Controlled duplication for scheduling real-time precedence tasks on heterogeneous multiprocessors

Jagpreet Singh* and Nitin Auluck
Department of Computer Science & Engineering
Indian Institute of Technology, Ropar
Rupnagar, India, 140001
Email: {jagpreets, nitin}@iitrpr.ac.in

***Abstract*—Duplication based heuristics have been used for scheduling precedence constrained tasks with significant communication in them. Duplicating heavily communicating subtasks of a task on to the same processor improves the schedulability as a larger number of tasks meet their deadlines. However, this reduction comes at the cost of extra computing power required for duplicating subtasks. In this paper, we propose a novel real-time controlled duplication based heuristic called RTCDA for scheduling such tasks on heterogeneous multiprocessors. We observe that duplication is not always required. The decision whether to duplicate or not is decided by the deadlines of the tasks. If a task can meet its deadline without duplication, then it creates more schedule holes and vice versa. RTCDA can utilize these schedule holes to improve the success ratio. Simulation results show that the proposed algorithm gives a better performance than other similar algorithms.**

I. Introduction

Heterogeneous distributed real-time systems (HDRTS) have gained popularity as they allow applications with strict timing constraints to run on high performance and low cost hardware of varying capabilities. The timing constraints in HDRTS's are fulfilled by employing an efficient real-time scheduler. The scheduling algorithm allocates and schedule jobs to ensure that all the tasks meet their deadlines and the overall *schedulability* improves.

*PhD Research Scholar, Student Author.

The problem on real and non-real time systems differs on the basis of the objective of the algorithm. Non real-time algorithms aim to minimize the maximum schedule length, or *makespan* [1] whereas real-time algorithms aim to increase the schedulability.

Some non-real time scheduling approaches have been extended to work on real-time systems [1], [2]. Duplication has proved as a vital heuristic for minimizing the makespan [3]. By duplicating the communicating subtasks on a single processor, the interprocessor communication costs can be minimized, which reduces the makespan. Duplication is a well researched heuristic for non real-time scheduling of a single task graph on heterogeneous multiprocessors [1], [4], [5].

Employing duplication for improving the schedulability has received some attention by researchers. Ranaweera et al. [6] used duplication for enhancing the schedulability of periodic time critical applications for pipelined execution on heterogeneous systems. Auluck et al. [2], [7] proposed algorithms that are extensions of the original duplication strategy proposed in [1]. The algorithm proposed in [2], called RT-DBA, (real-time duplication based algorithm) comes closest to this work. RT-DBA has a few shortcomings that we have tried to address. First of all, it performs duplication for all the tasks, which may not always be required as our motive is not to minimize the makespan, but to meet the deadlines. A late deadline can be met without duplication.

Excessive duplication can reduce possible schedule holes created due to precedence delays, which can be utilized by the other tasks. Secondly, RT-DBA does not consider the current processor-scheduling load. Finally, it does not utilize schedule holes for scheduling or duplication. We observe that doing so can result in a better utilization of computing power.

This paper proposes a novel static real-time *Controlled Duplication* algorithm (RTCDA) for scheduling of periodic independent precedence-constrained tasks. It uses schedule holes for scheduling and duplication. The precedence-constrained tasks are represented by a *directed acyclic graph* (DAG) [2]. To the best of the authors knowledge, the concept of controlled duplications has not been used in real-time systems before.

A. Motivation

The design of any duplication heuristic in a non real-time system has two significant steps: *where* and *how* duplication is performed. Mainly, there are two strategies which are being used for the first step: duplicating subtasks in schedule holes [4] or allocating extra space other than holes [1]. The first approach adds more to the computational complexity of the algorithm but is more effective than the second. A number of approaches are used for the second step: 1) duplicate a single immediate predecessor (SIP) [4], 2) duplicate a chain of predecessors till the root node (COP) [1], 3) duplicate the immediate predecessors first and then the ancestors (IPFA) [5].

Duplication in real-time systems adds two more challenges to the above: *when* and *how much* duplication is to be performed. The first challenge has arisen because of our objective of meeting deadlines. In case, the deadline of a task is higher, there are enough chances to meet it without duplicating any subtask, which can create more schedule holes for other tasks to use, hence, increasing the schedulability. If it is decided to perform duplication, our motive is to control the amount of duplication according to the deadline, which is the second challenge. In this paper, we explore the first three challenges. Due to the space constraint, we leave the fourth challenge for future work.

The remainder of this paper is organized as follows: the system model is discussed in Section 2. Section 3 discusses the algorithm. Some initial simulation results are described in Section 4. Finally, section 5 concludes the paper.

II. System model

The system consists of a set P of m heterogeneous processors and a task set T of n precedence-constrained tasks. All the processors $p \in P$ are networked with a fully connected contention free network. It is assumed that the local memory of a processor is used for data exchange between subtasks assigned to it. A vector of the form $G(V_i, E_i, \mu_i, c_i), rt_i, pe_i, dl_i$ is used to represent a task $t_i \in T$. The first element of the vector is the directed acyclic graph G. The node set V_i represent the jobs[1] s_{ijk} (k is used for instance, V_i remains the same during instances) of t_i and the edges in E_i, the communications between the jobs. An edge $e_{ij} \in E$ represents the communication from node s_{ijk} to node s_{ilk}. A positive weight $\mu_i(j, p_q)$ is associated with node s_{ijk} which represents its computation cost on processor $p_q \in P$ and the non-negative weight $c_i j, l$ associated with edge $e_{ij} \in E$ represents the communication cost from s_{ijk} to s_{ilk}. The elements μ_i and c_i are matrices of the order $v_i \times m$ and $v_i \times v_i$ (v_i is the number of subtask in task t_i). We further assume that the DAG has single *entry* and *exit* nodes. If a DAG has multiple entry (exit) nodes then they are connected to zero-cost pseudo entry (exit) nodes with zero-cost edges. Performing this operation does not affect the final schedule. Next, rt_i is the release time of the task t_i and pe_i represent its period. The deadline dl_i is the relative end-to-end deadline of the task t_i, i.e. the exit node $s_{i(exit)k}$ of the k^{th} invocation of task t_i should finish by the absolute time $dl_{ik} = rt_{ik} + dl_i$, where rt_{ik} $\big(rt(t_i) + (k-1) \times pe(t_i)\big)$ and dl_{ik} are the release time and deadlines of the k^{th} invocation of the task t_i.

III. Real-Time Controlled Duplication Algorithm (RT-CDA)

The proposed algorithm, RT-CDA, is a com-

[1](the terms node, job and subtask have been used interchangeably)

bination of list-based and duplication scheduling heuristics. The complete strategy can be divided into three steps. First, the separate priority schemes for tasks $t_i \in T$ and subtasks $s_{ijk} \in V_i$ are described. These priorities direct the scheduler to select tasks and then subtasks for allocation to the processors. Next, a dynamic upper bound scheme is proposed. The question of *when* to duplicate can be solved by the proposed dynamic upper bound. The duplication is performed using IPFA [5] by utilizing schedule holes. The third step explains the complete scheduling scheme based on the upper bound and the deadlines.

A. Assigning Priorities

The tasks in the task set are considered for scheduling one at a time and are prioritized according to well known real-time scheduling heuristics: *earliest deadline first* (EDF) and *rate-monotonic* (RM) [8]. After a task instance t_{ik} is released on time rt_{ik}, it is inserted into the tasks schedule queue (SQ_t) according to EDF or RM. Both EDF and RM priority schemes are compared in the result section.

After the selection of a task t_i for scheduling, all the subtasks s_{ijk} of t_i are inserted in the subtasks schedule queue (SQ_{st}) according to non-decreasing order of their *b-level* ($bl(s_{ijk})$) values. The ties are broken using s-level ($sl(s_{ijk})$) values. The b-level (s-level) stands for the bottom (start) level, which is evaluated recursively in a bottom-up (top-down) fashion, traversing of the task graph starting from the exit (entry) node as shown by following equations.

$$bl(s_{ijk}) = \overline{\mu_i(j)} + \max_{s_{ilk} \in succ(s_{ijk})} \left(c_i(j,l) + bl_i(s_{ilk})\right) \tag{1}$$

$$sl(s_{ijk}) = \max_{s_{ilk} \in pred(s_{ijk})} \left(c_i(l,j) + sl_i(s_{ilk}) + \overline{\mu_i(l)}\right) \tag{2}$$

In the equations above, $bl(s_{i(exit)k}) = \overline{\mu_i(exit)}$ and $sl(s_{i(entry)k}) = zero$, whereas $succ(s_{ijk})$ and $pred(s_{ijk})$ is the list of immediate successors and predecessors of s_{ijk} and $\overline{\mu_i(j)}$ represents the average execution cost of subtask s_{ijk}. The $bl(s_{ijk})$ value is the critical path from the subtask s_{ijk} to $s_{i(exit)k}$. We have used bl as the primary priority parameter because, the critical path based algorithms are known to generate better schedules. Secondly, sl is basically the distance of a subtask s_{ijk} from $s_{i(entry)k}$. The subtask with lower sl is given higher priority, as it is present at a higher level in the task graph and should be scheduled first. It is worth noting that sorting the nodes according to bl, also performs a topological sort on all the $s_{ijk} \in V_i$, which satisfy the precedence constraints.

B. Upper Bound

In our scheme, for every k^{th} invocation of task t_i, we define a dynamic *upper bound* (UB_{ik}). The UB_{ik} is the time up to which the task t_{ik} can be definitely scheduled even without duplication. This bound is dynamically computed at run time considering the release time rt_{ik} of t_{ik}. To further strengthen the UB, the algorithm also considers the current computing load (subtasks already being scheduled).

The key idea for evaluating UB_{ik} is to schedule all the subtasks $s_{ijk} \in V_i$ of a task t_{ik} on a single processor (p_q) by using holes in the subtasks already scheduled on p_q. After this step we will get UB_{ik} of task t_{ik} on processor p_q i.e. $UB_{ikq} = EFT(s_{i(exit)k})$, where $EFT(s_{i(exit)k})$ is the earliest finish time of the exit node of t_{ik}. The UB_{ik} for the task t_{ik} can be evaluated using $UB_{ik} = \min_{p_q \in P} UB_{ikq}$.

As, UB is a trivial bound, it may not seem effective in the first place. But, the authors have observed that, even a popular scheduling algorithm heterogeneous earliest finish time (HEFT) [9] with single duplication or without duplication, generate schedules more than this UB for tasks having higher CCR values. Hence, UB becomes more effective with increasing CCR values as the tasks are scheduled on a single processor because of high communication cost between the nodes. This bound is used as a key element in our algorithm. A task t_{ik} having an absolute deadline $(rt_{ik} + dl_i) \geq UB_{ik}$, can be scheduled without duplication by RTCDA.

C. RTCDA Algorithm

The proposed algorithm RTCDA is an extension of the non-real time scheduling algorithm called heterogeneous earliest finish time (HEFT) [9].

HEFT works on a single task graph and schedules a subtask on to the processor which finishes it at the earliest. Unlike HEFT, RTCDA works on periodic real-time task sets. Conceptually, both algorithms differ on the length of the generated schedule. The RTCDA algorithm is driven by the UB. In the worst case, it makes sure that the schedule for a single task instance always follow its UB. However, HEFT has been observed to generate schedules more than the UB.

The algorithm begins by calculating the hyper period (hp) of the task set T. The hp is evaluated as the least common multiple of all tasks periods. The schedule is generated from $zero$ time unit till the hp. Next, the ready tasks are inserted into the SQ_t according to EDF or RM. A task t_{ik} (if present), is fetched from the head of the SQ_t for processing. At first, the UB_{ik} is evaluated. Then, the algorithm checks whether the task t_{ik} deadline is greater than or equal to the UB_{ik} i.e. $dl_{ik} \geq UB_{ik}$. If it is, then RTCDA can meet the deadline of t_{ik} without duplication, otherwise it schedules the task with duplication. The subtasks $s_{ijk} \in V_i$ are inserted into the SQ_{st} for processing according to their b-level (bl) and s-level (sl). Before scheduling, RTCDA calculates the earliest start time of all $s_{ijk} \in V_i$ ($EST(s_{ijk}, p_{ub})$) on processor p_{ub} if they all execute according to their order in SQ_{st} on processor p_{ub}. The p_{ub} represents the processor on which the current task t_{ik} has the UB_{ik}.

A subtask s_{ijk} is fetched from the head of the SQ_{st}. Next, the algorithm finds the earliest finish time of s_{ijk} with or without duplication as decided by the UB_{ik}. If the processor on which s_{ijk} has the earliest finish time is same as that of p_{ub}, s_{ijk} is scheduled on p_{ub}. In the other scenario, RTCDA makes sure that, scheduling s_{ijk} on any processor p_z other than p_{ub} does not increase the worst schedule length i.e. UB by satisfying the following equation for all $\{s_{ilk} \in succ(s_{ijk})\}$.

$$EFT_{s_{ijk}} + c_i(j,l) \leq EST(s_{ilk}, p_{ub}) \qquad (3)$$

If a subtask s_{ijk} satisfies the above equation then it is scheduled on p_z other wise on p_{ub}.

During the evaluation of $EFT(s_{ijk})$ on a processor p_z, the algorithm checks for the possibility of duplicating predecessors if it improves the $EFT(s_{ijk})$ on p_z. The RTCDA is flexible in the selection of *how* duplication is performed. We have used the IPFA as proposed in [5]. In this strategy, the immediate predecessors s_{ilk} of subtask s_{ijk} are considered for duplication in the schedule holes. The predecessors are selected for duplication according to non-increasing order of the time they delay s_{ijk}. Next, the algorithm recursively attempts to duplicate the predecessors of any duplicated subtasks. Thus, as many ancestors as allowable are duplicated, in a breadth-first fashion. Duplication recursively continues until no further duplication is possible.

IV. Results

In this section, we present the comparative analysis of RTCDA-RM/EDF with the two other relevant algorithms: RTDBA and RTHEFT-RM/EDF. RTDBA is proposed in [2] with only rate-monotonic scheduling where as RTHEFT is our real-time extension of HEFT [9] with duplication. Importantly, the time complexity of RTHEFT is same as that of RTCDA. All the algorithms are tested on three thousand task sets (500 per graph) generated with a well known real-time benchmark, Task Graphs For Free (TGFF) [10]. The parameters used for the simulation are; Number of tasks in a task set (4-400), Number of subtasks in a task (10-1000), CCR (1,3,5,7,10) and Utilization (0.1,0.3,0.5,0.7,1). For all these preliminary results the task deadlines have been set equal to the their periods. All the algorithms have been implemented in C++.

Figures 1 and 2 show the effect of varying CCR and utilization (UT) on the algorithms. The utilization of the task set is measured as the summation of the utilizations of all the tasks in the task set. For a single task, it is defined as $UT_i = \frac{total\_computation_i}{period_i}$, whereas $total\_computation_i$ is the sum of the averages of the subtasks execution costs in the task.

The results show that RTCDA offers better performance than the others. It has remained consistent during variation in the CCR values. Even at higher utilizations, UT=0.6 and UT=0.8, it has achieved guarantee ratios (GR or SR) of 95% and 75% respectively, which is at minimum 20% more than all others. The guarantee ratio is defined as $GR = \frac{number\_of\_tasksets\_meeting\_deadlines}{total\_number\_of\_tasksets}$. The primary reason of this performance, is controlled duplication, which leads to more schedule holes

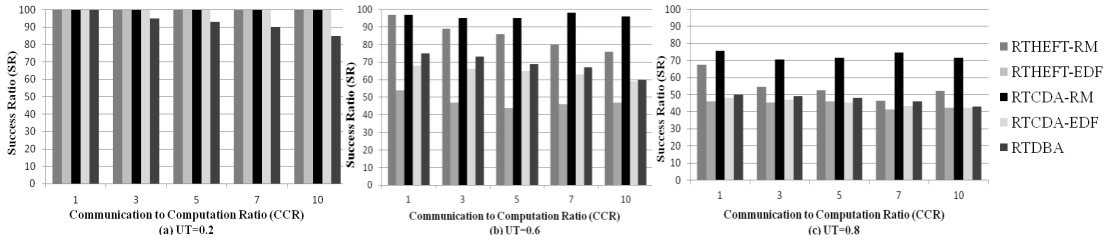

Fig. 1. Effect of CCR on the performance of RTHEFT-RM/EDF, RTCDA-RM/EDF, RTDBA

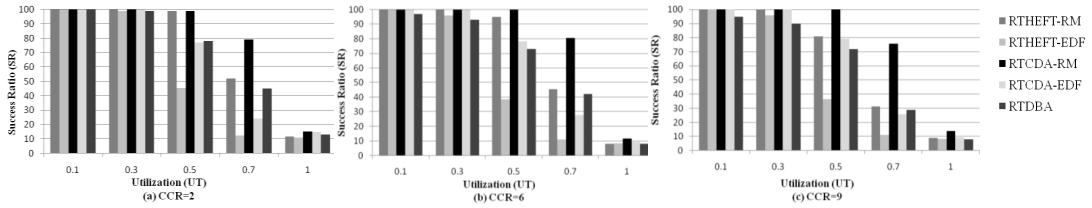

Fig. 2. Effect of Utilization on the performance of RTHEFT-RM/EDF, RTCDA-RM/EDF, RTDBA

for the other tasks to use. With increasing CCR, the schedule holes also increase, which is why RTCDA is more effective at higher CCR and UT values. We also observe that all other algorithms dominate RTDBA. This is because, RTDBA does not utilizes schedule holes for the sake of lower complexity. With the increase in utilization (Figure 2), the success ratios of all the algorithms decrease. But, again RTCDA achieves an SR of more than 75% for a higher utilizations of 0.7 and 0.8.

V. Conclusion and Future Directions

From the simulations, we observe that duplication is not always required for the scheduling of real-time static tasks. Instead, it (duplication) depends on the deadlines of the tasks. Also, the upper bound strategy has effectively answered the *when* duplication challenge. In the future, the fourth duplication challenge of *how much* needs to be looked at. According to our observation, controlling the amount of duplication can further improve the success ratio. Furthermore, an extensive comparison analysis is required which should include different duplication strategies, variation in deadlines and release times and different comparison metrics.

References

[1] R. Bajaj and D. P. Agrawal, "Improving scheduling of tasks in a heterogeneous environment," *IEEE Trans. Parallel Distrib. Syst.*, vol. 15, pp. 107–118, February 2004.

[2] N. Auluck and D. Agrawal, "Enhancing the schedulability of Real-Time heterogeneous networks of workstations (NOWs)," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 20, no. 11, pp. 1586–1599, 2009.

[3] I. Ahmad and Y.-K. Kwok, "On exploiting task duplication in parallel program scheduling," *IEEE Trans. Parallel Distrib. Syst.*, vol. 9, pp. 872–892, September 1998.

[4] S. Bansal, P. Kumar, and K. Singh, "Dealing with heterogeneity through limited duplication for scheduling precedence constrained task graphs," *J. Parallel Distrib. Comput.*, vol. 65, pp. 479–491, April 2005.

[5] S. Baskiyar and C. Dickinson, "Scheduling directed a-cyclic task graphs on a bounded set of heterogeneous processors using task duplication," *J. Parallel Distrib. Comput.*, vol. 65, pp. 911–921, August 2005.

[6] S. Ranaweera and D. P. Agrawal, "Scheduling of periodic time critical applications for pipelined execution on heterogeneous systems," in *Parallel Processing, International Conference on*, Los Alamitos, CA, USA, 2001, p. 0131.

[7] N. Auluck, "An integrated scheduling algorithm for precedence constrained hard and soft Real-Time tasks on heterogeneous multiprocessors," in *Lecture Notes in Computer Science*, vol. Volume 3207/2004, 2004, pp. 199–207.

[8] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a Hard-Real-Time environment," *Journal of the ACM (JACM)*, vol. 20, no. 1, pp. 46 – 61, 1973.

[9] H. Topcuouglu, S. Hariri, and M. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 13, pp. 260–274, 2002.

[10] R. P. Dick, D. L. Rhodes, and W. Wolf, "Tgff: Task graphs for free," 1998.