UDC 004.021

Olga Rusanova, Igor Boyarshin, Anna Doroshenko.

ENERGY-AWARE TASK SCHEDULING ALGORITHM FOR MOBILE COMPUTING

The paper describes a new algorithm for task scheduling that utilizes the ability of processors and individual cores to dynamically switch between different operating voltage levels, thus sacrificing performance for energy saving and vice versa. The algorithm constructs a complete scheduling strategy and specifies the voltage level mapping for the tasks to be executed with in order to meet the desired time constraint.

Key words: energy-aware task scheduling, voltage levels, mobile computing, time constraint.

Fig.: 3, Tabl.: 1. Bibl.: 4.

Target setting. With the rapid advancement of mobile computing technologies, it has become an essential problem on mobile platforms to satisfy the ever-growing demand of processing power while at the same time keeping the energy consumption levels to a minimum. It is thus important for the new scheduling algorithms to be developed that focus not only on delivering the most performant assignment possible, but also make the energy savings a core factor.

Actual scientific researches and issues analysis. The research into the field has shown that there exist few papers that concern themselves with the issue of energy-aware task scheduling. The problem demands increasing attention especially in light of the accelerated spread of mobile technologies and their application as well as the advancements made in the development of processors capable of effective dynamic voltage level switching.

Not investigated parts of general subject. The vast majority of scientific work related to the subject at hand either describes a purely abstract method to account for energy consumption or neglects such integral part of task scheduling as data transmission between the nodes altogether. For example, the authors of [1] do provide a simple method for accounting for the system energy consumption, but do not consider the cost of data transfer between the tasks. Therefore, it is only natural to develop an algorithm that better meets the real-world requirements.

Research objective. The objective of this paper is to propose a new task scheduling algorithm that meets the following goals. First, the algorithm operates under realistic environment, namely the presence of transmission cost between nodes for data transfer. Second, considers the ability of modern processors to dynamically switch between different voltage levels depending on the nature of the task to be

executed. And third, minimizes the total energy consumption of the system while satisfying the provided time constraint.

Principal statements. For the rest of the paper the following aspects of the environment shall apply:

- All cores are able to dynamically and independently switch between voltage levels V_1 through V_N, where N is the total number of voltage levels available in the system; the switching cost in terms of time and energy is negligible and considered zero.
- Each core is able to perform an unlimited number of transmissions (both inbound and outgoing) simultaneously in parallel with the execution of the workload.
- If two nodes are assigned to the same core their respective transmission cost is considered instantaneous and thus zero.
- Any node can be assigned to any core and start its execution no sooner than all transmissions for it from parent nodes have finished. Root nodes (such that have no parent nodes) can be started immediately.
- For each node its execution time and energy consumption values for each voltage policy are considered to be known prior to the execution of the algorithm and not to change or fluctuate during runtime.
- At runtime each core shall change its voltage level in accordance with the execution policy of the task assigned to it at any given moment.

General structure. The proposed algorithm has several stages and expects the following inputs: a directed acyclic task graph G that specifies the data relations between nodes and the corresponding transmission cost $T_{i,j}$ of the link between nodes i and j for each link in the graph; for each node, a set of N pairs (W_i, E_i), where N is the total number of voltage levels available in the system, W_i and E_i are the corresponding weight (execution time) and energy consumption values associated with voltage policy i for this node; desired execution time D that denotes the maximum time constraint for the overall system of tasks; amount of cores C available to the algorithm, where each core is capable of switching between N voltage levels.

The algorithm is as follows:

1. Assign the slowest execution policy to all nodes in the system.
2. Find the critical path CP with critical time CT based on the task graph G and current execution policies of each node.
3. If CT is greater than the desired time D, then find the earliest node on CP that can improve its execution policy. If no such node is found, then the problem is unsolvable under the given constraints. Otherwise improve the execution policy of this node and go to Step 2.
4. For each node based on its current execution policy find its early time E and late time L, corresponding to the earliest possible and latest possible execution start

time of this node in order to meet the timing constraint *D*, not considering the transmission costs. Calculate the delta as the difference between *L* and *E*: *d* = *L* - *E*.

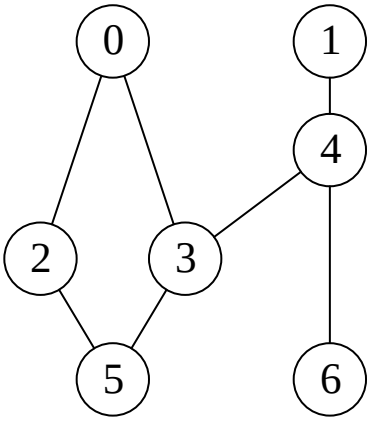
5. Perform the task scheduling, prioritizing the nodes with lowest delta d and assigning each task no the core that would yield the earliest start time for the task considering the transmission costs.

6. If the total execution time of the resulting system does not meet the time constraint *D*, then find the earliest node with start time greater than its late time. For this node, recursively find the earliest parent(s) that this node started immediately after, not accounting for transmission time, and whose execution policy can be improved. The resulting nodes are the ones that must be sped up in order for the initial node to be able to start earlier. Improve their policy and go to Step 4. If the process of finding such nodes does not yield sufficient results, then the system cannot be further improved to meet the time constraint and so the problem is unsolvable under given conditions.

7. Otherwise if the system meets the desired time then output the mapping of the tasks to cores and the resulting execution policy of each task.

Example. In order to help better understand the algorithm and confirm its validness, an example featuring the key aspects of the algorithm at work shall now be presented.

For this example, the task graph is given in Fig. 1 along with the metrics of weight (execution time) and energy consumption (denoted *W* and *E*, respectively) for each of the two available voltage level policies for each task 0 through 6. Transmission cost for all links is considered to be 1. Let the desired time *D* be equal to 12.



T	0		1		2		3		4		5		6	
W	4	7	1	3	2	4	4	6	2	3	1	2	5	7
E	34	18	42	36	39	34	34	22	39	36	42	39	20	18

Fig. 1. The task graph and parameters of nodes

The steps of the algorithm will be as follows.

At first, all tasks are assigned the slowest execution policy.

The critical path shall be found to be through tasks 0, 3, 5 with critical time *CT* equal to 15. As *CT* = 15 is greater than the desired time *D* = 12, the earliest task on critical path is found and its execution policy improved. In this case the execution policy of task 0 is improved.

The critical path of the graph for the new configuration is recalculated and now goes through tasks 1, 4, 3, 5 with *CT* = 14. This *CT* is still greater than *D* = 12, so the earliest task on the *CP* is found (Task 1 in this case) and its execution policy improved.

The critical path of the graph is recalculated yet again, this time it is through tasks 0, 3, 5 with *CT* = 12. This critical time happens to be equal to the desired time *D* = 12, so the current configuration might just be enough. The algorithm moves to the next stage.

The scheduling is done for the current configuration. The result of the scheduling is shown in Fig. 2.

As the total time is 13 which is greater than the desired time 12, the algorithm shall find the earliest task whose start time was greater than its late time L. In this case such task is task 3: it should have started by 4 but started at 5.

Now the algorithm shall recursively find the parent(s) of task 3 that potentially delayed its start and whose execution policy can be improved.

The parents of task 3 are tasks 0 and 4. But task 0 did not delay the execution of task 3, as can be seen from Fig. 2: indeed, task 3 could have started at time point 4 as far as the data from task 0 is concerned. But task 4 with its transmission for task 3 comes with no gaps before the start of task 3, that is why task 4 delayed the start of task 3. As the only parent of task 4 (task 1) cannot further improve its execution policy, the execution policy of task 4 is improved.

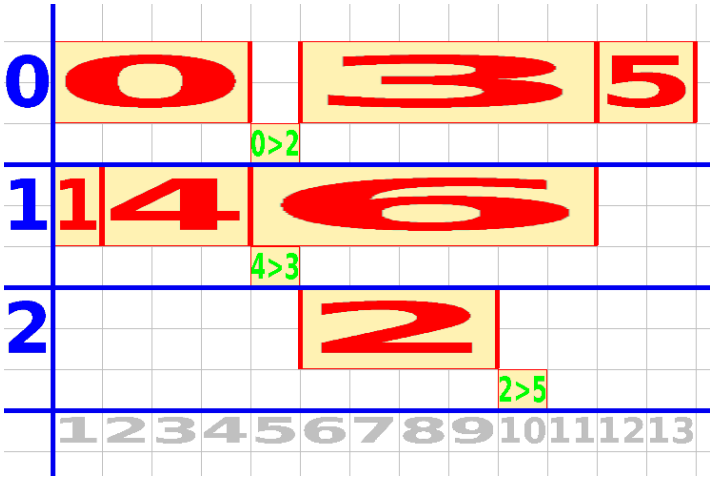


Fig. 2. Intermediate scheduling result

The scheduling is done for the new configuration of the system and is shown in Fig. 3. This time the total execution time of the system is 12 and meets the desired time. The final mapping of execution policies for the tasks is as follows: the fastest execution policy is set for tasks 0, 1, 4; the default execution policy is left for tasks 2, 3, 5, 6. The total energy consumption for the system is thus 228 units.

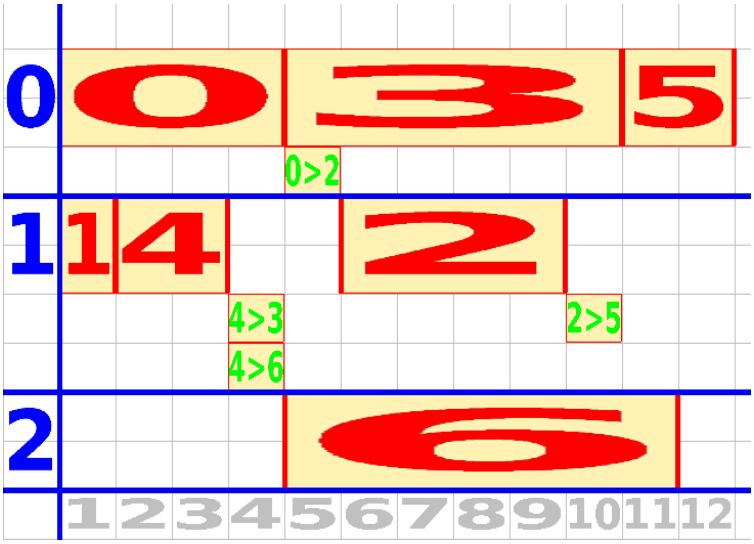


Fig. 3. Final scheduling result

Testing. In order to test the algorithm a program suit has been developed in C++ that allows for easy testing of the system by printing out the steps the algorithm takes in each scenario. A random task graph generator has also been developed so that the system could be tested under numerous time constraints and configurations. The settings of the random task graph generator can be tuned to change the range of the node weights and data transmission volumes, as well as set the overall node count and the connectivity metric of the system.

As a means of measurement of the efficiency of the algorithm a series of tests has been conducted. The inputs were generated in the following way: for a newly-generated task graph with connectivity C (where C is calculated as the number of links in the graph divided by the maximum possible number of links), calculate $CT_{slowest}$ as the critical time of the graph with all tasks set to the slowest policy, $CT_{fastest}$ as the critical time of the graph with all tasks set to the fastest policy, $E_{slowest}$ as the total energy consumption of the system with all tasks set to the slowest policy, $E_{fastest}$ as the total energy consumption of the system with all tasks set to the fastest policy, and let desired time D be equal to the half sum of $CT_{slowest}$ and $CT_{fastest}$; E and T are the resulting energy consumption and execution time of the system that the algorithm generated.

The results of the tests for a number of runs are shown in Table 1.

Table 1

Testing results for different system configurations

C	$CT_{fastest}$	$CT_{slowest}$	$E_{fastest}$	$E_{slowest}$	D	E	T
0.3	13	20	291	172	16	230	16
	18	28	345	204	23	251	22
	11	17	290	172	14	196	14
0.5	24	38	276	164	31	218	31
	24	37	350	208	30	274	29
	23	34	304	180	28	271	28
0.7	17	30	283	168	23	220	22
	27	44	296	176	35	253	33
	22	36	311	184	29	239	29
0.9	27	44	296	176	35	242	35
	30	47	317	188	38	263	38
	29	45	305	180	37	247	37

It is clear from the results that if the desired time is set to be halfway between the slowest and fastest critical times, then the corresponding energy consumption is also halfway between its slowest and fastest setting, albeit with some deviations from the mean. From that the conclusion can be made that as the desired time changes linearly, so does the resulting energy consumption. The pattern is also repeated if the target point is set not in one half, but one fourth of the range.

The predominant deviations towards the increment from the mean for the energy consumption levels in all cases can be explained in the following way: the fastest critical time can seldom be achieved because it is calculated without the account for the transmission costs. And in the case of actual planning those transmission costs prevent the system from reaching the target times. Thus, the algorithm is forced to allocate additional resources in the form of improving the voltage policies, and so the total energy consumption levels increase accordingly.

Conclusion. With the development of new processors with the ability to dynamically switch between different voltage levels depending on the task being executed, a new series of task scheduling algorithms must be developed to utilize this ability in order to provide appropriate energy consumption levels on mobile platforms. The conducted studies of the new task scheduling algorithm for energy-aware scheduling have shown that it is more than capable of providing noticeable energy savings when the desired execution time of the system of tasks allows for it.

References

1. Li, Y., Chen, M., Dai, W., Qiu, M. (2017). Energy optimization with dynamic task scheduling mobile cloud computing. In Proceedings of the IEEE Systems Journal 11(1) (pp. 96–105).

2. Liu, Y., Veeravalli, B., and Viswanathan, S. (2007). Critical-path based low-energy scheduling algorithms for body area network system. In Proceedings of the IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (pp. 301-308).

3. Bezerra, P. T. et al. (2013). Dynamic frequency scaling on android platforms for energy consumption reduction. In Proceedings of the 8th ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems (pp. 189-196).

4. Mei, J., and Li, K. (2012). Energy-aware scheduling algorithm with duplication on heterogeneous computing systems. In Proceedings of the ACM/IEEE 13th International Conference on Grid Computing (pp. 122–129).

AUTHORS

Olga Rusanova (supervisor) – associate professor, Department of Computer Engineering, National Technical University of Ukraine “Igor Sikorsky Kyiv Polytechnic Institute”.

E-mail: olga.rusanova.v@gmail.com

Igor Boyarshin – student, Department of Computer Engineering, National Technical University of Ukraine “Igor Sikorsky Kyiv Polytechnic Institute”.

E-mail: igor.boyarshin@gmail.com

Anna Doroshenko – student, Department of Computer Engineering, National Technical University of Ukraine “Igor Sikorsky Kyiv Polytechnic Institute”.

E-mail: annadoroshenko03@gmail.com