A Variant of FPZL Algorithm

Komal Bhalotiya[1], Dr. D. V. Padole[2] and Prof. Radhakrishna Naik[3]

[1] Computer science and Engineering, Nagpur University, G. H. Raisoni College of Engineering Nagpur, Maharashtra, India

[2] Electronics Engineering, Nagpur University, G. H. Raisoni College of Engineering Nagpur, Maharashtra, India

[3] University of Pune, G. H. Raisoni College of Engineering Nagpur, Maharashtra, India

Abstract

Fixed priority until Zero Laxity (FPZL) algorithm is the minimally dynamic scheduling algorithm. In FPZL, fixed priorities are assigned to the jobs and are scheduled accordingly until the state of zero laxity is reached and it does not follow any priority order. FPZL concentrates on scheduling more jobs as far as possible. The algorithm which is proposed in this paper is the variant of FPZL which employs the priority order. Instead of giving up processing of some tasks completely, this algorithm completes some portion of the tasks and at the same time if any zero laxity task arrives it takes that task also into consideration and schedules accordingly. In this way, this algorithm tries to process more jobs as compared to the FPZL algorithm and increases the CPU Utilization.

Keywords: *Real Time Operating System, Multi-Processor, Scheduling Algorithm.*

1. Introduction

Real time Multiprocessor systems are now common place. For scheduling tasks on the Multiprocessors there have been two approaches, partitioning and global scheduling. In global scheduling the tasks are stored on a single priority ordered queue; The global scheduler selects for execution the highest priority tasks from this queue. In partitioning, each task is assigned to a single processor on which each of its job will execute and processors are scheduled independently of each other. There is one more approach which is considered earlier as a "middle" approach in addition to the above approaches. In this approach each job is assigned to a single processor while a task is allowed to migrate. You can consider it as inter-processor task migration is allowed but at job boundaries. One taxonomy is presented earlier in which two dimensions were given for ranking the scheduling schemes. One dimension is the complexity of the priority scheme and other is the degree of migration allowed. Along with first dimension the scheduling algorithms are classified according their task priorities which can be static, dynamic, and fully dynamic. In static one the priorities will be fixed. In dynamic the priorities dynamic but fixed within a job and other one is fully dynamic where tasks will be having fully dynamic priorities without any job boundaries. Dynamic scheduling can be preemptive or Non-preemptive. The second dimension is degree of migration allowed. Again this dimension is categorized as no migration, migration allowed but only at job boundaries and third one is unrestricted migration. In unrestricted migration jobs are also allowed to migrate [9].

In aggregation, the following are the performance parameters on which basis algorithms can be compared.

- CPU Utilization
- Task Migration
- Number of pre-emptions
- Success Ratio
- High Throughput
- Resource utilization
- Effectiveness

Parallelism and Urgency (Deadline Satisfaction) are the two factors which were taken into consideration while designing scheduling algorithms for Real time Multiprocessor system. Being a Real time Multiprocessor system, sole focus on either deadline satisfaction or parallelism is not sufficient. Hence, One of the simple but effective ways to consider both urgency and parallelism is to assign the highest priority to any zero-laxity task, where laxity of a task at any time is defined as remaining time to deadline minus the amount of remaining execution. We denote this policy as the *ZL (Zero Laxity) policy [12]*.So, there is need for such an algorithm which considers both the factors.

In 2011, Robert Davis and Alan Burn has presented one algorithm named "Fixed Priority until Zero Laxity (FPZL)". FPZL is a minimally dynamic global scheduling algorithm. Under minimally dynamic also it follows pre-emptive scheduling. FPZL algorithm does not follow any priority order. The aim behind FPZL algorithm was to schedule as many tasks as far as possible.

In this paper, we present a dynamic global scheduling algorithm which is the variant of FPZL algorithm. This algorithm also follows the pre-emptive scheduling scheme. In this algorithm we have employed the assignment of priorities. In addition to the employment of assigning priorities we have added the concept of processing some portions of task rather than giving up tasks completely unscheduled. This is because; there are many situations where some portions of tasks are very important which is to be scheduled on a priority basis than the other portions of the task. In the year this concept was proposed where the task is logically divided into two subtasks, mandatory and optional. Using this concept the mandatory portion of every task will be executed first and later on the optional portions will be executed.

This paper shows the comparison between the modeled Fixed Priority until Zero Laxity algorithm and its Variant which employs the Dynamic priorities with the zero Laxity concepts in terms of the CPU Utilization, taskset schedulability, and context switches. With reference to the performance parameters presented above we can consider the taskset schedulability as a success Ratio also. As Success ratio measures the Number of tasks successfully scheduled to the Total Number of tasks arrived at the scheduler.

The remainder of the paper is organized as follows: Section II describes the Review of Literature. Section III describes the proposed framework. Section IV shows the Results Analysis. Section V describes the Conclusion and Future Work.

2. Literature Survey

Jinkyu Lee, Arvind easwaran, and Insik Shin[12] presented the First ZL schedulability test for any work conserving scheduling algorithm that employs this policy. In this paper the authors have investigated the ZL policy and then investigated the characteristics of LLF scheduling which also employs the ZL policy and then they have derived some LLF specific schedulability tests on Multiprocessors. For conducting this schedulability test they have assumed that the system comprised of m identical unit capacity processors which has restricted the system utilization.

Robert Davis and Alan Burn [3] has presented another zero laxity based scheduling algorithm in which the priority of a job can change at most once during its execution and hence bounding the number of pre-emptions. The key idea behind FPZL is that the jobs are scheduled according to the fixed priorities assigned until a zero laxity state is reached. Zero Laxity state is the state where remaining execution time of a job is equal to the time to its deadline. This kind of Zero Laxity job will be missing its deadline unless it executes continually until completion. In FPZL such zero laxity jobs gets the higher priority. As the priority of a job changes at most once during its execution this algorithm is considered as a minimally dynamic scheduling algorithm and hence it does not follow any priority order.

3. Proposed Framework

For designing this algorithm, we have taken the FPZL Algorithm as a basic source. We have modeled this algorithm first and then implemented the proposed algorithm. After that we have compared the performance of the modeled algorithm with proposed one in terms of 3 performance parameters namely CPU Utilization, Context switches and the taskset schedulability.

In the modeled FPZL algorithm, sometimes it happens that some task sets are not schedulable. In many situations it may happen that the tasks which were not scheduled in the modeled algorithm might have some critical portion which should be executed. Hence, for this reason, the modeled algorithm is altered to schedule as many tasks as possible. For that purpose in this algorithm the concept of mandatory and optional task is included. It means that the tasks will be logically divided into two subparts: Mandatory and optional. By including the concept of Mandatory and optional tasks we can complete execution of some portions of every task rather than completely giving up the processing of some tasks [4].

3.1 Basic Assumptions of the system

1. All tasks are assumed as Independent tasks. It means that tasks can only be blocked when there is contention of processors.
2. All tasks are considered as sporadic tasks where each job of a task may arrive at any time once a period has elapsed.
3. There will be interference by only other zero laxity tasks.
4. Tasksets are considered with constrained deadlines. Constrained deadlines means deadline will be either less than or equal to its period.
5. Whenever a task starts to execute, it will not voluntarily suspend itself.
6. All processors are assumed as homogeneous Processors.

7. Period of the task is equal to its Deadline.
9. CPU Utilization of a task is computed using following formula:

Utilization of a task = Computation time of that task / Period of that task

3.2 Algorithm

Step-1: a) Take input of tasks containing computation time, mandatory portion (in %), Optional portion (in %), deadline, period.

b) Input processor parameters including Number of processors, period and capacity.

Step-2: Priorities are assigned to all the tasks including mandatory and optional tasks according to the Utilization. The task which will utilize the system more is assigned highest priority and accordingly the priorities are assigned. Schedule tasks according to these assigned priorities.

Step-3: The mandatory tasks will be executed first and then the optional tasks will be executed.

Step-4: While executing mandatory tasks if any zero laxity task arrives, then give that zero laxity task a higher priority than the task which was currently executing and add that pre-empted task to the ready queue and allocate the processor to the zero laxity tasks till it completely gets executed.

Step-5: Repeat step 3,4 for optional tasks also.

Step-6: Schedule tasks from ready queue.

Step-7: Follow steps 3, 4, 5, 6.

4. Result Analysis

The following result shows the case studies of both the algorithms and then a comparative study including some performance parameters.

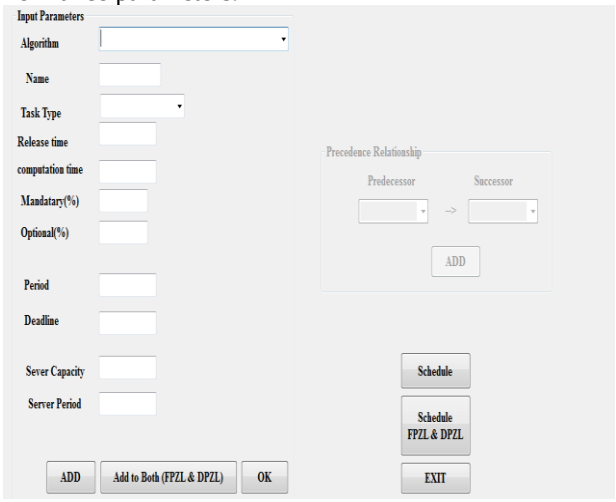

Fig. 1 Initial screen when the application starts

Abbreviations used:

Ui = CPU Utilization
Ci = Computation time of a task
Pi = Period of a task
Di = Deadline

CASE STUDY 1:

Table 1: Initial Task set

Ti	*Ci*	*Pi*	*Di*	*Mandatory (%)*	*Optional (%)*
T1	4	10	10	50	50
T2	6	15	15	40	60
T3	5	12	12	60	40
T4	8	20	20	70	30

Table 1 shows the initial taskset. The complete set is applicable for the proposed algorithm which is the variant of FPZL scheduling algorithm. Mandatory and optional fields are not applicable for the Modelled FPZL algorithm.

Table 2 shows the Input parameters of the processors. Input parameters include the number of Processors, capacity of processor and the period of processor. Capacity and period of processor will be same for all the processors, as all processors are considered as homogeneous processors.

Table 2: Input parameters of the processors

Input Parameter	*Value*
Capacity	3
Period	25
Number of processors	2

Now the priority will be assigned to the entire Mandatory first and then the optional tasks. In a similar way the mandatory tasks will be executed first and then the optional one.

In the modelled FPZL algorithm, the mandatory and optional portions are not taken into consideration. So, the complete tasks will be taken here for scheduling. Here the task which will utilize the system more is assigned highest priority and accordingly the priorities are assigned and all the tasks are scheduled according to their unique assigned priority.

While execution the task T2 reaches to the state of zero laxity. Hence the highest priority is assigned to it and this task is executed completely. As per the provided processor capacity and period, large portion of tasks T3, T4 and

small portion of task T1 remained un-schedulable. To schedule this remaining portions of tasks the extra processor time is required which is in this case comes to 14. Hence CPU utilized in this case is 100% and context switches are 7.

When above tasks are scheduled using the Modified version of FPZL algorithm, Mandatory and optional fields are taken into consideration. From the percentage given in these fields first of all the time of mandatory and optional tasks are computed. Then according to the utilization, under mandatory tasks the task which will utilize the system more is assigned highest priority and so on. The same will happen in case of optional tasks also.

Let's see how the tasks are selected for scheduling. In case of Mandatory tasks, Task T4 is having highest Utilization hence it will be selected first for scheduling. After that Task T3 is having next highest priority hence it will be selected next for execution. Then T2 will be selected next and lastly Task T1 in mandatory case will be selected for scheduling.

In case of Mandatory tasks, T2 is having highest utilization. So, it will be selected first for execution. Then T4 is having next highest Utilization so, it will be selected next for execution. Among T1 and T3, T3 is having highest utilization, so it will be selected next for execution. Lastly optional portion of task T1 will be executing.

In modelled FPZL algorithm large portions of tasks T3, T4 and small portion of task T1 were remained un-schedulable. But when scheduled with the modified version of FPZL it is noted that more tasks are executed. In this algorithm Mandatory tasks T2 and T4 are completely scheduled. In a similar way optional portions of tasks T2 and T4 are completely get scheduled. small portion of taskT1 and large portion of task T3 remained un-schedulable. In addition, a small portion of optional tasks remains un-schedulable. The following table shows the comparison of above case study between these two algorithms.

Table 3: Comparative study between proposed two algorithms

Algorithm/Parameter	***Schedulability***	***Success Ratio***	***CPU Utilization (%)***	***Context switches***
Modelled FPZL Algorithm	Average	0.25	100%	7
Modified FPZL Algorithm	High	0.50	100%	13

From the above comparative study between the proposed algorithms, it is clear that the proposed algorithm is good in terms of schedulability, Success Ratio, and CPU Utilization. But the context switching rate is very high in comparison with the modelled one.

CASE STUDY 2:

Consider the following taskset with 5 tasks and 3 processors. The following table shows the initial taskset for both the algorithms.

Table 4: Initial Task set

Ti	***Ci***	***Pi***	***Di***	***Mandatory (%)***	***Optional (%)***
T1	3	8	8	30	70
T2	5	12	12	50	50
T3	7	10	10	60	40
T4	9	20	20	80	20
T5	11	14	14	65	35

The following table shows the input parameters of the processors.

Table 5: Input parameters of the processors

Input Parameter	***Value***
Capacity	3
Period	21
Number of processors	3

In the above case, if scheduled with modelled FPZL algorithm then all the tasks were completely scheduled except task T2. For processing task T2 more processor capacity is required or there is need to increase the processor.

If this taskset is scheduled with the proposed algorithm then all the tasks completely get scheduled.

The following is the comparative result of the above case study.

Table 6: Comparative study between proposed two algorithms

Algorithm/Parameter	***Schedul-ability***	***Success Ratio***	***CPU Utilization (%)***	***Context switches***
Modelled FPZL Algorithm	Average	0.8	100%	15
Modified FPZL Algorithm	High	1.0	100%	29

CASE STUDY 3:

Consider one more case where the taskset is same which we have considered in case study 2. Only the processor parameters are changed which is given in following table.

Table 7: Input parameters of the processors

Input Parameter	*Value*
Capacity	3
Period	25
Number of processors	4

In the above case, if tasks are scheduled using FPZL algorithm then it will process all the tasks completely. It will give better results in terms of schedulability context switches and success ratio. But in this case, the modeled FPZL algorithm the CPU Utilization is not efficient. Here the processors are underutilized.
As the algorithm is proposed for Multiprocessor system, the processors must be efficiently utilized. So, when the tasksets are scheduled using the proposed algorithm then it is noted that a very small portion of a task remain un-schedulable. But, in this case the CPU is utilized completely. This is shown in following table.

Table 8: Comparative study between proposed two algorithms

Algorithm/Parameter	*Success Ratio*	*CPU Utilization (%)*	*Context switches*
Modelled FPZL Algorithm	1.0	58%	20
Modified FPZL Algorithm	0.98	100%	38

5. Conclusion

In this paper, two algorithms are proposed. First algorithm is the modelled algorithm which was proposed earlier. This algorithm was proposed to schedule more tasks as far as possible. But in modelled one, many times due to the provided processor capacity and its period some tasks remains un-schedulable. In many situations, there may be a condition where from that un-schedulable task some portion of a task is important to process. For this reason, instead of giving up task completely, important portion of that task is processed. This concept is already proposed by logically dividing the task into mandatory and optional portions and processing mandatory tasks before optional tasks. But in this paper, this concept is implemented with the concept of zero laxity which is currently the topic of research. As compared to the modelled FPZL algorithm this proposed algorithm processes more tasks and as a result it increases the Success ratio of the algorithm. It also increases the efficiency of the CPU Utilization. This algorithm increases the context switching parameter which may be a disadvantage. In future, context switches can be targeted to reduce.

References

[1] K. Ramamritham, J.A. Stankovic, and P. F. Shiah, "Efficient Scheduling Algorithms for Real-Time Multiprocessor Systems," IEEE Trans. Parallel and Distributed Systems, vol. 1, no. 2, pp. 184194, Apr. 1990.

[2] Apurva shah,ketan Kotecha,"Adaptive Scheduling Algorithm for real time multiprocessor System",IEEE Advance computing Conference,2009.

[3] Robart Devis,"FPZL Schedulability Analysis", IEEE Real time and embedded Technology and Application Symposium,2011.

[4] Radhakrishna Naik, Vivek Joshi , R.R. Manthalkar," IUF Scheduling Algorithm for improving schedulability,predictability and sustainability of the real time system", Second International Conference on Emerging Trends in Engineering and Technology, ICETET-2009.

[4]Radhakrishna Naik, R.R. Manthalkar, Mukta Dhopeshwarkar" Modified IUF Scheduling Algorithm for Real time Systems", Third International Conference on Emerging Trends in Engineering and Technology,2010 .

[5]Annie s. Wu,han yu,kuo chi lin."An incremental Genetic Algorithm Approach to multiprocessor Scheduling",IEEE Transaction on Parallel and Distributed System,2004.

[6] S. R. Vijayalakshmi,Dr. G. Padmavathi,"A Performance study of GA and LSH in multiprocessor Job Scheduling,International Journal Of Computer Science,2010

[7] Geoffery Black,Ronald Dreslinski,Trevor Mudge,"A Survey of Multicore Processors",2009.

[8] K. Ramamritham, J.A. Stankovic, and P. F. Shiah, "Efficient Scheduling Algorithms for Real-Time Multiprocessor Systems," IEEE Trans. Parallel and Distributed Systems, vol. 1, no. 2, pp. 184194,Apr.1990.

[9] J. Carpenter, S. Funk, et al. A categorization of real-time multiprocessor scheduling problems and algorithms. In J. Y. Leung, editor, *Handbook on Scheduling Algorithms, Methods, and Models, page* 30.130.19 , 2004.

[10] K. Lakshmanan, Rajkumar,"Scheduling Parallel Real time Tasks on Multicore Processors", Real Time System Symposium,IEEE 2010.

[11] Fanxin Kong,yang yi,qingxu deng,"Energy Efficient Scheduling of Real time tasks on Cluster based Multicores",2011.

[12] Jinkyu lin,Insik Shin,"LLF Schedulability Analysis on multiprocessor System",Real time system symposium,2010.

[13] Geoffrey Blake, Ronald G. Dreslinski, and Trevor Mudge,"A survey of multicore Processors",November 2009.

[14] Mostafa R. Mohamed, Medhat H. A. Awadalla,"Hybrid Algorithm for Multiprocessor task scheduling", IJCSI International Journal of Computer Science Issues, Vol. 8, Issue 3, No. 2, May 2011.

[15]Oscar H. Ibarra and Chul E. Kim," Heuristic Algorithms for scheduling Independnt tasks on non-identical processors", Journal of association for computing machinery,vol 24,no 2, April 1977,pp 280-289.

[16] Dan McNulty, Lena Olson, Markus Peloquin,"A comparision of scheduling algorithms for multiprocessors", December 2010.

[17] Sanjoy K. Baruah, Member, IEEE, and Joe¨ l Goossens," Rate-Monotonic Scheduling on Uniform Multiprocessors", IEEE Transactions on Computers, VOL. 52, NO. 7, JULY 2003.

[18] Jia Xu," Multiprocessor Scheduling of Processes with Release Times, Deadlines, Precedence, and Exclusion Relations" IEEE Transactions on Software Engineering, VOL. 19, NO. 2, FEBRUARY 1993 .

[19]Krithi Ramamritham,John A. Stankovic,and perng-fei Shiah,"Efficient scheduling algorithms for Real time Multiprocessor Systems",IEEE transactions on parallel and distributed systems, VOL 1,NO 2,APRIL 1990.

[20] Marko Bertogna,Michele Cirinei, and Giuseppe,Lipari, "Schedulability Analysis of Global Scheduling Algorithms on Multiprocessor Platforms", IEEE Transactions on parallel and Distributed Systems, VOL 20 , no 4, April 2009

[21] Theodore P. Baker," Multiprocessor EDF and Deadline Monotonic Schedulability Analysis", Proceedings of the 24th IEEE International Real-Time Systems Symposium, 2003

Komal Bhalotiya is a Research student, pursuing M.E degree from G. H. Raisoni College of Engineering, Nagpur. And has done B.E from Shri Ramdeobaba Kamla Nehru Engineering College,Nagpur.

Dr. D. V. Padole completed his graduation in 2001 as an Electronics Engineer, Post-Graduation in 2003 & awarded as Doctorate from RTM Nagpur University, Nagpur (India) in year 2011. He is member of various professional societies like IEEE, ISTE and CSI. He is having more than 9 years of experience in teaching profession. Currently he is working as a Professor in Department of Electronics Engineering, G.H. Raisoni College of Engineering, Nagpur (India). His research interest includes Multiprocessor /Multicore systems, Embedded System Design, and Digital Signal Processing. Several research publications & Book Chapter contribution in the International Book are on his account. He worked as reviewer & Chaired technical sessions for several International conferences at India and abroad.

Radhakrishna Naik is a research scholar and pursuing Ph.D.