International Journal of Advanced Computer Research (ISSN (print): 2249-7277 ISSN (online): 2277-7970)
Volume-2 Number-4 Issue-6 December-2012

Real-Time Scheduling for Parallel Task Models on Multi-core Processors - A critical review"

Mahesh Lokhande[1], Mohd. Atique[2]
Research Scholar, Dept. of CS and Engineering, PIET, Nagpur, India[1]
Associate Professor, Dept. of CS, SGBAU, Amravati, India[2]

Abstract

Multi-core processor technology has been enhanced spectacularly and it is reasonably good in performance than single core processors thereby having the potential to enable computation-intensive real-time applications with precise timing constraints. Mostly traditional multiprocessor real-time scheduling is stick to Sequential models which ignore intra-task parallelism while Parallel models such as OpenMP have the capability to parallelize specific segments of tasks, thereby leading to shorter response times when possible. In this paper various research papers have been reviewed and are categorized as Sequential Real-Time Task based Research and Parallel Real-Time Task based Research. Also various approaches such as task splitting techniques, scheduling policies and techniques used are considered for comparing real time task scheduling in multi-core processors.

Keywords

Scheduling, task models, task splitting, task policies.

1. Introduction

Multi-core processors are considerably good in performance as compare to single core processors. Therefore, they have the potential to enable computation-intensive real-time applications with rigorous timing constraints that cannot be met on traditional single-core processors. Massively multi-core processors are rapidly gaining market share with major chip vendors offering an ever increasing number of cores per processor. However, most results in traditional multiprocessor real-time scheduling are limited to sequential programming models and ignore intra-task parallelism. From a programming perspective, the sequential programming model does not scale very well for such multi-core systems. Parallel programming models such as OpenMP present promising solutions for more effectively using multiple processor cores. Major chip manufacturers have recently ramped up the development of massively multi-core processors for a variety of reasons including power consumption, memory speed mismatch, and instruction-level parallelism limits. This development has shifted the scaling trends from processor clock frequencies to the number of cores per processor. For example, AMD has introduced a 12-core Opteron [1] processor targeting the datacenter server market, while Intel has developed a 48-core single-chip computer for cloud computing [2]. Intel also has recently put 80 cores in a Teraflops Research Chip [3] with a view to making it generally available, and ClearSpeed has developed a 96-core processor [4]. Tilera announced a 100-core processor, TILE-Gx100 [5].

While hardware technology is moving at a rapid pace, software and programming models have failed to keep pace. For example, Intel has set a time frame of 5 years to make their 80-core processor generally available due to the inability of current operating systems and software to exploit the benefits of multi-core processors [3]. As multi-core processors continue to scale, they provide an opportunity for performing more complex and computation-intensive tasks in real-time. However, to take full advantage of multi-core processing, these systems must exploit intra-task parallelism, where parallelizable real-time tasks can utilize multiple cores at the same time. By exploiting intra-task parallelism, multi-core processors can achieve significant real-time performance improvement over traditional single-core processors for many computation intensive real-time applications such as video surveillance, radar tracking, and hybrid real-time structural testing [6] where the performance limitations of traditional single-core processors have been a major hurdle.
Parallel programming models such as OpenMP [7], Java [8], Pthreads [9] and Cilk+ [10] are competent candidates for taking advantage of future massive multi-core processors. These models have the capability to parallelize specific segments of tasks, thereby leading to shorter response times when possible.

2. Literature Review

Various researches are carried out their research in recent past years for efficient real time task scheduling. For analyzing algorithms they use various terms like task types, task parameters, task priorities, scheduling categories, etc. Also various performance metrics such as Utilization bounds, Approximation Ratio, Resource Augmentation or Speedup factor and Empirical measures, are used to compare the effectiveness of different scheduling algorithms.

Task based Review

Various research papers have been reviewed and are categorized as 1) Sequential Real-Time Task based Research and 2) Parallel Real-Time Task based Research.

Sequential programming models proved to be quite useful when processor manufacturers pushed for faster and faster processor clock speeds. As the semiconductor vendors shift the scaling trends towards more and more processor cores, the benefits of sequential programming start to diminish in comparison to the inability to take advantage of the available parallelism. There are various Parallel programming models, e.g. OpenMP [7], are promising candidates that takes benefit of the future substantial multi-core processors. These models have the potential to parallelize particular segments of tasks.

Sequential Real-Time Task based Research

There has been extensive work on traditional multiprocessor real-time scheduling [11]. Most of this work focuses on sequential programming model, on multiprocessor or multi-core systems, where the problem is to schedule many sequential real-time tasks on multiple processor cores.

S. K. Dhall and C. L. Liu [12] studied the problem of scheduling periodic-time-critical tasks on multiprocessor computing systems. A periodic-time-critical task consists of an infinite number of requests, each of which has a prescribed deadline. The scheduling problem is to specify an order in which the requests of a set of tasks are to be executed and the processor to be used, with the goal of meeting all the deadlines with a minimum number of processors. Since the problem of determining the minimum number of processors is difficult, they consider two heuristic algorithms. These are easy to implement and yield a number of processors that is reasonably close to the minimum number. They also analyze the worst-case behavior of these heuristics.

Hard real-time systems require both functionally correct executions and results that are produced on time. This means that the task scheduling algorithm is an important component of these systems. K. Ramamritham, J. Stankovic, and P. Shiah [13], developed efficient scheduling algorithms based on heuristic functions to schedule a set of tasks on a multiprocessor system. The tasks are characterized by worst case computation times, deadlines, and resources requirements. Starting with an empty partial schedule, each step of the search extends the current partial schedule with one of the tasks yet to be scheduled. The heuristic functions used in the algorithm actively direct the search for a feasible schedule, i.e., they help choose the task that extends the current partial schedule. Two scheduling algorithms are evaluated via simulation. For extending the current partial schedule, one of the algorithms considers, at each step of the search, all the tasks that are yet to be scheduled as candidates. The second focuses its attention on a small subset of tasks with the shortest deadlines. The second algorithm is shown to be very effective when the maximum allowable scheduling overhead is fixed. This algorithm is hence appropriate for dynamic scheduling in real-time systems.

A. Khemka and R. K. Shyamasundar [14] developed an optimal scheduling algorithm and described that the feasibly schedules a set of m periodic tasks on n processors before their respective deadlines, if the task set satisfies certain conditions. The complexity of this scheduling algorithm in terms of the number of scheduled tasks and the number of processors and upper bounds on the number of preemptions in a given time interval and for any single task is also derived. The optimal algorithm is shown to be particularly useful when schedules are built from the integral flow values obtained from the corresponding maximum flow network.

In terms of a scheduling game representation of the problem, M. Dertouzos and A. Mok [15] discussed the problems of hard-real-time task scheduling in a multiprocessor environment. It is shown that optimal scheduling without a priori knowledge is impossible in the multiprocessor case even if there is no restriction on preemption owing to precedence or mutual exclusion constraints. Sufficient conditions are derived which will permit a set of tasks to be optimally scheduled at run time.

Joseph Y.T. Leun [16] considers the complexity of determining whether a set of periodic, real-time tasks can be scheduled on $m \geq 1$ identical processor with respect to fixed-priority scheduling. It is shown that the problem is NP-hard in all but one special case.

The complexity of optimal fixed-priority scheduling algorithm is also discussed.

John Carpenter and Shelby Funk [17] presented a new taxonomy of scheduling algorithms for scheduling preemptive real-time tasks on multiprocessors. They described some new classes of scheduling algorithms and considered the relationship of these classes to the existing well-studied classes. They also described known scheduling algorithms that fall under these classes and presented sufficient feasibility conditions for these algorithms. In this, the trade-offs involved in scheduling independent, periodic real-time tasks on a multiprocessor is also analyzed.

Parallel Real-Time Task based Research

There has also been extensive work on scheduling of one or more parallel jobs on multiprocessors [18]–[24]. However, the work in [18]–[21] does not consider task deadlines, and that in [22]–[24] considers soft real-time scheduling. In contrast to the goal (i.e. to meet all task deadlines) of a hard real-time system, in a soft real-time system the goal is to meet a certain subset of deadlines based on some application specific criteria.

C. D. Polychronopoulos and D. J. Kuck [18] extensively studied the problem of scheduling iterations of parallel loops among different processors in a parallel system. They proposed the Guided self scheduling technique which addresses the problem of uneven start times for each processor. Instead of using a fixed chunk size, they propose decreasing chunk sizes, calculated as a decreasing function of the current iteration number i being executed. As execution proceeds, smaller chunks improve the balance of the workload toward the end of the loop.

N. S. Arora and R. D. Blumofe [19] presented a user-level thread scheduler for shared-memory multiprocessors, and analyzed its performance under multiprogramming. They model multiprogramming with two scheduling levels: their scheduler runs at user-level and schedules threads onto a fixed collection of processes, while below this level, the operating system kernel schedules processes onto a fixed collection of processors. In this they consider the kernel to be an adversary, and their goal is to schedule threads onto processes such that efficient use of whatever processor resources are provided by the kernel can be done.

Considering the problem of scheduling dynamically arriving jobs in a non-clairvoyant setting, that is, when the size of a job in remains unknown until the job finishes execution, N. Bansal and K. Dhamdhere [20] focused on minimizing the mean slowdown, where the slowdown (also known as stretch) of a job is defined as the ratio of the flow time to the size of the job. They use resource augmentation in terms of allowing a faster processor to the online algorithm to make up for its lack of knowledge of job sizes.

Multiprocessor scheduling in a shared multiprogramming environment can be structured as two-level scheduling, where a kernel-level job scheduler allots processors to jobs and a user level thread scheduler schedules the work of a job on the allotted processors. In this context, the number of processors allotted to a particular job may vary during the job's execution, and the thread scheduler must adapt to these changes in processor resources. For overall system efficiency, the thread scheduler should also provide parallelism feedback to the job scheduler to avoid allotting a job more processors than it can use productively. W. J. Hsu, and C. E. Leiserson [21] provides an overview of several adaptive thread schedulers they have developed that provide provably good history-based feedback about the job's parallelism without knowing the future of the job. These thread schedulers complete the job in near-optimal time while guaranteeing low waste. They have analyzed these thread schedulers under stringent adversarial conditions, showing that the thread schedulers are robust to various system environments and allocation policies. To analyze the thread schedulers under this adversarial model, they have developed a new technique, called trim analysis, which can be used to show that the thread scheduler provides good behavior on the vast majority of time steps, and performs poorly on only a few.

J. M. Calandrino and J. H. Anderson [22] explored various heuristics that attempt to improve cache performance when scheduling real-time workloads. Such heuristics are applicable when multiple multithreaded applications exist with large working sets. In addition, a case study that shows how our best-performing heuristics can improve the end-user performance of video encoding applications is presented. A hybrid approach for scheduling real-time tasks on large-scale multicore platforms with hierarchical shared caches is proposed by J. H. Anderson, and D. P. Baumberger [23]. In this approach, a multicore platform is partitioned into clusters. Tasks are statically assigned to these clusters, and scheduled within each cluster using the preemptive global EDF scheduling algorithm. It showed that this hybrid of partitioning and global scheduling performs better on large-scale platforms than either approach alone. They also determine the appropriate cluster size to achieve the best

performance possible, given the characteristics of the task set to be supported.

J. M. Calandrino and D. Baumberger [24] discuss an approach for supporting soft real-time periodic tasks in Linux on performance asymmetric multicore platforms (AMPs). Such architectures consist of a large number of processing units on one or several chips, where each processing unit is capable of executing the same instruction set at a different performance level. They discuss deficiencies of Linux in supporting periodic real-time tasks, particularly when cores are asymmetric, and how such deficiencies were overcome. They also investigate how to provide good performance for non-real-time tasks in the presence of a real-time workload. It is shown that this can be done by using deferrable servers to explicitly reserve a share of each core for non-real-time tasks. This allows non-real-time tasks to have priority over real-time tasks when doing so will not cause timing requirements to be violated, thus improving non-real-time response times. Experiments show that even small deferrable servers can have a dramatic impact on non-real-time task performance.

There has been little work on hard real-time scheduling of parallel tasks. Anderson [25] propose the concept of a megatask as a way to reduce miss rates in shared caches on multicore platforms, and consider Pfair scheduling by inflating the weights of a megatask's component tasks. Preemptive fixed-priority scheduling of parallel tasks is shown to be NP-hard by Han in [26].

O.H. Kwon and K.-Y. Chwa [27] explore preemptive EDF scheduling of parallel task systems with linear speedup parallelism. In this they consider the problem of scheduling independent parallel tasks with individual deadlines so as to maximize the total work performed by the tasks which complete their executions before deadlines. They propose two polynomial-time approximation algorithms for nonmalleable parallel tasks and malleable tasks with linear speedup.

Q. Wang and K. H. Cheng consider a heuristic for non-preemptive scheduling. However, this work focuses on metrics like makespan [28] or total work that meets deadline [27], and considers simple task models where a task is executed on up to a given number of processors.

N. Fisher, S. Baruah, and T. P. Baker [29] presented a polynomial-time algorithm presented for partitioning a collection of sporadic tasks among the processors of an identical multiprocessor platform with static-priority scheduling on each individual processor. Since the partitioning problem is easily seen to be NP-hard in the strong sense, this algorithm is not optimal. A quantitative characterization of its worst-case performance is provided in terms of sufficient conditions and resource augmentation approximation bounds. The partitioning algorithm is also evaluated over randomly generated task systems.

Most of the other work, on real time scheduling of parallel tasks, also address simplistic task models. K. Jansen [30] studied the scheduling of malleable tasks, where each task is assumed to execute on a given number of cores or processors and this number may change during execution.

W. Y. Lee and H. Lee [31] proposed an optimal(it always finds out the feasible schedule if one exists) algorithm for real time scheduling of parallel tasks on multiprocessors, where the tasks have the properties of flexible preemption, linear speedup, bounded parallelism and bounded deadline. The algorithm always delivers the best schedule consuming the fewest processors among feasible schedules.

G. Manimaran, C. S. R. Murthy, and K. Ramamritham [32] studied non-preemptive EDF scheduling for moldable tasks, where the actual number of used processors is determined before starting the system and remains unchanged.

Parallel programming models introduce a new dimension to this problem, where jobs may be split into parallel execution segments at specific points. Recent results [33, 34] have considered different task models for parallel programming. Sebastien Collette and Liliyana Cucu [33] investigated the global scheduling of sporadic, implicit deadline, real-time task systems on multiprocessor platforms. They provided a task model which integrates job parallelism. They proved that the time-complexity of the feasibility problem of these systems is linear relatively to the number of (sporadic) tasks for a fixed number of processors. They proposed a scheduling algorithm theoretically optimal (i.e., preemptions and migrations neglected). Moreover, they provided an exact feasibility utilization bound. Lastly, they proposed a technique to limit the number of migrations and preemptions.

S. Kato and Y. Ishikawa [34] address Gang EDF scheduling, which applies the Earliest Deadline First (EDF) policy to the traditional Gang scheduling scheme, of moldable parallel task systems. They require the users to select at submission time the number of processors upon which a parallel task will run. They further assume that a parallel task generates the same number of threads as processors selected before the execution. In contrast, the parallel task model addressed in this paper allows tasks to have different numbers of threads in different stages,

which makes our solution applicable to a much broader range of applications.

From a programming perspective, the sequential programming model does not scale very well for multi-core systems. Parallel programming models such as OpenMP present promising solutions for more effectively using multiple processor cores. K. Lakshmanan, S. Kato, and R. R. Rajkumar [35] study the problem of scheduling periodic real-time tasks on multiprocessors under the fork-join structure used in OpenMP. They illustrate the theoretical best-case and worst-case periodic fork-join task sets from a processor utilization perspective. Based on observations of these task sets, a partitioned preemptive fixed-priority scheduling algorithm for periodic fork-join tasks is provided. The proposed multiprocessor scheduling algorithm is shown to have a resource augmentation bound of 3:42, which implies that any task set that is feasible on m unit speed processors can be scheduled by the proposed algorithm on m processors that are 3:42 times faster.

For describing the performance analysis of scheduling algorithms, conventionally utilization bounds such as those employed by C. L. Liu and J.W. Layland [37] are used. But there exist task sets with a total utilization slightly greater than and arbitrarily close to 1 that are unschedulable on a system with m processor cores, for this conventional utilization bounds may not be useful for analyzing the performance of the scheduling algorithms for such task sets. Resource augmentation bounds such as those presented S. Funk, J. Goossens, and S. Baruah [38] seem to be competent candidates for the performance analysis of scheduling algorithms.

The growing importance of parallel task models for real-time applications poses new challenges to real-time scheduling theory that has mostly focused on sequential task models. Notably, K. Lakshmanan [35] in his work on parallel scheduling for real-time tasks analyzes the resource augmentation bound using partitioned Deadline Monotonic (DM) scheduling, and does not consider other scheduling policies such as global EDF.

Secondly, K. Lakshmanan [35], considers a synchronous task model (a basic Fork-Join model), where each parallel task consists of a series of sequential or parallel segments. This model can be called synchronous, since all the threads of a parallel segment must finish before the next segment starts, creating a synchronization point. However, that task model is restrictive in that, for every task, all the segments have an equal number of parallel threads, and that number must not be greater than the total number of processor cores. While the work presented represents a promising step towards parallel real-time scheduling on multi-core processors, the restrictions on the task model make the solutions unsuitable for many real-time applications that often employ different numbers of threads in different segments of computation.

Moreover, a task stretch or task decomposition algorithm is used which decomposes each parallel task into a set of sequential tasks, that makes a master thread of execution requirement equal to task period, and assign one processor core exclusively to it. The remaining threads are scheduled using FBB-FDD [29] algorithm. Their results do not hold if, in a task, the number of threads in different segments vary, or exceed the number of cores. Hence may not be directly applicable to more general task models.

These limitations are overcome by considering a more generalized synchronous task model by Abusayeed Saifullah [36] in contrast to the restrictive task model addressed in [35], where a general synchronous parallel task model considered where each task consists of segments, each having an arbitrary number of parallel threads. Also a novel task decomposition algorithm is proposed that decomposes each parallel task into a set of sequential tasks. Each segment may contain an arbitrary number of parallel threads. That is, different segments of the same parallel task can contain different numbers of threads, and segments can contain more threads than the number of processor cores. This model is more portable, since the same task can be executed on machines with small as well as large numbers of cores.

In future, a general fork-join model with more advanced feature such as nested fork-join structures can also be developed. In addition, a task decomposition algorithm that may be applicable to more generalized task model can also be developed. Additionally, the resource augmentation bound considering other scheduling policies such as global EDF can be analyzed.

3. Comparative Study of various real time multiprocessor algorithms

Various real time algorithms are studied and are compared on the basis of task or models used. Table 1 and Table 2 show the comparison between various algorithms.

Table 1: Comparative study of various real time multiprocessor algorithms meant for Sequential Task/Model

Sr. No.	Algo. Name	Task Splitting Tech. / Sche. Policy/ Used Technique
1	RM-US	Categories tasks as heavy light based on certain threshold
2	SM-US	Categories tasks as heavy light and priority order is given by SM
3	HSA	Uses heuristic function to search tasks to be scheduled
4	MA	It focuses attention on a small subset of tasks with the shortest deadlines
5	SAN	Described that feasibly schedules a set of m periodic tasks on n processors before their respective deadlines, if the task set satisfies certain condition
6	OFPSA	Discusses the problem of scheduling periodic real time tasks

Table 2: Comparative study of various real time multiprocessor algorithms meant for Parallel Task/Model

Sr. No.	Algo. Name	Task Splitting Tech. / Sche. Policy/ Used Technique
1	GSS	It uses decreasing chunk sizes, where smaller chunks improve the balance of workload towards end of loop.
2	ULTS	Modeled multiprogramming with two levels at user level on to a fixed collection of processes and at kernel level on to a fixed collection of processors.
3	NCS	Use resource augmentation in terms of allowing a faster processor to the online algorithm to make up for its lack of knowledge of job sizes
4	ASPF	Based on trim analysis technique.
5	HRTS-LSM	Uses hybrid approach for scheduling with hierarchical shared caches.
6	PRTTS	Uses concept of mega task
7	H-Pfair	It finds an approximate job partition on two processors
8	FBB-FFD	Provides sufficient conditions for feasibility, a resource augmentation approximation ratio, and simulation results
9	EDF-ID	Schedules independent parallel tasks with individual deadlines
10	Opt-Algo	Finds feasible schedule using fewest processors
11	NP-EDF	Number of used processors is defined before starting the system
12	GSA-ST	Uses job parallelism on identical parallel machines
13	Gang-EDF	Derives schedulability test on the basis of Global EDF schedulability test i.e. BAR test
14	FPP-FJS	Developed synchronous task stretch model using Fork-Join task sets
15	DP-FJS	Developed general parallel synchronous task stretch model using Fork-Join task sets

4. Conclusion

In this review paper, various research papers have studied on the basis of various approaches such as task splitting techniques, scheduling policies and techniques used for comparing real time task scheduling in multi-core processors and are categorized as Sequential Real-Time Task based Research and Parallel Real-Time Task based Research. Even if extensive work done currently in this area, a tremendous scope for research on parallel models is still there. This review of existing algorithms is to reveal the research challenges existing in this field of real time task scheduling specifically on multi-core processors.

References

[1] "AMD sets the new standard for price, performance, and power for the datacenter," AMD Press Release, March 2010.
[2] "Single-chip cloud computer," Intel Research, Dec 2009.
[3] "Teraflops research chip," http://techresearch .intel. com/ProjectDetails.aspx ?Id = 151.
[4] "CoSy compiler for 96-core multi-threaded array processor,"http://www.clearspeed.com /newsevents/ news/ClearSpeedAce 011708.php.
[5] "Comming soon tile-gx100 the first 100 cores processors in the world," http:// internalcomputer. com/coming-soon-tilegx100-the-first-100-cores-rocessor-in-the-world. computer , Feb 2011.
[6] H.-M. Huang, T. Tidwell, C. Gill, C. Lu, X. Gao, and S. Dyke, "Cyber-physical systems for real-time hybrid structural testing: a case study," in ICCPS '10.
[7] "OpenMP," http://openmp.org.
[8] D. Lea, "A java fork/join framework," in Proceedings of the ACM 2000 Java Grande Conference, p. 3643, June 2000.
[9] Adrien Lamothe. Pthreads programming: A hands-on introduction, 2007.

[10] "IntelR CilkTMPlus," http:// software. intel. com/ en-us/articles/intel-cilk-plus.
[11] R. Davis and A. Burns, "A survey of hard real-time scheduling algorithms and schedulability analysis techniques for multiprocessor systems," University of York, Department of Computer Science, Tech. Rep. YCS-2009-443, 2009.
[12] S. K. Dhall and C. L. Liu, "On a real-time scheduling problem," OPERATIONS RESEARCH, vol. 26, 1978.
[13] K. Ramamritham, J. Stankovic, and P. Shiah, "Efficient scheduling algorithms for real-time multiprocessor systems," IEEE Transactions on Parallel and Distributed Systems, vol. 1, pp. 184–194, 1990.
[14] A. Khemka and R. K. Shyamasundar, "An optimal multiprocessor real-time scheduling algorithm," J. Parallel Distrib. Comput., vol. 43, no. 1, pp. 37–45, 1997.
[15] M. Dertouzos and A. Mok, "Multiprocessor online scheduling of hard-real-time tasks," IEEE Transactions on Software Engineering, vol. 15, pp. 1497–1506, 1989.
[16] J. Y.-T. Leung and J. Whitehead, "On the complexity of fixed-priority scheduling of periodic, real-time tasks," Performance Evaluation 2, p. 237250, Dec 1982.
[17] J. Carpenter, S. Funk, P. Holman, A. Srinivasan, J. Anderson, and S. Baruah, "A categorization of real-time multiprocessor scheduling problems and algorithms," in Handbook on Scheduling Algorithms, Methods, and Models. Chapman Hall/CRC, Boca, 2004.
[18] C. D. Polychronopoulos and D. J. Kuck, "Guided selfscheduling: A practical scheduling scheme for parallel supercomputers," IEEE Transactions on Computers, vol. C-36, no. 12, pp. 1425–1439, 1987.
[19] N. S. Arora, R. D. Blumofe, and C. G. Plaxton, "Thread scheduling for multiprogrammed multiprocessors," in SPAA '98.
[20] N. Bansal, K. Dhamdhere, J. Konemann, and A. Sinha, "Nonclairvoyant scheduling for minimizing mean slowdown," Algorithmica, vol. 40, no. 4, pp. 305–318, 2004.
[21] K. Agrawal, Y. He, W. J. Hsu, and C. E. Leiserson, "Adaptive task scheduling with parallelism feedback," in PPoPP '06.
[22] J. M. Calandrino and J. H. Anderson, "Cache-aware realtime scheduling on multicore platforms: Heuristics and a case study," in ECRTS '08.
[23] J. M. Calandrino, J. H. Anderson, and D. P. Baumberger, "A hybrid real-time scheduling approach for large-scale multicore platforms," in ECRTS '07.
[24] J. M. Calandrino, D. Baumberger, T. Li, S. Hahn, and J. H. Anderson, "Soft real-time scheduling on performance asymmetric multicore platforms," in RTAS '07.
[25] J. H. Anderson and J. M. Calandrino, "Parallel real-time task scheduling on multicore platforms," in RTSS '06.
[26] C.-C. Han and K.-J. Lin, "Scheduling parallelizable jobs on multiprocessors," in RTSS '89.
[27] O.H. Kwon and K.-Y. Chwa, "Scheduling parallel tasks with individual deadlines," Theor. Comput. Sci., vol. 215, no. 1-2, pp. 209–223, 1999.
[28] Q. Wang and K. H. Cheng, "A heuristic of scheduling parallel tasks and its analysis," SIAM J. Comput., vol. 21, no. 2, pp. 281–294, 1992.
[29] N. Fisher, S. Baruah, and T. P. Baker, "The partitioned scheduling of sporadic tasks according to static-priorities," in ECRTS. IEEE Computer Society, 2006, pp. 118–127.
[30] K. Jansen, "Scheduling malleable parallel tasks: An asymptotic fully polynomial time approximation scheme," Algorithmica, vol. 39, no. 1, pp. 59–81, 2004.
[31] W. Y. Lee and H. Lee, "Optimal scheduling for real-time parallel tasks," IEICE Trans. Inf. Syst., vol. E89-D, no. 6, pp. 1962–1966, 2006.
[32] G. Manimaran, C. S. R. Murthy, and K. Ramamritham, "A new approach for scheduling of parallelizable tasks in realtime multiprocessor systems," Real-Time Syst., vol. 15, no. 1, pp. 39–60, 1998.
[33] S. Collette, L. Cucu, and J. Goossens, "Integrating job parallelism in real-time scheduling theory," Inf. Process. Lett., vol. 106, no. 5, pp. 180–187, 2008.
[34] S. Kato and Y. Ishikawa, "Gang EDF scheduling of parallel task systems," in RTSS '09.
[35] K. Lakshmanan, S. Kato and R. R. Rajkumar, "Scheduling parallel real-time tasks on multi-core processors," in RTSS '10.
[36] Abusayeed Saifullah, Kunal Agrawal, Chenyang Lu, and Christopher Gill, "Multi-core Real-Time Scheduling for Generalized Parallel Task Models", in RTSS'11.
[37] C. L. Liu and J.W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," J. ACM, vol. 20, no. 1, pp. 46–61, 1973.
[38] S. Funk, J. Goossens, and S. Baruah, "On-line scheduling on uniform multiprocessors," in Proceedings of the IEEE Real-Time Systems Symposium (December 2001), IEEE Computer. Society Press, 2001, pp. 183–192.

Mahesh Lokhande was born in Akola, India. He completed B.E. in Electronics and Communication from SGB Amravati University Amravati (MS), and M.Tech. in Information Technology from RGTU, Bhopal.

He is presently working as an Assistant Professor in the Department of Electronics and Communication Engineering at JIT, Borawan of RGTU, Bhopal. He has versatile teaching, administrative and industrial experience over 13 years. He has published some research papers in national/international journals and conferences. His research interests include Real Time Operating Systems and Image Processing. He is a life member of ISTE New Delhi.

Dr. Mohammmad Atique has completed Ph.D. in Computer Science and Engineering from SGB. Amravati University (MS) in the area of Soft Computing. He is presently working as an Associate Professor at SGB Amravati University, Amravati. He is a member of International Neural network Society (U.S.A.), Fellow IETE New Delhi, Life Member of ISTE, New Delhi, Sr. Member CSI, Mumbai and Fellow IE, Kolkata. He has vast teaching and administrative experience over 23 years at both college and university levels. He has published many research papers in national/international journals and conferences. His main research area includes Soft computing, Real time operating Systems, Artificial Neural Network and Machine Intelligence.