DOI: 10.15439/2015F313
ACSIS, Vol. 5

Synthesis of Power Aware Adaptive Schedulers for Embedded Systems using Developmental Genetic Programming

Stanisław Deniziak
Department of Computer Science
Kielce University of Technology
Kielce, Poland
Email: s.deniziak@tu.kielce.pl

Leszek Ciopiński
Department of Computer Science
Kielce University of Technology
Kielce, Poland
Email: l.ciopinski@tu.kielce.pl

***Abstract*—In this paper we present a method of synthesis of adaptive schedulers for real-time embedded systems. We assume that the system is implemented using multi-core embedded processor with low-power processing capabilities. First, the developmental genetic programming is used to generate the scheduler and the initial schedule. Then, during the system execution the scheduler modifies the schedule whenever execution time of the recently finished task occurred shorter or longer than expected. The goal of rescheduling is to minimize the power consumption while all time constraints will be satisfied. We present real-life example as well as some experimental results showing advantages of our method.**

I. Introduction

BESIDES the cost and performance, power consumption is one of the most important issue considered in the optimization of embedded systems. Design of energy-efficient embedded systems is important especially for battery-operated devices. Although the minimization of power consumption is always important, since it reduces the cost of running and cooling the system.

Embedded systems are usually real-time systems, i.e. for some tasks time constraints are defined. Therefore, power optimization should take into consideration that all time requirements should be met. Performance and power consumption are orthogonal features, i.e. in general, higher performance requires more power. Hence, the optimization of embedded system should consider the trade-off between power, performance, cost and perhaps other attributes.

Performance of the system may be increased by applying a distributed architecture. The function of the system is specified as a set of tasks, then during the co-design process, the optimal architecture is searched. Distributed architecture may consist of different processors, dedicated hardware modules, memories, buses and other components. Recently, the advent of embedded multicore processors has created an interesting alternative to dedicated architectures. First, the co-design process may be reduced to task scheduling. Second, advanced technologies for power management, like DVFS (Digital Voltage and Frequency Scaling) or big.LITTLE [1], create new possibilities for designing low-power embedded systems.

Optimization of embedded systems is based on assumptions that certain system properties are known. For example, to estimate the performance of the system, execution times for all tasks should be known. Sometimes it is difficult to precisely predict all required information. Therefore, to guarantee the proper design, the worst case estimation is used. During the operation of the system it may occur that certain system properties may significantly differ from estimations or may dynamically change. It may be caused by too pessimistic estimation, by data-dependence or by some unpredictable events. In such cases the idea of self-adaptivity may be used to optimize some system properties.

In this paper we present the novel method for synthesis of the power-aware scheduler for real-time embedded systems. We assume that the function of the system is specified using the task graph that should be executed by the multicore processor supporting the big.LITTLE technology. The scheduler is generated automatically using the developmental genetic programming (DGP). The scheduler is self-adaptive, i.e. it dynamically reschedules tasks whenever any task finished its execution earlier or later than expected. In the first case the goal of the rescheduling is the reduction of power consumption by moving some tasks to low-power cores. In the second case, the system is rescheduled to satisfy all time constraints by moving some tasks to high-performance cores. Example shows the benefits of using our methodology.

The rest of the paper is organized as follows. Next section presents the related work. Section III presents the concept of the developmental genetic programming with respect to other genetic approaches. In Section IV we present our method. Section V describes an example and experimental results. The paper ends with conclusions.

II. Related Work

Although there are a lot of synthesis methods for low-power embedded systems [2], the problem of optimal mapping of a task graph onto the multicore processor is rather a variant of the resource constrained project scheduling (RCPSP)[3] one, than the co-synthesis. Since the RCPSP is NP-complete, only heuristic approach may be applied to real-life systems. Among the proposed heuristics for solving RCPSP, ones of the most efficient are methods based on genetic algorithms [4][5][6].

For systems that may dynamically change during operation, some methods of rescheduling was proposed. In [7] the number of tasks that receives a new start time, after rescheduling, is minimized. Another approach [8] proposes to reschedule the remaining tasks, such that the sum of deviations of the new finishing times from the original ones is minimized. In [9] the sum of deviations of starting and finishing times of all tasks is minimized. Proactive scheduling [10] does not perform rescheduling, but it minimizes the perturbations, caused by delays, by maximization of the minimum or total free slacks of task executions.

Three different methods of applying big.LITTLE technology for minimizing the power consumption were proposed[11]. In the cluster switching, low-power cores are grouped into “little cluster", while high performance cores are arranged into “big cluster". The system uses only one cluster at a time. If at least one high performance core is required then the system switches to the “big cluster", otherwise the “little cluster" is used. Unused cluster is powered off. In CPU migration approach, low-power and high-performance cores are paired. At a time only one core is used while the other is switched off. At any time it is possible to switch paired cores. The most powerful model is a global task scheduling. In this model all cores are available at the same time.

The big.LITTLE technology is quite new and is mainly used in mobile devices. According to our best knowledge there are no applications of this technology to design low power real-time embedded systems, as well as an adaptive scheduling method for such systems.

III. Developmental Genetic Programming

Genetic algorithms (GA) [12] are very commonly used in wide spectrum of optimisation problems. Main advantage of GA approach is the possibility of getting out from the local minima of optimization criterion. Thus, GA is efficient for global optimization of complex problems, like RCPSP or multi-objective optimization of distributed real-time systems [13].

Although GA approach usually give satisfactory results it may be inefficient for hard constrained problems. In these cases a lot of individuals obtained using genetic operators correspond to not feasible solutions (e.g. schedule that exceeds required deadline or incorrect schedule, in the RCPSP). Such individuals should not be considered during the evolution. It is provided by defining the constrained genetic operators, that produce only correct solutions. But such constrained operators may create infeasible regions in the search space. Such regions may contain optimal or close to optimal solutions. This problem is illustrated on Fig. 1. Assume that we optimize the power consumption of the real-time system. Due to constraint violation, the solutions above the dotted line are not valid, hence they are never produced during the evolution. But it is possible that such “forbidden" individuals may be used

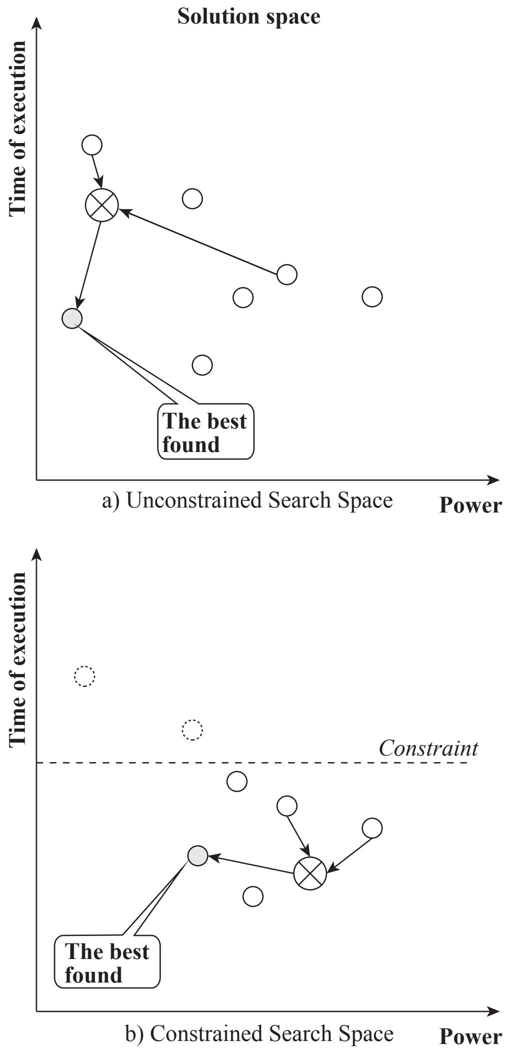

Fig. 1. Search space in GA

during the crossover or mutation to produce highly optimized solutions (Fig. 1a). Unfortunately, the search space should be limited to consider only valid solutions (Fig. 1b) and the optimal solution may never be obtained.

Above problem may be eliminated by using the Developmental Genetic Programming (DGP). DGP is an extension of the GA by adding the developmental stage. This method first time was applied to optimize analog circuits [14]. The main difference between DGP and GA is that in the DGP genotypes represent the method building the solution, while in the GA genotypes describe the solution. Thus, during the evolution, a method of building a target solution is optimized, instead of a solution itself.

In the DGP the search space (genotypes) is separated from the solution space (phenotypes). The search space is not constrained, all individuals are evolved. Thus, all of them may take part in the reproduction, crossover or mutation. There is no “forbidden" genotypes.

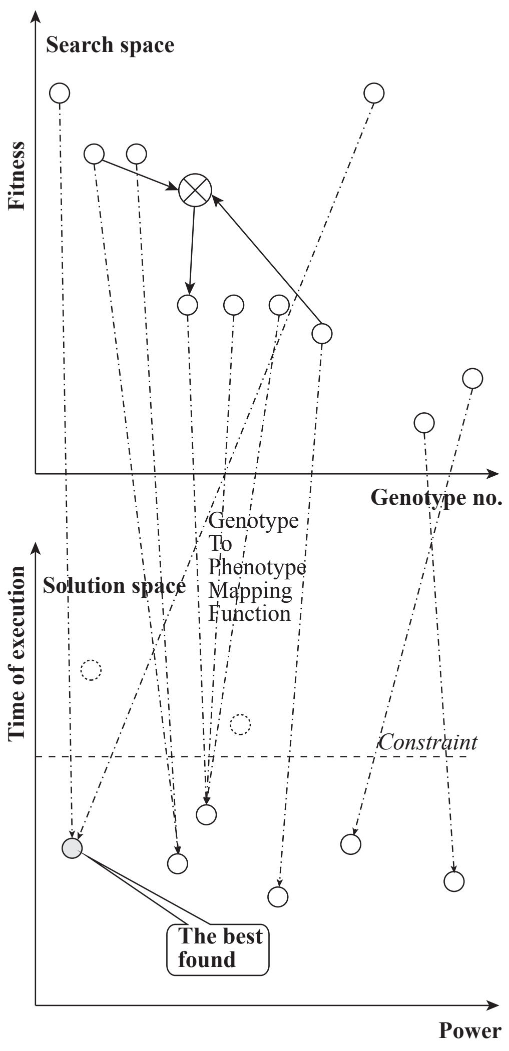

Fig. 2. Genotype to Phenotype mapping

Phenotypes are created by using genotype-to-phenotype mapping function, which always produces a valid solution. During the evolution, the fitness of the genotype is evaluated according to the quality of the corresponding phenotype. Fig. 2 presents the idea of the DGP. The search space consists of the genotypes that evolve without any restrictions. Any genotype may be mapped onto phenotype representing valid solution. This can be assured by using constrained mapping function. This function never produces solutions above the constraint line.

This idea of DGP is taken from biology, where the genotype corresponds to the chromosome containing information used for synthesis of proteins. The application of DGP occurred successful in many domains [15], where human-competitive results were obtained. High efficiency of the DGP-based optimization was also proved for hardware-software codesign[16] and cost minimization in real-time cloud computing[17].

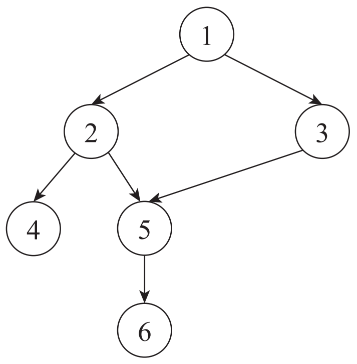

Fig. 3. A sample task graph

IV. Synthesis of Adaptive Scheduler

Idea of our approach is based on the observation that when the DGP will be applied for the RCPSP problem, then except the final schedule we also obtain the scheduler dedicated to the optimized system. Thus, instead of the implementation of static schedule we may implement this scheduler, which may adapt to any perturbation during the system operation. We assume that the system is specified as a task graph. This is very widely used method of specification of real-time embedded systems. We also assume that for each task, the time of execution and the average power consumption is known for each available processor core. Usually these parameters are estimated using the worst case estimation methods. During the system operation, the scheduler will dynamically modify the schedule to minimize the power consumption whenever it will be possible to move some tasks to low-power cores, i.e. when execution time of finished tasks will occur shorter than estimated. In our method it is also possible to use average execution time, instead of the worst case estimation. When same task will be delayed then the scheduler will try to find the new schedule that satisfies the time requirements. We use ARM multicore processors with big.LITTLE technology for implementation of the target systems. Such system consists of two processors, usually quad-core. The first of them has higher performance (about 40%), but consumes more power. The second one is slower, but it is optimized to use much less energy (about 75%), to execute the same task. The goal of optimization is to find the makespan for which the power consumption is as small as possible, while all time constraints are met.

A. Task Graph

The function of an embedded system is specified as a set of tasks. Between certain tasks may be the relationship that specifies the order of their execution. This may be specified as a task graph, which is the acyclic directed graph where nodes correspond to tasks and edges describe required order of execution. A sample task graph is given on Fig.3.

TABLE I
A SAMPLE LIBRARY OF RESOURCES

Task #	Core #	Execution time [ns]	Power Consumption [mJ]
1	0	537	5
1	1	537	5
1	2	537	5
1	3	537	5
1	4	671	3
1	5	671	3
1	6	671	3
1	7	671	3
2	0	1072	11
...	...	...	...
6	7	176	1

TABLE II
SCHEDULER'S PREFERENCES

Step	Option	P
1	a. The highest performance	0.16(6)
	b. The lowest power	0.16(6)
	c. The lowest time * power	0.16(6)
	d. Determination by second gene	0.16(6)
	e. The fastest starting core	0.16(6)
	f. The fastest finishing core	0.16(6)
2	List scheduling	1

B. Resources

Estimated execution parameters are given in a library of available resources. A resource is a core of a processor, which is able to execute a task. For each core the execution time and the power consumption are given. Part of a sample ARM Cortex-A15/Cortex-A7 database, for task graph from Fig.3, is presented in Table I.

C. Strategies of scheduling

The scheduler creates a makespan in two steps:

1) task assignment: tasks are assigned to cores according to preferences specified for each group of tasks (Table II),
2) task scheduling: this step is executed only when more than one task is assigned to the same core. During this step a selected group of tasks is scheduled using the scheduling strategy specified for this group (Table II).

Initial population consists of randomly generated genotypes. During initialization, preferences defining the decision table for the scheduler are assigned to each gene. Table II contains the set of possible preferences that the scheduler may choose. The last column in Table II shows a probability of the selection.

The first option prefers the core with the highest performance. Second one prefers a core with the lowest power consumption. Third option prefers a core with the best ratio of the power consumption to the time of execution. Fourth option allows using a core, that cannot be obtained as a result of the remaining options. The next option prefers a core, which could start an execution of the task as soon as possible (other cores might be busy). The last option prefers a core which could be the first to finish a task (be freed). For the second step only one option is available, the list scheduling method.

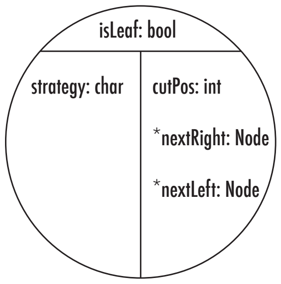

Fig. 4. A node of the genotype

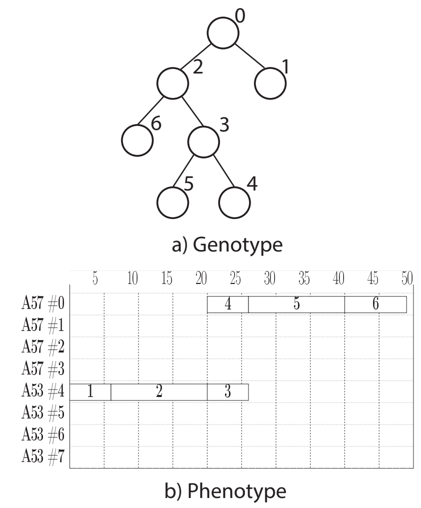

Fig. 5. A sample genotype (a) and the corresponding phenotype (b)

D. Genotype

The genotype has a form of binary tree corresponding to the certain procedure of task scheduling[18]. Every node in the genotype has the structure presented on Fig. 4.

The first field *isLeaf* determines a type of the node in a tree. When the node is a leaf this field equals true. Then, the field named *"strategy"* defines the strategy of scheduling for group of tasks assigned to this node. All possible strategies are given in Table II. In this case, information from the other fields is omitted. When the node is not a leaf, a content of the field *"strategy"* is neglected. In this case, *cutPos* contains a number describing which group of tasks should be assigned and scheduled by the left node and which one by the right one. Thus, *nextLeft* and *nextRight* must not be null pointers.

The simplest genotype consists of only one node, which is also a leaf and a root. A sample genotype and the corresponding phenotype are presented on Fig. 5.

During the evolution a genotype may grow up but the size of the tree is limited. If a tree will be too large, the performance of genotype to phenotype mapping, required for the fitness evaluation, would be slightly decreased. Size limit of the genotype also avoids constructing too many unused branches.

An initial genotype tree may grow up as an effect of genetic operators: mutation and crossover. An action associated with

TABLE III
THE RULES OF MUTATIONS

Is a leaf?			
Yes		No	
Draw: switch leaf/node or not?			
Yes	No	Yes	No
Set *isLeaf* as FALSE. If *nextLeft* or *nextRight* is NULL - create a new leaf for it.	Draw new strategy	Set isLeaf as TRUE	Change value for a randomly chosen field: *cutPos*, *nextLeft* or *nextRight*

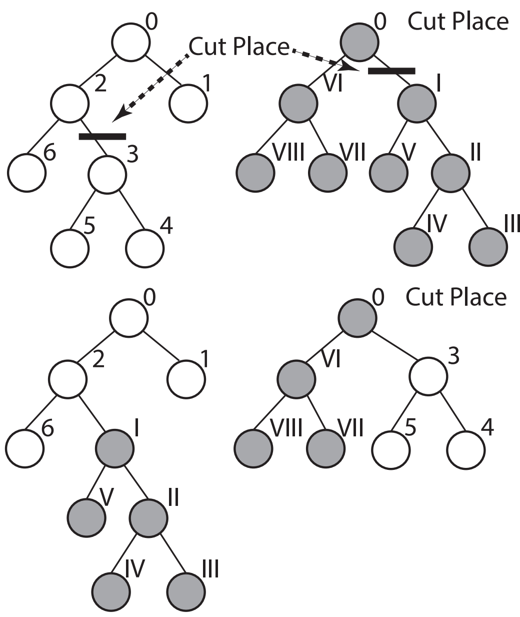

Fig. 6. An example of the crossover

the mutation depends on the type of the node and is presented in the Table III [18].

The crossover is used to create new individuals that are a combination of genes of parent genotypes. First, points of cut for both trees are drawn, then the cut branches are exchanged. In this way two new genotypes are created. A sample crossover is presented on Fig. 6.

With every genotype an array is associated. Its size is equal to the number of tasks and contains indexes of cores. If for given task, strategy 'd' is chosen, the core with an index taken from the array is used. During the mutation, a position in the array is randomly chosen. Then, a new index is randomly generated. During the crossover, parts of the arrays from both genotypes are swapped. The array defines the alternative scheduling strategy that is not driven by the performance or power consumption.

E. Genotype to phenotype mapping

The first step, during the genotype-to-phenotype mapping, is to assign strategies to tasks (i.e. preferences for assigning tasks to resources). For the example from Fig. 11, this step

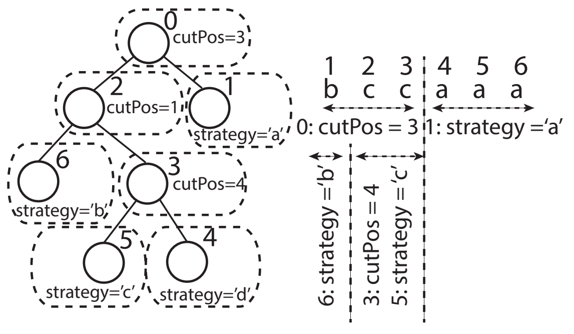

Fig. 7. The first step in genotype-to-phenotype mapping

is illustrated on Fig. 7. Node 0 groups tasks into two sets: {1,2,3} and {4,5,6}. The first set is partitioned by node 2 into next two groups. In the first one, there is only task 1. Tasks 2 and 3 belong to the second one, which is partitioned again by node 3. The *cutPos* parameter of node 3 equals 4, this means that tasks should be partitioned into group from 2 to 5 and the rest. But tasks 5 and 6 are outside of the set assigned to node 3, these tasks are assigned to node 1. Thus, there is only one group of tasks {2, 3} assigned to node 5.

In the second step, all tasks without any predecessor in the task graph, or with predecessors having already assigned core, are being searched for. These tasks are assigned to cores according to preferences determined in the first step. This step is repeated as long as there are tasks without assigned cores. In the third step, the total power consumption of the solution is calculated. For this purpose, the resource library (Table I) is used.

F. Parameters of DGP

During the evolution, new populations of schedulers are created using genetic operations: reproduction, crossover (recombination) and mutation. After the genetic operations are performed on the current population, a new population replaces the current one. The evolution is controlled by the following parameters:

- *population size*: the number of individuals in each population is always the same. The value of this parameter is determined according to the value of "number of tasks" * "number of cores",
- *reproduction size*: number of individuals created using the reproduction,
- *crossover size*: the number of individuals created using the crossover,
- *mutation size*: the number of individuals created using the mutation.

Finally, the selection of the best individuals by a tournament is chosen [12]. In this method, chromosomes (genotypes) are drawn with the same probability in quantity defined as a size of the tournament. The best one is taken to the next generation. Hence, the tournament is repeated as many times as the number of chromosomes for a reproduction, crossover and mutation is required. A size of the tournament should be defined carefully. It should not be too high, because the selection pressure is too strong and the evolution will be too greedy. It also should not be too low, because the time of finding any better result would be too long.

G. Self-adaptability of the scheduler

Finding the best makespan for low-power real-time embedded system is not the only goal of our approach. DGP methods are very effective in solving optimization problems and very often give the optimal solution. From the other side, they give results in relatively long time, thus genetic approach can not be used for rescheduling in real-time systems.

In the DGP the scheduling is performed during the genotype to phenotype mapping. This process is very fast, therefore it can be executed during the system operation. So, instead of implementing the final schedule we implement the method which creates this schedule. We observed that such approach has great self-adaptability capabilities.

Since the DGP has to consider only valid makespans. The genotype to phenotype mapping is a constrained process. If for a set of preferences defined by the genotype, it is not possible to obtain the valid phenotype then the mapping selects the next matching resource. Thus, the preferences specified by the genotype need not be strictly adhered. For example, if for given task preference suggest assigning this task to the low-power core, then if this decision will lead to an infeasible makespan, the scheduler will choose another, faster core that best matches to this preference.

Therefore, scheduler is able not only to build a correct solution, but also modify it if any unpredictable events will occur. E.g. if a task execution will be longer than expected, then the scheduler could move some tasks from a slower to a faster core, to fulfill time requirements. Similarly, if a task will be finished before its predicted end time, scheduler can move other tasks from a faster core to slower one, to save some energy.

H. Fitness function

A fitness function determines the optimization goal of the DGP. In the presented approach, two options are possible. In the first one, the cheapest solution which has to be finished before a deadline is searched for. Such fitness function is applied when hard real time constraints have to be satisfied. In the second one, the DGP should find the fastest solution, which does not exceed a given power consumption. This case concerns systems with soft real-time requirements.

V. Example and Experimental Results

We have verified advantages of the presented method using example of the complex multimedia system, which was described in [19]. The result has been compared with the method based on Least-Laxity-First Scheduling Algorithm [20].

A. Task Graph and Run-time Parameters

The sample system is a multimedia player implemented as a real-time embedded system. The specification of the system consists of 40 tasks. Fig. 8 presents the task graph describing the system, details are given in [19]. We consider shared memory architecture, thus the communication between tasks may be neglected. The execution time is critical - it is typical soft real-time system. If the deadline is only slightly exceeded, the quality of the system is decreased, but the solution may be accepted. If the system exceeds hard deadline, then result is unacceptable and system should be redesigned.

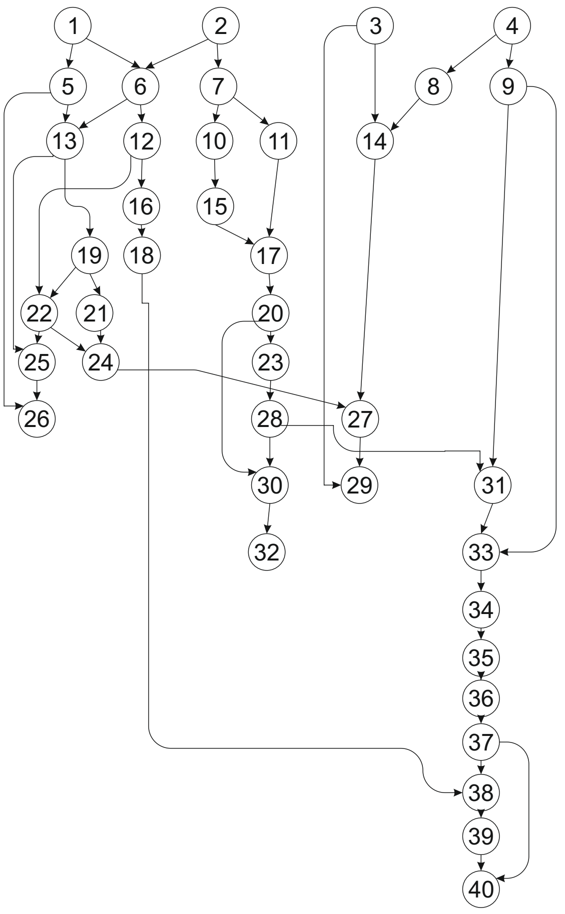

Fig. 8. Task graph of the multimedia system

We assumed, that the application will be implemented in software running on system consisting of two 4-core processors. One of them is ARM Cortex A57 and the second is ARM Cortex A53. The first processor is faster for about 25%, but consumes about 50% more power. Both processors support ARM big.LITTLE technology. Run-time parameters for both processors are given in Table IV.

The goal of our methodology is to create a self-adapting scheduler, which run the program tasks, balancing them between the cores, to minimize the power usage. The scheduler should be able to reschedule remaining tasks, whenever any task will finish its execution before or after expected time frame.

TABLE IV
EXECUTION TIME AND POWER CONSUMPTION

Task	**Processor cores**			
	A57 (high performance)		***A53 (energy efficient)***	
	energy	***time***	***energy***	***time***
1	5	537	3	671
2	11	1072	6	1340
3	5	537	3	671
4	4	376	2	470
5	73	7337	37	9171
6	11	1072	6	1340
7	110	10958	55	13698
8	74	7358	37	9198
9	11	1051	6	1314
10	6	559	3	699
11	5	486	3	608
12	3	286	2	358
13	13	1298	7	1623
14	37	3679	19	4599
15	21	2065	11	2581
16	53	5253	27	6566
17	75	7523	38	9404
18	11	1076	6	1345
19	4	409	2	511
20	4	409	2	511
21	11	1076	6	1345
22	2	157	1	196
23	260	26018	130	32523
24	2	176	1	220
25	2	197	1	246
26	260	26018	130	32523
27	236	23607	118	29509
28	6	559	3	699
29	11	1072	6	1340
30	110	10958	55	13698
31	5	486	3	608
32	3	286	2	358
33	11	1072	6	1340
34	4	409	2	511
35	4	409	2	511
36	236	23607	118	29509
37	74	7414	37	9268
38	3	253	2	316
39	2	179	1	224
40	2	176	1	220

B. Genetic parameters

We assumed that the deadline for the system from Fig.4 is equal to 100000 ns. The Power Aware Scheduler and the optimized makespan were generated using DGP. During the experiments, the following values of genetic parameters were used:

- the evolution was stopped after 100 generations,
- each experiment was repeated 7 times,
- the population size was equal to 128,
- tournament size was equal to 10,
- the number of mutants in each generation, was equal to 20%,
- the crossover was applied for creation of 40% genotypes,
- 20% of individuals were created using reproduction.

The values of parameters described above were tuned according to method described in our previous work [18], thus we will describe it here very shortly. In the first step, we estimated an influence of the tournament size. When this parameter was too small, the evolution got stuck. When the tournament size was too big, the DGP found semi optimal solution very fast, but a further optimization was not possible. Next, the influence of crossover and mutation for obtaining the best solution has been tested. It has been done by searching for the best solution using different combination of these parameters. Thus the best values of these parameters have been chosen. Finally, the best combination of other evolution parameters has been evaluated.

C. Least-Laxity-First Algorithm

One of the most known algorithms for scheduling tasks in real-time embedded systems is the Least-Laxity-First Algorithm (LLF). Basic LLF method schedules task according to the least laxity (slack time). The laxity is defined as a difference between an execution time and a task deadline. The goal of LLF is to find the schedule that satisfies all deadlines. It does not take into account power or cost optimization. Therefore, we modify this method by favouring energy-efficient cores. In other words, during scheduling, the method first tries assign a task to low-power core, only when it will violate the time constraint, the task will be assigned to more efficient core. Our Low Power LLF (LPLLF) method is used only for reference, to verify that the DGP is efficient also for power optimization in real-time embedded systems.

D. Power-aware Scheduling

The makespans obtained using DGP and LPLLF methods are presented on Fig. 9. On Y-axis different cores are represented, while the time of execution is represented by the X-axis. Numbers correspond to the following tasks. The experimental results proved that the presented method is more efficient than LPLLF. Energy consumption for the system scheduled using DGP equals 990mJ, while the same example scheduled using LPLLF requires 1018mJ. To meet the deadline, the LPLLF method assigned the long task 36 to the most efficient core. But in the DGP, more energy-efficient solution was found by assigning some shorter tasks, that in total consume less power than task 36, to the faster core.

Above experiment showed that the scheduler constructed using DGP is able to find highly optimized solutions. More experiments proving this remark are given in our previous work [16][17][18].

E. Self-adaptivity capabilities

Static scheduling is based on estimation of execution times for all tasks. During the system operation, time of execution may significantly be shorter (e.g. if the worst case estimation was applied) or longer (e.g. in case of the most likely estimation). Therefore, the system may be additionally optimized during run-time, by using self-adaptive scheduling. In this way certain system parameters (power consumption, performance) may be improved. Scheduler generated using our method consists of series of system construction functions, corresponding to each gene. These functions are flexible, i.e. design decisions

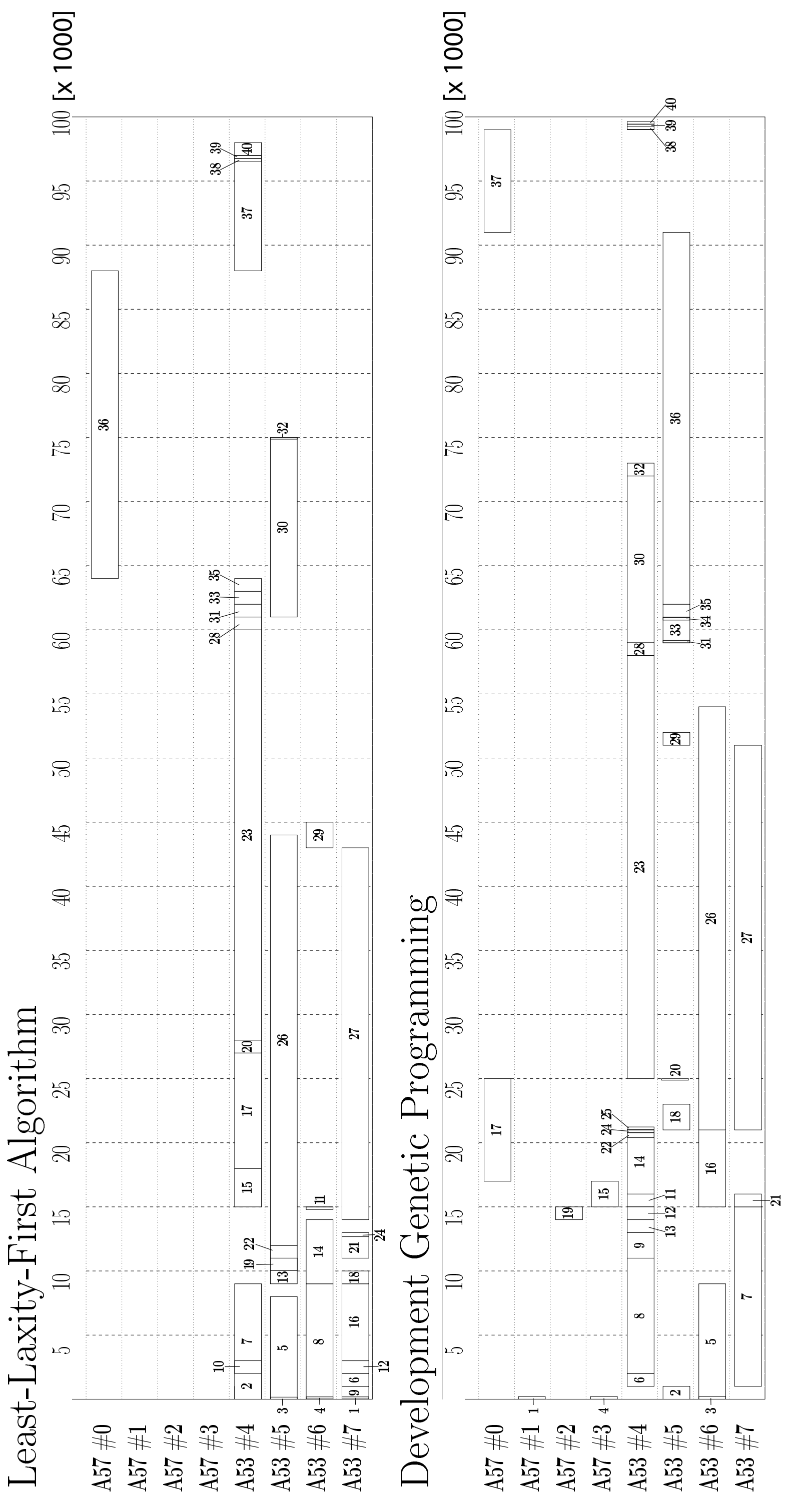

Fig. 9. Makespans obtained using LLF and DGP

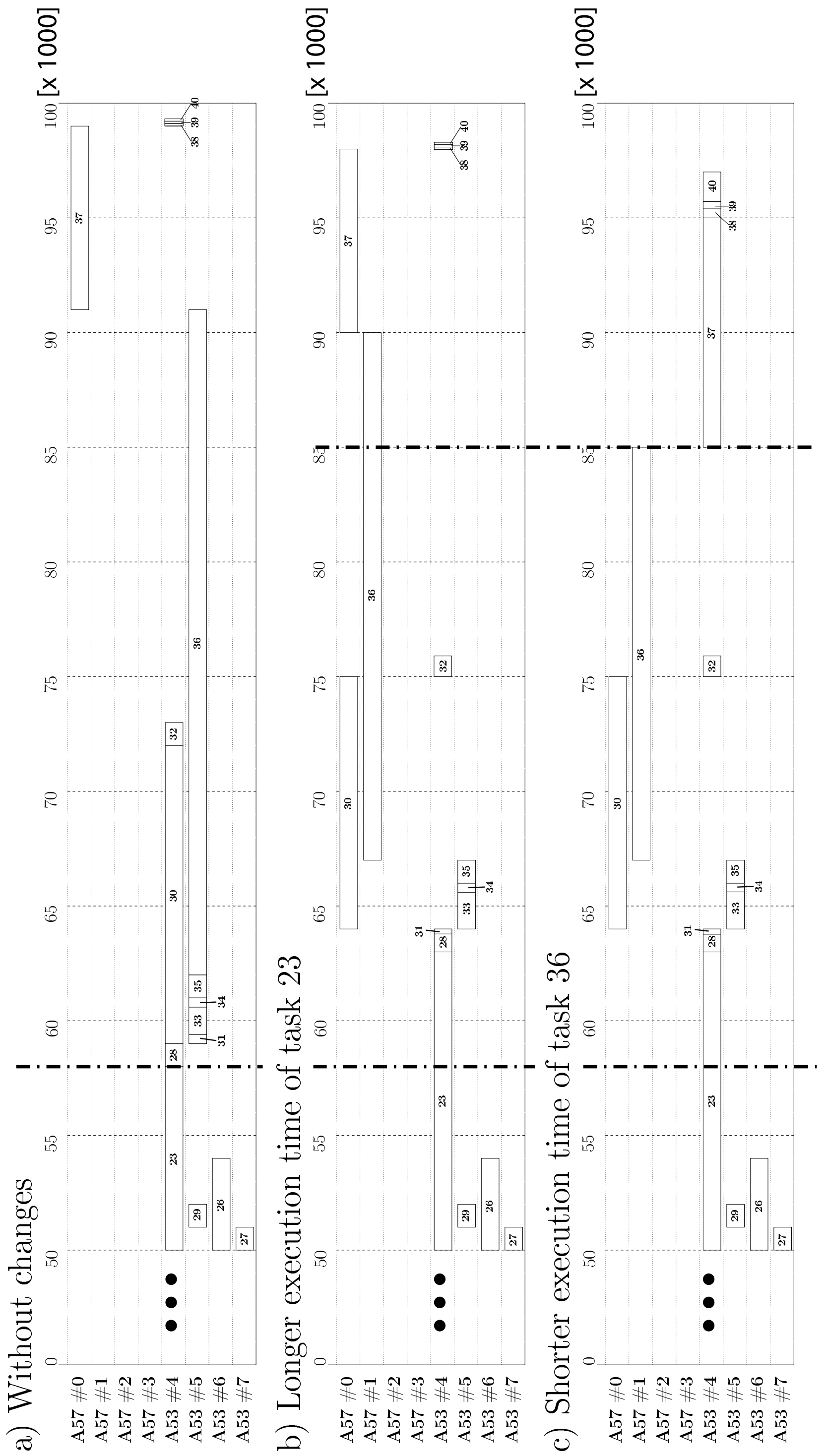

Fig. 10. Self-adaptation capabilities of power-aware scheduler

are driven by preferences, they are not strictly defined. This is necessary to assure that only feasible makespan will be created. Flexibility of the system construction functions provides to self-adaptivity capabilities of the scheduler.

An example of the self-adaptivity is presented on Fig. 10. If the execution time of task 23 will be too long, then all succeeding tasks should be postponed and the system would exceed the deadline. But our scheduler adapts to the delayed end time of task 23, and despite the fact that it uses the same construction functions, some tasks (tasks 30 and 36 in our example) will be assigned to more efficient cores (Fig. 10b). In other case, if an execution time of a task 36 will be shorter than it was expected, the scheduler will assign some tasks (task 37 in our example) to low power core (Fig. 10c). In this way power consumption will be reduced.

To verify the capabilities of self-adaptivity of the scheduler we performed some simulations of different changes in execution times for some task. Table V presents results obtained for cases when execution times occurred longer than estimated. In all cases our scheduler was able to adapt to this situation and new makespans that satisfied the deadline were created.

Table VI presents results obtained for another cases, where execution times for some tasks occurred shorter than expected, i.e. estimation was too pessimistic. In such case the scheduler has an opportunity to reduce the power consumption. It should be noticed that in some cases the scheduler found more energy-efficient makespans, but a lot of makespans were not changed. The main reason is that in the most cases there was not possible to improve the power consumption because all remaining tasks were already assigned to low-power cores. It is visible when we compare results obtained for different deadlines. When the deadline is shorter than 95 000 there is still a possibility for improvement. For longer deadlines, even when all tasks were faster it was not possible to decrease the power consumption.

VI. Conclusions

In this paper the method of automatic synthesis of power-aware schedulers for real-time distributed embedded systems was presented. Starting from the system specification in the form of the task graph, we use developmental genetic programming to optimize the scheduling strategy that minimizes the power consumption. Finally, the best makespan as well as the optimized scheduler are generated. The scheduler has powerful capabilities of self-adaptation. This feature may be used to dynamically minimize the power consumption as well as to increase the system performance.

The presented method is dedicated to ARM big.LITTLE technology, developed for a low power systems. But, since we use general optimization method, it would be easily adapted to other energy-efficient architectures.

The computational experiments confirmed, that the schedulers, generated using DGP, are efficient and flexible. For the sample system our method gave better results than results obtained using LLF-based method. Moreover, simulations showed that the scheduler is able to quickly and effectively react to any changes of task execution times, by rescheduling remaining tasks.

Despite the above advantages of our method, there is still possible to improve the methodology. In the future work, we will consider special types of adaptive genes, that could support more possibilities for self-adaptation, we will also consider using other scheduling methods, alternative to list scheduling, e.g. based on mathematical/constrained programming [21].

References

[1] big.LITTLE Processing with $ARMCortex^{TM}$ - A15 & Cortex-A7, ARM Holdings, September 2013, http://www.arm.com/files/downloads/big.LITTLE_Final.pdf.

[2] J.Luo, N.K. Jha, Low Power Distributed Embedded Systems: Dynamic Voltage Scaling and Synthesis, Proc. 9th Int. Conference High Performance Computing - HiPC 2002, Lecture Notes in Computer Science, vol. 2552, 2002, pp. 679-693. http://dx.doi.org/10.1007/3-540-36265-7_63

[3] Hartmann S., Briskorn D., A survey of variants and extensions of the resource-constrained project scheduling problem, European journal of operational research : EJOR. - Amsterdam : Elsevier, Vol. 207., 1 (16.11.), pp. 1-15 (2010). http://dx.doi.org/10.1016/j.ejor.2009.11.005

[4] Hartmann, S. (1998). An competitive genetic algorithm for resource-constrained project scheduling. Naval Research Logistics, 45(7), 733-750. http://dx.doi.org/10.1002/(SICI)1520-6750(199810)45:7%3C733::AID-NAV5%3E3.3.CO;2-7

[5] Xiang Li, Lishan Kang, Wei Tan, "Optimized Research of Resource Constrained Project Scheduling Problem Based on Genetic Algorithms", Lecture Notes in Computer Science, Vol. 4683, 2007, pp 177-186. http://dx.doi.org/10.1007/978-3-540-74581-5_19

[6] Hossein Zoulfaghari, Javad Nematian, Nader Mahmoudi, and Mehdi Khodabandeh. 2013. A New Genetic Algorithm for the RCPSP in Large Scale. Int. J. Appl. Evol. Comput. 4, 2 (April 2013), 29-40. http://dx.doi.org/10.4018/jaec.2013040103

[7] K.M. Calhoun, R.F. Deckro, J.T. Moore, J.W. Chrissis, J.C.V. Hove, Planning and re-planning in project and production scheduling, Omega The international Journal of Management Science 30 (3) (2002) 155-170. http://dx.doi.org/10.1016/S0305-0483(02)00024-5

[8] S. Van de Vonder, E.L. Demeulemeester, W.S. Herroelen, A classification of predictive-reactive project scheduling procedures, Journal of Scheduling 10 (3) (2007) 195-207. http://dx.doi.org/10.1007/s10951-007-0011-2

[9] H. Sakkout, M. Wallace, Probe backtrack search for minimal perturbation in dynamic scheduling, Constraints 5 (4) (2000) 359-388. http://dx.doi.org/10.1023/A:1009856210543

[10] M. Al-Fawzan, M. Haouari, A bi-objective model for robust resource-constrained project scheduling, International Journal of Production Economics 96 (2005) pp.175-187. http://dx.doi.org/10.1016/j.ijpe.2004.04.002

[11] Brian Jeff, "Ten Things to Know About big.LITTLE". ARM Holdings, 2013,http://community.arm.com/groups/processors/blog/2013/06/18/ten-things-to-know-about-biglittle

[12] Z. Michalewicz, Genetic Algorithms + Data Structures = Evolution Programs, Springer-Verlag Berlin Heidelberg, 1996. http://dx.doi.org/10.1007/978-3-662-03315-9

[13] Dick, R.P., Jha, N.K.: MOGAC: A Multiobjective Genetic Algorithm for the CoSynthesis of Hardware-Software Embedded Systems. IEEE Trans. on ComputerAided Design of Integrated Circuits and Systems 17(10), 920-935 (1998). http://dx.doi.org/10.1109/43.728914

[14] Koza, J., Bennett III , F. H., Andre, D., Keane, M. A., 1998. Evolutionary Design of Analog Electrical Circuits Using Genetic Programming. In: I. C. Parmee (ed.), Adaptive Computing in Design and Manufacture. http://dx.doi.org/10.1007/978-1-4471-1589-2_14

[15] J.R.Koza, R.Poli, "Genetic Programming", In Edmund Burke and Graham Kendal, editors. "Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques", Chapter 5. Springer, 2005. http://dx.doi.org/10.1007/0-387-28356-0_5

[16] S.Deniziak, A.Górski, "Hardware/Software Co-Synthesis of Distributed Embedded Systems Using Genetic Programming", Lecture Notes in Computer Science, Springer-Verlag, 2008, pp.83-93. http://dx.doi.org/10.1007/978-3-540-85857-7_8

TABLE V
SELF-ADAPTATION FOR DIFFERENT DELAYS

Case	Delay	Deadline [ns]	Time without resche-duling	Time after resche-duling	Time-out [%]	Energy [mJ]
0	(none)	90000	89756	-	0	1494
1	T2 + 53%	90000	90466	88612	0,52	1586
2	T8+20%	90000	91595	89741	1,8	1586
3	T17+3%	90000	90038	89994	0,04	1550
4	T28+36%	90000	90007	89964	0,01	1550
5	T36+1,3%	90000	90056	89967	0,06	1496
0	(none)	95000	94463	-	0	1275
1	T8 + 32%	95000	96817	93404	1,9	1312
2	T15 + 20%	95000	94903	92766	0	1265
3	T29 + 51%	95000	94903	94903	0	1275
4	T33 + 43%	95000	94995	95039	0,04	1276
0	(none)	100000	99846	-	0	990
1	T7: +30%	100000	103955	99772	3,95	1319
2	T14: +50%	100000	100765	99846	0,77	1238
3	T15: +20%	100000	100259	96211	0,26	1071
4	T17: +35%	100000	102479	98431	2,48	1071
5	T23: +15%	100000	104724	98822	4,72	1163

TABLE VI
SELF-ADAPTATION FOR DIFFERENT EXTRA TIMES

Case	Time decrease	Deadline [ns]	time	energy [mJ]	Energy without rescheduling [mJ]
0	(none)	90000	89756	1494	1494
1	T17 - 43%	90000	89987	1464	1494
2	T23 - 34%	90000	89649	1376	1494
3	T27 - 15%	90000	89756	1494	1494
4	T30 - 25%	90000	89756	1494	1494
5	T36 - 38%	90000	80785	1494	1494
6	All -30%	90000	88916	1116	1494
7	All -50%	90000	83398	998	1494
0	(none)	95000	94463	1275	1275
1	T7 - 21%	95000	94463	1281	1275
2	T16 - 53%	95000	91810	1275	1275
3	T30 - 48%	95000	94463	1275	1275
4	All -30%	95000	94543	1275	1275
5	All -50%	95000	69690	1275	1275
0	(none)	100000	99846	990	990
1	T7: -30%	100000	99846	990	990
2	T14: -50%	100000	99846	990	990
3	T15: -20%	100000	99433	990	990
4	T17: -35%	100000	99324	953	990
5	T36: -35%	100000	91371	953	990
6	All -30%	100000	81970	953	990
7	All -50%	100000	89570	953	990

[17] S. Deniziak, L. Ciopiński, G. Pawiński, K.Wieczorek and S. Bąk "Cost Optimization of Real-Time Cloud Applications Using Developmental Genetic Programing", Proc. of the 7th IEEE/ACM International Conference on Utility and Cloud Computing, 2014, pp.774-779. http://dx.doi.org/10.1109/UCC.2014.126

[18] K.Sapiecha, L. Ciopiński, and S. Deniziak. "An application of developmental genetic programming for automatic creation of supervisors of multi-task real-time object-oriented systems." IEEE Federated Conference on Computer Science and Information Systems (FedCSIS), 2014. http://dx.doi.org/10.15439/2014F208

[19] Hu, Jingcao, and Radu Marculescu. "Energy-and performance-aware mapping for regular NoC architectures." Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on 24.4 (2005): 551-562. http://dx.doi.org/10.1109/TCAD.2005.844106

[20] Han, Sangchul and Park, Minkyu, Predictability of Least Laxity First Scheduling Algorithm on Multiprocessor Real-Time Systems, Proc. of EUC Workshops, Lecture Notes in Computer Science, vol.4097, 2006, pp.755-764 http://dx.doi.org/10.1007/11807964_76

[21] Sitek, P. "A hybrid CP/MP approach to supply chain modelling, optimization and analysis." Computer Science and Information Systems (FedCSIS), 2014 Federated Conference on. IEEE, 2014. http://dx.doi.org/10.15439/2014F89