# Improving the Efficiency of Functional Parallelism by Means of Hyper-scheduling

Udo Hönig and Wolfram Schiffmann
Department of Mathematics and Computer Science
University of Hagen
{Udo.Hoenig, Wolfram.Schiffmann}@FernUni-Hagen.de

## Abstract

*By means of a comprehensive test bench of 36000 test cases we evaluated the efficiency of functional parallel programs. For all the test cases schedules have been computed by various well known heuristics. We assumed a homogeneous target system (e.g. a compute cluster of equally powerful interconnected nodes) that can be part of a grid computing environment which supports the execution of parallel programs. Unfortunately, the efficiencies of the investigated schedules were pretty low. For this reason, we propose a new Hyper-scheduling approach that reduces the amount of idle times by interweaving subsequent schedules from the parallel job queue. First results confirm that Hyper-scheduling significantly improves efficiency.*

## 1 Introduction

In order to accelerate demanding computations one uses parallel processing. Three different kinds of parallelism can be distinguished: 1. Job Parallelism, 2. Data Parallelism, and 3. Functional Parallelism.

In the case of *job parallelism* we concurrently execute two or more independent but still sequential programs. This kind of parallel processing is quite simple to accomplish. Very often the same program is used with different input data. Typical applications are parameter studies or rendering numerous image frames for artificial movies.

*Data parallelism* can be characterized by compute demanding operations that must be applied to a great number of data items. Thus, one can run many copies of a program, which process different parts of the input data in the same manner. If there are no dependencies between the data, this kind of parallelism is the same as that of job parallelism. An example is the computation of the Mandelbrot set in a specific rectangular area. This is an embarrassingly parallel problem and can be easily parallelized. If some data areas depend on each other (e.g. when solving differential equations) the corresponding data between some processing elements have to be exchanged by communication over the network and the processing has to be synchronized as well.

If two or more different sequences of instructions can be executed concurrently we talk about *functional parallelism*. The instruction sequences of the parallel program define (sub)tasks, whose dependencies among each other can be described by means of a Directed Acyclic Graph (DAG), also known as a *task graph* [1]. Weights on the nodes specify the workload of a specific task and weights on the edges specify the communication overhead that must be taken into account, if the two connected subtasks are not assigned to the same processing element.

A special case of functional parallelism is the *workflow parallelism* [2, 3, 4]. Instead of tasks, workflows consist of a set of *programs* that depend on each other. Most current grid environments provide means for the composition and execution of those workflows. Before its execution, a workflow has to be specified by an appropriate workflow description language [3, 4, 5]. Data is usually exchanged via files. Input files are processed by one or more programs that produce intermediate files which are then forwarded to succeeding programs which cannot be launched before all required input files are available. In this way, workflow parallelism is very similar to functional parallelism and therefore it can be modeled by DAGs as well.

Within this paper, we assume a homogeneous target system (e.g. a compute cluster of 32 equally powerful nodes) that can be part of a grid computing environment which supports the execution of parallel programs or workflows. It is also assumed that each of the available computing nodes, we will call them "target processing elements" (TPE) throughout this paper, executes just one task at a time and that we have accurate estimates for the computation and communication times of the corresponding task graph problems. This scheduling model, which is widely used in literature (see [6], for further references), abstracts from many architectural specifics like memory-subsystems or caches

and therefore allows a platform independent analysis of the algorithms' properties. Because the computed schedules usually do not use all the available TPEs we will denote the schedules' processing elements by the abbreviation SPE. This distinction between different types of processing elements is also very important for the proposed Hyper-scheduling algorithm.

The primary contributions of this paper consist of an efficiency analysis of this DAG-modeled parallelism (parallel programs and workflows) and a new approach to improve the observed poor efficiency. The proposed Hyper-scheduling algorithm is based on already scheduled task graph problems and thus it can be used in combination with any scheduling algorithm.

The paper is organized as follows. In section 2 we describe the task scheduling problem in more detail and give an overview of algorithms for solving this problem. In section 3 we present an efficiency analysis of those algorithms with respect to a comprehensive test bench that comprises 36000 task graph problems with up to 250 task nodes. In section 4 we propose a Hyper-scheduling approach that can improve the observed poor utilization of the compute nodes. Finally, section 5 concludes the paper and gives a brief survey of the authors' future work on scheduling efficiency.

## 2 Minimizing the schedule length

The main objective of task graph scheduling is the minimization of the schedule length, which is the time period between the start of the first task and the completion of the last task. Since the scheduling problem is known to be NP-complete [7] in its general form, many researchers felt motivated to devise various heuristic algorithms. Unfortunately, only few attempts were made to find real optimal schedules even for small problem sizes [1, 8, 9]. The lack of optimal schedules forced researchers to analyze their newly developed algorithms in relation to several already established approaches. As such relative comparisons always depend on the selected set of algorithms, their results will probably change with every new combination.

In the past, another problem of scheduling heuristics analysis was the absence of a publicly available standard test set. Without such a test set, a comparison of different scheduling algorithms required the availability of their implementations. Additionally, every researcher had to create an own test set, making a comparison of scheduling heuristics harder or sometimes even impossible.

These shortcomings could only be tackled by means of a comprehensive test bench of task graph problems covering 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 using informed search algorithms [10, 11] and the task graphs' size must be limited in such a way that the optimal schedules can be found in a reasonable period of time. By considering these aspects, a comprehensive test set (Small Test Bench – STB) of 36000 task graph scheduling problems with up to 24 tasks was developed [12]. The task graphs were generated randomly and structured concerning the graphs' size, their meshing degree (MD), their average edge length[1] (EL) and their node- and edge-weights (heavy (H) or light (L)). In addition, the assumed number of available processing elements was considered as well. To emphasize a certain graph property (e.g. a high meshing degree), the random numbers were determined by a Gaussian distribution. Since this test bench should also provide test cases which are unbiased with respect to one or more graph attributes, subsets with uniform distributed attributes (random - (R)) were created as well. With respect to the fact that most publications in literature use larger task graphs with sometimes even more than 1000 tasks, the question arises, if the results produced by using such small task graphs are of any meaning to real world scheduling problems. In order to answer this question, we created another test bench (Large Test Bench – LTB) which follows the same structure as the one already described, but which consists of task graphs with up to 250 tasks. This limit was introduced firstly, to keep the time required to download the test bench acceptable (about 400 MB) and secondly, to facilitate its processing by standard PC equipment. Nevertheless, the requires hard disk space for all test cases and their corresponding schedules is about 4.8 GB. Of course, no optimal schedules could ever be computed for graphs of that size. Both test benches are available at

`http://valexia.fernuni-hagen.de/OptSchedHome.html`

By means of these two test sets, we analyzed and compared the behavior of several deterministic and stochastic scheduling heuristics. A description of the investigated deterministic algorithms, namely DLS, ETF, HLFET and MCP, can be found in [6]. The stochastic algorithms include a pure Random List Scheduler (RLS), a Genetic Algorithm (Genetic List Scheduler – GLS), a Simulated Annealing approach (Simulated Annealing List Scheduler – SALS) and an ant-colony-based list scheduler (Ant List Scheduler – AntLS) [13]. Further details on these stochastic algorithms as well as some results concerning the obtained schedules' lengths have already been published [14]. The cited paper also confirms that the STB's and LTB's results are highly correlated. Thus, we can conclude, that comparisons based on STB's optimal schedules will also be meaningful for the larger task graph problems of the LTB.

[1] The length of an edge is proportional to the number of nodes that are shortcut by this edge.

**Table 1. Maximum and average Speedup regarding both test sets**

| | Speedup | AntLS | DLS | ETF | GLS | HLFET | MCP | RLS | SALS |
|---|---|---|---|---|---|---|---|---|---|
| STB | max. | 5.80 | 5.80 | 5.80 | 5.80 | 5.80 | 5.80 | 5.80 | 5.80 |
| | avg. | 1.60 | 1.55 | 1.55 | 1.59 | 1.54 | 1.55 | 1.57 | 1.60 |
| LTB | max. | 14.58 | 14.77 | 14.10 | 14.33 | 14.71 | 14.64 | 13.92 | 14.33 |
| | avg. | 2.07 | 2.04 | 2.03 | 2.04 | 2.01 | 2.05 | 1.93 | 2.09 |

## 3 Evaluation of efficiency

Publications about scheduling problems and algorithms usually focus on the achieved schedules' length and speedup. In very few cases only, the number of used SPEs and the efficiency, whose definition [15] is shown in equation 1 are considered as well. An efficient schedule combines a high speedup with a low number of used SPEs. The benefit of an efficiently used computer system is an improved division of the costs incurred and therefore either reduced costs for the particular users or, in case of an commercial provider, a better asset for the resources' provision.

$$Efficiency = \frac{Speedup}{Number\ of\ SPE} * 100 \qquad (1)$$

Before one can analyze the efficiency of an heuristic's schedule, one has to examine the number of SPE this schedule uses and its speedup. Since the analysis of a single schedule would not be representative for a given heuristic, we will present average values when discussing the speedup and the number of used SPEs.

Table 1 shows the maximum and average speedup of the observed heuristics' schedules. To point up that the results are very similar for smaller and larger task graph scheduling problems, we will consider both test sets simultaneously. Later, we will focus on the larger task graphs only. As can be seen in the first row of table 1, all heuristics achieve the same maximum speedup of 5.8 when considering the small task graphs. This value is equal to the best speedup found by the optimal algorithm. Concerning the larger test cases, the optimal values differ from algorithm to algorithm. Nevertheless, the maximum speedups achieved by the heuristics are still quite similar. The average speedup of the smaller task graph scheduling problems is always very low, ranging from 1.54 to 1.60. Keeping in mind, that the average speedup of the optimal schedules is 1.61, this result is surprisingly good. Row four of table 1 reveals a very low average speedup for the large task graph scheduling problems with a maximum value of 2.09. Before conducting this analysis, we expected a higher value, because the task graphs had up to 250 tasks and the expected target architectures had up to 32 TPE. The observed values can be reasoned by the existing data dependencies, which inhibit a larger-scale parallelization. Anyway, these results show a still considerable acceleration, which can justify the additional efforts of parallel program execution.

As already mentioned in section 2, the considered test sets are structured regarding several task graph properties and system sizes. Table 2 shows the average number of SPEs with respect to the LTB's scheduling problems with given properties. Obviously, the HLFET heuristic's schedules require more SPEs then those of the other heuristics. In contrast, ETF uses the available resources more economically than the other considered algorithms. Overall, the difference between the best and the worst scheduling algorithm is rather small in all cases.

Table 2 also shows the impact of some task graph properties on the number of required SPEs. Obviously, for task graphs with either a low meshing degree, long edges or low communication costs, the corresponding schedules use more SPEs than on average. This observation can be explained as follows: A low meshing degree means a low number of dependencies and thus more tasks that can be executed in parallel. Similarly, long edges cause a low number of dependencies between the task graphs first tasks and therefore a small number of tasks that have to be executed sequentially. Finally, low communication costs reduce the effect of the existing data dependencies because the receiving node can be executed earlier.

Another observation of this analysis reveals, that the number of used SPEs is usually much smaller than that of the available TPEs. The larger the target architecture, the smaller is the percentage of the SPEs used. Concerning target systems with 32 TPEs, less than one third of them is used on average.

Now as we know, that only a small fraction of the available TPEs is really involved in the parallel programs' execution, our next objective is to find out how efficiently the SPEs are used. For this reason, we calculate the efficiency of every task graph scheduling problem separately. Then, we calculated the average values for all test cases with given properties. As can be seen in table 3, the analyzed heuristics achieve similar results. With the exception of very small target architectures, the obtained values are below 60%. Efficiency boosting task graph properties are a low meshing degree, long edges or low communication costs. These ob-

**Table 2. Average number of SPEs used for the LTB**

| | AntLS | DLS | ETF | GLS | HLFET | MCP | RLS | SALS |
|---|---|---|---|---|---|---|---|---|
| 2 TPE | 2.00 | 2.00 | 2.00 | 2.00 | 2.00 | 2.00 | 1.99 | 2.00 |
| 4 TPE | 3.94 | 3.94 | 3.86 | 3.87 | 3.96 | 3.87 | 3.87 | 3.93 |
| 8 TPE | 6.53 | 6.38 | 5.86 | 5.99 | 6.82 | 5.92 | 5.95 | 6.47 |
| 16 TPE | 8.55 | 8.37 | 7.78 | 7.95 | 8.97 | 7.86 | 7.90 | 8.49 |
| 32 TPE | 10.23 | 10.04 | 9.45 | 9.62 | 10.65 | 9.54 | 9.57 | 10.17 |
| All TPE | 6.25 | 6.14 | 5.79 | 5.88 | 6.48 | 5.84 | 5.86 | 6.21 |
| MD Low | 7.15 | 6.99 | 6.59 | 6.70 | 7.41 | 6.65 | 6.68 | 7.09 |
| MD Avg | 6.13 | 6.04 | 5.71 | 5.80 | 6.35 | 5.75 | 5.77 | 6.10 |
| MD High | 5.33 | 5.29 | 5.02 | 5.09 | 5.48 | 5.06 | 5.07 | 5.32 |
| MD Rand | 6.39 | 6.26 | 5.84 | 5.95 | 6.68 | 5.89 | 5.92 | 6.34 |
| EL Long | 10.16 | 10.15 | 10.14 | 10.15 | 10.16 | 10.15 | 10.15 | 10.15 |
| EL Short | 4.23 | 4.11 | 3.58 | 3.65 | 4.53 | 3.61 | 3.62 | 4.15 |
| EL Avg | 5.86 | 5.76 | 5.49 | 5.67 | 6.09 | 5.58 | 5.63 | 5.89 |
| EL Rand | 4.75 | 4.56 | 3.95 | 4.07 | 5.14 | 4.00 | 4.03 | 4.65 |
| LNode LEdge | 6.25 | 6.12 | 5.74 | 5.86 | 6.46 | 5.79 | 5.82 | 6.21 |
| LNode HEdge | 5.92 | 5.81 | 5.55 | 5.59 | 6.21 | 5.57 | 5.58 | 5.87 |
| HNode LEdge | 6.51 | 6.41 | 6.08 | 6.16 | 6.68 | 6.13 | 6.13 | 6.47 |
| HNode HEdge | 6.31 | 6.19 | 5.84 | 5.95 | 6.53 | 5.91 | 5.90 | 6.28 |
| RNode REdge | 6.26 | 6.17 | 5.77 | 5.88 | 6.50 | 5.81 | 5.86 | 6.22 |

**Table 3. Efficiency of the SPEs' usage for the LTB (in %)**

| | AntLS | DLS | ETF | GLS | HLFET | MCP | RLS | SALS |
|---|---|---|---|---|---|---|---|---|
| 2 TPE | 84.16 | 83.72 | 84.39 | 83.73 | 80.56 | 84.06 | 78.59 | 86.02 |
| 4 TPE | 58.66 | 58.26 | 58.93 | 58.46 | 56.59 | 59.29 | 54.42 | 59.27 |
| 8 TPE | 40.84 | 41.00 | 44.28 | 43.58 | 38.19 | 44.30 | 41.08 | 41.23 |
| 16 TPE | 33.54 | 33.57 | 37.07 | 36.76 | 30.60 | 37.02 | 35.02 | 33.90 |
| 32 TPE | 30.85 | 30.94 | 34.47 | 34.19 | 27.98 | 34.34 | 32.67 | 31.27 |
| All TPE | 49.61 | 49.50 | 51.83 | 51.34 | 46.78 | 51.80 | 48.35 | 50.34 |
| MD Low | 53.67 | 53.84 | 56.04 | 54.96 | 50.79 | 56.05 | 51.30 | 54.51 |
| MD Avg | 48.88 | 48.75 | 51.00 | 50.68 | 46.10 | 50.95 | 47.80 | 49.62 |
| MD High | 46.85 | 46.24 | 48.62 | 48.82 | 44.05 | 48.50 | 46.47 | 47.35 |
| MD Rand | 49.03 | 49.17 | 51.64 | 50.91 | 46.19 | 51.70 | 47.85 | 49.87 |
| EL Long | 63.46 | 62.77 | 62.78 | 62.61 | 61.15 | 63.30 | 58.63 | 63.43 |
| EL Short | 40.03 | 39.83 | 43.70 | 43.82 | 37.04 | 43.59 | 42.00 | 41.02 |
| EL Avg | 52.88 | 52.92 | 54.37 | 52.98 | 50.01 | 54.02 | 49.40 | 53.53 |
| EL Rand | 42.07 | 42.47 | 46.46 | 45.96 | 38.93 | 46.30 | 43.39 | 43.37 |
| LNode LEdge | 50.13 | 49.92 | 52.56 | 51.93 | 47.31 | 52.47 | 48.96 | 50.69 |
| LNode HEdge | 40.32 | 39.79 | 42.11 | 42.31 | 35.72 | 42.12 | 39.29 | 41.27 |
| HNode LEdge | 56.70 | 57.25 | 59.14 | 58.16 | 55.46 | 59.12 | 55.64 | 57.21 |
| HNode HEdge | 50.66 | 50.95 | 52.98 | 52.15 | 48.34 | 52.75 | 49.04 | 51.50 |
| RNode REdge | 49.92 | 49.53 | 52.09 | 51.76 | 46.93 | 52.17 | 48.59 | 50.68 |

servations can be reasoned by the reduced effect of the existing communication dependencies.

It can therefore be summarized, that the functional parallel programs given by the LTB have a low degree of parallelism and thus can only use a small subset of the available resources. Furthermore, the actually occupied resources are used very inefficiently, posing the question if the parallelization of functional parallel programs is rational. In some cases, the achievable speedup might justify this wasteful use of expensive resources. Nevertheless, new methods are required which improve the target architectures' utilization.

## 4 Maximizing the utilization by Hyper-scheduling

As we have seen in the previous section, the computed schedules show a poor efficiency of predominantly less than 60%. This is the price we have to pay for shortening the execution time by means of parallelism. Due to the constraints imposed by the DAG, the TPEs of the target system are not utilized permanently.

Fortunately, the users submit a great number of parallel programs and thus it is obvious that the efficiency could be improved by interweaving multiple subsequent schedules. Very often, when two (or even more) subsequent schedules require less SPEs than TPEs are available, the corresponding parallel programs can even be processed in parallel completely.

In order to preserve the benefits of parallel processing, the parallel programs should be executed as soon as possible in the order they have been submitted. Thus, they will be entered into a parallel job queue from where they will be successively scheduled by a heuristic. The basic idea of the here proposed *Hyper-scheduling* approach is to reduce the amount of idle times by interweaving subsequent DAG-schedules from the parallel job queue. For this purpose, we select a DAG-schedule from the top of the parallel job queue. Its SPEs must then be assigned and aligned (in time) to the best fitting TPEs.

To accomplish this, we sort the TPEs according to their *finishing* times in ascending order. Similarly, the SPEs of a selected DAG-schedule are also ordered in ascending order with respect to their *starting* times. In Figure 1 an ideal matching situation is depicted. Here, the size of the target system is equal to the number of the required SPEs. In general, there will be more TPEs available than SPEs needed by the DAG-schedule. In this case, we assign the number of required SPEs to the first TPEs.

Next, we have to align the DAG-schedule relating to the needed TPEs' utilization. Initially, we align the earliest task of the first SPE to the first TPE and check if this would result in an overlap between the first tasks of the other SPEs

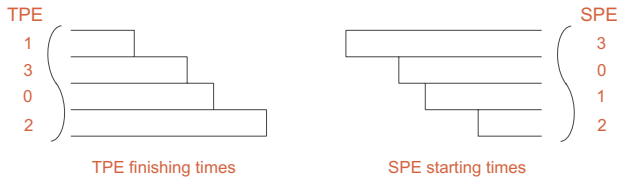


**Figure 1. Principle of assignment and alignment illustrated by a perfect fitting schedule**

with the utilization of the corresponding TPEs. Note, that the relative positions of the tasks in the subsequent DAG-schedule must be preserved.

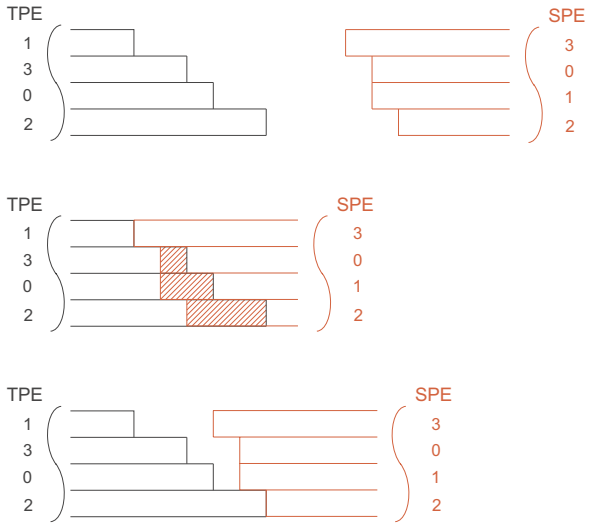


**Figure 2. Aligning a non-perfect fitting DAG-schedule**

Aligning the DAG-schedule as described above can result in an overlapping between the TPEs' and SPEs' busy time. An example with a nonfitting DAG-schedule is shown in Figure 2. In this example we have to delay the whole DAG-schedule by the maximum amount of all the overlaps.

Even if this delay will leave some time periods of idle TPEs, the sketched Hyper-scheduling approach will still improve the target system's utilization. Only in the worst case that all the SPEs are busy from the beginning, no improvements could be achieved.

In Figure 3 some example DAG-schedules and the resulting hyper-schedule are shown. In order to guarantee that the SPEs will be executed as scheduled, the TPEs must be reserved for the corresponding time periods.

We analyzed this algorithm by means of the already described test bench with task graphs ranging from 25 to 250

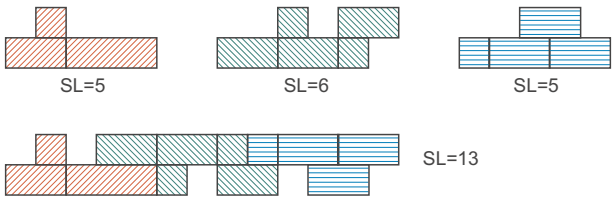


**Figure 3. Hyper-scheduling of three example DAG-schedules**

nodes. Because the scheduling heuristics behave very similar in most cases, we will focus on schedules computed by the HLFET-heuristic for target architectures consisting of 4 and 8 TPE within this work. By using a small system size of 4 TPE, we avoid side effects caused by the parallel execution of multiple programs as it can happen in larger architectures. Therefore, the improvements presented in table 4 can unambiguously assigned to the interweaving of subsequent DAG-schedules. Because a larger number of processing elements will provide a better basis for the Hyper-scheduler's operation, the results presented in table 4 have to be understood as some kind of lower bound of the algorithm's capability. In order to verify this assertion, some results obtained for target architectures of 8 TPE are added as well (see table 5). An analysis of larger target architectures was not included, because the low average number of required processing elements (see table 2 for further information) would facilitate the parallel execution of the parallel programs to a very high degree.

In order to get meaningful values, we generated 25 sequences of 10 randomly selected schedules for each considered test case property. These sequences were forwarded to the Hyper-scheduler for processing. The first column of table 4 shows the average values one obtains when placing the schedules without, the second column those when placing the schedules with the proposed Hyper-scheduler. As can be easily seen, interweaving subsequent DAG-schedules can improve the efficiency by up to approximately 2 %. The table's third column shows the additional speedup when applying our algorithm. This value ranges from 1.00004 to 1.04929 indicating a maximum speedup of almost 5 %. When considering individual sequences, these values can be nearly doubled compared to the average ones.

Maximum increase can be observed for test cases with a high meshing degree, because the corresponding task graphs tend to have a small number of starting nodes and are therfore particularly suitable for Hyper-scheduling. In contrast, task graphs with predominantly long edges usually have a large number of starting tasks which are scheduled as soon as possible by most scheduling algorithms. For this reason, the Hyper-scheduler achieves poor results when applied on test cases basing on task graphs with predominantly long edges.

**Table 4. The effect of Hyper-scheduling (HS) on efficiency and speedup regarding target architectures of 4 TPE**

| | Efficiency without HS (in %) | Efficiency with HS (in %) | additional Speedup |
|---|---|---|---|
| MD Low | 62.851 | 63.205 | 1.00563 |
| MD Avg | 52.383 | 53.310 | 1.01769 |
| MD High | 40.854 | 42.868 | 1.04929 |
| MD Rand | 52.806 | 53.503 | 1.01319 |
| EL Long | 86.135 | 86.139 | 1.00004 |
| EL Short | 36.018 | 37.590 | 1.04364 |
| EL Avg | 63.977 | 64.293 | 1.00493 |
| EL Rand | 41.147 | 42.153 | 1.02444 |
| LNode LEdge | 51.813 | 52.586 | 1.01491 |
| LNode HEdge | 36.618 | 37.728 | 1.03031 |
| HNode LEdge | 65.464 | 66.373 | 1.01388 |
| HNode HEdge | 51.654 | 52.528 | 1.01692 |
| RNode REdge | 50.274 | 51.186 | 1.01814 |

Table 5 shows the results when considering target architectures of 8 processing elements. As expected, the measured values are much better than for the smaller system size. Again, the proposed algorithm shows the best improvements for test cases with a high meshing degree. The average Speedup measured for these test cases was 1.49, the maximum value we observed for a single sequence was 2.07. By applying the Hyper-Scheduler, the efficiency was improved by 10.45 % on average and 20.005% on maximum. Again, the worst results were found for test cases with predominantly long edges. It is obvious, that the values shown in table 5 are affected by the fact that the average number of used TPE is only 6.82 (see table 2 for details). Nevertheless, it is very unlikely that two or more schedules can be completely executed in parallel because they require only very few processing elements.

## 5 Conclusion

In this paper, we used a huge test bench with a total of 36000 test cases and up to 250 task nodes to compare different DAG-scheduling heuristics regarding their speedup and efficiency. Because the used test bench covers a broad spectrum of conceivable task graph problems, we also analyzed and discussed the influence of various task graph properties on the efficiency. Essentially, we found out that the efficiency of the DAG-schedules is mostly below 60% which means, that a lot of the provided computing power

**Table 5. The effect of Hyper-scheduling (HS) on efficiency and speedup regarding target architectures of 8 TPE**

| | Efficiency without HS (in %) | Efficiency with HS (in %) | additional Speedup |
|---|---|---|---|
| MD Low | 35.315 | 36.989 | 1.04740 |
| MD Avg | 28.364 | 32.315 | 1.13929 |
| MD High | 21.334 | 31.790 | 1.49010 |
| MD Rand | 28.180 | 31.147 | 1.10528 |
| EL Long | 64.544 | 64.600 | 1.00086 |
| EL Short | 18.011 | 25.544 | 1.41824 |
| EL Avg | 33.973 | 36.911 | 1.08648 |
| EL Rand | 20.609 | 25.275 | 1.22640 |
| LNode LEdge | 27.803 | 31.667 | 1.13897 |
| LNode HEdge | 18.988 | 23.219 | 1.22282 |
| HNode LEdge | 37.255 | 40.940 | 1.09891 |
| HNode HEdge | 27.746 | 31.520 | 1.13601 |
| RNode REdge | 26.887 | 31.022 | 1.15379 |

is wasted. For this reason, we proposed a new Hyper-scheduling algorithm which ameliorates the efficient use of the available resources. The presented results show a significant improvement of efficiency and speedup.

In our future work we will extend this approach to heterogeneous target systems which represent more appropriate models for real grid environments. In addition, we plan to develop an alternative approach based on a Multi-DAG-Scheduling concept.

**Acknowledgment** The authors would like to thank Rajkumar Buyya and Anthony Sulistio from Melbourne University for motivating our research on the topic of scheduling efficiency.

## References

[1] Kwok, Y.-K., Ahmad, I.: Benchmarking and Comparison of the Task Graph Scheduling Algorithms, Journal of Parallel and Distributed Computing, Vol. 59, No. 3, pp. 381–422, Academic Press, Inc., 1999

[2] Deelman E. et al.: Pegasus: Mapping Scientific Workflows onto the Grid, European Across Grids Conference, pp. 11–20, Springer, 2004

[3] Yu, J., Buyya, R.: A taxonomy of scientific workflow systems for grid computing, SIGMOD Rec., Vol. 34, No. 3, pp. 44–49, ACM Press, 2005

[4] Wieczorek M., Prodan R., Fahringer T., Scheduling of scientific workflows in the ASKALON grid environment, SIGMOD Rec., Vol. 34, No. 3, pp. 56–62, ACM Press, 2005

[5] Fahringer T., Pllana S., Villazón A.: A-GWL: Abstract Grid Workflow Language, International Conference on Computational Science, Springer, pp. 42–49, 2004

[6] Kwok, Y.-K., Ahmad, I.: Static Scheduling Algorithms for Allocating Directed Task Graphs to Multiprocessors, ACM Computing Surveys, Vol. 31, No. 4, pp. 406–471, ACM Press, 1999

[7] Coffman, E.G.: Computer and Job-Shop Scheduling Theory, John Wiley & Sons, 1976

[8] Kwok, Y.-K., Ahmad, I.: Efficient Scheduling of Arbitrary Task Graphs to Multiprocessors Using a Parallel Genetic Algorithm, Journal of Parallel and Distributed Computing, Vol. 47, No. 1, pp. 58–77, Academic Press, Inc., 1997

[9] Ahmad, I., Kwok, Y.-K.: Optimal and Near-Optimal Allocation of Precedence-Constrained Tasks to Parallel Processors: Defying the High Complexity Using Effective Search Techniques, ICPP '98: Proceedings of the 1998 International Conference on Parallel Processing, pp. 424–431, IEEE Computer Society, 1998

[10] Hönig, U., Schiffmann, W.: A Parallel Branch–and–Bound Algorithm for Computing Optimal Task Graph Schedules, Proceedings of the Second International Workshop on Grid and Cooperative Computing (GCC'03), LNCS 3033, pp. 18–25, Springer-Verlag, 2004

[11] Hönig, U., Schiffmann, W.: 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 (PDPTA'04), Vol. 2, pp. 842–848, CSREA Press, 2004

[12] Hönig, U., Schiffmann, W.: A comprehensive Test Bench for the Evaluation of Scheduling Heuristics, Proceedings of the sixteenth IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS '04), pp. 437–442, 2004

[13] Bank, M., Hönig, U., Schiffmann, W.: An ACO-based approach for scheduling task graphs with communication costs, Proceedings of the 2005 International Conference on Parallel Processing (ICPP 2005), pp. 623–629, IEEE Computer Society, 2005

[14] Hönig, U., Schiffmann, W.: Comparison of nature inspired and deterministic scheduling heuristics considering optimal schedules, Adaptive and Natural Computing Algorithms, pp. 361–364, Springer Wien NewYork, 2005

[15] Bansal, S., Kumar, P., Singh, K.: An improved duplication strategy for scheduling precedence constrained graphs in multiprocessor systems, IEEE Transactions on Parallel and Distributed Systems, Vol. 14, No. 6, IEEE Press, 2003