchronization mechanism, and the procedure $barrier()$ makes each processor wait until all processors reach the execution point.

2.3 Processor Optimization using High-level Synthesis

In this implementation, because each processor concurrently executes the parallel description provided by a user, the coarse-grain scheduling already finishes on the stage of the algorithm description. However each processor may work faster by applying the processor optimization to each processor.

In the beginning, we plan to design the processors, which have a variety of instruction sets, by exploiting the difference of the code sequences that each processor executes. However because the design system exploits the coarse-grain parallelism, each processor would be homogeneous. Consequently we adopt the design method that each processor is designed as a homogeneous processor in this implementation. Therefore each processor is synthesized using a high-level synthesis, and will eventually be output as the processor information in hardware description language (HDL). It is assumed that the logic synthesis system, named PARTHENON, will be utilized for logic synthesis.

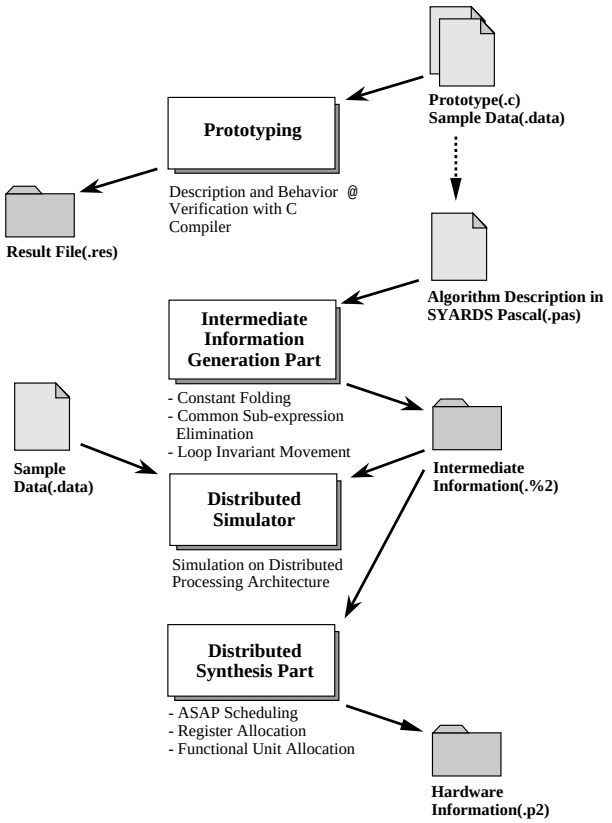

Figure 2: System overview of SYARDS

3 Design System SYARDS

3.1 System Overview

The system overview of the newly implemented system is depicted in Figure 2. In case of designing with a behavioral description, the algorithm to be implemented would already be described in a high-level language. The new design stage, the prototyping, is introduced so that the behavioral description, to which the parallel statements are added in the form of the newly introduced functions, may be verified with an existing C compiler. Since the newly introduced functions should be implemented by some means, each function is implemented with UNIX system calls, and eventually linked to a library at a run-time.

On the next stage, the algorithm described in C language at the prototyping should be translated into a behavior description in Pascal, which can be processed on the SYARDS. We obtain the intermediate information, which includes the parallel information, by means of processing the behavioral description with the intermediate information generation part. Besides, the intermediate information generation part which can processes C language is also under construction. The distributed simulator verifies the behavior of the algorithm with the intermediate information. Because the number of steps necessary to execute the algorithm can be evaluated with the distributed simulator, the effect of the distributed synthesis in case of executing on the distributed architecture can also be evaluated.

Finally the scheduling and the hardware allocation are performed with the distributed synthesis part, and the hardware information is obtained. Besides the hardware resources required for the execution of the behavioral description can also be estimated from the hardware information. We adopt the ASAP (As Soon As Possible) scheduling as the time-constrained scheduling algorithm, and the left-edge algorithm as the register allocation algorithm[9].

3.2 Implementation Methodology

In this section, we briefly mention how the description in HDL required to implement a system finally is obtained. The target architecture of this system is depicted in Figure 3. Since each processor is translated into the hardware information after hardware allocation, the description in HDL can be obtained by processing the hardware information.

The control unit, which controls the operation of each processor, should be designed so that it may instruct a proper operation against the demand of the newly introduced functions (e.g. *spin_lock*(), *spin_unlock*(), and *barrier*()) in the distributed synthesis. In order to implement the control unit, it is necessary that the templates described in HDL are prepared, and that the only operations that an arbitrary algorithm requires are synthesized. Because the processor units and the control unit described in HDL can be synthesized with the existing logic synthesis tool (e.g. PARTHENON), the hardware resources required for the implementation can also be estimated.

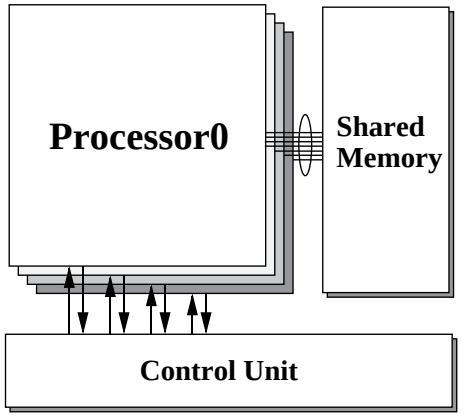

Figure 3: Target architecture

4 Applications

4.1 DSP Algorithms

The hardware designed with this system implements 8 digital signal processing (DSP) algorithms, as shown in Figure 4, which include 3 algorithms for the image processing and 5 algorithms for speech processing, and are chosen owing to their necessity of real-time processing.

Algorithms for Image Processing
- Discrete Cosine Transformation DCT (Scale: 192)
- Template Matching TEMP (Scale: 313)
- Median Filter MEDIAN (Scale: 264)

Algorithms for Speech Processing
- Pitch Extraction PITCH (Scale: 142)
- PARCOR Lattice Filter PARCOR (Scale: 142)
- Auto Correlation Function AUCOR (Scale: 42)
- HMM Speech Recognition FORWARD (Scale: 348)
- Humming Window HAMM (Scale: 50)

Figure 4: DSP algorithms applied to SYARDS

The Scale in Figure 4 denotes the number of the static steps on the intermediate information, and generally represents the scale of the algorithm. In this application, those DSP algorithms will be applied to this

system, and the effect of the distributed synthesis will be evaluated with the distributed simulator. Furthermore the effect of the processor optimization on each processor and the hardware resources (e.g. registers and functional units) to execute the algorithm will be measured.

4.2 HMM Speech Recognition

In this section, we concentrate on the HMM (Hidden Markov Model) speech recognition algorithm[10], as the relatively successful application among the abovementioned DSP algorithms.

```
{ process forking }
id := process_fork(nproc);
w := id;
while w < nword do
 begin ...
  for t := 1 to num do
   begin
    spin_lock(lok);
    alpha[t] := alpha[t-1] * a[0] * b[code[t-1]-1];
    spin_unlock(lok);
    for i := 1 to nstate-1 do
     begin
      spin_lock(lok);
      bb   := b[i*vq_level+code[t-1]-1];
      aa1 := a[(i-1)*nstate+i];
      aa   := a[i     *nstate+i];
      spin_unlock(lok);
      alpha[i*max_nsymbol+t]
            := alpha[(i-1)*max_nsymbol+t-1] * aa1 * bb
             + alpha[i      *max_nsymbol+t-1] * aa   * bb;
     end; ...
  tmp := ln(alpha[(nstate-1)*max_nsymbol+num]) - sum_c;
  spin_lock(lok); likeli[w] := tmp; spin_unlock(lok);
  w := w + nproc;
 end;
{ process joining }
process_join(nproc, id);
```

Figure 5: Behavioral description of HMM speech recognition

First the behavior of the algorithm can be verified with the algorithm description in C language on the prototyping stage of this system. Second a HMM speech recognition algorithm is described in the SYARDS Pascal, as shown in Figure 5, using the result of the prototyping stage. This program is the main part of the Forward algorithm for HMM speech recognition, and actually represents that the likelihood to each word is calculated by using the HMM parameters stored in advance, when the vector quantization (VQ) code of a word is input. Consequently the likelihoods to the *nword* words stored in advance should be calculated in order to recognize one word.

If the algorithm were executed on the conventional uni-processor architecture, the likelihood to each word would be calculated sequentially. However on our distributed synthesis system, likelihoods can be calculated with the arbitrary numbers of processor concurrently. The behavioral description in Figure 5 realizes this strategy, which actually utilizes the loop splitting in the parallel programming[8].

Because the variable *id* stores the value which can identify each processor and the number of processors *nproc* adds to the variable *w* at the end of the main loop, the calculations of the likelihoods to the words 0, 4, 8 are assigned to the processor 0, if the number of processors is assumed to 4. The 4 times recognition speed can be obtained by adopting this strategy in case of comparing with the execution on the uni-processor architecture. Therefore the distributed synthesis is important on the application field, which requires high-speed processing, and is extremely efficient for the application needed in real-time processing.

5 Evaluation

We apply the 8 DSP algorithms mentioned in the previous chapter to this design system, and then evaluate the effect of the distributed synthesis, the effect of the processor optimization, and the hardware resources required for its implement. In the following sections, we will indicate those consequences and consider on the consequences.

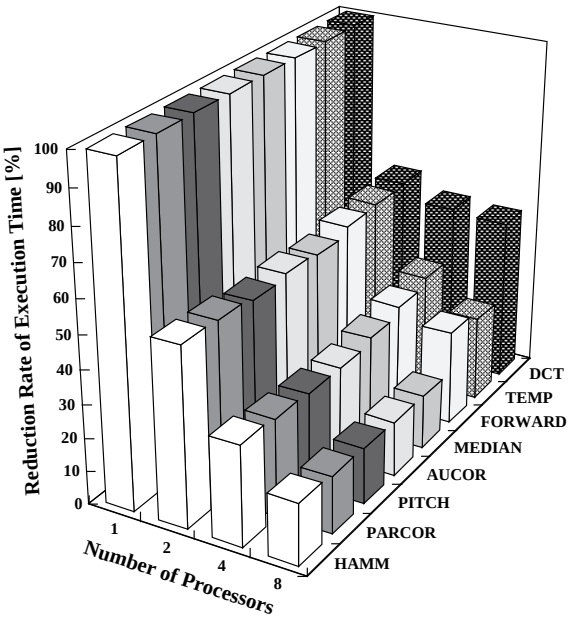

Figure 6: Relation between reduction rate of execution time and number of processors

5.1 Effect of Distributed Synthesis

The effect of the distributed synthesis on the DSP algorithms, when the number of processors is 1, 2, 4 and 8, is evaluated. The relation between the reduction rate of execution time and the number of processors is depicted in Figure 6, where the number of steps is converted, as the number of steps required for executing with 1 processor before the processor optimization is 100%.

Since the algorithms are relatively appropriate for this design methodology, the almost ideal result can be obtained. However as for the discrete cosine transformation (DCT), because the barrier synchronization is frequently used, the effect of increasing the number of processors can not be obtained as expected. Besides as for the HMM speech recognition (FORWARD) and the template matching (TEMP), the spin lock mechanism which protects the shared memory satiates the effect of the distributed processing.

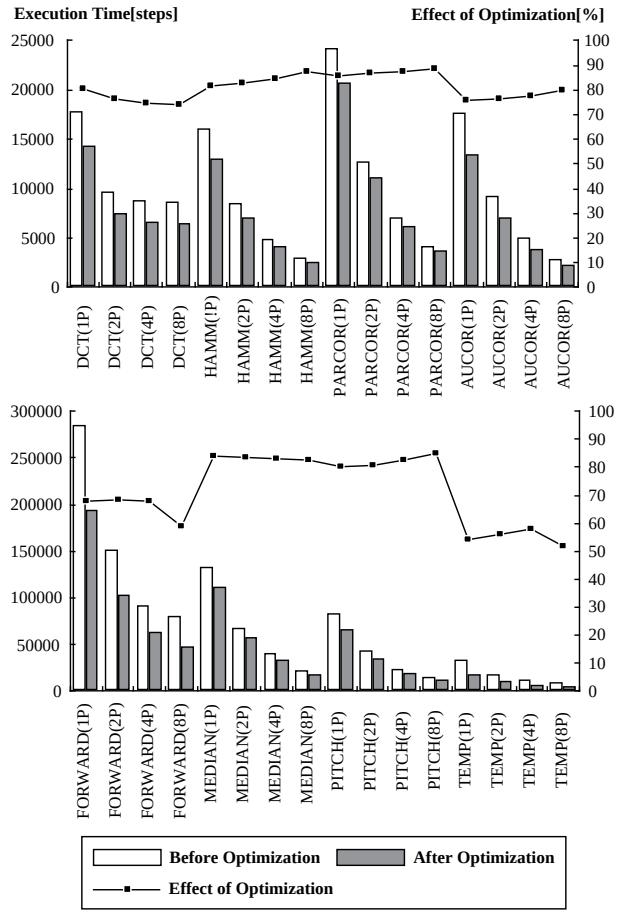

Figure 7: Effect of optimization on DSP algorithms

5.2 Effect of Processor Optimization

The number of execution steps before and after the optimization are evaluated on the DSP algorithm, and then the result of calculating the optimization effect (i.e. after optimization / before optimization × 100) is shown in Figure 7. Consequently the 80% optimization effect can almost be obtained on every applications. In particular, because HMM speech recognition (FORWARD) and the template matching (TEMP) includes a large quantity of fine-grain parallelism, the 50–60% optimization effect can be obtained.

This consequence indicates that since the parallelism that a DSP algorithm includes is generally not so much, the optimization using the fine-grain parallelism on a conventional high-level synthesis system results in the 50% effect at its maximum. Therefore this evaluation also indicates the importance of the distributed synthesis.

5.3 Evaluation on hardware resources

The number of the functional units required per processor each is depicted in Figure 8, where the MUL, the DIV, and the additional F shows a multiplier, a divider, and for floating-point operations respectively. In this evaluation, although both an addition and a subtraction are allocated to an ALU, an arbitrary operation can be allocated to an arbitrary functional unit by changing the allocation information.

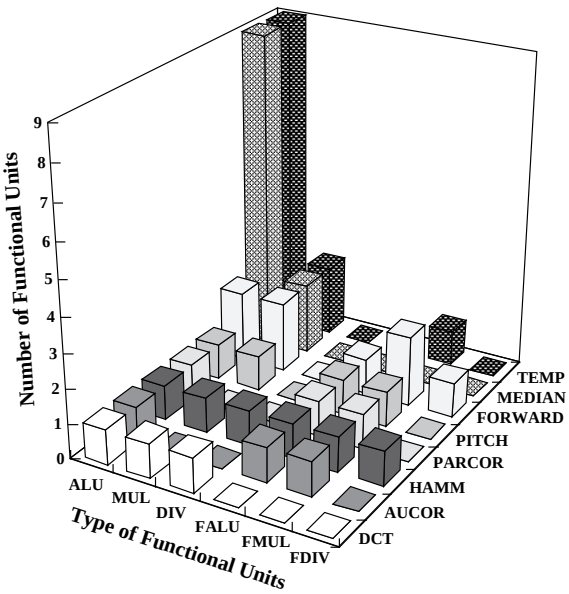

Figure 8: Number of functional units required per processor each

Consequently although the number of each functional

unit required per processor each is approximately 1 or 2, 9 ALUs are necessary for the 2 image algorithm, i.e. the median filter (MEDIAN) and the template matching (TEMP). Because the ASAP scheduling is exploited as a scheduling strategy, 9 ALUs are necessary so that the operations for 9 pixels may be executed concurrently.

The number of registers required per processor each is shown in Figure 9. Since integer variables and floating-point variables are allocated into different registers respectively, the median filter (MEDIAN), the template matching (TEMP), the discrete cosine transformation (DCT), and the HMM speech recognition (FORWARD) requires a large quantity of integer registers. Because the global variables, which are generated in great quantities on the algorithm description, are simply allocated into registers in this implementation, the number of registers increases. The advanced global register allocation is needed in order to solve the problem, which should be considered in the future.

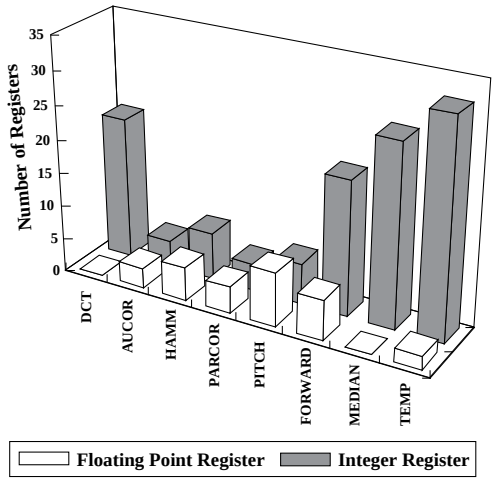

Figure 9: Number of registers required per processor each

6 Conclusions

In this paper, we introduce the new design method, namely the distributed synthesis, on the design system SYARDS, and design a variety of DSP algorithms, and then evaluate the effect of the distributed synthesis and the processor optimization. This evaluation indicates that the newly introduced distributed synthesis effectively works on the DSP applications which requires real-time processing, and that the system can work faster by the conventional optimization.

We also indicate that the design system, which can support the structural design of the system by choosing an appropriate architecture from a variety of template architectures, are necessary, and that the distributed synthesis is also effective as one of the template architectures.

References

[1] Hironobu Kitabatake and Katsuhiko Shirai: "Functional Design of a Special Purpose Processor Based on High Level Specification Description," IEICE Trans. FUNDAMENTALS, vol.E75-A, pp.1182-1190, no.10 (1992-10).

[2] Toshiyuki Takezawa and Katsuhiko Shirai: "Interactive Design Environment of VLSI Architecture for Digital Signal Processing," Proc. of ICASSP-90, pp.1085-1088 (1990-4)

[3] Katsuhiko Shirai and Toshiyuki Takezawa: "Development and Evaluation of Architecture Design System for Application Specific Integrated Circuits," International Journal of Computer Aided VLSI Design, Vol.1, pp.391-414 (1989-3).

[4] Rajesh K. Gupta and Giovanni De Micheli: "Hardware-Software Cosynthesis for Digital Systems," IEEE Design & Test of Computers, vol.10, no.3 (1993-9).

[5] Rolf Ernst, Joerg Henkel and Thomas Benner: "Hardware-Software Cosynthesis for Microcontrollers," IEEE Design & Test of Computers, vol.10, no.4 (1993-12).

[6] Donald E. Thomas, Jay K. Adams and Herman Schmit: "A Model and Methodology for Hardware-Software Codesign," IEEE Design & Test of Computers, vol.10, no.3 (1993-9).

[7] Asawaree Kalavade and Edward A. Lee: "A Hardware-Software Codesign Methodology for DSP Applications," IEEE Design & Test of Computers, vol.10, no.3 (1993-9).

[8] Steven Brawer: "Introduction to Parallel Programming," Academic Press, Inc. (1989).

[9] Daniel D. Gajski, Nikil D. Dutt, Allen C-H Wu, and Steve Y-L Lin: "HIGH-LEVEL SYNTHESIS," Kluwer Academic Publishers (1992).

[10] X.D.Huang, Y.Ariki, M.A.Jack: "Hidden Markov Models for Speech Recognition," Edinburg University Press (1990).