A COMPREHENSIVE TEST BENCH FOR THE EVALUATION OF SCHEDULING HEURISTICS

Udo Hönig and Wolfram Schiffmann
Lehrgebiet Rechnerarchitektur
FernUniversität Hagen
58084 Hagen, Germany
{Udo.Hoenig, Wolfram.Schiffmann}@FernUni-Hagen.de

ABSTRACT
Computational problems can usually be divided into a number of (sub)tasks. The data dependencies between these tasks can be described by a so called task graph. Since the task graph scheduling problem is known to be NP-hard, researchers devised an innumerable number of heuristic algorithms to solve this problem best. Up to now, these heuristics were analyzed by using a set of task graph problems, which were rarely revealed to the public. A comparison of different scheduling algorithms presupposes the availability of their implementations. Our contribution consists of a test bench with 36000 task graph problems and their optimal solutions with respect to homogeneous target architectures. These test cases are structured according to several task graph properties and target architectures' sizes and enable researchers to analyze the overall quality of their heuristics' results, to detect strengths and weaknesses of their algorithms and to compare them with those of other researchers. We used the current intermediary version of this test bench to compare some well known scheduling algorithms, namely DLS [1], ETF [2], HLFET [3] and MCP [4]. Surprisingly, the heuristics show a very similar behavior regarding the quality of the obtained results, exhibiting the same strengths and weaknesses, differing only by few percent.

KEY WORDS
Task Graph Scheduling, Optimal Schedules, Test Bench

1 Introduction

If a computational problem can be divided into a number of subtasks, the data dependencies between these subtasks are usually described by means of a directed acyclic graph (DAG) also called *task graph*. In order to model the execution of a task graph on a homogeneous computing system, the task graph is attributed by weights at the nodes and the edges. While the node weights denote the computational effort of a subtasks, the edge weights correspond to the time for transferring intermediate results via the network between two computers. This time is only incurred, if the connected tasks are scheduled to different computers. Else the edge weight is considered to be zero.

The problem of scheduling task graphs to a parallel computer architecture has received a large amount of attention in the last decade [5, 6, 7]. Most often, the objective of scheduling is to minimize the overall computation time (also called *schedule length*). In order to achieve this objective, a scheduling algorithm has to map the tasks to the available computers and to determine the corresponding starting times. The scheduling problem in its general form has been proven to be NP-complete [8]. This has stimulated researchers to devise an innumerable number of heuristic algorithms based on quite different assumptions and basic ideas. Although its developers claim that their algorithms are efficient it is very difficult or even impossible to compare these algorithms to each other.

This shortcoming could only be tackled by means of a comprehensive collection of task graph problems which cover a broad spectrum of meaningful graph properties along with the corresponding optimal schedules. Because of the NP-completeness, the optimal schedules for such a test bench suite can only be computed by means of *informed* search algorithms [9, 10] and the size of the task graphs must be limited in such a way that the optimal schedules can be computed in a reasonable period of time.

In this paper, we introduce a test bench that comprises 36000 task scheduling problems with task graphs ranging from 7 to 24 nodes and target architectures ranging from 2 to 32 parallel computers. While the computation of the optimal schedules is not yet finished (in progress since November 2003 on a 16 Pentium-4 cluster computer) we will present intermediary results regarding the already computed 31756 optimal schedules (September 2004). By means of this test bench, we evaluated the performance of four well known heuristic algorithms. Surprisingly, we found out that – even though a comprehensive and diverse test bench was used – the heuristic algorithms behave very similarly. This finding approves the necessity of test benches like the one proposed in this paper. Thus, in the near future we will publish our test bench on the web and hope that it will be used thoroughly as a benchmark in future research on task graph scheduling.

To our knowledge there exists no comparable test bench for the evaluation of heuristic scheduling algorithms. Although in [11] a test bench for task graphs with up to 5000 tasks is provided these task graphs do not take communication overhead into account. Also, it is not guaranteed that the minimum schedule length is given for every task graph problem, because of the NP-completeness the run time of the search algorithm was limited to 10 minutes. In [12] a performance study of 15 heuristic scheduling algorithms is presented. In contrast to our work, the number of task graphs is much lower (≈ 350) and for most of the investigated task graphs (250) the optimal schedules are not known.

The rest of this paper is organized as follows. In section 2 the details of the test bench and its creation will be described. Then, we sketch how the optimal schedules were computed and how reliability has been ensured. Section 4 discusses the impact of the task graph properties on the computational effort while computing the optimal schedules. It also provides a comparison of four well known heuristic algorithms based on the test bench's task graph problems. Section 5 concludes the paper.

2 Creation of the Test bench

2.1 The Task Graph Generator

Most authors who publish a new heuristic do rarely reveal the task graphs they used for its evaluation. Sometimes, even the task graphs' properties are not explained. An exception to this is [12], where the used test set is available at the author's web-page. Although these task graphs can be used to evaluate and compare heuristics, the size of this test bench is too small (≈ 350 task graphs) for a comprehensive evaluation.

Up to now, there exists no suitable test bench for a thorough examination of scheduling heuristics and the effects, which task graph properties have on the quality of the heuristics' results. For this reason, we created a new test bench, consisting of randomly generated task graphs and structured concerning the following parameters:

- **Task Graph Size**: The size of a task graph is defined by the number of its nodes.
- **Meshing degree**: The meshing degree of a task graph defines the extent to which the nodes are connected with each other.
- **Edge-length**: The length of an edge indicates the distance between the connected nodes.
- **Node- and Edge-weight**: These parameters describe the time, required for a task's computation or the transfer of data. In literature, both parameters are usually merged to the Computation to Communication Ratio (CCR). Although this merging facilitates a close examination of the relation between both parameters, a separate analysis of each parameter becomes impossible.

In order to obtain a large set of test cases, structured with respect to the parameters described above, we had to develop a task graph generator. Besides the number of the task graphs to generate, the task graph generator's input contains a common prefix of the task graphs file names and a detailed information about the task graphs' properties (see subsection 2.2 for more details).

To emphasize a certain graph property (e.g. short ranging edges), the random numbers generator has to be focused to a dedicated range of values. This could either be achieved by a uniform distribution with limited range of values or by Gaussian distribution. In contrast to the first alternative, the second one does not limit the diversity of task graphs, since it allows a blending of the properties up to a certain degree (e.g. few short edges in a graph with predominantly long edges). For this reason, the task graph generator determines the random numbers by means of an equal or Gaussian distribution.

In order to describe a given task graph problem within one file, the task graph generator merges the task graph and the target architecture's description. Because of the assumed homogeneity, the architecture's description can be limited to a single integer, representing the number of processing elements. Every generated task graph problem is stored by using a file format, which is very similar to the formats proposed by [11]. Referring to this similarity, we call it Standard-Task-Graph-Format (STG) as well. The first row of every task graph file consists of two integer values, representing the task graph's size and the size of the target system for which the schedule should be computed. Then, the task descriptions follow. Every task is firstly described by a single row with three integers, representing the task's index, the required computation time and the number of its direct predecessors. Secondly, for every edge to a direct predecessor follows a separate line, including two integers, the parent's task index and the edge weight.

2.2 The task graph parameters

The task graph problems were generated for one target architecture's size and cloned to the other sizes afterwards by changing the files' indicator for the number of processors. To keep the computation time of the optimal schedules in reasonable boundaries, the upper bound of the task graphs' size is set to 24. The graphs are grouped into three categories in order to provide a structured set of task graphs that can even be processed by slow optimal scheduling algorithms. Within these groups, whose sizes range from 7 to 12, 13 to 18 and 19 to 24 tasks, the task graph's sizes are equally distributed.

Considering the meshing degree, the task graphs are subdivided into four categories. When a task graph is weakly meshed, the Number of Sons (NoS) is low – therefore this category is called 'NoS_Low' (see figure 1). In this group, every node is connected to 25% (expectation value) of all nodes with a higher task index. In case of a medium meshing, the expectation value is set to 50% ('NoS_Avg') and in case of strong meshing it is set to 75 % ('NoS_High'). The deviation is 25% in all cases. In order to provide task graphs which are unbiased with respect to meshing, the test bench includes a category of task graphs with random meshing degree ('NoS_Rand'). Here, the meshing is computed using uniform distribution with a lower bound of 1% and an upper bound of 100%.

The 'edge length'-Parameter (EL) is organized in a similar way. For task graphs with short edges ('EL_Short'), the expectation value is set to 25%. For average long edges ('EL_Avg') it is set to 50% and in case of long edges ('EL_Long') its value is 75%. The deviation is always 25%. In order to provide task graphs with random edge length ('EL_Rand'), a group of graphs with uniformly distributed edge lengths was generated as well. Here, the lower bound is 1%, the upper 100%.

In literature, node and edge weights are usually merged to a parameter called 'Computation-to-Communication-Ratio' (CCR). But deciding on a fixed proportion would reduce the diversity of generated task graphs. For this reason, we distinguish both graph-properties as separate parameters. Possible weightings are heavy, light and random. The first two characteristics are Gaussian distributed, the last one uniform. In case of heavy nodes/edges, the expectation value is 15 (time units) with a deviation of 3. Light nodes/edges use the same deviation but an expectation value of only 5. The uniformly distributed random weights are bounded by the values 1 and 20.

2.3 Structure of the Test bench

The generated test bench has to be large enough to avoid, that single outliers distort the analysis' results. Nevertheless, it has to be compact enough to be transmittable across standard Internet connections. Therefore, the number of test cases, belonging to every combination of characteristics, was limited to 25. Only the test classes with equally distributed Node- and Edge-Weights (RNodeREdge) were extended to 50 test cases, to provide a broader range of task graphs not affected by the weightings. Since test cases with random weightings were just included as a reference, combinations of random and weighted properties, e.g. RNodeHEdge, were not generated.

7200 task graphs (see figure 1) were generated this way. These task graphs were cloned to different target system sizes (2, 4, 8, 16 and 32 TPEs) by modifying the appropriate parameter in the first line of every STG-File. In this way, we get 36000 in total.

The test bench is organized as a flat-file database[1], facilitating a free and easy access by the means of any operating system and its shell. In addition, the database becomes independent of special database management systems, which should be beneficial for the test bench's spreading.

[1] A flat-file database renounces an explicit database management system and stores the data in text files which are organized hierarchically by the directory tree of the operating system's filesystem.

For every task graph an optimal schedule has to be computed and stored in the directory of the respective task graph category. The task graph's and schedule's file names differ only by their extension.

The directory tree, representing the test bench's structure is shown in figure 1. On the right side, one can see the total number of task graphs which are available within every partial subtree at the corresponding level.

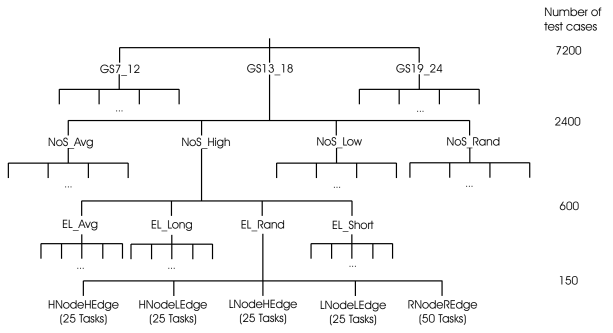

Figure 1. The test bench's structure.

3 Computation of the Schedules

3.1 Managing the computation of a huge number of test cases

The computation of the optimal schedules is carried out by a pruned Depth-first-Search-algorithm (DFS), a serial enhancement of the Branch-and-Bound algorithm described in [9]. Although the current version of this algorithm is much faster than its predecessor, the necessary computation time for computing an optimal schedule varies between a few milliseconds and several days. A prediction of the required computation time is almost impossible. Therefore, handling the test cases in a sequential way would lead to a delayed availability of useful intermediate results, because the algorithm can get stuck at the more difficult test cases for some time.

In order to achieve an early access to a large set of optimal schedules, we parallelized the computation process by using the time-slice-method known from multitasking operating systems: Every computation is interrupted after a predefined period of time. The current intermediate result and all information required to restart the scheduling process are accumulated to a checkpoint which is stored in a file. Every time the scheduling algorithm is started, it firstly checks the availability of a checkpoint to the given task graph and uses it as starting point in case of existence.

The scheduling algorithm is started by a shell-script that receives the file names of the task graph scheduling jobs in a work pool file. These file names are forwarded to the scheduling algorithm in a queue, always waiting for the end of the predecessor's computation. When every test case got its time slot, the script removes all task graphs from the input file whose schedule's computation is finished. The remaining file is used as input for the next turn of the computation. This procedure is repeated until the input file is empty.

The schedules are stored in the so called Standard-Schedule-Format, using 'ssf' as file name extension. This file format stores the number of tasks and the target architecture's size in the first row. Then, every task is described in a separate row, showing the task's index, the processor it is started on, its starting time and its finishing time.

3.2 Quality assurance by a second optimal algorithm

The claim of computing optimal schedules leads to high demands on the accurateness of the algorithm's planning and implementation. A hand-crafted quality check is infeasible, because of the problem's complexity. A comparison of the algorithm's and some heuristics' results is not meaningful, because heuristics do not find optimal solutions in general.

As a result of these procedures' incapability, we decided to realize a second optimal algorithm, whose results can be compared to those of the pruned DFS algorithm. A detailed description of this alternative implementation, that bases upon the well known IDA*-algorithm, can be found in [10].

This comparison comprised 3600 test cases. Although the optimal schedules themselves differed very often, their optimal schedule lengths were always equal. For this reason, we rate the reliability of our algorithm to be very high. In the meantime, new versions of the pruned DFS-algorithm are evaluated by comparing their results with those of the preceding version. This extended test set obviously still includes the results of the initial comparison with the IDA*-algorithm.

4 Results

4.1 Observations while computing the optimal results

The computation of the optimal schedules is performed on a PC-Cluster with 16 Computers (Pentium-4 2,6 GHz Processors). Using this system, from November 2003 until beginning of September 2004, 31756 of 36000 schedules were computed. It became obvious, that the required computation time does not only depend on the task graph's size, but also on its meshing degree and its edge lengths.

The higher the meshing degree or the shorter the edges, the faster the computation can be finished. The explanation of these observations is quite simple: Every additional edge reduces the search space's size, because it defines an order between the connected nodes. The alternative task order (and its sub-search-space) is of no relevance any more. Short edges reduce the search space as well, because a task has to be scheduled shortly after the last predecessor.

Considering the 4244 optimal schedules which still have to be computed, 53 belong to the medium category of size (13 to 18 tasks) and 4191 to the large one (19 to 24 tasks). All schedules of the small task graph category are computed – the completion of the medium category will take place soon. According to the discussion of the preceding paragraph, the majority of the task graphs whose optimal schedules are still not computed have either a low meshing degree (1787 test cases) or long edges (2592 test cases).

4.2 Comparison of scheduling heuristics

Using the interim version of the test bench, we compared the DLS- [1], ETF- [2], HLFET- [3] and MCP-heuristic [4]. This examination considers the quota of the overall found optimal solutions as well as a close investigation of the algorithms' strengths and weaknesses.

To our surprise, the heuristics show a very similar behavior regarding the quality of the obtained results. They exhibit nearly the same strengths and weaknesses, differing only by few percent. Figure 2 gives an overview of the heuristics' results with respect to the optimal solutions. Here, the most successful algorithm is the DLS-heuristic, followed by MCP, HLFET and ETF. The difference between the best and the worst algorithm is quite small (945 test cases $\approx$ 2,98 %).

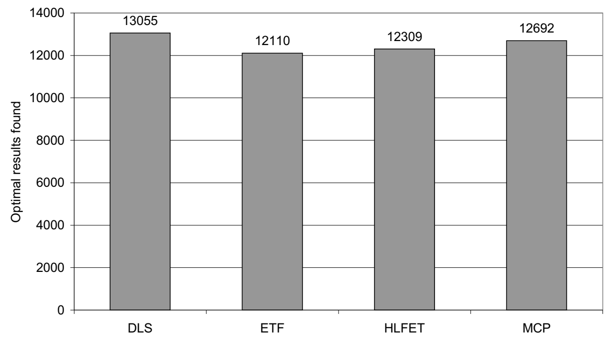

Figure 2. Comparison considering the number of found optimal solutions.

The investigated heuristics find optimal solutions more frequently, if the target architecture's size does not limit the parallelism of the schedule. Considering large architectures, the heuristics find more often the optimal solution as when a small system is used. Again, DLS performs best followed by MCP, HLFET and ETF. The success rate of MCP and HLFET differs by only 0,43 % for target systems with at least 8 processors. See figure 3 for more details.

All algorithms show an interesting behavior regarding the edge length: If the target system is small, the algorithms

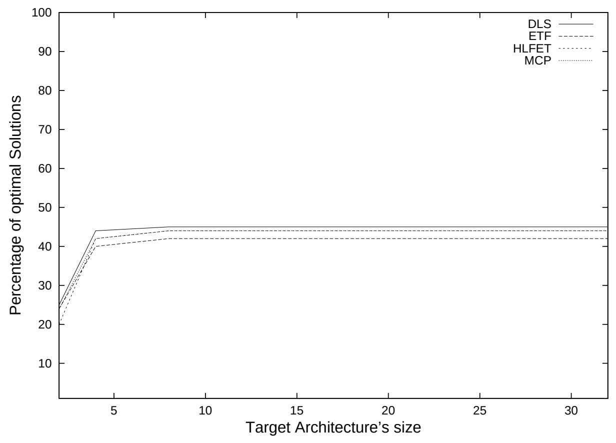

Figure 3. Influence of the target architecture's size.

find only half as much optimal solutions if the task graph has predominantly long edges, as for task graphs with a majority of short edges. For large target systems, the relation is almost balanced.

Figures 4 and 5 show the meshing degree's effect on the quality of the heuristics' results regarding a large target system. In this case, it seems to be more difficult to find an optimal solution, when task graphs are strongly meshed. In contrast to that, when the target system is small, a strong meshing seems to ease the heuristic's task. For both meshing degrees, DLS performs better than HLFET which again finds more optimal solutions than ETF. Considering the weakly meshed task graphs, MCP finds (0,61 %) more optimal schedules than DLS, but for the strongly meshed graphs MCP is just slightly (0,06 %) better than ETF.

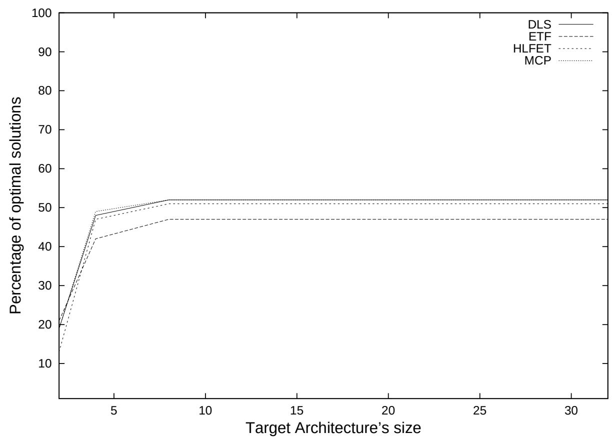

Figure 4. Effect of a low meshing degree (25%) on the quality of the heuristics' results.

Another conformity of the analyzed scheduling heuristics can be observed when the size of the task graphs is scaled. The percentage of found optimal solutions de-

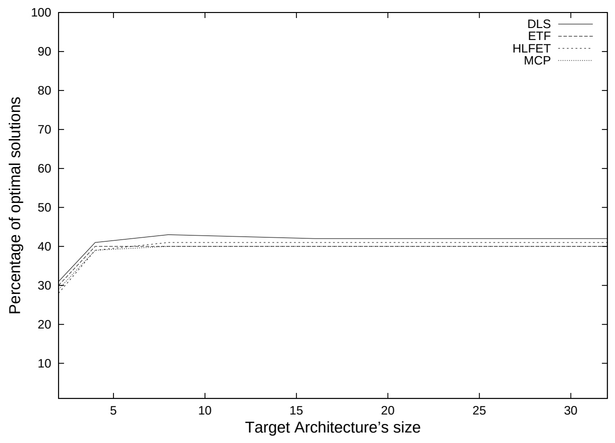

Figure 5. Effect of a high meshing degree (75%) on the quality of the heuristics' results.

creases, when the task graphs' size increases. See figure 6 for more details.

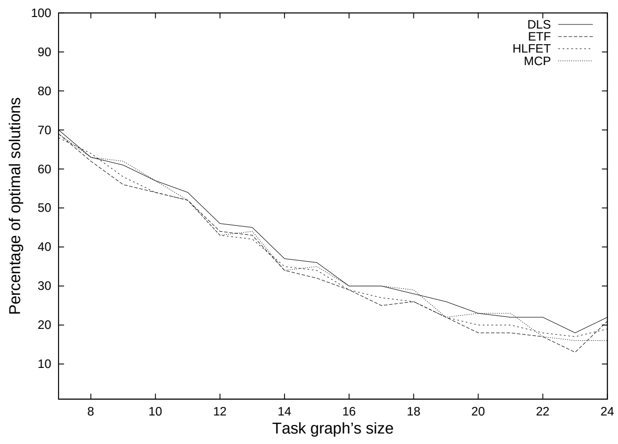

Figure 6. Impact of the task graphs' size.

The weights of nodes and edges effect the success rate of the compared scheduling heuristics as well. While all heuristics achieve good results regarding task graphs with heavy nodes or light edges, the quality drops down, when the test cases have light nodes or heavy edges predominantly. Table 1 shows the results for a target system's size of 32 processing elements.

Table 1: The influence of node- and edge weights on the percentage of optimal solutions found by the investigated heuristics.

	DLS	ETF	HLFET	MCP
Heavy Nodes	60,57	53,14	59,56	57,29
Heavy Edges	35,31	33,41	32,45	33,55
Light Nodes	34,33	34,52	31,66	34,86
Light Edges	59,63	54,30	58,80	58,66

5 Conclusion

In this paper, we presented a comprehensive test bench for the evaluation of task graph scheduling heuristics. It consists of 36000 task graph scheduling problems and their optimal solutions. By using this test bench, researchers are not only enabled to analyze the overall quality of their heuristics' results, they can also detect strengths and weaknesses of their algorithms. In addition, the heuristics' results can be compared more objectively to each other, facilitating an unbiased comparison of multiple scheduling algorithms.

Using the current interim version of the test bench, we compared the well known DLS-, ETF-, HLFET- and MCP-heuristics. Surprisingly, the heuristics show a very similar behavior regarding the quality of the obtained results. They exhibit the same strengths and weaknesses, differing only by few percent. We guess that these results will also hold for other heuristics not yet investigated by our benchmark suite.

As soon as all optimal schedules will be computed, we will publish our test bench on the web and hope that it will be used thoroughly as a benchmark in future research on task graph scheduling.

Acknowledgments: The authors would like to thank Mr. Markus Bank for implementing the investigated heuristics.

References

[1] G.C. Sih, E.A. Lee, A Compile-Time Scheduling Heuristic for Interconnection-Constrained Heterogeneous Processor Architectures, *IEEE Transactions on Parallel and Distributed Systems, 4*(2), 1993, 75–87

[2] J.J. Hwang, Y.C. Chow, F.D. Anger, C.Y. Lee, Scheduling Precedence Graphs in Systems with Interprocessor Communication Times, *SIAM Journal on Computing, 18*(2), 1989, 244–257

[3] T.L. Adam, K.M. Chandy, J. Dickson, A comparison of List Scheduling for Parallel Processing Systems, *Communications of the ACM, Vol. 17*, 1974, 685–690

[4] M.-Y. Wu, D.D. Gayski, Hypertool: A Programming Aid for Message-Passing Systems, *IEEE Transactions on Parallel and Distributed Systems, 1*(3), 1990, 330–343

[5] E. Bampis (ed.), Special Issue: Task scheduling problems for parallel and distributed systems, *Parallel computing, 25*(1), 1999

[6] M.-Y. Wu, W. Shu, J. Gu, Efficient Local Search for DAG Scheduling, *IEEE Transactions on Parallel and Distributed Systems*, 2001, 617–627

[7] S. Bansal, P. Kumar, K. Singh, An improved duplication strategy for scheduling precedence constrained graphs in multiprocessor systems, *IEEE Transaction on Parallel and Distributed Systems, 14*(6), 2003

[8] E.G. Coffman (ed.), *Computer and Job-Shop Scheduling Theory* (New York, John Wiley & Sons, 1976)

[9] U. Hönig, W. Schiffmann, A Parallel Branch–and–Bound Algorithm for Computing Optimal Task Graph Schedules, *Proceedings of the Second International Workshop on Grid and Cooperative Computing*, Shanghai, China, 2003, 747–755

[10] U. Hönig, W. Schiffmann, Fast optimal task graph scheduling by means of an optimized parallel A*-Algorithm, *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications*, Vol. II, Las Vegas, U.S.A., 2004, 842-848

[11] Kasahara Laboratory, http://www.kasahara.elec.waseda.ac.jp/schedule/index.html, 2004

[12] Y.-K. Kwok, I. Ahmad, Benchmarking the Task Graph Scheduling Algorithms, *Proceedings of the 12th International Parallel Processing Symposium*, Orlando, U.S.A., 1998, 531-537

[13] Y.-K. Kwok, I. Ahmad, Static Scheduling Algorithms for Allocating Directed Task Graphs to Multiprocessors, *ACM Computing Surveys, 31*(4), 1999, 406–471