**Improved schedulability analysis of EDF on multiprocessor platforms[2005]**

Here, in this paper the problem of schedulability analysis of sporadic tasks in global scheduling, with EDF scheduler algorithm is discussed. They provide two contributions. First, the two recently proposed test are poorly working in presence of heavy sets and neither tests dominates other. Second, a new schedulability test is introduced that improves percentage of accepted task sets.

INTRODUCTION

Computational power can be increased by using more than one processor rather than powerful single processor. FPGA based system allow developers to choose number of cores, buses and connections to memory and peripherals. A future is there where a real time application will dynamically and adaptively scale from single-processor to multiprocessor implementation, depending on desired performance and quality of service. The problem of scheduling real-time tasks on m processors can be solved by: partitioning tasks to processors , or with global scheduler. In first case, tasks are partitioned at offline to all the processors and they can be treated as m single processor scheduling problems, whose optimal solutions are known. This is analogous to bin-packing problem .This is simple and efficient with apriori and fixed task set, partitioning approaches is used.

Task migration happens if a task moves from one processor to another during execution. For these type of tasks, global shared queue for all processors is maintained. P-fair class of algorithms, based on concept of quantum: rime line divided into equal size called quanta, and at each quanta scheduler allocates tasks to processors. Disadvantage, all processor need to synchronize when scheduling decision taken and also with small quantum, overhead (migration and context switches) is high.

Regarding EDF , one has proposed schedulability test based on utilization bound assuming tasks have relative deadlines equal to period, second model to tasks with deadline less than or equal to period. Some negative result s of EDF scheduler: - loses its optimality on multiprocessor, migration overhead taken into account.

In some architecture with no cache and simpler structure, migration overhead is lower. In theory, FP or EDF algorithm can solve problem of scheduling.

CONCLUSION

Showed that BAK test does not dominate GFB test.They proposed BCL test, improving percentage of accepted tasks when considering heavy tasks.GFB is effective for light tasks , whereas BCL for heavy tasks.

**An Optimal Real-Time Scheduling Algorithm for Multiprocessors [2006]**

Here they introduce an optimal algorithm meeting all task deadlines as well as utilization demand not exceeding the utilization capacity of all processors. The algorithm named LLREF is based on fluid scheduling model and fairness notion using T-L plane (Time and Local remaining execution time plane) to describe fluid schedules without using time quanta, unlike Pfair algorithm. So scheduling is represented as repeated occurring of T-L planes and feasibly scheduling on single T-L Plane results in optimal schedule. Also it has bounded overhead where bound is independent of time quanta.

INTRODUCTION

Multiprocessor architectures are desirable for embedded systems with high computational workloads, where additional, cost-effective processing capacity is needed.

One aspect of multiprocessor scheduling is degree of run-time migration allowed for job instances of task across processors. The types of migration are: 1) full migration, where jobs are allowed to migrate during their execution, this indicates global scheduling strategy, maintaining single shared global queue and processor wide scheduling decision made by global scheduling algorithm, 2) no migration, where jobs are statically partitioned across processors. At run-time, job instances are scheduled on respective processor through local scheduling algorithm, and 3) restricted migration, where jobs can migrate only at boundaries.

Partitioned approach trumps global approach in two ways. First, once task allocated to processor, they become single processor problems whose optimal solution exist. Second, not migration leads to reduced overhead. If task set is fixed and known a-priori, partitioned approach gives appropriate solutions.

The advantages of Global scheduling on partitioned approach are 1) for real-time tasks entering and leaving the systems, it may become necessary to reallocate tasks to processors in partitioned approach. 2) Partitioned approach does not produce optimal solution for periodic task sets, since this problem is analogous to bin-packing problem which is NP- hard. 3) In architectures with no cache and simpler structure, migration overhead does not have much significance

Pfair algorithm with full migration and fully dynamic priorities have shown to achieve schedulable utilization bound that equals total capacity of all processors. However, they incur significant runtime

overhead due to quantum based scheduling approach: under Pfair, tasks are decomposed in small uniform segments, scheduled causing frequent scheduling and migration.

This figure shows example task set which global EDF we cannot schedule. Here, T1 misses its deadline when system is given two processors. The second shows where all task meet their deadline. The figure shows that we need more scheduling events to split tasks to construct optimality, like done by Pfair quantum based approach.

So, here LLREF is introduced based on T-L Plane envisioning us that scheduling is just repetition of T-L planes over various sizes, so feasible schedule in one T-L plane implies feasible scheduling over all times. Also, they define two scheduling events and show their occurrence to maintain fairness of optimal schedule, and establish scheduling optimality.

CONCLUSION

We present all the properties of LLREF as discussed above. We also show that algorithm overhead is bounded in terms of number of scheduler invocations.LLREF guarantees local feasibility in T-L Plane as algorithm first selects tokens which appear to be going out of the T-L plane because they are closer to NLLD.

The two theorems proposed here are reason for establishing local feasibility in-L plane. Their proofs are directly related to LLREF in two ways: one is that they depend on critical moment, and other is range allowed for Sj-1 is computer under assumption of LLREF, where j is c or b.So, scheduling policies are locally feasible as long as they maintain critical moment concept. The rules of scheduling policies are: 1) selecting as many as tokens as possible up to M at every secondary event, and 2) including all tokens on NLLD for selection at every secondary event. So we could be just selecting all tokens on NLLD at every event.(i.e. if number of tokens on NLLD is less than M, any token that is not on NLLD can be selected).Its computational complexity is as same as LLREF since it omits sorting process instead it just needs more queue operations.

Based on algorithm complexity and frequency of event C, algorithm can be selected. It is considered that each task’s parameters including execution times and periods are larger than system clock ticks, so they can be assumed floating point numbers but it may not be sufficient when task execution time is shorter so integer execution times are better assumption. But even with integer task parameters certain optimality can be achieved e.g. - error between each task’s deadline and its completion time is bounded within finite number of clock ticks. Many other aspects such as arrival model, time constraints and execution time model can be relaxed.

**Multiprocessor Scheduling Using Hybrid Particle Swarm optimization with Dynamically Varying Inertia [2007]**

Developed a new hybrid approximation algorithm. It is a static allocation of tasks involving Particle Swarm Optimization and Simulated-Annealing which minimizes cost.PSO with dynamically reducing inertia yields better results than fixed inertia. This method gives efficiency in finding optimal results.

INTRODUCTION

This paper offers static allocation of application whose tasks have diverse execution requirements. Advantages include no runtime overhead, can be designed using complex algorithmic mechanisms which fully utilize known properties of given application. This method avoids local optimum and fast convergence to targeted solution.

CONCLUSION

Several solution to TAP problem have been defined but most of them are NP-complete. This algorithm finds near optimal assignment with reasonable time. Hybrid PSO performs better than local PSO and global PSO.

**A Flexible Real-Time Locking Protocol for Multiprocessor [2007.1]**

For previously discussed scheduling algorithms to be useful, support for semaphore-based locking is needed. For global scheduling, no mechanism for this is proposed and for partitioned, most prior semaphores are inefficient and restrict critical sections. Here, a new flexible locking protocol is introduced applied to both partitioned and global scheduling. This allows unrestricted critical-section nesting, but designed to deal with short-nested accesses efficiently.

To support workload of real-time applications on multicore platforms, efficient multiprocessor-based techniques for scheduling and synchronizing tasks are needed. There do not exist adequate synchronization methods for multiprocessor scheduling approach. Here we look on this issue and specifically our emphasis is on lock-based synchronization provided via semaphores.

The shortcoming in previously proposed real-time multiprocessor locking schemes.1) I an all schemes, partitioned scheduling is assumed but global approaches are getting much attention now so global approaches must be extended to provide support for locking. 2) Previously, they impose restrictive assumptions like nested accesses of global critical sections are forbidden.3) Most prior schemes are inefficient when implementing nob0nested accesses.

So the locking scheme here overcomes all these shortcomings.

In prior work, three basic approaches to scheduling a sporadic task system is proposed: partitioning, Pfair-based global scheduling and non-Pfair based global scheduling. Here, in this paper we consider deadline-based scheduling algorithm in partitioned case, we consider partitioned EDF wherein EDF is used on each processor. In global case, we consider non-Pfair global EDF and Pfair PD2  algorithm.

Previously priority ceiling protocol (PCP) and stack-based resource allocation protocol (SRP) were considered for uniprocessor synchronization. They both prevent deadlock and limit duration of priority inversions (which occur when a job is blocked by job of lower priority) to be at most the length of one outermost critical section.

Previously two multiprocessor variants of PCP were presented for systems where partitioned, static priority scheduling is used. Later, several related protocols were presented for systems scheduled by P-EDF.Later, implementation of SRP for P-EDF was presented where tasks share common resource must be assigned to same processor. Another implementation of SRP for P-EDF shows that when task waits for resource to be released held by a task on another processor, it busy-waits, preventing any useful work being done on processor. In work involving global scheduling, approaches for implementing non-nested locks have been presented for PD2 and for G-EDF.

SO the multiprocessor synchronization protocol, called Flexible multiprocessor locking protocol(FMLP) provide contributions in three ways.1) it is first to be optimized to execute non-nested resource accesses more efficiently.2)It is first to be applied to G-EDF(a variant of G-EDF that allows jobs to suspend and become non-preemptable) and PD2 in addition to P-EDF.3)It supports nested resource accesses without constraining limitations and allows short critical sections to be implemented using busy-waiting mechanisms and long-waiting sections implemented by suspending blocked tasks.

CONCLUSION

The FMLP is capable of being implemented on systems scheduled by P-EDF, G-EDF and PD2 .It is first to be adapted for use under both partitioning and global scheduling algorithms and also the first to be optimized for short non-nested resource requests, still allowing for other types of resource accesses. Here, it is shown that FMLP, particularly, G-EDF variant of FMLP, has superior performance with respect to schedulability that previous multiprocessor locking schemes. They have also proposed GSN-EDF scheduling algorithm, which is used to bound impact of non-preemptive blocking in globally-scheduled systems with tasks that can become non-preemptable and suspend at arbitrary points in time, even if it does not require synchronization. In future, they will compare performance difference between classifying resource as short or long, and compare FMLP performance to other non-semaphore based synchronization mechanisms, such as lock-free and wait-free approaches.

**Hierarchical Scheduling Framework for Virtual Clustering of Multiprocessors [2008]**

Here, they consider approach called cluster-based scheduling. Here, each task is assigned to processor cluster, tasks in each cluster is globally scheduled among themselves and clusters are scheduled on multiprocessor platform. Here they develop techniques to support this algorithm, and consider properties that minimize processor utilization of individual clusters.

INTRODUCTION

Here we consider approach using notion of cluster. A cluster is set of m’ processor where 1<=m’<=m with m processors in total. Here, tasks are assigned to cluster and globally scheduled under a cluster. It is combination of both partitioned and global scheduling as we assign tasks to m cluster of size one, and on other end it assign tasks to single cluster of size m. It is further classified in two types: Physical cluster, where it holds static one-to-one mapping between its m’ processors and some m’ out of m processor in platform while Virtual cluster, allows dynamic one-to-many mapping from m’ processors to some m0 out of m processors in platform. Scheduling in virtual cluster is equivalent to scheduling them globally on m0 processors with amount of concurrency at most m’.So, physical cluster does not share processors while virtual cluster can.

Example: Consider a sporadic task set with t1=t2=t3=t4=(3,2,3) and t5=(6,4,6) and t6=(6,3,6). The notation (T, C, D) with T minimum separation, C as worst-case execution requirement and D relative deadline. Let it be scheduled on 4 processors and it can be seen under partitioned scheduling it is not schedulable as no processor can be allocated more than one task. The task is not schedulable with any algorithm other than cluster based scheduling as shown in figure. In cluster-based scheduling, t1, t2 and t3 are executed on cluster C1 with 2 processors under f=global LLF, and tasks t4, t5, and t6 executed on cluster C2 with 2 processors using global EDF.



Virtual Clustering is less sensitive to task-cluster mapping. For example, with task t7= (6, 1, 6). If t7 is assigned to first cluster C1 with tasks t1, t2 and t3 then physical based clustering cannot accommodate 2 clusters on 4 processors while virtual-based clustering can accommodate them on 4 processors by dynamically allocating slack in second cluster C2 to first cluster C1.

It can also be used to restrict on amount of concurrency. Suppose m tasks can thrash L2 cache if they run parallel at same time, so one may force on some of m tasks to run in parallel to prevent thrashing of L2 cache. It can be easily done with m tasks assigned to cluster of m’ processors.

In physical clustering, there is intra-cluster scheduling as clusters are assigned to disjoint processors whereas virtual clustering supports hierarchical scheduling: intra-cluster as well as inter-cluster. Under inter-cluster, physical processors are dynamically assigned to virtual clusters, and under intra-cluster processor allocations given to a virtual cluster are assigned to tasks in that cluster. Let a task be divided into three clusters employed using global EDF strategy. Now in physical clustering, each cluster can analysed with techniques of global EDF.But in virtual clustering, there is also need to develop techniques to schedule C1, C2 and C3 on multicore platforms.

Previously, in uniprocessor scheduling, there is a notion of component interface, to specify resources required for scheduling component’s (collection of tasks and scheduler) tasks. So, here they denote cluster along with tasks and scheduler assigned to it as component in hierarchical multiprocessor scheduling framework. So, here it proposes component interface specifying resources required to schedule tasks in component’s cluster. So, after this inter-cluster scheduler can use component interface to allocate processor supply, and then intra-cluster scheduler can use this processor supply to schedule tasks in cluster. The issues faced here are confusion to enable component interface to carry information about concurrent execution of tasks in component. Example, if a single task cannot execute in parallel then multiple processors cannot be used concurrently to satisfy execution requirement of this task. So, here a solution to this is provided. Here, we include all concurrency constraints of tasks in component in component’s interface. So, interface can demand enough processor supply from inter-cluster scheduler so intra-cluster scheduler can handle task-level concurrency constraints, freeing inter-cluster scheduler from all worries.

Here, first we introduce hierarchical multiprocessor scheduling framework to support virtual-based clustering. Then introduce multi-processor model based interface that captures task-level concurrency and specifies total resource requirements of component, so inter0cluster scheduler can schedule clusters using their interfaces only. We extend existing schedulability conditions for global EDF multiprocessor scheduling which will help in supporting development of component interfaces. Then, we optimize problem of developing component interface with aim to minimize its total resource requirement. On basis of global EDF schedulability condition, there is a solution: processor utilization required by component interface increases as number of processors allocated to component’s cluster increases. So, optimal solution is when there are smallest number of processors with guarantee of schedulability of component.

CONCLUSION

Here, the resource requirements and concurrency constraints within each cluster are abstracted into MPR interfaces, each transformed into periodic tasks used for inter-cluster scheduling. Also they have developed technique to minimize processor utilization of individual cluster.

The concept of isolating inter-cluster scheduler form concurrency constraints can be applied to other scheduling algorithms as well and this empowers clusters with different intra-clusters scheduler to be scheduled on same platform.

Instead of MPR model based component interface, we can also consider general explicit deadline resource model for multiprocessors similar to EDP model for uniprocessor platform.

**Priority Assignment for Global Fixed Priority Pre-emptive Scheduling in Multiprocessor Real-Time Systems [2009]**

This paper proves that Audsley’s Optimal Priority Assignment (OAP) algorithm, devised for uniprocessor scheduling is applicable to multiprocessor with three conditions hold with respect to schedulability tests. It is also shown that combination of optimal priority assignment policy and simple compatible schedulability test is highly effective in terms of number of tasksets deemed to be schedulable. Here also the performance of assignment policies such as Deadline Monotonic, and extension of TkC priority assignment policy called DkC is examined. So, the result provides that Deadline Monotonic has poor performance in multiprocessor case while DkC is highly effective.

INTRODUCTION

Scheduling is classified into global and partitioned scheduling. Real-time scheduling algorithms are categorised into three classes based on when priorities can change: fixed task-priority, fixed-job priority and dynamic-priority. Here, we focus on global fixed task-priority pre-emptive scheduling also known as global FP scheduling.

The three fundamental results regarding priority in case of uniprocessor fixed priority scheduling are: 1)Rate Monotonic priority Ordering (RMPO) is optimal for independent synchronous periodic tasks(share common release time) with implicit deadlines(equal to their periods ) .2) Deadline Monotonic Priority Ordering is optimal for independent synchronous tasks with constrained deadlines(less than or equal to periods).3) An optimal priority algorithm solving the problem of priority assignment of asynchronous tasksets, for tasks with arbitrary deadlines(which may be greater than their periods).

Here global FP scheduling is based on “Dhall effect”.Dhall showed that under global FP scheduling under RMPI, set of periodic tasks with implicit deadlines and total utilisation greater than 1 can be unschedulable on m processors. Later, TkC priority assignment policy was developed to circumvent Dhall effect as it assigned priorities based on a task’s period (Ti) minus k times its worst-case execution time (Ci), where k is real value computed on basis of number of processors.

Later on, with the help of RM-US {¥} priority assignment policy, which gives higher priority to tasks with utilization greater than ¥, utilization bound for global FOP scheduling of periodic tasksets with implicit deadlines was given. It was shown that maximum utilization bound for global FP scheduling of tasksets is (√ (2-1) m), when priorities are defined as scale-invariant function of worst-case execution times and periods. Later, for sporadic tasksets with constrained deadlines DM-DS {¥} priority assignment was given which gives highest priority to atmost m tasks with densities greater than threshold ¥ .Later, Slack Monotonic priority assignment called SM-US {¥} was given where using threshold of 2/ (3+√5), it gave utilisation bound of 2/ (3+√5) m for sporadic tasks with implicit deadlines.

Global multiprocessor scheduling is difficult than uniprocessor as task can use only one processor at a time, even when several are free. This restriction manifests itself as critical instant effect where simultaneous release of tasks does not necessarily lead to worst-case response times. There are no exact tests for global scheduling of sporadic tasks, only known for periodic test case.

Here, it is found that priority assignment is the key for known approaches applied on multiprocessor scheduling on sporadic tasks with constrained deadline to be effective under feasibility test.It is very important to find appropriate priority ordering for fixed priority scheduling. It has been shown that DMPO , the response time test for global FP scheduling for sporadic tasks with constrained deadlines for global scheduling outperformed all other test including those for global FP , global EDF, and EDZL.But DMPO is not optimal for multiprocessors despite being on uniprocessors.

Here schedulability test for global FP scheduling is categorised into two: OPA-compatible and OPA-incompatible. Optimal priority assignment with OPA schedulability test make more tasksets schedulable, than using OPA-incompatible schedulability test with DMPO.Also heuristic priority assignment policies: CMPO and DkC are developed. Here, it is shown that DkC significantly outperforms DMPO, giving close to optimal results.

CONCLUSION

Here, it is proved that OPA is optimal priority assignment policy with respect to any global FP schedulability test for periodic or sporadic tasksets that complies with 3 conditions. The deadline-based sufficient test (“DA test”) and response time test for sporadic tasksets are OPA-compatible, while any exact test for periodic tasksets are OPA-incompatible. We have also adapted UUnifast algorithm to multiprocessor case, giving UUnifast-Discard algorithm.UUnifast-Discard generates tasksets with specific parameter settings, facilitating empirical study of schedulability test effectiveness without problem of cofounding variables.DA test can schedule more tasksets by using OPA algorithm rather than DMPO and DkC priority assignment policy is effective when applied in conjunction with DA test, and highly effective when applied with RTA test.

It is seen that with increasing processing capacity improvements are large with appropriate priority assignment .Example, in 16 processor case, utilisation level at which 0.5 of the tasksets were schedulable increased from 0.28m/0.29m (for DA/RTA test with DMPO) to 0.58m/0.59m (for RTA test with DkC or DA test with optimal priority assignment).There is an increase in usable processing resource of 100% or more. The OPA algorithm requires polynomial number of schedulability test (n (n+1)/2) to solve problem of optimal priority assignment for any OPA-compatible global FP schedulability test.

**Semi-Partitioned Scheduling of Sporadic Task Systems on Multiprocessors [2009.1]**

A new algorithm for scheduling g of sporadic task system with arbitrary deadlines on identical multiprocessor platforms. Semi-structured scheduling, where most tasks remain fixed to processor and some migrate. Tasks migrate once in each period and only if they cannot be partitioned any more. This is then subject to EDF. This algorithm shows smaller number of context switches.

INTRODUCTION

Manufacturers have shifted to multiprocessor due to heating problems in single one and power consumption.

2 types of scheduling: Partitioned Scheduling, tasks first assigned to the processors and then executed without migration.

Global Scheduling, tasks stored in global queue, number of highest priority tasks equal to number of processor are executed.

Advantages of Partitioned Scheduling include reducing problem into that of uniprocessor and no runtime overhead as there is no migration. Disadvantage is that deadline to be missed on m processors, if total processor utilization exceeds (Bm+1)/ (B+1) where B=floor (A) where A is maximum utilization of individual tasks. For A=1 and m-> 0 worst-case utilization is bounded by 50%.

In worst-case, global scheduling like PFAIR and LLREF are known optimal algorithms. They can schedule successfully, if processor utilization <=100% but there are number of context switches and migrations. There are also algorithms, EDZL and EDCL, perform with less context switches but worst-case processor utilization is still 50%.

Semi-structured scheduling helps in reducing number of migrations while few tasks may migrate to improve processor utilization. Benefits of these algorithm:-

* Allow tasks to migrate if they cannot be assigned to any individual processor, dominating previous classical partitioned scheduling.
* Allows migratory tasks to migrate only once in each period, to bound number of context switches to be smaller than complementary algorithms. This keeps runtime processor performance as much as possible.
* Performs scheduling policy to EDF.
* Available for all categories of periodic and sporadic task systems with implicit, constrained and arbitrary deadlines.

CONCLUSION

According to results, this algorithm offers competitive schedulability to best-known algorithm designed for arbitrary-deadline systems, with smaller number of context switches.

**Scheduling Parallel Real-Time Tasks on Multi-core Processors [2010]**

Sequential programming model does not scale well with multi-core platforms. Parallel programming platforms like OpenMP provide effective solutions.Here,we study real-time scheduling on multiprocessors under fork-join structure used in OpenMP.We also provide partitioned pre-emptive fixed-priority scheduling algorithm for periodic fork-join task sets. This algorithm has resource augmentation bound of 3.42

INTRODUCTION

For faster and faster processor clock speeds ,sequential programming model was suitable , but with more cores oncoming it started to diminish in comparison to inability to take advantage of parallelism.OpenMP have capability to parallelize shorter segments of tasks , leading to shorter response times when possible.

Most scheduling theory are based on sequential programming model but parallel programming brings a new dimension, where jobs are split into parallel execution segments at specific points. Here fork-join model by OpenMp is considered.OpenMP is mature system for parallel programming, and can play pivotal role in shaping future programming paradigms for multi-core processors.

Each basic fork-join task begins as master thread that executes sequentially until it encounters fork construct, splitting into multiple parallel threads. After this parallel execution, join construct synchronizes and terminates parallel threads. This can be repeated multiple times. Here nested fork-join structures are not focused. Also it is assumed that parallel execution regions can be pre-empted individually on respective processors.

Real-time systems like radar tracking, autonomous driving and video surveillance exhibits data-parallel model. Here, we study scheduling of periodic real-time tasks with implicit deadline employing fork-join parallelization construct. Also fixed-priority pre-emptive algorithms are focused because easy support on Linux and Windows, POSIX and real-time specification for Java. Also OMP\_NUM\_THREADS variable in OpenMP specifies static number of threads.

This paper contributes: best-case and worst-case fork-join task sets, task set transform to reduce scheduling penalty, task partitioning algorithm for deadline-monotonic scheduling.

CONCLUSION

Here, it is shown that worst-case schedulable utilization for fork-join task sets on multiprocessor can be slightly gr=eater than and close to 100%. Even with large number of core. Also, a task stretch transform is defined to avoid fork-join structures as much as possible. These can be scheduled using partitioned pre-emptive fixed-priority multiprocessor scheduling algorithms. Pre-empting parallel threads during execution may result in better system utilization but it also leads to overheads such as cache pollution and synchronization delays.

**RUN: Optimal Multiprocessor Real-Time Scheduling via Reduction to Uniprocessor [2011]**

There are signification overhead for pre-emption and migration. RUN reduces multiprocessor to uniprocessor. It has an upper bound of O(log m ) average preemptions for job on m processors and reduces to Partitioned EDF whenever proper partitioning is found.

INTRODUCTION

Several solutions for multiprocessor real-time scheduling based on periodic-preemptive-independent (PPID) tasks with implicit deadlines is presented. Partitioned approaches achieve 50% utilization but global achieve full utilization through migration. Example, consider two processor with three tasks, T1, T2 and T3, each requiring 2 units of work with 3 units of time. Two tasks scheduled on two processors makes third to miss its deadline. McNaughton’s wrap-around algorithm, involves migration works whenever all jobs have same deadline.



Most optimal algorithms rely on proportional fairness and McNaughton’s algorithm. They enforce deadline equality by proportionally subdividing workloads and imposing deadline of each tasks on all other tasks causing many tasks to execute between every two system deadlines, leading to context switching and migration overheads. RUN has weak proportional fairness and lower overhead.

The Approach is that a real-time task is presented as infinite sequence of jobs. Each portion of work has its deadline r and release time t between which it has to be executed. They are independent, fully preemptable, not periodic i.e. tasks have fixed rate and job of task with rate u <=1 requires u(r-t) execution time.

RUN produces schedule for fixed-rate tasks with few preemptions per job, ignoring number of processors, jobs, tasks, averaging less than 3 preemptions per job. RUN 1) reduces multiprocessor to uniprocessor scheduling through two operation: DUAL and PACK 2) solve with well-known techniques 3) transforming solutions back to multiprocessor.

3-task example:-It creates dual task set {t1\*,t2\*,t3\*} where ti\* has same deadline as ti and complementary workload of 3-2=1.Ti\* represents Ti’s idle time, Ti\* executes exactly when Ti is idle. These 3 tasks can be scheduled to execute on single dual processor before their deadline is at 3.For original task set, schedule is be blocking Ti when Ti\* executes in dual schedule.

RUN performs sequence of transformations, iteratively reducing number of uniprocessor systems. Its uses EDF and iteratively unpacks the solutions. Since dual and its primal task cannot execute simultaneously, it constrains the task that may execute next level up. Also there are as many tasks selected for executions as processors.

CONCLUSION

RUN outperforms prior optimal algorithms as only few pre-emption points per job are generated on average. RUN reduces to more efficient partitioned approach of EDF whenever worst-fit bin-packing finds partition, and it scales well as number of tasks and processors increase. The overhead of RUN is low enough to justify implementation on multiprocessor architectures and at present, it works for fixed-rate task sets with implicit deadlines.

**Priority Based Dynamic Round Robin (PBDRR) Algorithm with Intelligent Time Slice for Soft Real-time Systems [2011.1]**

Here, a new variant of Round Robin algorithm is proposed suitable for soft real-time systems. As it gives larger context switches, larger waiting time and larger response time, despite performing good in timeshared systems, it is not suitable for soft real time systems. So, Priority Based Dynamic Round Robin algorithm (PBDRR), calculates intelligent time slice for individual processes and changes after every round of execution. It works on dynamic time quantum concept and performs better in terms of reduced number of context switches, average waiting time and average response time.

INTRODUCTION

Real Time Systems must have well defined fixed and response time constraints and processing within defined constraints# types of RTS are there: Hard RTS, failure to meet deadline leads to system failure. Firm RTS, failure to meet deadline tolerated. Soft RTS, it doesn’t lead to system failure, but performance degrades. Space research, weather forecast, seismic detection, audio conferencing, video conferencing, money withdrawal from ATM, railway and flight reservation etc. are some of the applications of real time systems. Simple RR cannot be applied to Soft RTS because of longer waiting and response time. In an algorithm for soft RTS there are Intelligent Time Slice (ITS) for all processes used as time quantum. By taking dynamic time concept with ITS, a new algorithm is proposed.

Some well-Known real time scheduling:-

* Rate Monotonic Algorithm – It is apriori and fixed priority scheduling where higher frequency tasks get high priority and at any instance scheduler executes higher priority task.
* EDF –Here tasks with earliest deadline have highest priority
* Minimum Laxity-First –Laxity defined as difference of deadline for task completion and amount of computation remaining to be performed. It selects task with minimum laxity for next execution.
* Maximum-Urgency-First –Combination of fixed and dynamic scheduling. Task has urgency, defined as combination of two d=fixed priorities and dynamic priority. Criticality, a fixed priority is highest among three, a dynamic priority (inversely proportional to laxity of task) and then user priority (fixed priority).

Optimal dynamic are EDF and MLF algorithm but RM is fixed optimal algorithm. Scheduling mechanism of Maximum-urgency first may cause critical task to fail which is resolved by modified maximum-urgency-first.

So, they have proposed dynamic time quantum changing with every round of execution instead of taking static quantum. So PBDRR performs better than Modified RR in terms of context switches, average waiting time and average turnaround time.

CONCLUSION

This algorithm reduces overhead and saves memory spaces. Also, in future deadline can be considered as on input parameter in addition to priority in this algorithm.

**Multi-core real-time scheduling for generalized parallel task models [2012]**

Here, they address the problem of scheduling periodic parallel tasks with implicit deadlines. They consider a synchronous task model where each task consists of segments, each having number of parallel threads synchronizing at end of segment. Here in this method decomposes each parallel tasks into set of sequential tasks. This has resource augmentation bound of 4 and 5 when decomposed tasks scheduled with EDF and partitioned deadline monotonic scheduling. Afterwards, they consider each node in DAG unit as execution requirement.

INTRODUCTION

By intra-task parallelism, multi-core processors can achieve real-time performance improvement compared to single-core processors for computation-intensive real time applications like video surveillance, radar tracking and hybrid real-time structural testing.

The work on parallel scheduling of real-time task with intra-task parallelism analyses resource augmentation bound using partitioned Deadline Monotonic Scheduling. A resource augmentation bound under scheduling policy quantifies processor speed-up factor (how much to increase speed) w.r.t optimal algorithm to guarantee schedulability of task under policy. A synchronous task model, called synchronous, since all threads of parallel segment should finish before start of next segment, synchronization point. Also, all segments have equal number of parallel threads and execution requirements of all thread in segment are equal and number of threads in every segment is no greater than total number of processor cores. This work consider resource augmentation bound under partitioned DM only and not under other policies EDF.

Here, in contrast to previous work each segment can contain arbitrary number of parallel thread i.e. different segment s of parallel task can contain different number of threads and segments can contain more threads than number of cores. Also execution requirements of threads can vary. So, finally this model can work on both large as well as small number of cores.

In context of DAG, we show that we can transform unit-node DAG tasks into synchronous tasks, and then used proposed decomposition to obtain same resource augmentation bound as before.

By running simulations based on synthetic workloads, results of performance of proposed decomposition indicates safe and sufficient bounds. Also in these simulations , resource augmentation are at most 2.4 and 3.4 for EDF and partitioned DM scheduling respectively., smaller than corresponding theoretical bounds.

CONCLUSION

They can provide more better bounds. In future, they plan to consider more general DAG task where nodes have arbitrary execution requirements, and to provide analysis requiring no transformation to synchronous model. They would also address issues such as cache effects, pre-emption penalties and resource contention.

**A generalized parallel task model for recurrent real-time processes[2013]**

A recurrent task is DAG, a period and a relative deadline. Each vertex of DAG is sequential job, while edges of DAG present precedence constraints between these jobs. All jobs of DAG are released simultaneously and has to complete execution within relative deadline of their release. The task may release jobs in unbound manner with successive releases occurring at least specified period apart. So the problem here is to determine if such recurrent task can be scheduled to always meet all deadlines upon specified number of processors that are dedicated for use of this task.EDF is shown as good approximate scheduling algorithm. Polynomial and pseudo-polynomial schedulability test are presented for determining if given can task can be scheduled by EDF to always meet all deadlines on specified number of processors.

INTRODUCTION

Many real-time systems can be modelled as composed of finite number of independent recurrent processes or tasks, each of which generates potentially infinite sequence of jobs. The scaling trends in processor is moving from increasing clock speed to number of cores. An execution environment with individual tasks allowed to execute exclusively on dedicated cores allows for more expressive task models than relatively simple recurrent task models .These models may allow for partial parallelism within task and precedence dependencies between different parts of task.

Here, they study task model named sporadic DAG model. Each recurrent task is modelled as DAG, G=(V,E) executing upon m identical processors dedicated to exclusive use of particular task. Each vertex is sequential job characterized by worst-case execution time pv Each directed edge represents precedence i.e for (v,w) v must complete execution before w begins execution.Groups of jobs not constrained by precedence can execute parallel. There are also deadline parameter D and period T. Let a task release dag-job at time t when it becomes available for execution, so now subject to precedence constraints |V| jobs become available. All |V| jobs released at some time-instant t must complete execution by time-instant t+D.A minimum interval of duration T must elapse between successive releases of dag-jobs.

It is known that pre-emptive scheduling of given precedence-constrained jobs is NP-hard, which is seen to hold for sporadic DAG as well. In this paper, they extend to deal with scheduling of sporadic DAG tasks for which deadline parameter D is greater than period parameter T.They also show that “synchronous arrival sequence”, in which successive dag-jobs are released exactly period T time-units apart is not worst-case behaviour of sporadic DAG task. They also show that EDF has speedup bound no larger than 2 for scheduling sporadic DAG tasks. Two test with different run-time complexity – one with polynomial run-time while other with run-time pseudo-polynomial in representation of task- is ran on EDF schedule.

CONCLUSION

For exploiting multi-core computing capabilities, it is necessary that more expressive models capable of exposing parallelism within these workloads be used. This paper contains DAG based models with recurrent models used in real-time systems. By allowing for both dependencies and parallelism within workload, such DAG task model seems appropriate for modelling real-time workloads.

If deadline parameter of recurrent DAG task is no larger than its period, then at most one dag-job may be active at any instant of time. This led to development of efficient scheduling algorithms and schedulability test with good speedup bound. They have introduced concepts and techniques for dealing with tasks where deadline parameter may exceed the period and have applied these to analysis of sporadic-DAG tasks scheduled using EDF.

Some generalizations that are pertinent are: Each vertex may have different deadline parameter and release time specified; multiple sporadic DAG tasks may be present sharing given collection of processors.

**Harmonic-Aware Multi-Core Scheduling for Fixed-Priority Real-Time Systems[2014]**

New semi partitioned approach to schedule sporadic tasks on multicore platforms based on Rate Monotonic Scheduling Policy. The fact that utilization bound of a task set increases as tasks periods become closer to harmonic on single processor platforms is exploited. Here, we have to assign and split tasks in semipartitioned approach based on this fact and guarantee schedulability. This algorithm can schedule any task set with system utilization bounded by Liu&Layland’s bound for N tasks, which is, N(21/n-1).

INTRODUCTION

With more complicated applications,multi-core platforms come into picture to obtain high performance. It is well-known that scheduling real-time tasks on multiprocessor platform is NP-hard.

The optimal algorithms on uniprocessor platforms, Rate Monotonic Scheduling (RMS) and EDF are not optimal on multi-core platforms.

Here also most tasks are assigned to one particular processor, as done in partitioned approach whereas, few tasks(no more than (M-1) task with M processors) are allowed to split and get assigned to different processors. Also, by running semi-partitioned approach on Linux OS and on Intel Cor-i7 4-cores showed that overhead in task migration in relatively low. Here, semi-partitioned strategy is proposed based on RMS.

The contributions are: First, take harmonic relation among tasks for fixed-priority semi-partitioned scheduling strategy on multi-core platforms. Two, Harmonic semi-partition for light tasks (HSP-light) is intended for tasks sets with utilization no more than 0.5 for each task. HSP for tasks with utilization factor no more than 1. Also task sets with utilization higher than Liu&layland’s bound may be schedulable with this approach.

CONCLUSION

The proposed approach can take advantage of harmonic relations among task periods and improve schedulability.There is also a metric they introduced to quantify how close task set is to harmonic set. Also analysed and presented simple schedulability test method for HSP and HSP-light.Also, it is proved that scheduling algorithm can schedule any task set with system utilization bounded by Liu&Layland’s bound.

**PASS: Priority Assignment of Real-Time Tasks with Dynamic**

**Suspending Behaviour under Fixed-Priority Scheduling[2015]**

Self-suspension is getting prominent in real-time systems like, I/O intensive systems applications interacting with I/O devices, multicore-processors with tasks synchronizing and communicating with each other, and computation offloading systems with coprocessors like GPUs .Here, they show that rate-monotonic, deadline monotonic and laxity-monotonic perform poor in dynamic self-suspending systems in terms of speed-up factors. The proposed PASS approach shows feasible priority assignment on speed-2 uniprocessor, if one exists on unit-speed processor.

INTRODUCTION

Self-suspension is suspension of tasks by OS when accessing external devices like disks, GPU or synchronizing other tasks. It is more frequent in systems with more interaction with external and physical devices. This suspension may delay range from microseconds to milliseconds and this may have negative impact on timing predictability and cause intractability in hard real-time schedulability analysis.

It is hard for an optimal polynomial time solution to exist since HRT self-suspension on uniprocessor is NP-hard. If a scheduling algorithm has speed-up factor α, it guarantees to produce feasible schedule (all tasks meet their deadline) for given input task set on processor with speed α, if it admits feasible schedule on unit-speed processor. Here, we show that classical fixed-priority scheduling algorithms like rate-monotonic and deadline-monotonic cannot support self-suspending task systems. So PASS is introduced with strong performance in terms of non-trivial speed-up factors.

Previously, it has been shown that category of fixed-relative-deadline schedulers may yield non-trivial resource augmentation performance guarantees but it can only be applied to task model where each task is allowed to suspend at most once and has fixed interleaving pattern between computation and suspension phases. The approach of ILP-based priority assignment algorithm cannot be applied to general self-suspending task model, and may incur excessively high runtime complexity due to ILP-based solution. There are majorly two problems faced: 1) fully general self-suspending task model is not supported 2) quality of tests is unknown.

Here, first they prove that classical fixed-priority scheduling algorithms, RM, DM and laxity-monotonic has speed-up factor of ∞, meaning that their corresponding priority assignment schemes are ineffective when self-suspensions are present. Also, the proposed approach PASS is validated via GPU-offloading case study implemented on top of real hardware and extensive simulation results using traces and randomly generated workload parameters.

CONCLUSION

It is shown that there is no lower bound on processor speed-up factor for classical scheduling algorithms and PASS finds a feasible priority assignment on speed-2 uniprocessor, if one exists on unit-speed processor. The results of GPU offloading implementation and extensive simulations give positive indication for use of this practice.

**An unfair semi-greedy real-time multiprocessor scheduling algorithm [2016]**

Most algorithms adhere to fairness rule but results in large number of scheduling overheads affecting the practicality of algorithms. The tasks are executed proportion to their utilization at each time quantum or at end of each time slice in fluid schedule model. Here, the algorithm introduced produces very few scheduling overheads while maintaining high schedulability.It relaxes fairness rule and adopts new semi-greedy criterion instead. The reduction in scheduling overheads trumps the loss of missing few deadlines.

INTRODUCTION

The correctness of real-time system is producing output within deadlines, so an optimal scheduling is one which successfully schedules all tasks without missing deadline for any schedulable taskset.

Most algorithm achieve optimality by following fairness rule which forces task to make progress in their execution proportionate to their utilization. P-fair follows fairness rule. Deadline Partitioning algorithm like LLREF(Largest Local Remaining Executions First), LRE-TL(Largest Remaining Execution-Time and Local Time Domain) and DP-Wrap(Deadline Partitioning Wrap) partially follows fairness rule at end of each TL-Plane in fluid schedule model, corresponding to deadline of tasks in system. Fairness rule produces overheads in terms of tasks pre-emption’s and migrations which lead to processor being busy in execution of scheduler itself rather than actual work. It is found that pre-emption and migration delays can high as 1 ms on system with 24 cores running on 2.13 GHz with three levels of cache memory.

DP algorithms divide time into T-L planes bounded by two successive deadlines and end of each T-L plane corresponds to deadline of task in system. So at the beginning of each TL-Plane, all tasks are allocated local executions proportion to utilizations and assembled until end of time slice at which they all are pre-empted resulting in numerous preemptions and migrations.

Here, the algorithm USG(Unfair Semi-Greedy) algorithm is introduced where “Unfair” as fairness rule is totally relaxed, “Semi-Greedy” as two policies are employed : Non-Preemptability Policy to avoid problem of greedy schedulers as well as to reduce scheduling overheads, and Zero-Laxity Policy to maintain criticality and increase schedulability of algorithm.

CONCLUSION

The algorithm here reduced scheduling overheads by total relaxation of fairness rule. This algorithm uses global job queue ordered by increasing laxity. Tasks with zero laxity have higher priority and are scheduled for immediate execution. If tasks with less laxity arrive while tasks with greater laxity are running , the former are made to wait until they reach zero laxity. So, USG can be implemented in real-time applications with toleration of few deadline misses. Here, also comparison between two preemption methods in USG is shown viz. minimum remaining work vs. maximum laxity, showing that pre-empting task with maximum laxity outperforms pre-empting task with minimum remaining work in terms of scheduling overheads and improving schedulability.

**On Harmonic Fixed-Priority Scheduling of Periodic Real-Time Tasks with Constrained Deadlines [2016.1]**

Harmonic task set, i.e. tasks periods are integer multiples of each other, better utilize a processor to achieve high system utilization. Currently, harmonic task sets is limited to tasks with deadlines equal to their periods. Here, the concept of “harmonic task” is extended to tasks with constrained deadlines, i.e., deadlines less than or equal to their periods. It is also shown that harmonic task set with constrained deadlines have better schedulability than non-harmonic with same task utilization. With task harmonic relationship, it is shown that, partitioning approach can greatly improve schedulability of real-time tasks on multicore platforms

INTRODUCTION

Multi-core real-time scheduling is divided into three categories: partitioned, semi-partitioned and global scheduling. Partitioned approach is usually adopted for real-time system design due to better predictability and ease of design. Partitioned approach face the problem of partitioning real-time tasks for effective utilization of processing cores is NP-hard solved heuristic, for example , transforming to bin-packing problem and applying bin-packing heuristics, such as First-fit, worst-fit and best-fit.

It is known that harmonic task set can achieve utilization as high as one, on single processor according to rate monotonic scheduling policy. It was proposed to adjust loads on single core processors by allocating harmonic tasks together. Later, a polynomial time method was proposed to determine feasibility of task set by getting feasibility of harmonic task set transformed from original tasks set. All previous works indicate that with harmonic relationship among tasks in action system schedulability is improved but major limitation is that they target solely on periodic tasks with implicit deadlines according to RMOS schemes. Here, we extend to concept of ”harmonic task” to constrained deadlines, scheduled according to deadlines monotonic scheduling (DMS) policy. It is also shown that general harmonic task set has better schedulability than non-harmonic ones.

Here, we tackle problem of partitioned fixed-priority scheduling of real-time tasks on homogeneous multi-core platforms based on DMS scheme with help of harmonic relationship. We also develop metric to quantify harmonicity between two tasks and develop two partitioning algorithms based on this metric.

CONCLUSION

We extend the concept from implicit deadlines to constrained deadlines for “harmonic task set” which give better schedulability than non-harmonic ones. This is the first research that defines “harmonic tasks set” for periodic tasks with constrained deadlines.