Techniques for Optimization of Net Algorithms

[†‡]Anatoly Prihozhy, [†]Daniel Mlynek, [‡]Michail Solomennik, [†]Marco Mattavelli

[†] *Signal Processing Laboratory, EPFL, Lausanne, Switzerland*
[‡] *Department of Information Technology, ABAPRB, Minsk, Belarus*
prihozhy@yahoo.com, daniel.mlynek@epfl.ch, mi-sol@tut.by, marco.mattavelli@epfl.ch

Abstract

In this paper, techniques for optimization of net algorithms describing parallel asynchronous computations and derived from cycling and branching behavioral descriptions are presented. The parallelization level of the algorithms is defined by a set of parallel operator pairs. The optimization techniques cover the two key steps of parallelization flow: the generation of an optimal initial set of parallel operator pairs to meet the constraints on the execution time or implementation cost, and the generation of the final set of pairs to solve the net algorithm existence problem. The quality of the proposed techniques is evaluated by experimental results. The techniques based on the minimization of the net algorithm critical paths estimated using the maximal weight cliques of the sequential and parallel operator graphs constitute the most efficient approach to the generation of the initial and final sets of parallel operator pairs.

Keywords*: net algorithm, existence problem, critical path, parallelization, optimization.*

1. Introduction

The optimization during automatic parallelization of computations is the most important task in the system development process, since its results significantly influence the final computing system parameters [4-8,11-15]. Usually, the optimization task is formulated as a scheduling problem [4,6-8,10,15]. Scheduling of two types of computation is possible: synchronous and asynchronous. In the majority of scheduling tools, two types of optimization criterion are considered: to minimize the execution time of the given algorithm at the specified constraints on computing resources and to minimize the implementation cost of the computations at the specified constraints on the algorithm execution time.

2. Related work

The known sequential scheduling techniques [4,6,12,15] optimizing synchronous parallel computations introduce control steps and distribute computations into the steps. These include: as soon as possible (ASAP), as late as possible (ALAP), list scheduling, integer linear programming formulation (ILPF), path-based scheduling, scheduling for pipelines, sequential scheduling through transforming the behavioral specification, and others.

Concurrent scheduling [6,10] does not introduce control steps and states. It defines only precedence and concurrency between operators, which conserves both time and resources. The scheduling aims at optimization of asynchronous digital systems that can be represented using concurrent models such as Petri Nets [9], CASCADE control graphs [2], Predicate/Transition Nets [14], Micropipelines [3], Tangram asynchronous models [1].

Techniques for concurrent scheduling are proposed in [6,10,13]. The technique described in [6] schedules non-cyclic non-branching task graphs, analyzing dynamically the schedule critical paths. The technique presented in [10,13] defines the net schedule concurrency level with a set of pairs of operations to be executed in parallel.

In this paper we propose and characterize techniques for optimization of parallel computations represented as net algorithms. Our objective is to develop a tool for optimizing the net algorithms derived from the transformed data flow cyclic computational model. First, we optimize the net algorithm parallelization level. Second, we solve the existence problem and generate a net algorithm that meets the constraints on execution time and implementation cost.

3. Net algorithms

We derive the net algorithms from the One Basic Block Model (OBBM) of a specified behavior proposed in [12]. In VHDL, the OBBM is generated for every process by means

Proceedings of the International Conference on Parallel Computing in Electrical Engineering (PARELEC'02)
0-7695-1730-7/02 $17.00 © 2002 IEEE

```
entity GCD is
   port(XP,YP: in Integer;
        RES: out Integer)
end GCD;

architecture GCD_BEHAVIOR of GCD is
begin
 GCD_OBBM: process
    variable X, Y : Integer:=0;
    variable C0: Boolean:=True;
    variable C1: Boolean;
 begin
  if C0 then  X := XP; Y:=YP;  end if;
  C0:=X=Y;
  if C0 then  RES<=X;  wait on XP,YP; end if;
  if not C0 then
     C1:= X < Y;
      if C1 then Y:=Y-X; end if;
      if not C1 then X:=X-Y; end if;
   end if;
 end process GCD_OBBM;
end GCD_BEHAVIOR;
```

Figure 1. OBBM of GCD sequential algorithm

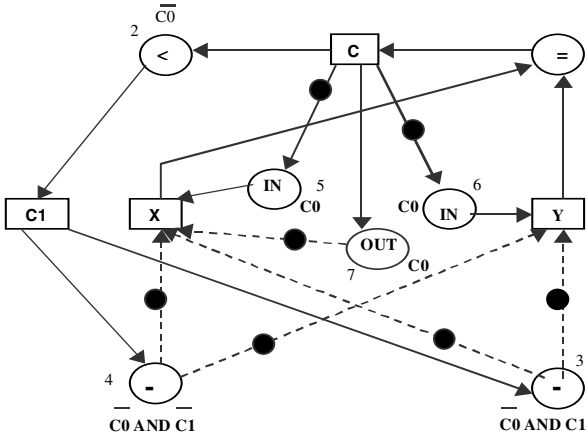

Figure 2. GCD net algorithm

of eliminating the loop, exit, next, function and procedure call statements. The transformed process is constructed as a single unbounded loop and consists of assignment, wait and conditional statements.

The net algorithm is represented by a directed labeled graph constructed on the set of variables and set of operators. The graph is initially generated using a handshake mechanism. Request and acknowledge edges that can be labeled by tokens are used for this purpose. The graph nodes may be labeled by conditional expressions defining whether the operator or variable assignment is executed or not. The graph operates through firing nodes. A time delay associated with a node depends on the value of conditional expression. The graph can account for the data and resource flow, which meets the time and cost constraints.

The VHDL-based OBBM of a GCD algorithm presented in Fig.1 is mapped to the net algorithm shown in Fig.2. The net algorithm graph consists of seven operator-nodes, four variable-nodes, twelve requests, and five acknowledge edges. Six operator-nodes are labeled with conditional expressions and seven edges are initially labeled with tokens. Input operators 5 and 6 assign the values of ports XP and YP to the variables X and Y. Output operator 7 assigns the value of variable X to the port RES.

The net algorithm execution time is estimated as the maximal clique weight of the graph $G_{\sim D}=(N, \sim D)$ defining the sequential execution of nodes, where N is the set of operator-nodes and $\sim D$ is the complementation of set D of concurrent node pairs. The net algorithm implementation cost is estimated through the maximal clique weights of the graph $G_{\sim D}=(N, D)$ defining the concurrent execution of nodes.

4. Parallelization flow

The parallelization flow, we have developed, is presented in Fig 3. First, a sequential algorithm is translated to the Data-Control Flow Graph (DCFG). Then, two types of transformation are applied to the algorithm: the transformation of data flow and the transformation of control flow. The transformation results in the OBBM of the behavior.

The optimization of parallelization level is treated as looking for a set of pairs of operators that can be executed in parallel and minimize the execution time or implementation cost.

To solve the existence problem is to check, if any net algorithm exists which realizes the generated set of operator pairs. The objectives of the net algorithm synthesis are the (1) generation of the flow relation, (2) labeling of nodes with conditional expressions, and (3) initial marking of edges with tokens.

The transformation of the net algorithm is performed by merging compatible nodes and by folding the graph.

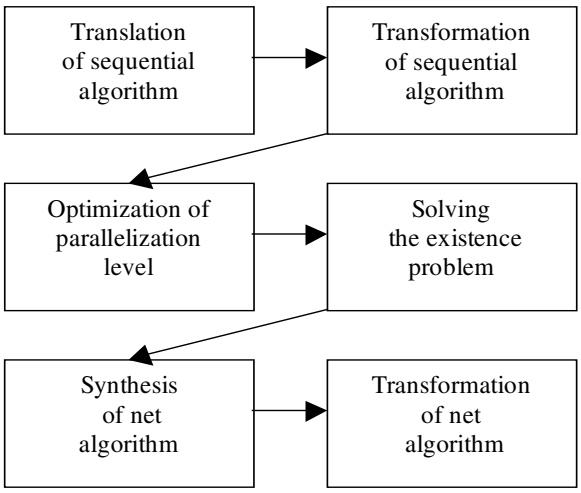

Figure 3. Parallelization flow

5. Optimization of parallelization level

Depending on the optimization criterion, we have developed two techniques $A1_D^{optim}$ and $A2_D^{optim}$ (Fig.4) for optimization of the net algorithm parallelization level.

Technique $A1_D^{optim}$. To minimize the execution time T_D (Fig.4a) we start with the empty set $D=\varnothing$ of pairs and step by step choose a pair from the maximal set D_M and add it to the set D. We break the process when the actual cost C_D of the algorithm implementation becomes greater than the bounding cost C_0.

Technique $A2_D^{optim}$. To minimize the implementation cost C_D (Fig.4b) we start with the maximal set $D=D_M$ of pairs and step by step choose and remove pairs from the set D. We break the process when the actual execution time T_D becomes greater than T_0.

It is proved in [10], such sets D exist for which no net algorithm can be constructed. For this reason, we solve the existence problem and modify the set through using the techniques $A1_D^{exist}$ and $A2_D^{exist}$.

Technique $A1_D^{exist}$. We start with the set D generated by the technique $A1_D^{optim}$ and step by step remove pairs from D (Fig.4a). We break the process when the current set D allows for the generation of a net algorithm.

Technique $A2_D^{exist}$. We start with the set D generated by the technique $A2_D^{optim}$ and step by step add pairs to D (Fig.4b). We break the process when the generation of a net algorithm is possible for the current set D.

Fig. 5 represents the parallelization process in the time-cost space, when the execution time T_D is being minimized. The time, first, decreases from T_{max} to T_{optim}, and then increases to T_{exist}. The cost, first, increases from C_{min} to C_{optim} and then decreases to C_{exist}.

Fig. 6 represents the parallelization process in the time-cost space, when the implementation cost C_D is being minimized. The cost, first, decreases from C_{max} to C_{optim} and then increases to C_{exist}. The time, first, increases from T_{min} to T_{optim}, and then decreases to T_{exist}.

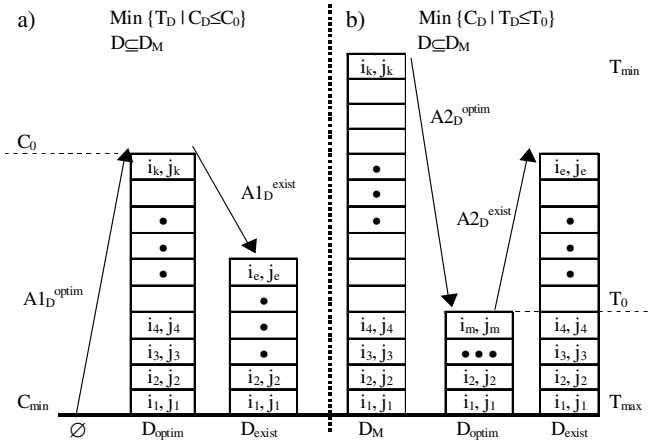

Figure 4. Searching for a parallelization level of the net algorithm

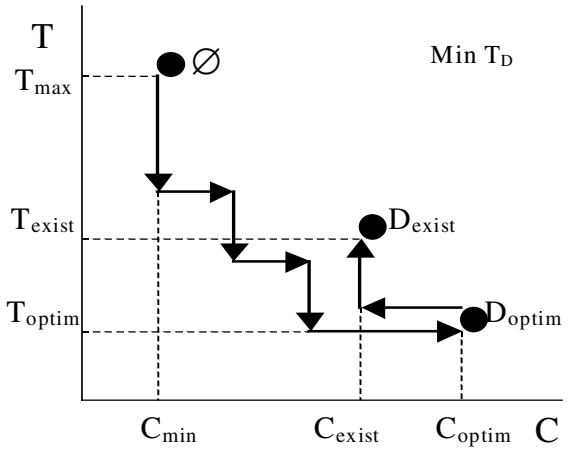

Figure 5. Parallelization process in the *time-cost* space when minimizing the execution time

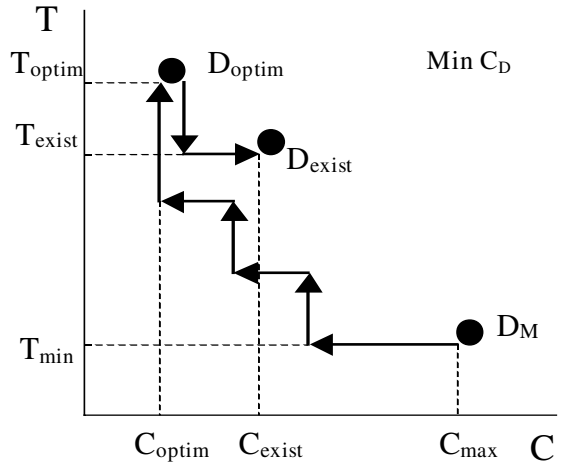

Figure 6. Parallelization process in the *time-cost* space when minimizing the cost

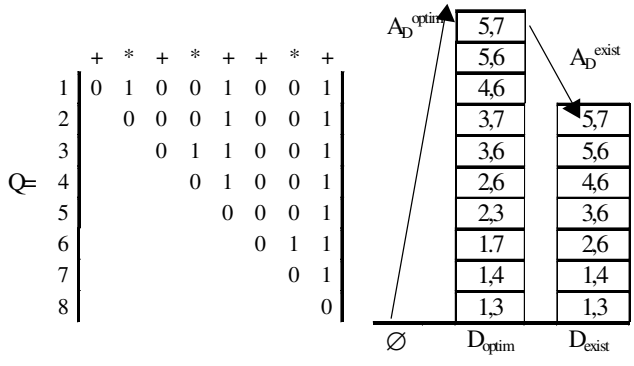

Figure 7. Optimization of parallelization level of an algorithm with 8 operators including 5 additions and 3 multiplications on 2 adders and 1 multiplier

An example optimization of the parallelization level is shown in Fig.7. The matrix Q describes the maximal parallelization potential of the behavior. In the matrix, an element q_{ij} equals *0* if operators i and j are executed in parallel, and equals *1* if the operators are executed sequentially.

6. Optimization techniques

In this section, our objective is to generate an optimal or quasi-optimal set D. We propose three techniques for the selection at each step of an appropriate pair to be added to or removed from the set D.

Technique MDT. It aims at the *maximal decrease of the execution time T_D and implementation cost C_D* during the set D generation process. When the pair is being added, the technique chooses a pair splitting a maximal weight clique of the sequential operator graph $G_{\sim D}$ without possibly increasing the maximal clique weight of the parallel operator graph G_D. When the pair is being removed, the technique chooses a pair splitting a maximal weight clique of the parallel operator graph G_D without possibly increasing the maximal clique weight of the sequential operator graph $G_{\sim D}$.

The following two techniques TRT and TCT preliminary order the operators and associated with them rows and columns of the matrix Q_D. Two values associated with each operator are used for the ordering:

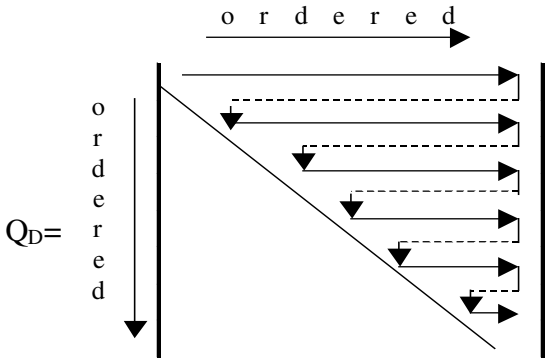

Figure 8. *Traversal of the rows* technique (TRT)

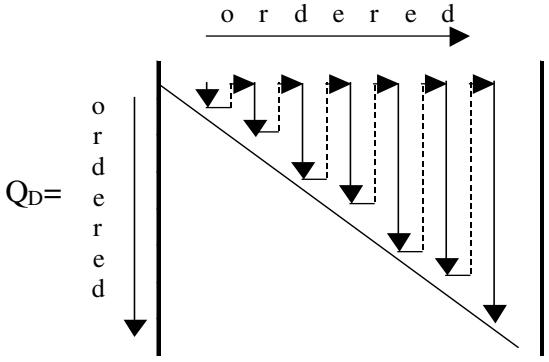

Figure 9. *Traversal of the columns* technique (TCT)

- The ratio "maximal clique weight of graph $G_{\sim DM}$ to maximal clique weight of graph G_{DM}"; only the cliques including the operator are taken into account.
- The difference "latest start time to earliest start time" for the operator.

Technique TRT. It is based on the *traversal of the rows* of matrix Q_D (Fig.8). It looks through the rows from the first to the last and at each step adds to or removes from the set D a pair of operators that can be executed in parallel and does not imply the exceeding time and cost. The technique generates the set D with the maximal (minimal) number of parallel operator pairs.

Technique TCT. It is based on the *traversal of the columns* of matrix Q_D (Fig.9). It looks through the columns from the first to the last and at each step adds to or removes from the set D a pair of operators. The technique preferably parallelizes operators belonging to the maximal weight cliques. Due to this strategy, the technique tries to reduce the critical paths of the net algorithm to be synthesized.

7. Existence problem

After optimizing the set D, we construct a matrix Q^x_D (Fig.10) from the matrix Q_D. The elements q_{ij} and q_{ji}, $i<j$ of the matrix equal *1* and *0* respectively when $(i,j)\notin D_M$. In this case the operators i and j are executed sequentially in such a way as the operator j is executed after the operator i. The elements equal *0* when $(i,j)\in D$. In this case, the operators i and j are executed in parallel. The elements equal x_{ij} and $\sim x_{ji}$ respectively when $(i,j)\notin D_M \backslash D$. Here x_{ij} is a Boolean variable and ~ denotes a logical negation operation. In this case, the operators i and j are executed sequentially, but it is not known which of them is a predecessor and which of them is a successor. The variable x_{ij} takes value *1* when i is a predecessor of j, and takes value *0* when j is a predecessor of i.

Fig.10 presents an example matrix Q^x_D. The following pairs of operators are executed sequentially in such a way as the first operator precedes the second one: {(1,2),(1,5), (1,8),(2,5),(2,8),(3,4),(3,5),(3,8),(4,5),(4,8),(5,8),(6,7), (6,8), (7,8)}. The pairs of parallel operators are: {(1,3),(1,4),(2,6), (3,6), (4,6), (5,6), (5,7)}. The unordered pairs of sequential operators are: {(1,6),(1,7),(2,3),(2,4),(2,7),(3,7), (4,7)}.

	+	*	+	*	+	+	*	+
1	0	1	0	0	1	x	x	1
2	0	0	x	x	1	0	x	1
3	0	x	0	1	1	0	x	1
Q^x_D =4	0	x	0	0	1	0	x	1
5	0	0	0	0	0	0	0	1
6	0	0	0	0	0	0	1	1
7	x	0	0	0	0	0	0	1
8	x	x	x	x	0	0	0	0

Figure 10. An example matrix Q^x_D

$$L_1 = \sum_{k<i} (x_{ki} \oplus x_{kj}) + \sum_{i<k<j} (x_{ik} \equiv x_{kj}) + \sum_{j<k} (x_{ik} \oplus x_{jk}) = 0$$
$$(k,i) \notin D \quad (i,k) \notin D \quad (i,k) \notin D$$
$$(k,j) \notin D \quad (k,j) \notin D \quad (j,k) \notin D, \text{ for } (i,j) \in D$$

$$L_2 = \sum_{i<k<j} (\sim x_{ij} \& x_{ik} \& x_{kj}) + (x_{ij} \& \sim x_{ik} \& \sim x_{kj}) = 0$$
$$(i,j) \notin D$$
$$(i,k) \notin D$$
$$(k,j) \notin D$$

$$L_3 = x_{ij} = 1, \text{ for } (i,j) \notin D_M$$

Figure 11. Formulation of the existence problem with a combined logical equation

To solve the net algorithm existence problem, we look for such values of the variables x_{ij} as the matrix $Q^x{}_D$ would describe a transitive relation. When the relation is transitive, the parallelization level of the net algorithm exactly equals *D*.

We formulate the transitivity requirement to the relation as a combined logical equation constructed on the matrix $Q^x{}_D$ and presented in Fig.11. In the equation, an OR-operation is denoted by +, an AND-operation is denoted by &, a XOR-operation is denoted by ⊕, an equivalence operation is denoted by ≡, and a negation operation is denoted by ~. We can consider the equations L_1 and L_2 as written in the form of sum of products. Every product that is evaluated to *1* is treated as a conflict. When we can find the values of variables x_{ij} for which the sum has no conflicts a net algorithm exists, otherwise we must modify the set *D* in such a way as to avoid the conflicts. We propose three branch and bound techniques for the set *D* modification.

Technique MNC. It minimizes the number of conflicts remaining in the equations L_1 and L_2 after removing some pairs from the set *D* (the execution time is being minimized) or adding some pairs to the set (the implementation cost is being minimized). Several pairs can be removed or added at one step. As a result, existing conflicts can disappear and new conflicts can appear. The objective of this technique is to quickly remove all conflicts from the equations.

Technique MNP. It minimizes the number of pairs removed from the set *D* or added to the set in order to avoid all conflicts. The technique finds out which of the pairs in set *D* implies the greatest number of conflicts. These pairs are being removed from or added to the set first of all. The objective of this technique is to maximize the number of pairs in the final set *D* and to generate a net algorithm with the maximal parallelization level at the bounded resources.

Technique MCPL. It minimizes the critical path length represented by the maximal clique weight of the graph $G_{\sim D}$ when the execution time is being minimized and of the graph G_D when the implementation cost is being minimized. At each step, the technique searches for the pairs that do not possibly imply the increase of the critical path length.

8. Experimental results

In order to find out which of the techniques are the most efficient, we have made several experiments. Results for the fifth order wave filter benchmark [4] are presented in Tables 1 to 4. Table 1 describes the benchmark and its parallelization potential.

Table 2 gives a comparison of three techniques optimizing the level of parallelization: TRT, TCT, and MDT. The TRT includes the maximal number of pairs in *D* (187) and produces the minimal number of conflicts (89), but the average clique weight (critical path) of 15.00 is the highest one. Instead, the MDT produces the minimal critical path of 11.65, but the number of pairs in *D* is also minimal and equals 155, and the number of conflicts is maximal and equals 279. After modifying the set *D* to solve the conflicts, we can conclude that the TCT technique is the most efficient because it produces the shortest critical and average paths at the same constraint on the implementation cost.

Table 3 presents results providing a comparison of the techniques MNC, MNP, and MCPL proposed for solving the existence problem. All the techniques start with the same initial set *D* for which no net algorithm exists. It is obvious, the MCPL is much more preferable compared to the MNC and MNP due to the maximal (19) and average (17.50) clique weights are significantly less for the MCPL technique.

Results of optimizing net algorithms for the filter at different constraint on the number of adders and multipliers are presented in Table 4. The TCT and MCPL techniques were used. When the parallelization level is being optimized, the number of pairs in the set *D* varies from 92 to 225. This implies variations in the maximal (from 26 to 17) and average (from 19.15 to 14.13) clique weights, and in the number of Boolean variables (from 144 to 11) and conflicts (from 271 to 13). To solve the existence problem, from 58% to 3% of pairs are removed from the set *D*. Due to the set reduction, the maximal and average clique weights slightly increase.

Table 1. Parameters of fifth-order wave filter

N	Parameter	Value
1	Number of operators	34
2	Number of additions	26
3	Number of multiplications	8
4	Number of operator pairs	561
5	Number of pairs of sequential operators	325
6	Number of pairs of parallel operators (D_M)	236
7	Execution time of addition	1
8	Execution time of multiplication	2
Net algorithm at maximal parallelization level:		
9	Cliques of graph $G_{\sim DM}$	12
10	Cliques of graph G_{DM}	636
11	Maximal clique weight	17
12	Average clique weight	13.33
13	Number of adders	5
14	Number of multipliers	3

Table 2. Generation of set *D* for the *FILTER*: 2 adders, 1 multiplier

Parameter	Technique		
	TRT	TCT	MDT
Optimizing set *D*			
Set D	187	184	155
Cliques of graph $G_{\sim D}$	26	30	62
Cliques of graph G_D	188	187	162
Maximal clique weight	20	20	20
Average clique weight	15.00	14.33	11.65
Boolean variables	49	52	81
Conflicts	89	107	279
Solving conflicts			
Maximal clique weight	20	20	23
Average clique weight	18.18	17.93	17.99

Table 3. Solving conflicts for the *FILTER*: 2 adders, 2 multipliers

Parameter	Technique		
	MNC	MNP	MCPL
Initial set D	Cardinality of D is 194, maximal clique weight is 17, average clique weight is 14.25, number of Boolean variables is 42, number of conflicts is 70		
Final set D	137	141	114
Boolean variables	99	95	122
Cliques of graph $G_{\sim D}$	24	22	44
Cliques of graph G_D	110	114	88
Maximal clique weight	22	22	19
Average clique weight	17.58	17.64	17.50

Table 4. Parameters of optimized net algorithms for the *FILTER*

Parameter	Value				
Adders	1	2	2	3	3
Multipliers	1	1	2	2	3
Optimizing set *D*					
Set D	92	184	194	224	225
Cliques of graph $G_{\sim D}$	13	30	24	18	16
Cliques of graph G_D	99	187	185	291	291
Maximal clique weight	26	20	17	17	17
Average clique weight	19.15	14.33	14.25	14.22	14.13
Boolean variables	144	52	42	12	11
Conflicts	271	107	70	16	13
Solving conflicts					
Set D	39	103	114	210	219
Pairs removed from D	58%	44%	41%	6%	3%
Cliques of graph $G_{\sim D}$	76	116	44	20	16
Cliques of graph G_D	47	83	88	234	315
Maximal clique weight	28	20	19	18	17
Average clique weight	22.58	17.93	17.50	15.10	14.75
Boolean variables	197	133	122	26	17

7. References

[1] Berkel K., Burgess R., Kessels J., Roncken M., Schalij F., Peeters A., *Asynchronous Circuits for Low Power: A DCC Error Corrector*, IEEE Design & Test of Computers, Summer 1994, pp.22-32.

[2] Faou C., Mermet J., *Introducing CASCADE control graphs in VHDL*, VHDL for Simulation, Synthesis and Formal Proofs of Hardware, J.Mermet (Ed.), Kluwer Academic Publishers, 1992, pp.291-307.

[3] Furber S., *Asynchronous Design*, Low Power Design in Deep Submicron Electronics, W.Nebel and J.Mermet (Ed.), Kluwer Academic Publishers, 1996, pp.461-492.

[4] Hwang T., Lee J., Hsu Y., *A Formal Approach to the Scheduling Problem in High- Level Synthesis*, *IEEE Trans.on CAD*, Vol.10, No.4, 1991.

[5] Jordan A.,Butrylo B. *Parallel Computations of Electromagnetic Wave Propagation*, Proc. PARELEC'98, BTU, Poland, 1998, pp.296-300.

[6] Kwong Y.-K., Ahmad I,, *Dynamic Critical-Path Scheduling: An Effective Technique for Allocating Task Graphs to Multiprocessors*, IEEE Trans. Parallel and Distributed Systems, vol 7, N 5, 1996, pp.506-521.

[7] Laskowski E., Tudruj M., *Program Graph Partitioning for Execution by Processor Clusters with Look-Ahead Connection Reconfiguration*, PARELEC'98, Poland, 1998, pp.94-99.

[8] Mozipo A., Massicote D., Quinton P., Risset T. *Automatic Synthesis of a Parallel Architecture for Kalman Filtering Using MMAlpha*, PARELEC'98, Poland, 1998, pp.201-206.

[9] Peterson J. *Petri Net Theory and the Modeling of Systems*. Prentice-Hall, Inc., N.J., 1981.

[10] Prihozhy A. *Net Scheduling in High-Level Synthesis*, *IEEE Design & Test of Computers*, Spring, 1996, pp.24-33.

[11] Prihozhy A., Merdjani R., Iskandar F., *Automatic Parallelization of Net Algorithms*, Proc. PARELEC' 2000, Canada, *IEEE CS Press*, CA, 2000, pp.24-28.

[12] Prihozhy A., *High-Level Synthesis Through Transforming VHDL Models.* in Book "System-on-Chip Methodologies and Design Languages", Kluwer Academic Publishers, 2001, pp.135-146.

[13] Prihozhy A., Solomennik M., *Generating Concurrent Net Schedules from Sequential VHDL Models*, Proc. FDL'2001, France, 2001, ECSI.

[14] Rammig F., *Approaching System Level Design,* in Book "VHDL for Simulation, Synthesis and Formal Proofs of Hardware", J.Mermet (Ed.), Kluwer Academic Publishers, 1992, pp.259-276.

[15] Ravasi M., Mattavelli M., Mlynek D., *Scheduling Strategies for 2D Wavelet Coding Implementations*, Proc. ISCAS'2000.