International Journal of Applied Engineering Research ISSN 0973-4562 Volume 12, Number 16 (2017) pp. 5681-5689

Schedulability Analysis for Rate-Monotonic Algorithm in Parallel Real-Time Systems

Leena Das

Department of Computer Science & Engineering, KIIT University, campus-15, Bhubaneswar-751024, Odisha, India.

Orcid Id: 0000-0002-8719-1991

Durga Prasad Mohapatra

Department of Computer Science & Engineering, NIT, Sector-2, Rourkela-69008, Odisha, India.

Sourav Mohapatra

Department of Computer Science & Engineering, NIT, Sector-2, Rourkela-769008, Odisha, India.

Abstract

Content Rate- Monotonic algorithm (RMA) is a widely used static priority scheduling algorithm. For application of RMA on various systems, first it is essential to determine the system's feasibility. Various existing algorithms perform the analysis by reducing the scheduling points in a given task set. In this paper, we develop an algorithm to compute the RMA schedulability in a parallel real-time system. We discuss the bottlenecks of serial execution and show that the parallel algorithm to overcomes these limitation. The proposed algorithm is parallelized using OpenMP. The results obtained show the improvement due to the use of parallel algorithm in a multiprocessor environment.

Keywords: Parallel, Real-Time, RMA, Schedulability.

INTRODUCTION

Real-time systems are different from other types of systems in the sense that they must provide accurate results in both temporal and logical aspects. Apart from being able to carry out the required task, the system must do it within a certain period of time. For example, when a temperature sensor in a thermal plant gives a warning, then the system must turn on cooling mechanisms within a certain interval of time barring which the plant's operation may fail. Here the response of the system must be correct as well as be completed in a given interval of time. A real-time system can be divided broadly into three spheres; environment, controller and controlled object. The controller gets input from the environment and then provides information to the controlled object. The time it takes to provide the instruction is known as execution time. The time after which an event repeats itself in the environment is known as the period of that event. Every event needs a certain time interval before which it has to be executed. This is known as the deadline.

Thus a real-time system's primary goal is to provide a scheduling algorithm where all the deadlines are met, taking into account the period and execution time. To accomplish this there are a number of algorithms primarily categorized into two categories: static priority and dynamic priority algorithms. Static priority algorithms are those in which the priority assigned are static in nature. While in dynamic algorithms, priority changes dynamically. This work concentrates on static priority scheduling, most specifically RMA.

Over the last decade, interest in parallel programming has grown tremendously and hardware systems that contain different levels of parallelism have become mainstream. It is common to find laptop and desktop systems that contain a small number (2-8) of these Shared-Memory Processors (SMP) chips. Furthermore, high-end computing systems are now able to contain hundreds of these SMP chips, resulting in machines that are capable of running more than 1000 hardware threads simultaneously. As processor speed begins to stagnate, software developers are being forced to exploit parallelism that is available in these systems in order to improve the performance of their applications. In this paper we aim using the parallel techniques on schedulability analysis problems.

A. MOTIVATION

Before the proper scheduling is carried out in a given system, it is of high importance that the task set is analyzed first. In

most of the situations, the computation of schedulability analysis requires less time than carrying out the actual schedulability but gives valuable information regarding the nature of dataset. In this work, we implement a parallel algorithm for RMA schedulability. In a multiprocessor environment there are number of processors which can be assigned some work. Instead of performing serial analysis in a single processor, by taking the advantage of the processors, parallel analysis can provide a better solution. The Improved Time Demand Analysis [ITDA] algorithm gives good enough performance but that can still be improved in case of multiprocessor environment by running algorithm parallely and we explore that.

B. OBJECTIVES

The objectives of this paper are as follows:

- To implement improved time demand analysis algorithm in a parallel multiprocessor system
- To show that parallel execution in a multiprocessor environment performs better than serial execution.

BASIC CONCEPT

Rate- Monotonic Algorithm (RMA) is one of the most widely used and effective scheduling algorithms. RMA uses mathematical model of static priority based scheduling where the priority is the period of the tasks. It supports the intuition that the tasks that occur more frequently should be given higher priority. RMA was first proposed in [1]. Working of RMA depends on the periods of the tasks. These feasibility tests are generally known as schedulability bounds. For an algorithm to work, it must be within certain limits. Schedulability bound provides this limit. The work in [1] gave an initial feasibility bound. It was improved by Seto and Lehoczy in [2], [3] by the using a time demand analysis function to make it an exact feasibility test. Our work focuses on this aspect of RMA scheduling. Various types of other bounds were also proposed. The concept of the harmonic period was explored by Kuo et. al in [4] that showed the schedulability of tasks satisfying the harmonic conditions. This report now analyses the initial method proposed in [2], [3], often termed as time demand analysis and implements it. We then propose an algorithm to improve this implementation.

Real-time systems can be divided into three broad categories

- Hard real-time Systems
- Firm real-time Systems
- Soft real-time Systems

The event that determines a course of action is known as a task. On the occurrence of a task, the system does processing and responds accordingly. There are three types of tasks [22],][23].

- Periodic Tasks
- Aperiodic Tasks
- Sporadic tasks

Periodic Tasks are the tasks that occur after a specified interval of time. For example a sensor sending temperature data every 10 seconds. We say that this task is periodic with a period of 10 seconds. On the other hand, an aperiodic task is one where the task can occur after any amount of time after the occurrence of the last instance, except immediately. Sporadic tasks are aperiodic tasks where the repetition period can be zero. In our paper, we deal with periodic tasks only. Now we describe the various notations used in our paper.

Ci : Execution time of i^{th} task

Ti : Period of i^{th} task

Ui : Utilization of i^{th} task

wi : Time demand function value

t : Current time point

Other notations have been described as and when used.

RMA was first proposed in [1] in 1973 as an optimal scheduling algorithm for static priority task set. The priority was assigned to the periods of the tasks. For a task set T (T_1; T_2; :::; T_n) period(T_i) < period(T_j) =) priority(T_i) > priority(T_j)

For validating the feasibility of a task set by determining whether it is schedulable or not, a variety of tests has been developed. The first universal feasibility bound for all types of scheduling systems is given by

$$\sum_{i=1}^{n} U \leq 1 \qquad (1)$$

The sum of utilization of all the tasks in the task set should be less than equal to 1. This gives the necessary upper bound for any scheduling algorithm including RMA. But this is not sufficient. We refer to this bound as schedulability bound 1. A tighter feasibility test was proposed in [1] which stated that a periodic static priority system is feasible if

$$\sum_{i=1}^{n} U_i \leq n(2^{1/n} - 1) \qquad (2)$$

where n is the total number of tasks in the task set. The value tends to ln(2) as n tends to 1. This shows that in any sufficiently large task-set, if the total utilization is less than 0.693, then it can be scheduled. This, however, is a sufficient

condition only. Even if the total utilization remains greater than this bound, it can still be static feasible. To take into account this factor, various necessary and sufficient tests were proposed in [2], [6], [14], [16], [17],[21], [25], [26]. The initial test proposed in [2], [6] was further improved in [14], [17]. These tests remained pseudo-polynomial. The proposed tests can be divided into two types; iterative [6], [25] and as-per-task basis [2], [17],[18],[19]. The later analyses feasibility on task arrival times known as scheduling points. The former uses an iterative technique. Recently the work by Allah et. al. in [16] improved the work done by Bini and Buttazzo in [14] by restricting the actual number of scheduling points. We now discuss the exact feasibility test upon which we propose our improvement.

A. SCHEDULABILITY TEST

Phase I of a task is defined as the time when the first instance of the task is released into the system. Two tasks Ti and Tj are said to be of same phasing if Ii = Ij. In the work [1], it was shown that in the scenario where the phasing of all the tasks is equal, it results in the longest response time. This is generally known as the critical instant. This scenario creates the worst case task set phasing and thus can be used as a criterion for the schedulability of the given task set. The idea can be re-framed as "A periodic task set can be scheduled by a fixed priority scheduling algorithm if the deadline of the first job of each task is met while using that algorithm from a critical instant". Since RMA is a fixed priority scheduling algorithm, the condition satisfies for it.

Let T be a task set T (T_1; T_2; :::; T_n) with tasks having increasing periods (thus decreasing priority). As per RMA's priority property, a task having lower period has to be scheduled before the task of a higher period. Thus a task Ti can only be affected by the tasks T_j where period(T_j) > period(T_j). This gives the intuition that while checking for the schedulability of the task only that task should be considered whose period is more than the current one (or priority less). This ordering can be achieved by sorting the task set in ascending order of their period. When the task set is processed, the tasks are encountered with decreasing priority. At the time of critical instance, a value is calculated. This value gives the estimate as to how much the system is feasible. These ideas were stated mathematically in [2] as follows

$$w_i(t) = C_i + \sum_{1}^{i-1} [t / T_k] * C_k \qquad (3)$$

where $w_i(t)$ is the time demand function of i^{th} task when it is released at the critical instant. As can be seen, the tasks from starting to i^1 only contribute to this value function. The tasks after the i^{th} task cannot affect as they have lower priority (as task set is sorted). After this calculation, the schedulability bound was given as

$$w_i(t) < t \qquad (4)$$

where $w_i(t)$ can be interpreted as demand and the time t can be seen as the supply. So, the demand must be less than the supply for the task set to be feasible. Equation (4) involves checking time instances for the demand of a given task. The time instances that must be checked relates to the tasks having higher priority than the current one. As mentioned earlier, the property of RMA asserts that only higher priority tasks can affect the current task. Thus only those time instances need to be checked that are multiples of the period of all the high priority tasks. Mathematically,

$$t = j * Tk; \qquad (5a)$$

$$k = 1, 2, \ldots i; \qquad (5b)$$

$$j = 1,2,....,[T_i / T_k] \qquad (5c)$$

Combining Equation (3) and (5a, 5b, 5c) an algorithm can be implemented using Equation (4) as the checking condition. We shall refer this algorithm to as Time Demand Analysis (TDA). The algorithm for TDA is given in Algorithm 1 and explained below. We can divide the algorithm into three phases for better understanding [20]:

- Determining the Order of execution - The task-set is sorted as per increasing period. The sorted task sets are sequentially processed one by one. Every task is then subjected to TDA. The task, if any, at which the TDA gives the result as un-schedulable, is the threshold task. Since the task set is sorted, any task after that also will be un-schedulable.
- Determining the Discrete time points - Schedulability test, as mentioned earlier need only be performed at certain time intervals. Using Equation (5), those time points are calculated for every task. These are the points where an instant of one of the tasks occurs and thus needs to be checked for schedulability.
- Computing the time demand function value - The value calculated from Equation (3) can be computed after all the time points are calculated. At every time point, this value is computed and continually summed over. The final sum is the required value that is compared to Equation (4). This value is then checked as per Equation (4) which gives the decision whether schedulable or not.

Algorithm 1: TDA

Input: task set

Output: schedulable or not

1 Sort the task-set;
2 repeat
3 Calculate time points from Equation (5);
4 Calculate w(t) from Equation (3);
5 if w(t) >= t then
6 return Non-Schedulable;
7 exit algorithm;
8 until all tasks in the sorted set have been processed;
9 return schedulable;

B. IMPROVED TIME DEMAND ANALYSIS.

TDA can be further improved by using optimization techniques. The improved algorithm, termed as Improved Time Demand Analysis (ITDA) is given in Algorithm 2. The equations required for the computation of the schedulability are also stated below.

$$r = t/w(t) \quad (6)$$

$$r \geq 1 \quad (7)$$

$$m = k + [r * (n / MAX)] \quad (8)$$

Where r is the ratio and k is the task position of the current task and m is the next task to be executed.

C. ITDA FOR MULTIPROCESSOR

The ITDA is extended to multiprocessor environment and named as MITDA. MITDA takes the number of processors available in the multiprocessor environment as input it divides the task set among the specified number of processors. After the division is done, the algorithm checks the feasibility of tasks in each allocated processor. MITDA algorithm is presented in Algorithm 3 and described below,

- Allocate the tasks to processors: The tasks are divided among the processors. This division is done by balancing the utilization using Utilization Balancing Algorithm [24].
- Determine schedulability of each processor: Each processor has a number of tasks allocated to it which are balanced in terms of total utilization. On each of these task-sets, ITDA is performed. The result obtained gives the schedulability of each processor. Combining the results of the individual processors, the total schedulability can be estimated.

Algorithm 2: ITDA

Input: task set

Output: Schedulable or Unschedulable

1. Sort the task-set;
2. k = 1;
3. Repeat
4. Task considered k;
5. Calculate time points from Equation (5);
6. Calculate w(t) from Equation (3);
7. if w(t) >= t then
8. return Not-schedulable;
9. exit algorithm;
10. Calculate ratio r using Equation (6);
11. Calculate m using Equation (8);
12. k = m;
13. until last task is not checked OR ratio >= 1 ;
14. Return schedulable;

Algorithm 3: MITDA

Input: task set, processors

Output: schedulable or unschedulable

1. Sort the task-set;
2. k = 1;
3. Initialize utilization = 0 for each processor;
4. Repeat
5. Find processor Pk with min utilization;
6. Assign next task Ti to Pk;
7. Update utilization of Pk+ = utilization of Tk;
8. until all tasks have been allocated;
9. for each processor Pk do
10. Considered allocated task-set for Pk;
11. k = 1;
12. repeat
13. Task considered k;
14. Calculate time points from Equation (5);
15. Calculate w(t) from Equation (3);
16. if w(t) >= t then
17. return Notschedulable;
18. exit algorithm;
19. Calculate ratio r using Equation (6);
20. Calculate m using Equation (8);
21. k = m;
22. until last task is not checked OR ratio >= 1;
23. return schedulable;

D. OPENMP AS A PARALLEL API

Parallelization can be achieved via a lot of different ways. The use of pThreads is the most native and close to the system. It provides a much faster way to compute but the flexibility offered is not much. To achieve better scalability and flexibility, use of various APIs are recommended [5].

OpenMP is an API that provides thread based multiprogramming shared memory solution. It is a set of compiler directives and library routines for parallel application programmers. It greatly simplifies writing multi-threaded programs in C++. Most of the constructs in OpenMP are compiler directives. With the help of these directives it creates and assigns threads to various sections. We describe some of the commonly used directives [8],[9],[10],[11].

Compiler directives appear as comments in the source code and are ignored by compilers unless mentioned otherwise - usually by specifying the appropriate compiler flag. OpenMP compiler directives are used for various purposes:

1. Spawning a parallel region
2. Dividing blocks of code among threads
3. Distributing loop iterations between threads serializing sections of code.
4. Synchronization of work among threads
5. Compiler directives have the following syntax:
6. Sentinel directive-name [clause...]

There are also various run time routines to perform other works. These routines are used for a variety of purposes that include setting and querying the number of threads, querying a thread's unique identifier (thread ID), a thread's ancestor's identifier, the thread team size, setting and querying the dynamic threads feature, querying if in a parallel region, and at what level, setting and querying nested parallelism, setting, initializing and terminating locks and nested locks and querying wall clock time and resolution.

In this work we use combination of these routines and compiler directives to implement the parallel algorithm.

LITERATURE SURVEY

Modern real-time embedded applications present high computation requirements which need to be realized within strict time constraints. The current trend towards parallel processing in the embedded domain allows providing higher processing power. However, in some embedded applications, the use of powerful enough multi-core processors, may not be possible due to energy, space or cost constraints. A solution to this problem, given in [12] is to extend the parallel execution of the applications, allowing them to distribute their workload among networked nodes, on peak situations, to remote neighbor nodes in the system.

In this context, this paper presents the Partitioned-Distributed-Deadline monotonic Scheduling algorithm for fork-join parallel/distributed fixed-priority tasks [7],[13]. This paper studies the problem of scheduling fork-join tasks that execute in a distributed system, where the inherent transmission delay of tasks must be considered and cannot be deemed negligible, as in the case of multicore systems. The scheduling algorithm is shown to have a resource augmentation bound of 4, which implies that any task set that is feasible on m unit-speed processors and a single shared real-time network, can be scheduled by our algorithm on m processors and a single real-time network that are 4 times faster.

Dimakopoulos [15] classified the models according to the memory abstraction they present to the programmer and then presented the representative ones in each category. They covered shared-memory models, distributed-memory models, and models for Gpus and accelerators. Hybrid models combine the aforementioned ones in some way. They finally concluded with a summary of other models that do not fit directly in the above categories.

PARALLEL MITDA

In this section, we discuss our proposed work on MITDA which gives the schedulability analysis for a system having more than one processor. In today's world of increasing number of processors and cores per processor, this is highly essential to consider multiprocessor systems.

Thus parallelizing an algorithm gives much better results. As far the literature survey is concerned, there is no algorithm to parallelize schedulability analysis of RMA. The above two reasons are the motivation behind the work of proposing a parallel algorithm for ITDA.

A. CHALLENGES FOR PARALLEL ALGORITHM

There are a number of challenges that are faced while parallelizing a given algorithm. Some of the problems must be discussed so that they can be overcome while designing the algorithm. Three of the major problems are given below;

- Some problems do not have work-efficient parallel algorithms that exhibit massive parallelism. That is, they simply cannot be solved with a large number of parallel execution units without significantly increasing the computation complexity. Some problems have known parallel algorithms with ample parallelism but questionable numerical stability. While computing in parallel, the final result depends on which thread of execution completes its assigned section first. Working around this problem creates a bottleneck which makes the complexity same as serial execution.

- Some applications that have parallel algorithms are plagued by catastrophic load imbalance due to highly non-uniform data distribution. Pre-processing data to make it feasible to be given as input increases both space and time complexity.

To parallelize ITDA, each of the problems must be addressed in context to their schedulability and parallelism.

B. OVERCOMING THE CHALLENGES

Each of the problems is examined properly. First, it has to be determined that whether ITDA executes massive parallelism or not. In Section 2 we have presented the equations based on which ITDA works. By analyzing how the task-set is handled by the equations, the required property can be found out.

For each task, the scheduling points are calculated. For each scheduling point, the time demand function value is compared which determines the schedulability. Each task has a computation (calculation of scheduling points) assigned to it. For a task set of 1000 tasks, there would be a high number of such calculations. This calculation can be performed in parallel. Thus ITDA overcomes the problem of being possible to be parallelized.

For the second problem, the dependency of the computation for each of the task must be analyzed. This was established in Section 2 that the computation of one task's time demand function is independent of any other task's computation. Thus ITDA satisfies the second criteria.

In the third problem, the task set must be properly load balanced without any added computation. As for MITDA, a load balanced division algorithm is used to allocate tasks to each processor, this problem is thus solved. Thus ITDA can be parallelized by the means of a parallelizing construct.

C. PROPOSED ALGORITHM

The algorithm developed earlier in Algorithm 3 can be modified to introduce parallelism for MITDA. The proposed algorithm is described in steps given below.

- Allocate the tasks to processors: The tasks are divided among the processors. This division is done by balancing the utilization using Utilization Balancing Algorithm. This section is performed by the master thread. [22].
- Create a team of threads: The number of threads created is equal to the number of processors present in the system. Each of the thread is given a processor to compute the schedulability analysis for it.
- Determine schedulability of each processor: Each processor has a number of tasks allocated to it which are balanced in terms of total utilization. On each of these task-sets, ITDA is performed. The result obtained gives the schedulability of each processor. Combining the results of the individual processors, the total schedulability can be estimated. This function is carried out individually in each of the threads.

Algorithm 4: Parallel MITDA

Input: task set, processors

Output: Schedulable or Unschedulable

1. Sort the task-set;
2. k = 1;
3. Initialize utilization = 0 for each processor;
4. Repeat and find processor Pk with min utilization;
6. Assign next task Ti to Pk;
7. Update utilization of Pk+ = utilization of Tk;
8. until all tasks have been allocated;
9. Create a team of threads;
10. for each processor Pk do
11. Allocate the processor to a thread;
12. Run ITDA() on each thread;

Table 4.3.1: The Performance of Parallel MITDA comparing with serial MITDA

No of Tasks	Ratio of Serial to Parallel height
500	4.25
1000	4.52
2000	4.67
5000	4.78
10000	4.89
20000	4.91
30000	4.95
40000	4.99
50000	5.04

D. IMPLEMENTATION

For implementing this algorithm, shared memory architecture is considered. In shared memory architecture, the memory is

shared by the threads doing parallel execution. Further to achieve parallelism, OpenMP API is being used. OpenMP is an API for multi-platform shared-memory parallel programming in C/C++. The primary reason for using OpenMP is its flexibility, scalability, and capability to incrementally parallelize a serial program, unlike message-passing libraries which typically require an all or nothing approach.

The algorithm has been implemented for a static input. In Step 9 of Algorithm 4, the threads are created using OpenMP section directive. Steps 10-12 are executed by the master thread. In step 12, ITDA is run on each individual thread. The number of threads and processors used are taken as predefined input. The output of the algorithm is whether the task set is schedulable or not.

E. RESULT

The parallel algorithm works better than serial implementation. Each of the processor's work is assigned to a separate thread which causes the total load to be divided. The figure shows the relative performance gain ratio of parallel to serial execution. On a task set of around 500 tasks, the improvement ratio is around 4.5. This ratio can be further improved by increasing the number of threads in OpenMP.

```
sourav@SOURAV-PC:~/fyp/parallel$ ./rma
-------------------------------------------------------------------------
        RMA Scheduability Analysis in Multiprocessor Environement
        Improved Time Demand Analysis
        Parallelized using OpenMP
        Dept of Computer Science and Engineering - NITR
-------------------------------------------------------------------------

Enter Task Set size: 10000

                PARAMETERS
-----------------------------------
NO OF PROCESSORS        4
NO OF TASKS             10000
THREADS IN OPENMP       4
-----------------------------------

Press Enter to proceed

                Task allocation to processors
-------------------------------------------------------------
Processor #0  |  Processor #1  |  Processor #2  |  Processor #3  |
-------------------------------------------------------------
      1241           1253           1254           1247
-------------------------------------------------------------

Total Time taken for parallel execution:        129ms
Total Time taken for serial execution:          496ms


-----------------------------------
Time improvement ratio:                 3.84496

sourav@SOURAV-PC:~/fyp/parallel$
```

Figure 4.4.1. Performance gain of parallel MITDA wrt serial implementation

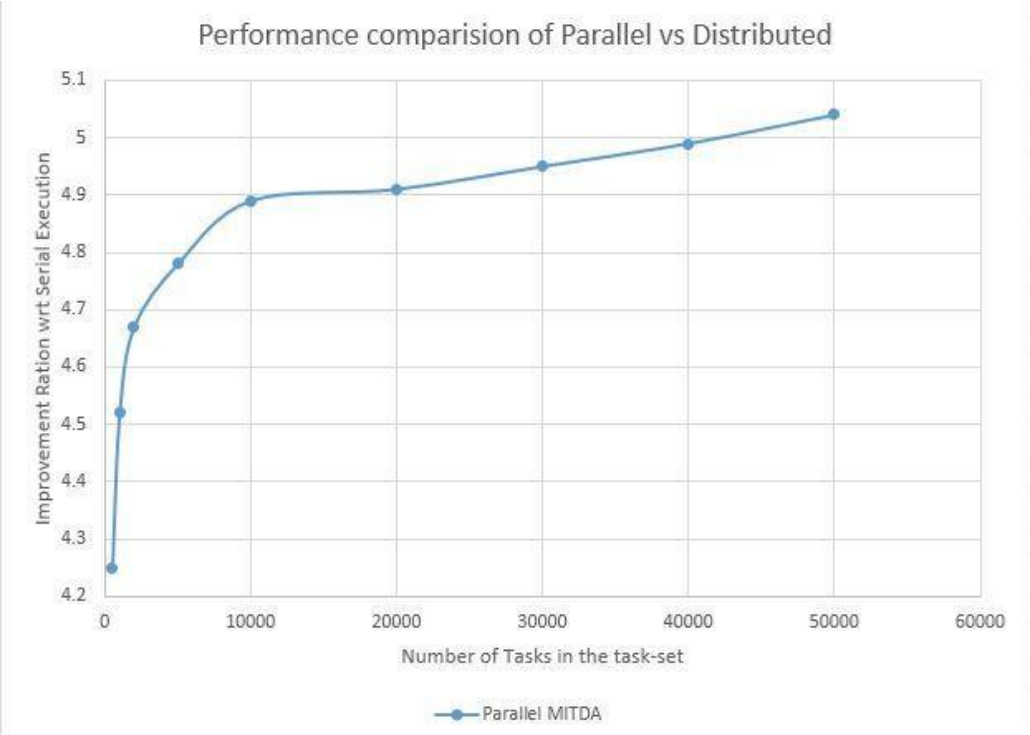

Figure 4.4.2. Performance gain of parallel MITDA wrt serial implementation

CONCLUSIONS

In this paper, we have discussed the schedulability test for RMA - ITDA and MITDA. We explored the possibility of extending the MITDA to parallel systems and taking advantage of the multiple numbers of processors present. The proposed Parallel MITDA algorithm is shown to give a high performance ratio than serial. This result shows that in a multiprocessor environment, the serial execution can be replaced by parallel execution.

ACKNOWLEDGMENT

Fore most I would like to express my gratitude to my guide Dr. Durga prasad Mohapatra for continuous support for my Ph.D work. His guidance helped me all the time of research and writing the thesis paper. My sincere thanks goes to Sourav Mohapatra for his contribution for this work. Last but not the least I would like to thank my organization KIIT University to encourage my Ph.D work.

REFERENCES

[1] C. L. Liu and J.W. Layland. Scheduling Algorithms for Multi-Programming in a Hard Real-Time Environment: Journal of the Assn of Computing Machinery (ACM) 20, 1, pp.40-61, 1973.

[2] Lehoczky, John, Lui Sha, and Ye Ding. The rate monotonic scheduling algorithm: Exact characterization and average case behavior. Real Time Systems Symposium Proceedings, IEEE, 1989.

[3] Seto, D., J. P. Lehoczky, L. Sha, and K. G. Shin. On Task Schedulability in Real-Time Control System, Proceedings of 17th Real-Time Systems Symposium, pp. 13-21, 1996.

[4] T. W. Kuo, Y. H. Liu, and K. J. Lin. Efficient On-Line Schedulability Tests for Priority Driven Real-Time Systems, Proceedings of the IEEE Symposium on Real- Time Technology and Applications, 2000.

[5] Chapman, Barbara, Gabriele Jost, and Ruud Van Der Pas. Using OpenMP: portable shared memory parallel programming. Vol.10. MIT press, 2008.

[6] Audsley, Neil, et al. Applying new scheduling theory to static priority pre-emptive scheduling. Software Engineering Journal 8.5: pp.284-292, 1993.

[7] Birman, Kenneth P., and Robbert van Renesse, eds. Reliable distributed computing with the Isis toolkit. Vol.85. Los Alamitos: IEEE Computer society press, 1994.

[8] R.Thakur and W.Gropp ,Open Issues in MPI implementation, in Proc. 12th Asia-Pacific Computer System Architecture Conference(ASSAC 2007), August 2007, pp. 327-328.

[9] W. Gropp & R. Thakur. Issues in Developing a Thread-Safe MPI Implementation, in Proc. of the 13th European PVM/MPI Users Group Meeting, pp. 12-21, 2006.

[10] D. Buntinas, G. Mercier and W. Gropp. Implementation and Evaluation of Shared-Memory Communication and Synchronization Operations in MPICH2 using the Nemesis Communication Subsystem, Parallel Computing, Volume 33, Issue 9, pp. 634-644, 2007.

[11] R. Thakur, W. Gropp, and B. Toonen. Minimizing Synchronization Overhead in the Implementation of MPI One-Sided Communication, in Proc. of the 11th European PVM/MPI Users Group Meeting, Recent Advances in Parallel Virtual Machine and Message Passing Interface, Lecture Notes in Computer Science, LNCS 3241, Springer, pp. 57-67, 2004.

[12] Garibay-Martnez, Ricardo, et al. On the scheduling of fork-join par-allel/distributed real-time tasks. Industrial Embedded Systems (SIES), 9th IEEE International Symposium on, IEEE, 2014.

[13] Coulouris, George, Jean Dollimore and Tim Kindberg; Gordon Blair. Distributed Systems: Concepts and Design (5th Edition), 2011.

[14] Bini, Enrico, and Giorgio Buttazzo. A hyperbolic bound for the rate monotonic algorithm. Real-Time Systems,13th Euromicro Conference on, IEEE, 2001.

[15] V.V. Dimakopoulos. Parallel Programming Models, in Smart Multicore Embedded Systems, M. Torquati, K. Bertels, S. Karlsson, F. Pacull eds., Springer Science+Business Media, New York, pp. 3-20, 2013.

[16] Allah, Nasro Min, and S. Islam and Wang YongJi. Enhanced Time Demand Analysis, World Applied Sciences Journal 9.1., 2007.

[17] Manabe, Yoshifumi, and Shigemi Aoyagi. A feasibility decision algo-rithm for rate monotonic and deadline monotonic scheduling, Real-Time Systems 14.2: pp.171-181, 1998.

[18] Bini, Enrico, and Giorgio C. Buttazzo. Schedulability analysis of periodic fixed priority systems". IEEE Transactions on Computers 53.11: pp.1462-1473, 2004.

[19] Bini, Enrico, and Giorgio C. Buttazzo. The space of rate monotonic schedulability, In Real-Time Systems Symposium 2002, RTSS. 23rd IEEE, pp. 169-178, 2002.

[20] Thomas H.. Cormen, Charles Eric Leiserson, Ronald L. Rivest and Clifford Stein. Introduction to algorithms. Vol. 6. Cambridge: MIT press, 2001.

[21] Enrico Bini and Giorgio C. Buttazzo. Measuring the Performance of Schedulability Tests, Springer, Real-Time Systems, Volume 30, Issue 1, pp 129-154, 2005.

[22] Rajib Mall. Real Time System: Theory and Practice 3rd Edition, Pearson Education: 390-395, 1986.

[23] Lalatendu Behera and Durga Prasad Mohapatra. Schedulability Analysis of Task Scheduling in Multiprocessor Real-Time Systems Using EDF Algorithm, International Conference on Computer Communication and Informatics, 2012.

[24] F. Zhang and A. Burns. Schedulability Analysis for Real-Time Systems with EDF scheduling , IEEE Transaction on computers, vol. 58, no. 9, pp.1250-1258, 2009.

[25] Sjodin, Mikael, and Hans Hansoon,"Improved response-time analysis calculations." Real-time Systems Symposium,1998 Proc. The 19th IEEE,1998.

[26] Joseph, Mathai and Paritosh Pandya, " Finding response times in a real-time system" The Computer Journal 29.5(1986) pp. 390-395.