Control Synthesis of For-Loops in a Pipeline System

Heung Sun Yoon, Jae Young Park, and Jong Tae Kim
Department of electrical and computer engineering,
Sungkyunkwan University, Suwon, Gyeonggi-Do, South Korea

Abstract - Pipelining has been a basic technology for high performance digital system, but handling loops in pipeline system is very difficult problem to be solved. In this paper we discuss how to model nested loops in pipeline scheduling and perform automated control synthesis. In our work loops are unrolled partially and treated as conditional branches. We propose a global controller that consists of an FSM controller for each cluster and the FSM activation control part. And we also discuss how to generate control specifications for each FSM.

Keywords: Control synthesis, Loop, Pipeline architecture

1 Introduction

Pipelining is the key implementation technique used to make fast digital systems for many DSP algorithms. In pipelining, each input computation task (e.g. an instruction) is subdivided into a sequence of subtasks and each of these subtasks is executed during the clock cycle by a specialized hardware stage that operates concurrently with other stages in the pipeline. Every clock cycle has the same time period. Successive tasks are initiated at some fixed or variable intervals, which are integer multiples of a clock cycle and are bounded by the execution time of a task. In this fashion, execution of subtasks of consecutive tasks may overlap in time on different parts of the pipeline circuits. If a task is initiated every clock cycle, i.e., the initiation interval is 1, this is the fastest pipeline design and there is no resource sharing between the executions of subtasks. However, if the design is constrained on the cost, i.e., there are not enough functional modules to be allocated, some resources must be shared by more than one subtask. Since operators are shared between stages of the pipe and within a stage, handling loop in pipeline system is very difficult problem to be solved. There are several loop pipelining algorithms used in high level synthesis or optimizing compiler [1][2][3]. The loop folding is to achieve pipelining effect by initiating the next iteration of a loop before the current iteration is finished. Thus this transformation achieves speedups. However, if there are data dependency between successive loop iterations, i.e., the data produced at the last operation of the current iteration is consumed at the first operation of the next iteration, the loop folding has no advantages. For-loop is deterministic loop construct which uses an index variable. This index specifies the range of loop iteration and is assumed to be known prior to entering the loop body. Loops are either unrolled completely or treated as conditional branches. Figure 1 shows a general model for a nested loop. The nested loops are numbered 1 through n from the inner most nested loop to the outer most one. The number of time steps in a loop body i can be calculated using the following recursive formula.

\[
TS^i_1 = s^i_1 \times k^i_1. \\
TS^i_n = (b^i_n + TS^i_{n-1} + a^i_n) \times k^i_n.
\]

where \(s^i_1\) is the size of the inner most loop, \(k^i_n\) denotes the number of iterations for the nth nested loop, \(b^i_n\) and \(a^i_n\) are the number of time steps before, and after the \((n - 1)\)th nested loop. \(TS^i_n\) denotes the number of time steps in the loop upto the nth nested loop from the inner most nested loop and \(TS^i_n\) means the number of time steps for the inner most nested loops of the loop body i.

Figure 1 A model of nested loop body
Let’s also assume that there are \( m \) different loop bodies in a DFG and each loop body \( i \) is nested \( n_{i} \) times as shown in Figure 2, where \( i = 1, 2, \ldots, m \). The total number of time steps \( P \) for the input CDFG is given by  
\[
P = \sum_{i=1}^{m} T_{S_{i}} + \sum_{j=2}^{n_{i}} p_{j},
\]
where \( T_{S_{i}} \) is the total number of the time steps in the \( i \)th loop body which has \( n_{i} \) nested loops, \( p_{j} \) is the number of time steps between loop bodies, \( p_{1} \) is the time steps before the first loop body, and \( p_{m+1} \) is the number of time steps after the last loop body.

In our model loop capacity \( \text{LC}_{i} = \lceil T_{S_{i}} / L \rceil \), where \( T_{S_{i}} \) is the number of time steps in the inner most nested loop of the loop body \( i \). The loop capacity \( \text{LC}_{i} \) is defined as the maximum number of different jobs that can be executed concurrently by a loop body.

## 2 Pipeline Scheduling

If loop unrolling is not practical, we schedule the DFG as follows. First, the input DFG is partitioned with the intention of clustering the loops separately from the rest of the DFG. For example, in Figure 3, this method would result in 3 clusters: Cluster 1 (nodes before the loop body), Cluster 2 (the loop body), and Cluster 3 (the nodes after the loop body).

Loops are detected easily during syntax analysis of structured languages so that the CDFG can be marked accordingly. Loops are transformed into a DAG by breaking the feedback edge and adding the node \( v_{L} \) after the last operation of the loop body as in the Figure 3(a). Next, each cluster is scheduled separately. The performance depends on the loop body \( i \) which has the minimum time steps of the inner most loop.

## 3 Control Synthesis

### 3.1 Control Specification

In this section we discuss how to synthesize control specifications for FSMs. As explained in the previous section, the input DFG is clustered so that loops are isolated from the rest. Each of the resulting clusters is scheduled independently and the controller for each cluster is implemented as a separate FSM. Initially each FSM is in a wait state until the loop handler generates control signals to activate the FSMs. The Global controller consists of an FSM controller for each cluster and the FSM activation control part. We describe how to construct these components later in this section. Figure 3.11 shows the global controller of the CDFG shown in Figure 3.9.

There are 3 clusters which are the set of operations before, within, and after the loop body. There are 3 corresponding FSMs: FSM 1, 2, and 3 control cluster 1, 2, and 3, respectively.

### 3.2 Synthesizing the FSMs

The outline of the process to generate control specifications for each FSM is as follows:

1. Find MESs and PEMs for each time step.
2. Attain the patterns of the overlapping time steps for each FSM from the scheduling and allocation phases.
3. Decide the states for each pattern.
4. For each FSM determine the state transition.

![Figure 2 A generalized DFG with loops](image-url)
received, it will then be activated at time step 1, and goes back to the wait state at time step 4.

3.3 FSM Activation Control

FSM activation control provides the initiation, loop enter, loop exit control signals. A simple way to construct the initiation controller is by using an $Lt$-bit vector whose $i$th bit is 1 if a new initiation is started at the $i$th time unit of each cycle. For example for the cycle $L_t = 10, L = 2$, and $LC = 3$, the 10-bit initiation vector is 1010100000. Then a shift register controller takes this vector and cyclically rotates it once every clock cycle. If at the start of any clock the leftmost bit is 1, a new initiation signal is sent to the necessary FSM. This is very simple method, but if $Lt$ is very large (especially if loops are involved) this is not practical. Instead of using this method we can build the initiation controller using a counter and the supporting combinational circuits. Counter counts up to $Lt$ and is reset at $Lt$. The combinational circuit provides the initiation control and reset signals.

4 Conclusions

Pipelining has been a good methodology for designing fast digital system, but pipeline architectures become quite complex to design if loops are involved. Automated controller design tools in high level for pipeline system are necessary to cope with such complexity and explore the design space efficiently. In this paper we discuss how to model nested loops in pipeline scheduling and how to synthesize of controllers for pipelined data paths. We use the general nested loop model and loops are unrolled partially and treated as conditional branches in our work. We propose a global controller that consists of an FSM controller for each cluster and the FSM activation control part. And we also discuss how to generate control specifications for each FSM.

5 References

