A Nested Loop Pipelining in C Descriptions for System LSI Design

Masahiro Nambu, Takashi Kambe†, Shuji Tsukiyama‡
Graduate School of Science and Engineering, Kinki University
3-4-1 Kowakae, Higashi-Osaka City, Osaka, Japan
†Depart. of Electric and Electronic Engineering, Kinki University
‡Faculty of Science and Engineering, Chuo University, Tokyo, Japan

Abstract— Behavioral synthesis from C language is now a key technology of system LSI design. Since large streaming data are usually processed by nested loops in behavioral description of system LSI, it is important to synthesize a circuit which can process such data efficiently. Nested loop pipelining is a useful implementation technique of the description to synthesize a circuit such that both computational throughput and hardware utilization are maximized. In this paper, we propose an algorithm for nested loop pipelining, which can produce pipeline stages with different processing times. We show two practical experimental results in order to demonstrate the performance of the proposed algorithm.

I. Introduction

High level design methodologies are becoming more and more important in the design of large system LSI devices, and various tools for behavioral synthesis from C or other high level language have been proposed [1]. Such a behavioral synthesis is a key to achieving the productivity required in system LSI designs.

In system LSI applications, streaming data are processed by circuits generated from loops described in a high level language, and processing time of streaming data is usually dominant in the total time. Therefore, loops are the most critical portions of high level hardware description for synthesizing an efficient system LSI. The loop pipelining is one of important techniques to accelerate streaming data processing, but traditional loop pipelining is applied only to the innermost loop, and large streaming data are usually processed by nested loops.

Some researchers have proposed several methods such as loop unrolling and loop transformations for nested loops pipelining (for example, see [2, 3, 4]). But, they cannot produce efficient solutions, if there are data dependences that cross multiple iterations in nested loops. Moreover, the number of cycles in a pipeline stage must be constant.

In this paper, we propose an automatic pipelining algorithm for nested loops with data dependence, and show two applications in order to see how the performance can be improved by the proposed algorithm. The algorithm is implemented on Bach system [5] which is a C-based architecture design tool. Namely, the proposed algorithm accepts a hardware description written in Bach C language, and analyzes the nested loops in the description, then rewrites it so that Bach system can synthesize a circuit with appropriate pipelines.

In order to explain what is done by the proposed algorithm, let us consider an example shown in Fig. 1 such that an outer loop repeats three inner loops 100 times and each innermost loop has 8 iterations. Moreover, let us assume that the process of a single iteration of each inner loop is executed in 10 cycles and there is no dependence among data. Namely, no data is written at the k-th cycle and is read at the h-th ($h < k < 11$) cycle in the next iteration. Then, the total cycles of each inner loop can be reduced from 80 to 17 by pipelining as shown in Fig. 1. However, if pipelining cannot be applied to the outer loop, the total processing time cannot be reduced from 5,100 cycles.

If there is no data dependence between inner loops, we can consider a pipelined process as shown in Fig. 2 where each stage consists of a thread containing an inner loop. Then, the total processing time can be reduced to 1,734 cycles.

In this paper, we propose an algorithm to rewrite a given hardware description so as to synthesize a pipelined circuit shown in Fig. 2. The algorithm can treat an inner loop without destroying the loop description, and produces an efficient pipelined circuit with small additional circuit for not only the inner loops but also the outer loop by using BACH system.

II. Algorithm

Data dependence is important for loop pipelining. There are two kinds of data dependences in a loop. One data dependence is called *flow dependence*, which occurs for write and read operations in the process executed in a single iteration of a loop. In Fig. 3, variable A gets a value in the beginning of the process of an iteration

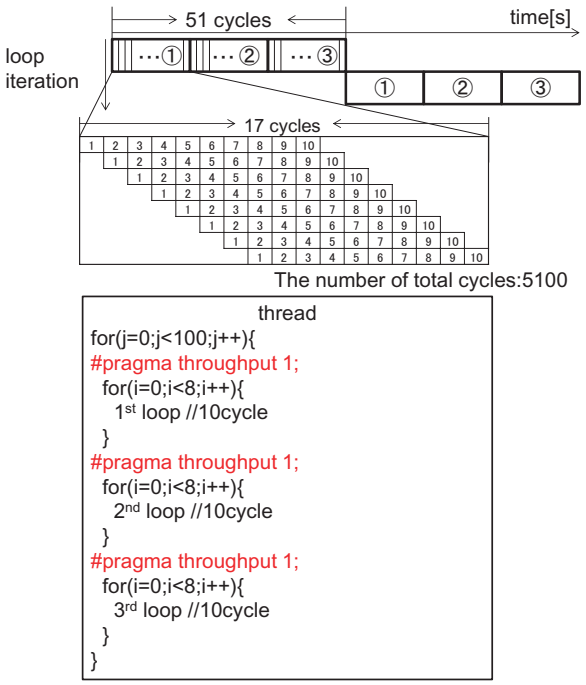

Fig. 1. pipelining for inner loop

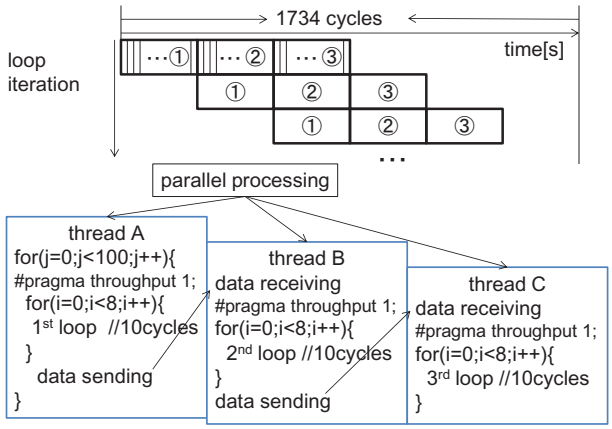

Fig. 2. pipelining for outer loop

```
loop {
  . . .
  A = G + C;
  . . .
  . . .
  G = A + D;
  . . .
}
```

Fig. 3. data dependences

and the value is read afterward in the same iteration, and hence this is a flow dependence. On the other hand, variable G gets a value at the end of the process in the loop and it is read in the beginning of the next iteration. This data dependence is called *iteration dependence*.

The data consistency for flow dependence cannot be kept, if write and read operations are executed simultaneously. But, it can be kept by sending and receiving consistent data between pipeline stages as shown in Fig. 2.

However, in the case of iteration dependence, the read/write operation in the next loop must wait until finishing a previous loop, and hence data communication between stages is not suitable for keeping data consistency. In such a case, we set two inner loops with iteration dependence to single thread, and make a pipeline such that pipeline stages have different processing times, as shown in Fig. 4.

The outline of the proposed algorithm is shown in Fig. 5, which consists of six steps. First, a given Bach C description is synthesized, and the overall loop structure is extracted from the control data flow graph (CDFG). Next, loop pipelining called *pipelining of innermost loops* is executed for every innermost loop, which is described below. Then, pipelining called *pipelining of outer loops* is executed from inner loop to outer loop repeatedly, which is also described below. Finally, Bach C description is updated, and a pipelined circuit at RT level is synthesized by Bach system.

A. *Pipelining of innermost loops*

In the pipelining of innermost loops, the following steps are conducted for each innermost loop which has no external memory access, and selects innermost loops to which *throughput pragma* of Bach system is applied for generating a pipelined circuit. The throughput pragma of Bach system can generate a pipelined circuit with a specified throughput automatically.

1. With the use of Eq.(1), estimate the processing time Ti_p of the pipelined circuit obtained from the loop by loop pipelining, where $Count$ is the number of iterations in the loop, $Nstage$ is the number of stages in the pipeline, and Tp is the number of cycles in a stage.

$$Ti_p = (Count + Nstage - 1) \times Tp \qquad (1)$$

If there is no multi-cycle operation in the loop, then Tp is set to 1, otherwise Tp is set to the maximum number of cycles of a multi-cycle operation in the loop. $Nstage$ is calculated by Eq.(2), where TL is the total number of cycles in a single iteration of the loop. Namely, $Nstage$ is set to the largest value,

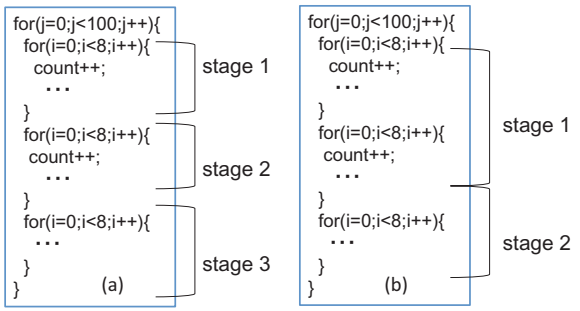

Fig. 4. Example of iteration dependence

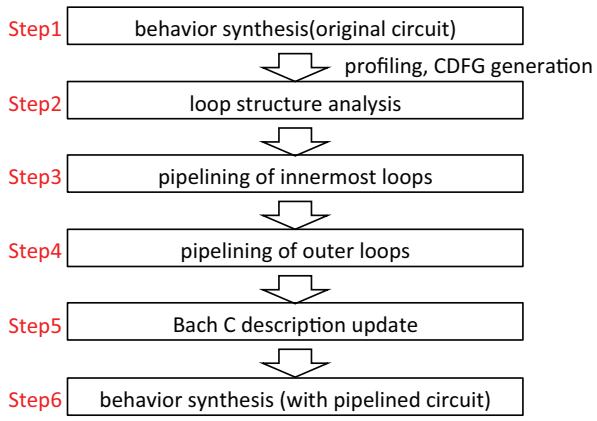

Fig. 5. Outline of the proposed algorithm

since the larger $Nstage$ is, the faster the pipelined circuit is.

$$Nstage = \lceil TL/Tp \rceil \tag{2}$$

2. If Ti_p is less than $Count \times TL$, then apply throughput pragma of Bach system to this loop.

3. The processing time of this loop is updated to Ti_p for use in pipelining of outer loop.

B. *Pipelining of outer loops*

The loop pipelining of outer loops is a *functional level pipelining* (or *thread based pipelining*). Namely, each pipeline stage executing a thread by a functional module starts its process when control signal *start* is received, and sends the control signal to the next stage when the process is completed. The functional level pipelining permits stages to have different processing times, while each stage of the conventional pipeline must have the same processing time. Moreover, the loop pipelining of outer loops is hierarchical, that is, it conducts the following steps repeatedly from inner loop to the outermost loop level by level. Therefore, when the following steps are applied to an outer loop, each inner loop contained in the outer loop is treated as a *block* whose processing time is known, and the process of a single iteration of the loop is a sequence of blocks.

1. Compute the total processing time (the number of cycles) TL of the process in a single iteration of the loop, and find the largest processing time Tp required by a block in the loop.

2. Divide blocks in the loop into threads in such a way that the processing time of a thread is not greater than Tp.

3. Repeat the following process until no iteration dependence is found among stages. For two stages A and B between which an iteration dependence[6] is found, set these stages together to a single stage AB, and update Tp.

4. If a flow dependence[6] is found between stages, then concatenate all flow-dependent data between these stages in order to send it by one cycle.

5. With the use of Eq.(3), estimate the processing time To_p of the pipelined circuit obtained from the loop by loop pipelining, where Cs is the number of stages in the pipeline and calculated by an equation similar to Eq.(2). Ts is the average time of one data communication between two stages and its synchronization. In the equation, Ts is the average time required for each data communication between two stages and for its synchronization.

$$To_p = (Count + Cs - 1) \times (Tp + Ts) \tag{3}$$

6. If To_p is less than $Count \times TL$, then mark the loop so that loop pipelining is applied to the loop, and update the processing time of this loop to To_p. The update of the description is conducted in the next step (see Fig. 5).

III. Experimental Results

The proposed algorithm is applied to two hardware descriptions; one is Discrete Cosine Transform (DCT) circuit for JPEG encoder and the other is Particle Mask Correlation (PMC) circuit for particle tracking system. The updated descriptions were transformed to gate level logic by using Synopsys Design Compiler, and mapped to Hitachi 0.18 μm CMOS library cells. The clock frequency of the circuits was 100MHz, and the external memory access time is assumed to be 50ns.

TABLE I
PIPELINING OF INNERMOST LOOPS

circuit		before	after	ratio
DCT	time[μs]	19,209	16,941	0.88
	area[gate]	254,450	387,057	1.52
PMC	time[ms]	97,156	97,156	1.00
	area[gate]	22,209	22,209	1.00

TABLE II
PIPELINING OF OUTER LOOPS

circuit		before	after	ratio
DCT	time[μs]	16,941	8,322	0.49
	area[gate]	387,060	492,776	1.27
PMC	time[ms]	97,156	81,325	0.84
	area[gate]	22,209	30,080	1.35

A. *Results of pipelining of innermost loops*

The DCT circuit has a two level nested loop structure, that is, an outer loop with 300 iterations contains fourteen innermost loops. Among these innermost loops, two loops have external memory access, and hence the loop pipelining is not applied to these loops. Each of remaining twelve innermost loops has 8 iterations such that single iteration is executed in 10 cycles, and has no multi-cycle instruction. Therefore, the number Tp of cycles in a stage becomes one, and each loop can be executed in 17 cycles by loop pipelining. As shown in Table I, we can obtain a circuit 12% faster than the original circuit, although the area becomes 52% larger, since no resource sharing is conducted. On the other hand, every innermost loop of PMC circuit has external memory access, and hence the loop pipelining is not applied to any innermost loops.

B. *Results of pipelining of outer loops*

The pipelining of the top level loop of DCT circuit generated three stages, whose cycles were 2752, 204, and 2688, respectively. The total number of cycles of the pipelined circuit estimated by Eq.(3) was 831,708, and that of the DCT circuit obtained by the pipelining of innermost loops only was 1,694,101. These values are almost same as those appeared in row "time"in Table II. The differences of these values were produced, because the number of cycles except for loops is not taken into account in the estimation.

The pipelining of outer loops of PMC circuit generated two stages in second level of hierarchy, whose cycles were 462 and 609, respectively. As the number of iterations of the top level loop is 2010, the total number of cycles of the pipelined circuit estimated by Eq.(3) was 7,803,229,187, and that of the original PMC circuit was 9,715,649,557. These values are almost same as those appeared in Table II. The difference of the values come from the same reason as in DCT circuit.

IV. CONCLUSION

In this paper, we proposed a loop pipelining algorithm for nested loops, and demonstrated its effectiveness by two examples. We could produce DCT circuit about 50% faster than the pipelined circuit obtained only by pipelining of innermost loops, and the area increase was 27%. In the case of PMC circuit, we could produce about 16% faster circuit than the original, but since registers for keeping data consistency was introduced and no resource sharing was conducted, the area became 35% larger.

We continue the research to construct a useful high level synthesis tool, and will support a nested pipelining with memory access to get higher performance.

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] S M Logesh, et al. :"A Survey of High-Level Synthesis Techniques for Area, Delay and Power Optimization," International Journal of Computer Applications 32(10):1-6, October 2011.

[2] Petkov, D., R. Harr, et al.: "Efficient pipelining of nested loops: unroll-and-squash, " Parallel and Distributed Processing Symposium., Proceedings International, IPDPS 2002.

[3] Turkington, K., G. A. Constantinides, et al. : "Outer Loop Pipelining for Application Specific Datapaths in FPGAs, " Very Large Scale Integration (VLSI) Systems, IEEE Transactions on 16(10): 1268-1280, 2008.

[4] Morvan, A., S. Derrien, et al. : "Efficient nested loop pipelining in high level synthesis using polyhedral bubble insertion, " Field-Programmable Technology

[5] K.Okada, et al. : "Hardware Algorithm Optimization Using Bach C," IEICE Trans. Fundamentals vol.E85-A, No.4, pp835-841, 2002.

[6] I. Nakata: "Compiler architecture and its optimization," (in Japanese), Asakura book, 1999.

[7] K.Jyoko, T.Ohguchi, et al. : "C-based Design of a Particle Tracking System," The proceeding of SASIMI2006, pp.92-96, 2006.

[8] A. Eguchi, et al. : "An Application Specific Circuits Design for a LVCSR System," The proceeding of ECCTD09, pp.790-793, 2009.