Cyclic Scheduling of Tasks with Unit Processing Time on Dedicated Sets of Parallel Identical Processors

Přemysl Šůcha[1] and Zdeněk Hanzálek[1,2]
[1]Department of Control Engineering, Faculty of Electrical Engineering,
Czech Technical University in Prague, Czech Republic, {suchap,hanzalek}@fel.cvut.cz
[2]Merica, Czech Republic, hanzalek@merica.cz

This paper presents an integer linear programming (ILP) model for cyclic scheduling of tasks with unit processing time. Our work is motivated by digital signal processing (DSP) applications on FPGA (Field-Programmable Gate Array) architectures hosting several kinds of identical arithmetic units. These hardware resources can be formalized as dedicated sets of parallel identical processors. We propose a method to find an optimal periodic schedule of DSP algorithms on such architectures. The accent is put on the efficiency of the ILP model. We show advantages of the model in comparison with common ILP model on benchmarks and randomly generated instances.

Keywords: [Multi-processor Scheduling, Cyclic Scheduling, Digital Signal Processing Applications].

1 Introduction

Integer Linear Programming (ILP) formulations find their application in scheduling as well as other areas in engineering. The challenge for researchers is to provide faster solutions for ILP based approaches. The modeling problem plays a crucial role on the solution efficiency.

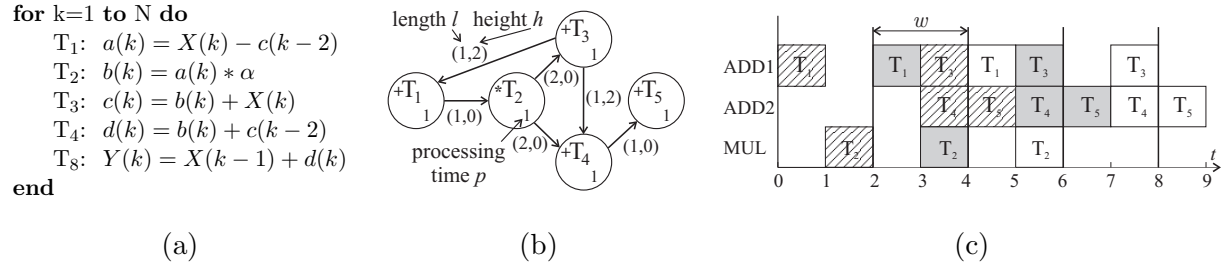
In this paper, we concentrate on the efficiency of the ILP model used for automatic parallelization of computation loops. Our work is motivated by DSP (Digital Signal Processing) applications executed on FPGAs (Field Programmable Gate Arrays) hosting several kinds of identical arithmetic units. These hardware resources can be formalized as dedicated processors.

The DSP application is usually implemented in the form of an iterative loop. A parallel implementation of the loop requires each operation of the loop to be mapped on the arithmetic unit at a given time, which is formulated as a cyclic scheduling problem. The arithmetic units operating with real numbers are usually pipelined, i.e. the input data are usually passed to the unit in one clock cycle, while the result of a computation is available after several clock cycles denoted as the *input–output latency* of the unit formalized as *precedence delays*.

An example of the arithmetic library for FPGA is FP32 [1] implementing a 32-bit floating point number system and compliant with IEEE standards. Another library using a logarithmic number system arithmetic is HSLA [2]. In both cases, rather complex arithmetic is required. Therefore, scheduling of such dedicated HW resources has to carefully consider the algorithm structure, in order to achieve the desired performance of the application. Scheduling also helps to choose the appropriate arithmetic library prior to the algorithm implementation.

Cyclic scheduling deals with a set of operations (tasks) that have to be performed an infinite number of times [3]. This approach is also applicable if the number of loop repetitions is large enough. The aim is then to find a periodic schedule with a minimum period. One repetition of the loop, called an *iteration*, can be performed in several periods. Therefore, the schedule is called *overlapped* [4], since the execution of the operations belonging to different iterations can interleave.

Modulo scheduling and *software pipelining* [5] are related terms to *cyclic scheduling*, which are usually used in the compiler community. A common ILP based approach (further called the



(a) (b) (c)

Figure 1: (a) An example of a computation loop lattice wave digital filter (LWDF). (b) Corresponding data dependency graph G. (c) An optimal schedule with $w = 2$.

binary method) uses a binary decision variable x_{it} determining whether a computation of task i starts at clock cycle t. Unfortunately, the size of such an ILP model (corresponding to the time complexity) depends on the period length and therefore it is not suitable for HW architectures with longer input–output latencies of the arithmetic units (e.g. FP32 or HSLA). For example the ILP approach with x_{it} decision variable is used in [6], where the additional objective is to optimize the word–length of the arithmetic units. In [7], the ILP model is extended by automatic processor selection from a library and [4] applies the approach to architectures with interprocessor communication. ILP formulation using the binary method is also employed in [8] to schedule multifunction loop accelerators. A general solution using ILP is shown in [9] but the size of the ILP model is independent of the period length only for a monoprocessor solution.

With respect to these works, our solution leads to a simpler ILP formulation with less integer variables. We propose a method based on ILP to find the optimal periodic schedule. Motivated by FPGA architectures with standard arithmetic units operating with real numbers, we formulated an ILP model for cyclic scheduling of tasks with unit processing time on dedicated sets of parallel identical processors. We call this ILP model the *integer method*, since we use an integer variable to represent the start time of a task while the binary method uses a set of binary variables for the same purpose.

This paper is organized as follows: Section 2 describes the notation and the Basic Cyclic Scheduling problem assuming an unlimited number of processors. The following section presents a formulation of the cyclic scheduling of tasks with unit processing time on dedicated sets of parallel identical processors. Section 4 presents the results demonstrated on benchmarks and shows a comparison with an ILP model using one binary decision variable for each task and each clock cycle within the period. Section 5 concludes the paper.

2 Problem Statement

Operations in an *iterative loop* can be considered as a set of n tasks $\mathcal{T} = \{T_1, T_2, ..., T_n\}$ to be performed K times. One execution of set $\mathcal{T}$ labeled with the integer index $k \geq 1$ is called an *iteration*. The scheduling problem is to find the start time $s_i(k)$ of every occurrence of T_i [3]. Figure 1(a) shows an illustrative example of a simple computation loop (LWDF filter [10]).

Data dependencies of this problem can be modeled by a directed, double weighted graph $G = (\mathcal{T}, \mathcal{E})$. Each task (node in $\mathcal{T}$) is characterized by the processing time p_i. Edge $e_{ij} \in \mathcal{E}$ from the node i to j is weighted by a couple of integer constants l_{ij} and h_{ij}. Length l_{ij} represents the minimal distance in clock cycles from the start time of task T_i to the start time of T_j and is always

greater than zero.

When considering pipelined processors, the processing time p_i represents the time to feed the processor (i.e. new data can be fed into the pipelined processor after p_i clock cycles) and length l_{ij} represents the time of the computation (i.e. the input–output latency). The parameter height h_{ij} specifies the shift of the iteration index (dependence distance) related to the data produced by T_i and consumed by T_j.

Figure 1(b) shows the data dependence graph of the iterative loop shown in Figure 1(a). Processing time p_i is equal to 1 for all arithmetic units. The length l_{ij} corresponds to the input–output latency of the given unit. ADD and MUL unit have 1 and 2 clock cycles input–output latency respectively.

When assuming a *periodic schedule* with *period* w (i.e. the constant repetition time of each task), each edge e_{ij} in graph G represents one precedence relation constraint

$$s_j - s_i \geq l_{ij} - w \cdot h_{ij}, \tag{1}$$

where s_i denotes the start time of task T_i at the first iteration.

The aim of the cyclic scheduling [3] is to find a periodic schedule while minimizing the period w. The scheduling problem is simply solved when the number of processors is the number of processors not limited, i.e. is sufficiently large. The minimal feasible period, given by the *critical circuit* c in graph G, is equal to $\max_{c \in C(G)} \left(\sum_{e_{ij} \in c} l_{ij} \right) / \left(\sum_{e_{ij} \in c} h_{ij} \right)$, where $C(G)$ denotes a set of cycles in G. The critical circuit can be found in polynomial time [3], and we use this value to determine the lower bound of the period in our scheduling problem.

When the *number of processors* m is restricted, the cyclic scheduling problem becomes NP–hard [3]. Unfortunately, in our case the number of processors m is restricted and the processors are grouped into dedicated sets up to their capability to execute specific operations. Three iterations of the optimal schedule to the loop from Figure 1(a) are shown in Figure 1(c). In this case, a hardware architecture with two ADD units and one MUL unit is considered.

3 Integer Linear Programming Formulation

In this section we define an Integer Linear Programming formulation for the problem of cyclic scheduling of tasks with unit processing time on dedicated sets of parallel identical processors $\mathcal{P} = \{\mathcal{P}_1, \ldots \mathcal{P}_d, \ \ldots, \mathcal{P}_D\}$. First we solve a problem for one dedicated set of parallel identical processors (i.e. all tasks in $\mathcal{T}$ are processed on $\mathcal{P}_1 = \mathcal{P}$) which is further generalized in Section 3.2. To define the ILP formulation, we introduce several integer and binary variables.

Let $\hat{s}_i$ be the remainder after division of s_i (the start time of T_i in the first iteration) by w and let $\hat{q}_i$ be the integer part of this division. Then s_i can be expressed as follows

$$s_i = \hat{s}_i + \hat{q}_i \cdot w, \quad \hat{s}_i \in \langle 0, w-1 \rangle, \quad \hat{q}_i \geq 0. \tag{2}$$

This notation divides s_i into $\hat{q}_i$, the index of the *execution period*, and $\hat{s}_i$, the number of clock cycles within the execution period.

Furthermore, let $\hat{x}_{ij}$ be a binary decision variable such that $\hat{x}_{ij} = 1$ if and only if $\hat{s}_i \leq \hat{s}_j$ (i.e. T_i is followed by T_j or both T_i and T_j start at the same time) and $\hat{x}_{ij} = 0$ if and only if $\hat{s}_i > \hat{s}_j$ (i.e. T_j is followed by T_i). And let $\hat{y}_{ij}$ be a binary decision variable such that $\hat{y}_{ij} = 1$ if and only if $\hat{s}_i = \hat{s}_j$ (i.e. tasks T_i and T_j are processed at the same time on different processors).

In the rest of the paper we consider $p_i = 1$ for all $T_i \in \mathcal{T}$. The period w is assumed to be a constant, since multiplication of two decision variables cannot be formulated as a linear inequality.

Therefore, the optimal w is found by iterative calls of the ILP problem. The ILP problem using the above defined variables is:

$$\min \sum_{i=1}^{n} \hat{q}_i \tag{3}$$

subject to:

$$\hat{s}_j + \hat{q}_j \cdot w - \hat{s}_i - \hat{q}_i \cdot w \geq l_{ij} - w \cdot h_{ij}, \qquad \forall (i,j); \exists e_{ij} \in \mathcal{E} \tag{4}$$

$$\hat{s}_i - \hat{s}_j + w \cdot \hat{x}_{ij} + (1-w) \cdot \hat{y}_{ij} \geq 1, \qquad \forall i,j \in \{1,\ldots,n\}; i < j \tag{5}$$

$$\hat{s}_i - \hat{s}_j + w \cdot \hat{x}_{ij} - \hat{y}_{ij} \leq w - 1, \qquad \forall i,j \in \{1,\ldots,n\}; i < j \tag{6}$$

$$-\hat{x}_{ij} + \hat{y}_{ij} \leq 0, \qquad \forall i,j; i < j \text{ and } T_i, T_j \in \mathcal{T} \tag{7}$$

$$\sum_{j=i+1}^{n} \hat{y}_{ij} \leq m - 1, \qquad \forall i \in \{1,\ldots,n-m\} \tag{8}$$

where: $\hat{s}_i \in \langle 0, w-1 \rangle$; $\hat{q}_i \geq 0$; $\hat{s}_i, \hat{q}_i \in \mathbb{Z}$; $\hat{x}_i, \hat{y}_i \in \{0,1\}$

3.1 Constraints of the ILP Model

Constraint (4) is a direct application of precedence constraint (1). Constraints (5), (6), (7) and (8) limit the number of processors used at a given time. The binary decision variables $\hat{x}_{ij}$ and $\hat{y}_{ij}$ define the mutual relation of tasks T_i and T_j ($i \neq j$) within the execution period. Their relation is expressed with constraints (5) and (6). There are three feasible combinations:

1. When $\hat{x}_{ij} = 0$ and $\hat{y}_{ij} = 0$, constraint (6) is eliminated in effect. Constraint (5) reduces to $\hat{s}_j + 1 \leq \hat{s}_i$, i.e. T_j is followed by T_i within the execution period.
2. When $\hat{x}_{ij} = 1$ and $\hat{y}_{ij} = 0$, constraint (5) is eliminated in effect. Constraint (6) reduces to $\hat{s}_i + 1 \leq \hat{s}_j$, i.e. T_i is followed by T_j within the execution period.
3. When $\hat{x}_{ij} = 1$ and $\hat{y}_{ij} = 1$, constraints (5) and (6) are equivalent to $\hat{s}_i = \hat{s}_j$, i.e. T_i and T_j are scheduled at the same time within the execution period.
4. Combination $\hat{x}_{ij} = 0$ and $\hat{y}_{ij} = 1$ is not feasible due to constraint (7).

The number of available processors is limited using the variable $\hat{y}_{ij}$. When $\hat{y}_{ij} = 1$, there is a resource conflict between tasks T_i, T_j and they can not be scheduled on the same processor. Since we consider the unit processing time of the tasks, this relation between the tasks expressed with $\hat{y}_{ij}$ is symmetric ($\hat{y}_{ij} = \hat{y}_{ji}$) and transitive (if $\hat{y}_{ij} = 1$ and $\hat{y}_{jk} = 1$ then $\hat{y}_{ik} = 1$). The relation can be expressed by graph G^y where the nodes represent the tasks in $\mathcal{T}$. There is an edge between T_i and T_j iff $\hat{y}_{ij} = 1$. Due to the symmetry and transitivity, the graph G^y consists of disjoint complete subgraphs H_u^y, such that $G^y = \left\{H_1^y, \ldots, H_u^y, \ldots, H_U^y\right\}$, where $U \leq w$. Each H_u^y corresponds to the set of tasks executed at the same time. Then for each H_u^y, it holds that the number of nodes in H_u^y is equal to the chromatic number $\chi(H_u^y)$ which is equal to $\Delta(H_u^y) + 1$, where $\Delta(H_u^y)$ is the maximum degree in H_u^y [11]. In our case $\Delta(H_u^y) = \sum_{j=1}^{n} \hat{y}_{ij}$ where T_i is a node of H_u^y.

In order to limit the maximal number of processors used, we limit the number of nodes in the biggest H_u^y. It can be expressed as a condition $\sum_{j=1}^{n} \hat{y}_{ij} \leq m-1, \forall i \in \{1,\ldots,n\}$ and can be further simplified to the form of constraint (8) using the following lemma

Lemma 1. *Constraint* $\sum_{j=1}^{n} \hat{y}_{ij} \leq m-1, \forall i \in \langle 1,n \rangle$ *is equivalent to constraint* $\sum_{j=i+1}^{n} \hat{y}_{ij} \leq m-1$, $\forall i \in \langle 1, n-m \rangle$, *in the ILP model.*

Proof. $\sum_{j=1}^{n} \hat{y}_{ij} = \sum_{j=1}^{i} \hat{y}_{ij} + \sum_{j=i+1}^{n} \hat{y}_{ij}$. By induction we show that the first term of this addition can be omitted.

$i = 1$: $\sum_{j=1}^{i} \hat{y}_{ij} = \hat{y}_{11} = 0$

$i = 2$: $\sum_{j=1}^{i} \hat{y}_{ij} = \hat{y}_{21} + \hat{y}_{22} = \hat{y}_{21}$. If $\hat{y}_{21} = 0$ then $\hat{y}_{21}$ has no effect on constraint (8). If $\hat{y}_{21} = 1$ then nodes T_1 and T_2 belong to the same H_u^y and the number of nodes in H_u^y is already limited in constraint (8) for $i = 1$.

$i = k$: $\sum_{j=1}^{i} \hat{y}_{ij} = \hat{y}_{k1} + \hat{y}_{k2} + \ldots + \hat{y}_{kk} = \hat{y}_{k1} + \ldots + \hat{y}_{k(k-1)}$. If $\hat{y}_{k1} + \ldots + \hat{y}_{k(k-1)} = 0$ then $\hat{y}_{k1}, \ldots, \hat{y}_{k(k-1)}$ have no effect on constraint (8). If $\hat{y}_{k1} + \ldots + \hat{y}_{k(k-1)} > 0$ then at lest one node T_v ; $v \in \langle 1, k-1 \rangle$ belongs to the same H_u^y. Thereafter, the number of nodes in H_u^y is already limited by constraint (8) for some $i = v$.

Finally, constraint (8) can be formulated for $i \in \langle 1, n-m \rangle$, since $\sum_{j=n-m+1}^{n} \hat{y}_{ij} \leq m-1$ is always satisfied. □

Please notice that we do not directly assign the tasks to the processors but we only restrict the number of tasks executed at the same time. The above shown ILP formulation requires $2 \cdot n$ integer and $n^2 - n$ binary variables constrained in $n_e + 3/2 \cdot (n^2 - n) + n - m$ constraints. It is obvious that the size of the ILP model depends only on the number of tasks n, the number of edges n_e and the number of processors m.

3.2 Dedicated Sets of Parallel Identical Processors

The above mentioned formulation of the processor constraints allows one to generalize the approach to the problem with dedicated sets of parallel identical processors where each task is assigned to a dedicated set of parallel identical processors $\mathcal{P}_d$ such that $\mathcal{P} = \{\mathcal{P}_1, \ldots, \mathcal{P}_d, \ldots, \mathcal{P}_D\}$ where D is the number of these sets. Then $m_d = |\mathcal{P}_d|$ denotes the number of processors in set $\mathcal{P}_d$ and m, the number of processors, is equal to $\sum_{d=1}^{D} m_d$. For example, in one of the benchmarks in Section 4.1 we assume to have 3 dedicated sets (one for addition, one for multiplication and one for division). Furthermore, n_d denotes the number of tasks assigned to the set $\mathcal{P}_d$ and n, the number of tasks, is equal to $\sum_{d=1}^{D} n_d$.

The tasks assigned to different processors are not conflicting, i.e. they do not impose any restriction with respect to the feasibility of the schedule. Therefore, constraints (5), (6), (7) and (8) are generated separately for each set $\mathcal{P}_d$ with m_d processors and n_d tasks. On the other hand, constraint (4) remains the same, since it represents the precedence constraints among the tasks that may run on different processors.

3.3 Objective Function

We recall that the goal of cyclic scheduling is to find a feasible schedule with the minimal period w. Therefore, w is not constant as we assumed in the ILP formulation above and it is a positive integer value. Period w^*, the shortest period resulting in a feasible schedule, is constrained by its lower bound and its upper bound [12]. The optimal period w^* can be found iteratively by formulating one ILP model for each iteration. These iterative calls of ILP do not need to be performed for all w between the lower bound and the upper bound, but the interval bisection method or simple incrementing of the period can be used, until w^* is found.

Using the ILP formulation we are able to test the schedule feasibility for a given value of w. In addition, we minimize the secondary criterion formulated using the objective function of ILP.

One of the simplest objectives is to minimize the iteration overlap by objective function (3) as is assumed in this paper. Another criterion is the minimization of the iteration length or minimization of the data storage units for transfers among the tasks [13].

4 Results

The presented scheduling technique was implemented and run on an AMD Opteron at 2.2 GHz using the ILP solver tool CPLEX [14]. The complexity of integer linear programs can be estimated by using the number of integer variables in the ILP model, but each change of the problem instance may lead to a significant change of the algorithm computation time.

4.1 Standard Benchmarks

We compare the result of the binary method (e.g. [4, 8, 6]) and the integer method (shown in Section 3) on standard benchmarks: the second order wave digital filter (WDF) [15], the Jaumann filter (JAUMANN) [16], the recursive least squares filter (RLS) [12], the seventh-order biquadratic IIR filter (IIR7) [17] and the fifth–order wave digital elliptic filter (ELLIPTIC) [18].

					integer method			binary method		
Benchmark	n	arithmetic units	q_{max}	$w*$	#var	*#constr*	*CPU time*	*#var*	*#constr*	*CPU time*
WDF	8	2ADD ($l_{ij}=2$), 2MUL ($l_{ij}=3$)	3	9	48	65	**0.016**	80	39	**0.063**
	8	2ADD ($l_{ij}=9$), 2MUL ($l_{ij}=2$)	3	29	48	65	**0.016**	240	79	**0.047**
JAUMANN	17	2ADD ($l_{ij}=9$), 2MUL ($l_{ij}=2$)	2	58	202	293	**0.016**	1003	161	**0.063**
	17	2ADD ($l_{ij}=11$), 2MUL ($l_{ij}=8$)	2	82	202	293	**0.016**	1411	209	**0.688**
RLS	26	2ADD ($l_{ij}=9$), 2MUL ($l_{ij}=2$), 2DIV ($l_{ij}=2$)	4	26	320	454	**0.047**	702	135	**0.172**
	26	2ADD ($l_{ij}=11$), 2MUL ($l_{ij}=8$), 2DIV ($l_{ij}=28$)	2	74	320	454	**0.031**	1950	279	**0.953**
IIR	29	2ADD ($l_{ij}=9$), 2MUL ($l_{ij}=2$)	8	20	450	657	**0.125**	609	134	**0.234**
	29	2ADD ($l_{ij}=11$), 2MUL ($l_{ij}=8$)	7	30	450	657	**0.063**	899	133	**0.156**
ELLIPTIC	34	2ADD ($l_{ij}=2$), 2MUL ($l_{ij}=3$)	3	29	774	1147	**0.063**	1020	150	**0.313**
	34	2ADD ($l_{ij}=9$), 2MUL ($l_{ij}=2$)	2	96	774	1147	**0.063**	3298	284	**576.250**
	34	2ADD ($l_{ij}=11$), 2MUL ($l_{ij}=8$)	2	134	774	1147	**0.063**	4590	360	**1.406**

Table 1: Benchmarks

Experiments are summarized in Table 1 where n denotes the number of tasks, $\hat{q}_i$ denotes the upper bound of q_{max} given *a priory*. The parameters of dedicated sets of processors including the number of processors in the set, function of the unit and corresponding input–output latency l_{ij} in brackets are shown in the column *arithmetic units*. In our experiments, we have primarily considered the parameters of the FP32 and HSLA libraries [1, 2]. The algorithm results are given by the shortest period resulting in a feasible schedule w^*. The parameters of the ILP models (integer and binary) in Table 1 are the number of ILP variables *#var*, the number of ILP constraints *#constr* and the *CPU time*. The CPU time is the time required to compute the optimal solution, given as a sum of iterative calls of the ILP solver.

4.2 Randomly Generated Instances

To compare the time complexity of the integer method with the one of the binary method thoroughly, we evaluate the average CPU times for randomly generated instances.

The randomly generated graph G consists of $\lceil 2/3 \cdot n \rceil$ edges of height $h_{ij} = 0$ and n edges of height $h_{ij} > 0$. The height $h_{ij} > 0$ is chosen from a uniform distribution on the interval $\langle 1, 3 \rangle$ (and rounded toward the nearest integer). The out degree of the nodes in the generated graph G is less then or equal to 4. All tasks (corresponding to the nodes) were associated with one set consisting of two parallel identical processors. To demonstrate the influence of the precedence delay on the

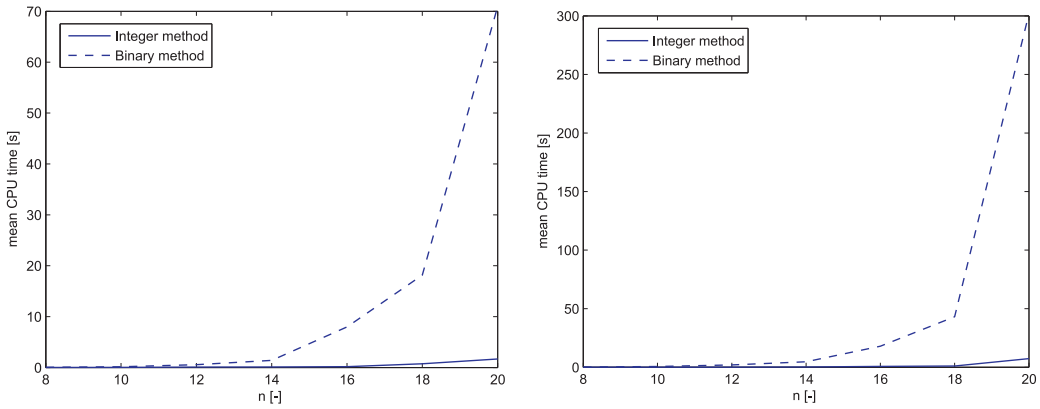


Figure 2: Comparison of integer and binary method.

CPU time, we performed two benchmarks, one for $l_{ij} = 6$ (see Figure 2(a)) and one for $l_{ij} = 9$ (see Figure 2(b)).

For each number of tasks n, 500 test instances have been randomly generated. The mean *CPU time* in dependence on the number of nodes (tasks) n is shown in Figures 2(a) and (b). The results show considerably lower time requirements for the integer method. From the comparison between Figures 2(a) and 2(b), it follows that for instances with greater precedence delays ($l_{ij} = 9$), the two methods differ even more.

5 Conclusions

This paper presents an ILP–based cyclic scheduling on dedicated sets of parallel identical processors. In this paper, we have focused on problems with unit processing time of tasks which is common in most FPGA architectures. Experimental results show that our model is more effective for problems with a longer period than the commonly used ILP model where the number of variables is independent of the period length (the binary method) [4, 8, 6]. The time complexity of the presented ILP model can be further optimized by using the *elimination of redundant processor constraints* presented in [13].

6 Acknowledgement

This work was supported by the Ministry of Education of the Czech Republic under Project 2C06017 and Research Programme MSM6840770038.

References

[1] Celoxica Ltd. (2004) *Platform Developers Kit: Pipelined Floating-point Library Manual*. http://www.celoxica.com.

[2] Matoušek, R., Tichý, M., Pohl, A. Z., Kadlec, J., and Softley, C. (2002) Logarithmic number system and floating-point arithmetics on FPGA. *International Conference on Field-Programmable Logic and Applications (FPL '02)*, pp. 627–636.

[3] Hanen, C. and Munier, A. (1995) A study of the cyclic scheduling problem on parallel processors. *Discrete Applied Mathematics*, **57**, 167–192.

[4] Sindorf, S. L. and Gerez, S. H. (2000) An integer linear programming approach to the overlapped scheduling of iterative data-flow graphs for target architectures with communication delays. *PROGRESS 2000 Workshop on Embedded Systems, Utrecht, The Netherlands*.

[5] Rau, B. R. and Glaeser, C. D. (1981) Some scheduling techniques and an easily schedulable horizontal architecture for high performance scientific computing. *MICRO 14: Proceedings of the 14th annual workshop on Microprogramming*, Piscataway, NJ, USA, pp. 183–198.

[6] Kum, K.-I. and Sung, W. (2001) Combined word-length optimization and high-level synthesis of digital signal processing systems. *IEEE Transactions on Computer–Aided Design of Integrated Circuits and Systems*, **20**, 921–930.

[7] Kazuhito, I., Lucke, E., and Parhi, K. (1999) ILP based cost-optimal DSP synthesis with module selection and data format conversion. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, **6**.

[8] Fan, K., Kublur, M., Park, H., and Mahlke, S. (2006) Increasing hardware efficiency with multifunction loop accelerators. *CODES+ISSS '06: Proceedings of the 4th international conference on Hardware/software codesign and system synthesis*. ACM Press.

[9] Fimmel, D. and Müller, J. (2001) Optimal software pipelining under resource constraints. *Journal of Foundations of Computer Science*, **12**.

[10] Vesterbacka, M., Palmkvist, K., Sandberg, P., and Wanhammar, L. (1994) Implementation of fast dsp algorithms using bit-serial arithmetic. *National Conf. on Electronic Design Automation, Stockholm*, March.

[11] West, D. B. (2001) *Introduction to Graph Theory*, second edition. Prentice Hall.

[12] Šůcha, P., Pohl, Z., and Hanzálek, Z. (2004) Scheduling of iterative algorithms on FPGA with pipelined arithmetic unit. *10th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS 2004), Toronto, Canada*.

[13] Šůcha, P., Hanzálek, Z., Heřmánek, A., and Schier, J. (2007) Scheduling of iterative algorithms with matrix operations for efficient FPGA design–implementation of finite interval constant modulus algorithm. *Journal The Journal of VLSI Signal Processing*, **46**, 35–53. Springer.

[14] ILOG, Inc. (2005) *CPLEX Version 9.1*. http://www.ilog.com/products/cplex/.

[15] Fettweis, A. (1986) Wave digital filters: theory and practice. *Proceedings of the IEEE*, **74**, 270–327.

[16] Heemstra, S. M., Gerez, S. H., and Herrmann, O. E. (1992) Fast prototyping of datapath-intensive architectures. *IEEE Transactions on Circuits and Systems I*, **39**, 351–364.

[17] Rabaey, J. M., Chu, C., Hoang, P., and Potkonjak, M. (1991) Fast prototyping of datapath-intensive architectures. *IEEE Des. Test*, **8**, 40–51.

[18] Bonsma, E. and Gerez, S. (1997) A genetic approach to the overlapped scheduling of iterative data-flow graphs for target architectures with communication delays. *ProRISC Workshop on Circuits, Systems and Signal Processing*.