Muhammad Talal al. International Journal of Recent Research Aspects ISSN: 2349-7688, Vol. 2, Issue 1, March 2015, pp. 88-94

Survey on Task Scheduling For Multi-core Systems

Muhammad Talal[1], M. Daud Abdullah[2]

[1]Student, Comsats institute of information technology, Wah Campus, Pakistan
[2]Assistant Professor, Comsats institute of information technology, Wah Campus, Pakistan

Abstract: **- In this paper, we review the literature on efficient task scheduling on multi-core system. The number of cores on one chip are increasing very rapidly. For achieving high performance without more power consumption and without heating up the system, multi-core processing technology is used. In addition for fully utilization of system resources in more efficient way task scheduler are developed, multi-core task scheduling is one of the most challenging problem. Such task scheduler which increase the performance of system and also provide task parallelism between different tasks on multi-core. The main goal of task scheduler is to optimize the performance so that it can minimize the task executing time and maximize resource utilization. In this survey paper we are going to discuss different types of task schedulers which work on different techniques.**

Keywords: Multicore, Threads, Task scheduler, Multiple nodes

I. INTRODUCTION

With the development in IC (integrated circuit) technology and multi-core system architecture is well known because of their high performance, low cost and less power consumption characteristics due to which they are being widely used. The number of processing units are increased with in the single chip as shown in Figure 1. A multi-core processor can perform more than one operation at a time as per core. Hardware technology is moving with rapid speed as ClearSpeed has developed a 96-core processor and Intel recently put 80 cores, on the other side software and programming models are facing failure to keep such speed. As in multi-core there are more processing units this means it can execute more than one instruction at a time which increases performance and throughput of the system. Theoretically adding more core in the same chip will twice the efficiency , but in reality the speed of single core is more , and more heat is produced by more cores as the consume more power for execution of program so the cooling cost also increase.

In this survey paper, we are concerned with the efficient utilization of all the cores and resources between them. For this purpose we need efficient task scheduling techniques for multiprocessor systems. The task scheduling has achieved wide attention.

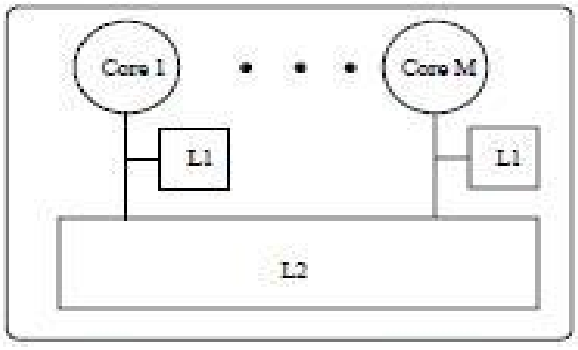

Fig: 1.

A lot of task schedulers are developed and researchers are developing more interest in task scheduler for multicore platform. In this paper we are going to discuss different task scheduler which work on different techniques for different types of system like Task scheduler for Real time systems and recovery in multi-core architectures and many more.

II. SURVEY OF DIFFERENT TECHNIQUES OF TASK SCHEDULING

A. Spread –cognizant scheduling - [1]

The Problem we are going to address is the issue on real time systems, that the parallel scheduling of tasks are discouraged, while we solve this problem by encouraging the co-scheduling ,ensuring the constraints of real time. Also concerned with the effective and efficient usage of common caches.

Discouraging the task execution together is not much harder or difficult than executing them together. Spread is the factor we want to minimize.

Spread: if grouping tasks has spread of K and ith quantum of computation of each task must be scheduled with in interval of [t, t +k] and t is the time unit. All the tasks in the group is to be scheduled when one task of group is scheduled, to get perfect parallelism.

In Pfair task scheduling task by scheduling their sub-tasks on earliest-deadline basis. In case of deadline between two or more scheduling tie-breaking rules are used. In Early-release the sub-tasks able to execute before their window, the time requirement of other tasks are ensured. Executing a sub-task earlier or not to release earlier this decision in arbitrary. Here we are trying to reduce the spread and meet real time constraints. Global scheduling in which there is single running queue is more suitable for reducing the Spread than other approaches like partitioning. Tasks of same wait, submitted at same time get consecutive slots in the scheduler queue, so that task will scheduled in close proximity of time, until disturbed by high priority with late arrival task. Deadline scheduling

method is capable to meet the time constraints if sub tasks or jobs are early released or allow to early execution

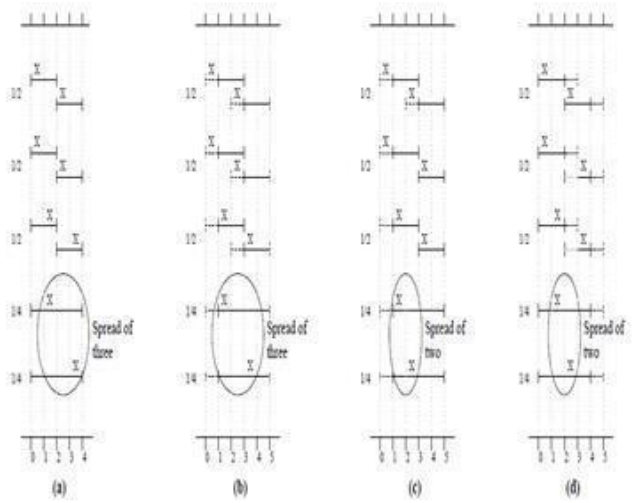

Fig: 2(A)

In Figure 2(A), in set (a) all the tasks are schedule without early release. Inset (b) the results are the same as in (a) after early releasing by 1 quantum in same scheduling. Dash lines shows how much early a task is released. In set (c) we reduce the spread by 1 quantum if selective tasks are allowed to early release. Instead of early release as we did in (b) and (C), we allow the task to miss their deadline by one quantum, it also reduce the spread by 1 quantum. How much quantum early release (X) execution of a task is performed is calculated according to the equation shown in Figure: 2(B).The performance of L2 cache is also improved by using this technique.

$$W_{\max} = \max_{T \in \tau} wt(T).$$

$$X = \begin{cases} 3, & \text{if } W_{\max} \le 1/3 \\ 4, & \text{if } 1/3 < W_{\max} \le 1/2 \\ 2 \times \left\lceil \frac{1}{1-W_{\max}} \right\rceil - 1, & \text{if } W_{\max} > 1/2 \end{cases}$$

Fig: 2(B)

B. Task scheduling for Multicore processor system to minimizing recovery time in case of single node fault – [2]

Many processors recently developed have multicores in single chip. If multicore processor failed, all the jobs that are executing, have to be re-executed. The algorithm which we are proposed is on the base of check point, assuming that the state of task is saved when send the results to other node or processing unit. If the computation is based on the results of series previous task, and fault occur at a single node unit then all the computation are made again i.e. double job have to be done and time to recover is also include in it . The method we are going to proposed can reduced up to 50% execution time including recovery time in case of single node failure. The only drawback of this method is if no failure is occur there is overhead in normal scenarios.

Satellite launching require performing thousands of tasks with much concern about time. An embedded system also requires multitasking within integrated circuit. Today's applications based on either soft real-time system or hard real-time system involving multiplicity of tasks within time constraint. So we have to migrate from dual-core to multi-core system along with heterogeneity of co-processors which improves the performance of the system. Each core is capable for executing the tasks independently, if a failure of shared resource or core itself then all the processor stop task execution and all of processors need to recover. If the communication in the processors is done by network technologies for connecting computers then it takes much longer time to recover from failure as it consumes a lot of time. In many scheduling algorithms there is no consideration of networks. The algorithm we proposed to recover from saved state is based on Sinnen algorithm of scheduling. In case of failure of processing node check pointing is the recovery method. When data results are transfer after complete execution the state of node is saved. If processor or node fails, it find the closest ancestor processor which is not affected by this failure and recover for the saved state from that node or processor.

Proposed Method

DAG is a graph which represent the dependence between groups of tasks like shown in figure: 3(A). The task 3 is depend on task 1 and task 2 means that it only runs or executes when 1 and 2 task complete their executions, while task 1 and task 2 are in parallel execution , they delivered the their results to task 3.

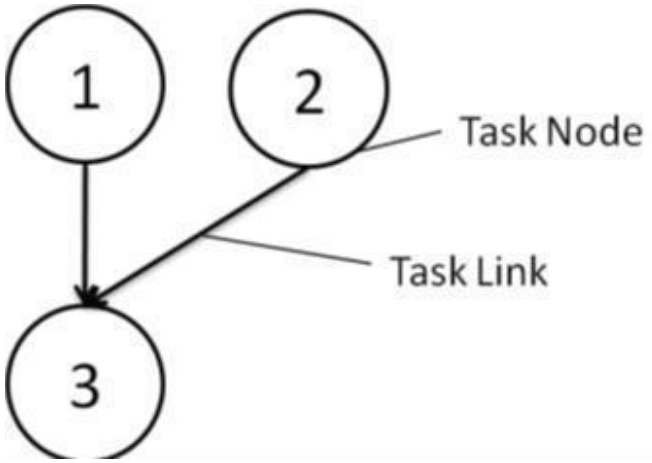

Fig: 3(A)

What happened if the die A is fail on which tasks 1, 2 and 3 are running? The execution results are lost and need to re-computation and recovery time is also included, it means more than double of the jobs have to be done again, and the tasks have to shift on other die B.

Task Scheduling and recovery in existing systems

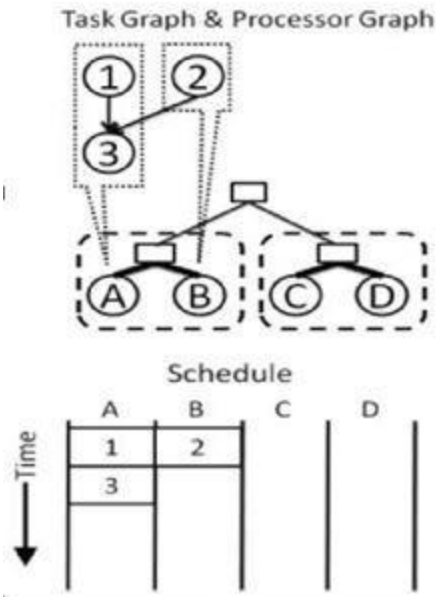

Fig: 3(B)

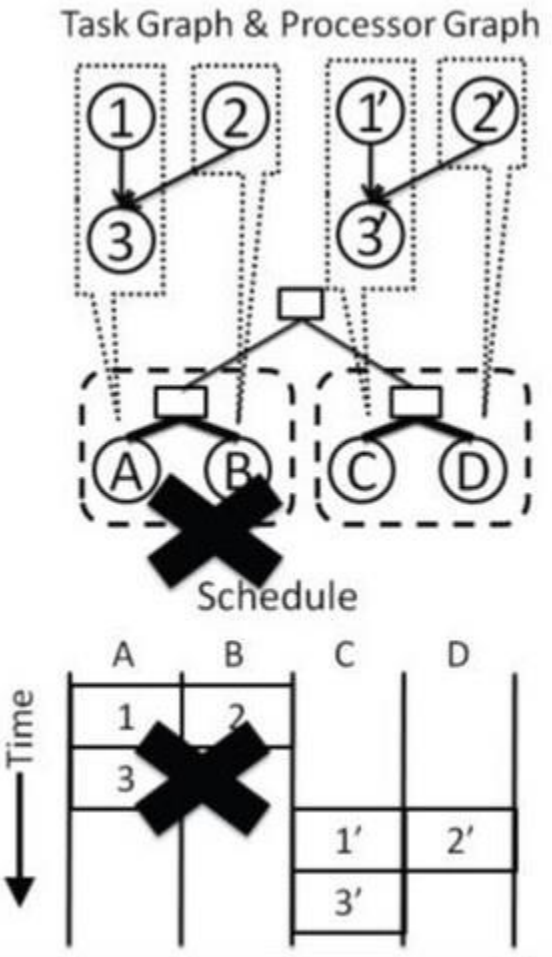

Fig: 3(C)

In figure: 3(B & C) task scheduling in existing system and recovery from failure in existing system is show respectively. According to proposed system if fault occurs at the same time when task 3 is executing, we only need to re-execute the task 3, as the states of task 1 and task2 are saved or stored. The saved results are re -used by the task 3 and recovery time in also reduced. As shown in Figure 4.But there is over head in the proposed method if no failure occur and executing is in normal scenarios. As shown in figure: 3(D & E) task scheduling and recovery from failure in proposed system.

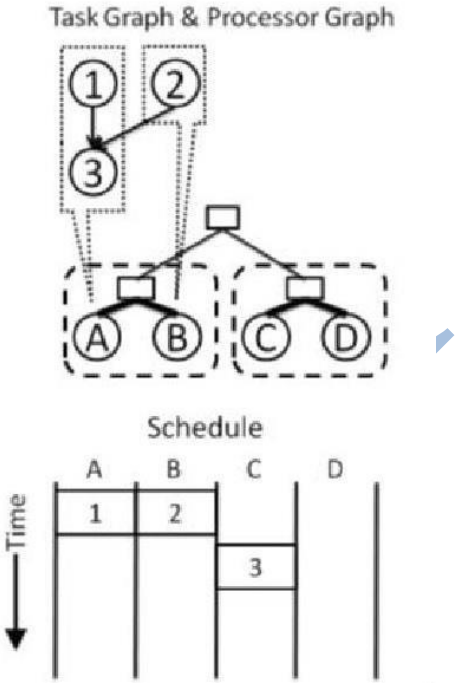

Fig: 3(D)

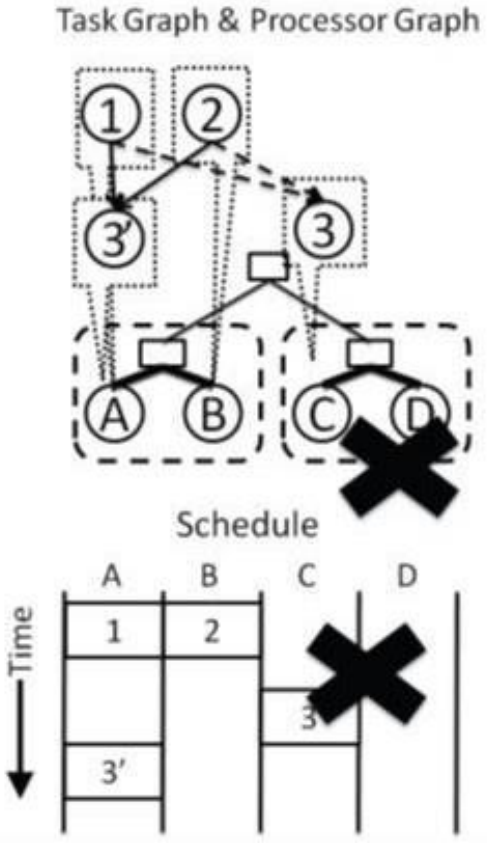

Fig: 3(E)

Algorithms for this proposed system is given in [2] by details.

C. Task scheduling on Adaptive Multi-core - [3]

The number of the processors in a chip are increasing very rapidly and the complexity is reduce due to the thermal and power constraints. More the number of cores in a single chip enable parallelism in applications in thread level Parallelism (TLP) but the sequential suffer for poor in the instruction level parallelism (ILP). In this research paper we proposed the adaptive multicore architectures that is able to make coalescing simple physical cores to make more complex virtual cores to make sequential code faster. This type of adaptive architecture can seamlessly exploit the both Instruction level parallelism and Thread level Parallelism. Previous work is only focus on the adaptive multicore for sequential work load. But we focus on the more real scenario where both sequential and parallel applications exist on adaptive multi-core platform. Both offline and online schedulers are developed which intelligently assign or re configure the cores to applications to make maximum utilizations of all the cores. Experiment results prove that the efficiency of symmetric and asymmetric is increased by adaptive muti-core scheduling.

The transitions towards multiple cores architectures are irreversible in computer systems. There are still a lot of applications which are of sequential workload, this type of applications suffer from limited instruction level parallelism (IPL). The asymmetric multicore has lack of flexibility to adjust in dynamic work load, as during the design mixture of complex and simple cores are freeze. Such a design of multi-core, which can changed or tailor itself according to the applications to manage the workload at run-time. This type of adaptive architecture is consist of set of simple cores, which can coalesced together to make virtually more complex core at run-time and the single core of virtually complex core can dis joined at any time. Such adaptive multicores perform well in diverse workload which is a mixture of both TLP and ILP. An adaptive Multi-core is shown in figure: 4(A)

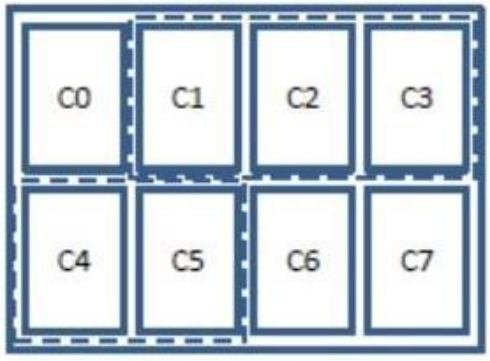

Fig: 4(A)

The performance evaluation of adaptive multicore is shown in Figure: 4(B) that how well this virtual complex core which consist of physically simple cores in performed when ILP or TLP applications run parallel. Both types of applications are supported concurrently in adaptive architectures. In reality it is difficult for perform limit the study of adaptive architectures with real workload. So an intelligent scheduler is employ to reconfigure and assign the cores to the applications to minimize the make span.

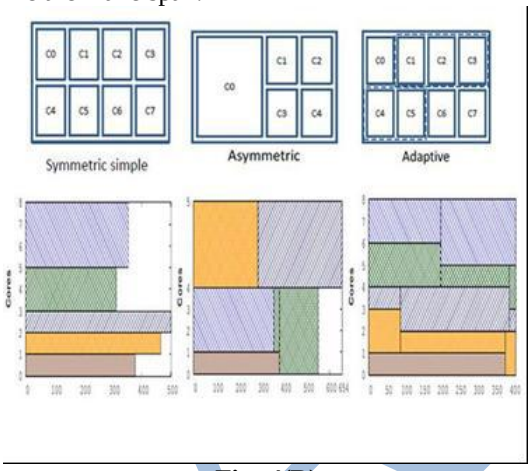

Fig: 4(B)

The sequential applications are restricted to use one core in symmetric multi-core, while multiple cores are used by parallel application and can get benefits. On the other hand in adaptive multi-core architecture the number of allocated or assigned cores for an application is carefully done, TLP applications use multiple simple cores and ILP can use one core, in this way adaptive multi-core manage dynamic and diverse work load. For adaptive multi-core we developed an scheduler that is given mixture of ILP and TLP tasks , namely Bahurupi , another online scheduler like Bahurupi is also developed that is easily integrated in any contemporary OS . Bahurupi is simple yet elegant approach toward the core coalition on base of software-hardware cooperative solution. The architecture is cluster based on which realistic constraints on the scheduler. In Bahurupi the coalition is built with in the cluster. In Figure: 4(C) there are two coalitions of two (C0, C1) and four of (C4-C7 cores). In the Figure: 4(C) one parallel application can run its threads on C2 and C3 core. One sequential application of medium level is run on or scheduled on coalition (C0 , C1) and one high level ILP is on other coalition (C4 – C7).

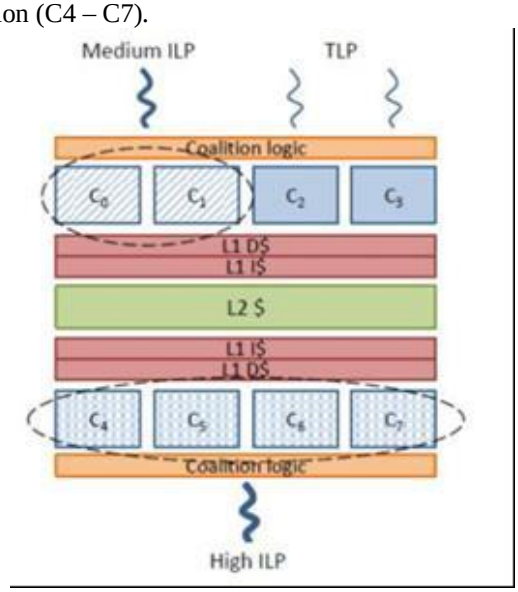

Fig: 4(C)

A new instruction is added to instruction set architecture called Sentinel, it is the first instruction of each block of code. Blocks are constructed at compile time along with the information about live-out or live-in register. Physically global pc, global register file and synchronization logic are shared by all the cores. The cores communicate through the live-out or live-in value of register. L1 cache is commonly shared by the cores in the coalition while L2 cache is shared or used by all the cores irrespective of the cores are in or out the coalition. The overhead of reconfiguration of L1 cache or resources of adding or removing core from coalition, Bahurupi only takes 100 cycles to reconfigure. This overhead is very small as compared to the execution time of application. For High level architecture of Bahurupi reader may read [3]. Performance of Bahurupi is shown in figure: 4(D)

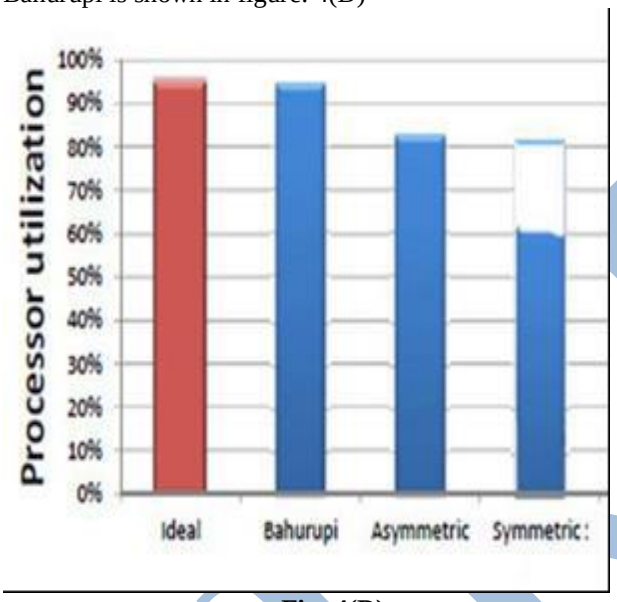

Fig: 4(D)

D. Multi-Socket Multicore System –[4]

OpenMP API shared memory that allows programmer to show concurrency at a high level and place the burden of parallel execution scheduling on the openMP run time system. This allows to develop the scientific computer fatter with many core processors. But effective task scheduling on multi socket multi-core with shared memory increase the complexity. Task scheduling we proposed is based on strategy of hierarchical scheduling. A thread on single chip is allowed to steal work on the behalf of other threads that shared a cache. Qthread is threading library which is open-source is extended for the implementation of our scheduler, which accept the openMP program through ROSE compiler.

Parallel task programming allows programmer to specify the parallel task with the size of problem, leave the responsibility on the processor to perform them at run-time. It is realized that the in future multicore processors are improving the performance than increasing the performance of single core. Meet the challenges, time conflicts, load management and minimize the time overhead are the properties of efficient scheduler. When tasks are not distributed among the processing units the load imblancement arise due to which idleness is shown. If load is distributed equally, maximum utilization of processors, while it include some overhead cost and if load balancing is between the sockets it take more overhead. Our approach is combination of work stealing and shared queues for low overhead load distribution. We have developed a

method using ROSE compiler which runs openMP program with Qthread library. ROSE compiler is capable for source to source translator. In Qthread there is a variety of synchronization methods which are non-blocking and potentially blocking. The concept of lightweight threads is intended match with future hardware threads. Cooperative multitasking is used in Qthreads runtime.

With minimum overhead cost the optimal solution for multithreaded scheduling of DAG's is stealing as proved by Blumofe et Al. This solution is implemented on run time scheduler. Idle shepherd is obtained more work by stealing the task from task queue of old busy shepherd. The burden of load work is on idle shepherd, interruption for busy shepherd are minimized in work stealing scheduler. The new tasks whose data is fresh in processor cache is first to schedule and the last to be stolen. To reduce the limits of work stealing and shared queues, create a hierarchical approach Multi-thread shepherds. Developed one shepherd for all the cores which are on the same chip, these cores have shared socket and memory. Shared queue can reduce performance or act as bottleneck, the number of processing units per chip are bounded and intra –chip locking operation is fast with in the chip. Qthread MTS 32 core is faster or have much better performance than ICC and GCC. In execution time MTS is faster than ICC for 5 of 7 benchmarks and 4 of 6 benchmarks faster than GCC, just slower for 2 benchmarks. If interested about the benchmarks or want to learn more about its performance reader may refer to [4].

E. A User space Library for Multicore Real-Time Scheduling –[5]

A computer hardware having many cores is present or founded everywhere, theory for scheduling task on multi-core is increasing rapidly. Real time operation system are not suitable for this type of repaid change. It is proposed that software application run outside the real time operation system's kernel and run in user space. We first describe the user space scheduler which supports the preemptive and dynamic priority, migration of real tasks on multi-cores. Complex scheduling algorithms and locking protocols are developed by the researchers of real time systems in recent years. Most of them focus on the resource allocation techniques in multi-core systems. The research for implementation is focus to modify the kernel of real time operating system which supports the techniques of resource allocations. Example is CEDF (Clustered Earliest deadline first) algorithm.

The approach "user space" may be much more useful. A user space library which can support preemptive and dynamic priority scheduling on multi-core system for real time tasks. In library real time tasks are taken as user-level thread that share a single address space. The overhead measurements of this library generally ranges from 1 to 10 of microseconds,

overhead in worst case when task migrates between the cores take very few hundred of microsecond. We choose user-level thread solution because of two reasons, first one is that it takes less overhead time than kernel-level and secondly single address space is standard practice for parallel threads in single application. The library allows application to use innovative scheduling techniques in multi-core system. The application can incorporate real-world knowledge in making schedule decision. For example real world deadlines are used to set the dynamically deadlines of real time tasks while the other typical applications are scheduled safely under real time operating system on other cores. Only static-priority scheduling is supported by the open-source real time operating systems, while our library support dynamic priority scheduling.

A function named schedule() is defined in library which is responsible of context switching between the user-level threads. The clustered earliest deadline first scheduler switch the execution of tasks in ready queue, if any task has early deadline than currently running task. When library is initialized it creates an idle task which can be run when there is no real time task is in ready queue, it has lowest possible priority. There is another function named fast_swapcontext() in library which is called by schedule() function for context switching . There are many other function in library like make_context(), get_context and many more , if interested you may concern [5].

F. Task scheduling in multicore with dispatcher schema – [6,7]

In this technique processor is responsible for handling signal of control and co-processor is responsible for data computation. This is because dispatcher is needed to select which sub-task is assigned or dispatched to which co-processor. The migration polices are also implemented with these polices allocation of resources, efficiency and performance is also improved. In large systems or in embedded systems in which system is make more dedicated for the application, there is one general purpose processor and others are co-processors to increase the performance, some applications have dynamic work load like network. So to full fill the customer requirement such systems may use. In multicore for task scheduling researches are made for better synchronization protocols. In heterogeneous multi- core systems there are number of co-processors which are executing the task at very high speed and one general purpose processor. Different types of dispatcher schemas are used in dispatcher mechanism like partition schema, global schema and hybrid schema. This technique give better performance, as it use dispatcher mechanism of different sub-tasks. The policy of task migration increase the utilization of resources.

The processor execute the program instruction which tells the processor what to do like read/write data. The improvement in performance is gain by using heterogeneous multicore system which work on software algorithm. The proposed system, tasks are executed in preemptive in processor while in co-processor task execution is non-preemptive, but tasks may migrate or jump from one co-processor to other, like in partition schema each server is firstly allocate with a co-processor and then all the tasks are done by that co-processor [6].

III. CONCLUSION

After this survey, the scheduling technique which we find more better efficient are Adaptive Multi-core Task scheduling is the best technique for design big system or it may use in mega processing centers, in addition to adaptive multi-core task scheduling in such big systems single point failure recovery is must be implements so that in case of failure system can easily recover and executed part is not require to re-executes, but the limitation is that it only support single point failure. Such algorithms must be developed which support recovery form maximum failure points.

Currently researchers are working of Self-adaptive Task Scheduling for Dedicated Heterogeneous systems. Developing new scheduler for mobile platforms and for scheduler for hard real time system gain much attention. With the fast advancement in hardware technology task schedulers are also developed with more many support of maximum utilization of the hardware in effective way.

REFERENCES

[1]. James H. Anderson and John M. Calandrino, “Parallel task scheduling on multicore platforms”, ACM SIGBED Review - Special issue: The work-in-progress (WIP) session of the RTSS 2005 Homepage archive Volume 3 Issue 1, January 2006 Pages 1-6

[2]. Gotoda, S. and Ito, M. and Shibata, N, “Task Scheduling Algorithm for Multicore Processor System for Minimizing Recovery Time in Case of Single Node Fault”, Cluster, Cloud and Grid Computing (CCGrid), 2012 12th IEEE/ACM International Symposium on Publication Year: 2012, Page(s): 260- 267

[3]. Pricopi, M. and Mitra, T., “Task Scheduling on Adaptive Multi-Core”, Computers, IEEE Transactions on Volume:63 , Issue: 10, Publication Year: 2014 , Page(s): 2590- 2603

[4]. Stephen L. Olivier and Allan K. Porterfield and Kyle B. Wheeler and Jan F. Prins, “Scheduling task parallelism on multi-socket multicore systems”, Published in: Proceeding ROSS '11 Proceedings of the 1st International Workshop on Runtime and Operating Systems for Supercomputers, Pages 49-56, year 2011

[5]. Malcolm S. Mollison and James H. Anderson, “Bringing Theory Into Practice: A Userspace Library for Multicore Real-Time Scheduling”, Published in: Real-Time and Embedded Technology and Applications Symposium (RTAS), 2013 IEEE 19th Conference, on 9-11 April 2013, Page(s):283 – 292

[6]. Poonam Karande and S.S.Dhotre and Suhas Patil, “Task Management for Heterogeneous Multi-coreScheduling”, International Journal of Computer Science and Information Technologies, Vol. 5 (1) , 2014, 636-639

[7]. Binotto, A.P.D and Pedras, B.M.V and Götz, M and Kuijper, A and Pereira, C.E and Stork, A and Fellner, D.W, "Effective Dynamic Scheduling on Heterogeneous Multi/Manycore Desktop Platforms", Published in:Computer Architecture and High Performance Computing Workshops (SBAC-PADW), 2010 22nd International Symposium on,Date of Conference:27-30 Oct. 2010, Page(s):37 - 42.

