Search-based Scheduling for Parallel Tasks on Heterogenous Platforms

Robert Dietze✉
Department of Computer Science
Chemnitz University of Technology
Chemnitz, Germany
dirob@cs.tu-chemnitz.de

Gudula Rünger
Department of Computer Science
Chemnitz University of Technology
Chemnitz, Germany
ruenger@cs.tu-chemnitz.de

To cite this version:
Dietze R., Rünger G.: Search-based Scheduling for Parallel Tasks on Heterogenous Platforms. In: Euro-Par 2019: Parallel Processing Workshops, pages 333–344. Springer, 2020. ISBN: 978-3-030-48340-1, The final authenticated version is available online at
`https://doi.org/10.1007/978-3-030-48340-1_26`

Abstract

Scheduling is a widely used method in parallel computing, which assigns tasks to several compute resources of the parallel environments. In this article, we consider parallel tasks as the basic entities to be scheduled onto a heterogeneous execution platform consisting of multicores of different architecture. A parallel task has an internal potential parallelism which allows a parallel execution for example on multicore processors of different type. The assignment of tasks to different multicores of a heterogeneous execution platform may lead to different execution times for the same parallel tasks. Thus, the scheduling of parallel tasks onto a heterogeneous platform is more complex and provides more choices for the assignment and for finding the most efficient schedule. Search-based methods seem to be a promising approach to solve such complex scheduling problems. In this article, we propose a new task scheduling method HP* to solve the problem of scheduling parallel tasks onto heterogeneous platforms. Furthermore, we propose a cost function that reduces the search space of the algorithm. In performance measurements, the scheduling results of HP* are compared to several existing scheduling methods. Performance results with different benchmark tasks are shown to demonstrate the improvements achieved by HP*.

1 Introduction

The execution time of compute-intensive applications depends strongly on the efficient utilization of compute resources. Task-based applications are partitioned into a set of tasks each of which can be assigned to different execution units. Independent tasks can be executed concurrently on the execution units which may lead to a significant reduction of the execution time of the application. For such a reduction of the execution time an efficient utilization of all execution units is needed. A common approach to determine such an assignment of tasks to compute resources is the use of task scheduling methods.

Parallel computing environments within or across institutions are often composed of nodes with different capabilities. Achieving a high efficiency when executing parallel applications on such a heterogeneous system strongly depends on the methods used to schedule the tasks of a parallel application. The heterogeneous compute resources considered in this article consist of several multicore nodes. Each node may have a different architecture which leads to differences in the performance. For the scheduling of parallel tasks two properties of the compute nodes are particularly important, the number of processor cores and the computational speed of each node.

Many proposed task scheduling methods focus on sequential tasks that are assigned to exactly one processor of a compute node. Large applications that are based on parallel programming models may be decomposed into a set of parallel tasks. The term *parallel task* describes a task that can be executed on a single compute node using an arbitrary number of processor cores. Since the tasks are independent from each other, a flexible execution order and a concurrent execution on one compute node are possible. The parallel execution time of each parallel task depends on the number of cores utilized. Thus, for the assignment of parallel tasks to heterogeneous platforms, the particular compute node and the number of processor cores to be used on this node have to be determined for each task. The resulting scheduling problem becomes increasingly complex due to the increasing number of options for placing tasks. Consequently dedicated scheduling methods are required.

Since task scheduling is a NP-complete problem, many of the proposed scheduling methods are based on heuristics[1, 2, 4, 12, 18]. Heuristic scheduling methods may find solutions that are acceptable for a specific use case but finding an optimal solution is not guaranteed. In certain scenarios the optimal solution of a scheduling problem is needed, e.g. for evaluating the quality of heuristic scheduling methods. In the worst case, a search of the entire solution space is required to find such an optimal solution. For the proposed scheduling problem, the search space contains all possible assignments of tasks to compute nodes. Additionally, for each node, all possible combinations for assigning tasks to a number of processor cores have to be considered. Since the computation time required to find an optimal solution can be extremely long, informed search-based algorithms which prune the search space are advantageous. It has been shown that informed search algorithms, such as the A* search algorithm [8], find an optimal solution if an admissible and consistent cost function is used [5].

In this article, we propose a new task scheduling method HP* for assigning parallel tasks to heterogeneous platforms which is based on the A* search algorithm. The goal of HP* is to find an assignment that provides a total execution time that is as low as possible. Furthermore, a cost function is proposed that is able to reduce the solution space searched by HP*. Experiments with programs from the SPLASH-3 benchmark suite [14] used as parallel tasks are performed on a heterogeneous compute cluster and show the competitive behavior of HP*.

The rest of the article is organized as follows: Section 2 defines a scheduling problem for parallel tasks and describes the modeling of the task execution times. Section 3 proposes a the new search-based scheduling algorithm HP* for parallel tasks and the cost function used. Section 4 presents experimental results. Section 5 discusses related work and Section 6 concludes the article.

2 Scheduling of Parallel Tasks on Heterogeneous Platforms

In the following, the considered scheduling problem for the execution of parallel tasks on heterogeneous platforms is described. Furthermore, a cost model for parallel tasks with unknown program structure is presented.

2.1 Scheduling Problem

The scheduling problem considered in this article comprises of n_T parallel tasks T_i, $i = 1, \dots, n_T$ that are independent from each other. A parallel task can be executed on a single compute node utilizing an arbitrary number of processor cores. The number of cores used by each task is fixed during the task execution. The tasks are non-preemptive, i.e. their execution can not be interrupted. For each task $T_i, i \in \{1, \dots, n_T\}$, its parallel execution time using p cores of compute node $N_j, j \in \{1, \dots, n_N\}$ is denoted by $t_{i,j}(p)$.

The considered heterogeneous platform consists of n_N multicore compute nodes $N_1, \dots, N_{n_N}$. The heterogeneity of the platform results from the different architectures of each node. Thus, each compute node N_j, $j \in \{1, \dots, n_N\}$ might have a different computational speed and a different number of processor cores p_j. It is also stated that each processor core can execute only one task at a time. Thus, each parallel task might be executed on 1 to p_j cores of a node N_j, $j \in \{1, \dots, n_N\}$. However, several tasks can be executed on a node at the same time depending on the number of cores utilized on a compute node.

A solution for the scheduling problem described above is an assignment of the tasks T_i, $i = 1, \dots, n_T$ to the compute nodes N_j, $j = 1, \dots, n_N$. For each task $T_i, i \in \{1, \dots, n_T\}$, the resulting schedule S provides the following information:

- the compute node and the number of cores to be utilized,
- the calculated start time s_i and finish time e_i.

The total execution time $T(S)$ of a schedule S is the difference between the earliest start time and latest finish time of all tasks. We assume that the execution of the first task starts at time 0, thus, the total execution time is identical to the latest finish time of all tasks. This can be expressed as $T(S) = \max_{i=1,\ldots,n_T} e_i$. The goal is to determine a schedule S such that the total execution time $T(S)$ is minimized.

2.2 Cost Model for Parallel Tasks

The decisions made by scheduling methods are usually based on predictions of the execution times of single tasks. These predictions can be completely determined by benchmark measurements or can be calculated using a specific cost model. Since the program structures of the parallel tasks are unknown, existing cost models for parallel programming, such as PRAM[7], BSP[16], or LogP[3], can not be used for the considered scheduling problem. Thus, we use the following runtime formula to model the execution time $t_{i,j}$ of each task $T_i, i = 1, \ldots, n_T$ on a compute node $N_j, j = 1, \ldots, n_N$ depending on the number of utilized processor cores p:

$$t_{i,j}(p) = f_j \cdot (a_i/p + b_i + c_i \cdot \log p) \tag{1}$$

The parameter f_j denotes the performance factor of node N_j that describes the computational speed of the compute node N_j. It is defined as the ratio between the sequential execution time of a task on a reference node N_r and the compute node N_j. The remaining part of Eq. (1) represents the execution time of task T_i on the reference node N_r. The structure of this part was chosen to cover the runtime behavior of typical parallel tasks. It consists of a parallel computation time a_i that decreases linearly with the number of cores p, a constant sequential computation time b_i and a parallelization overhead c_i that increases logarithmically with the number of cores p (e.g. for synchronization or communication). To determine the parameters a_i , b_i and c_i of a task T_i, first, the execution times are measured on the reference node with different numbers of cores. Then the concrete values of the parameters are calculated based on a least squares fit of these execution times. Table 1 summarizes the notations used to describe the scheduling problem.

3 Search-based Scheduling Algorithm

In this section, we propose a new task scheduling method HP* for assigning parallel tasks to heterogeneous platforms, which is based on the A* search algorithm. First, a short description of the A* search algorithm is given.

3.1 The A* Search Algorithm

The A* search algorithm is commonly used to find the shortest path in a directed graph with positive edge weights. The goal of the algorithm is to find the

Table 1: Notation of the scheduling problem

Notation	Meaning
n_T	Number of parallel tasks
$T_1, \dots T_{n_T}$	Independent shared memory tasks
n_N	Number of compute nodes in the heterogeneous cluster
$N_1, \dots N_{n_N}$	Compute nodes of the cluster
p_j	Number of processor cores of compute node N_j, $j = 1, ..n_N$
f_j	Performance factor of compute node N_j
$t_{i,j}(p)$	Parallel execution time of task T_i on p cores of node N_j
$T(S)$	Total execution time of schedule S

shortest path in a graph G from a start node s to a nonempty set of goal nodes T. For its search, the algorithm uses a function $f(n)$ representing the cost of a path from s to a goal node via node n. The function $f(n)$ consists of two parts: the actual cost $g(n)$ from s to n and the estimated cost $h(n)$ from n to a goal node. The cost function $f(n) = g(n) + h(n)$ is called admissible if the heuristic function $h(n)$ underestimates the exact cost $h^*(n)$ for each node n, i.e. $h(n) \leq h^*(n)$. For any pair of adjacent nodes x and y with edge weight $d(x, y)$, $f(n)$ is called consistent if the following holds:

$$h(x) \leq d(x, y) + h(y) \qquad (2)$$

In [5], it was shown that using an admissible and consistent function $f(n)$ the A* search algorithm is guaranteed to find an optimal solution.

Algorithm 1 shows the pseudocode of the A* search algorithm presented in [8]. First, the start node s is marked 'open' and the cost function $f(s)$ is evaluated. Then, the 'open' node n with the smallest cost $f(n)$ is selected and marked 'closed'. Each unmarked successor u of n is marked 'open' and $f(u)$ is calculated. Nodes u that are 'closed' are marked 'open' if the current cost $f(u)$ is lower than the cost when they were marked 'closed'. The algorithm continues selecting the next node n with the smallest cost $f(n)$ (line 2) until a goal node is reached.

3.2 Scheduling Parallel Tasks with HP*

For the scheduling of parallel tasks onto heterogeneous platforms, we propose a new scheduling method HP* (Heterogeneous Parallel task scheduling based on A*) based on the A* search algorithm. As in the A* search algorithm, a directed graph with positive edge weights is used as an input for HP*. Therefore, the considered scheduling problem described in Sect. 2.1 is transformed into such a graph where each node n represents a partial schedule S_n. The initial node s is an empty schedule, i.e. where no tasks have been scheduled yet. The successors of a node are created by adding all possible assignments of a task to the respective schedule. The weight $d(n, u)$ of the edge between a node n and its successor u is the difference between the total execution times of the

Algorithm 1: A* search algorithm.

```
Mark $s$ 'open' and calculate $f(s)$
Select the open node $n$ with the smallest value $f(n)$
if ($n \in T$) then Mark $n$ 'closed' and terminate the algorithm
else
  Mark $n$ 'closed'
  for (each successor $u$ of $n$) do
    Calculate $f(u)$
    if ($u$ is not marked 'closed') then Mark $u$ 'open'
    else if (the current value of $f(u)$ is lower as when $u$ was 'closed')
     then
      Mark $u$ 'open'
    end if
  end for
  Proceed with line 2
end if
```

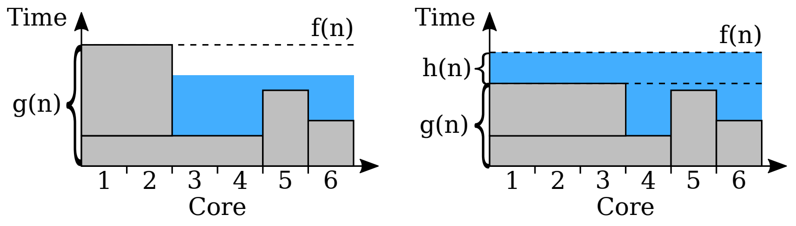

Figure 1: Illustration of the calculation of the cost function $f(n)$ with scheduled tasks (gray) and remaining workload (blue) that is lower (left) or greater (right) than the computational capacity.

corresponding schedules S_n and S_u, i.e. $d(n, u) = T(S_u) - T(S_n)$. Each complete schedule is a goal node in terms of the A* search algorithm. A schedule is called complete if all tasks are assigned to compute nodes.

According to the A* search algorithm the cost function $f(n) = g(n) + h(n)$ consists of two parts:

- $g(n)$ which is the total execution time $T(S_n)$ of the schedule S_n corresponding to node n,
- $h(n)$ which is a heuristic for the total execution time of the remaining tasks.

For the calculation of the function $h(n)$ it is assumed that the remaining tasks can be distributed 'optimally' to all cores. It is also assumed that in a node n the tasks $T_x, ..., T_{n_T} | x \in \{1, ..., n_T\}$ have not been scheduled yet. The remaining sequential workload W_s is then calculated as

$$W_s = \sum_{i=x}^{n_T} t_{i,r}(1) \tag{3}$$

using Eq. (1) considering the reference node N_r. The computational capacity available on all cores of the compute nodes regarding to a schedule S_n is defined as

$$K(S_n) = \sum_{j=1}^{n_N} \sum_{k=1}^{p_j} (T(S_n) - \max_{T_i \in C_{j,k}} e_i). \tag{4}$$

For a compute node N_j, $j = 1, \ldots, n_N$, the set $C_{j,k}$ denotes all tasks that have been assigned to core k of this node. For each node n, the function $h(n)$ can be computed as follows:

$$h(n) = \begin{cases} (W_s - K(S_n))/\sum_{j=1}^{n_N} (p_j \cdot f_j), & \text{if } W_s > K(S_n) \\ 0, & \text{otherwise} \end{cases} \tag{5}$$

If the remaining workload is bigger than the available computational capacity, then $h(n)$ is set to the difference divided by the total compute power, i.e. the sum of $p_j \cdot f_j$ over all nodes N_j, $j = 1, \ldots, n_N$. Otherwise, there is enough computational capacity available for the remaining workload which leads to $h(n) = 0$. Figure 1 shows an illustration of the calculation of the proposed cost function $f(n) = g(n) + h(n)$ with tasks scheduled already (gray) and the remaining workload (blue). In this example, the remaining workload is either lower (left) or greater (right) than the computational capacity.

In Algorithm 2 the pseudocode of the HP* method is shown. HP* maintains two lists, L_{open} and L_{closed}. L_{open} contains all nodes that have been created but not visited yet. The list L_{closed} is used to avoid that nodes are revisited. At the beginning, both lists are empty and the initial node s_{init} represents an empty schedule. $f(s_{init})$ is calculated and the node is added to L_{open}. In each step of the main loop (lines 4–19), the node s_{cur} with the smallest value $f(s_{cur})$ is selected and removed from L_{open}. If s_{cur} represents a complete schedule, the solution is found and the algorithm terminates. If s_{cur} is already part of L_{closed}, s_{cur} is skipped and the algorithm continues with the next node. Otherwise, s_{cur} is added to L_{closed} and a task T is selected that has not been scheduled in s_{cur} yet. For each possible assignment of task T, a new node s is created. This is done by an iteration over all compute nodes N_j, $j = 1, \ldots, n_N$ and all numbers of cores p from 1 to p_j. In each step of this iteration, all possible assignments of task T to p cores of node N_j are generated. Each assignment is added to the schedule used in s_{cur} and the resulting schedule is represented by a new node s. Then value $f(s)$ of this new node s is calculated and s is added to L_{open}.

Algorithm 2: Pseudocode of the HP* method.

```
Let $L_{open}$ and $L_{closed}$ be empty lists
Let $s_{init}$ be a node with an empty schedule
Add $s_{init}$ to $L_{open}$ and calculate $f(s_{init})$
while ($L_{open}$ is not empty) do
  Let $s_{cur}$ be the node in $L_{open}$ with the smallest value $f(s_{cur})$
  Remove $s_{cur}$ from $L_{open}$
  if ($s_{cur}$ is a complete schedule) then Terminate the algorithm
  if ($s_{cur} \notin L_{closed}$) then
    Select an unscheduled task $T$
    Add $s_{cur}$ to $L_{closed}$
    for ($j = 1, \dots, n_N$ and $p = 1, \dots, p_j$) do
      for (each assignment of task $T$ to $p$ cores of compute node $N_j$) do
        Create a new node $s$ as a copy of $s_{cur}$
        Add the assignment to $s$ and calculate $f(s)$
        Add $s$ to $L_{open}$
      end for
    end for
  end if
end while
```

4 Experimental Results with Parallel Tasks on a Heterogeneous Compute Cluster

In the following, we present experimental results of the scheduling method for the execution of parallel tasks on a heterogeneous compute cluster.

4.1 Experimental Setup

The heterogeneous compute cluster used consists of 3 nodes with a total of 16 processor cores. Table 2 lists the properties of these compute nodes. The compute node `sb1` is used as reference node for the determination of the parameters described in Sect. 2.2. The scheduling method described in Sect. 3.2 is implemented in C++ using the gcc compiler with optimization level 2. Additionally, we have implemented three existing heuristic scheduling methods which are suitable for the scheduling of parallel tasks on heterogeneous platforms:

HCPA: The HETEROGENEOUS CRITICAL PATH AND ALLOCATION method [11] transforms a heterogeneous compute cluster with individual computational speeds of the processors into a "virtual" homogeneous cluster with equal speed. Then, an existing method for homogeneous compute clusters (i.e., CPA [13]) is used for the scheduling.

Δ-CTS: The Δ-CRITICAL TASK SET method [17] is an extension of an existing

Table 2: List of nodes of the utilized heterogeneous compute cluster.

Nodes	Processors	number of cores	total RAM	GHz
`skylake1`	Intel i7-6700	4	16 GB	3.40
`hw1`	Intel i7-4770K	4	16 GB	3.50
`sb1`	Intel Xeon E5-2650	8	32 GB	2.00

scheduling method for sequential tasks on heterogeneous compute clusters (i.e., HEFT [19]) to parallel tasks. In each step, the method selects a set of tasks with similar sequential execution time. For each of these particular tasks, the compute node and number of cores are determined separately such that the earliest finish time of the task is minimized. The maximum number of cores to be used by each task depends on the number of selected tasks.

WLS: The Water-Level-Search method [6] combines list scheduling with a search based approach. The method uses a limit for the predicted total execution time that must not be exceeded by the finish time of any task. First, a list scheduling approach is applied several times while the limit is increased until all tasks are scheduled. All computed finish times of all tasks are collected in a set of limits. Then a binary search on this list is performed to find the smallest limit where all tasks can be scheduled.

A separate front-end node of the compute cluster is responsible for the scheduling and for starting the task execution using SSH connections to the compute nodes. The tasks are executed according to the determined schedule and the total execution time is measured. The measurements are performed five times and the average result is shown.

As parallel tasks two application tasks and two kernel tasks from the SPLASH-3 benchmark suite [14] are used. Unless otherwise stated, the default parameters or the provided "parsec-simlarge" parameter sets are used for the different benchmark tasks. The following application and kernel tasks were selected:

- **BARNES (application):** Simulation of a particle system using the Barnes-Hut algorithm. The number of particles is set to 2^{18}.
- **FMM (application):** Simulation of a particle system using the Fast Multipole Method. The number of particles is set to 2^{19}.
- **CHOLESKY (kernel):** Cholesky factorization of a sparse matrix. The input matrix "tk29.O" of size $13\,992 \times 13\,992$ is used.
- **LU (kernel):** LU factorization of a dense matrix. The size of the input matrix is set to 4096×4096.

4.2 Performance Results with Benchmark Tasks

In the following, the search-based scheduling method (HP*) proposed in Sect. 3.2 and the scheduling methods (HCPA, Δ-CTS, WLS) described in the previous subsection are investigated in several measurements. These methods are used to determine schedules for the execution of the SPLASH-3 benchmark tasks on a heterogeneous cluster. The heterogeneous cluster used for the following measurements consists of all compute nodes listed in Table 2.

Fig. 2 (top) shows the measured total execution times of the BARNES application tasks (left) and FMM application tasks (right) of the SPLASH-3 benchmark depending on the number of tasks. For both types of application tasks the measured times using the HP* method are lower or equal than the results of the three heuristic scheduling methods (HCPA, Δ-CTS, WLS). The results of WLS and HP* show a more steady increase compared to HCPA and Δ-CTS. Especially for Δ-CTS, a strong increase of the execution times for 7 and 13 tasks can be observed. This behavior might be caused by the heuristics used by Δ-CTS. Using HP* leads to slightly lower or equal measured execution times compared to WLS, except for 7 and 8 tasks where HP* achieves up to 23% lower execution times.

Fig. 2 (bottom) shows the measured total execution times of CHOLESKY kernel tasks (left) and LU kernel tasks (right) depending on the number of tasks using all compute nodes of Table 2. For the CHOLESKY tasks, the execution times using HCPA are up to 87% higher than the best results. A reason for these significant differences might be that HCPA favors a parallel task execution that uses many cores for each task. However, the execution times of the CHOLESKY tasks are too small to achieve a proper reduction of the parallel execution time for increasing numbers of cores. The other methods achieved very similar results, except for task numbers between 3 and 7 where Δ-CTS leads to execution times that are up to 42% higher. For the LU tasks, the differences between the results of the methods used are smaller. The execution times using HCPA and Δ-CTS are slightly higher than for WLS and HP* with large increases for 7 and 13 tasks. As for the application tasks, the execution times for 7 and 8 tasks are up to 11% lower using HP* compared to WLS. All in all, HP* leads to lower or equal execution times with a more steady increase compared to the other methods.

5 Related work

Search-based approaches have been applied to many task scheduling problems. A comparison of search-based and heuristic approaches for scheduling independent tasks onto heterogeneous systems can be found in [2]. An experimental comparison of several scheduling algorithms including A*, genetic algorithms, simulated annealing, tabu search as well as popular list scheduling heuristics is given in [9]. The work considers the problem of mapping sequential tasks with dependencies onto a homogeneous cluster. A few algorithms for solving the task

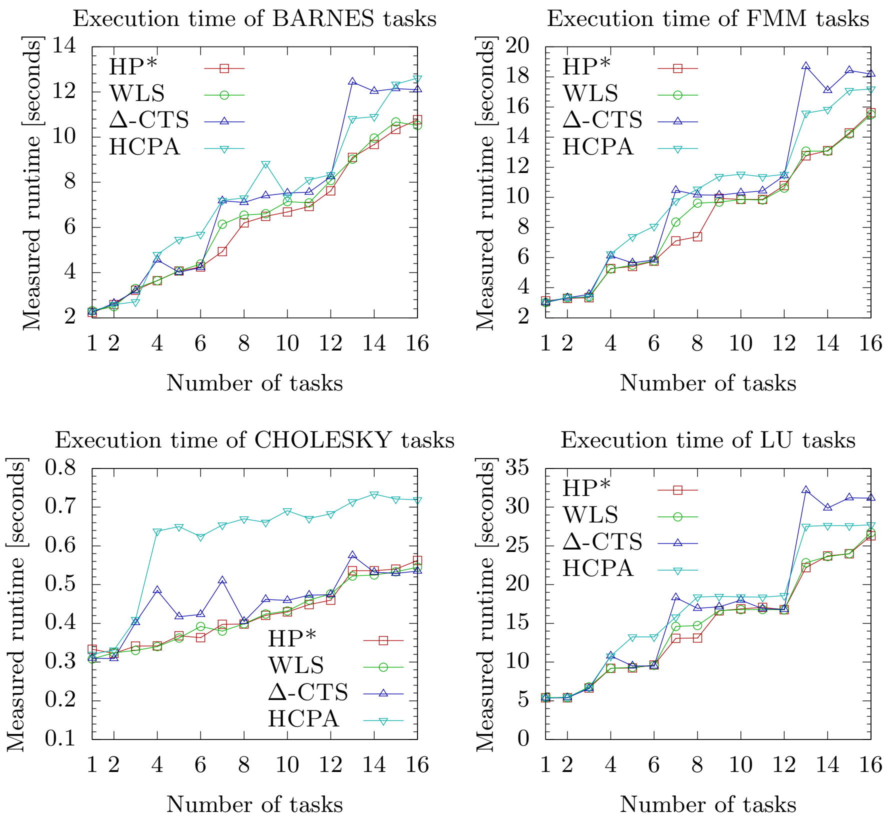

Figure 2: Top: Measured total execution times of BARNES application tasks (left) and FMM application tasks (right) depending on the number of tasks using all compute nodes of Table 2. Bottom: Measured total execution times of CHOLESKY kernel tasks (left) and LU kernel tasks (right) depending on the number of tasks using all compute nodes of Table 2.

scheduling problem based on the A* search algorithm have been reported in the literature. Kwok and Ahmad [10] proposed a scheduling algorithm for the assignment of sequential tasks to homogeneous platforms based on the A* search algorithm. A number of pruning techniques to reduce the search space as well as a parallelization of the algorithm are presented. Sinnen [15] proposed a scheduling algorithm based on the A* search algorithm using an improved cost function along with several pruning techniques to reduce the search space. In contrast to these works, we consider the scheduling of parallel tasks to heterogeneous platforms.

6 Conclusion

In this article, we have proposed a task scheduling method HP* for assigning parallel tasks to heterogeneous platforms, which is based on the A* search algorithm. In addition, a cost function has been proposed that is able to reduce the search space of our algorithm. Measurements with benchmark tasks have been performed and the scheduling results of HP* have been compared to several existing scheduling methods. Our performance results demonstrate that the use of HP* can lead to a reduction of the total execution times of the resulting schedules in comparison with known algorithms.

Acknowledgments

This work was supported by the German Ministry of Science and Education (BMBF) project "SeASiTe", Grant No. 01IH16012A/B and the German Research Foundation (DFG), Federal Cluster of Excellence EXC 1075 "MERGE".

References

[1] Arabnejad, H., Barbosa, J.G.: List scheduling algorithm for heterogeneous systems by an optimistic cost table. IEEE Transactions on Parallel and Distributed Systems **25**(3), 682–694 (March 2014)

[2] Braun, T.D., Siegel, H.J., Beck, N., Bölöni, L.L., Maheswaran, M., Reuther, A.I., Robertson, J.P., Theys, M.D., Yao, B., Hensgen, D., Freund, R.F.: A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. Journal of Parallel and Distributed Computing **61**(6), 810 – 837 (2001)

[3] Culler, D., Karp, R., Patterson, D., Sahay, A., Schauser, K., Santos, E., Subramonian, R., von Eicken, T.: LogP: Towards a realistic model of parallel computation. In: Proc. of the 4th ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming (PPOPP'93). pp. 1–12. ACM (1993)

[4] Daoud, M.I., Kharma, N.: A high performance algorithm for static task scheduling in heterogeneous distributed computing systems. Journal of Parallel and Distributed Computing **68**(4), 399 – 409 (2008)

[5] Dechter, R., Pearl, J.: Generalized best-first search strategies and the optimality of a*. J. ACM **32**(3), 505–536 (Jul 1985)

[6] Dietze, R., Hofmann, M., Rünger, G.: Water-level scheduling for parallel tasks in compute-intensive application components. J. of Supercomputing pp. 1–22 (2016)

[7] Fortune, S., Wyllie, J.: Parallelism in random access machines. In: Proc. of the 10th Annual ACM Symp. on Theory of Computing. pp. 114–118. ACM (1978)

[8] Hart, P.E., Nilsson, N.J., Raphael, B.: A formal basis for the heuristic determination of minimum cost paths. IEEE Transactions on Systems Science and Cybernetics **4**(2), 100–107 (July 1968)

[9] Jin, S., Schiavone, G., Turgut, D.: A performance study of multiprocessor task scheduling algorithms. J. Supercomput. **43**(1), 77–97 (Jan 2008)

[10] Kwok, Y.K., Ahmad, I.: On multiprocessor task scheduling using efficient state space search approaches. J. Parallel Distrib. Comput. **65**(12), 1515–1532 (Dec 2005)

[11] N'Takpé, T., Suter, F.: Critical path and area based scheduling of parallel task graphs on heterogeneous platforms. In: Proceedings of the 12th International Conference on Parallel and Distributed Systems (ICPADS 2006). vol. 1, pp. 3–10. IEEE (2006)

[12] Radulescu, A., van Gemund, A.J.C.: Low-cost task scheduling for distributed-memory machines. IEEE Transactions on Parallel and Distributed Systems **13**(6), 648–658 (June 2002)

[13] Radulescu, A., Van Gemund, A.: A low-cost approach towards mixed task and data parallel scheduling. In: Proceedings of the International Conference on Parallel Processing. pp. 69–76. IEEE (2001)

[14] Sakalis, C., Leonardsson, C., Kaxiras, S., Ros, A.: Splash-3: A properly synchronized benchmark suite for contemporary research. In: Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS 2016). pp. 101–111. IEEE (2016)

[15] Sinnen, O.: Reducing the solution space of optimal task scheduling. Computers & Operations Research **43**, 201 – 214 (2014)

[16] Skillicorn, D.B., Hill, J., McColl, W.: Questions and answers about bsp. Scientific Programming **6**(3), 249–274 (1997)

[17] Suter, F.: Scheduling δ-critical tasks in mixed-parallel applications on a national grid. In: Proc. of the 8th IEEE/ACM Int. Conf. on Grid Computing. pp. 2–9. IEEE (2007)

[18] Topcuoglu, H., and, S.H.: Performance-effective and low-complexity task scheduling for heterogeneous computing. IEEE Transactions on Parallel and Distributed Systems **13**(3), 260–274 (March 2002)

[19] Topcuoglu, H., Hariri, S., Wu, M.Y.: Task scheduling algorithms for heterogeneous processors. In: Proc. of the 8th Heterogeneous Computing Workshop (HCW'99). pp. 3–14. IEEE (1999)