IOSR Journal of Computer Engineering (IOSR-JCE)
e-ISSN: 2278-0661,p-ISSN: 2278-8727, Volume 17, Issue 6, Ver. I (Nov – Dec. 2015), PP 77-86
www.iosrjournals.org

Efficient Dynamic Scheduling Algorithm for Real-Time Multi-Core Systems

Pratap Suryawanshi[1], Radhakrishna Naik[2], Ishwar Chaudhary[3]
[1,2,3]*(Computer Science and Engineering Department, Marathwada Institute of Technology, India)*

Abstract : *Imprecise computation model is used in dynamic scheduling algorithm having heuristic function to schedule task sets. A task is characterized by ready time, worst case computation time, deadline and resource requirements. A task failing to meet its deadline and resource requirements on time is split into mandatory part and optional part. These sub-tasks of a task can execute concurrently on multiple cores, thus achieving parallelization provided by the multi-core system. Mandatory part produces acceptable results while optional part refines the result further. To study the effectiveness of proposed scheduling algorithm, extensive simulation studies have been carried out. Performance of proposed scheduling algorithm is compared with myopic and improved myopic scheduling algorithm. The simulation studies shows that schedulability of task split myopic algorithm is always higher than myopic and improved myopic algorithm.*

Keywords: *Dynamic real time scheduling, heuristic, imprecise computation, multi-core, real time systems, resource requirement.*

I. Introduction

In real time systems, correct computational results generated on time are required for a system to work properly [1]. With wide applications of real time systems need for high computing hardware platform is also there. Multi-core systems are emerging rapidly in real time systems. Multi-core system has high computing power, more reliability and low energy consumption which makes them suitable for real- time systems. To optimally exploit these high computing platforms efficient scheduling algorithms are also needed.

Many efficient algorithms are there which take processor requirement into consideration. *Stankovic* and *Ramamritham* proposed heuristic scheduling algorithms which take processor requirement as well as resource requirement into consideration [2]. Locking of resources and waiting of tasks might cause tasks to miss their deadlines and thus reducing the predictability in real time systems [2]. In this paper dynamic scheduling algorithm is developed which exploits the parallelization in tasks and strives to meet deadlines of tasks with reduced computational complexity.

Optimal scheduling algorithms such as Rate Monotonic, Earliest Deadline First etc. do not take resource requirement into consideration. Heuristic scheduling algorithm performs better than these algorithms. Heuristic scheduling algorithm considers complete task set while searching for feasible schedule and hence has a computational complexity of $(O(n^2))$. Myopic scheduling algorithm using heuristic function for scheduling considers only three tasks of complete task set while searching for feasible schedule [3]. Myopic algorithm has reduced complexity of $(O(n))$. Task split myopic algorithm developed in this paper has same computational complexity as that of myopic algorithm but with higher schedulability and less slack times between tasks executing on cores.

Parallelizable task scheduling algorithm in [4] exploits parallelization in periodic tasks to schedule task sets. Performance of myopic algorithm on heterogeneous processors is show in [5]. Minimal earliest finish time algorithm shows that processor having smallest earliest available time performs better. Effect of different window size and processing times of tasks on the performance of algorithm is shown in [6]. Myopic algorithm performs better with tasks having smaller processing times and performance further increases for larger window sizes.

Main purpose of task split myopic algorithm is to exploit the parallelization provided by the multi-core processing platform. Parallelization in task set is exploited through *imprecise computation model* which splits task into *mandatory* sub-task and *optional* sub-task. Task not meeting its deadline is allowed to execute on multiple cores, thus achieving parallelization and better utilization of all *executing* cores. Better utilization of *executing* cores increases the probability of task meeting its deadline.

Rest of the paper is organized as follows; Section 2 gives brief information on background model. In section 3, related work is discussed. In section 4, task split myopic scheduling algorithm is proposed. In section 5, case studies and simulation results are presented. In section 6, performance analysis of improved myopic and task split myopic is illustrated. Section 7 concludes the paper.

II. Background

2.1 Multi-Core Processing Model

Multi-core processing model is assumed for scheduling proposed algorithm. A quad core processing platform is considered having shared global memory. $Core_0$ is reserved for scheduling and decision making also called as scheduler while $core_1$, $core_2$ and $core_3$ will be executing scheduled tasks also called as executing-cores.

In shared global memory model task are loaded into shared global memory and arrive at scheduler ($core_0$). Scheduler distributes these tasks to executing-cores for execution. The communication between scheduler and executing-cores is through dispatch queues, each executing core has its dispatch queue. Scheduler will be running in parallel with executing-cores scheduling newly arrived tasks and updating dispatch queues. This organization ensures executing-cores will always find some tasks in dispatch queues when they finish executing current tasks.

Executing-cores are of homogeneous kind and do not impose any relationship between them and tasks, thus tasks are free to execute on any executing-core. Task finds a first available core among executing-core for execution.

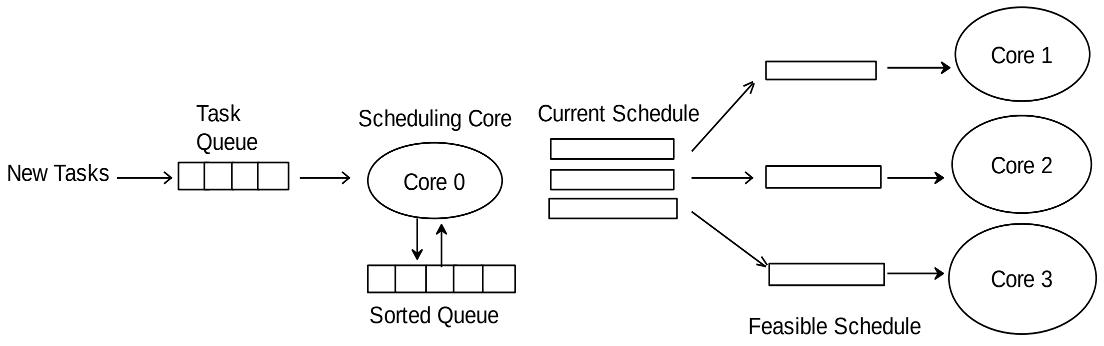


Figure: 1 Multi-Core Architecture for Split Myopic Algorithm

1.2 Task Model

Tasks are dispatchable entities and have following notations:

- Task arrival time, T_A
- Task deadline, T_D
- Task worst case execution time, T_P
- Task resource requirement, T_R

Assumptions of tasks

- Task consist of mandatory sub-task and optional sub-task
- Tasks are independent, non preemptive and non-periodic
- Task uses a resource either in shared mode or exclusive mode
- For a schedule to be feasible following condition must be true for every task,

$$0 \le T_A \le T_{EST} \le T_D - T_P \quad \text{.... (1)}$$

where, T_{EST} is the earliest start time of a task

2.3. Terminology

i. Feasible Task: A task is said to be feasible if its timing constraints and resource requirements are met in the schedule.

ii. Partial Schedule: A partial schedule is a feasible schedule for a subset of tasks. A partial schedule is said to be strongly feasible if all the schedules obtained by extending the current schedule by any of the remaining tasks are also feasible.

iii. $EAT_s(EAT_k)$ **:** It is the earliest time when resource R becomes available for shared (or exclusive) usage.

iv. $EST(T_i)$ **:** It is the earliest start time of a task at which it can start its execution. Assume, r_i is the ready time of a task. C is the set of cores, R_i be the resource requested by the task, $EST(T_i)$ is given by,

$$EST(T_i) = MAX(T_i, Min_{j \in c}(availtime(j)), MAX(EAT_k)) \quad \text{.... (2)}$$

$EST(T_i)$ where $availtime(j)$ denotes the earliest time at which the core C_j becomes available for executing a task and the third term denotes the maximum among the earliest available times of the resource requested by task (T_i), in which $u = s$ for shared mode and $u = e$ for exclusive mode.

v. LAXITY (T): Expresses the laxity degree of task T, it is given by, $LAXITY(T) = \frac{T_D - EST(T)}{T_C}$ .... (3)

vi. Mandatory sub-task: Sub-task (T_i') of task (T_i) that is required to complete its execution before its deadline to produce acceptable imprecise results.

vii. Optional sub-task: Sub-task (T_i'') of task (T_i), refines the imprecise result upon its completion before its deadline, but can be terminated at its deadline even if its execution is still going on.

viii. Upper Bound Utilization of Cores $(UB(C_i))$: Core (C_u) among executing cores having busy time greater than other two cores is given by,

$$UB(T) = MAX(C_i, C_j, C_k) \quad ..... (4)$$

where C_i, C_j, C_k are executing cores.

ix. Lower Bound Utilization of Cores $(LB(C_i))$: Core (C_l) among executing cores having busy time less than other two cores is given by,

$$LB(T) = MIN(C_i, C_j, C_k) \quad ..... (5)$$

where C_i, C_j, C_k are executing cores.

x. Middle Bound Utilization of Cores $(MB(C_i))$: Core (C_m) among executing cores having busy time in between UB core and LB core is given by,

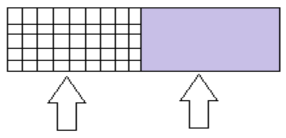
$$MB(T) = C_i \Lambda C_j \Lambda C_k \Lambda UB \Lambda LB \quad ..... (6)$$

where C_i, C_j, C_k are executing cores.

III. Related Work

3.1 Imprecise Computation Model

If any time critical task fails to complete and produce results by its deadline, timing fault occurs. Imprecise computation technique minimizes this difficulty; it achieves graceful degradation and produces approximate results of acceptable quality [7].



Mandatory Sub-Task Optional Sub-Task
Figure: 2 Logical Decomposition of Task

Basic strategy is to minimize effect of timing faults by leaving less important tasks unfinished if necessary. Imprecise Computation technique splits task into *mandatory* sub-task and *optional* sub-task. Programmers structure every time critical task so that it can be logically decompose into mandatory sub-task and optional sub-task [7]. Mandatory sub-task is required to complete its execution before deadline to generate acceptable result. Optional sub-task refines this result. It can be left unfinished and can be terminated at its deadline, if necessary.

The result produced by task which completes its execution before deadline is precise, has a zero error. If the task is terminated before its completion, the intermediate result produced at that point is acceptable as long as mandatory sub-task is complete. This intermediate result is called imprecise computation.

3.2 Efficient Dynamic Scheduling Algorithm for Multiprocessor Real Time System

Efficient scheduling algorithms based on heuristic functions were proposed by *Ramamritham* and *Stankovic*. Following simple heuristic functions were compared,

- Minimum Deadline First (Min_D)
- Minimum Processing Time First (Min_P)

- Minimum Earliest Time First (Min_S)
- Minimum Laxity First (Min_L)

Out of these functions (Min_D) performed better. Above simple heuristic functions were further integrated to give better results. Integrated heuristic functions take resource requirement into consideration while simple heuristic functions only consider processor requirement [2]. Following are the integrated heuristic functions,

- $Min\_D + Min\_P$
- $Min\_D + Min\_S$

Among the above functions, $Min\_D + Min\_S$ performs better.

3.3 Myopic Scheduling Algorithm

Myopic algorithm was also proposed by *Ramamritham* and *Stankovic* with a heuristic function which considers deadline and resource requirements. At each search level myopic algorithm searches tasks within a window for partial feasible schedule. In myopic algorithm partial schedule is extended by one of the remaining task in a specified window and having best heuristic value.

Myopic algorithm has a reduced computational complexity of *O(n)* than original heuristic algorithm *O(n^2)*. Complexity is reduced due to limiting the search space for finding a feasible partial schedule.

- **{Tasks-remaining}:** The tasks that remain to be scheduled. Tasks in {Tasks-remaining} are arranged in the order of increasing deadlines.
- N_r **:** Number of tasks in {Tasks-remaining}.
- k **:** Maximum number of tasks in Tasks-remaining considered for myopic scheduling algorithm
- N_k **:** Actual number of tasks in { Tasks-remaining} considered by the myopic algorithm at each step of scheduling, given by,

$$N_k = \min(k, N_r) \qquad \dots (7)$$

- **{Tasks-considered}:** The first N_k tasks in {Tasks remaining}.

Termination conditions of algorithm are either,

- A complete schedule is obtained.
- Maximum number of backtracks or number of evaluations of H functions has been reached.
- No more backtracking is possible.

3.4 Improved Myopic Scheduling Algorithm

Improved myopic scheduling algorithm was proposed by *Zhu Xiangbin*. Myopic algorithm does not consider processing time while calculating heuristic value of task. *Zhu Xiangbin* added laxity parameter to heuristic function. Due to this task having low laxity has a higher probability of getting processor time than task with high laxity [8]. In improved myopic scheduling algorithm heuristic function was given by,

$$H(T) = T_D + W_1 * IEST(T) + W_2 * Laxity(T) \qquad \dots\dots (8)$$

Steps in improved myopic scheduling algorithm,

```
Begin;
1) Order the task in non-decreasing order of their deadlines and then start with an empty partial schedule.
2) Determine whether the current vertex is strongly feasible by performing feasibility check for k tasks in the feasibility check window. If the current vertex is strongly feasible, feasible=TRUE, otherwise, feasible=FALSE;
3) If (feasible is TRUE)
    3.1) Computer the values of $H$ function for all tasks in feasibility check window.
    3.2) Extend the schedule by the task having the smallest $H$ value.
4) Else
    4.1) Backtrack to the previous search level.
    4.2) Extend the schedule by the task having the next best $H$ value
5) Move the feasibility check window by one task.
6) Repeat steps 2-5 until terminal condition is met.
End;
```

IV. Task Split Myopic Scheduling Algorithm

In this section proposed dynamic scheduling algorithm is discussed, which exploits parallelism present in tasks to meet their deadlines. The proposed task split myopic algorithm is a variant of myopic scheduling algorithm in [1]. Myopic or improved myopic algorithm checks first k tasks for feasibility, which is also known as feasibility check window. Larger the size of feasibility check window, higher the scheduling cost.
The proposed scheduling algorithm is similar to myopic and improved myopic scheduling algorithm except that it parallelizes task when there is not enough capacity left on core to meet that task's deadline.

Steps in task split myopic scheduling algorithm,

```
Begin;
1) Order the task in non-decreasing order of their deadlines and then start with an empty partial schedule.
2) Determine whether the current vertex is strongly feasible by performing feasibility check for k tasks in the feasibility check window. If the current vertex is strongly feasible, feasible=TRUE, otherwise, feasible=FALSE;
3) If (feasible is TRUE)
    3.1) Computer the values of $H$ function for all tasks in feasibility check window.
    3.2) Extend the schedule by the task having the smallest $H$ value.
4) Else
    4.1) Split the task leading to infeasibility into mandatory sub-task and optional sub-task
    4.2) Extend the schedule by the task having    the next best $H$ value
5) Move the feasibility check window by one task.
6) Repeat steps 2-5 until terminal condition is met.
End;
```

The termination conditions are either
1) A complete feasible schedule has been found or,
2) Maximum number of backtracks are reached or,
3) No more task splitting is possible

The heuristic function $H(T) = T_D + W_1 * IEST(T) + W_2 * Laxity(T)$ is an integrated heuristic which captures the deadline, resource requirement and laxity of a task [4]. The algorithm instead of backtrack splits the task into mandatory sub-task and optional sub-task when the task cannot meet its deadline. Mandatory sub-task will execute on earliest available core while optional sub-task will execute on second earliest available core. Tasks use the required resource in shared or exclusive mode. Resource requirements are denoted as below,
$T_R = (T_{R1}(1), T_{R2}(2), ...T_{RN}(r))$, where

$$T_R = 0, T\_does\_not\_require\_resource$$
$$T_R = 1, T\_require\_resource(R_i)\_in\_shared\_\mathrm{mod}\,e$$
$$T_R = 2, T\_require\_resource(R_i)\_in\_exclusive\_\mathrm{mod}\,e$$

The complexity of proposed algorithm for scheduling non-periodic, non-preemptive tasks is O(kn) which is same as that of myopic algorithm and improved myopic algorithm.

V. Performance Evaluation

To study the effectiveness of imprecise computation model in meeting task's deadline, three case studies have been stimulated. Here focus is on whether or not all the tasks in the task set finish before their deadlines. Therefore the most important metric is success ratio, which is defined as number of task sets found schedulable to the number of task sets considered for scheduling.

Here it is shown how the workload of task sets are distributed to the executing cores under the influence of imprecise computation model in task split myopic scheduling algorithm. Another performance metric considered is upper bound utilization which is defined as the time instance till which executing cores are remaining busy while executing task sets. Low upper bound utilization denotes equal distribution of tasks to executing cores and less time required to complete a task set.

Simulation Parameters

Three case studies, each with different scenario are simulated here. There are three executing cores and two single instance resources (R_1 and R_2). Values of W_1, W_2 and number of task splits are fixed to 1, 1 and 2 respectively. Textured boxes represent the mandatory sub-task while shaded boxes represent optional sub-task in schedules obtained by task split myopic algorithm.

Simulation Notations

- Task ID is denoted by, $\mathbf{ID_i}$
- Ready time of task, $\mathbf{R_i}$.
- Computation time of task, $\mathbf{C_i}$.
- Computation time of mandatory sub-task, $\mathbf{M_i}$.
- Computation time of optional sub-task, $\mathbf{O_i}$.
- Deadline of task, $\mathbf{D_i}$.
- Resource requirement of task, $\mathbf{RES_i}$.

Generation of Tasks

i) The computation time C_i of task T_i is randomly chosen between $Min\_C(1)$ and $Max\_C(50)$.

ii) The deadline of each task T_i is randomly chosen in the range computation time at max, $C_{\max}$ and $C_{\max}+R$, where R, is the simulation parameter also called as laxity parameter and

$$C_{\max} = Ready\_Time(R_i) + C_i \text{ .... (10)}$$

iii) The resource requirements of a task are generated based on the input parameters *USeP* and *ShareP*.

5.1 Case Study 1

Table 1: Task Set 1

$\mathbf{ID_i}$	$\mathbf{R_i}$	$\mathbf{C_i}$	$\mathbf{M_i}$	$\mathbf{O_i}$	$\mathbf{D_i}$	$\mathbf{RES_i}$
1	0	10	5	5	12	2
2	0	8	4	4	18	1
3	0	20	10	10	28	0
4	6	15	8	7	34	2
5	8	4	2	2	43	1
6	15	23	12	11	52	0

In case study I there are six tasks having resource requirements. Their parameters are shown in Table 1. At the first node, feasibility check window is having tasks T_1, T_2 and T_3 task with best heuristic value (T_1) will be selected and it will execute on *core₁*. At this stage first partial feasible schedule is obtained, now the window will have tasks T_2, T_3, T_4, again task T_2 is having best heuristic value, so it will execute on *core₂*. Feasibility window will move by one step having tasks T_3, T_4, T_5, T_3 is having best heuristic value, it will execute on *core₃*, and so on.

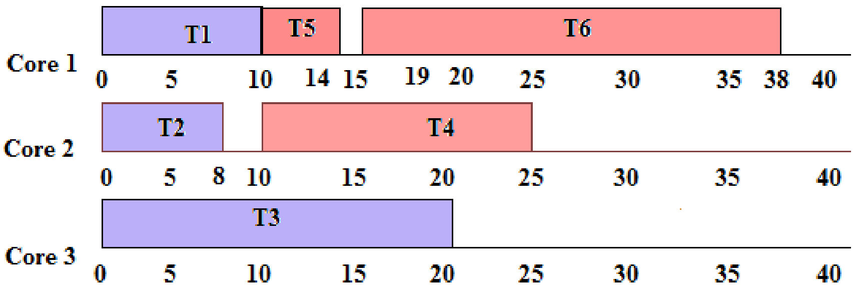


Figure 3: Feasible schedule of improved myopic for case study I

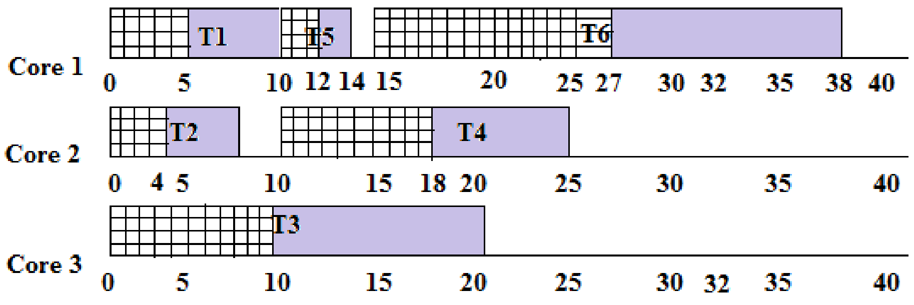


Figure 4: Feasible schedule of task split myopic for case study I

Fig. 3 represents the schedule obtained by improved myopic algorithm while Fig. 4 represents the schedule obtained by task split myopic algorithm. Both are feasible schedules. In Fig. 4 every task executes its sub-tasks in sequential manner or mandatory and optional sub-task are executing on single core for every task because no task in schedule is leading to infeasibility. Tasks in task set are meeting their deadlines after scheduling them with improved myopic or task split myopic. Upper bound utilization is 38 time units for both scheduling algorithms.

5.2 Case Study II

Table 2: Task set 2

ID_i	R_i	C_i	M_i	O_i	D_i	RES_i
1	0	10	5	5	12	2
2	0	14	7	7	18	1
3	0	27	14	13	28	1
4	8	11	6	5	38	0
5	12	14	7	7	40	2
6	15	30	15	15	51	1

In second case there are again six tasks and their parameters are as shown in table 2. Below Fig. 5 represents the schedule obtained by improved myopic algorithm while Fig. 6 represents the schedule obtained by task split myopic algorithm for case study II. Both are feasible schedules. In Fig. 3, while building schedule at window T_4, T_5, T_6, task T_6 is leading to infeasibility so improved myopic algorithm will backtrack to next best task in window, hence, T_5 will execute first on earliest available core, then T_6 will execute and lastly T_4 will execute, thus finding a feasible schedule.

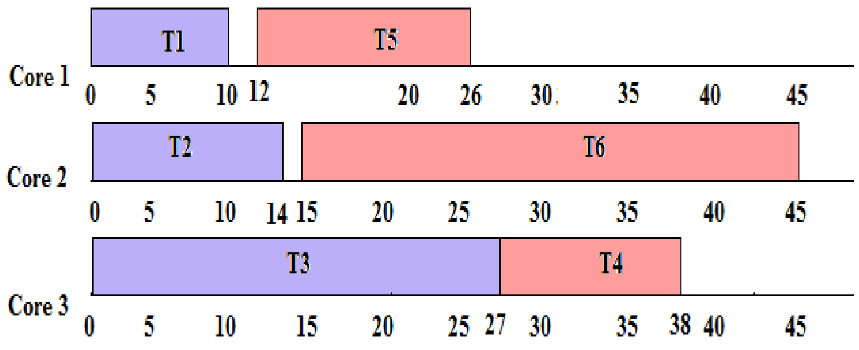


Figure 5: Feasible schedule of improved myopic for case study II

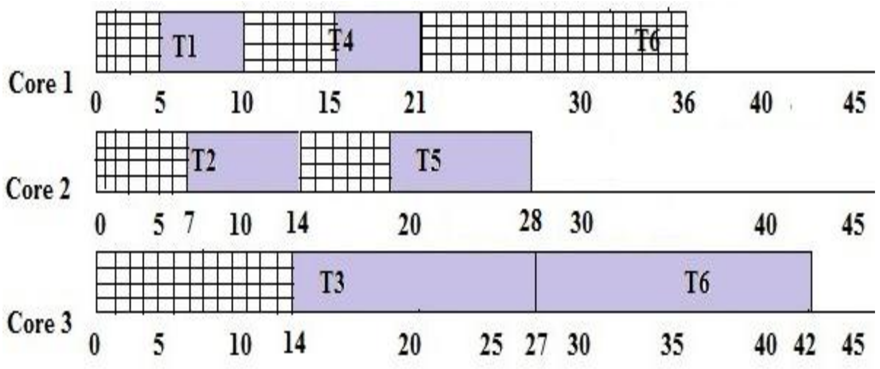


Figure 6: Feasible schedule of task split myopic for case study II

In task split myopic algorithm while scheduling task set in case study II, task T_6 is leading to infeasibility as shown in Fig. 5, so it will be split and mandatory sub-task will execute on earliest available core i.e. *core*$_1$ and optional sub-task will execute on middle bound core i.e. *core*$_3$. Upper bound utilization of cores is 45 time units for improved myopic algorithm while it is 42 time units for task split myopic algorithm.

5.3 Case Study III

Table 3: Task set 3

ID_i	R_i	C_i	M_i	O_i	D_i	RES_i
1	0	20	10	10	22	1
2	0	23	12	11	23	2
3	0	40	20	20	47	0
4	15	20	10	10	50	1
5	25	25	13	12	60	0
6	30	44	22	22	74	2

Third case study consists of six tasks and their parameters are shown in Table 3. Fig. 7 shows the infeasible schedule obtained by improved myopic algorithm and Fig. 8 shows the feasible schedule obtained by task split myopic algorithm for the task set in case study III.

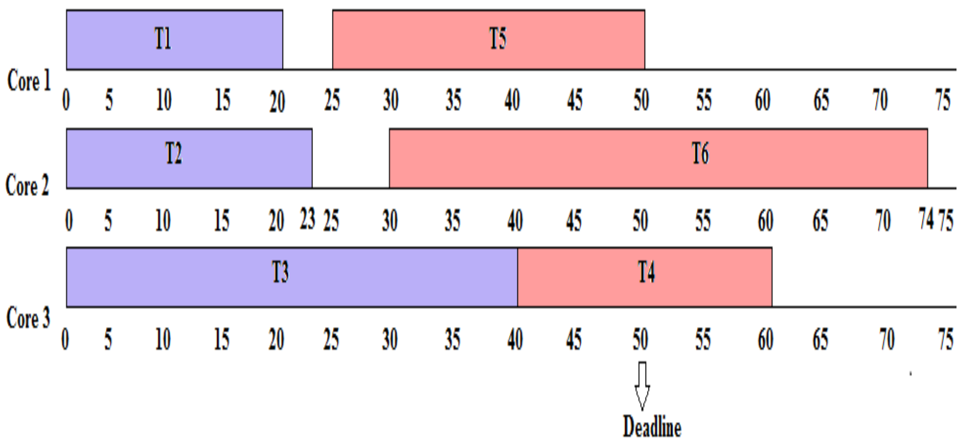


Figure 7: Infeasible schedule of improved myopic for case study III

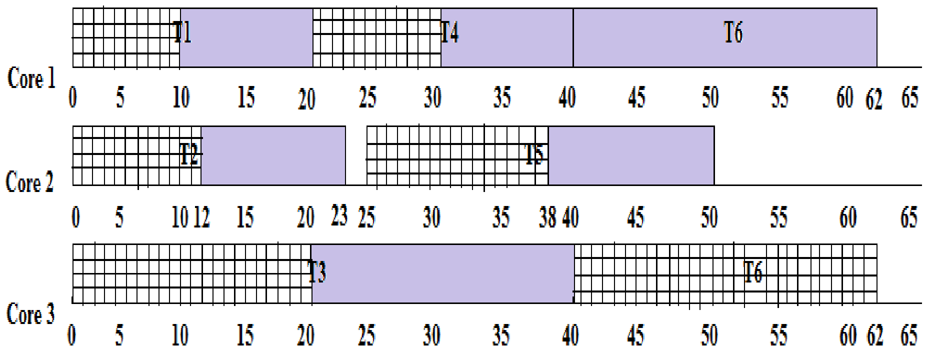


Figure 8: Feasible schedule of task split myopic for case study III

In Fig. 7, while building schedule at window T_4, T_5, T_6, task T_6 is leading to infeasibility so improved myopic algorithm will backtrack to next best task in window, hence T_5 will execute first on earliest available core, then T_6 will execute and lastly T_4 will execute, but this T_4 task also leading to infeasibility. Improved myopic algorithm is not able to find feasible schedule even after backtrack for the task set in case study III. In task split myopic algorithm while scheduling task set, task T_6 is leading to infeasibility, so it will be split and mandatory sub-task will execute on earliest available core i.e. $core_1$ or $core_3$ and optional sub-task will execute on $core_1$ or $core_3$ whichever not used by mandatory sub-task. Upper bound utilization of cores is 74 time units for improved myopic algorithm while it is 62 time units for task split myopic algorithm.

VI. Performance Analysis

Performance analysis of improved myopic algorithm and task split myopic algorithm is carried out on the basis of three parameters namely, Upper bound utilization, success ratio and average waiting time of task set.

Upper bound utilization of executing cores is given by the core having highest busy time among three executing cores. It also denotes time required by cores to finish execution of a task set.

Waiting time of task in a task set is given by, $W.T. = EST(T_i) - Ready\_Time(R_i)$ .. (11)

It shows for much time task has to wait once it is ready till it gets scheduled.

Table 4: Comparative Study of Algorithms with respect to Upper Bound Utilization

	Algorithm / Case Study	Improved Myopic (time units)	Task Split Myopic (time units)
Upper Bound Utilization	Case Study I	38	38
	Case Study II	45	42
	Case Study III	74	62

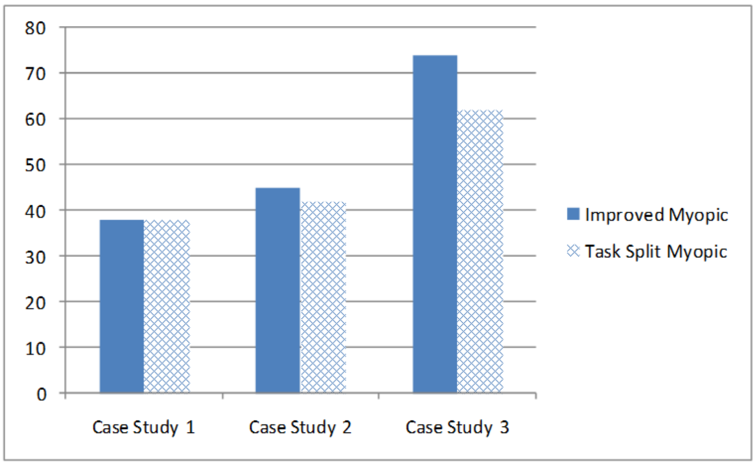


Figure 9: Upper Bound Utilization Analysis

Table 5: Comparative Study of Algorithms with respect to Average Waiting Time

	Algorithm / Case Study	Improved Myopic (time units)	Task Split Myopic (time units)
Average Waiting Time	Case Study I	1	1
	Case Study II	3.5	1.3
	Case Study III	4.16	2.5

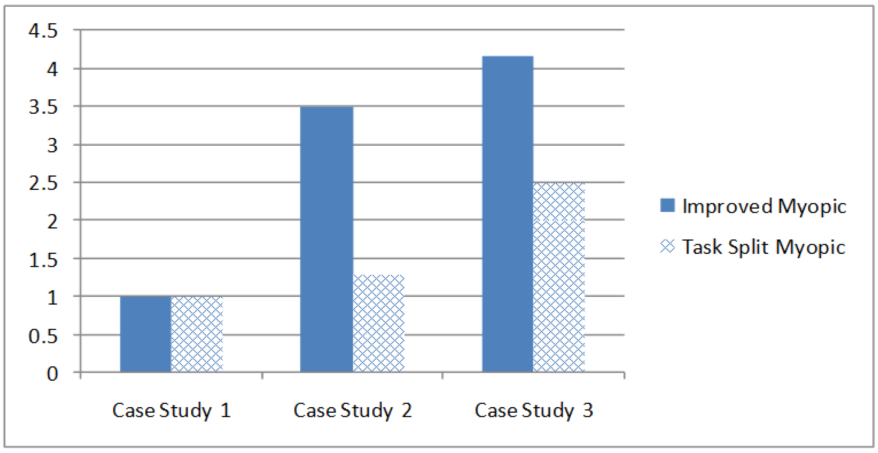


Figure 10: Average Waiting Time Analysis

Success ratio in scheduling task sets in case study I, II and III by improved myopic algorithm is 0.6 while that of task split myopic algorithm is 1.

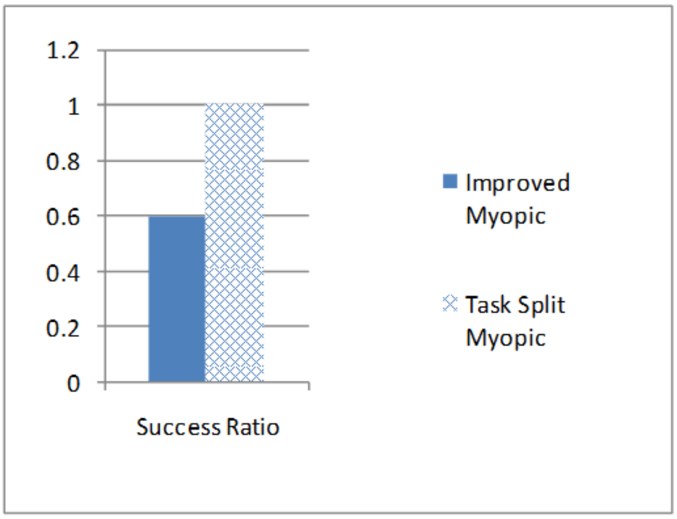


Figure 11: Success Ratio Analysis

VII. Conclusion

Once a feasible schedule is obtained by a dynamic scheduling algorithm it is guaranteed that all the tasks in task set will execute before their deadlines. Multi-Core processing platform requires efficient dynamic scheduling algorithm to exploit its processing power optimally.

Task split myopic algorithm exploits the parallelization present in tasks to schedule the tasks under the influence of imprecise computation model. This use of parallelization helps tasks in meeting their deadlines when there is not enough capacity left on cores. Splitting tasks into mandatory sub-task and optional sub-task also results in better utilization of cores.

Upper bound utilization of cores in split myopic algorithm is reduced by 11.46% than improved myopic algorithm.

Average waiting time of tasks in task split myopic algorithm is 43.7% less than improved myopic algorithm.

In three case studies improved myopic algorithm achieves 66.33% schedulability while task split myopic algorithm achieves 100% schedulability.

References

[1] Handbook of Real-Time and Embedded Systems (Chapman & HallCrc Computer & Information Science Series), 2008.
[2] W, Zhao and K. Ramamritham, “Simple and Integrated Heuristic Algorithms for Scheduling Tasks with Time and Resource Constraints” J. Syst. Software, 1987.
[3] K. Ramamritham, J.A. Stankovic, and P.F. Shiah, “Efficient Scheduhng Algorithms for Real- Time Multiprocessor Systems”, IEEE Trans. on Parallel and Disfribufed System, Vol.1, No.2, pp.184194, April, 1990.
[4] G Manimaran and C.S.R Murthy, “An Efficient Dynamic Scheduling Algorithm for Real-Time Systems”, IEEE Trans. on Parallel and Distributed Systems, Vo1.9, No.3, pp.312-319, March, 1998.
[5] Yang Yuhai, Yu Shengsheng, ” A new Dynamic Scheduling Algorithm for Real-Time Heterogeneous Multiprocessor Systems”, Intelligent Information Technology Application, pp. 112-115, December 2007.
[6] Kazi Sakib, M.S. Hasan, “Effects of Hard Real Time Constraints in Implementing the Myopic Scheduling Algorithm”, Journal of Computer Science, Vol. 1 and 2, July, 2014
[7] Jane W.S. Liu and Kwei-Jay Lin, “Algorithms for Scheduling Imprecise Computations”, Computer Systems, 1991. pp. 58-68, May, 1991.
[8] Zhu Xiangbin and Tu Shiliang,“An Improved Dynamic Scheduling Algorithm for Multiprocessor Real Time Systems”, Parallel and Distributed Computing, Applications and Technologies, pp.710-714, August, 2003.
[9] Hitesh P. Daulani, Dr. Radhakrishna Naik and Pavan S. Wankhade,“Precedence Constraint Task Scheduling for Multicore Multikernel Architecture”, IOSR Journal of Computer Engineering (IOSR-JCE), Volume 16, Issue 4, Ver. II , PP 43-53, August 2014.
[10] W. Zhao, K. Ramamritham, and J. A. Stankovic, “Scheduling Tasks with Resource Requirements in Hard Real-Time Systems,” IEEE Trans. Software Eng., vol. SE-12, May 1987.
[11] Robert I. Davis and Alan Burns, “A Survey of Hard Real-Time Scheduling for Multiprocessor Systems”, © ACM, (2010).
[12] Rekha Kulkarni, Suhas Patil, “A Survey on Improving Performance of Real Time Scheduling for Cloud Systems”, International Journal for Innovative Research in Science and Technology, Volume 1,Issue 7, pp. 171-173, January, 2015.