Delay and Sampling-Rate Aware Architectural Synthesis in Presence of Communication Overhead
A. Itradat, Student member, IEEE, M.O. Ahmad, Fellow, IEEE, and Ali Shatnawi
Department of Electrical and Computer Engineering, Concordia University, Montreal, Quebec, Canada
Emails :{a_itrada, omair}@ece.concordia.ca

Abstract—This paper presents a technique for the minimization of the delay and sampling rate in the architectural synthesis of cyclic data flow graphs (DFGs) representing DSP algorithms taking into consideration the interconnect communication delays. In this paper, the question of optimizing the I/O delay without sacrificing the iteration period (sampling rate) with non-negligible interconnects communication overhead is addressed. The proposed technique operating on the cyclic DFG of a DSP algorithm is designed to evaluate the relative firing times of the nodes by using Floyd-Warshall’s longest path algorithm so that the communication overhead is taken into consideration to provide an optimized time and processor schedule. Moreover, the proposed scheme is applied to well-know DSP benchmarks and the obtained synthesis results reveal that the proposed scheme is efficient in minimizing the I/O delay without sacrificing the sampling rate.

I. INTRODUCTION
The problem of architectural synthesis for iterative algorithms has received a great deal of attention in recent years. However, most of the techniques for multiprocessor scheduling dealing with simplified problem in which the time to communicate data from one processor to another (interconnect delay) is not taken into consideration [1], thus eventually leading to unrealistic schedules. Well-known examples of such techniques are a cyclo-static scheduling method that uses exhaustive search [2] and the optimum unfolding technique [3]. These methods have one thing in common in that they have not considered the interconnect communication delays (ICD). Curtis and Madisetti [4] have shown that inclusion of the ICDs is essential in a realistic development of multiprocessor schedules. Their objective was to use realistic structural and behavioral level description of a DSP algorithm that takes into consideration the ICDs to find simultaneously rate optimal and processor optimal schedules. They developed the so called DSMP-C1 method for this purpose. However, this method is computationally intensive, practical only for small DSP algorithms, and suitable for homogeneous multiprocessor systems. The techniques proposed in [5,6] use integer linear programming (ILP) to consider the inter-processor communication delay during the scheduling of the DSP applications mapped onto pre-defined homogeneous multiprocessor systems with different topologies ranging from weak connected to strongly connected topologies. It is well-known that ILP is a very time consuming technique for large size problems. Moreover, the problem of optimizing I/O delay (i.e. the delay between consuming the input sample(s) and producing the output sample that belongs to the same iteration) has not been addressed. Despite the fact that the techniques in [4-6] can produce good results in terms of throughput, the I/O delay has not been considered.

With the advances in ASIC synthesis techniques, the idea of using heterogeneous multiprocessor systems for the implementation of DSP algorithms is gaining a widespread usage because of the area efficiency of such systems. Scheduling iterative signal processing algorithms onto such heterogeneous multiprocessor systems imposes even a greater need for considering the ICDs of the target architecture. A scheduling algorithm for heterogeneous multiprocessor implementation taking ICDs into consideration has been proposed in [7]. However, the main impediment with this algorithm is the requirement of a large memory space. A real-time scheduling algorithm called RT-SDA for heterogeneous multiprocessor systems can be found in [8]. This algorithm takes communication delays of a multiprocessor system into account by employing the so called task duplication. This algorithm is suitable for acyclic data flow graphs, requiring an effort to convert the cyclic data flow graphs into acyclic ones.

In the proposed technique, the critical paths and circuits are identified and the I/O delay-optimality is achieved by introducing a path from the output node to the input node whose length is equal to the negative of critical path delay bound. Introducing such a path results in a new critical circuit in which no flexibility is available for the scheduling of the entire nodes and hence their time schedule is fixed. The interconnect communication delay between a pair of nodes of different type and not belonging to any of the critical I/O paths are treated as a non-computing nodes, whereas those between a pair of nodes of same type is taken into account by re-adjusting the firing times of the appropriate nodes of DFG. The proposed technique operating on the a cyclic DFG of a DSP algorithm are developed to evaluate the relative firing times of the nodes in the DFG by using the Floyd-Warshall’s shortest path algorithm [9].

II. DATA FLOW GRAPH MODEL AND OPTIMALITY CRITERIA
The data flow graph (DFG) is proven to be an efficient representation of the system specification due to its ability to expose the hidden concurrency between the operations of the underlying algorithm. Since DSP applications are known for their inherent parallelism, the DFG model is thus suitable for the behavioral representation of DSP applications. A graph G can be represented by the pair (V, E), where V is a set of nodes, and E is a set of elements called edges. Each edge is associated with a pair of nodes. The symbols v1,v2,...,vn−1, vn are used to represent the nodes, and the symbols e1,e2,..., are used to represent the edges of a graph.

A direct path Pk(v0v1) is a finite sequence of distinct nodes v0,v1,...,vk and distinct edges such that the edge (v1,v2) is present in the path Pk(v0v1). If v0 = vi, then this path is called a directed circuit or loop. Each loop in a DSP graph must contain at least one
ideal delay element for the graph to be computable. The data flow graph that contains at least one directed circuit is called the cyclic graph, otherwise it is acyclic. The path length, \( \text{len}[P] \) of a path \( P \) between the nodes \( v_1 \) and \( v_2 \) is defined as the minimum elapsed time between consuming the input operand(s) at its initial node \( v_1 \) and producing an output at its terminal node \( v_2 \), and it is given as

\[
\text{len}[P] = \sum_{v_j \in \text{path}} d_{v_j} - T \cdot N_P + ICD_P
\]

where \( T \) is the iteration period, \( N_P \) the total number of ideal delays in a path \( P \), and \( ICD_P = \sum_{c \in \text{path}} d_c \), \( d_c \) being the delay of the communication node \( c \) in \( P \). In order to illustrate the computation of a path length, consider the example shown in Fig. 1. Assume that the iteration period \( T \) is \( 6 \) time units, the computational delay for the nodes \( v_1 \), \( v_2 \) and \( v_3 \) are \( 1 \), \( 2 \), and \( 1 \), respectively, and the delay of the communication node (dummy node) \( c \) is \( 1 \). Then, \( \text{len}[P_{v_1v_3}] = (1+2+1) - (1)(6) + 1 = -1 \).

A. Optimality Criteria

1. Sampling-Rate-optimal: The iteration period (sampling rate) bound is the minimum time between producing or consuming successive output or input streams. For a cyclic DFG, it is given by

\[
T_0 = \max_{v_c \in \text{circuits}} \left[ \frac{D_C}{N_C} \right]
\]

where \( D_C = \sum_{v_c \in \text{circuit}} d_{v_c} \), \( d_{v_c} \) being the computational delay of node \( v_c \) in \( C \), and \( N_C \) is the total number of ideal delays in circuit \( C \). When the interconnect communication overhead is non-negligible, then the iteration period bound is given by

\[
T_{0_{\text{comm}}} = \max_{v_c \in \text{circuits}} \left[ \frac{D_C + ICD_C}{N_C} \right] + 2(N_C - 1)
\]

where \( ICD_C = \sum_{c \in \bar{E}(C)} d_c \), \( d_c \) being the delay of communication node \( c \) in \( C \). The second term represents the least number of interprocessor communication channels required for the scheduling of the operations on a circuit.

2. I/O Delay-optimal: The I/O delay bound is the minimum delay between consuming an input sample and producing the corresponding output sample that belongs to the same iteration. In the present of the interconnect communication overhead, this bound is computed based on the operation on the critical I/O path and the iteration period bound \( T_{0_{\text{comm}}} \) and is given by

\[
L_0 = \max_{\text{path} \in \text{I/O Path}} \text{len}[P] + \left( \sum_{v_j \in \text{I/O path}} d_{v_j} - T_{0_{\text{comm}}} \right) + 1
\]

the second term represents the least number of processing units required for the scheduling of the nodes in the critical I/O path.

III. I/O DELAY AND RATE-AWARE SCHEDULE

It has been shown in [10] that the schedule might not exist if both the iteration period and the number of processors are fixed, which means that processor and sampling-rate optimality may not, in general, be achieved at the same time. Further, we will show that the I/O delay-optimality is achievable for any value of \( T \), thus making it possible to attain both rate- and delay optimality.

A non-critical circuit in a DFG has a positive slack time for scheduling. In the time schedule, the slack time represents more delay between adjacent nodes than what is implied by the data dependency determined by the governing graph. Let us refer to this extra delay which results between nodes \( v \) and \( w \) as the shimming delay of the edge \((v,w)\). Obviously the total shimming delay of all the edges of a given circuit is equal to its slack time.

THEOREM 1 A DFG can always be scheduled delay optimally for an iteration period \( T_{0_{\text{comm}}} \).

Proving this theorem is equivalent to establishing that scheduling can be carried out such that the shimming delays assigned to edges of a non-critical circuit do not make the length of any I/O path greater than the delay bound \( L_0 \), or equivalently, that the time schedule can be built such that none of the nodes of the I/O critical paths uses the scheduling flexibility (mobility) provided by the shimming delays.

A simple technique to ensure the delay optimality will now be presented. The delay optimality is achieved if and only if the output node is scheduled to finish execution exactly after \( L_0 \) time units of the firing time of the input node. The scheduling of the output node, in this case, is equivalent to the scheduling assuming that there is a critical circuit containing the critical I/O path. If such a circuit does not exist we can alter the given DFG to introduce such a circuit by adding a new edge from the output node to the input node with a total delay of \(-L_0\).

The implementation of this modification is quite straightforward. Before proceeding further, let us first define certain terms that are used later to build the time schedule. Let \( Q^0 \) be an \( N \times N \) matrix in which \( Q^0_{ij} = d_{ij} + d_{i} - T_{0_{\text{comm}}} \cdot N_e \) for all \( e=(v_i,v_j) \in E(G) \), where \( N_e \) is the number of ideal delays associated with edge \( e \), and \( d_{i} \) is the delay of the communication node \( c \) of edge \( e=(v_i,v_j) \). All other entries are \(-\infty \). Next, applying Floyd-Warshall's longest path algorithm to \( Q^0 \) results in a matrix \( Q^f \), where \( Q^f_{ij} \) could be finite or infinite. An infinite value implies that there is no directed path connecting node \( v_i \) to node \( v_j \) and a finite value of \( Q^f_{ij} \) represents the longest distance from node \( v_i \) to node \( v_j \). The finite entries of this matrix are given by

\[
Q^f_{ij} = \max_{k \in \text{path}} (\text{len}[P_{ij}])
\]

After the matrix \( Q^f \) is computed the entry \( Q^f_{ij} \) is altered as \( Q^f_{ij} = -Q^f_{ij} \), where \( I \) and \( O \) are the indices of the input and the output nodes respectively. In this case, the input node is chosen as a reference node to be scheduled first, and hence the firing time of the output node is fixed relative to that of the input node.

In the proposed scheme, identifying the set of nodes in each critical or sub-critical circuit and in each critical I/O path is very necessary during the allocation of nodes to processors. Based on this

\[
d_{i} = 1 \quad d_{j} = 1 \quad d_{v_2} = 2
\]

Fig. 1 An example for computing the path length.
identification process, the processor allocation algorithm can easily allocate the nodes of each critical or near-critical loop to the same processor resulting in zero the interconnect communication delay. Another benefit of this loop identification arises if the entire set of nodes in a loop cannot be assigned to the same processor then such nodes can be marked for their urgency with respect to ICD and then their firing times can be, accordingly, re-adjusted.

A. Identifying the critical loops and paths

After the Floyd-Warshall's technique is applied, the final matrix \( Q^f \) can be used to identify the longest loops (critical and near-critical) for each node. The nodes with the largest diagonal entry are belonging to critical loop(s). The near-critical loop contains a node not on any critical loop(s) represent the shortest distance from the node to itself. The nodes of identical diagonal entries in \( Q^f \) may not form a complete loop or they may even belong to different loops, thus we need to find other nodes on the same near-critical loop. In general, a node may be on more than one loop such that there is more than one path from the node to itself. Among all the loops from the node to itself, there is at least one with the longest distance. This longest loop through the node not on the critical loop is a near-critical loop. It should be noted that there may be more than one near-critical loop of the same distance. More details about identifying near-critical loops are given in Algorithm 1. A near-critical loop contains nodes with identical diagonal entries, but it may contain nodes with other diagonal entry values. Thus, nodes with the same diagonal entry values may not form a complete loop. Further, these nodes may not even be in the same loop. We need to find the nodes on the same near-critical loop. The identification algorithm for all the nodes in each critical or near critical loop involves the following steps.

**Algorithm 1: Identification the set of nodes in each critical and near-critical loop.**

1. Choose any node \( v \) with the largest diagonal entry as target node, set \( i = 0 \) and \( j = 0 \) where \( i \) is criticality level of the loop and \( j \) is the number of the loop within the criticality level \( i \).
2. Add the node to set \( LP_v(i) \), where \( i \) is criticality level of the loop \( LP_j \), \( j \) is the number of the loop within the criticality level \( i \) and \( j \) is the type of the selected node.
3. Among all the next nodes to \( v \), select the node \( k \) satisfying the following two conditions:
   a. Node \( k \) has maximum diagonal entry value, and 
   b. \( Q^f_{jk} = Q^f_{ik} \). In conflicting cases choose any node satisfy the two conditions. Add the selected node to \( LP_v(i) \).
4. If the selected node in step 3 is already chosen in the same iteration, then this selection should be backtracked all the way until finding another node could be selected under the same two conditions mentioned in step 3.
5. Repeat steps 3-4 until node \( v \) is reached and a new critical or near-critical loop containing the node \( v \) is identified, \( j = j + 1 \).
6. Choose any node other than the previous target node with largest diagonal entry. If the diagonal entry of the chosen node is less than the current target node then \( i = i + 1 \).
7. Repeat steps 1-5 until all nodes \( v \) with a finite entry \( Q^f_{iv} \) have been picked, or when diagonal entry value of the next node is less or equal to \( \beta \) (threshold) thus all the important critical or near-critical loops have been identified.

We also use the final matrix \( Q^f \) to identify the longest paths (critical) for each node \( v \) by employing the following condition

\[
Q^f_{IL} + Q^f_{VO} = Q^f_{IO}
\]  

**B. Time and processor schedules**

In a heterogeneous system, nodes of different types are executed on different processing units, whereas two or more nodes of the same type may or may not be executed on different processors. Thus, the ICD between any two nodes, i.e. the nodes having precedence dependency, being executed on two different processors has to be taken into consideration while building an initial time schedule, whereas the ICD between any two nodes being executed on a single processor can be neglected. In the proposed technique, in order to take into consideration the ICD between a pair of nodes of different types (but at least one of them not a member in any critical I/O path) and having a direct edge, the original DFG is first modified by inserting a communication (dummy) node. These dummy nodes are not scheduled; they are taken into consideration while building the initial scheduling of other nodes. However, during the building of the initial time schedule, the ICD between a pair of nodes of the same type and having a direct precedent is not taken into consideration, since at this stage it is not yet determined whether the two nodes in question are assigned to a single or two different processors.

The initial time schedule is then built iteratively based on the node mobilities. In this technique, the earliest and the latest firing times at which each node can be scheduled to fire are iteratively calculated by using the longest path matrix \( Q^f \). The node mobility in a schedule or the range of control steps at which the corresponding node can be scheduled is equal to the difference between its calculated latest and earliest firing times. These latest and earliest firing times are found relative to a reference node and according to the intra and inter iteration precedence constraints. The detailed scheduling algorithm can be found in [11].

In [11], the nodes of the modified DFG are initially assigned onto a heterogeneous multiprocessor without considering the ICDs between the nodes of the same type. The entire nodes of each critical I/O path are assigned to same processing unit. It is to be noted in this paper that in addition to the modification on the processor allocation scheme which is advanced by identifying critical paths and circuits, the original scheduling algorithm is modified to deal with heterogeneous processing element such that Adder, Multiplier, and ALU's as well. The result of this algorithm is a processor allocation matrix, which specifies as to which processor a certain node is assigned. In the next step, this initial time and processor schedules are modified depending on the compatibility of the ICD and the firing times of the two corresponding nodes of the same type.

**IV. Experimental Results**

The proposed scheme is first applied to a well-known benchmark problem of a second-order IIR filter. The modified DFG for this filter is shown in Fig. 2. In the modified DFG, a new path \((Q^f_{II} = -Q^f_{IO} = -4)\) is introduced to ensure the delay optimality. The computational times of the addition and multiplication nodes are assumed to be 1 and 2 cycles, respectively. The target multiprocessor system is assumed to have ICD of 1 cycle. Using the proposed technique, the time schedule and the corresponding processor allocation are shown in Fig. 3. A superscript represents the offset of the iteration to which the corresponding node belongs to with respect to the current iteration. For this benchmark problem, the iteration period bound \(T_0_{comm}\) as given in (2) is equal to 5. It is also seen from Fig. 3 that using the proposed technique, the final iteration period is found to be 5 cycles. Thus, the time schedule is optimal in terms of the iteration period. Furthermore, the I/O delay \(L_0\) as computed using (3) is equal to 4. It is shown in Fig. 3 that nodes of the critical path i.e., 1, 2, 6, and 8 are assigned to the same processor, namely, ADDER. Further, the output node is scheduled to fire at control step 3. Hence, the output stream will be available 4 cycles later to the latching of the input stream.
K. Parhi and D.G. Messerschmitt, “Static rate-optimal scheduling of...

V.K. Madisetti and B.A. Curtis, “A quantitative methodology for rapid...

Fig. 3 The processor allocation matrix

Fig. 4(a) shows the sampling rate obtained for some intensive DSP applications compared to the sampling rate bound. Fig. 4(b) gives I/O delay obtained with and without the incorporation of the proposed architectural synthesis technique when applied to the six DSP benchmark problems. It is seen from Fig. 4(b) that the proposed scheme is efficient in minimizing the I/O delay without sacrificing the iteration period (shown in Fig. 4(a)).

V. CONCLUSION

The optimization of both sampling-rate and I/O delay in the architectural synthesis of DSP cyclic data flow graphs (DFG) mapped onto multiprocessor architectures with non-negligible interconnect overhead has been addressed. The problem of finding a schedule with optimal I/O delay without sacrificing the rate optimality (sampling rate) has been investigated. In the proposed technique, the original DFG representing a DSP algorithm has been modified by inserting communication (dummy) nodes to represent the ICDs between the nodes of different types before building an initial schedule. Moreover, in order to ensure the I/O delay, the original DFG has been altered by adding a new edge, associated with a total delay of \((-L_0\)) connecting the output node to the input node. The modified DFG is then used to build a parallel and processor schedule iteratively based on the mobility of each node. The proposed scheme has been tested by applying it to well-known intensive DSP benchmark problems, giving rate- and I/O delay-optimal schedules in most of the cases.

REFERENCES


