List Based Task Scheduling Algorithms on Heterogeneous Systems - An overview

Hamid Arabnejad

Universidade do Porto,
Faculdade de Engenharia, Dep. de Engenharia Informática,
hamid.arabnejad@fe.up.pt

Abstract. Task scheduling is key issue in obtaining high performance in heterogeneous systems. Task scheduling in heterogeneous systems is a NP-problem, therefore several heuristic approaches were proposed to solve it. These heuristics are categorized into several classes, such as list based, clustering and task duplication scheduling. Here I consider the list scheduling approach. In this paper, I will have an overview on six well-known list based scheduling algorithms (HEFT, CPOP, HCPT, HPS, PETS and lookahead) and compare the results of them.

Keywords: task scheduling , static scheduling , heterogeneous system

1 Introduction

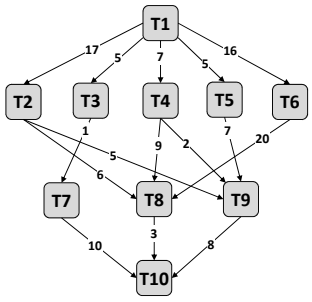
A heterogeneous system can be defined as a range of different system resources, which can be local or geographically distributed, utilized to executing computationally intensive application. The efficiency of executing parallel applications on heterogeneous systems critically depends on the methods used to schedule the tasks of a parallel application. The objective is to minimize the overall completion time or *makespan*. The task scheduling problem for heterogeneous systems is more complicated than that in the homogeneous computing systems, because of the different execution rates among processors and possibly different communication rates between different processors. The DAG scheduling problem has been shown to be NP-complete [2, 3, 9], even for the homogeneous case, therefore the research effort in this field has been mainly to obtain low complexity heuristics that produce good schedules. The task scheduling problem is broadly classified in two major categories, namely Static Scheduling and Dynamic Scheduling. In Static category, all information about tasks such as execution and communication time for each task and its relation with other tasks are known before hand; in Dynamic category, such information is not available and decisions are made in runtime. In another way, Static scheduling is compile-time scheduling and Dynamic scheduling is run-time scheduling. Static scheduling algorithms are universally classified into two major groups, namely, Heuristic-based and Guided Random Search-based algorithms. Heuristic-based algorithms give near-optimal solutions but with polynomial time complexity and acceptable performance in

comparison with Guided Random Search-based algorithms which give optimal solutions with exponential time complexity. The Heuristic-based group is composed by three subcategories that are: list, clustering and duplication scheduling. Clustering heuristics were mainly proposed for homogeneous systems and the aim is to form clusters of tasks that are then assigned to processors. The duplication heuristics produce the shortest *makespans* but they have two disadvantages: one is the higher time complexity, such as cubic in relation to the number of tasks; and second, they have lower efficiency because the main strategy is to duplicate the execution of tasks, resulting in more processor power used. Efficiency is an important characteristic, not only due to the energetic cost but also, in a shared resource, less efficiency means less processors available to run other concurrent applications. List scheduling heuristics, on the other hand, produce the most efficient schedules, without compromising the *makespan* and with a complexity that is, in general, quadratic in relation to the number of tasks. In this paper, I present an overview on list based scheduling algorithm for a bounded number of fully connected heterogeneous processors.

This paper is organized as follows: in Section 2, I introduce the DAG scheduling ; in Section 3, I present an overview on list based scheduling algorithms on heterogeneous systems; in Section 4, I present results of comparison for these algorithms based on several measure parameter and, finally, conclusions in Section 5.

2 DAG Scheduling

The problem addressed in this paper is the static scheduling of a single application on a heterogeneous system. An application can be represented by a *Directed Acyclic Graph* (DAG), $G = (V, E, P, W)$, as shown in Figure 1.



Task	P1	P2	P3
T1	23	32	28
T2	36	36	16
T3	2	3	3
T4	16	17	13
T5	25	17	28
T6	25	38	37
T7	30	21	29
T8	17	26	36
T9	16	10	21
T10	26	16	26

Fig. 1: Application model and computation time matrix of the tasks in each processor

Where V is the set of v nodes, and each node $v_i \in V$ represents an application task, which includes its instruction that must be executed on the same machine. E is the set of e communication edges between tasks, each $e(i, j) \in E$ represents

the task-dependency constraint such that task n_i should complete its execution before task n_j can be started. P is the set of p heterogeneous processors available in the system. W is a $v \times p$ computation cost matrix, where v is the number of tasks and p is the number of processors in the system. $w_{i,j}$ gives the estimate time to execute task v_i on machine p_j. The mean execution time of task n_i can be calculated by $\overline{w_i} = (\sum_{j \in P} w_{i,j})/p$. Each edge $e(i,j) \in E$ is associated with a non-negative weight $c_{i,j}$ representing the communication cost between the tasks n_i and n_j. Note that, when task i and j are assigned to the same processor, the real communication cost is considered to be zero because it is negligible compared to interprocessor communication costs. Additionally, in our model, I consider that processors are connected in a fully connected topology. The execution of tasks and communications with other processors can be done for each processor simultaneously and without contention. Also, the execution of any task is considered nonpreemptive. These model simplifications are common in list scheduling problem [4, 8], and I consider them in order to have a fair comparison to the state of the art algorithms.
Next, I present some of the common attributes used in task scheduling, that I will refer in the following sections.

- **pred**($\mathbf{n_i}$) : denotes the set of immediate predecessors of task n_i in a given DAG. A task with no predecessors is called an *entry* task, n_{entry}. If a DAG has multiple entry tasks, a dummy entry task with zero weight and zero communication edges is added to the graph.
- **succ**($\mathbf{n_i}$) : denotes the set of immediate successors of task n_i. A task with no successors is called an *exit* task, n_{exit}. Like the entry task, if a DAG has multiple exit tasks, a dummy exit task with zero weight and zero communication edges from current multiple exit tasks to this dummy node is added.
- **makespan** : it is the finish time of the exit task in the scheduled DAG, and is defined by $makespan = AFT(n_{exit})$ where $AFT(n_{exit})$ denotes the Actual Finish Time of the exit task.
- **Critical Path**(**CP**) : the CP of a DAG is the longest path from n_{entry} to n_{exit} in the graph. The length of this path $|CP|$ is the sum of the computation costs of the tasks and intertask communication costs along the path. The $|CP|$ value of a DAG is the lower bound of the makespan.
- **EST**($\mathbf{n_i}, \mathbf{p_j}$) : denotes the *Earliest Start Time* of a node n_i on a processor p_j
- **EFT**($\mathbf{n_i}, \mathbf{p_j}$) : denotes the *Earliest Finish Time* of a node n_i on a processor p_j

The *objective function* of the scheduling problem is to determine an assignment of tasks of a given DAG to processors such that the *Schedule Length* is minimized. After all nodes in the DAG are scheduled, the schedule length will be the *Actual Finish Time* of the exit task.

3 List-Based Scheduling Algorithms

In this section, I present a brief survey of task scheduling algorithms, specifically list based heuristics. In the past years, the research on static DAG scheduling has focused on finding suboptimal solutions to obtain a good solution in an acceptably short time. List scheduling heuristics usually generate good quality schedules at a reasonable cost. In comparison with clustering algorithms, they have lower time complexity and in comparison to task duplication strategies, their solutions use less processors, generating more efficient schedules.
A large number of list scheduling algorithms have been developed by researchers in the past. This type of scheduling algorithms has three phases: the prioritizing phase for giving a priority to each task; the selection phase for select the task based on its priority from the ready tasks in current time; and a processor selection phase for selecting a suitable processor that minimizes the heuristic cost function. If two or more tasks have equal priority, then the tie is resolved by selecting a task randomly. The two last phases are repeated until all tasks are scheduled to suitable processors.

Here, I describe the list-based scheduling heuristic algorithms, for scheduling tasks on a bounded number of heterogeneous processors, selected to be compared, namely, CPOP, HEFT, HCPT , HPS, PETS and lookahead.

3.1 CPOP Algorithm

The CPOP (Critical Path On a Processor) algorithm proposed in [8], like other list-based scheduling algorithm, has three phases, namely, *task prioritizing*, *task selection* and *processor selection* phase. In first phase, task prioritizing, the CPOP algorithm used the upward rank ($rank_u$) and downward rank ($rank_d$) to give a priority to each task in DAG. The upward rank ($rank_u$) represents the length of the longest path from the task to exit task, including the computational cost of the task and is given by $rank_u(n_i) = \overline{w_i} + \max_{n_j \in succ(n_i)}\{\overline{c_{i,j}} + rank_u(n_j)\}$ for exit task $rank_u(n_{exit}) = \overline{w_{exit}}$, and the downward rank ($rank_d$) represents the length of the longest path from a start task to the task and is given by $rank_d(n_i) = \max_{n_j \in pred(n_i)}\{rank_d(n_j) + \overline{w_j} + \overline{c_{j,i}}\}$ for entry task $rank_d(n_{entry}) = 0$

The CPOP algorithm, after calculating the upward and downward rank value for all tasks in the DAG, the priority of each task is equal to $rank_d + rank_u$. In CPOP algorithm, tasks are categorized into critical path tasks and non-critical path tasks. The CPOP algorithm defined a $CPprocessor$ as the processor that minimizes the overall execution time of the critical path assuming all the critical path nodes are mapped onto it. In task selection phase, the CPOP algorithm select the task with highest priority from ready task. In the next phase, processor selection phase, if the selected task is a CP task, it is mapped to $CPprocessor$, otherwise it should be mapped to the processor that minimize its earliest finish time.

3.2 HEFT Algorithm

The HEFT (Heterogeneous Earliest Finish Time) algorithm [8] is highly competitive in that it generates a comparable schedule length to other scheduling algorithms, with a low time complexity. Like most of list-based scheduling algorithm it hast three phases. In *task prioritizing* phase, it used $rank_u$ (described before in CPOP algorithm) to assign priority to the task. In *task selection* phase, the HEFT algorithm select the task with highest priority from the ready list as the selected task. And in the *processor selection* phase, the HEFT select the processor that allows the EFT (Earliest Finish Time) of the selected task. However, the HEFT algorithm uses an insertion policy that tries to insert a task in an earliest idle time between two already scheduled tasks on a processor, if the slot is large enough to accommodate the task.

3.3 HCPT Algorithm

The HCPT (Heterogeneous Critical Parent Trees) algorithm [4] uses a new mechanism to construct the scheduling list L, instead of assigning priorities to the application tasks. HCPT divides the task graph into a set of unlisted-parent trees. The root of each unlisted-parent tree is a critical path node (CN). A CN is defined as the node that has zero difference between its $AEST$ and $ALST$. The $AEST$ is the *Average Earliest Start Time* of the task and it is equivalent to $rank_d$ as shown by $AEST(n_i) = \max_{n_j \in pred(n_i)}\{AEST(n_j) + \overline{w_j} + \overline{c_{j,i}}\}$ and for entry task $AEST(n_{entry}) = 0$. The *Average Latest Start Time* ($ALST$) of the task can be computed recursively by traversing the DAG upward, starting from the exit task and given by $ALST(n_i) = \min_{n_j \in succ(n_i)} \left\{ALST(n_j) - \overline{c_{i,j}}\right\} - \overline{w_i}$ and for exit task $ALST(n_{exit}) =$.
The algorithm has also two phases, namely *listing tasks* and *processor assignment*. In the first phase, the algorithm starts with an empty queue L and an auxiliary stack S that contains the CNs pushed in decreasing order of their ALSTs, i.e. the entry node is on top of S. Consequently, $top(S)$ is examined. If $top(S)$ has an unlisted parent (i.e. has a parent not in L), then this parent is pushed on the stack S. Otherwise, $top(S)$ is popped and enqueued into L. In the processor assignment phase, the algorithm tries to assign each task $n_i \in L$ to a processor p_j that allows the task to finish its execution as earlier as possible.

3.4 HPS Algorithm

The HPS (High Performance task Scheduling) [6] algorithm has three phases, namely, *level sorting*, *task prioritization* and *processor selection* phase. In the level sorting phase, the given DAG is traversed in a top-down fashion to sort tasks at each level in order to group the tasks that are independent of each other. As a result, tasks in the same level can be executed in parallel. In the task prioritization phase, priority is computed and assigned to each task using the attributes *Down Link Cost* (DLC), *Up Link Cost* (ULC) and *Link Cost* (LC) of the task. The DLC of a task is the maximum communication cost among all the

immediate predecessors of the task. The DLC for all tasks at level 0 is 0. The ULC of a task is the maximum communication cost among all the immediate successors of the task. The ULC for an exit task is 0. The LC of a task is the sum of DLC, ULC and maximum LC of all its immediate predecessor tasks.

At each level, based on LC values, the task with highest LC value receives the highest priority followed by the task with next highest LC value and so on in the same level. In the processor selection phase, the processor that gives the minimum *EFT* for a task is selected for executing that task. It has an insertion-based policy, which considers the insertion of a task in an earliest idle time slot between two already scheduled tasks on a processor.

3.5 PETS Algorithm

The PETS (Performance Effective Task Scheduling) algorithm [5] has the same three phases as HPS. In the level sorting phase, like HPS, tasks are categorized in levels so that in each level the tasks are independent. In the task prioritization phase, priority is computed and assigned to each task using the attributes *Average Computation Cost* (ACC), *Data Transfer Cost* (DTC) and the *Rank of Predecessor Task* (RPT). The ACC of a task is the average computation cost on all the p processors. The DTC of a task n_i is the amount of communication costs incurred to transfer the data from task n_i to all its immediate successor tasks; for an exit node $DTC(n_{exit}) = 0$. The RPT of a task n_i is the highest rank of all its immediate predecessor tasks; for an entry node $RPT(n_{entry}) = 0$. The rank is computed for each task n_i based on its ACC, DTC and RPT values and is given by $rank(n_i) = round\{ACC(n_i) + DTC(n_i) + RPT(n_i)\}$.

At each level, the task with highest rank value receives the highest priority followed by the task with next highest rank value and so on. A tie is broken by selecting the task with a lower ACC value. As some of the other task scheduling algorithms, in the processor selection phase, it selects the processor that gives the minimum *EFT* value for executing the task. It also uses a insertion-based policy for scheduling a task in an idle slot between two previously scheduled tasks on a given processor.

3.6 Lookahead Algorithm

The Lookahead scheduling Algorithm [1] uses a new methodology to select the suitable processor for selected task in each step of scheduling. In Lookahead algorithm, first, calculate the upward rank for all tasks in a given DAG as same as HEFT. But in Processor selection phase, unlike the HEFT algorithm that select the processor based on earliest finish time for current task, the Lookahead algorithm for test each processor, first assign the selected task to the processor and then schedule the selected task children and save the maximum EFT of children as EFT selected task on the processor. After test all processor for assigning selected task to them, the lookahead select the processor with minimum EFT based on its children.

4 Experimental Result and Discussion

This section presents performance comparison of the DMCP algorithm with the algorithms presents above. For this purpose, I consider randomly generated application graphs. I first present the comparison metrics used for the performance evaluation.

4.1 Comparison Metrics

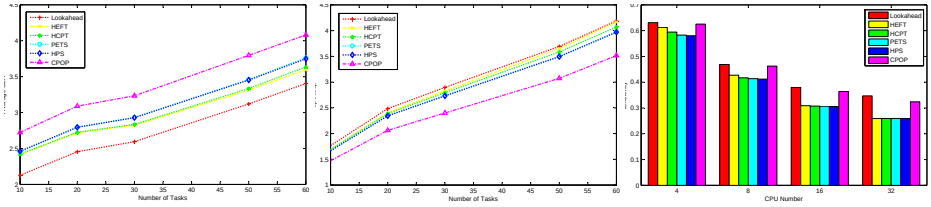
The comparison metrics are Scheduling Length Ratio, Speedup, Efficiency.

$$SLR = \frac{makespane(solution)}{\sum_{n_i \in CP_{MIN}} \min_{p_j \in P} (w_{(i,j)})} \qquad Speedup = \frac{\min\limits_{p_j \in P} \left[\sum\limits_{n_i \in V} w_{(i,j)} \right]}{makespane(solution)}$$

The denominator in SLR is the minimum computation of tasks on critical path. With any algorithm, there is no makespane less than the denominator of SLR equation. Therefore, the algorithm with lower SLR is the best algorithm. Average SLR values over several task graphs are used in our results. In *Speedup*, the sequential time is obtained by the sum of the processing time on the processor that minimizes the total computation cost [8].
For the general case, Efficiency is defined as the Speedup divided by the number of processors used $Efficiency = Speedup/\{Number\ of\ processors\ used\}$.
The DAGs used in this simulation setup were randomly generated using the program in [7] which consider the following parameters: *width* as the number of tasks on the largest level; *regularity* is the uniformity of the number of tasks in each level; *density* is the number of edges between two levels of the DAG. These parameters may vary between 0 and 1. An additional parameter, *jump*, indicates that an edge can go from level l to level $l + jump$. In this paper, I used this synthetic DAG generator for making the DAG structure which includes the specific number of nodes and their dependencies. To obtain computation and communication costs, two extra parameters are used: *CCR* and *beta*. The first parameter, CCR(Communication to Computation Ratio) is ratio of the sum of the edge weights to the sum of the node weights in a DAG; and beta(Range percentage of computation costs on processors) is the *heterogeneity factor* for processors speed. A higher value for β implies higher heterogeneity and very different computation costs among processors and a low value implies that the computation costs for a given task is almost equal among processors [8]. The average computation cost of a task n_i in a given graph $\overline{w_i}$ is selected randomly from a uniform distribution with range $\left[0, 2 \times \overline{w_{DAG}}\right]$, where $\overline{w_{DAG}}$ is the average computation cost of the given graph. The computation cost of each task n_i on each processor p_j is randomly set from the range of $\overline{w_i} \times \left(1 - \frac{\beta}{2}\right) \leq w_{i,j} \leq \overline{w_i} \times \left(1 + \frac{\beta}{2}\right)$.
In this paper, I consider DAGs with 10, 20, 30, 40, 50 and 60 tasks; the number of processors equal to 4, 8, 16 and 32; CCR of 0.1, 0.5, 0.8, 1, 2, 5 and 10; width equal to 0.1, 0.4, 0.8; regularity equal to 0.2,0.8 ; density equal to 0.2, 0.8; Beta equal to 0.1, 0.2, 0.5, 1 and 2 ; and jumps of 1, 2, and 4. These combinations

give 30,240 different DAG types. Since 10 random DAGs were generate for each combination, the total number of DAGs used in our experiment was 302,400.

Figure 2 shows the results of SLR, Speedup and Efficiency produced by the Lookahead, HEFT, HCPT and CPOP algorithm. We can see that Lookahead has the lower SLR for all DAG sizes. Consequently, Lookahead achieved better Speedups. The second best algorithm in terms of SLR is HEFT, as was referred before to be the state of art algorithm so far. Attending to Efficiency, Lookahead is also the best one. CPOP that follows very closely Lookahead in terms of Efficiency, has the poor SLR.



(a) (b) (c)

Fig. 2: (a) Average SLR; (b) average SpeedUP and (c) Efficiency comparison for Random Graphs

If I want to discuss about the result with respect to hardware feature, I should compare results for different values for CCR and beta (heterogeneity factor).

Figure 3 shows the SLR values for each algorithm with respect to different values of CCR. As shown, in lower CCR values, the improvement is not significant and HCPT, HEFT and Lookahead algorithm have close SLR values but in higher CCR, Lookahead algorithm shows better SLR than the other algorithms.

Also, Figure 4 show the same condition in improvement like as CCR for beta (heterogeneity factor); In lower beta, all algorithms have very closely same SLR values ; But with increasing beta parameter, as shown, Lookahead shows better improvement in terms of SLR and then HEFT and HCPT.

5 Conclusion

In this paper, we had an overview on the most well-know list-based scheduling algorithms on heterogeneous systems. My result shows, among of all these algorithms, Lookahead shows better performance in terms of SLR, SpeedUp and efficiency in overall But if we want review them based on hardware feature, as shown in Figure 3 for lower CCR all algorithms have same performance in term of SLR and also in Figure 4, as like CCR factor, shown for lower heterogeneity factor, all algorithms have same SLR values.

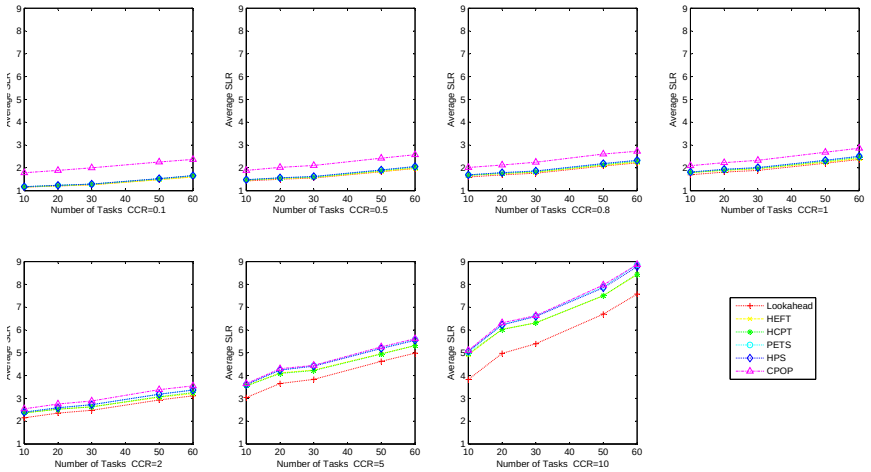


Fig. 3: Average SLR with respect to different CCR values

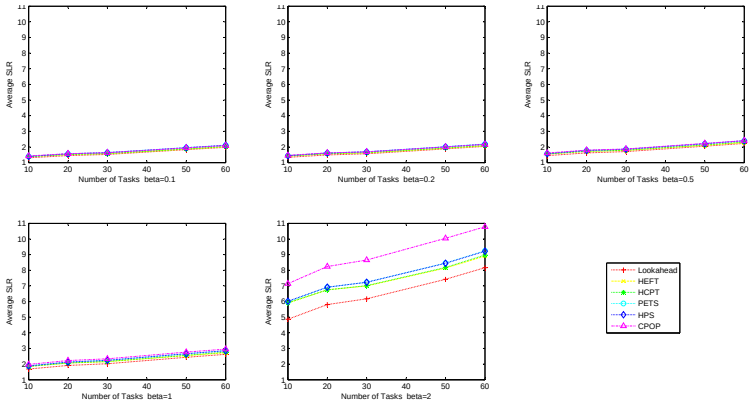


Fig. 4: Average SLR with respect to different beta (heterogeneity factor values

References

1. Luiz F. Bittencourt, Rizos Sakellariou, and Edmundo R. M. Madeira. DAG scheduling using a lookahead variant of the heterogeneous earliest finish time algorithm. In *PDP*, pages 27–34. IEEE Computer Society, 2010.
2. E. G. Coffman, editor. *Computer and Job-Shop Scheduling Theory*. Wiley, 1976.
3. M. R. Garey and D. S. Johnson. *Computers and intractability; a guide to the theory of NP-completeness*. W.H. Freeman, 1979.
4. Tarek Hagras and Jan Janecek. A simple scheduling heuristic for heterogeneous computing environments. In *ISPDC*, pages 104–110. IEEE Computer Society, 2003.
5. E. Ilavarasan and P. Thambidurai. Low complexity performance effective task scheduling algorithm for heterogeneous computing environments. *Journal of Computer Sciences*, 3(2):94–103, 2007.

6. E. Ilavarasan, P. Thambidurai, and R. Mahilmannan. High performance task scheduling algorithm for heterogeneous computing system. volume 3719 of *Lecture Notes in Computer Science*, pages 193–203. Springer, 2005.
7. DAG Generation Program. http://www.loria.fr/~suter/dags.html, 2010.
8. Haluk Topcuoglu, Salim Hariri, and Min-You Wu. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, PDS-13(3):260–274, March 2002.
9. J.D. Ullman. Np-complete scheduling problems. *Journal of Computer and System Sciences*, 10(3):384–393, 1975.