Assigning and Scheduling Hierarchical Task Graphs to Heterogeneous Resources

Panayiotis Alefragis[1], Christos Gogos[3], Christos Valouxis[1,2], George Goulas[1,2], Nikolaos Voros[1], and Efthymios Housos[2]

[1]*Technological Educational Institute of Western Greece. Dept. of Computer & Informatics Engineering, Greece*

[2]*University of Patras-Greece. Dept. of Electrical and Computer Engineering, Greece*

[3]*Technological Educational Institute of Epirus. Dept. of Accounting and Finance, Greece*

Keywords: *Hierarchical Task Graphs, integer programming, parallel processing*

Abstract

Task Scheduling is an important problem having many practical applications. More often than not, precedence constraints exist between tasks, and a common way to capture them is through Directed Acyclic Graphs (DAGs). A DAG might contain a great number of tasks representing complex real life scenarios. It might be the case that logical groupings of tasks exist giving a hierarchical nature to the graph. Such Hierarchical Task Graphs (HTGs) have nodes that are further analyzed to DAGs or to other HTGs. In this paper a method of solving an HTG problem is presented based on the idea of gradually solving the problem by replacing subgraphs with virtual nodes. Integer Programming is used to generate virtual nodes that replace a subgraph, results from solving the subgraph problem using. So a series of subproblems are solved and starting from the deeper levels of the HTG a solution to the full problem emerges.

Introduction

Hierarchical task graphs (HTG) are directed graphs where nodes can either be simple or composite activities. Each hierarchy level is a set of interconnected directed acyclic graphs (DAG). Simple tasks are considered atomic, requiring a single resource to be used for their execution while a composite activity can use multiple resources at different hierarchical levels. Each composite activity can be represented as a subtree in the HTG. HTG is a typical high level representation of computer program kernels, but they can also be used in the scheduling of multiple development teams that are involved in multiple concurrent projects where each

activity generates results that are used by other activities or involve recurring activities. The usual goal of these problems is to reduce the global makespan of the presented problem.

In this paper, we present a method that uses a MIP model to solve individual DAG sub-problems to optimality and a heuristic approach to solve the whole problem using a bottom up traversal of the HTG tree. The example is derived from the solution of an HTG for the assignment and scheduling of computational kernels to embedded multicore architectures. In the example presented in Figure 1, a two level HTG with 6 nodes at the top level is presented. Three resources are available to perform the tasks, in our case they are heterogeneous processors. The top level is presented on the left side, where node 5 is a composite activity and all other nodes are atomic ones. Node 5 is a DAG only containing simple nodes and is presented on the right side of the same figure.

Table 2. After the execution of activity, the result has to be “communicated” to the dependent tasks if these tasks are not performed by the same resource. This can be perceived as the communication time of variable values in our example, or the transportation time between two sites in a production example, or the collaboration time between two individuals that work on the same problem. In this simplified example, the “communication” cost is depicted by the values on the arcs of the DAGs. In the more general case, it can be the result of a function that involves the communicating resources, the time that this communication occurs, other available resources that help to perform the communication, etc. The required processing time to perform composite task 5 contains currently contains no values as it may be executed by different processors in parallel. Which processors and for how long task 5 will occupy will be the result of solving the sub-problem represented by the DAG at the lower level of HTG.

Two approaches to solve the problem of scheduling a Hierarchical Task Graph will be presented. The first approach embeds each subgraph to the graph that contains it. This process can occur recursively until no more composite tasks exist. Then the resulting flat DAG can be assigned and scheduled at the available resources using either a heuristic method like HEFT or if the size of the problem permits it using an ILP solver. The second approach uses an ILP formulation in order to solve each subgraph in turn and uses the generated subgraph schedule as

a resource based clustering of the contained tasks to form the schedule of the graph that contains the subgraph.

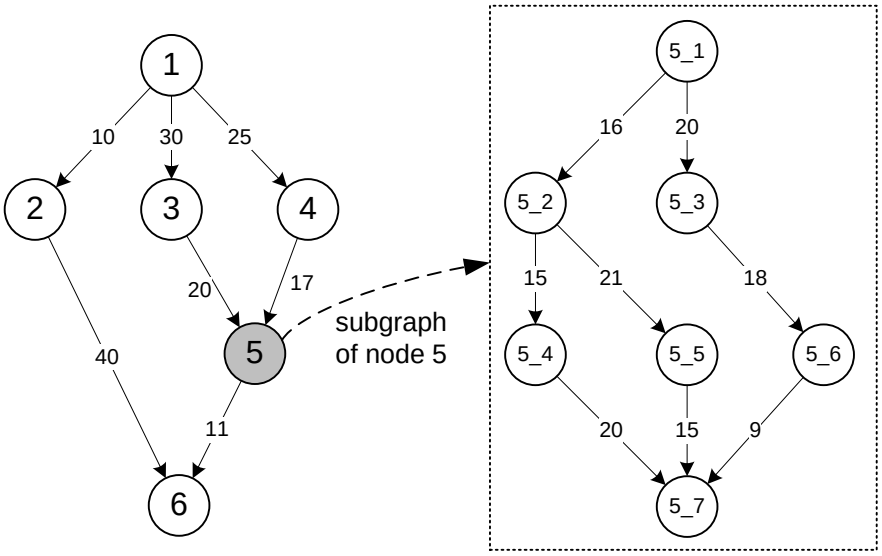

Figure 1: Example of a Hierarchical Task Graph

Table 1: Execution Times for Tasks of the HTG

	Tasks					
Resources	1	2	3	4	5	6
R0	15	16	38	19	-	17
R1	22	31	40	20	-	25
R2	37	42	51	30	-	22

Table 2: Execution Times for Tasks of the Subgraph

	Tasks						
Resources	5_1	5_2	5_3	5_4	5_5	5_6	5_7
R0	10	29	38	19	20	15	29
R1	15	32	40	25	28	36	31
R2	37	27	26	32	47	30	42

DAG makespan optimization

DAG scheduling is a well-studied subject [1],[2]. Several methods exist that can be used in order to assign tasks to resources and create individual schedules for each resource. What is interesting is that the individual resource schedules cannot be decided in isolation due to task dependencies and the "communication" cost they imply. Heuristics based methods are able to efficiently solve DAGs with several hundreds of nodes. These methods either belong to the category of list scheduling with prominent examples being Earliest Time First (ETF) [3], Heterogeneous Earliest Finish Time (HEFT) [4],[5] and Critical Path on a Processor (CPOP) [6] or to the category of clustering where examples of such algorithms are Dominant Sequence Clustering (DSC) [7] and Linear Clustering Method (LCM) [8]. Mathematical programming based methods exhibit much bigger execution time and do not scale well for bigger instances, but are able to find high quality solutions [9],[10],[11],[12].

ILP Model for solving a task DAG

As our test case is part of a compiler tool chain and each execution requires the solution of 100 - 10000 DAG it was a requirement that each DAG formulation and solution should take no more that some seconds. A practical observation was that if the total number of nodes for each DAG is less than 30 then the problem can be efficiently solved using an ILP solver within the available time budget. In our experiments this restriction has been satisfied using the open source IP solver COIN-CBC running on a current PC. The topography of the DAG seems to have little impact on the execution time of the approach due to the fact that equations are effectively generated for all pairs of nodes.

The mathematical model uses the set of tasks T and the set of available resources P. Each resource can process a given task at different time horizon, i.e. that the problem model is heterogeneous which in our example means that the processors are not identical. For project planning problems this would mean that the worker will have different skills and could handle the same task with a different execution duration. The execution time of each task t by each resource p is given by the parameter w_{tp}. The "communication" cost between task t and task t', when they are not assigned at the same resource is given by $c_{tt'}$, 0 otherwise. We have two sets of decision variables. The binary variables y_{tp} depict an assignment of task $t \in$

T to resource $p \in P$ and take the value 1 if task t is assigned at processor p and 0 otherwise. The integer variables x_t are defined over each $t \in T$ and their value represents the start time of task t by the resource corresponding to the y_{tp} variable with value 1. Since the problem is described as a DAG with a set of nodes V and a set of edges E, it should be noted that the set of tasks and the set of nodes are conceptually identical, while the set of edges represents precedence constraints between tasks with weights of edges associated with communication costs.

The objective function of the model is shown in equation (1) representing the target of minimizing the total schedule length, which is also known as makespan.

$$\text{minimize } x_{t_s} \qquad t_s \text{ is the sink node of the DAG} \quad (1)$$

Three groups of constraints are defined. The first one ensures that each task should be assigned at exactly one resource (2).

$$\sum_{p \in P} y_{tp} = 1 \qquad \forall t \in T \qquad (2)$$

For each task t let T_t be the set of tasks that have to be completed before t starts execution. The second group of constraints states that for each task t the corresponding start time should be greater than all the finish times of tasks that belong to T_t. In addition, when task t is scheduled to a different resource than a task t' that it depends on, the "communication" cost between them should also be considered. In order to model this constraint three new variables are introduced et_t, $z_{tt'}$ and $k_{ptt'}$. et_t is an integer variable and corresponds to the execution time of task t derived from equation (3). Variable $z_{tt'}$ is binary and equals 1 when both tasks t and t' are assigned at the same resource and 0 otherwise. It is defined by variable $k_{ptt'}$ which corresponds to the product of binary variables $y_{t'p}$ and $y_{t'p}$. Since the product between variables violates the linearity of the model variable $k_{ptt'}$ assumes its value indirectly through inequalities (4), (5) and (6). $z_{tt'}$ is defined by equation (7) as the sum of products between variables $y_{t'p}$ and $y_{t'p}$. Finally, inequality (8) states that the difference between execution times for task t and task t' provided that t depends on t' should be no less than the processing time of task t on the designated resource plus the extra communication time needed when tasks are not assigned at the same resource.

$$et_t = \sum_{p \in P} w_{tp} y_{tp} \qquad \forall t \in T \qquad (3)$$

$$k_{ptt'} \leq y_{tp} \qquad \forall p \in P; \forall t \in T, t' \in T_t \qquad (4)$$

$$k_{ptt'} \leq y_{t'p} \qquad \forall p \in P; \forall t \in T, t' \in T_t \tag{5}$$

$$k_{ptt'} \geq y_{tp} + y_{t'p} - 1 \qquad \forall p \in P; \forall t \in T, t' \in T_t \tag{6}$$

$$z_{tt'} = \sum_{p \in P} k_{ptt'} \qquad \forall t \in T, t' \in T_t \tag{7}$$

$$x_t - x_{t'} \geq et_t + c_{tt'}(1 - z_{tt'}) \quad \forall t \in T, t' \in T_t \tag{8}$$

For each task *t*, let T'_t be the set of tasks for which there is no path in the graph that connects then to task *t*. Two tasks *t* and *t'* are considered to be independent if by starting from *t* (or *t'*) and then recursively examining all predecessors of this task, the entry node is met before node *t'* (or *t*). The third group of constraints guarantees that when two independent tasks are assigned at the same resource, they will not overlap. Given that two independent tasks *t* and *t'* are are assigned at the same resource then $z_{tt'}=1$. If x_t, the start time of task *t*, is less than $x_{t'}$, the start time of task *t'*, then x_t plus the execution time of task *t* (et_t) should be less than $x_{t'}$ (9). Likewise, when $x_{t'}$ is less than x_t inequality (10) assures that no execution overlap occurs. Binary variable $m_{tt'}$ ensures that exactly one of inequalities (9) and (10) should hold.

$$(1 - z_{tt'})M + x_{t'} - x_t \geq et_{t'} - M(1 - m_{tt'}) \; \forall t \in T, t' \in T'_t \tag{9}$$

$$(1 - z_{tt'})M + x_t - x_{t'} \geq et_{t'} - M(m_{tt'}) \quad \forall t \in T, t' \in T'_t \tag{10}$$

The proposed model is adequate to solve real world problems that arise during the mapping and scheduling of sequential Scilab code in an automatic parallelizing compiler toolflow that targets embedded multicore architectures in the context of the EU founded FP7 ICT ALMA project.

In the case where an HTG does not contain-subgraphs that represent recurring events (loops), it is possible to embed these subgraphs that are deeper in the hierarchy of the HTG to the subgraph that contains them. In a computer program this can be perceived as function inlining, as each subgraph is a DAG of tasks. In a project schedule this can be perceived as including the activities of each department in the global scheduling of a store and then the produced solution will be embedded in the assignment and scheduling problem of the firm, where staff

members (resources) that perform the tasks do not work for a single department or store. After flattening, the resulting graphs will be a DAG. For example, Figure 2 represents the HTG of Figure 1 after the flattening transformation is performed.

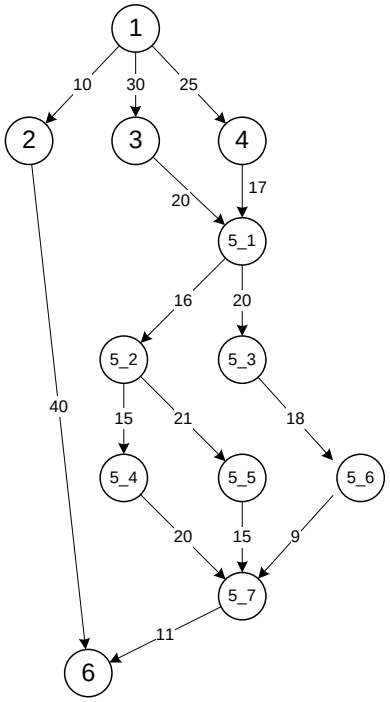

Figure 2: Flattened DAG of Figure 1 HTG

At the lowest level of an HTG most subgraphs will be DAGs that contain only simple tasks. After solving such a subgraph, the emerged solution will specify the assigned resource that each task should be scheduled on and for each task the sequence and the estimated time that it will start executing relative to the execution start time of the initial task. For example, when the subgraph of Figure 1 is solved, the solution presented in Figure 3 may arise. This solution schedules tasks 5_1, 5_2, 5_5, 5_4 and 5_7 in resource 0, schedules no tasks to resource 1 and schedules tasks 5_3 and 5_6 on resource 2. The makespan of the given schedule is 124 time units. If the generated graph can be efficiently solved by the ILP model, the ILP formulation is passed to an ILP solver and the optimal solution is produced.

Bottom-up solution of HTG by subgraph replacement with virtual nodes

With or without flattening at the lower levels, most large problems that are represented by an HTG cannot be totally converted to a DAG. In order to use the

subgraphs solutions in the solution process of the enclosing subgraph, the following procedure is applied. The HTG is traversed bottom up and the subgraphs that can be solved without any modification, i.e. they only contain simple tasks or their subgraphs can be flattened, are solved applying a DAG algorithm. The selection of the solution algorithm is done based on characteristics of the graph. If the graph is small an ILP solver is used, if the graph is large a heuristic is applied. For the graphs that contain composite tasks that we have a solution for their subgraph the following transformation is performed. The composite task is replaced by a number of virtual tasks that are equal to the number of the used resources in the solution found. Each virtual task has an execution time of the combined work of the assigned tasks on the associated resource. All incoming and outgoing dependency arcs of the composite resource are replicated and the outer DAG is solved in a similar manner. This process continues iteratively until the top HTG layer is solved. For example, in Figure 4, pseudo task 5_R0 is a task that has an execution duration of 124 and should be executed in processor R0 and pseudo task 5_R2 is a task that has an execution duration of 56 and should be executed on processor R2. Theoretically, a pseudo task 5_R1 could have been created, but since it contains no tasks on the subgraph's schedule, it can be neglected. It is important to note that by specifying the schedule start time of the pseudo task that contains the root node of the subgraph, all other pseudo tasks are relatively positioned based on the solution of the subgraph. If for example pseudo task 5_R0 is scheduled to start at time point 100 then pseudo task 5_R2 should start at 130 and should finish at 186. If any of the virtual tasks is delayed to start compared to the offset of the subgraph solution, a new makespan for the subgraph solution should be calculated.

The virtual task that contains the root node of the subgraph (5_R0 in the example, since node 5_1 is scheduled on processor R0) should inherit the incoming edges of the nodes that were immediate predecessors (node 3 and node 4) of the composite task that was removed. Likewise the pseudo task that contains the sink node of the subgraph (5_R0 in the example, since node 5_7 is scheduled on processor R0) should inherit the outgoing edges to the nodes that were immediate successors (node 6) of the composite task that was removed.

The graph shown in Figure 4 is solved using the mathematical module using execution information included in Table 3. The information about the resource

that each pseudo task should be assigned is implicitly included in the model and the only variable about the subgraph that remains to be decided is the start time of the pseudo task containing the root node of the subgraph. A natural extension is to calculate the total execution time of the assigned tasks of each virtual task for all the available resource and constraints that only one resource should be assigned at a virtual task and that no two virtual tasks that belong to a virtual task group should be assigned at the same resource. This will provide the required flexibility to exchange the assigned work to a resource when the surrounding tasks information is available during the solution of the outer subgraph.

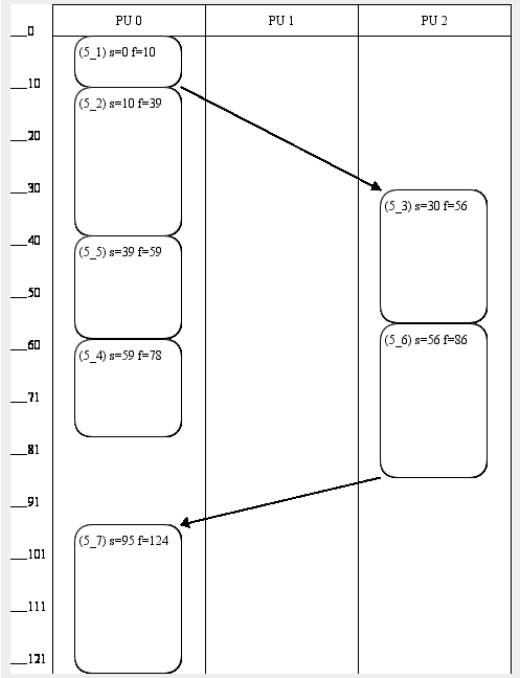

Figure 3: Schedule of subgraph

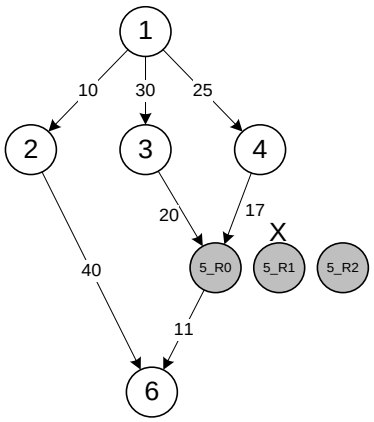

Figure 4: HTG with Composite Task 5 Replaced by Pseudo Tasks 5_R0, 5R1 and 5R2. Task 5_R1 is immediately dropped since no tasks of the subgraph are scheduled in processor R1.

A special case exists, when the subgraph represents the body of nested recurring events (loops). In this case, the subgraph solution represents a recurring task schedule that will be executed for each loop iteration and thus it is not possible to determine the exact makespan of the virtual tasks in the general case. In this case, for all the resources that are used in the solution of the subgraph, it is only possible to estimate the execution time of the assigned tasks before the execution of the virtual task. For all the other tasks that are assigned at the processor and scheduled after the virtual task, their start time can only be determined in relation to the finish time of the recurring virtual task.

Table 3. Execution Times for Tasks and Pseudo Tasks of the HTG

	Tasks						
Resources	1	2	3	4	5_R0	5_R2	6
R0	15	16	38	19	124	X	17
R1	22	31	40	20	X	X	25
R2	37	42	51	30	X	56	22

Our current approach is to split the outer DAG scheduling problem into two sub problems, the “before” sub problem that determine tasks scheduling of all resources before the recurring event and the “after” subproblem for all the tasks after the execution of the virtual tasks. This implies that tasks that depend on the virtual tasks will belong to “after” sub problem, tasks that the virtual tasks depend on will belong to the “before” sub problem and independent tasks can be assigned and scheduled before, after or in parallel to the virtual tasks using resources that are not used by the virtual tasks. The above process may lead to a possibly suboptimal solution. On the other hand, this approach guarantees that no prolonged wait time for tasks that require results generated during the execution of the virtual tasks will occur, thus preventing execution blocking on all the other resources. The side effect is that no deterministic execution makespan of the subgraph can be determined if the number of recurring events is not algebraically determined. For our application problem, the ALMA toolchain provides feedback from a platform simulator and thus the real execution time of the recurring task can be determined, irrelevant to the actual iterations number. In our use case, during the subsequent applications of the described optimization algorithm the estimation of the actual execution time of the recurring event will be used,

allowing the algorithm to improve resource utilization, as the whole DAG can be scheduled as in the case where the subgraph does not include loop nests.

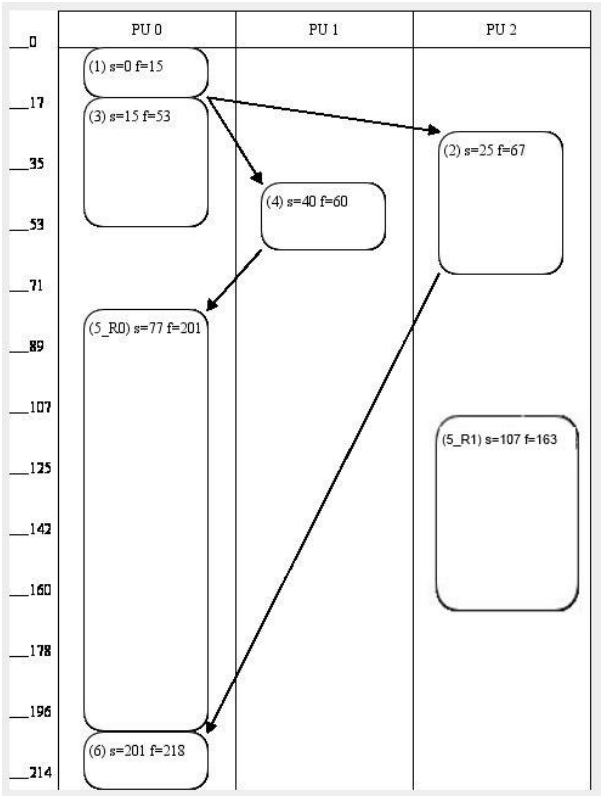

Figure 5: Schedule of the graph including pseudo tasks

ILP model modifications

A set of modification to the ILP model that was previously described are needed in order to recursively solve each subgraph until the original HTG is fully scheduled. The main modification that is necessary is the replacement of each composite task with a number of pseudo tasks, equal to the number of the resources used in the solution of the corresponding subgraph. These pseudo tasks have predefined values for their corresponding binary variables y_{tp} since the assigned resource that each one of them will be scheduled to is known. For each pseudo task group, the value of variable $x_{t,}$ which denotes the start time of pseudo task t, has to be determined only for the pseudo task that contains the root node of the subgraph. The values of variables x_t for the remaining pseudo tasks in a group are determined based on the offset exposed in the solution of the subgraph. Regarding the constraints, only the pseudo task that contains the root node of the subgraph and the pseudo task that contains the sink node of the subgraph have to

be included in the set of tasks that participate in the generation of the second group of constraints. Pseudo tasks that neither contain the root nor the sink node of the subgraph only have to participate to the third group of constraints in order to prevent simultaneous execution of tasks by the same resource.

This is required as no edges exist between the pseudo tasks and all the other tasks in the DAG, making them look independent. It should be noted that pseudo tasks share the same set of independent tasks with the composite task that they replace. Consequently, for each pseudo task, constraints belonging to the third group have to be generated with respect to all other tasks that have no direct or indirect path to the composite task that they replace.

The final schedule of the graph of Figure 4 is presented in Figure 5. The HTG is finally scheduled, having a makespan of 218 time units. The proposed algorithms have been applied to code sources that are represented by HTGs with up to 15 layers, 200 to 500 composite tasks and 1000 to 2000 leaf tasks that included recurring tasks and managed to produce parallel solutions in less than 1h using a typical PC.

Conclusions and future work

In this paper, a generic algorithmic approach to assign and schedule hierarchical task graphs to heterogeneous resources is presented. The proposed approach is currently applied to a parallelizing compiler tool chain for automatic parallelization of sequential SciLab programs to embedded multicore architectures but can be used in other application areas. We are currently working in supporting multiple solutions selection for the composite tasks during the bottom up solution process as well as the application of meta-heuristic local search optimization techniques for post processing of the generated solutions. We also plan the inclusion of a more detailed modeling of both the “communication” model between tasks as well as a more detailed modeling of the processing resources.

Acknowledgement: This work is co-funded by the European Union under the 7th Framework Program under grant agreement ICT-287733**,** project “Architecture oriented parallelization for high performance embedded Multicore systems using scilAb (ALMA)”.

References

1. Sinnen O., Task Scheduling for Parallel Systems. John Wiley & Sons, 2007.

2. Canon L.-C., E. Jeannot, R. Sakellariou, and W. Zheng, Comparative Evaluation Of The Robustness Of DAG Scheduling Heuristics, in Grid Computing, S. Gorlatch, P. Fragopoulou, and T. Priol, Eds. Springer US, pp. 73–84, 2008.

3. Hwang J.-J., Y.-C. Chow, F. D. Anger, and C.-Y. Lee, Scheduling Precedence Graphs in Systems with Interprocessor Communication Times, SIAM Journal on Computing, vol. 18, no. 2, pp. 244-257, Apr. 1989.

4. Topcuoglu H., S. Hariri, and M.-Y. Wu, Performance-effective and low-complexity task scheduling for heterogeneous computing, IEEE Transactions on Parallel and Distributed Systems, vol. 13, no. 3, pp. 260 –274, Mar. 2002.

5. Bittencourt L. F., R. Sakellariou, and E. R. M. Madeira, DAG Scheduling Using a Lookahead Variant of the Heterogeneous Earliest Finish Time Algorithm, in 2010 18th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP), pp. 27 – 34, 2010.

6. Kwok Y.-K. and I. Ahmad, Dynamic critical-path scheduling: an effective technique for allocating task graphs to multiprocessors, IEEE Transactions on Parallel and Distributed Systems, vol. 7, no. 5, pp. 506-521, 1996.

7. Gerasoulis A. and T. Yang, A comparison of clustering heuristics for scheduling directed acyclic graphs on multiprocessors, Journal of Parallel and Distributed Computing, vol. 16, no. 4, pp. 276-291, Dec. 1992.

8. Kim S. J. and J. C. Browne, A general approach to mapping of parallel computation upon multiprocessor architectures, in International Conference on Parallel Processing, vol. 3, pp. 1-8, 1988.

9. Davare A., J. Chong, Q. Zhu, D. M. Densmore, and A. L. Sangiovanni-Vincentelli, Classification, Customization, and Characterization: Using MILP for Task Allocation and Scheduling, Technical Report Identifier: EECS-2006-166, EECS Department, University of California at Berkeley, California, Dec. 2006.

10. Valouxis C., C. Gogos, P. Alefragis, G. Goulas, N. Voros and E. Housos, DAG Scheduling using Integer Programming in heterogeneous parallel execution environments, in Proceeding of the Multidisciplinary International Conference on Scheduling: Theory and Applications (MISTA 2013), pp. 392-401, Gent, Belgium, 26th - 30th August 2013.

11. Davidovic, T., Liberti, L., Maculan, N., & Mladenovic, N., Mathematical Programming-Based Approach to Scheduling of Communicating Tasks, Les Cahiers du GERAD, 2004.

12. Tompkins, M. F., Optimization techniques for task allocation and scheduling in distributed multi-agent operations, Doctoral dissertation, Massachusetts Institute of Technology, 2003.