

An Integer Linear Programming Approach to the Overlapped Scheduling of Iterative Data-Flow Graphs for Target Architectures with Communication Delays

Sacha L. Sindorf and Sabih H. Gerez

Department of Electrical Engineering, University of Twente, The Netherlands

E-mail: s.h.gerez@el.utwente.nl

Abstract— This paper considers the scheduling of *homogeneous synchronous data-flow graphs* also called *iterative data-flow graphs* (IDFGs) on a multiprocessor system. Algorithms described by such graphs consist of a core computation that is iterated “infinitely often”. The computation does not contain data-dependent decisions. All scheduling decisions for such algorithms can be taken at compile time.

Fine-grain parallelism is assumed where the basic tasks are primitive operations (such as additions) and the inter-processor communication times are just a few clock cycles. Scheduling methods for such a model have recently been presented by several authors. These approaches assign operations to processors and data transfers to links at appropriate times. The work presented here extends the one reported in [16] based on integer linear programming. Optimal results to problems of reasonable size were found after acceptable computation times.

I. INTRODUCTION

An algorithm that contains computations that can be executed simultaneously, offers possibilities of exploiting the parallelism present by implementing it on appropriate hardware such as a *multiprocessor* system. The class of algorithms considered in this paper is limited to algorithms that can be represented by *homogeneous synchronous data-flow graphs* [1], also called *iterative data-flow graphs* (IDFGs) [2]. Such an algorithm consists of a core computation that is iterated an infinite number of times and that does not contain data-dependent decisions. Many algorithms in the field of *digital signal processing* (DSP) belong to this class.

A parallel implementation of an algorithm implies that each operation in the data-flow graph is mapped on a hardware unit and a time instant. Finding these mappings is called (*processor*) *assignment* and *scheduling* respectively. A schedule is called *nonoverlapped* if all operations belonging to the same iteration have to finish before the operations of the next iteration can start; if operations belonging to different iterations can execute simultaneously, the schedule is called *overlapped* [3, 4]. Overlapped scheduling is sometimes also called *loop folding* [5, 6]. An overlapped schedule can be characterized by two entities: the *iteration period* T_0 , the time between the invocation of the same operation in subsequent iterations, and the *latency*, the time that passes between the consumption of the first input and

the production of the last output in the same iteration. Both times are expressed in multiples of the global system clock period.

Schedules for IDFGs can be completely determined at compile time due to the fact that the algorithms that they represent do not require data-dependent decisions. Such schedules are called *static*. In this paper only *fully-static* overlapped schedules are considered which means that all iterations have the same schedule and the same processor assignment. Two different optimization goals for scheduling are normally distinguished: in *time-constrained* scheduling, the goal is to use as little hardware as possible for a given execution speed, while in *resource-constrained* scheduling, the goal is to execute the IDFG as fast as possible on a given hardware configuration. In this paper, the resource-constrained problem is addressed.

The multiprocessor scheduling problem for iterative algorithms has received quite some attention in the last years. However, most of the publications have considered the simplified problem in which the time to communicate data from one processor to another was negligible (see the references in [7]). When communication delays are taken into account, often target architectures are considered in which setting up a communication or performing the transfer of a single data word requires tens or even hundreds of system clock cycles (see e.g. [8]). In such an architecture, it only makes sense to have parallel implementations if the basic tasks performed by a single processor, require a similar number of cycles. This type of parallelism is called *coarse grain*. A detailed account of issues related to the scheduling with *interprocessor communication* (IPC) delays can be found in [9].

In this paper, however, *fine-grain* parallelism is considered. The basic tasks are primitive operations such as additions and multiplications and the processors in the target architecture are supposed to transfer a data word in just a few cycles. Suitable architectures for fine-grain parallelism have received increased attention in the last years due to the fact that advances in VLSI technology have made it possible to integrate a large number of processors on a single chip [10–13].

The communication-sensitive scheduling problem for

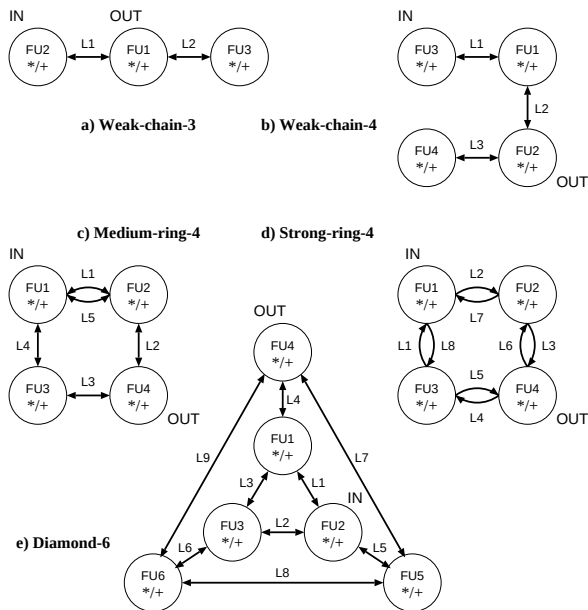


Fig. 2. Multiprocessor configurations as used for the benchmarks.

typical architectures that have been used for benchmarking. Note that some of the processors carry the labels IN and OUT: these labels indicate at which processor the inputs to the computations arrive and at which the outputs have to be delivered.

The problem formulation allows a separate processor assignment for each of the input and output nodes in the IDFG as part of the specification. The assigned processor is indicated by $FIXk_i$ with $i \in IN \cup OUT$. Also the times at which inputs are available and outputs should become available can be specified separately for each input and output. These times are indicated by $FIXj_i$ with $i \in IN \cup OUT$.

Now that the main aspects of the problem have been presented, the problem to be solved can be formulated. Given are: an IDFG, a target architecture, an iteration period and the times and processors for the inputs and outputs. The goal of an ILP formulation is to check whether a mapping of the IDFG on the target architecture exists that respects the iteration period and the timing constraints on inputs and outputs. If it exists, the mapping should be reported. For each processor, it will detail which computation is executed at each clock cycle (if any) and for each link it will detail which data is transported at each clock cycle (if any). Besides, the result should not contain spurious data transports.

The main goal is to solve the resource-constrained scheduling problem. Only the IDFG, the target architecture, the input times and the processor assignment for the inputs and outputs is considered given. The primary objective is to find the solution with the fastest iteration period T_0 . The secondary objective is to minimize the *latency* for the minimal iteration period. When the IDFG has a single input and a single output, the latency is the time difference between the consumption of the input and the production

of the output. Otherwise, a latency can be defined between each of the input-output pairs. Values for the output times $FIXj_i$ with $i \in OUT$ can be derived from latencies and the input times. In the rest of this paper, it will be assumed that there is a single input and a single output. The latency associated with this pair will be denoted by L_t .

With T_0 and L_t fixed, the scheduling problem becomes a *decision problem*: there either exists a solution or not. Each new value combination of the parameters T_0 and L_t will result in a different ILP. In order to achieve the main goal, the ILP solver is called in the body of two nested loops: the outer loop increments T_0 until a solution is found and the inner loop varies L_t . The next section provides more information on the values that the parameters can assume.

III. PREPARATORY CALCULATIONS

Although one could directly define the constraints of an ILP formulation from the problem description, it is worthwhile to analyze the description first. In this way, the numbers of variables and inequalities in which the variables occur, can be reduced significantly. This leads to smaller search spaces and shorter solution times.

In order to reduce the search space, one should consider the precedence constraints. The fact that the input and output times as well as the iteration period are known, and that a certain execution order has to be observed, makes that a *scheduling range* [2] can be computed for each computational node. This is an interval consisting of the earliest and latest clock cycles in which the computation can start. The earliest time is called the *as soon as possible* (ASAP) time while the latest time is called the *as late as possible* (ALAP) time. For a given IDFG, a T_0 , input and output times, the scheduling ranges can be computed using longest-path algorithms for graphs, such as the Bellman-Ford algorithm [2, 22, 23]. For each computational node $i \in N$ the ASAP and ALAP times will be denoted by $ASAP_i$ respectively $ALAP_i$. The range will be denoted by Rx_i : $Rx_i = [ASAP_i, ALAP_i]$.

ASAP and ALAP times may have negative values (due to the delay elements that lead to negative offsets in precedence constraints). For reasons of convenience, a constant value is added to all ASAP and ALAP values such that the minimum ASAP value becomes 1. The maximum ALAP value determines the universe of time values that have to be taken into consideration when mapping computations on time instants. This maximum will be denoted by $ALAP_{max}$.

The scheduling process should also route the *transports* of data between processors. Note that the mapping of data transports is a one-to-many mapping as opposed to the one-to-one mapping of operations. A transport will in general claim different links at different time instants in order to make sure that data produced by some computation reaches all processors that consume these data. On the other hand, just a single processor needs to be found on which to execute a computational node.

Transports also have scheduling ranges. The earliest

start time for the transport of the result of computational node $i \in N$ will be denoted by $ASAPy_i$ and can easily be computed from the ASAP time of the computation itself:

$$ASAPy_i = ASAP_i + C_i$$

An expression for the latest start time for the same transport denoted by $ALAPy_i$ is somewhat more complex and considers all edges $i \Rightarrow ip$ outgoing from i :

$$ALAPy_i = \max_{ip} \{D_{i,ip} T_0 + ALAP_{ip} - \delta\}$$

The maximum is required here because the transport of data along one edge may in principle be independent from the transport of the same data along other edges. In a way similar to the definition of $ALAP_{max}$ for computations, $ALAPy_{max}$ can be defined to denote the maximum of all transport ALAPs. $ALAPy_{max}$ delimits the universe of time values available for data transports. The scheduling range for transports will be denoted by Ry_i .

In order to reduce the search space, meaningful values for T_0 and L_t should be tried. There are several factors that contribute to the lower bound of T_0 , called the *iteration period bound* (IPB). It is well-known [24] that a lower bound on the iteration period can be derived from the topology of the IDFG when the IDFG contains loops (each loop contains at least one delay element in order for the IDFG to be computable). The bound follows from the fact that an algorithm should be executed sufficiently slow such that data produced in one of the nodes in the loop can return to the node in time for the computation of the next iteration. Assuming that all loops in the IDFG are contained in the set L , the bound is given by:

$$IPB_1 = \max_{l \in L} \left\lceil \frac{\sum_{i \in \gamma(l)} C_i}{\sum_{i \Rightarrow ip \in \eta(l)} D_{i,ip}} \right\rceil$$

where $\gamma(l)$ denotes the computational nodes in a loop l and $\eta(l)$ the edges. The loop that leads to the maximum value in the expression is called the *critical loop*. It is not necessary to enumerate all loops. Many efficient algorithms exist that can compute the bound [25].

Another cause for the existence of a lower bound on the iteration period lies in the resource-constrained nature of the problem. A processor can at most be utilized for 100%:

$$IPB_2 = \left\lceil \frac{\sum_{i \in N} C_i}{|P|} \right\rceil$$

Yet another trivial lower bound is given by the duration of the IPC time or by the longest pipeline period of a computation on a processor. In a fully-static schedule the iteration period should be larger than this duration. Combining all bounds, one gets the following expression for the IPB:

$$IPB = \max \left(IPB_1, IPB_2, \delta, \max_{i \in N} L_i \right) \quad (1)$$

The maximum of the three lower bound computations gives a minimum value for T_0 . This value has been used in the implementation reported in this paper.

The lower bound for the iteration period can be chosen even tighter by taking the delay time δ for IPC into account. This will be illustrated for the IDFG of Figure 1 assuming that an addition lasts one clock cycle, a multiplication two and a data transport two. The critical loop is $c_4-c_2-d_1$. This loop implies an iteration period of 3 (a total duration of computations of 3 to be executed within a single iteration) and needs to be executed in a single processor in order not to lose time on IPC. The IDFG has a second loop that shares node c_2 with the first: $c_3-c_1-c_2-d_1-d_2$. The processor executing the critical loop when $T_0 = 3$ is fully occupied, implying that computations c_3 and c_1 should be executed on a second processor and that communication to and from c_2 on the other processor is necessary. The total time to execute the second loop then becomes 8 (2 additions, 1 multiplication and 2 IPCs) for which two iterations are available (2 delay nodes in the loop). The conclusion is that no schedule with $T_0 = 3$ can exist and that the tighter lower bound of $T_0 = 4$ should be used.

Another reasoning to come to the same conclusion starts with the observation that the loop $c_3-c_1-c_2-d_1-d_2$ contains two delay nodes. The total duration of computations in this loop is 4. Because of the fully-static scheduling requirement, any schedule with $T_0 < 4$ will necessitate that the execution of the loop is distributed across two or more processors. This immediately means that at least two IPC delays have to be added to the total duration of the loop resulting in a total loop execution time of 8 for two iterations or 4 for one iteration.

Finding a tighter lower bound in this way, is quite complex for the general case. To the knowledge of the authors, no general procedure is known for computing a lower bound for T_0 that takes IPC into account.

A lower bound on L_t is quite straightforward to find. One simply computes the ALAP time of the output. An upper bound on L_t is given by the sum of the durations of all computations in the IDFG plus δ times the number of links in the shortest path from the input to the output processor. This is an upper bound because all computations can take place on a single processor and sufficient time is available to transport the data from input to output.

IV. ILP CONSTRAINTS

The scheduling problem as described above has been cast into a zero-one ILP formulation. This means that the problem is formulated in terms of variables that can assume either the value 0 or the value 1. These variables are collected into a vector $\mathbf{x}$. The objective of the problem is to minimize or maximize a linear function of these variables:

$$\mathbf{c}^T \mathbf{x}$$

So, $\mathbf{c}$ is a vector of constants. In addition a number of equality and inequality constraints have to be respected:

$$\mathbf{A}_1 \mathbf{x} = \mathbf{b}_1$$

$$\mathbf{A}_2 \mathbf{x} \leq \mathbf{b}_2$$

where $\mathbf{A}_1$, $\mathbf{A}_2$ are constant matrices and $\mathbf{b}_1$, $\mathbf{b}_2$ are constant vectors. Once a problem has been formulated in this format, an ILP solver will be able to look for a solution.

The variables of the scheduling problem are:

- $X_{i,j,k}$; $X_{i,j,k} = 1$ implies that computation $i \in N$ starts at time j on processor k .
- $Y_{i,j,l}$; $Y_{i,j,l} = 1$ implies that data produced by computation $i \in IN \cup N$ is sent at time j through data transportation link l .

The constraints for the scheduling problem have been collected in Figure 3. A short explanation of them will be given here. A more detailed explanation can be found in [26]. Although the constraints are directly derived from [16], it should be stated that substantial modifications have been made in the details.

Equation 2 expresses the fact that at most one operation is executed at the same time on a processor. For every processor, the entire universe of time instants ($1 \leq j \leq ALAP_{\max}$) has to be scanned for conflicts. Note that this universe is never smaller than T_0 . This means that if a computation i_1 has been mapped to start on some time instant j on a processor k , no other computations i_2 can start for the duration of the pipeline period L_{i_2} before time instant j or any time instant that is separated a multiple of T_0 from j .

For this reason, the formulation not only considers the scheduling range Rx_i of an operation, but its iterated version with a period of T_0 . The set of time instants that is obtained in this way, is called $SLOTSx_i$. One of the advantages of using this formulation is that no variables $X_{i,j,k}$ are created for values of j that do not make sense. This approach is new with respect to [16].

Equation 3 states that every operation i has to start on exactly one processor k at exactly one time j .

Equation 4 is the counterpart of Equation 2 for data transports. Some extensions with respect to [16] have been made for handling bidirectional links and IPC times larger than 1. The set Wb contains all bidirectional paired links $lb_u = \{l_u, l_v\}$ and all unidirectional links $lb_w = \{l_w\}$. Wherever l_u is tested for data conflicts, l_v has to be tested for the same conflicts. The set $SLOTSy_i$ is the counterpart of $SLOTSx_i$ for the universe of time instants available for transportation. The requirement that data produced by an operation i passes a link between two processors at most once is expressed by Equation 5.

Equation 6 states that a data transport can only leave a processor at some time if it had arrived at the processor at an earlier time or was produced on that processor itself. In order to capture the topology of the target architecture the following notation is used: fr_l indicates the processor where a link l originates from and to_l the processor that the link connects to.

The correct transportation of input data is controlled by Equation 7. A binary constant α is needed to express the availability of the input i on its input processor, $FIXk_i$. When input is available after the defined time of input, $FIXj_i$, α becomes one, allowing input transportations from $FIXk_i$. A transportation of input i is still pos-

TABLE I
THE FOURTEEN DIFFERENT SCHEDULING PROBLEMS.

Problem	IDFG	Hardware			
		Structure	+	*	δ
A	second	strong-ring-4	1	2	2
B	second	medium-ring-4	1	2	2
C	second	weak-chain-4	1	2	1
D	third	weak-chain-4	1	2	1
E	third	weak-chain-3	1	2	1
F	third	diamond-6	1	2	1
G	jaumann	weak-chain-3	1	5	1
H	jaumann	weak-chain-3	1	5	2
I	lattice	weak-chain-3	1	5	1
J	lattice	weak-chain-3	1	5	2
K	elliptic	weak-chain-4	1	2	1
L	elliptic	weak-chain-4	1	2	2
M	elliptic	weak-chain-3	1	2	1
N	elliptic	weak-chain-3	1	2	2

sible with $\alpha = 0$, because the processor can be a non-input processor transporting the input from an incoming link.

Precedence relations between computations and transports, where a computation ip depends on a computation i or on a result of i from an earlier iteration, are handled by Equation 8. The constraint is, in a sense, a counterpart for Equation 6 and states that the execution of a computational node can only start if its inputs have been delivered through a link in time or produced earlier on the same processor. A similar situation for data arriving from an input rather than from another computation is controlled by Equation 9.

Equation 10, finally, takes care that an output is available in time at its destination processor. It is either produced on the destination or has to arrive through an incoming link.

It was stated earlier that the ILP formulation should in principle check whether a solution is possible for a given T_0 - L_t combination. As such, no objective function is necessary. However, the constraints do not prevent solutions in which data is transported through unnecessarily long paths or even to processors that do not need the data. These spurious transports can be prevented by counting all data transports in the objective function of the ILP problem:

$$\text{Minimize : } \sum_i \sum_j \sum_l Y_{i,j,l} \\ \forall i \in IN \cup N, \forall j \in SLOTSy_i, \forall l \in W$$

V. EXPERIMENTAL RESULTS

The ILP formulation has been applied to the combinations of IDFG and target-architecture benchmarks presented earlier in [14] and listed in Table I. The IDFGs were already given in [14] (and also in [2]) and are not included in this paper except for **second** (Figure 1) and **elliptic** (Figure 4). The target architectures are displayed in Figure 2. The table gives the durations of additions, multiplications

$$\sum_{i \in N} \left\{ \sum_{q=0}^{L_i-1} \left\{ \sum_{\substack{p=0 \\ (j+p*T_0-q) \in SLOTSx_i}}^{\lfloor (ALAP_{\max}-j+q)/T_0 \rfloor} X_{i,j+p*T_0-q,k} \right\} \right\} \leq 1 \quad 1 \leq j \leq T_0, \forall k \in P \quad (2)$$

$$\sum_{j \in SLOTSx_i} \sum_{k \in P} X_{i,j,k} = 1 \quad \forall i \in N \quad (3)$$

$$\sum_{i \in IN \cup N} \sum_{q=0}^{B-1} \sum_{l \in lb} \left\{ \sum_{\substack{p=0 \\ (j+p*T_0-q) \in SLOTSy_i}}^{\lfloor (ALAPy_{\max}-j+q)/T_0 \rfloor} Y_{i,j+p*T_0-q,l} \right\} \leq 1 \quad 1 \leq j \leq T_0, \forall lb \in Wb \quad (4)$$

$$\sum_{j \in SLOTSy_i} Y_{i,j,l} \leq 1 \quad \forall i \in IN \cup N, \forall l \in W \quad (5)$$

$$Y_{i,j,l} \leq \left\{ \sum_{\substack{jp < j \\ jp \in Ry_i}} \sum_{\substack{lp \\ to_{lp} = fr_l}} Y_{i,jp,lp} + \sum_{\substack{jp \leq j - C_i \\ jp \in Rx_i}} X_{i,jp,fr_l} \right\} \quad \forall i \in N, \forall j \in Ry_i, \forall l \in W \quad (6)$$

$$Y_{i,j,l} \leq \left\{ \sum_{\substack{jp < j \\ jp \in Ry_i}} \sum_{\substack{lp \\ to_{lp} = fr_l}} Y_{i,jp,lp} + \alpha \right\} \quad \forall i \in IN, \forall j \in Ry_i, \forall l \in W \quad (7)$$

where:

$$\alpha = \begin{cases} 1 & \text{if } j \geq FIXj_i \text{ and } fr_l = FIXk_i \\ 0 & \text{otherwise} \end{cases}$$

$$X_{ip,j,k} \leq \sum_{\substack{jp < j + D_{i,ip}*T_0 \\ jp \in Ry_i}} \sum_{\substack{l \\ to_l = k}} Y_{i,jp,l} + \sum_{\substack{jp \leq j + D_{i,ip}*T_0 - C_i \\ jp \in Rx_i}} X_{i,jp,k} \quad (\forall i, ip \in N | i \Rightarrow ip, i \neq ip), \forall j \in Rx_{ip}, \forall k \in P \quad (8)$$

$$X_{ip,j,k} \leq \sum_{\substack{jp < j + D_{i,ip}*T_0 \\ jp \in Ry_i}} \sum_{\substack{l \\ to_l = k}} Y_{i,jp,l} + \alpha \quad (\forall i \in IN, \forall ip \in N | i \Rightarrow ip), \forall j \in Rx_{ip}, \forall k \in P \quad (9)$$

where:

$$\alpha = \begin{cases} 1 & \text{if } j \geq FIXj_i \text{ and } k = FIXk_i \\ 0 & \text{otherwise} \end{cases}$$

$$1 \leq \sum_{\substack{jp < FIXj_{ip} + D_{i,ip}*T_0 \\ jp \in Ry_i}} \sum_{\substack{l \\ to_l = k}} Y_{i,jp,l} + \sum_{\substack{jp \leq FIXj_{ip} + D_{i,ip}*T_0 - C_i \\ jp \in Rx_i}} X_{i,jp,k} \quad (\forall i \in N, \forall ip \in OUT | i \Rightarrow ip), k = FIXk_{ip} \quad (10)$$

Fig. 3. The constraints for the ILP formulation of the scheduling problem.

and IPC for each combination. No pipelining is assumed for the execution of the computations and the input time is set to 1 in all benchmarks.

The ILP solver *BonsaiG* was used for the experiments. It can be freely downloaded from *NEOS* or the problems can be submitted to a remote server running the software [27]. The Benchmarks A through E were run remotely; the remaining ones locally on an *HP 9000/785/C3000* server. The results are presented in Table II in which they are compared to the results of [14] which were obtained by an

approach based on *genetic algorithms* (GAs).

From the results, it can be seen that the T_0 could not be improved in any of the cases. The column labeled IPB contains the value computed according to Equation 1. It can be proved in most cases, following a reasoning as given earlier in this paper, that the $T_0 > IPB$ that was found, is optimal in the presence of IPC.

As far as the latency is concerned, several improvements with respect to the GA approach can be reported. The situation is even better than can be directly observed from

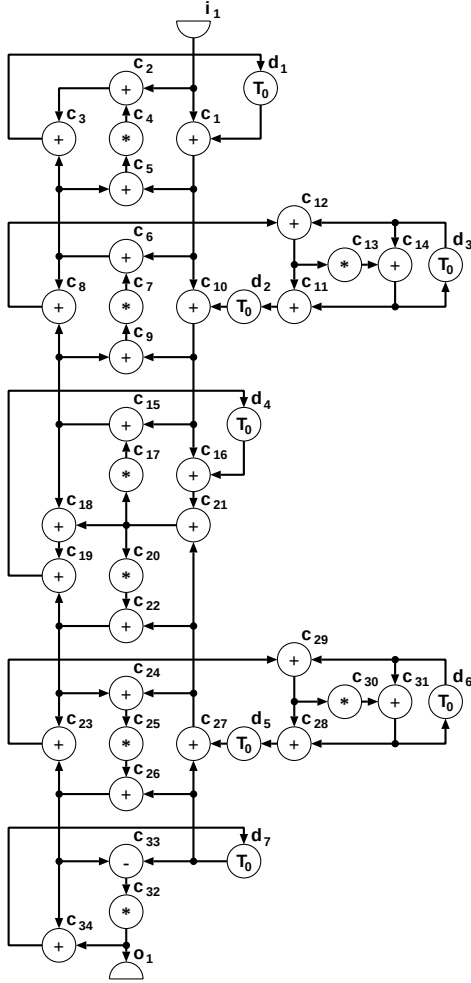


Fig. 4. The IDFG of a fifth-order wave digital elliptic filter, called elliptic.

the table because the GA approach did not route inputs and outputs to specified processors.

The comparison of the CPU times seems to be in favor of the GA approach. One should, however, take into account that the run times reported for GA are averages of 50 runs and that not all runs resulted in the best results reported in the table. The CPU times for ILP are cumulative counting the times spent on all T_0 - L_t combinations until a solution is found (the GA approach also dynamically increases these parameters). It has to be observed that Benchmarks M and N were too complex to be solved for ILP.

The result obtained for Benchmark K is shown in Figure 5. In the figure, the main numbers indicate the index of the computations in the IDFG. Their superscripts indicate the iteration number. The links in this benchmark are bidirectional. The number of the processor to which the data flows is given in the lower right corner. The ILP problem corresponding to this solution had 3920 variables and 2992 constraints.

TABLE II
THE RESULTS FOR THE BENCHMARKS.

	ILP			GA [14]			IPB	Notes
	T_0	L_t	CPU	T_0	L_t	CPU		
A	4	5	1619s	4	6	11s	3	(1)
B	4	5	-	4	9	12s	3	
C	3	7	244s	3	7	12s	3	
D	5	4	4s	5	5	24s	5	
E	6	4	75s	6	4	25s	6	(2)
F	3	8	88249s	3	9	27s	3	
G	16	9	1s	16	9	33s	16	(3)
H	18	10	41s	18	13	43s	16	
I	16	30	13s	16	30	27s	14	(4)
J	18	31	16s	18	31	33s	14	(3)
K	18	15	1062s	18	16	134s	16	(3)
L	21	14	720s	21	18	129s	16	(5)
M	18	-	-	18	20	111s	16	(3,6)
N	21	-	-	21	14	111s	16	(5,6)

Notes: (1) ILP-solver time-outs ($>10,000$ s)
(2) First evaluation with 'local' *BonsaiG*
(3) $T_{0,initial} = 18 \neq IPB$
(4) $T_{0,initial} = 16 \neq IPB$
(5) $T_{0,initial} = 21 \neq IPB$
(6) evaluation time > 48 hours

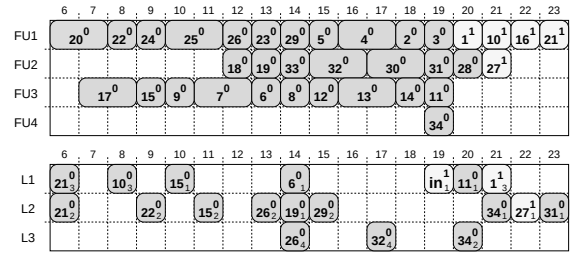


Fig. 5. The schedule found for Benchmark K.

VI. CONCLUSIONS

This paper has presented an ILP approach to the overlapped scheduling problem for target architectures with nonnegligible IPC times. The approach improves some aspects of the formulation given in [16]. By its nature, ILP provides optimal solutions and has exponentially growing computation times. It turns out that it is feasible to apply ILP to benchmarks of reasonable size. Improvements can be expected when the ILP formulation is split into sub-problems that are solved separately.

REFERENCES

- [1] E.A. Lee and D.G. Messerschmitt, "Synchronous data flow," *Proceedings of the IEEE*, vol. 75, no. 9, pp. 1235–1245, September 1987.
- [2] S.M. Heemstra de Groot, S.H. Gerez, and O.E. Herrmann, "Range-chart-guided iterative data-flow-graph scheduling," *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, vol. 39, pp. 351–364, May 1992.
- [3] K.K. Parhi and D.G. Messerschmitt, "Static rate-optimal scheduling of iterative data-flow programs via optimum unfolding," *IEEE Transactions on Computers*, vol. 40, no. 2, pp. 178–195, February 1991.

- [4] S.H. Gerez, S.M. Heemstra de Groot, E.R. Bonsma, and M.J.M. Heijligers, "Overlapped scheduling techniques for high-level synthesis and multiprocessor realizations of DSP algorithms," in *Advanced Techniques for Embedded System Design and Test*, J.C. Lopez, R. Hermida, and W. Geisselhardt, Eds., pp. 125–150. Kluwer Academic Publishers, Boston, 1998.
- [5] G. Goossens, J. Rabaey, J. Vandewalle, and H. De Man, "An efficient microcode compiler for application specific DSP processors," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 9, no. 9, pp. 925–937, September 1990.
- [6] T.F. Lee, A.C.H. Wu, Y.L. Lin, and D.D. Gajski, "A transformation-based method for loop folding," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 13, no. 4, pp. 439–450, April 1994.
- [7] V.K. Madisetti, *VLSI Digital Signal Processors, An Introduction to Rapid Prototyping and Design Synthesis*, IEEE Press and Butterworth Heinemann, Boston, 1995.
- [8] J. Sanchez and H. Barral, "Multiprocessor implementation models for adaptive algorithms," *IEEE Transactions on Signal Processing*, vol. 44, no. 9, pp. 2319–2331, September 1996.
- [9] S. Sriram and S.S. Bhattacharyya, *Embedded Multiprocessors, Scheduling and Synchronization*, Marcel Dekker, New York, 2000.
- [10] D.C. Chen and J.M. Rabaey, "A reconfigurable multiprocessor IC for rapid prototyping of algorithmic-specific high-speed DSP data paths," *IEEE Journal of Solid-State Circuits*, vol. 27, no. 12, pp. 1895–1904, December 1992.
- [11] D. Shoemaker, F. Honoré, C. Metcalf, and S. Wand, "Numesh: An architecture optimized for scheduled communication," *The Journal of Supercomputing*, vol. 10, no. 3, pp. 285–302, 1996.
- [12] E. Waingold, M. Taylor, D. Shikrishna, V. Sarkar, W. Lee, V. Lee, J. Kim, M. Frank, P. Finch, R. Barua, J. Babb, S. Amarasinghe, and A. Agarwal, "Baring it all to software: Raw machines," *IEEE Computer*, pp. 86–93, September 1997.
- [13] P. Nussbaum, B. Girau, and A. Tisserand, "Field programmable processor arrays," in *Evolvable Systems: From Biology to Hardware*, M. Sipper, D. Mange, and A. Perez-Urbe, Eds., pp. 311–322. Springer, Berlin, 1998, Lecture Notes in Computer Science, No. 1478.
- [14] E.R. Bonsma and S.H. Gerez, "A genetic approach to the overlapped scheduling of iterative data-flow graphs for target architectures with communication delays," in *ProRISC Workshop on Circuits, Systems and Signal Processing*, November 1997.
- [15] S. Tongsimma, E.H.M. Sha, and N.L. Passos, "Communication-sensitive loop scheduling for DSP applications," *IEEE Transactions on Signal Processing*, vol. 45, no. 5, pp. 1309–1322, May 1997.
- [16] K. Ito, T. Iwata, and H. Kunieda, "An optimal scheduling method for parallel processing system of array architecture," in *Asia and South Pacific Design Automation Conference, ASP-DAC '97*, 1997, pp. 447–454.
- [17] G.L. Nemhauser and L.A. Wolsey, *Integer and Combinatorial Optimization*, John Wiley and Sons, New York, 1988.
- [18] M.R. Garey and D.S. Johnson, *Computers and Intractability, A Guide to the Theory of NP-Completeness*, W.H. Freeman and Company, San Francisco, 1979.
- [19] C.T. Hwang, J.H. Lee, and Y.C. Hsu, "A formal approach to the scheduling problem in high-level synthesis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 10, no. 4, pp. 464–475, April 1991.
- [20] V.K. Madisetti and B.A. Curtis, "A quantitative methodology for rapid prototyping and high-level synthesis of signal processing algorithms," *IEEE Transactions on Signal Processing*, vol. 42, no. 11, pp. 3188–3208, November 1994.
- [21] P.W.G. Poiesz, S.H. Gerez, and E.R. Bonsma, "Generation, genetic optimization and VHDL-based verification of detailed iterative static schedules for multiprocessor systems," in *ProRISC 10th Annual Workshop on Circuits, Systems and Signal Processing*, Mierlo, The Netherlands, November 1999.
- [22] T.H. Cormen, C.E. Leiserson, and R.L. Rivest, *Introduction to Algorithms*, MIT Press, Cambridge, Massachusetts, 1990.
- [23] S.H. Gerez, *Algorithms for VLSI Design Automation*, John Wiley and Sons, Chichester, 1999.
- [24] R. Reiter, "Scheduling parallel computations," *Journal of the ACM*, vol. 15, no. 4, pp. 590–599, October 1968.
- [25] A. Dasdan, S.S. Irani, and R.K. Gupta, "Efficient algorithms for optimum cycle mean and optimum cost to time ratio problems," in *36th Design Automation Conference*, 1999.
- [26] S.L. Sindorf, "An exact method for the scheduling of iterative algorithms on multiprocessor systems," M.S. thesis, University of Twente, Department of Electrical Engineering, August 2000, EL-S&S-009.00.
- [27] "NEOS server for optimization," <http://www-neos.mcs.anl.gov/>.