Proceedings of the International Multiconference on Computer Science and Information Technology pp. 771–777
ISBN 978-83-60810-22-4
ISSN 1896-7094

New Conception and Algorithm of Allocation Mapping for Processor Arrays Implemented into Multi–Context FPGA Devices

Piotr Ratuszniak, Oleg Maslennikow
Technical University of Koszalin
Department of Electronics and Informatics
ul.Śniadeckich 2, 75-543 Koszalin, Poland
Email: {ratusz, oleg}@ie.tu.koszalin.pl

***Abstract*—In the paper authors present new concept of realization of algorithms with regular graphs of information dependencies, in form of systolic arrays realized in multi-context programmable devices. Processor matrix efficiency depends on both allocation and schedule mapping. Authors use evolution algorithms and constraint programming to determine allocation mapping and optimize runtime of set algorithm. Authors compared the runtime of Cholesky's algorithm for banded matrices in which the new concept has been used with ones obtained by use of linear and non-linear allocation mapping for processor matrix.**

I. Introduction

MODERN VLSI technology allows placing a whole SoC (System-on-Chip) in one programmable unit. Some of the SoC system modules (especially modules containing specialized processing units) should be created as programmable FPGA units, which the main advantage is the possibility of fast and multiple changes in its internal structure in order to make the processing most efficient. Additionaly realization of SoC systems (or its fragments) in form of FPGA units allows to lower power drain, because highly efficient calculations can be realized through parallel hardware processing, and appropriate reconfiguration can be used dynamically in the background during calculations [1]. One of the models of parallel architectures created for linear algebra algorithms, is a parallel architecture model with a virtual topology. FPGA systems are effective hardware platforms for realization of such architectures also under efficiency and price criteria. However to achieve high efficiency, including high operating frequency a rule of locality of connections must be fulfilled. Examples of parallel architectures with regular network of local connections are processor matrix architectures [2], in which a calculations are realized in systolic path [3,4,5].

In this paper we propose a new concept of parallel processor architecture, similar to the systolic processor array, in which every processing element realizes the amount of operations corresponding to only one vertex of algorithm dependence graph realized by the unit. Realization of this concept allows total elimination of the control unit in a parallel processor and limiting the runtime of the algorithm to the value compared with the value computed through critical path of its information dependency graph. In order to optimize runtime and to automate the design process of new architectures a constraint programming was used, and also an evolutionary program was written in which designs appropriate allocation mapping of input algorithm graph into parallel processor architecture (through decomposition of the graph mentioned above into subgraphs). Evolutionary algorithm operate based on given volume of the target FPGA unit and hardware complexity of each graphs vertex (expressed by amount of CLB blocks of FPGA unit), the main criteria of optimization is the minimal runtime of the given algorithm corresponding to the shortest (critical) path in all graph. Designed architecture is meant for realization in a multi-context programmable unit, which concept was used by different research teams, including polish ones [6-8]. Advantages of the designed concept and the mapping algorithm are shown for the parallel processor project which decomposes LLT symmetrical matrices based on Cholesky's algorithm.

An important issue in the process of designing SoC systems is efficiency and quality of this process, because of this the use of standard IP–Core [9] components is advised. Because of that, during designing of the said algorithm we put special attention to the possibility of full automation and assumed strict boundaries for the runtime. We worked on our own IP–Core generator (project JGEN[19]), which will be used for both logical and structural level designing of specialized parallel architectures, in which the described algorithm is used.

II. New concept of allocation mapping of regular algorithms implemented into multi-context FPGA devices

Multi-context FPGA unit contains [6-8] a certain number p of identical configuration memory CM blocks, in which each p different configurations (contexts) could be stored. However in every moment of time there can be only one context active (one CM block). Such CM organization allows for fast changes of the unit's configuration (in ideal form during one cycle of system clock), which in turn means

This work was not supported by any organization

that the configuration could be changed while the system is operational. The amount of cycles necessary for changes in unit configuration is relatively small when compared to the cycles required for realization of the said algorithm, that's because in the following part of the paper it was deliberately overlooked. The mentioned possibilities allow for another, more effective way of digital systems implementation in such FPGA units. The structure of the whole complex system can be divided for the p substructures of similar hardware complexity in such a way, that the calculation results given by a substructure "i" would be used as input data in substructure "i+1". In such case the system can be designed in much smaller (up to p times) p-context FPGA unit, where in the single CM blocks configurations of every substructure of the system are stored. During calculations, calling of an appropriate substructures is realized by activation of appropriate CM blocks [10]. Realization efficiency of the whole system in a multi-context unit depends on the way of partitioning the structure of the whole system into certain number of substructures, and the way of designing of these substructures which should ensure similarity of their hardware complexity while maintaining assumed efficiency of the unit. Existing methods of decomposition of the system's structure into s substructures [1, 6], work on the VHDL description level and aren't designed for dividing the architecture of parallel units. However in paper [1] it is proposed to design projects of s substructures based on algorithm information dependence graph which the unit will be realizing instead of trying to decompose the existing project of the parallel unit. This idea was realized in [8] by forming of 2-stage method of acquiring substructures of parallel unit with processor matrix architecture. Sadly, a drawback of the described method is that its first stage is realized heuristically, which does not allow automation. Besides that, acquisition of the processor matrix architectures is based on linear and non-linear space-time mapping, which doesn't guarantee the required unit efficiency. Relating to this, we propose the new concept of parallel unit architecture, in which every processing unit (EP) realizes only one vertex of the algorithm's graph. In this case of use of multi-context FPGA units, it allows for total elimination of the control unit and lines that pass control signals in parallel unit. It also minimizes algorithm runtime to that of the approaching minimal possible value (critical graph path).

The information dependency graph's decomposition for multi-context programmable units (figure 1) has several limits considering space mapping. First limits are conditions of locality and causality, it means that every operation needed to perform current operation must be executed directly in the previous of current context of the programmable unit. In practice it means acquiring arguments for current operation in the same or previous context (causality) and the data written in the memory for the time of changing of the unit's context must be remembered only for the next configuration (locality). Another limit of space mapping is the maximal size of single context (analogy to BPP — bin packing problem). The parameters for the proposed method will be as follows: maximal context size, estimated size of the structures which realize all kinds of operations occurring in said algorithm (measured in CLB) in the chosen arithmetic. In methods used for acquiring parallel architectures described in papers [5, 8] the designer defines allocation and schedule mapping function in linear or non-linear form (usually in form of vector) which does not allow for total automation of the process. The proposed method generates allocation mapping automatically, and also attributes the smallest possible number of clock cycle for singular operations (of this mapping) in which these operations will be executed. The main criteria of optimization is the minimal number of clock cycles needed for realization of set linear algebra algorithm in every context of the programmable unit for the different space mapping, assuming that all graph nodes are realized in single clock or the virtual cycle.

III. Genetic algorithm and constraint programming for graph partitioning

There are many algorithms designed for partitioning and vertex coloring [11], which also use genetic algorithms [12, 13]. It is difficult however to find a method for directed graphs with weights for nodes which would be the merge of the bin packing problem algorithm with the scheduling algorithm, and that this would fulfill earlier assumptions of volume, causality and locality. The method of information dependency graph decomposition, described in paper [8] does not allow for its automation, and additionally in the case of the use of linear or non-linear functions of allocation mapping it's difficult to achieve a set structure or parallel architecture. Some methods use graph transformations specific to the given algorithm, which allow bigger influence on the shape of the parallel structure and on the parameters of the processing elements, but these transformations cannot be generalized for different algorithms. For these reasons we decided to use the genetic algorithm, which allows define the structure of the designed parallel architectures and allows for full automation of the design and the optimization process. In the case of multi-context FPGA unit architectures a number of contexts, or a maximum size of substructure depending on a set unit model can be set. After the defining of the maximum size of substructure and hardware complexity of each kind of operations in the algorithm, the essential number of contexts can be calculated automatically. In practice a minimal number of contexts is calculated with an assumed margin, because during the projects syntheses for given FPGA unit, it's difficult to use all available programming space.

First trials of the use of genetic algorithms for partitioning information dependency graphs for linear algebra algorithms we presented in paper [10], however after further investigations a decomposition of information dependency graphs for bigger matrices of chosen linear algebra algorithms (for graphs with nodes number greater than 500) has proved to be problematic while using standard recombination operators (mutation and crossover). Some genetic algorithms used for the different decomposition tasks and other operations on graphs (including

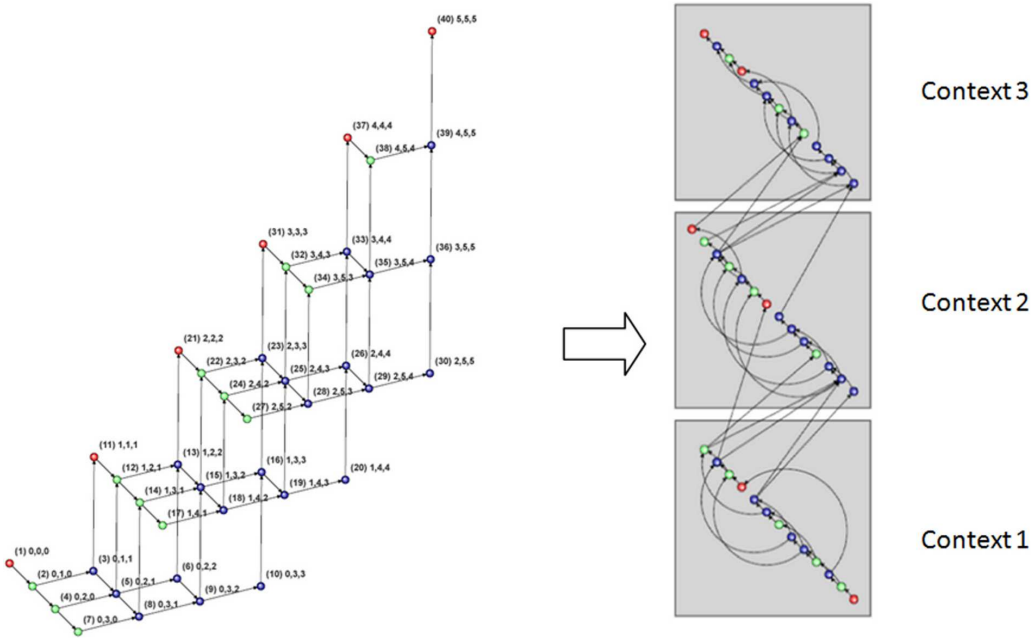

Fig. 1. Cholesky algorithm information dependency graph for band matrix (matrix size N=6, matrix band width L=4) and decomposition for 3 contexts FPGA Devices

operations with constraints) with different modifications of genetics operators are shown in paper [14], however it is stressed, that they were used for graphs of sizes limited to 100 or 900. In this paper we also propose certain modifications of the genetic operators and use of constraint programming for the generation of initial population, and present results of graph division containing even up to 2500 nodes, acquired in estimated time not exceeding 15min. To accelerate the runtime of the genetic algorithm (to get more generations in a set amount of time) we decided to make it parallel. Many methods of parallelizing of the genetic calculations were considered, for example supercomputers, GPU processors, GRID network, but in the end we created their own dispersed application of one client - multiple servers type designed for running on popular PC class computers [15]. In this application a model of genetic algorithm (similar to Island model) [16] has been used. A similar rule of independent populations was used, but there is a slightly different way of exchanging the best solution between clients and multiply servers. The copy of the best individual replaces the worst individual in all island. This exchange is realized in amount of time given by the user.

Parallel implementation increases the number of calculated generations and also allows for use of different modifications of the algorithm on different servers at the same time.

IV. Proposed Genetic Algorithm

A. Data representation

In the proposed algorithm we decided for coding the division groups with the use of numbers. Divisions are represented as integer chains of n-dimension, where n corresponds to the number of information dependency graphs nodes, and the range of numbers is limited from 0 to m-1, where m is the number of contexts of the programmable unit. Similar way of division graph coding is presented in paper [17]. At the figure 3 is an exemplary division and its representation for graph and contexts from figure 1 is shown.

$$[0\,0\,0\,0\,0\,0\,0\,0\,0\,1\,0\,0\,0\,0\,1\,1\,1\,1\,1\,1\,1\ldots$$
$$\ldots 1\,1\,1\,1\,2\,1\,2\,2\,2\,1\,2\,2\,2\,2\,2\,2\,2\,2\,2\,]$$

Fig. 3. Group coding with use of numbers for figure 1 graph

B. Generating of initial population

Two methods of generating the initial population were implemented in the program. In the first method values were generated in a random way (random assignment of graph nodes to contexts), and in the second one the initial population was generated by using constraint programming. The use of constraint programming was meant to generate permissible solutions in the shortest time frame, those solutions could be later optimized with use of the genetic algorithm. It allowed for shortening of the time in which the program was calculating permissible solutions. Exact results comparing the use of both methods are shown in the following part of the paper. The whole program was created using .NET Remoting technology on Microsoft .NET platform and that's why in the constraint programming module we decided to use a library designed by prof. Andy Chun's team named NSolver [18]. The module which generates the initial population with use of constraint programming finished its run after the experimentally chosen duration of 2 min or after generating a whole initial population of 100 individuals.

We investigated the use of two heuristics used for searching of the solution space: "Random" and "MinSizeMinValue"

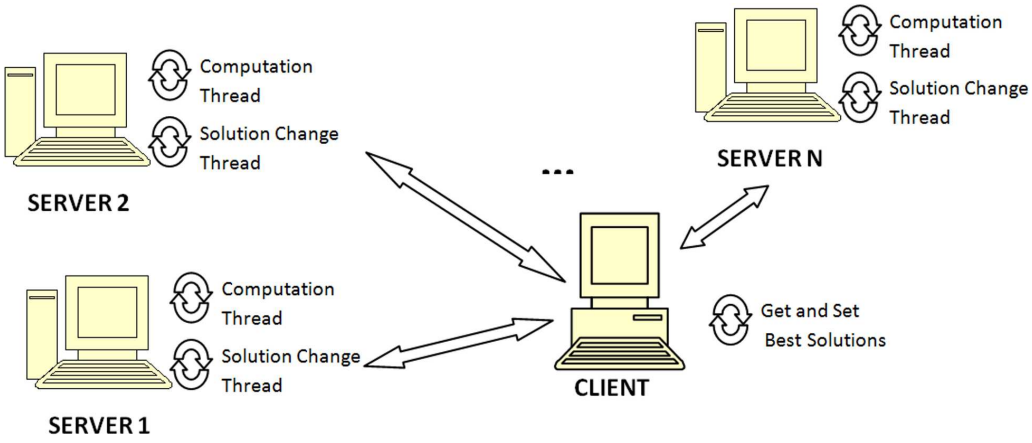

Fig. 2. Parallel implementation of genetic algorithm

heuristics. The second one was based on searching through the solution space from the smallest values for smallest indexes in the chromosome. Through experiments it was concluded, that a "Random" heuristic gives better results for smaller graphs (with nodes number less than 200) and heuristic "MinSizeMinValue" allows for faster acquisition of solutions for larger graphs. There was a problem for finding solutions for contexts number greater than 5, which could result from the binary representation of all graph nodes in all contexts, because of the limited number of operations on array variables. We conduct intensive research on the changes of representation of variables in the constraint programming module, which probably would allow in the near future for acquiring solutions for the number of contexts greater than 5.

C. Recombination operators

In the presented genetic algorithm a standard one point crossover operator with fixed probability was used, the value of probability was experimentally chosen at 0.2 level. Bigger influence on convergence had modifications introduced to mutation operator. At first a mutation with a fixed probability was used, which caused a different number of mutations for different graph sizes. With the constraints described earlier it caused a long time of coming to permissible solutions for larger graphs (with node number greater than 500). The first introduced standard modification was setting the probability of the mutation of a single position in chromosome in accordance to the size of the graph, so that the small number of genes would mutate (the change of assignment of context to the small number of nodes). Another change was the introduction of variable probability of mutation in accordance with the runtime of the algorithm [14]. The program started with such a probability of mutation, so that there would be a change in only one position per chromosome. Next, for following periods of time, if there wasn't an improvement of the best solution, the probability of mutation was raised in such way, that one more gene would change. The next modification included in mutation operator was limitation of range of values that a gene in chromosome could accept. The range of possible values for a given gene in chromosome was calculated basing on linear projection with a margin of (-1, +1), so called "window". The width of the "window" could rise with the runtime of the algorithm if there wasn't an improvement in the best solution. Described modifications allowed for much faster acquisition of permissible solutions (graph division that fulfilled constraints described earlier).

D. Objective function

The evolutionary algorithm work consists of two stages: the stage of finding the permissible solution and the latter stage of optimization. At stage one, for all individuals a number of unfulfilled rules of locality and causality is calculated, and so the allocation mapping is determined. Objective function F at this stage is dependant on locality, causality and overflow errors and is calculated according by equation (1),

$$F1 = 1 + EN(3EN - SE) + \\ +EN\frac{CN - OE}{CN} + EN\frac{GS - CL}{GS} \quad (1)$$

where:
EN - edge number in all graph,
SE - space projection errors, locality an casuality errors,
CN - contexts number of the multi-context FPGA device,
OE - context overflow errors, number of overflow contexts,
GS - graph size (CLB), hardware size of maximum parallel architecture assign for realization all operation in graph,
CL - context layout (CLB), maximum different between contexts size.

Based on experiments we also made objective function at this stage dependent on equal arrangement of processing elements in all contexts (in CLB), which caused faster minimizing of overflow errors. During second stage of the algorithm, after finding the permissible solution, the value of the objective function is calculated according to one of two equations depending on the permissibility of the individual. For the individuals that do not fulfill constraints (non-permissible) a function that gives much lower values was assigned (similarly to the penalty function), and for the individuals that fulfill the

constraints objective function additionally is dependant on the number of clock cycles needed to realize all contexts (whole graph).

In the second stage of genetic algorithm run, the value of objective function for individuals that fulfill the constraints (allocation mapping without locality, direction errors and contexts overflow) is calculated accordingly to equation (2),

$$F2 = 3EN^2 + CN * EN + 2EN - T \tag{2}$$

where T—tacts number(schedule mapping).

For the individuals that don't fulfill the constraints (non-permissible solutions) the value of the objective function is calculated accordingly to equation (3).

$$F3 = 3 - \frac{SE}{3EN} - \frac{OE}{CN} \tag{3}$$

Values of F3 function are always smaller than values of F2 function, which is an equivalent of the penalty function [17] for the individuals that do not fulfill the constraints.

E. Selection operator

In the presented algorithm, an elitist selection model was used in order to pass the best solution to the next population. In the developed model of the parallel genetic algorithm the best solutions are passed to all Island populations. Elite selection model ensures "keeping", of the best solution found, that fulfills all constraints (causality, locality and context size), which during the mutation process could be easily changed into solutions that do not fulfill all the constraints (non-permissible solutions). Elitist selection allows for keeping the best solutions, which has a strong influence in case of strict time boundaries for finding permissible solution.

F. Genetic algorithm termination criteria

Because of the targeted use of the presented method in the IPCore generator, which will be used for designing the parallel architectures for linear algebra algorithms (project JGEN [19]), strict time limits for a runtime were set. The maximal runtime of the algorithm was limited to 15min. The termination of calculations also occurred if there wasn't an improvement of the best solution for a time period longer than 5 min.
All calculatons were performed on PC class computer—Dell OptiPlex760 with Intel Core 2 Quad 2,2 GHz processor,4GB RAM and Microsoft Windws Vista Enterprise operating system.

V. Example. Cholesky LLT decomposition algorithm

This designed algorithm was used for the partition of information dependency graph for Cholesky's decomposition algorithm of banded matrices with band width of 3, 5 and 7, which are frequently occurring in practical numerical computations. The basic parameter of optimization was the number of clock cycles needed for realization of operations in all contexts of FPGA unit. In tables 1, 2 and 3 indispensable numbers of tacts acquired with linear (4) and non-linear (5) mapping for banded matrices received according to the method [8] are presented.

In this tables for comparison are presented results acquired with use of the proposed algorithm using evolutionary calculations with random initial population and initial population generated with use of constraint programming.

Based on results presented in table 1, one can conclude that results acquired with the use of the proposed algorithm are better than results acquired in paper [8]. It could also be concluded, that these are optimal results, because they are the same as the results for the critical path of the graph (minimal number of clock cycles with maximally parallel realization). For this relatively small space of possible combinations (max. NC, Nijnodes number=600, C-contexts number=7), algorithm has always finished its run before the 15min limit, after not being able to find a better solution in 5min. Results of graph decomposition for a matrix with band width equal to 5 are shown in table 2. Also in this case with the use of the proposed algorithm acquired results were better than those acquired with use of method [8] and close to the optimal values (critical graph path). In some cases generation of the initial population with the use of constraint programming caused acquiring of better solutions (for example for matrix size 40, 50) in comparison with random initial population. There were also cases of getting a worse solution (for example for matrix size 30, 60), probably because the acquired initial solution shifted search area to the local minimum, from which the algorithm didn't ¿get outř in a given time period. It should be stressed however, that constraint programming allowed for getting the permissible results in a shorter time, which was described in detail in the following part of the paper.

The next research objects were graphs for Cholesky's matrix with band width = 7. Results for decomposition of those graphs are shown in table 3.

Also in this case a significant advantage of the proposed algorithm over the linear and nonlinear space mapping described in [8] is clearly visible. It should be noticed, that the division was executed even for graphs as large as approximately 2700 nodes, and still the runtime was less than 15min. Based on results shown in tables 1, 2, 3 one can see, that with the increase of the matrix size and band width, and what corresponds ij the number of nodes in the graph, resulting mapping required more clock cycles to realize whole Choleksy's algorithm. But still the results were close to their optimal values (based on critical path).
Another area of research was the influence of the initial population generated with use of constraint programming on the time in which a permissible solution was found by genetic algorithm for a given problem. During research data considering the best solutions were gathered after 1, 2 and 3min from the beginning of calculations. The results of the comparison of the algorithm with random and generated with use of constraint programming population are presented in table 4. Based on the results show in table 4, one can conclude that with the use of very strict time limits results acquired with use of constraint programming for the generation of the initial

TABLE I
CHOLESKY ALGORITHM GRAPH PARTITIONING RESULTS FOR BANDED MATRIX (BAND WIDTH = 3)

Band width		**3**							
Matrix size		**30**	**40**	**50**	**60**	**70**	**80**	**90**	**100**
Graph nodes number		**172**	**232**	**292**	**352**	**412**	**472**	**532**	**592**
Graph edges number		**20885**	**27935**	**34985**	**42035**	**49085**	**56135**	**63185**	**70235**
Max. Context size (CLB) XC4VLX100= 12288 CLB		**12000**							
Contexts number		**2**	**3**	**3**	**4**	**5**	**5**	**6**	**7**
Min tacts (critical graph path)		**88**	**118**	**148**	**178**	**208**	**238**	**268**	**298**
Linear allocation mapping T1 (4)		117	157	197	237	277	317	357	397
Nonliner allocation mapping T2(5)		494	859	1324	1889	2554	3319	4184	5149
Realization time(tacts) (Gen.Alg.)	avg	88	118	148	178	208	238	268	298
	best	88	118	148	178	208	238	268	298
Realization time(tacts) (Gen.Alg.+Constr.Progr.)	avg	88	118	148	178	208	238	268	298
	best	88	118	148	178	208	238	268	298

TABLE II
CHOLESKY ALGORITHM GRAPH PARTITIONING RESULTS FOR BANDED MATRIX (BAND WIDTH = 5)

Band width		**5**							
Matrix size		**30**	**40**	**50**	**60**	**70**	**80**	**90**	**100**
Graph nodes number		**410**	**560**	**710**	**860**	**1010**	**1160**	**1310**	**1460**
Graph edges number		**28950**	**39050**	**49150**	**59250**	**69350**	**79450**	**89550**	**99650**
Max. Context size (CLB) XC4VLX160= 16896 CLB		**16000**							
Contexts number		**2**	**3**	**4**	**4**	**5**	**6**	**6**	**7**
Min tacts (critical graph path)		**88**	**118**	**148**	**178**	**208**	**238**	**268**	**298**
Linear allocation mapping T1 (4)		175	235	295	355	415	475	535	595
Nonliner allocation mapping T2(5)		494	859	1324	1889	2554	3319	4184	5149
Realization time(tacts) (Gen.Alg.)	avg	88,5	122,5	153,3	183,3	214,8	246	285	317,8
	best	88	120	152	182	214	245	285	316
Realization time(tacts) (Gen.Alg.+Constr.Progr.)	avg	89,0	120,8	152,3	187	214,8	246	285	317,8
	best	89	120	151	187	214	245	285	316

TABLE III
CHOLESKY ALGORITHM GRAPH PARTITIONING RESULTS FOR BANDED MATRIX (BAND WIDTH = 7)

Band width		**7**							
Matrix size		**30**	**40**	**50**	**60**	**70**	**80**	**90**	**100**
Graph nodes number		**728**	**1008**	**1288**	**1568**	**1848**	**2128**	**2408**	**2688**
Graph edges number		**39835**	**54385**	**68935**	**83485**	**98035**	**112585**	**127135**	**141685**
Max. Context size (CLB) XC4VLX200= 22252 CLB		**20000**							
Contexts number		**3**	**3**	**4**	**5**	**6**	**7**	**7**	**8**
Min tacts (critical graph path)		**88**	**118**	**148**	**178**	**208**	**238**	**268**	**298**
Linear allocation mapping T1 (4)		233	313	393	473	553	633	713	793
Nonliner allocation mapping T2(5)		494	859	1324	1889	2554	3319	4184	5149
Realization time(tacts) (Gen.Alg.)	avg	95,8	131,5	171,5	206,0	258,0	280,8	310,5	367,8
	best	95	130	171	205	258	280	309	367
Realization time(tacts) (Gen.Alg.+Constr.Progr.)	avg	97,0	129,0	167,0	206,0	258,0	280,8	310,5	367,8
	best	97	128	167	205	258	280	309	367

population are usually better, but this advantage lowers with the prolongation of computation runtime.

VI. CONCLUSIONS AND FUTURE TASKS

Based on conducted research, one can conclude, that genetic algorithms might be effectively used for the decomposition of graphs of chosen algorithms meant for realization in a multi-context programmable units. We managed to perform such modifications of the algorithm, that with the tight time limits (15min) it was possible to decompose graphs of even few thousand nodes. Proposed method gives much better results, in terms of shorter realization time of given algorithm, than methods described in [8], and can be totally automated. During future research we intend to modify the data representation and constraints construction in the initial population generator module, so that there would be more initially permissible solutions for a larger number of contexts. We intend to compare the our with the other hybrid genetic algorithms.

REFERENCES

[1] O. Maslennikow, “Podstawy teorii zautomatyzowanego projektowania reprogramowalnych równoległych jednostek przetwarzających dla jednoukładowych systemów czasu rzeczywistego.” Wyd. Uczelniane Politechniki Koszalińskiej, 2004, stron 273.

TABLE IV
BEST RESULTS OF PARTITIONING GRAPH AFTER 1,2 AND 3 MIN COMPUTING TIME FOR GENERATED AND RANDOM POPULATION.

Band width		**5**	**7**	**7**	**7**	**5**	**7**	**7**	**7**	**5**	**7**	**7**	**7**
Matrix size		**60**	**30**	**40**	**50**	**60**	**30**	**40**	**50**	**60**	**30**	**40**	**50**
Graph nodes number		**860**	**728**	**1008**	**1288**	**860**	**728**	**1008**	**1288**	**860**	**728**	**1008**	**1288**
Graph edges number		**59250**	**39835**	**54385**	**68935**	**59250**	**39835**	**54385**	**68935**	**59250**	**39835**	**54385**	**68935**
Contexts number		**4**	**3**	**3**	**4**	**4**	**3**	**3**	**4**	**4**	**3**	**3**	**4**
Max. Context size (CLB)		16000	**20000**	**20000**	**20000**	**16000**	**20000**	**20000**	**20000**	**16000**	**20000**	**20000**	**20000**
Min tacts (critical graph path)		**178**	**88**	**118**	**148**	**178**	**88**	**118**	**148**	**178**	**88**	**118**	**148**
Computing time		**1 min**				**2 min**				**3 min**			
Realization time (G.A.)	best	189	99	-	-	188	98	136	172	186	97	134	171
	avg	190,4	100,4	-	-	188,4	99,4	137,0	172,0	187,1	98,1	135,4	171,1
	find(%)	90	70	0	0	90	70	80	40	90	80	50	80
Realization time (G.A.+C.P.)	best	187	99	132	173	187	98	131	172	187	98	131	172
	avg	187,2	100,4	132,4	173	187,2	99,2	132	172,8	187,2	98,6	131,6	172,4
	find(%)	100	100	100	100	100	100	100	100	100	100	100	100

[2] Kung S. Y. “VLSI Array Processors”, Englewood Cliffs, N. J., Prentice Hall, 1988

[3] Kung H. T.,Leiserson C. E. “Systolic Arrays for VLSI”, Technical Report CMU-CS-79-103, Carnegie-Mellon University, Pitsburg, 1978

[4] Quinton P., Robert Y. “Systolic algorithms and architectures” Prentice Hall, Englewood Cliffs, 1991

[5] E. Lipowska-Nadolska, M. Kwapisz,K.Lichy “Systoliczne przetwarzanie sygnałów cyfrowych”. Akademicka Oficyna Wydawnicza EXIT, Warszawa 2007r.

[6] Canto E., Moreno J. M., Cabestany J., Lacadena I., Inserser J.M. A Method for Improving the Functional Density on Dynamically Reconfigurable Logic by Temporal Bipartitioning. Proc. 7-th Int.Conf. Mixed design of integrated circuits systems, MIXDES’2000, Gdynia, Poland, pp. 155–160.

[7] Kiełbik R., Moreno J. M., Napieralski A., Szymański T. “High-Level Partitioning for Dynamically Reconfigurable Logic.” Proc. 7-th Int.Conf. Mixed design of integrated circuits systems, MIXDES’2000, Gdynia, Poland, pp. 171–174.

[8] O. Maslennikow, “Realizacja architektur macierzy procesorowych w dynamicznie reprogramowalnych układach FPGA”, VII Krajowa Konferencja “Reprogramowalne układy cyfrowe”, RUC’2004, Szczecin, 2004, pp. 225–232.

[9] Fields C. “Design reuse strategy for FPGA’s”, Xcell Jurnal, Xilinx 2000

[10] Ratuszniak P., Maslennikow O., Sołtan p., Słowik A. “Zastosowanie algorytmów ewolucyjnych w projektowaniu równoległych jednostek przetwarzających do realizacji w wielokontekstowych układach FPGA”, V Krajowa Konferencja Elektroniki, Darłówko Wschodnie, 2006

[11] “Optymalizacja dyskretna. Modele i metody kolorowania grafów” Pod redakcją Marka Kubale, WNT, Warszawa, 2002

[12] Fleurent Ch., Ferland J. A. “Genetic and hybrid algorithms for graph coloring” Annals of Operations Research 63, 1996,s.437-461

[13] Galinier P., Hao J. “Hybrid evolutionary algorithms for graph coloring”, Jurnal of Combinatorial Optymization, 1999

[14] Michalewicz Z. “Genetic Algorithms + Data Structures = Evolutionary programs”, Springer-Verlag Berlin Heidelber, 1996

[15] Ratuszniak P., Bernatowicz.D. “Równoległa programowa realizacja algorytmów ewolucyjnych z wykorzystaniem technologii .NET Remoting” Prace XIV Konferencji Krajowej KOWBAN 2007, Szklarska Poręba, 2007

[16] Gordon V. S., Whitley D. “Serial and parallel genetic algorithms as function optimizers”, Proceedings of the fifth international Conference on genetic Algorithms, Illinois, 1993

[17] von Laszewski G.“Intelligent Structural Operators for the K-Way Graph partitioningProblem” Proceddings of the Fourth international Conference on Genetic Algorithms., San Mateo, CA, 1991

[18] H. W. Chun, “NSolver”, .NET constraint-programming software library,http://www.cs.cityu.edu.hk/~hwchun/ [Accesed Apr. 1, 2009]

[19] http://kik.weii.tu.koszalin.pl/mvl/jgen/