A Parallel Branch–and–Bound Algorithm for Computing Optimal Task Graph Schedules

Udo Hönig and Wolfram Schiffmann

FernUniversität Hagen, Lehrgebiet Technische Informatik I, 58084 Hagen, Germany
{Udo.Hoenig|Wolfram.Schiffmann}@FernUni-Hagen.de
http://www.informatik.ti1.fernuni-hagen.de/

Abstract. In order to harness the power of parallel computing we must firstly find appropriate algorithms that consist of a collection of (sub)tasks and secondly schedule these tasks to processing elements that communicate data between each other by means of a network. In this paper, we consider task graphs that take into account both, computation and communication costs. For a homogeneous computing system with a fixed number of processing elements we compute all the schedules with minimum schedule length. Our main contribution consist of parallelizing an informed search algorithm for calculating optimal schedules based on a Branch–and–Bound approach. While most recently proposed heuristics use task duplication, our parallel algorithm finds all optimal solutions under the assumption that each task is only assigned to one processing element. Compared to exhaustive search algorithms this parallel informed search can compute optimal schedules for more complex task graphs. In the paper, the influence of parameters on the efficiency of the parallel implementation will be discussed and optimal schedule lengths for 1700 randomly generated task graphs are compared to the solutions of a widely used heuristic.

1 Introduction

A task graph is a directed acyclic graph (DAG), that describes the dependencies between the parts of a parallel program [8]. In order to execute it on a cluster or grid computer, it's tasks must be assigned to the available processing elements. Most often, the objective of solving this *task graph scheduling problem* is to minimize the overall computing time. The time that a task i needs to compute an output by using the results from preceding tasks corresponds to the working load for the processing element to which that task is assigned. It is denoted by a node weight w_i of the task node. The cost of communication between two tasks i and j is specified as an edge weight c_{ij}. If both tasks are assigned to the *same* processor, the communication cost is zero.

Task graph scheduling comprises two subproblems. One problem is to assign the tasks to the processors, the other problem consists of the optimal sequencing of the tasks. In this paper, we suppose a homogeneous computing environment, e.g. a cluster computer. But, even if we assume identical processing elements, the

problem to determine an *optimal* schedule has been proven to be NP–complete, apart from some restrained cases [8]. Thus, most researchers use heuristic approaches to solve the problem for reasonable sizes of the task graph. Three categories can be distinguished: list–based, clustering–based and duplication–based heuristics. List–based heuristics assign priority levels to the tasks and map the highest priority task to the best fitting processing element [10]. Clustering–based heuristics embrace heavily communicating tasks and assign them on the same processing element, in order to reduce the overall communication overhead[1]. Duplication-based heuristics also decrease the amount of communication while simultaneously the amount of (redundant) computation will be increased. It has been combined with both list-based [2] and cluster-based approaches [9]. In order to evaluate the quality of all those heuristics in a unified manner it would be desirable to compare the resulting schedules lengths of those heuristics to the optimal values. The parallel Branch–and–Bound algorithm proposed in this paper can be used to create a benchmark suite for this purpose. Of course, this is only feasible for task graphs of moderate size (e.g. with lower than 30 tasks).

Usually, there are multiple optimal schedules that provide a solution set of the task graph scheduling problem. All these schedules are characterized by the same minimal schedule length. To compute these optimal schedules we have to search for all the possible assignments and sequences of the tasks.

The simplest algorithm to compute the set of optimal schedules enumerates all possible solutions and stores only the best ones. But, even for a small number of tasks the number of solutions will be enormous. If we want to get the set of optimal schedules in an acceptable period of time and with maintainable memory requirements we have to devise a more skillful algorithm.

The basic idea to shorten the effort of an exhaustive search is to perform a structured or *informed* search that reduces the state space. Thus, the number of schedules that have to be investigated is much smaller than the number of possible processor assignments multiplied by the number of possible sequences. In this way, an informed search can manage more complex task graphs than exhaustive search strategies. The informed search is often based on a A* algorithm ([3], [5]). In this paper, we will present a Branch–and–Bound approach and its implementation on a parallel virtual machine [4].

The paper is organized as follows. In the next section, the concepts of the Branch–and–Bound algorithm will be explained. The third section is concerned with the parallelization of that algorithm. It describes how the workload is partitioned and how load balancing will be achieved. In the fourth section, we present results and discuss the influence of various parameters on the efficiency of the parallel implementation.

2 Branch–and–Bound algorithm

If we want to shorten the time to compute the set of optimal solutions, we are not allowed to consider every valid schedule (assignment of the tasks to the processing elements plus determination of the tasks' starting times). Instead,

we have to divide the space of possible schedules into subspaces that contain *partially similar* solutions.

These solutions should have in common that a certain number of tasks is already scheduled in the same way. The corresponding subspace contains all the schedules that are descended from the partial schedule but differ in the scheduling of the remaining tasks. Each partial schedule can be represented by a node in a decision-tree. Starting from the root, which represents the empty schedule, all possible schedules will be constructed. At any point of this construction process, we can identify a set of tasks that are ready to run and a set of idle processing elements to which those tasks can be assigned. In this way, each conceivable combination will produce a new node in the decision tree (*Branch*).

Supposed we have an estimate t_{best} for the total schedule length, we can exclude most of the nodes that are created in the Branch part of the algorithm. This estimate can be initialized by any heuristic. Here, we used the heuristic from Kasahara and Narita [6]. After the creation of a new partial schedule (node of the decision tree), we can estimate a lower bound of it's runtime $t_{partial}$ by means of it's current schedule length and the static b-level values of the remaining (yet unscheduled) tasks. The lower bound $t_{partial}$ is computed by the sum of the partial schedule length plus the maximum of the static b-level values. If $t_{partial}$ is greater than t_{best}, we can exclude the newly created node from further investigation. By this deletion of a node (*Bound*) we avoid the evaluation of all the schedules that depend on the corresponding partial schedule (subspace of the search space). In this way, we accelerate the computation of the solution set for the task graph problem.

As long as a node's $t_{partial}$ is lower or equal to the current t_{best} we continue to expand this node in a depth-first manner. When all the tasks of the graph are scheduled, a leaf of the decision tree is reached. If $t_{best} = t_{complete}$ we add the corresponding schedule to the set of best schedules. If $t_{best} > t_{complete}$ we clear the set of best schedules, store the new (complete) schedule into the set of best schedules and set $t_{best} = t_{complete}$. Then we continue with the next partial schedule.

The pruning scheme above is further enhanced by a selection heuristic that controls the order of the creation of new nodes. By means of this priority controlled breadth-first search we improve the threshold t_{best} for pruning the decision tree as early as possible. Likewise to the Bound phase, this procedure reduces further the total number of evaluations. By proceeding as described above, all possible schedules are checked. At the end of the search procedure, the current set of best schedules represents the optimal schedules for the task graph problem.

3 The parallel algorithm

The parallelisation of the sequential Branch–and–Bound algorithm requires a further subdivision of the search-space into disjunct subspaces, which can be assigned to the processing units.

As already described in Section 2, every inner node of the decision-tree represents a partial schedule and every leaf node corresponds to a complete schedule. The branching rule used by the algorithm, guarantees that the sons of a node will represent different partial schedules. Since the schedules are generated along the timeline, a later reunification of the subtrees, rooting in these sons, is impossible. Therefore two subtrees of the decision-tree always represent disjunct subspaces of the search-space, if none of their roots is an ancestor of the other one. Another result of these thoughts is that every part of the search-space can unambiguously be identified by it's root-node.

In order to achieve a balanced assignment of the computation to the available processing units, the algorithm generates a workpool, containing a certain number of subtree-roots. This workpool is managed by a master-process, which controls the distribution of the tasks to the slave-processes.

The workpool is created by means of a breadth-first-search which terminates, when a user defined number of elements is collected in the workpool. The nodes are numbered by the ordinal numbers of the nodes' permutations. These ordinal numbers allow an unambiguous identification of the nodes. By means of the root's ordinal number, it is possible to build the corresponding subtree. For that reason, the only information, that need to be stored in the workpool, are the ordinal numbers of the subtrees' roots. This helps to keep the required memory small.

The parallel algorithm can be partitioned into three parts, called Initialisation, Computation and Finalisation.

During the *Initialisation*-Phase, the master launches the slave-processes and splits the whole task into a number of smaller subtasks. The number of subtasks depends on the size of the workpool, which is specified by the user.

The *Computation*-Phase begins as soon as the master assigns a subtask to every slave[1]. Then, the master has to wait until it receives a message from one of the slaves which indicates that the slave has completely analysed a given subtree or that it has found an improved schedule. In the last case, the master only stores the broadcasted schedule length and the process-id of the sending slave. In the other case, it sends a new subtask to the slave if the workpool is not empty, otherwise the slave will stay idle.

As soon as all subtasks are processed and all slaves are idle, *Finalisation*-Phase will take place. The master informs all slaves about the end of the computation-process. This request is necessary, because the sending of messages appears asynchronously and it is not guaranteed, that every message that indicates a new best solution was already sent and received. Every slave receiving the finalisation message, compares the global best solution to it's own recent results and possibly deletes it's own suboptimal results. If the slave recognizes, that it's own recent temporal solution is better than the global best solution, it sends an appropriate broadcast to the master and to all other slaves. Then, the slave sends the master an acknowledgement to indicate, that it finished the adjustment successfully. The master waits, until it receives an acknowledgement from every slave. Then

[1] It is required that the workpool contains more tasks than slaves.

it requests the complete schedule from the last slave that reported to have found the best solution so far. Additionally it requests some bookkeeping information of all slaves. The slaves terminate after sending their replies. Finally, the master creates the output-file and terminates as well.

4 Results

To achieve an efficient informed search algorithm, there are some constraints that should be analysed before starting the computation of larger problems. It was found, that some of the most important constraints that influence the search-speed are independent of the given task graph. These aspects belong to the algorithm's properties such as the size of the workpool and the number of processing elements that are involved in the search-process.

Additionally, we demonstrate the suitability of our approach for the evaluation of scheduling heuristics. For this purpose, we analyse the heuristic of Kasahara and Narita [6] using a test bench of approximately 1700 task graphs.

4.1 Size of the workpool

After it's creation, the workpool includes the complete search space, subdivided into a user-defined number of subspaces. Apparently, the size of a subspace is determined by the number of schedules that it contains. If the number of subspaces will be increased, the size of every subspace will be reduced. In this way the workpool's size determines the granularity of the search space and the number of schedules one slave has to analyse.

Figure 1 shows how the runtime for different task graph problems depends on the size of the workpool. On the left side, we see how an increase of the workpool size can reduce the runtime on a parallel computing system with 30 processing elements. It is clearly visible that a workpool size between approx. 300 to 1200 elements (partial schedules) will be useful in this case. In contrast, we see on the right side of figure 1, that the situation for light-weight scheduling problems changes to get worse when using a parallel implementation. In this case, the minimal runtime is reached with the sequential implementation and no subtasks at all. The relative slowdown increases with the workpool's size and might become clearly more than 100 %.

Since it is difficult or even impossible to estimate the computational complexity of a schedule, the workpool size has to be chosen carefully in order to minimize the overall runtime.

4.2 Number of processing elements

Usually, the maximum speedup of a parallel program will be equal to the number of the available processing elements. In order to evaluate the scaling of the parallel implementation we used three task graphs that had sequential runtimes of approximately one minute each. The workpool size was set to 6000.

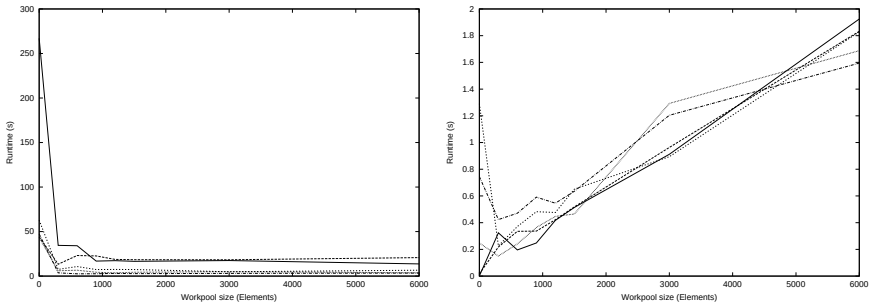

Fig. 1. Influence of the workpool's size on the runtime. Sequential runtime on the left side diagram: 50..250s; on the right side diagram: 0,01..1,2s

Figure 2 shows the relation between the speedup-factor and the number of used processing elements. For smaller numbers of processing elements, the speedup-factor increases almost linear (for one example even a slight superlinear speedup can be recognized). If the number of processing elements gets larger than 8 the speedup differs from being linear and it begins to move to a saturation limit of approximately 16.

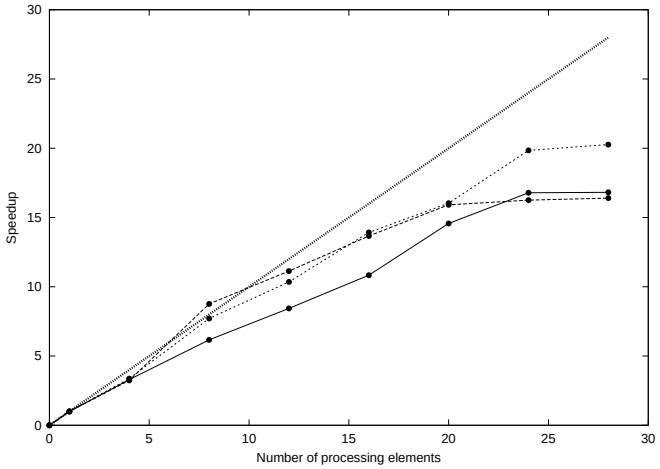

Fig. 2. Influence of the number of processing elements on the speedup-factor

4.3 Analysing scheduling heuristics

The computation of optimal schedules is a rather time-consuming process which is only possible for small to medium-size task graphs. Although most of the

proposed scheduling heuristics aim at large task graphs, this subsection should show, that the efficiency of those heuristics can also be analysed by considering smaller graphs for which the optimal schedule lenghts can be computed by the proposed algorithm.

A test with approximately 1700 computed optimal schedules was carried out to evaluate the heuristic's efficiency. The used graphs were generated randomly in terms of multiple possible settings of the DAGs' properties, e.g. the connection density between the nodes. Using such a wide spread variation of task graph properties, we can be sure that the results are independent of the chosen task graph set. This way, our approach enables scientists to evaluate and compare their heuristics' results more objectively. To demonstrate this new opportunity, we use the well-known heuristic of Kasahara and Narita, described in [6].

Table 1: Deviation of the analysed heuristic's results

Deviation	0%	$< 5\%$	$< 10\%$	$< 15\%$	$< 20\%$	$< 25\%$	$\geq 25\%$
Heuristic [6]	57.67%	71.46%	83.41%	90.96%	94.94%	97.14%	2.86%

Table 1 shows the deviation of this heuristic's results from the optimal schedule lengths. The heuristic finds a solution with the optimal schedule length for 57.67% of the investigated task graphs. Regarding to the other task graphs, the observed deviation from the optimal schedule length is rather low ($< 10\%$) in 83.41% of the cases. Only 2.86% of all solutions are worse than 25%.

The performance of heuristics is usually evaluated by comparing their solutions with the ones of another (well known) heuristic. We argue that it would be more meaningful to use the deviations from the optimal solutions introduced above. For this purpose we will soon release a benchmark suite that provides the optimal solutions for 36.000 randomly created task graph problems which cover a wide range of different graph properties.

5 Conclusion

In this paper we presented a parallel implementation of a Branch–and–Bound algorithm for computing optimal task graph schedules. By means of parallelization the optimization process is accelerated and thus a huge number of test cases can be investigated within a reasonable period of time.

The runtime needed for the computation of an optimal schedule is highly dependent of the workpool's size and the number of processing elements that are available for computation. In order to reduce the runtime, the size of the workpool has to be chosen carefully. A nearly linear speedup can be achieve, provided that an appropriate workpool size is used.

By means of the parallel Branch–and–Bound algorithm, the optimal schedules for a benchmark suite that comprises 1700 task graphs were computed. This allows for a more objective evaluation of scheduling-heuristics than comparisons between heuristics. We evaluated the solutions of the heuristic of Kasahara and Narita [6] by comparing the corresponding schedule lengths towards the optimal schedule lengths of all the 1700 test cases.

The authors' future work will include the release of a test bench, which will provide a collection of 36000 task graph problems together with their optimal schedule lengths. This benchmark suite will enable researchers to compare the performance of their heuristics with the actually best solutions.

6 Acknowledgement

The authors would like to thank Mrs. Sigrid Preuss who contributed some of the presented results from her diploma thesis.

References

1. Aguilar, J., Gelenbe E.: Task Assignment and Transaction Clustering Heuristics for Distributed Systems, Information Sciences, Vol. 97, No. 1& 2, pp. 199–219, 1997
2. 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, June 2003
3. Dogan A., Özgüner F.: Optimal and Suboptimal reliable scheduling of precedence-constrained tasks in heterogeneous distributed computing, International Workshop on Parallel Processing, p. 429, Toronto, August 21-24, 2000
4. Geist A., Beguelin A., Dongarra J., Jiang W., Mancheck R., Sunderam V.: PVM 3 Users Guide and Reference Manual, Oak Ridge National Laboratory, Tennessee 1993
5. Kafil M., Ahmad I.: Optimal Task assignment in heterogeneous distributed computing systems, IEEE Concurrency: Parallel, Distributed and Mobile Computing, pp. 42-51, July 1998
6. Kasahara, H., Narita, S.: Practical Multiprocessor Scheduling Algorithms for Efficient Parallel Processing. IEEE Transactions on Computers, Vol. C-33, No. 11, pp. 1023-1029, Nov. 1984
7. Kohler, W.H., Steiglitz, K.: Enumerative and Iterative Computational Approaches. in: Coffman, E.G. (ed.): Computer and Job-Shop Scheduling Theory. John Wiley & Sons, New York, 1976
8. Kwok, Y.-K., Ahmad, I.: Static scheduling algorithms for allocating directed task graphs to multiprocessors. ACM Computing Surveys, Vol. 31, No. 4, 1999, pp. 406–471
9. Park C.-I., Choe T.Y.: An optimal scheduling algorithm based on task duplication, IEEE Transactions on Computerss, Vol. 51, No. 4, April 2002
10. Radulescu A., van Gemund A. J.C.: Low-Cost Task Scheduling for Distributed-Memory Machines, IEEE Transactions on Parallel and Distributed Systems, Vol. 13, No. 6, June 2002