IOSR Journal of Computer Engineering (IOSR-JCE)
e-ISSN: 2278-0661, p- ISSN: 2278-8727Volume 16, Issue 4, Ver. II (Jul-Aug. 2014), PP 43-53
www.iosrjournals.org

Precedence Constraint Task Scheduling for Multicore Multikernel Architecture

Hitesh P. Daulani[1], Dr. Radhakrishna Naik[2], Pavan S. Wankhade[3]
[1](PG Student, CSE, MIT, Aurangabad, Maharashtra, India)
[2](Prof. and Head, CSE, MIT, Aurangabad, Maharashtra, India)
[3](PG Student, CSE, MIT, Aurangabad, Maharashtra, India)

Abstract: *In real-time systems tasks interact with each others to achieve common system goals, which develops precedence relation among tasks. Each task contributes to the output of system in some manner so contribution of each task should be taken into account. On the other hand processing platform market is booming with multi-core processors and various techniques are coming upfront to utilize processing power of each core efficiently. This paper presents the scheduling algorithm for real-time task set which respects precedence relationship among tasks as well as takes contribution of each task into account while scheduling the task set. Moreover paper targets the multi-core processing platform which has multikernel OS. Multikernel OS differs from single kernel OS in terms of number of kernels, communication among the tasks etc. Proposed scheduling algorithm is analyzed by simulating real-time task set on identical quad core processor and results are compared with PCD scheduling algorithm.*

Keywords: *Admission Controller, Multicore, Multikernel OS, PCF (Performance Contribution Factor), Precedence Constraint, Real-time scheduling*

I. Introduction

In real-time systems it is expected that desired/correct results should be achieved within prescribed deadline. Although now-a-days real-time systems are being more complex and higher accuracy of logical results are expected. In real-time system numerous finite numbers of tasks interact within themselves to achieve common system goals as a whole. This set of tasks collectively also known as task set, interact within themselves to share common resource, intermediate results, which in turn develops precedence relations among the tasks i.e. some task should precede others, so that intermediate results can be passed on to successor tasks which use them and achieve common system goals.

Real-time scheduling concerns with determining the order of the tasks, by which tasks should be executed and system's goals are achieved that too within prescribed deadline. Scheduling policy must also respect various constraints laid on the system.

Scheduling policies are also greatly affected by the processing platforms. Subsequently it can be noted that the scheduling algorithm by Liu and Layland [1] namely Rate Monotonic, Earliest Deadline First are optimal algorithms on uniprocessor processing platform but that is not the case when processing platform changed to multiprocessor. Processing platforms have witnessed huge amount of changes. In early phase of processing platform development which was represented by uniprocessor, have shifted towards multiprocessors, distributed architecture and multi-core processing platform gradually. Recent trend revolves around multi-core processing platform, which has multiple processing units mounted on single chip. Multi-core processor can also be seen as combination of single chip and multiprocessor as multiple processing units are mounted on single chip.

Heterogeneous core will boost the performance of multi-core processors [2]. To handle diversity created by heterogeneity of cores, cores will heavily rely upon operating system. Scheduling policy must exploit ample amount of processing power available with multi-core processors. To exploit the power of processing units, operating system also must be well design. Andrew Baumann et al. [3] suggests new operating system architecture which treats each core as independent and treats machine as network of independent cores. Results by Rami Matarneh [4] proved that multikernel operating system utilizes the core better than single kernel operating system.

In real-time systems, scheduling policy must utilize the processing capability of processing platform and this paper argues contribution of each task in task set should be taken into account during schedulability of whole task set. Less significant tasks must have less priority compare to higher significant tasks. Performance Contribution is the factor which indicates amount of contribution task makes to achieve common system goal. Accomplishment level must be taken into account if task executes partially. Contribution of task can be calculated whether it executes partially or completely in MSS.

In this paper, proposed scheduling policy will schedule the task set taking contribution factor and precedence relation into consideration. Contribution factor means, a task can execute whole of its instruction or few of instructions. Probability of each state is calculated and accomplishment level of each state is assumed so as to get cumulative performance of each task. Priorities are assigned on two factors instead of rate, deadline, laxity as one factor, two factors such as contribution of task in given structure and preceded tasks are considered as parameter for priority. Moreover proposed algorithm deals with identical multi-core platform managed by multikernel. Algorithm has two major phases as

1. Admission Controller

In this phase Admission Controller does analysis of tasks belonging task set and subsequently tasks are classified into different classes on the basis of their contribution to the output of system and preceded tasks. This phase is static as the classification is done pre-runtime.

2. Scheduling Strategy

In this phase tasks are assigned to processing unit for their execution. This phase is dynamic as decision regarding assignment of task to eligible core is taken while task set is executing depending upon PCF and preceded tasks if any.

The number of kernels is equal to number of cores. One kernel acts as master kernel which manages $core_0$ and rest of kernels act as slave kernels. One kernel manages one core [4]. As mention above proposed algorithm has two phases, admission controlling and scheduling strategy. Admission controller does classification which does all the related computations on $core_0$. Therefore no task is scheduled at $core_0$. In the end of paper performance is evaluated by simulating proposed algorithm on identical quad core processor and comparing results with PCD algorithm [5] which also considers contribution of each task and respects the precedence constraints.

II. Related Work

Real-time systems have prime constraint of deadline on them, which differs them from general purpose system. Therefore there must be proper order in which tasks execute so as to respect the deadline constraints. Lot of work has been done by researchers and academicians right from the first result publish in context of real-time scheduling by Liu and Layland [1] in the year 1973. Optimal fixed priority algorithm has been described in which one algorithm statically assigns priorities on the basis of rate known as Rate Monotonic algorithm, whereas another algorithm assigns priorities dynamically on the basis of deadline known as Earliest Deadline First [1]. It can be noted in fixed priority non-preemptive scheduling algorithms, assignments of priorities are assigned on the basis of certain parameter.

2.1 Multi-core Processor as Processing Platform

Since the first microprocessor which was 4 bit by Intel in 1971[6, 7], Paradigms in processing platforms have been shifted over time. Uniprocessors gave place to multiprocessors, distributed processors subsequently. Recent trends revolve around multi-core processors. If Moore's law is to believed performance will grow every 18 months since transistors on chips will double every 18 months [8]. As multi-core processors are in trend in market, manufactures are focusing on boosting the performance by the various factors like speed of processing units, mounting more number of processing units known as core on single chip. 100-core processor, TILE-Gx100 has been announced by Tilera [9].

2.2 Multikernel OS for Multi-core Processing Platform

Multikernel OS is new approach that has more than one kernel managing the functionality of OS. Architecture model by Rami Matarneh [4] elaborates the multikernel OS having kernels equal to number of cores; therefore each core will have one dedicated kernel. One kernel among the kernel will act as master kernel and rest of kernel will act as slave kernel. Master kernel is evoked first and it creates the slave kernels. Master kernel is only authorize to deal with system resources and if slave kernel needs the system resource it can use it via master kernel i.e. it requests the master kernel for using system resources. Also the inter task communication among the tasks executing on different cores needs intervention of master kernel. Tasks executing on same core does not need intervention of master kernel. Results proves that performance of multikernel OS is better than single kernel OS.

Architecture model by Andrew Baumann et al. [3] elaborates that OS architecture treats machine as network of independent cores. The architecture is built on perception that system can also be model as distributed system. Inter task communications among tasks is done by message passing. Result indicates message passing is better than other approaches such as share memory etc.

Barrelfish is experimental OS built upon architecture of operating system having multikernels [10].

2.3 Real-time Scheduling on Multi-core Processor Platform

Multi-core processor can be seen as multiprocessor on chip. Scheduling algorithms for multi-core processing platforms are classified into heterogeneous and identical categories on the grounds of the cores used in multi-core system [11]. Algorithms can again be classified under global scheduling algorithm and partition scheduling algorithm. In global scheduling algorithm single ready queue is maintained and task can be migrated or if needed can be scheduled on single core [11]. Even if task is preempted on one core can resume its execution on other core. On the other hand in partition scheduling algorithms tasks are partitioned into various partitions based on certain parameter and assigned to particular core. In partition scheduling algorithms once task assigned to core migration to another core is forbidden. Although another category can be consider for algorithm which falls between global scheduling algorithm and partition scheduling algorithm, which inherits some characteristics of global scheduling algorithms and some characteristics of partition scheduling algorithms.

Different approaches for scheduling on multi-core have been followed to achieve goals along with respecting the constraints on system such as reducing the power consumption, increasing processor utilization, fairness. Various parameters can be focused to design scheduling policy such as cache miss, context switch [12] to accomplishing goals.

2.4 Performance Contribution of Task

In some real-time systems, each task contributes to the final logical results. In MSS it may happen that some task may not perform to their fullest, but contributes up to certain extent. So contribution of task must be taken into account. In [5] contribution of each task with respect to other tasks in task set has been taken into account. Tasks are classified into four categories based on two parameter namely PCF and deadline. Tasks are again divided into two sections of mandatory which represent the compulsory executable part of task and other is optional which represents optional part of task. Task set is scheduled by PCD algorithm on basis of PCF which respects precedence constraints and deadlines imposed on task.

PCF of each task is calculated as

$$E_G = \sum_{k=0}^{k} p_k G_k \tag{1}$$

Where,

E_G= Expected MSS performance.

G_k = Performance rate at state k of MSS

p_k= Probability of system being in state k.

The probability that system reaches particular state is calculated by

$$p_k = \frac{L_{in}}{L_t} \tag{2}$$

Where,

L_{in}= Total number of messages task receives from other tasks.

L_t= Total of number messages associated with task.

Consider if task has one preceded task, then the task become dependable on preceded task for its execution. Probability of task execution having one preceded task depends upon execution of preceded task and accomplishment level of preceded task. For example if task t_2 is preceded by task t_1, then probability of execution of task t_2 depends on execution of task t_1, same can be calculated by using conditional probability as

$$p(t_2 \mid t_1) = \frac{n(t_2 \bigcap t_1)}{n(t_1)} \tag{3}$$

Equation (3) can be read as probability of execution of task t_2 given that task t_1 executes.

Consider if more than one tasks precedes certain task then the probability of execution depends upon all the tasks which precedes the certain task. Then the probability of execution of certain task which depends on more than one of tasks is calculated by Bayes rule. For example consider that task t_1, task t_2 precedes task t_3 then probability of execution of task t_3 can be calculated by Bayes rule as

$$p(t_3) = p(t_1)p(t_3|t_1) + p(t_2)p(t_3 \mid t_2) \tag{4}$$

Accomplishment level of system at state k is represented by G_k in equation (1) is assumed based on $G_k L_{in}$.

Putting the values of p_k and G_k in equation (1) gives the performance factor in the MSS.

2.5 Modeling Real-time systems

Modeling is important part of system engineering. Modeling helps to understand complex problem and their solution [13]. As the popular proverb says "Stitch in time saves nine". Modeling the system in design phase saves the extra time spent in changes in actual system which is engineered instead of modeling the system [14]. Systems are modeled before actual development of the system, to verify that system fulfils requirements for which, system is to be engineered. Therefore modeling real-time system can greatly help to visualize whether system satisfies the requirement in timely manner. Modeling real-time system can be focused on two parameter correctness and efficiency when processing platform like multi-core, provides parallelism [15]. Modeling inter-task communication mechanisms also have been studied in literature [16]. Systems engineered by modeling the system in design phase refer to model-driven-engineering. MDE was introduced by Kent [17]. Different approaches to model real-time systems such as component based model, timed based model can be followed [14]. In context of scheduling, component level based real-time systems have been proposed which have component as set of real-time task and real-time scheduler whereas components communicate via RPC [18]. Various tools can be used to design system model and verification like uppaal, smulinks etc.

2.6 Conclusion from Related Work

Uniprocessor architecture issues are outdated upto some extent whereas multi-core processors are booming in processor market. Efforts are taken to increase more and more performance of processor by increasing core count, speed of individual cores etc. This architectures need to exploit to solve complex problem like precedence constraints, contribution of each task into output of system. On the other hand numbers of approaches have been design to utilize processing power of processors; one of such approach is using powerful OS that has dedicated kernel for each core. Also scheduling policy must schedule the task set in proper way so as all the constraints imposed on system must be respected. Real-time scheduling has the prime constraint of deadline along with other constraint while scheduling task set. Tasks are scheduled based on certain parameters such as Deadline, Rate of task, Processor Utilization etc. Performance Contribution of task to the performance of system could be such parameter.

III. Related Terminology

3.1 Hybrid Algorithm

Proposed scheduling algorithm is hybrid scheduling algorithm as it does not belong to global scheduling algorithm category which can migrate task while execution i.e. task can be preempted on one processing unit and resume on another processing unit. Proposed algorithm neither belongs to partition scheduling algorithm category which statically divides the task set into different partition.

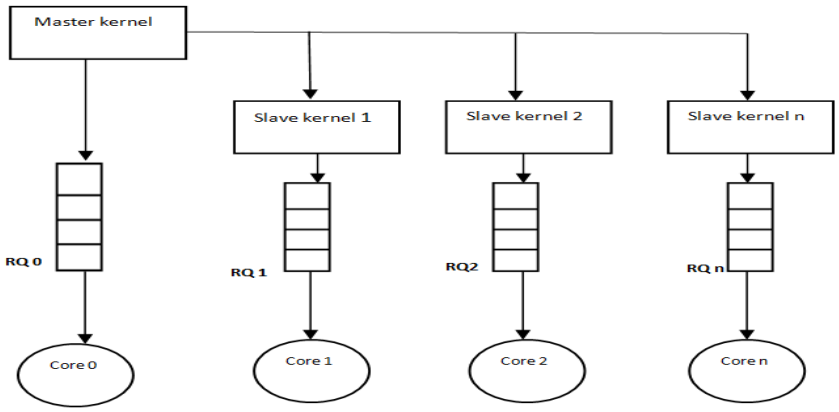

Fig. 1. Hybrid Scheduling Assignment

In proposed algorithm, ready tasks which can be executed, arrives at global queue which is also ready queue of $core_0$ which is managed by master kernel; Admission Controller classifies them on the basis of PCF and precedence relation. At the run time depending upon preceded tasks decision is taken at which core the particular task will be executed.

3.2 Constraints

In real-time systems deadline is de facto constraint implied on scheduling algorithm. As tasks interact among themselves to generate the logical result of system it imposes precedence relation constraints.

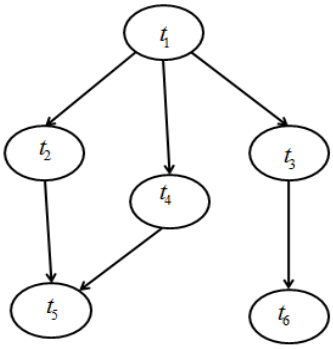

Fig. 2. Precedence Relations among Tasks

Respecting precedence constraints among task set refers to successor tasks cannot be executed until and unless their preceded tasks have been executed. Precedence among tasks can be represented by Directed Acyclic Graph (DAG) as

$$(P = V, E) \tag{5}$$

Where,

P = DAG representing precedence relation among tasks

V = Set of vertex in graph representing task set.

E = Set of edges in graph representing the messages among tasks as well as precedence relations among tasks.

Edges towards the vertex represents the precedence on the task, whereas edges outwards from vertex represents that vertex is predecessor for other tasks. Proposed algorithm schedules task set respecting deadline as well as precedence constraints laid on task.

3.3 Intercommunication among Tasks

Rami Matarneh [4] describes design architecture of OS which makes master kernel manager of slave kernel and only master kernel is authorized to deal with system resource which will certainly maintain system integrity. Communication among tasks executing on different core requires intervention of master kernel. Andrew Baumann et al. [3] describe design architecture of OS uses message passing for communication among tasks. This paper suggests that message passing should be done through intervention of master kernel and master kernel should only be authorized to deal with system resources to maintain system integrity. However proposed work is kept restricted from dealing to other approaches of communication among tasks and assumes task communications is done by message passing.

3.4 System Model

This paper focuses on periodic task represented by t_i which belongs to task set such that $t_i \in T$. Task t_i is characterized by (P_i, D_i, C_i, Z_i) where P_i represents the minimum time units after which task can be invoked again. D_i represents absolute deadline for task t_i on or before which task should get executed, C_i is execution time and Z_i is number of preceded task for task t_i. This paper considers ($P_i = D_i$) and ($C_i \leq D_i$).

IV. Praposed Work

This paper propose scheduling algorithm for periodic tasks, which can be scheduled on homogeneous multi-core processors having multikenel OS in non-preemptive manner. Proposed algorithm is hybrid algorithm which classifies the tasks of task set in two classes and determines strategy for allocating task to eligible processing unit i.e. core on multi-core processor . In the task set, tasks also have precedence constraints imposed which means successor task is executed after its preceded tasks are executed. The general view of scheduling policy is as follows

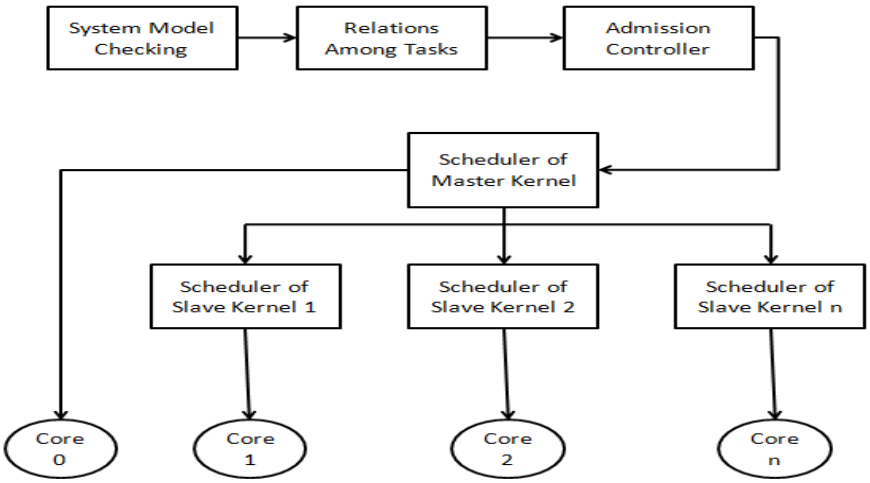

Fig. 3. Block Diagram

The working of algorithm is as follows:
Step 1: System modeling is done in EVENT STUDIO and checked against requirement. The technique was used for model checking in [5].
Step 2: Task collaboration diagram is generated, which is further evaluated and simplified.
Step 3: Task set arrives at the ready queue of $core_0$ which is global queue and managed by master kernel.
Step 4: Admission Controller does analysis of tasks belonging to task set which includes PCF and number of preceded task
Step 5: Admission controller classifies tasks of task set into two different classes on the basis of PCF and precedence relations.
Step 6: Tasks from class c_1 are executed as task do not have any precedence.
Step 7: Tasks from class c_2 are executed if all preceded tasks of task have been executed.

i. If task has only one preceded task, then the core where preceded task had executed is checked and if it is idle then, task is scheduled to be executed over the same core else it is scheduled to execute on another core which is idle. This approach is used to minimize the communication delay, and utilize the result which can be locally used on core where preceded task was executed.
ii. If task has more than one task preceding, then the core where preceded task that finishes its execution last is checked whether it is idle or not, if it is idle then task is scheduled on the same core else task is schedule to execute on another core that is idle.

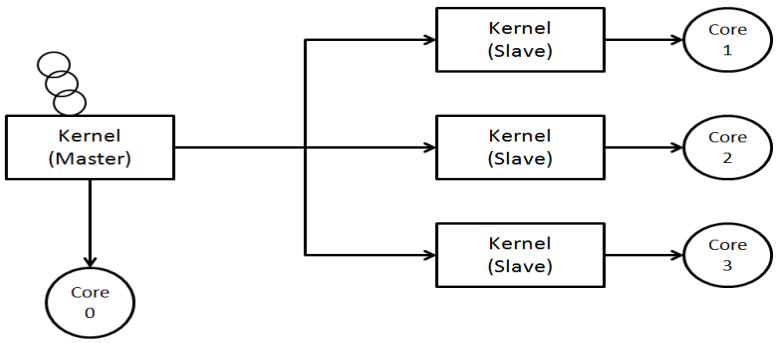

Fig. 4. Logical View of Scheduling Policy

As shown in fig. 4 tasks set arrives at global ready queue which is managed by master kernel. Admission Controller classifies task into different classes. Depending upon the strategy, master kernel allocates task to the core managed by slave kernel for execution. Proposed algorithm has two major phases namely Admission Controller and Scheduling Strategy.

4.1 Admission Controller

Admission Controller does classification of tasks statically which mean that it is done before run time. Classification is done on the basis of two parameter PCF and precedence relation. This classifies the tasks into two different categories depending on their PCF factor and preceded tasks.

Let T be the task set having finite number of task and C be set of classes to classify the tasks.
$T = \{t_1, t_2, t_3 \ldots t_n\}$
$C = \{c_1, c_2\}$

i. Calculate the PCF for each task in the task set and check the precedence relation.

This is done using analysis of task interaction collaboration diagram which suggest that how many number of message have been passed to succeeded task. Let N be the number of messages passed by preceded tasks. If one of these messages is failed then accomplishment level is going to reduce. Very such state is taken into account so as to calculate PCF

Classification of tasks can be done as follows

ii. $\forall$ task $t_i \in T$

$$c_1 = \{t_i \mid t_i.PCF = 0 \cap t_i.preceded\ tasks = NULL\} \quad (6)$$

and

$$c_2 = \{t_i \mid t_i.PCF \geq 0 \cap t_i.preceded\ tasks \neq NULL\} \quad (7)$$

4.2 Scheduling strategy

Assignment of task to the core is done at run time depending upon precedence of task and availability of processing unit i.e. core.

Assumptions, Notations and pseudo code

Assumptions

- Only one instance of each task is executed per period in non-preemptive manner.
- Task set can be represented by Directed Acyclic Graph regarding the precedence relation, which means there is no precedence constraint by successor task on predecessor tasks.
- All cores are homogeneous i.e. identical.
- Intercommunication among tasks is done by message passing.
- $Core_0$ is managed by master kernel and all the computations regarding classification by admission controller, other systems arithmetic computations are done at $core_0$. Therefore no task executes at $core_0$.
- All the tasks are single threaded.

Notations

t_i= i^{th} Task in Class.
t_n= Last Task in Class
$core_l$= l^{th} Core among all Cores.
pre = Number_of_preceded_task.
t_k= Preceded Task.
setflag_c_l = Boolean Value for l^{th} Core whether Busy or Idle.
setpflag = Boolean Value for all Preceded Task Executed.

Pseudo code

1. for $\forall$ task $t_i \in$ class c_1
2. i=0 and l=1
3. for $core_l$ to $core_n$
4. setflag_$c_l = 0$
5. execute ($t_i \rightarrow core_l$)
6. setflag_ $c_l = 1$
7. endfor
8. if $t_i \neq t_n$ then
9. Repeat statement 3-7
10. endfor
11. for $\forall$ task $t_i \in$ class c_2
12. i=k=0 and l=1
13. setpflag = 0
14. for k to pre
15. if all preceded tasks t_k are execute1 then
16. setpflag = 1
17. endfor
18. if setpflag = 1 then
19. Check preceded task t_k with highest response time and executed at $core_l$
20. if setflag_c_l = 0 and setpflag =1 then

21. execute ($t_i \rightarrow core_l$)
22. setfalg_c_l = 1
23. endif
24. else
25. $core_{l+1}$
26. Repeat statement 20-27
27. endelse
28. if $t_i \neq t_n$ then
29. Repeat statement 13-30
30. endfor

V. Performance Evaluation

Real-time systems should give correct results in prescribed time known as deadline. In context of scheduling, task should execute on or before its deadline. If execution of tasks within task set has to be done on or before deadline it is important that task should be assigned to processing unit in timely manner. Therefore it can be noted that waiting time must be minimum. Also it is equally important that task should finish its execution on or before deadline but not after deadline. Turnaround time parameter can be use to analyze that how much time task takes to execute.

Waiting time: Amount of time task spent in ready queue, waiting for the resources allocation, I/O manipulations and intercommunication with other tasks. For the sake of simplicity this paper assumes that task spent negligible waiting time in resource allocation, I/O manipulations and intercommunication with other tasks.

Turnaround time: Turnaround time is the time gap between ready time to completion time. In non-preemptive context turnaround time can also be measure as summation of waiting time and execution time of task.

Simulations of proposed scheduling policy are done on identical quad-core. $Core_0$ is managed by master kernel as shown in fig. 4. All the computation regarding Admission Controller, system related arithmetic computations are done at $core_0$, so no task will execute at $core_0$. All tasks in task set will be scheduled on $core_1$, $core_2$, $core_3$ depending upon their availability and scheduling policy.

Case study1:

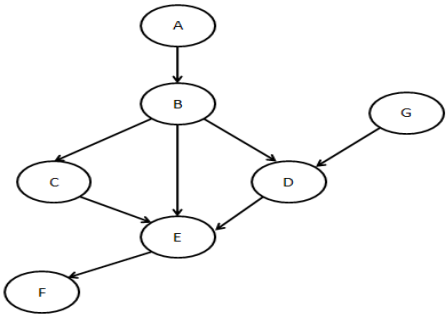

Fig. 5. Task Set 1 along with Precedence Relations

```
task 'a' ---> class 1
task 'b' ---> class 2
task 'g' ---> class 1
task 'c' ---> class 2
task 'd' ---> class 2
task 'e' ---> class 2
task 'f' ---> class 2


Task    excution time   deadline        PCF             class

 a              5         5            0.000           class 1
 b              3         6            0.900           class 2
 g              8         18           0.000           class 1
 c              2         10           0.225           class 2
 d              4         22           0.038           class 2
 e              1         23           0.481           class 2
 f              5         28           0.225           class 2
```

Fig. 6. Classifications of Tasks for Case Study 1

Fig. 5 shows the tasks and the precedence relationships among them. Fig. 6 shows task set and classification of the tasks on the basis of PCF and preceded task which is done statically.

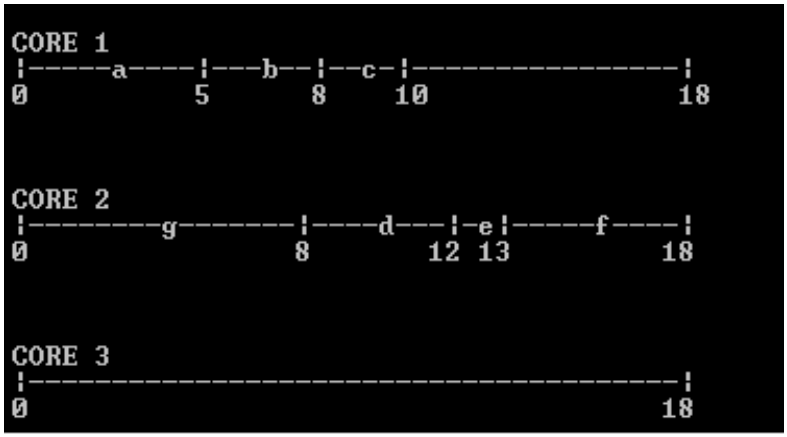

Fig. 7. Task Allocation for Case Study 1

Fig. 7 shows task allocation for case study 1 which have task set comprising seven tasks. Each task is allocated to processing unit according to scheduling strategy. Task is allocated in non-preemptive manner to processing unit for the time units needed to get task executed. It can be seen that proposed algorithm respects the constraints of precedence relation as well as deadline for each task.

Case study 2:

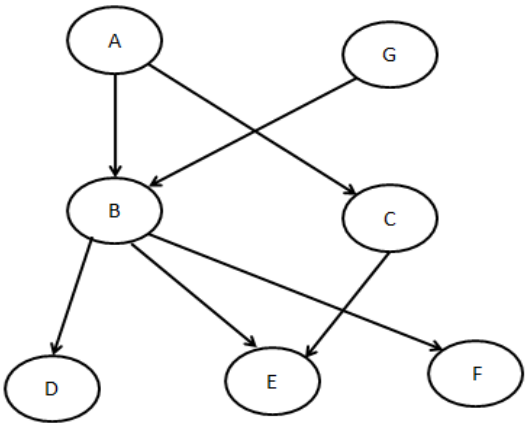

Fig. 8. Task Set 2 along with Precedence Relations

Fig. 8 shows the precedence relation among task set for case study 2. Fig. 9 shows the tasks of task set 2 and classification of the tasks on the basis of PCF and Preceded tasks.

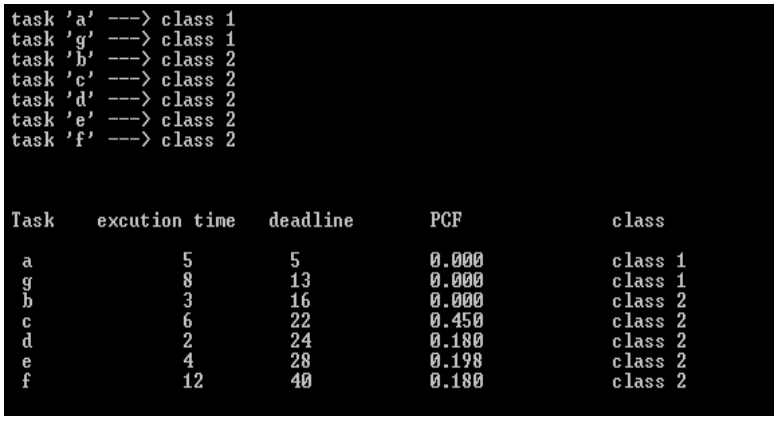

Fig. 9. Classification of Tasks for Case Study 2

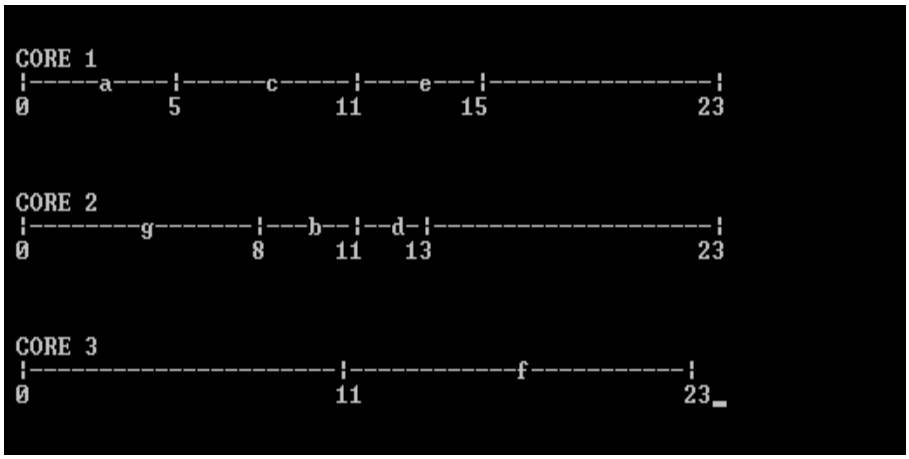

Fig. 10. Task Allocation for Case Study 2

Fig. 10 shows task allocation for case study 2 which have task set comprising seven tasks.

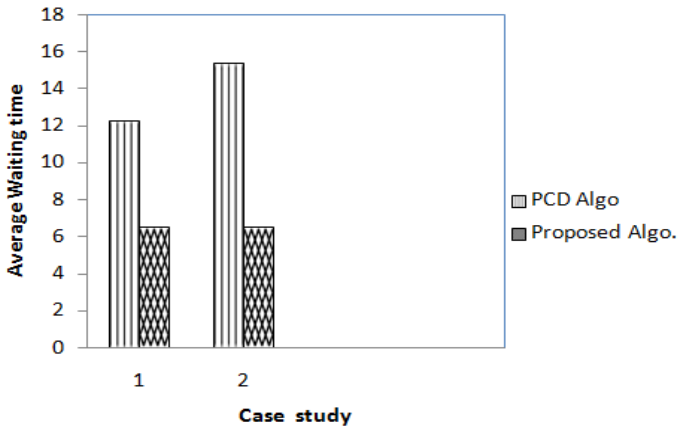

Fig. 11. Average Waiting Time Comparison

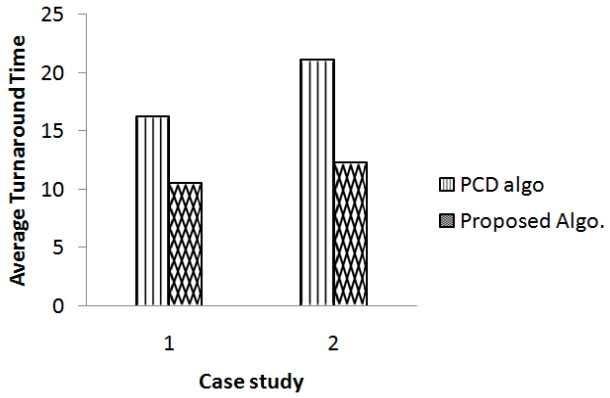

Fig. 12. Average Turnaround Time Comparison

As shown in fig. 11 and fig. 12 both average waiting time and average turnaround time of proposed scheduling policy is less than PCD algorithm for both task sets.

VI. Future Scope

Although performance is increased for the task set respecting precedence constraints, more performance could be gain if tasks can be executed in parallel manner provided more than one core is idle. Moreover heterogeneity in cores should be taken into account and scheduling policy must be modified so that tasks are assigned to different cores having different speeds, frequency. Depending upon performance contribution of task in output of system and capacity of cores, scheduling policy must be designed. Another

approach could be of designing scheduling policy targeted on the platform having clusters of core that means each cluster having homogeneous cores but the clusters are heterogeneous.

VII. Conclusion

Problem of precedence constraint tasks can be solved more effectively with this approach. This paper proposes new scheduling policy which addresses precedence constraint task scheduling considering precedence relation and performance contribution of each task into output of the system on multikernel multi-core processing platform which has more than one kernel. Scheduling policy is simulated for homogeneous quad core of which one core is reserved for computations by admission controller. Rest three cores are being used for scheduling the task set. However same problem of precedence constraint tasks has been addressed by PCD algorithm which is indeed for uniprocessors architecture. Results are compared with PCD algorithm, it is observed that average turnaround time has been improved further and average waiting time has been decreased because of use of multi-core architecture.

References

[1] C. L. Liu and James W. Layland, "Scheduling algorithm for multiprogramming in a hard Real-time environment", Journal of ACM 20, pp. 46-61, 1973
[2] Rakesh Kumar, Norman P.Jouppi and Parthasarathy Ranganathan "Heterogeneous Chip Multiprocessors", IEEE Computer Society pp32-38, .2005
[3] Andrew Baumann et al. "The Multikernel: A new OS architecture for scalable multicore systems", SOSP '09: 22nd ACM SIGOPS Symposium on Operating systems principles, pp. 29–44, 2009.
[4] Rami Matarneh "Multi Microkernel Operating Systems for Multicore Processors", Journal of Computer Science pp 493-500, 2009
[5] Radhakrishna Naik, R.R.Manthalkar, "Performance Contribution Based Scheduling Framework for Precedence Constraint tasks in Real-time system", International Journal of Computer Science and Engineering, Vol. 3 (2), 2011, 664-675, 2011
[6] B. Schauer, "Multicore processors-a necessity", [Online] Available:http://www.csa.com/discoveryguides/multicore/review.pdf
[7] Pawel Gepner, Michal F. Kowalik "Multicore Processors: New Way to Achieve High System Performance", Proceedings of the International Symposium on Parallel Computing in Electrical Engineering, 2006
[8] Gordon E. Moore, "Cramming More Components onto Integrated Circuits" Proceedings of the IEEE, VOL. 86, NO. 1, pp 82-85, January 1998
[9] Coming soon tile-gx100 the first 100 cores processors in the world, [Online].Available: http://internalcomputer.com/coming-soon tilegx100-the-first-100-cores-rocessorin-the-world. Computer, Feb 2011
[10] Andrew Baumann, "Your computer is already a distributed system. Why isn't your OS?", 12th Workshop on Hot Topics in Operating Systems, Monte Verità, Switzerland, 2009.
[11] Fan Ming, "Real-Time Scheduling of Embedded Applications on Multicore Platforms", FIU Electronic Theses and Dissertations. Paper 1243. 2014.
[12] Ajoy k. Datta, Rajesh Patel, "CPU Scheduling for Power/energy Management on Multicore Processors Using Cache Miss and Context Switch Data", IEEE Transactions on parallel and distributed systems volume.25 pp1190-1199,2014
[13] Bran Selic "The Pragmatics of Model-Driven Development The Pragmatics of Model-Driven Development", IEEE Computer Society IEEE software pp19-25, 2003.
[14] Joseph.Sitakis, "Modeling real time system-challenges and work direction", proceeding of int. conference on embedded software (EMSOFT 2001), Springer, LNOS 2211, 2001.
[15] Ahlem Triki, Jacques Combaz, "Model-Based Implementation of Parallel Real-Time Systems", Verimag Research Report TR-2013-11, 2013.
[16] N. Scaife and P. Caspi, " Integrating model-based design and preemptive scheduling in mixed time and event-triggered systems", Verimag Technical Report TR-2004-12, 2004.
[17] Stuart Kent, "Model Driven Engineering", in IFM 2002, volume 2335 of LNCS. Springer-Verlag, 2002.
[18] Jose L. Lorente, Giuseppe Lipari, Enrico Bini, "A Hierarchical Scheduling Model for Component-Based Real-Time Systems", IEEE, 2006.