PERFORMANCE TUNING OF ITERATIVE ALGORITHMS IN SIGNAL PROCESSING

Zdeněk Pohl, Jiří Kadlec *

Institute of Information Theory and Automation
Academy of Sciences of the Czech Republic
email: {xpohl,kadlec}@utia.cas.cz

Přemysl Šůcha, Zdeněk Hanzálek †

CAK, Department of Control Engineering
Czech Technical University in Prague
email: {suchap,hanzalek}@fel.cvut.cz

ABSTRACT

Presented high-level synthesis describes scheduling for wide class of DSP algorithms. Several FPGA vendors or even ASIC designs are targeted via Handel-C compiled by Celoxica DK3.1 compiler. Using our approach, the designer can easily change type of used pipelined arithmetic modules and then check new performance. The optimal time schedule is found by cyclic scheduling using Integer Linear Programming while minimizing the schedule period in the terms of clock cycles. Experimental results in HW implementation, performed on logarithmic arithmetic and floating-point arithmetic, confirm significant influence of the period on the resulting performance of DSP algorithms.

1. INTRODUCTION

This paper presents design methodology, which enables to the designer effectively explore at relatively high level of abstraction the optimal FPGA implementation of the DSP algorithm. The designer has to choose storage for input vectors, input/output data flow and select the arithmetic modules. The optimal schedule is found by cyclic scheduling.

Cyclic scheduling deals with a set of operations (generic tasks) that have to be performed an infinite number of times [1]. This approach is also applicable if the number of loop repetitions is large enough. Alternative terms to *cyclic scheduling*, used in the scheduling community, are *modulo scheduling* and *software pipelining*, used in the compiler community. Existing methods for scheduling of loops can be divided into heuristic approaches e.g. [1] and methods using integer linear programming (ILP) [2, 3]. The heuristics-based techniques do not guarantee optimal solutions but have much lower computing requirements making them applicable in code compilers. On the other hand ILP is not a polynomial algorithm but for problems with reasonable size it finds an optimal solution in a reasonable amount of time.

*This work been supported by the Grant Agency of the Academy of Sciences of the Czech Republic under Project 1ET300750402

†This work was supported by the Ministry of Education of the Czech Republic under Project 1M6840770004

	HSLA (112MHz)			FP32 (181MHz)		
	Slices [-]	BRAMs [-]	Latency [clk]	Slices [-]	BRAMs [-]	Latency [clk]
MUL	83	0	2	367	0	8
DIV	79	0	2	2198	0	8
ADD	1075	28	9	1158	0	11

Table 1. Summary of HSLA and FP32 library parameters measured for Xilinx Virtex II (XC2V6000–6). Each unit has 1 clock cycle time to feed p_i. Slices stands for basic FPGA elements. BRAMs is number of block-RAMs

In this paper we present *cyclic scheduling of tasks with precedence delays on set of dedicated processors* by ILP formulation as an extension of [2]. In addition to our previous publication mentioned above, in this article we consider two different architectures, we show an extension that allows to minimize the data transfers among the arithmetic units, we extend the problem from one dedicated processor to the set of them, and finally we put emphasis on experimental work resulting in HW implementations and their performance evaluation.

This paper is organized as follows: Section 2 describes used FPGA arithmetic libraries. The next section presents our scheduling algorithm. Experimental results are presented in Section 4. Section 5 concludes the paper.

2. FLOATING-POINT LIBRARIES

The logarithmic number system (LNS) arithmetic is an alternative approach to floating-point. A real number is represented in LNS as the fixed-point value of base two logarithm of its absolute value with a special arrangement to indicate zero and NaN. An additional bit indicates the sign. A LNS arithmetic implementation, the high speed logarithmic arithmetic (HSLA), has been described in [4].

Floating-point number system uses widely known IEEE format for storage of number. First bit stands for sign, consequent 23 bits are mantissa and finally the last eight bits holds exponent. The 32-bit FP arithmetic implementation used in our experiments is Celoxica pipelined floating-point library

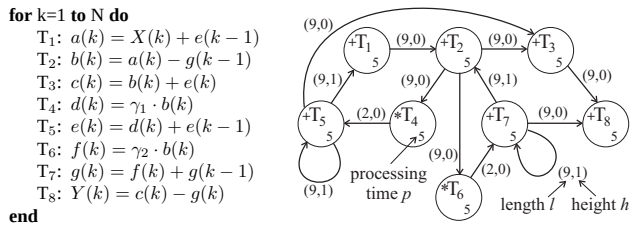
(FP32). Provided 32-bit pipelined modules were extended by input and output registers in order to increase clock performance. Parameters of HSLA and FP32 HW units are summarized in the Table 1.

3. FORMULATION AND SOLUTION OF THE SCHEDULING PROBLEM

The iterative algorithm can be implemented as a computation loop executing an identical set of operations repeatedly. Therefore our work, dealing with optimized implementation of such algorithms, is based on *cyclic scheduling*. The iterations of the cyclic schedule can overlap, therefore one can achieve better processor utilization.

3.1. Cyclic Scheduling Problem

The algorithm's n operations in a computation loop (see e.g. WDF filter [5] algorithm on Figure 1(a)) can be considered as a set of n *generic tasks* $\mathcal{T} = \{T_1, T_2, ..., T_n\}$ to be performed N times where N is usually very large. One execution of $\mathcal{T}$ is called an *iteration*. The scheduling problem is to find a start time s_i of every occurrence T_i [1].



(a) (b)

Fig. 1. (a) An example of a computation loop of wave digital filter (WDF). (b) Corresponding data dependency graph G.

Each task is characterized by processing time p_i. Data dependencies of this problem can be modeled by a directed graph G. Edge e_{ij} from the node i to j is labeled 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 the task T_i to the start time of T_j and it is always greater to zero.

The notions of the length l_{ij} and the processing time p_i are useful when we consider pipelined processors used in both libraries HSLA and FP32 presented in previous section. The processing time p_i represents the time to feed the processor (i.e. new data can be fed to the pipelined processor after p_i clock cycles) and length l_{ij} represents the time of computation (i.e. the input–output latency). Therefore, the result of a computation is available after l_{ij} clock cycles.

On the other hand, the height h_{ij} specifies the shift of the iteration index (dependence distance) related to the data produced by T_i and read (consumed) by T_j. Figure 1(b) shows oriented graph of WDF algorithm in Figure 1(a) considering HSLA library. To make the cyclic scheduling more difficult we are assuming processing of five WDF filters simultaneously which increases the processor utilization and significantly reduce the number of feasible solutions.

Assuming *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:

$$s_j - s_i \geq l_{ij} - w \cdot h_{ij}. \quad (1)$$

The aim of the cyclic scheduling problem [1] is to find a periodic schedule with minimal period w. Since N is assumed to be very large, the one iteration length λ is negligible for execution time of all iterations ($(N-1)\cdot w+\lambda$). The problem is solvable in polynomial time, assuming unlimited number of identical processors [1]. When the number of processors is restricted, the problem becomes NP–hard [1]. Unfortunately, in our case the number of processors is restricted and the processors are dedicated to execute specific operations (see Table 1). Due to the NP–hardness it is meaningful to formulate the scheduling problem as a problem of Integer Linear Programming (ILP), since various ILP algorithms solve instances of reasonable size in reasonable time.

3.2. Solution of Cyclic Scheduling on Dedicated Processors with Precedence Delays by ILP

The scheduling method shown below applies for cyclic scheduling on the architectures consisting of m dedicated processors (e.g. one addition unit, one multiplication unit, ...). Each task is *a priory* related to one dedicated processor. Therefore we introduce n_d as number of tasks related to d-th processor.

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

$$s_i = \hat{s}_i + \hat{q}_i \cdot w, \;\; \hat{s}_i \in \langle 0, w-1 \rangle , \;\; \hat{q}_i \in \langle 0, \hat{q}_{max} \rangle , \quad (2)$$

where constant $\hat{q}_{max}$ is *a priory* given upper bound of $\hat{q}_i$. This notation divides s_i into $\hat{q}_i$, the index of *execution period*, and $\hat{s}_i$, the number of clock cycles within the execution period. The schedule has to obey the two constraints explained in following paragraphs. The period w is assumed to be a constant in the subsection 3.2, since multiplication of two decision variables cannot be formulated as a linear inequality.

The first constraint is the *precedence constraint* restriction corresponding to Inequality (1). It can be formulated using $\hat{s}$ and $\hat{q}$

$$(\hat{s}_j + \hat{q}_j \cdot w) - (\hat{s}_i + \hat{q}_i \cdot w) \geq l_{ij} - w \cdot h_{ij}. \quad (3)$$

Hence, we have n_e inequalities (n_e is the number of edges in graph G), since each edge represents one precedence constraint.

Processor constraints are the second type of restrictions. They are related to the set of dedicated processor, i.e. at maximum one task is executed on one dedicated processor at a given time. It is guaranteed by Double–Inequality

$$p_j \leq \hat{s}_i - \hat{s}_j + w \cdot \hat{x}_{ij} \leq w - p_i. \quad (4)$$

where binary decision variable $\hat{x}_{ij}$ determines whether T_i is followed by T_j ($\hat{x}_{ij} = 1$) or T_j is followed by T_i ($\hat{x}_{ij} = 0$).

To derive a feasible schedule when both tasks are assigned to the same processor, Double–Inequality (4) must hold for each unordered couple of two distinct tasks, both assigned to d-th dedicated processor. Therefore, there are $\sum_{d=1}^{m}(n_d^2 - n_d)/2$ double–inequalities, i.e. there are $\sum_{d=1}^{m}(n_d^2 - n_d)$ inequalities specifying the processor constraints, where m is number of dedicated processors.

In addition we can minimize the iteration overlap by formulating the objective function as $\min \sum_{i=1}^{n} \hat{q}_i$.

The resulting ILP model containing precedence constraints (3) and processor constraints (4), use integer variables $\hat{s}_i \in \langle 0, w-1 \rangle$, $\hat{q}_i \in \langle 0, \hat{q}_{max} \rangle$ and $\hat{x}_i \in \langle 0, 1 \rangle$ and it contains $2n + \sum_{d=1}^{m}(n_d^2 - n_d)/2$ variables and $n_e + \sum_{d=1}^{m}(n_d^2 - n_d)$ constraints.

3.3. Iterative Minimization of the Period

Using ILP formulation we are able to test the schedule feasibility for the given w. 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 previous subsection, but due to the periodicity of the schedule it is a positive integer value. Period w^*, the shortest period resulting in a feasible schedule, is constrained by its lower bound w_{lower}, for which the feasibility needs to be tested, and its upper bound w_{upper}, which is feasible if at least one feasible solution exists. The values of w_{lower} and w_{upper} are found in polynomial time [2]. Optimal period w^* can be found iteratively by formulating one ILP model for each iteration. Using the interval bisection method, there are at a maximum $\log_2(w_{upper} - w_{lower})$ iterative calls of ILP.

3.4. Minimization of Data Transfers

The advantage of ILP formulation is the possibility to formulate various objective functions. E.g. if needed, this problem can be reformulated to minimize the data transfers among the tasks (i.e. the number of *intermediate results storage*). Therefore we add one slack variable Δ_{ij} each precedence constraint (3) resulting at

$$(\hat{s}_j + \hat{q}_j \cdot w) - (\hat{s}_i + \hat{q}_i \cdot w) + \Delta_{ij} = l_{ij} - w \cdot h_{ij}. \quad (5)$$

When $\Delta_{ij} = 0$, the intermediate result is passed to the next task without storing in registers or memory. On the other hand when $\Delta_{ij} > 0$, the memory or register is required. The aim is to minimize the number of $\Delta_{ij} > 0$. Therefore we introduce new binary variable Δ_{ij}^b which is equal to 1 when $\Delta_{ij} > 0$ and Δ_{ij}^b is equal to 0 otherwise. This relation is formulated as

$$(w \cdot (\hat{q}_{max} + 1)) \cdot \Delta_{ij}^b \geq \Delta_{ij}, \quad \forall e_{ij} \in G, \quad (6)$$

where $(w \cdot (\hat{q}_{max} + 1))$ represents an upper bound on Δ_{ij} and the objective is to minimize $\sum \Delta_{ij}^b$. Such a reformulated problem not only decides the feasibility of the schedule for the given period w, but if such a schedule exists, it also finds the one with minimal data transfers among the tasks.

4. EXPERIMENTAL RESULTS

4.1. Complexity of the Scheduling Problem

The presented scheduling technique was implemented and run on an Intel Pentium 4 at 2.4 GHz using non-commercial ILP solver tool GLPK[1]. In this section some results are shown on well known benchmarks found in the literature. One benchmark is the second order wave digital filter (5WDF) [5] consisting of eight tasks. It is extended to five channels by assuming five clock cycles processing time of each task (i.e. single channels are shifted by one clock cycle). The second benchmark is a differential equation solver (DIFFEQ) [6] consisting of ten tasks. Next benchmark is a seventh-order biquadratic IIR filter [7] with unrolled innermost loop (IIR7). The last one is RLS filter [8] which is the only benchmark using DIV operations.

Complexity of the scheduling algorithm is summarized in Table 2 where n is the number of tasks and m denotes the number of dedicated processors (arithmetic units). The column $size$ denotes number of ILP variables/constraints.

The scheduling algorithm results are given by w^*, the shortest period resulting in a feasible schedule. The column obj denotes the value of the objective function found while minimizing the overlap (i.e. $\sum \hat{q}_i$) for benchmarks 5WDF, DIFFEQ, IIR7 and RLS and $\sum \Delta_{ij}^b$ in case of benchmark DIFFEQ_REG, while minimizing the number of intermediate results storage. The time required to compute the optimum, given as a sum of iterative calls of the ILP solver, is shown in the column $CPU\ time$.

As follows from Table 2, the optimal solution for all benchmarks were found by GLPK solver in a reasonable

[1] GLPK 4.6 (http://www.gnu.org/software/glpk/)

Benchmark	ILP model			HSLA			FP32		
	n [-]	m [-]	$size$ [-]/[-]	w^* [clk]	obj [-]	$CPU\ time$ [s]	w^* [clk]	obj [-]	$CPU\ time$ [s]
5WDF	8	2	32/44	31	2	0,235	41	2	0,001
DIFFEQ	10	2	41/55	22	0	0,016	38	0	0,016
DIFFEQ_REG	10	2	65/67	22	3	0,218	38	3	0,218
IIR7	29	2	254/435	20	36	0,36 [2]	30	22	2,031
RLS	26	3	186/297	52	0	4,672	74	2	67,766

Table 2. Complexity of the Scheduling Algorithm.

Benchmark		f_{max} [MHz]	Period [ns]	MFLOPS [-]	Slices [-]	BRAMs [-]
WDF	HSLA	104,2	**297,6**	21,8	2344	44
	FP32	111,6	**367,4**	26,9	2448	16
DIFFEQ	HSLA	102,2	**215,2**	46,5	2520	44
	FP32	109,1	**348,2**	28,7	2637	16
DIFFEQ_REG	HSLA	107,7	**204,3**	49,0	2395	44
	FP32	112,0	**339,2**	29,5	2540	16
IIR7	HSLA	100,1	**199,9**	145,1	2844	44
	FP32	107,7	**278,6**	104,1	2921	16
RLS	HSLA	47,5	**547,2**	47,5	4787	57
	FP32	57,3	**1204,9**	21,6	7273	26

Table 3. Hardware implementation results on XC2V6000–6 for optimal schedules found by cyclic scheduling.

amount of time except IIR7 on HSLA. It was caused by large overlap of iterations, that increases the number of combinations. Using the commercial ILP solver CPLEX[2], the optimal schedule was found in 0,36 s.

4.2. HW Implementation

Hardware implementation results are summarized in Table 3. Each implemented algorithm has been designed for LNS and FP arithmetic (HSLA and FP32 library). One ADD and MUL unit was used in all designs. DIV unit has been used in lattice RLS algorithm only. Column f_{max} stands for maximal design clock. Column *Period* is the length of one period in ns of algorithm i.e. it is w^*/f_{max}.

Sets of test-vectors were generated by Matlab using bit-exact model of algorithm in given arithmetic. The test-vectors have been transferred using data stream manager (DSM) library to FPGA. The hardware platform for tests have been AlphaData ADM–XRC II (Celoxica RC2000PMC Mezzanine Card) PCI card with Xilinx Virtex II (XC2V6000-6) device. This platform enabled to test and compare bitstream statistics without being influenced by resource limitation.

5. CONCLUSIONS

This paper presents high-level synthesis approach used to optimize computation speed of iterative DSP algorithms. It is based on cyclic scheduling method using formulation by ILP. The advantage of the ILP model (presented in Section 3) in comparison with common ILP programs used for similar problems is that the number of variables is independent of period length. Moreover, the ILP approach enables to incorporate secondary objective and additional constraints.

The additional optimization criterion (presented in Section 3.4) reduced number of temporary registers. It helped to save resources and simplified source code. The lower number of input places connected to arithmetic units reduced the size of the input multiplexer, consequently the clock performance increased.

Semi-automatic HandelC code generation was implemented and it proved to be effective way how to turn DSP algorithm equations to hardware. This approach is advantageous for design upgrade to new arithmetic units and rapid prototyping of new applications.

[2] In this case the schedule was found by commercial tool CPLEX 8.0 (http://www.ilog.com/products/cplex/)

6. REFERENCES

[1] C. Hanen and A. Munier, "A study of the cyclic scheduling problem on parallel processors," *Discrete Applied Mathematics*, vol. 57, pp. 167–192, February 1995.

[2] P. Šůcha, Z. Pohl, and Z. Hanzálek, "Scheduling of iterative algorithms on FPGA with pipelined arithmetic unit," in *10th IEEE Real–Time and Embedded Technology and Applications Symposium*, May 2004.

[3] D. Fimmel and J. Müller, "Optimal software pipelining under resource constraints," *Journal of Foundations of Computer Science*, vol. 12, no. 6, pp. 697–718, 2001.

[4] J. Coleman, E. Chester, C. Softley, and J. Kadlec, "Arithmetic on the european logarithmic microprocessor," *IEEE Trans. Computers*, vol. 49, no. 7, pp. 702–715, 2000.

[5] A. Fettweis, "Wave digital filters: theory and practice," *Proceedings of the IEEE*, vol. 74, pp. 270–327, February 1986.

[6] E. G. P. Paulin, J. Knight, "Hal: A multi-paradigm approach to automatic data path synthesis," in *23rd IEEE Design Automation Conf*, Las Vegas, July 1986, pp. 263–270.

[7] J. Rabaey, C. Chu, P. Hoang, and M. Potkonjak, "Fast prototyping of datapath-intensive architectures," *IEEE Des. Test*, vol. 8, no. 2, pp. 40–51, 1991.

[8] A. Heřmánek, Z. Pohl, and J. Kadlec, "FPGA implementation of the adaptive lattice filter," in *Field–Programmable Logic and Applications*, ser. Lecture Notes in Computer Science, vol. 2778. Berlin: Springer, 2003, pp. 1095–1099.