MAXIMUM-THROUGHPUT SCHEDULING WITH LIMITED RESOURCES FOR ITERATIVE DATA-FLOW GRAPHS BY MEANS OF THE SCHEDULING-RANGE CHART

Sonia M. Heemstra de Groot and Otto E. Herrmann

University of Twente, Faculty of Electrical Engineering
Laboratory for Network Theory
P.O.B. 217, 7500 AE Enschede, The Netherlands

Abstract

This paper describes a technique for obtaining maximum-throughput schedules for iterative data-flow graphs, given a limited set of resources. The method is based on the use of the scheduling-range chart, where the scheduling window of each operation is displayed. This scheduling window, or scheduling range, is relative to a reference operation, and can be finite or infinite. The information in the scheduling-range chart is used during scheduling in order to optimize the sampling period. An algorithm based on this approach, for scheduling of acyclic as well as cyclic data-flow graphs, is presented.

1 Introduction

The practical value of a digital signal processing (DSP) algorithm is often dramatically increased when the algorithm is implemented in real time (think e.g. of audio applications, radar, sonar, etc.). Parallel processing is unavoidable for real-time implementations of high-speed and/or complex DSP algorithms. As a consequence, a number of multiprocessor architectures for real-time DSP have been proposed (e.g. [2,10,13]). The speed of the implementation will not only depend on technology aspects but also on the efficiency of the design tools. Among all of them [1], the tools that perform the scheduling and assignment to the hardware resources of the tasks that compose the DSP algorithm, have a crucial role.

For a big number of DSP algorithms, with no data-dependent conditionals, the scheduling process can be performed statically, at compile time [11]. Besides, DSP algorithms are characterized by their infinite repetition, which allows the exploitation of not only the parallelism between operations of the same iteration (*inter-iteration parallelism*), but also the parallelism between operations of different iterations (*intra-iteration parallelism*).

In this paper, the scheduling problem is modeled in a similar way as proposed by Schwartz in [14]. The algorithm specifications are mapped on a *data-flow graph* (DFG), where the

nodes represent tasks or delay elements and the directed arcs represent the precedence relations. Associated to each task T_i there is a processing time, tc_{T_i} . In this way, it is possible to model acyclic as well as cyclic graphs. Figure 1 shows the cyclic DFG of a second-order digital filter section. The DFG together with the set of hardware resources, that for reasons of simplicity will be considered as a set of identical processors, provide the model for the scheduling problem. Interprocessor communication delays are assumed to be neglectable with respect to the computation time. The (processor) scheduling of iterative DFG's consists in finding a suitable periodic ordering of the tasks and their assignment to the set of processors, such that the precedence relationships are not violated, and a certain objective function is optimized. When the main interest is speed, i.e. high sampling rates, the scheduling goal is to minimize the sampling period. Besides, there can be secondary goals, as the minimization of the number of processors or the register life time.

Hierarchy is provided by the DFG model since the nodes can represent operations of any type of complexity, from indivisible (atomic) operations (fine grain) up to complex DFG's (coarse grain). The parallelism present in the DFG increases as the grain size decreases; therefore, the atomic DFG shows the maximum parallelism present in the algorithm. Without loss of generality, the computational nodes in this paper will be assumed atomic and non-preemptive.

In [6,9] a scheduling algorithm is discussed that minimizes

Figure 1: DFG of a second order digital filter section. For reasons of simplicity, the duration of the operations (shown in the rectangle at the right) is given in an abstract time unit, TU.

the resources for optimal sampling rate. Here an algorithm will be presented that optimizes the sampling rate, for a fixed number of resources. The algorithm decisions are guided by the information displayed in the scheduling-range chart.

1.1 Maximum-Throughput Schedules of Iterative DFG's

Those schedules that achieve the maximum speed for a given number of resources will be called *maximum-throughput schedules*. For repetitive DSP algorithms, maximizing speed or sampling rate, consists in minimizing the sampling period T_0 .

In *acyclic DFG's* it is possible to use pipelining in order to increase parallelism, and reduce the sampling period T_0 , until the bound imposed by the hardware constraints of the limited resources. For unlimited resources, the speed bound, theoretically¹ becomes infinite².

The speed bound for *cyclic DFG's* depends not only on the number and computational delay of the hardware resources, but also on the topology of the graph. The minimum sampling period T_{0min} of a cyclic DFG is determined by its *critical loop* [12]. A critical loop is that directed loop for which the summation of the processing delays of each operation in the loop divided by the number of delay elements in the loop, n_l , is maximal. This quotient is T_{0min} .

Expressed in another way,

$$T_{0min} = \max_{\text{all directed loops } l} \left\{ \frac{\sum_{i \in l} tc_{T_i}}{n_l} \right\}, \quad (1)$$

Schedules that can be executed at this lower bound are called *rate optimal*. In the DFG of a second order digital filter section given in Figure 1, the critical loop is the one formed by nodes 4-2-6, and $T_{0min} = 3$ TU's.

2 The Scheduling-Range Chart

Given a reference operation whose position in the time schedule (assignment of the operation to specific, repetitive time-slot(s)) is fixed, it is possible to determine the *relative* allowed positions of the rest of the unscheduled operations. The set of positions where an operation can be scheduled is called the *scheduling range*. The *scheduling-range chart* [6] displays this information for every operation in the DFG. The difference between the length of the scheduling range of an operation and its duration is called its *mobility*. The scheduling range of an operation of an iterative DFG can be finite or infinite. When a limit of the scheduling range

¹Communication delays are not being considered.

²The sampling period can be made shorter than the duration of the longest atomic operation by using direct blocking [8,15], or cyclo-static techniques [14,15].

Figure 2: Notation for the scheduling range of operations.

has a well-defined position, i.e. when all the predecessors or successors of the operation have been assigned to a specific repetitive time slot (time schedule), it is called a *fixed limit*. The notation used in this paper is shown in Figure 2.

To build the scheduling-range chart, an operation is taken as a reference. This operation has no mobility. The scheduling range of the rest of the operations in the graph is determined by (the joined effects of) the following constraints:

- The *forward precedence relation* between operations. An operation cannot start its execution before all its predecessors have finished their execution. See Figure 3(a).
- *Backward precedence relation* between operations related by registers. Operations which are in a common path, and separated by a node representing one or more delay elements should be scheduled such that the value stored in a register is not rewritten before the successor operation(s) has read it. Expressed in another way: let T_i be an operation whose result is written to node D_m , a node representing a number m of delay elements. Let T_j be an operation that reads a value from D_m . Then, T_i and T_j should be scheduled such that the following relation is respected:

$$t_{T_i} + tc_{T_i} \leq t_{T_j} + mT_0 \quad (2)$$

where

t_{T_k} is the time when operation T_k starts its execution, tc_{T_k} is the execution time of task T_k , and T_0 is the iteration period.

This is shown in Figure 3(b).

- *Slack time*.

This is a consequence of the backward precedence relation among operations that belong to the same loop. An operation in (a) loop(s) has a limited mobility (with respect to a reference operation in the loop) equal to the shortest slack time of the loops of which it is a member. The slack time of a loop $\mathcal{L}$ is defined by:

$$\text{slack time} = n_{\mathcal{L}}T_0 - \sum_{T_i \in \mathcal{L}} tc_{T_i} \quad (3)$$

Figure 3: Scheduling range of operations for the DFG of Figure 1, for $T_0 = T_{0min} = 3 TU's$. Forward precedence (a), backward precedence (b), and slack times in loops (c).

where:

n_L is the number of delay elements in $\mathcal{L}$.

See Figure 3(c).

Figure 4 shows the scheduling-range chart corresponding to the DFG of Figure 1, for $T_0 = T_{0min}$. Operation 2 has been taken as reference. Operation 4 is also member of the critical loop and does not have mobility. Operations 1 and 3 are members of the non-critical loop 3-1-2-6-5; their mobility is equal to the slack time of that loop, i.e. 2 TU's. The rest of the operations have infinite scheduling range. As operations 2 and 4 have a fixed position in the chart, i.e. they are time scheduled, the scheduling range has a fixed limit for operations 3, 9, and 10, (whose only predecessor is 2), and 1 (whose only successor is 2).

The properties of the scheduling-range chart are discussed in more detail in [6]. There it is shown that any schedule that satisfies the backward and forward precedence relations, will automatically satisfy the slack time in loops.

3 Maximum-Throughput Scheduling with Limited Resources

When both the sampling period and the number of resources are fixed, as is done in [14], a solution for the scheduling

Figure 4: Scheduling range chart for the DFG of Figure 1, for $T_0 = T_{0min} = 3 TU's$

problem may not exist [6]. In order to guarantee a solution either of the two parameters should be left free to be optimized by the algorithm. In [6,9] a scheduling algorithm is discussed that minimizes the resources for optimal sampling rate. Here an algorithm will be presented that optimizes the sampling rate, for a fixed number of resources. The algorithm decisions are guided by the information displayed in the scheduling-range chart.

Since the schedule needs to be periodic in T_0 , we need to take into account the effect of the overlapping in time of the execution of operations of different iterations. This is performed by folding the execution of one iteration by modulo T_0 in the range $[0, T_0 - 1]$. In this way, the operations are grouped in T_0 equivalence classes [14], [8]. The operations in each equivalence class are executed in parallel. Therefore, for limited resources, the maximum size of any equivalence class cannot exceed the number of processors.

The iterative scheduling problem, as discussed in this paper, is NP-complete [4]. Only an informal sketch of a proof is given here. Since the operations in acyclic DFG's have an infinite scheduling range, the folding of the ranges on the interval $[0, T_0 - 1]$, will remove any constraint in their positioning. So, any instance of the bin-packing problem, which is NP-complete, can be reduced to an instance of the iterative scheduling problem.

3.1 Computational Iteration Period Bound

We define the the *computational iteration period bound*, T_{0b} , a lower bound imposed by the computational delays of the nodes in the DFG and the number of processors, P , (the topology of the graph is not considered) as:

$$T_{0b} = \left\lceil \frac{\sum_{T_j \in \mathcal{G}} tc_{T_j}}{P} \right\rceil \quad (4)$$

where,

$\mathcal{G}$ is the DFG, and

tc_{T_j} is the execution time of task T_j .

Figure 5: An acyclic data flow graph.

Consider the small acyclic DFG of Figure 5. The three operations have a duration of 1 TU. The respective computational iteration period bound, for $P = 1$, $P = 2$, and $P = 3$, is given by:

$$\begin{array}{ll} P = 1 & T_{0b} = \lceil \frac{3}{1} \rceil = 3 \\ P = 2 & T_{0b} = \lceil \frac{3}{2} \rceil = 2 \\ P = 3 & T_{0b} = \lceil \frac{3}{3} \rceil = 1 \end{array}$$

3.2 The Scheduling Algorithm

The scheduling algorithm introduced in this section is valid for cyclic as well as for acyclic DFG's. Acyclic DFG's are a special case of the first.

Cyclic DFG's have a minimum sampling rate with unlimited resources, T_{0min} , imposed by the topology of the graph and the hardware constraints [12]. For acyclic DFG's, the minimum sampling rate with unlimited resources, T_{0min} , is 0. With fixed resources, the sampling period T_0 has a lower bound given by T_{0b} .

So, with the number of processors fixed to be P , the lower bound for T_0 is given by:

$$\text{lower bound for } T_0 = \max(T_{0min}, T_{0b}) \quad (5)$$

The description of the algorithm is given below:

1. Determine the computational iteration period bound, T_{0b} .
2. If the DFG is cyclic, determine the critical loop, and T_{0min} (by i.e. the method proposed in [5]), otherwise, make $T_{0min} = 0$.
3. Make $T_0 = \max(T_{0b}, T_{0min})$.
4. Select a reference operation, and determine the scheduling-range chart for T_0 .
5. Partition the scheduling-range chart in sectors that correspond to each equivalence class, consecutively labeled from 0 to $T_0 - 1$.
6. Maintain for each equivalence class a pointer to the

first available level; initially, this pointer has a value 1 for all equivalence classes.

7. Select the unscheduled operation with the shortest scheduling range. In case of equal length, give priority to an operation with a fixed-limit range. Select by lowest label, if multiple operations have a range with fixed limit.
8. Assign the selected operation to the equivalence class(es) that has (have) the lowest first available level(s). Of course only those equivalence classes are considered, which are covered by the scheduling range of the operation. Since the level of a class is not associated to any processor, operations that cover more than one class can be assigned to different levels.
- When the operation cannot be assigned to the available empty classes, repeat the *procedure for iteration period adjustment* described in Section 4 until the operation has enough contiguous available classes to be assigned.
9. Update the scheduling-range chart to incorporate the fixed position of the operation just scheduled. Update the first available level of the affected equivalence classes.
10. Repeat steps 7, 8 and 9 until all operations have been scheduled.
11. Assign operations to processors.

- (a) Sort operations according to their computational delay; the longest first; in case of equal duration sort by label.
- (b) Maintain for each equivalence class a pointer to its first available level which now is associated to a processor. This pointer should be initialized to 1 for all equivalence classes.
- (c) Remove the first operation from the list and assign it to the first processor level that has empty all the equivalence classes to which the operation has been assigned to.

When there is no processor that fulfills this condition, repeat the *procedure for iteration period adjustment* described in Section 4 until the assignment of the operation is possible.

- (d) Update the first-available-level pointers.
- (e) Repeat 11b, 11c, and 11d until all the operations have been assigned to processors.

Postponing the processor-assignment phase, as described by the above algorithm, is not necessary when all the operations have unit length. Each level in the equivalence classes can be directly associated to a processor. Figure 6 sequentially shows the scheduling of the three unit-length operations of the small acyclic example of Figure 5, for a number of processors $P = 2$ and $P = 3$.

Figure 6: Maximum throughput scheduling for 2 and 3 processors for the DFG of Figure 5. The sequential steps in the algorithm are shown in (a), (b) and (c) respectively, with the corresponding equivalence-class assignment (i), scheduling-range chart (ii), and execution of the partial processor schedule (iii).

Figure 7 shows some steps of the scheduling of the cyclic DFG of Figure 1, for a number of processors $P = 3$, and $P = 4$, corresponding to the equivalence-class-assignment phase. The processor-assignment phase and final iterative processor schedule, for $P = 3$, are shown in Figure 8.

4 Iteration Period Adjustment

Fixing the iteration period T_0 and the number of processors P , as proposed by Schwartz in [14], does not always lead to a solution [9]. The rate-optimal scheduling algorithms proposed in [6] guarantee a solution by fixing T_0 and optimizing the number of resources. If we define the *computational processor bound*, P_{0b} , as:

$$P_{0b} = \left\lceil \frac{\sum_{T_j \in G} tc_{T_j}}{T_0} \right\rceil \quad (6)$$

then, those algorithms will provide a solution in a number of processors P such that:

$$P \geq P_{0b} \quad (7)$$

In this paper, the number of processors is fixed and the algorithm will optimize the iteration period. A solution is

guaranteed with a iteration period given by:

$$T_0 \geq \max(T_{0min}, T_{0b}) \quad (8)$$

The reasons why fixing both T_0 and P may not lead to a solution are the following:

- The precedence constraints in the DFG can be such that the equality in Equation 8 cannot be achieved by any schedule in P processors.

An example of such a DFG is given in Figure 9(a); $T_{0min} = 4$ TU's and $P_{0b} = 2$. The initial state of the scheduling-range chart is shown in Figure 9(b). The precedence constraints are such that the only operation with mobility is 7. The rest has a fixed position. There are only two possible class-assignments for operation 7 (its duration is 2 TU's and due to the non-preemption condition it should be assigned to consecutive classes): (0-1) and (1-2). However only classes (2-3) are available. A solution with P_{0b} in T_{0min} does not exist, either the number of processors or the iteration period has to be increased.

- A fully-static unblocked processor assignment may not exist, given the equivalence-class assignment.

Consider the DFG of Figure 10(a). This graph has

Figure 7: Some steps in the scheduling algorithm for the cyclic DFG of Figure 1, for a number of processors $P = 3$ and $P = 4$. The equivalence-class assignment for reference operation 2 and scheduling-range chart (a); an intermediate step is shown in (b); the time-schedule and equivalence-class assignment for all the operations is displayed in (c).

Figure 8: Processor assignment phase, for the DFG of Figure 1 for a number of processors $P = 3$ (a), and execution of final schedule for three consecutive iterations (b).

$T_{0_{min}} = 3$ and $P_{0_b} = 2$. The three operations are members of the critical loop. The corresponding scheduling-range chart and equivalence-class assignment, after the equivalence-class assignment phase is shown in (b). However it is not possible to assign the operations to only 2 processors without the use of direct blocking [8,14], or cyclo-static techniques [14,15]. A fully-static unblocked solution will require to increase either the number of processors or the iteration period.

- The use of a heuristic cannot guarantee an optimal solution.

The iterative scheduling problem is NP-complete. The use of heuristics, as those proposed in this paper, to find a schedule in polynomial time cannot guarantee a solution in the minimum sampling period, as given by the equality in Equation 8.

Therefore when fixing the number of processors to P , the scheduling algorithm should include some mechanism to increase T_0 , when necessary, in order to guarantee a solution.

Figure 9: Iteration-period adjustment during the equivalence class assignment phase. Recursive data flow graph (a). Initial state of the scheduling-range chart (b). Scheduling-range chart after the iteration-period adjustment (c). Final processor schedule (d).

Figure 10: Iteration-period adjustment during the processor-assignment phase. DFG (a). Initial time schedule and equivalence-class assignment (b). Time schedule and equivalence-class assignment after iteration-period adjustment (c). Final processor assignment (d).

The adjustment of T_0 can be necessary during the class-assignment phase and/or during the processor-assignment phase.

The procedure for adjustment of the iteration period consists of the following steps:

1. Introduce a new equivalence class within the scheduling range of the conflicting operation (the operation that cannot be scheduled or assigned to any processor) as shown in Figure 11.

Figure 11: Iteration period adjustment. The conflicting operation is i (a). A new equivalence-class is introduced within the scheduling range of the operation, increasing the number of equivalence classes from $L+1$ to $L+2$ (b).

2. Make $T_0 = T_0 + 1$.

3. Update the scheduling-range chart for the inclusion of the new equivalence-class, and increase in the sampling period.

Figure 12 shows the iteration-period adjustment during the equivalence-class assignment phase. The rectangles at the bottom show the equivalence-class assignment of operations.

Figure 9(c) shows how the procedure for adjustment of the iteration period works, during the equivalence-class assignment. The iteration period is increased from 4 to 5 TU's, and a new equivalence class, 4, is introduced. This increases the mobility of the not yet scheduled operations³, in this case only 7. The final processor schedule is shown in (d).

A two-processor unblocked fully-static solution to the problem of Figure 10(a) is shown in Figure 10(d) after applying the procedure for adjustment of the iteration period in the processor assignment phase (c).

5 Time-Complexity Issues

In order to determine the overall time complexity of the algorithm, the following main steps have to be considered:

1. The computation of T_{0min} . This value is necessary for any scheduling algorithm for iterative cyclic DFG's, not only the one presented in this paper. In [5] an efficient algorithm is presented that does not enumerate all loops. It has a time complexity of $O(de + d^4)$, where d is the number of delay elements and e is the number

³The increase in the sampling period affects the mobility of all the operations except the one chosen as a reference if, after the adjustment the algorithm is started from the beginning, since now none of the loops is critical. However, the selected strategy is to continue the scheduling and not to restart the complete scheduling processes with the new value of T_0 .

Figure 12: Iteration-period adjustment during the equivalence-class-assignment phase. There are no three consecutive free equivalence classes where operation k can be assigned (a). Therefore the sampling period T_0 is increased by one time unit, and a new available equivalence class is introduced (b). The scheduling proceeds (c).

of edges in the DFG.

2. The generation of the inequalities that express the forward and backward precedence relations. There is at most one forward and one backward precedence relation for each pair of computational nodes. So, the number of relations is $\mathcal{O}(c^2)$, where c is the number of computational nodes in the DFG.
3. The combination of all inequalities to find the scheduling range for each operation with respect to a reference operation. In [5] it is explained how the Fourier-Motzkin elimination method [3] can be adapted to operate in polynomial-time for the special type of inequalities occurring here. The result is an $\mathcal{O}(c^3)$ algorithm.
4. The updating of the scheduling ranges. Each time an operation is given a fixed position within its range (there are at most T_0 choices), the ranges of the other operations can be updated in $\mathcal{O}(c^2)$ time. So, as c operations have to be scheduled, this step needs $\mathcal{O}(cT_0 + c^3)$ time.
5. The adjustment of the iteration period. In the equivalence-class assignment phase this requires $\mathcal{O}(cT_0)$ time and in the processor-assignment phase $\mathcal{O}(cPT_0)$ time. It is assumed that the number of adjustments is a small number that does not grow with the problem size.

So, the total time complexity of the algorithm is:

$$\mathcal{O}(de + d^4 + c^3 + cPT_0).$$

6 Conclusions

An algorithm based on an alternative scheduling approach for iterative acyclic and cyclic DFG's with limited resources that exploits inter- and intra-iteration parallelism has been presented. The proposed method is based on guiding the scheduling algorithm with the information supplied by a scheduling-range chart.

The priority for the selection of operations based on mobility, and ranges with fixed-limits, intends to maximize the scheduling range of the yet unscheduled operations.

For cases where the precedence constraints do not allow a schedule in the originally selected optimal sampling period, T_0 , the algorithm provides an adjustment procedure, so a solution is always guaranteed.

The delay of the processor-assignment phase increases the efficiency of the algorithm, when the operations have processing times different from the unity, and are non-preemptive. The algorithm provides a procedure for the adjustment of T_0 for those cases where a solution requires more processors than the number of levels of the equivalence classes [7].

Acknowledgement

The authors are very grateful to Sabih Gerez for his many valuable suggestions in the preparation of this paper and his contribution to the time-complexity issues.

References

- [1] J.B. Clary. Design tools for real-time signal processing systems. *Proceedings of the SPIE*, 827:2-9, 1987.
- [2] G. Conte and D. del Corso, editors. *Multi-Microprocessor Systems for Real-time Applications*. D. Reidel Publishing Company, 1985.
- [3] G.B. Dantzig. *Linear Programming and Extensions*. Princeton University Press, Princeton, New Jersey, 1963.
- [4] M.R. Garey and D.S. Johnson. *Computers and Intractability. A Guide to the Theory of NP-Completeness*. W.H. Freeman & Company, San Francisco, 1979.
- [5] S.H. Gerez and S.M. Heemstra de Groot. *Implementation and Time-Complexity Issues of Range-Chart Scheduling*. Technical Report, University of Twente, March 1990. Int. Report EL-BSC-90N032.
- [6] S.M. Heemstra de Groot. *Scheduling of Iterative Data Flow Graphs by Means of the Scheduling-Range Chart*. Technical Report, University of Twente, October 1989. Internal Report EL-BSC-89N181.
- [7] S.M. Heemstra de Groot. *Study and Evaluation of Mul-*

- tiprocessor Scheduling Techniques of Atomic Operations for DSP Algorithms*. Technical Report, University of Twente, January 1989. Int. Report EL-BSC-89N013.
- [8] S.M. Heemstra de Groot and O.E. Herrmann. Evaluation of some multiprocessor scheduling techniques of atomic operations for recursive DSP graphs. In *Proceedings of the European Conference on Circuit Theory and Design*, pages 400–404, September 1989.
 - [9] S.M. Heemstra de Groot and O.E. Herrmann. Rate-optimal scheduling of recursive DSP algorithms based on the scheduling range chart. In *Proceedings of the IEEE International Symposium on Circuits and Systems*, May 1990. New Orleans.
 - [10] G.L. Lang, M. Dharssi, F.M. Longstaff, Ph.S. Longstaff, P.A.S. Metford, and M.T. Rimmer. An optimum parallel architecture for high-speed real-time digital signal processing. *IEEE Computer*, 47 – 57, February 1988.
 - [11] E.A. Lee and D.G. Messerschmitt. Static scheduling of synchronous data flow programs for digital signal processing. *IEEE Transactions of Computers*, C-36(1):24–35, January 1987.
 - [12] M. Renfors and Y. Neuvo. The maximum sampling rate of digital filters under speed constraints. *IEEE Transactions on Circuits and Systems*, CAS-28(3):196–202, March 1981.
 - [13] J. Santos, J. Parera, and M. Veiga. A hypercube multiprocessor for digital signal processing algorithm research. In *1988 International Conference on Acoustics Speech and Signal Processing*, pages 1698–1701, 1988.
 - [14] D.A. Schwartz. *Synchronous Multiprocessor Realizations of Shift Invariant Flow Graphs*. PhD thesis, Georgia Institute of Technology, 1985.
 - [15] D.A. Schwartz and T.P. Barnwell. Cyclo-static multiprocessor scheduling on the optimal realization on shift-invariant flow graphs. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 1384 – 1387, 1985.