World Applied Sciences Journal 27 (9): 1175-1182, 2013
ISSN 1818-4952

DOI: 10.5829/idosi.wasj.2013.27.09.1627

A Review of Power Efficient Load Balancing Algorithms for Multicore Systems

[1]Nadia Dilawar, [1]Muhammad Zakarya and [2]Izaz Ur Rahman

[1]Department of Computer Science, Abdul Wali Khan University, Mardan
[2]Brunel University, London, UK

Abstract: High performance computing including parallel and distributing systems now focus on power efficient execution. This is not only to the rising energy cost but these systems are playing a major role in global warming and greenhouse gas emissions. The birth of multicore systems is a major cause of huge power consumption and producing a lot of heat. Cores are the fundamental units that read and execute instructions in a computer program. A single core CPU has only one core that executes single instruction at a time. A multicore CPU has more than one independent processing core on a single chip to increase throughput and performance. Theoretically, by adding extra core to the same chip double the performance, but in practice speed of each core is slower than the single core processor. Likewise executing more instructions increases power consumption and thus produce extra high temperature. Soft-wares are written for multicore platform that distribute the workload amongst multiple identical or different cores. This functionality is called thread-level parallelism. These particular methods are called load balancing mechanism. In this paper we have summarized a number of load balancing algorithms that minimize the power consumption of multicore technology while maintaining performance to the best level. We have also compared two important algorithms proposed in the literature in terms of faster execution time and power efficiency.

Key words: Multicore · Energy efficiency · Load balancing · Distributed computing · Scheduling

INTRODUCTION

To use the multiple cores efficiently, the distribution or partitioning of threads among the cores is a critical and important concern. Multi-core hardware can increase the performance and save the power consumption in a much better way if all the cores stay similarly active. If the cores are not equally loaded, some cores will be overloaded and slow while other are idle and waste the CPU cycles. In worse, inappropriate workload partitioning can increase massage passing among cores and disputes for shared resources which reduce the performance significantly. To execute a workload on multicore processors the workload must be distributed amongst different cores using some proper mechanism or algorithm. There are few basic steps involve in designing parallel applications:

Partitioning Phase: In the first step of designing algorithm, designers need to find out the opportunities for parallel execution of the threads. The main purpose of this is to define number of small tasks that they can run in parallel on different cores [1]. The task is divided into a number of jobs and then these jobs are scheduled to individual cores. Dependability of different jobs must be considered when deciding to push jobs to core for execution. For example all jobs having dependency must me pushed to same cores to eliminate communication overhead or push then to neighboring cores to minimize the communication overhead.

Communication Phase: Due to some reasons tasks can't be executed concurrently those who were planned to be execute in parallel. In general, those tasks will be executed independently. Result of one task may be required to another task, so the data must be transferred between tasks to allow computation to continue. This massage passing or data transfer between tasks is specified in communication phase [1].

Corresponding Author: Nadia Dilawar, Department of Computer Science, Abdul Wali Khan University, Mardan, Pakistan.

Mapping Phase: In the absolute stage of designing of parallel algorithm, designers identify where tasks will execute. "This mapping problem does not arise on uniprocessors or on shared memory computers because they provide automatic task scheduling" [2].

Many cores might be homogeneous i.e. all cores having same characteristics and properties or heterogeneous i.e. different cores have different characteristics like speed, architecture, voltage or frequency scaling to achieve power efficiency [3] and etc. scheduling tasks for homogeneous cores is likely sample than scheduling tasks over heterogeneous cores as in homogeneous systems all the cores are operating with the same frequency and speed.

Parallel Programming: Multicore architectures basically consist of cluster of SMP nodes. OpenMP [4], MPI [5] and Heterogeneous MPI (HMPI) programming paradigms can be used for parallelization of instruction codes for such architectures. OpenMP uses shared memory and hence is viewed as a simpler programming paradigm than MPI which is primarily a distributed memory paradigm. As OpenMP applications cannot be scaled beyond more than single SMP node but MPI can be scaled to more SMP nodes. MPI based applications may introduce overhead in inter-node communication. Therefore proper programming mechanism should be implemented over these multicore architecture to make them efficient. In some cases MPI implementation gives better performance than the OpenMP implementation, but in other cases OpenMP implementation performs better than the MPI counterpart and requires less programming effort as well. HMPI are specially designed to concentrate on heterogeneous systems. MATLAB provide a HMPI toolbox to parallel program heterogeneous systems.

Power Aware Multicores: The superlative approach to diminish power consumption [6] of multi-cores having negligible influence on performance is to use dynamic voltage & frequency scaling (DVFS). Applying voltage scaling comprehensively to the entire machine would decrease energy usage, but may not be always optimal. Instead we propose to exploit the fact that there are hundreds to thousands of cores and each core can be voltage scaled separately, thus employing fine grain power management. Individual cores can be voltage and frequency scaled to any arbitrary voltage and frequency in the possible range, but this could be problematic and cumbrous. Similarly different cores running at different

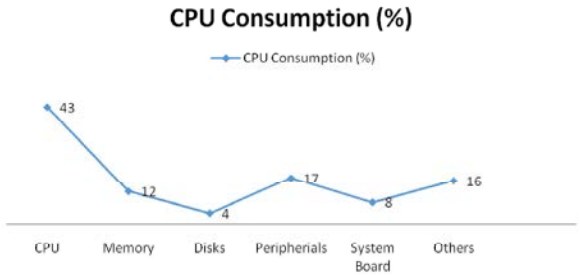

Fig. 1: Power consumption of different system components

frequency or different voltage [7] might create asynchronous interfaces, adding latency, meta-stability and also needed complex power delivery methodology [8]. In [28] the authors have shown some characteristics of multi-core ships and have compared different multicore CPUs. They have theoretically compared some load balancing algorithms that best fit for homogeneous multicores. Load balancing becomes more tedious when multiple cores or processors have different processing speed, architecture and different memory or cache. This work is an extension to [28], with some simulation results and some discussion on heterogeneous multicores or multiprocessors. According to 2005 study the power used by computer servers was about 0.6% of total US electricity consumption growing to 1.2% when cooling and secondary infrastructures are involved. In 2005, the cumulative electricity bill for operating servers was $2.7 billion (for US) and $7.2 billion (worldwide). The entire energy spent by servers doubled over the period 2000 to 2005 in world. Similarly, GHG emission is also getting to a critical edge. In 2007, it was projected that ICT industry accounts for about 2% of global CO_2 emissions; this is comparable to the amount of CO_2 emitted by the aeronautics. The following graph shows power consumption of different devices in a computer system. It is clear that the more power is consumed by the CPU.

Load Balancing: Load balancing is computer networking methodology to distribute work load across multiple computers or cores network links, central processing units, disk drive and other devices to achieve the optimal resource utilization, maximize throughput, minimize response time and avoid over load. In simple terms, load balancing is a method to spread tasks out over multiple resources. By processing tasks and guiding sessions on different servers, load balancing helps a network avoid annoying downtime and delivers optimal performance to

users [9]. There are virtual load balancing solutions that work in a manner similar to virtual applications or server environments. There are also physical load balancing hardware solutions that can be integrated with a network. The method used depends entirely upon the team implementing the solution and their particular needs. Load balancing is a computer networking methodology to distribute workload across multiple computers or a computer cluster, network links, central processing units, disk drives, or other resources, to achieve optimal resource utilization, maximize throughput, minimize response time and avoid overload. Using multiple components with load balancing, instead of a single component, may increase reliability through redundancy. The load balancing service is usually provided by dedicated software or hardware, such as a multilayer switch or a domain name server.

Following are the scheduling techniques that are briefly discussed in [10].

OLB Scheduler: The OLB Scheduler, like those found in Resource Management Systems and Distributed OSs, uses Opportunistic Load Balancing (OLB) and assigns the next queued job to the next available machine.

LBA Scheduler: The LBA Scheduler assigning each job to the machine where it is predicted, assuming that all machines are unused, to execute the fastest (a policy referred to as Limited Best Assignment or LBA).

Smart Scheduler: The Smart Scheduler assigns jobs to machines based both upon their expected performance on the various platforms as well as the loads on those machines [11].

In next section some literature review and recent research work is presented for homogeneous and heterogeneous multicores. In general the following factors must be kept in front while designing an efficient algorithm for such parallel and distributed systems [12].

- Workload must be equally distributed across the available cores that are capable of performing their executing.
- Resources i.e. core utilization should not be wasted. A task should not be allotted to a core that is over provisioned with respect to that task.
- A valid and exact mapping of a task to a core must be found if such an assignment exists. This process must not increase signi?cant overhead to the cost of executing that specific task.
- In case of splitting the tasks set to fully utilize the cores, unreasonably dependencies should not be considered. It is necessary to avoid splitting the tasks as it might increase the total execution time of the task. Execution time of a single task is given by:

$$\text{TotalET} = \text{ETcore}_i + (\text{TaskSplittime})_i + (\text{Waittime})_i$$

Hence total execution time of the tasks set is:

$$(\text{Total ET})_n = \sum_{i=0}^{n} \left(\text{ET core}i + \left(\text{Task Split Time}\right)n + \left(\text{Wait Time}\right)n\right)$$

- Task shifting is more advantageous to utilization than task splitting, if utilization is possible through light task shifting, huge task must not be considered as it will increase the communication overhead and hence might reduce performance.

Related Work: In [13] Bautista proposed a simple power-aware scheduling algorithm for multi-core systems. This scheduling algorithm moves the extra workload from overloaded cores to the less loaded cores. A complete task is moves from over loaded system to less loaded cores. This algorithm reduces the energy consumption while increasing or decreasing the frequency of the cores. Simple power aware technique is never able to maintain the workload equal on cores, so all cores are not equally utilized. Simple power-aware scheme do not split the task so it move the complete task to other core.

In [14] Raj Kumar introduced the technique of highest priority task splitting. Task is divided into two portions ô' and ô“. t’ of each splitting task is assign to next processing core and t” is must assign to the last core. Every time a task is allocated to a processing core, a schedulability test ensures that the tasks allotted to a core are schedulable with deadline monotonic. The existing problem in this approach is that all cores are not equally loaded.

In [15] N. Min Allah proposed a scheduling algorithm which finds the lowest core speed utilization and then the lightest task from the heavy core is shifted to the lowest loaded cores o maintain the uniform system speed. The existing problem in this approach is that there is no task splitting and all cores are not equally utilized. An RM algorithm is implemented for fixed priority tasks over multiple cores and has found that all the tasks are optimal and feasible. They have minimized the energy consumption to some significant level. Fig. 2 [15] shows the lowest core speed cores utilization after shifting.

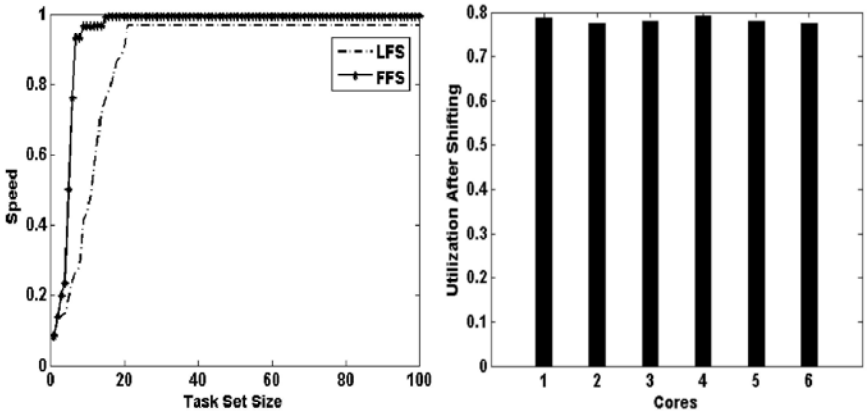

Fig. 2: comparison of lowest core speed and utilization

In [2] Seo presented a dynamic task repartitioning algorithm that distributes the given workload among the cores. All cores are divided into two groups, (i) Donator and (ii) Grantee. Donator moves task to grantee ant the grantee groups execute these tasks. Algorithm use cycle conserving is capable to maintain L for all processing cores so

$$L = \sum w_i/p_i - \sum cc_i/p_i$$

And Power is reduced using cycle conserving. There is no task splitting techniques in this approach, hence all cores are not equally utilized. Kato [1] presented a partitioned scheduling scheme for the multiprocessors. This technique assigns the tasks to specific processors an such a way that processor one is filled with tasks 100 percent utilized and remaining processor are filled according to some specific value. A task can be split in two subtasks and these two partitioned are assigned to different processor if any processors don't have the sufficient space. Split tasks are executed in any order. All processors are not equally utilized because processor 1 is 100 percent utilized and the remaining processors are utilized to some specific value.

In [16] M. Zakarya have proposed a new load balancing algorithm for scheduling real-time tasks in a multi-core processor technology. They have also introduced task splitting concept to balance the load on each core to minimize its energy requirements. They claim for having achieved workload partitioning with 100 percent precision. All cores are 100% utilized. This algorithm is capable to distribute workload among all processing cores equally and keep the utilization of all processing cores on a verage. This means that no processing core is extra loaded or extra burdened.

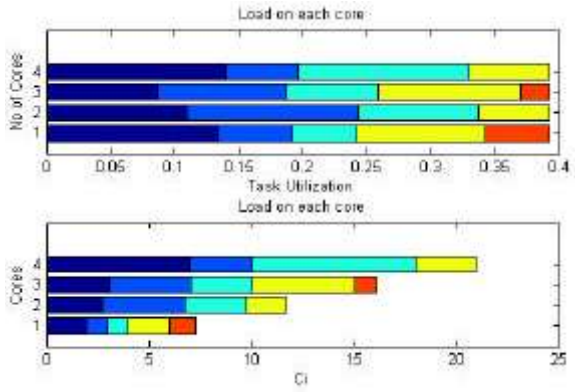

Fig. 3: load balancing for multi-cores

In proposed technique all the processing cores are considered to be homogeneous, so if the workload is distributed evenly, all cores will complete their work at the same time. Proposed workload partitioning algorithm can be used for reducing the energy consumption in multi-core systems. This proposed algorithm can easily be used with cycle-conserving technique which can update the utilization of each core dynamically on release and completion of a task. These tasks are capable to complete their execution earlier than it's provided WCET. Fig. 3 [16] shows 100% balancing for multicores.

In order to make full use of processing cores the authors in [17] proposed a task scheduling algorithm on the basis of task duplication. There algorithm composed of three steps of operations so that threads are allocated to processing cores more suitably. This algorithm not only increases the executive efficiency of task scheduling, but also can adjust scheduling sets according to the number of processing cores. This algorithm reduces communication overhead and keeps load balancing between cores and meanwhile speedup ratio of parallel program is improved.

Table 1: Comparative study of different load balancing algorithm on multicore platform

	Energy efficient workload balancing for real-time systems over multicore	Partitioned fixed-priority preemptive scheduling	Simple power-aware scheduling for multi-core systems	Power Efficient Rate monotonic Scheduling for multi-coresystems
Workload on each cores	100% Equal	88%	Never equal	95%
Task migration	m-1	N/A	Utilizationof cores are balanced	Yes
Task splitting	Yes	Yes	No	No
Number of splitted tasks	m-1	m-1	No	-
Extra load on any core	No	-	Yes	Yes
Energy expenditure due to unbalanced workload	No	-	Yes	Yes
Splitted objects per core	At least one and at most two	At most one-	-	
Utilization bound per core	100% with EDF	60% with partitioning DM scheduling	-	-
Number of splitted portions	Two	-	-	-
Execution order of splitted portions	Not in parallel and τ_i' before than τ_2"	-	-	-
Time complexity (average case)	$O(n^2)$	$O(n^2)$	$O(n^2)$	-
Task importance	Equal	Yes	-	IC based
Scheduling criterion	Task period	-	-	Task period

Load balancing over heterogeneous cores is a tedious job. The underlying OS scheduler needs to be heterogeneity-aware, so that it can tie jobs to cores according to features of cores. Some of the distinguished loads balancing algorithms proposed are [18] and [19]. Both of them assume a system with two core types i.e. fast & slow. These algorithms checks for unceasing performance monitoring to conclude optimal task-to-core assignment. IPC-driven algorithm in [18] intermittently samples tasks, IPC on available cores i.e. fast & slow, to decide the relative benefit for each task from running on the core i.e. fast. Those tasks that have a higher fast-to-slow IPC ratio have precedence in execution on the faster core, as they might be able to attain a relatively greater speed-up there. The method in [19] uses an analogous mechanism, except that the sampling method for tasks is further efficient & robust by using more than one sample per core type per task. In addition, the authors in [19] have also proposed an algorithm that tries to determine a globally optimal assignment by sampling performance of task groups rather than making local thread-swapping decisions. Both these methodologies promise meaningfully improved performance than naïve heterogeneous-agnostic policies with any kind of heterogeneous workload, but both are difficult to scale to many cores [20].

In [10] the authors have designed a scheduling framework for managing jobs and resources in a heterogeneous computational environment. They called it SmartNet that not only implements scheduling algorithms, but also provides the information essential for these algorithms to make clever judgments. SmartNet is designed to measure both machine affinity and loads and provide this information to its scheduling algorithms.

The performance of multi-core systems relies on an effective cache system. Some authors [21] have focused on cache i.e. L1, L2 and L3 performance to increase the core performance. In a multicore processor each core might have its own private cache or/and might share the LCC i.e. last level cache. LCC can also be private for some cores. Any way if these caches are designed or used in such a way that communication cost is minimized, core utilization will be effected. We can improve the cache system through adapting the cache usage as per core requirements and if needed store the mostly used instruction in LCC or in shared cache implementing any of the least recently used (LRU), most recently used (MRU), mostly used, rare used algorithms.

It is essential to develop efficient scheduling strategies to appropriately allocate core to instructions / tasks. However scheduling is challenging due to its inherent NP-completeness. We should have some new approaches based on Particle Swarm Optimization (PSO) where numerous candidate solutions (particles) coexist and collaborate indirectly. Each particle ?ies in the problem search galaxy looking for an optimal position to land. A particle regulates its position as time passes according to its private familiarity as well as to the knowledge it gains from its neighboring particles. In [22] the authors have presented a PSO based scheduling algorithm for distributed cloud computing [23, 24]. Their experiments accomplished using CloudSim and job data from real scienti?c problem show that their proposed PSO based scheduler balance the studied metrics efficiently as compared to random assignment based and genetic algorithms based schedulers. Similarly a lot of genetic algorithms [29, 30] are available that have tried to optimize the scheduling problem for heterogeneous multicore systems.

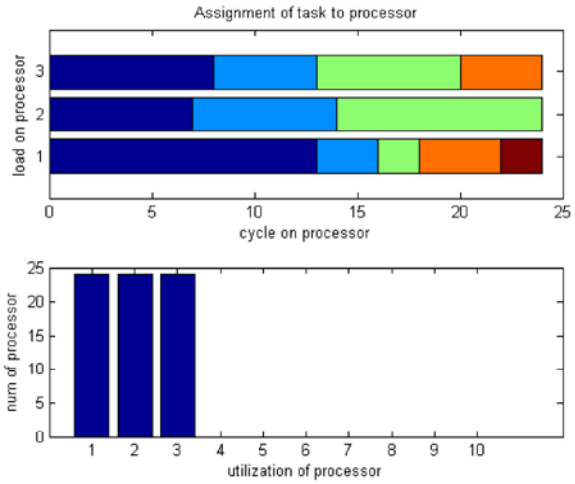

Fig. 4: Load balancing for multi-processors [25]

In [25] the authors have proposed a load balancing algorithm for multi-processors. They have proposed a TL-PLANE based approach to prioritize the tasks for real-time systems and have scheduled the tasks to more suitable processors to maintain their speed. Further they have implemented task splitting mechanism to utilize all the processors with the same frequency and speed. They have reduced the number of tasks which are splitted and have thus enhanced the execution time of every individual processor. Fig 4 [25] shows load balancing for multiprocessors having homogeneous architectures.

CONCLUSION

In this article we have summarized a number of load balancing algorithms that balance the load amongst a number of homogeneous cores. Load balancing [26] utilizes all the cores to its maximum performance and hence results in removal of energy usage that is wasted during idle clock cycles. It is shown that load balancing on heterogeneous cores or heterogeneous multi-processors is a tedious job. In literature only a few quality articles have addressed this issue but still no 100% results are achieved. Many core is not always increases the performance. There are some cases where performance for a specific program degrades when executes on more core as stated by Amdahl's Law. One reason might be the dependency of code, which might increase the wait time, communication overhead and etc. The limitation is Amdahl's Law, which states that the parallel speedup is limited by the serial code in a program i.e.

$$(\text{Parallel Speedup})_n = \sum_{i=0}^{n} \left(\text{Serial \%} + \left(1\text{-Serial \%}\right)/\text{N}\right)$$

It is clear from the above equation that if the serial percentage in a program is large, then parallel speedup saturates with small number of cores. Efficient and 100% resource utilization is a key to take full advantage of performance of computing machines [27]. Similarly creating unnecessarily dependencies on different components in the system significantly sinks overall system performance as it will results in increased communication overhead. There is a need to implement genetic algorithms, deferential equation (DE) based optimization, heuristics based algorithms and Particle Swarm Optimization (PSO) based intelligent algorithm to further decrease the power consumption of many core machines. Load balancing is high performance computing like cluster, grid and cloud [31, 32] needs researcher's attention in academia and research laboratories. Load balancing is a hot research issue especially in HPC like clusters, grid and cloud computing. The heterogeneity of these systems makes this issue a little bit complex but these factors must be considered to fully utilize these systems for maximum utilization and true benefits in terms of service providers. Performance for cost, benefits to service providers and efficient resource allocation are the issues that needs to be addressed.

ACKNOWLEDGMENT

The work is supported by iFuture: A Leading Research Group in Department of Computer Science, Abdul Wali Khan University, Mardan. The authors are thankful to Miss. Rishma Sadaf for her review and valuable comments.

REFERENCES

1. Kato, S. and N. Yamasaki, 2007. Real-Time Scheduling with Task Splitting on Multiprocessors, In Proceedings of the 13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA), pp: 441-450.
2. Seo, E., J. Jeong, S. Park and J. Lee, 2008. Energy Efficient Scheduling of Real-Time Tasks on Multicore Processors, IEEE Transactions on Parallel and Distributed Systems.
3. Zakarya, M. and I.U. Rahman, 2012. Towards Energy Efficient High Performance Computing Perceptions, Hurdles & Solutions. Technical Journal UET Taxila (Pakistan).

4. Duran, A., M. Gonzàlez and J. Corbalán, 2005. Automatic thread distribution for nested parallelism in OpenMP. In *Proceedings of the 19th annual international conference on Supercomputing,* pp. 121-130. ACM.
5. Anne C. Elster and David L. Presberg, 1993. Setting Standards For Parallel Computing: The High Performance Fortran and Message Passing Interface Efforts, Theory Center SMART NODE Newsletter, Vol. 5, No.3.
6. Khan, A.A. and M. Zakarya, 2010. Performance Sensitive Power Aware Multiprocessor Scheduling in Real-time Systems. Technical Journal UET Taxila (Pakistan).
7. Zakarya, M., I.U. Rahman and A.A. Khan, 2012. Energy crisis, global warming & IT industry: Can the IT professionals make it better some day? A review. In Emerging Technologies (ICET), 2012 International Conference on, pp: 1-6. IEEE.
8. Borkar, S., 2007. Thousand core chips: a technology perspective. In Proceedings of the 44th annual Design Automation Conference, pp: 746-749. ACM.
9. Zakarya, M., A.A. Khan and H. Hussain, 2010. Grid High Availability and Service Security Issues with Solutions.
10. Freund, R., T. Kidd, D. Hensgen and L. Moore, 1996. SmartNet: a scheduling framework for heterogeneous computing. In *Parallel Architectures, Algorithms and Networks, 1996. Proceedings. Second International Symposium on,* pp: 514-521. IEEE
11. Zakarya, M., I.U. Rahman, N. Dilawar and R. Sadaf, 0000. An integrative study on bioinformatics computing concepts, issues and problems. International Journal of Computer Science (IJCSI), 8(6).
12. Zakarya, M., 2013. DDoS Verification and Attack Packet Dropping Algorithm in Cloud Computing. World Applied Sciences Journal, 23(11): 1418-1424.
13. Bautista, D., J. Sahuquillo, H. Hassan, S. Petit and J. Duato, 0000. A Simple Power-Aware Scheduling for Multicore Systems when Running Real-Time Applications, Department of Computer Engineering (DISCA)Universidad Polit´ecnica de Valencia, Spain
14. Lakshmanan, K., R. Kumar and J.P. Lehoczky, 0000. Partitioned Fixed-Priority Preemptive Scheduling for Multi-Core Processors, Carnegie Mellon University Pittsburgh, PA 15213, USA
15. Min-Allah, N., H. Hussain, S.U. Khan and A.Y. Zomaya, 2011. Power efficient rate Monotonic scheduling for multi-core systems, Journal of Supercomputing.
16. Zakarya, M., N. Dilawar, M.A. Khattak and M. Hayat, 2013. Energy Efficient Workload Balancing Algorithm for Real-Time Tasks over Multi-Core.World Applied Sciences Journal, 22(10): 1431-1439.
17. Geng, X., G. Xu, D. Wang and Y. Shi, 2011. A task scheduling algorithm based on multi-core processors. In Mechatronic Science, Electric Engineering and Computer (MEC), 2011 International Conference on, pp: 942-945. IEEE
18. Becchi, M. and P. Crowley, 2006. Dynamic Thread Assignment on Heterogeneous Multiprocessor Architectures. In Proceedings of the 3rd Conference on Computing Frontiers. Computing Frontiers '06. ACM, New York, NY, USA, pp: 29-40.
19. Kumar, R. *et al*. 2004. Single-ISA Heterogeneous Multi-Core Architectures for Multithreaded Workload Performance. In Proceedings of the 31st Annual International Symposium on Computer Architecture. ISCA '04. IEEE Computer Society, Washington, DC, USA, 64.
20. Shelepov, D., J.C. Saez Alcaide, S. Jeffery, A. Fedorova, N. Perez, Z.F. Huang, ... and V. Kumar, 2009. HASS: a scheduler for heterogeneous multicore systems. ACM SIGOPS Operating Systems Review, 43(2): 66-75.
21. Chang, J. and G.S. Sohi, 2006. Cooperative caching for chip multiprocessors. ISCA,
22. Pacini, E., C. Mateos and C.G. Garino, 0000. Dynamic Scheduling based on Particle Swarm Optimization for Cloud-based Scientific Experiments.
23. Gillam, L., 2010. Cloud computing: Principles, systems and applications.
24. Gillam, L., B. Li, J. O'Loughlin and A.P. Singh Tomar, 2012. Fair Benchmarking for Cloud Computing Systems.
25. Zakarya, M., Uzma and A.A. Khan, 2013. Power Aware Scheduling Algorithm for Real-Time Tasks over Multi-Processors. Middle-East Journal of scientific Research, 15(1): 94-105.
26. Zakarya, M. and A.A. Khan, 2012. Cloud QoS, High Availability & Service Security Issues with Solutions. IJCSNS, 12(7): 71.
27. Shameem Akhter and Jason Roberts, 0000. Multi-Core Programming - Increasing Performance through Software Multi-threading, INTEL PRESS, ISBN 0-9764832-4-6.

28. Muhammad Zakarya, Nadia Dilawar and Naveed Khan, 0000. A Survey of Energy Efficient Load Balancing Algorithms over Multicores, International Journal of Research in Computer Applications & Information Technology, pp: 60-68.
29. Sathappan, O.L., P. Chitra, P. Venkatesh and M. Prabhu, 2011. Modified genetic algorithm for multiobjective task scheduling on heterogeneous computing system. International Journal of Information Technology, Communications and Convergence, 1(2): 146-158.
30. Munawar, A., M. Wahib, M. Munetomo and K. Akama, 2008. A survey: Genetic algorithms and the fast evolving world of parallel computing. In High Performance Computing and Communications, 2008. HPCC'08. 10th IEEE International Conference on, pp: 897-902. IEEE.
31. Randles, M., D. Lamb and A. Taleb-Bendiab, 2010. A comparative study into distributed load balancing algorithms for cloud computing. In Advanced Information Networking and Applications Workshops (WAINA), 2010 IEEE 24th International Conference on, pp: 551-556. IEEE.
32. Fang, Y., F. Wang and J. Ge, 2010. A task scheduling algorithm based on load balancing in cloud computing. In Web Information Systems and Mining, pp: 271-277. Springer Berlin Heidelberg.