International Journal on Recent and Innovation Trends in Computing and Communication ISSN: 2321-8169
Volume: 2 Issue: 11 3360 - 3365

Analysis of Various Decentralized Load Balancing Techniques with Node Duplication

Rajesh Tiwari
Associate Prof.
SSCET,Bhilai
raj_tiwari_in@yahoo.com

Dr. Manisha Sharma
Professor
BIT,Durg

Dr. Kamal K. Mehta
Associate Professor
NIRMA Univ. Ahamadabad

Abstract- Experience in parallel computing is an increasingly necessary skill for today's upcoming computer scientists as processors are hitting a serial execution performance barrier and turning to parallel execution for continued gains. The uniprocessor system has now reached its maximum speed limit and, there is very less scope to improve the speed of such type of system. To solve this problem multiprocessor system is used, which have more than one processor. Multiprocessor system improves the speed of the system but it again faces some problems like data dependency, control dependency, resource dependency and improper load balancing. So this paper presents a detailed analysis of various decentralized load balancing techniques with node duplication to reduce the proper execution time.

*****

1. Introduction

Parallel computers can be classified according to the level at which the hardware supports parallelism, with multi-core and multi-processor computers having multiple processing elements within a single machine, while clusters, MPPs, and grids uses multiple computers to work on the same task. Specialized parallel computer architectures are sometimes used alongside traditional processors, for accelerating specific tasks.

In addition to the trend of individual machines containing multiple processing units, there is a growing trend towards harnessing the power of multiple machines together to solve ever larger computational and data processing problems. Since the early to mid 1980's, many expensive computers containing multiple processors were built to solve the world's most challenging computational problems. Since the mid-1990's, people have built clusters of commodity PCs that are networked together and act as a single computer with multiple processors (which could be multicore themselves). This type of cluster of computers is often referred to as a beowulf cluster. Major Internet software companies such as Google, Yahoo, and Microsoft now employ massive data centers containing thousands of computers to handle very large data processing tasks and search requests from millions of users. Portions of these data centers can now be rented out at reasonable prices from Amazon Web Services, making it possible for those of us who can write parallel programs to harness the power of computers in 'the cloud'- out on the Internet away from our own physical location.

Today, even the next generation of netbooks and mobile phones, along with our current laptops and desktop machines, have dual core processors, containing two 'cores' capable of executing instructions on one CPU chip. High-end desktop workstations and servers now have chips with more cores (typicaly 4 or 8), which leads to the general term multicore to refer to these types of CPUs. This trend of increasing cores will continue for the foreseeable future - the era of single processors is gone.

Major classes of parallel computing platforms are multicore with shared memory, clusters of computers that can be used together and large data centers in the cloud.

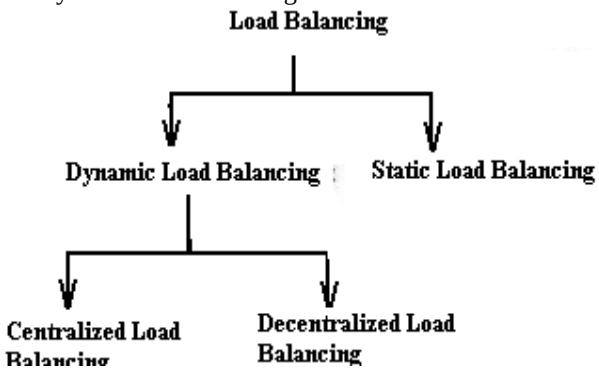
Parallel computing is a form of computation in which many calculations are carried out simultaneously, [13] operating on the principle where large problems can often be divided into smaller ones, which are then solved concurrently. There are several different forms of parallel computing: bit-level, instruction level, data level, and task level parallelism. Parallelism has been employed for many years, mainly in high-performance computing, but interest in it has grown lately due to the physical constraints preventing frequency scaling [2]. As power consumption (and consequent heat generation) by computers has become a concern in recent years, parallel computing has become the dominant paradigm in computer architecture, mainly in the form of multi-core processors [18].

Parallel computer programs are more difficult to write than sequential ones [16], because concurrency introduces several new classes of potential software bugs, of which race conditions are the most common. Communication and synchronization between the different subtasks are some of the greatest obstacles to getting good parallel program performance. The maximum possible speed-up of a program is easily calculated by Amdahl's law. According to Amdahl's law the speedup factor is inversely proportional to the sequential portion of the program. As the sequential portion if the program increases the speedup factor will decreases and vice-versa.

In addition, we consider multiprogramming operation, i.e. many independent parallel programs may run at the same time, sharing the machine in time and space. The programs may enter the system at any node and at any time. Each of these programs is assumed to consist of a dynamic set of parallel tasks that communicate with each other by interchanging messages. In such an environment, there exists the problem of how to (re)allocate the particular tasks to processors to achieve low response times.

2. Techniques of Load Balancing

There are two types of load balancing - static load balancing and dynamic load balancing.



Static Load Balancing- A parallel program with m communicating tasks and a multicomputer with n<m processors, the problem is to find a mapping of the tasks to the processors that minimizes the program's execution time. Heuristic algorithms for this NP-hard problem try to find a balanced allocation with minimal communication delays. However, they are designed to be executed off-line (e.g. in a separate step at link time running on the front-end of the parallel machine) and consider neither the actual load situation (due to multiprogramming) at run-time nor the data dependent behavior of the program. They are usually also unable to deal with dynamic task creation and deletion during program execution.

Dynamic Load Balancing- The other class of approaches originates from distributed systems where new tasks may enter the system at any time and at any node. The problem is to distribute the tasks as evenly as possible to avoid idling nodes. If, in addition, the tasks are elements of parallel programs, load balancing helps to process tasks at different processors at the same speed, which leads to low synchronization delays and thus to low program response times. Since the allocation is done at run-time, the basic mechanism is the migration of a task from one node to another which usually means the transfer of a considerable amount of data. Most of the proposals used a decentralized algorithm where load balancing agents at the particular nodes negotiate the possible transfer of tasks from overloaded to under loaded nodes. Inter task communication is usually ignored and if considered, it plays a minor role, since the algorithms are mostly dedicated to systems based on local area networks where all processors can be regarded as adjacent.

A task allocation mechanism needs a combination of features and goals of both classes.

Advantages of load balancing-

i. The overall system performance is enhancing by improving the performance of each node.
ii. Load balancing between processes minimize the process idle time.
iii. Long starvation time is avoided for small jobs.
iv. Resource utilization of available resources are very high with short response time.
v. Load balancing system having high Throughput, reliability with low cost.
vi. Load balancing uses extendable and incremental approach.

3. Goals of Load Balancing Algorithms

Important goals of load balancing algorithms:

i. Performance Improvement- reducing task response time while keeping acceptable delay.
ii. Job equality- equal treatment for all jobs in system beside of considering their origin.
iii. Fault tolerance- partially failure of system will not have endurance on performance.
iv. Modifiability- this has the ability to modify.

4. Literature Survey

Parallel programming is the mechanism in which more than one task can be performed in parallel. Here the load is balanced with decentralized load balancing mechanism.
The literature survey was done for the last 17 years from 1995 to 2012 in the area of parallel programming. In this paper a detailed analysis of 25 research paper is presented.
Manekar et. al. (2012) has discussed on OpenMp for shared memory computer model, MPI a message passing library for distributed memory computer model on cluster to solve multiple task simultaneously and bigger problem is based on design pattern's ,load balancing strategies and their merits and demerits. Based on this review researchers concluded that while distributing load in Static load balancing, runtime system does not need to know in advanced, whereas in Dynamic load balancing runtime overload is gathering information and distribution of task to different processes. Author concluded that if load change mechanism is faster than Static load balancing gives best performance and if load change mechanism is slower than dynamic load balancing gives best performance.

Reddy et. al (2012) explains that High Performance Computing(HPC) is required when system wants to solve complex computational problems. HPC shall be performed through the Parallel and distributed Algorithms. The class of Algorithms and class of Computer Architecture was explained in this paper. The Programming concepts like threads, fork, sockets and Par were used for HPC.

Barrett et. al (2012) suggested a MiniGhost which is a miniapp developed within the scope of the Mantevo project. It was designed to provide a means to explore the Bulk Synchronous Parallel programming model, supplemented with message aggregation, in the context of exchanging inter-process boundary data typically seen in finite difference and finite volume computations. This programming model was employed across a breadth of science domains, typically for solving partial differential equations. It also provides a means for exploring alternative programming languages as well as alternative semantics of MPI.

Radenski et. al (2011) introduces three parallel versions of recursive merge sort: shared memory (with OpenMP), message-passing (with MPI) and hybrid (with MPI and OpenMP). While others had developed merge sort algorithms with either multi-threading or message-passing , this paper

offers comparable multi-threaded, message-passing, and hybrid implementations. This paper reports performance experiments with the three approaches and draws conclusions accordingly. Experiments showed that shared memory merge sort (with OpenMP) is faster than message-passing merge sort (with MPI) when applied to arrays that fit entirely in RAM; the performance of hybrid merge sort falls between that of shared merges sort and message passing merge sort. The best programming paradigm depends on the nature of the problem, the hardware and software in cluster nodes, and the cluster network.

Yildirim et. al (2011) explained parallel transfer behavior of a TCP-based protocol, GridFTP, over wide area networks analyzed and several prediction models are presented and improved according to the characteristics of the transfers. It had been observed that the aggregate throughput starts to fall down in existence of congestion and none of the models could mimic this behavior. By using minimum information on historical results, authors had improved the current models to predict the throughput curve of GridFTP and observed promising results. Based on the results of their experiments, they conclude that they are able to predict the throughput behavior of parallel transfers with Full Second Order and Newton's Iteration with very good accuracy and with a limited data size of historical transfers.

Traff et. al(2010) discusses that users of MPI often complain about the poor performance of some of the MPI functions in their MPI libraries. They proposed self-consistent MPI performance guidelines to help in ensuring performance consistency of MPI libraries and a degree of performance portability of application codes. Conformance to such guidelines could, in principle, be checked automatically. It was seen that good MPI implementations are developed with some guidelines in mind, at least implicitly, but authors also showed examples of MPI libraries and systems where some of the rules are violated.

Bruneo et. al (2010) discussed that Grid computing implies the respect of stringent performance requirements and the adoption of optimization strategies, mainly if adopted in the business context. Assuming the gLite middleware as the specific reference environment, this paper addresses the problem to define and analyze an analytical model to investigate the performance of a gLite cluster under varying working conditions. A GSPN model had been developed that exploiting synchronization and concurrency provides a flexible and powerful way toward the evaluation of interesting performance indices such as the job waiting time, the drop probability, the site utilization under different scheduling policies, both in case of single and MPI jobs.

Tiwari et. Al (2009) concluded from comparative chart for scientific functions , that the parallel execution time is less as compare to sequential execution time. Further division of job into number of workers results less execution time. Theoretically, increasing the number of workers may reduce the execution time, but the operational observation does not follow this theory. Back end process takes time of CPU and practical output does not come as the theory.

Walters et. Al (2009) discussed that effective checkpoint MPI application using the LAM/MPI implementation with low overhead. Previous checkpointing implementations have typically neglected the issue of checkpoint storage. Author comprehensively addressed this issue with a comparison against all major storage techniques, including SAN- based strategies and commercial parallel file systems. This paper shows replication implementation had proven to be highly effective and resilient to node failures. This method was not scalable, particularly after 64 CPUs.

Uehara et. al(2009) developed an infrastructure which accelerates the research and development of many-core processors. Author described three main elements provided by their infrastructure. The first element was the definition of simple many-core processor architecture called M-Core. The second was SimMc, a software simulator of M-Core. The third was the software library MClib which helps the development of application programs for M-Core. Using SimMc, software performance on many-core processors with various number of cores can be evaluated. Using three benchmark programs of Equation Solver Kernel, Matrix Multiply and Himeno benchmark, the parallelization efficiency of M-Core and simulation speed of SimMc were evaluated with up to 64 nodes. As expected, as the number of nodes increases, the number of execution cycle is decreased.

Adve et. al(2008) explained main techniques for power consumption performance benefits – increased clock frequency and smarter but increasingly complex architectures – are now hitting the so-called power wall. The computer industry accepted that future performance increases must largely come from increasing the number of processors (or cores) on a die, rather than making a single core go faster.

Chandra et. al(2008) showed the performance of system under different type of load like I/O as well as CPU, MEMORY based on IOCM dynamic load balancing algorithm in heterogeneous computing system. There are number of different dynamic load balancing techniques for cluster systems; their efficiency depends on topology of the communication networks that connects nodes. This research had developed an efficient load balancing for I/O, CPU- and MEMORY-intensive tasks. For this author developed a new way to predict and calculate the load of cluster nodes. The proposed load balancing scheme aim to achieve the effective use of global disk resources in the cluster. This can minimize the average slow down of all parallel jobs running on a cluster and reduce the average response time of the jobs.

Bridgewater et. al(2007) explained Balanced overlay networks (BON) which was a novel decentralized load-balancing approach that encodes the balancing algorithm in the evolving structure of the graph that connects the resource-bearing nodes. BON is scalable, self-organized, and relies only on local information to make job assignment decisions. New jobs were assigned to a node by a random walk on the graph which not only samples the graph preferentially but also selects the highest-degree node that was visited on the walk. Each node's unused resource was proportional to its degree, so this approach works very well when a network is not loaded beyond its clipping point. When a BON is clipped, the relationship between load and in-degree breaks down, but the

balancing performance remains quite good due to the so-called "power of two choices" in ball-bin load balancing.

Dhakal et. al(2007) discussed a continuous-time stochastic model that had been formulated for the queues' dynamics of a distributed computing system in the context of load balancing. The model takes into account the randomness in delay and allows random arrivals of external loads.

Guo et. al(2006) developed scheduling and load balancing algorithms to ensure high throughput and low jitter for multimedia processing on a cluster-based Web server where a few computing nodes are separately reserved for high-performance multimedia applications. Authors considered the multimedia streaming service which requires on-demand transcoding operations as an example. In this paper, authors designed and implemented a media cluster and evaluated the efficiency of seven load scheduling schemes for a real MPEG stream transcoding service.

Krste et. al(2006) explained increasing clock frequency is the primary method of improving processor performance. Increasing parallelism is the primary method of improving processor performance. Even representatives from Intel, a company generally associated with the 'higher clock-speed is better' position, warned that traditional approaches to maximizing performance through maximizing clock speed have been pushed to their limit.

Pinar et. al(2005) posed and addressed the problem of distributing flexibly assignable work to processors to minimize load imbalance. This paper considers the problem in a general form, whereas exploiting problem-specific information might yield more efficient solutions. For instance, in the molecular dynamics application and in many other cases, each task can be assigned to one of atmost two processors. Authors identified new sources of flexibly assignable tasks where their techniques can be used to improve load balance.

Barker et. al(2004) presented the Implicit Load Balancing (ILB) component of the PREMA runtime system and demonstrated its effectiveness in balancing the coarse-grained computation of a parallel 3D advancing front mesh refinement program. They demonstrated performance improvements of 15 percent over traditional stop-and- repartition methods, 30 percent over intrusive explicit load balancing methods, and 42 percent over no load balancing on configurations of 128 processors.

Lastovetsky et. al(2002) presented new advanced features of the mpC parallel language that allow the programmer to define all main features of the implemented parallel algorithm that can have an impact on the performance of execution of the algorithm on a heterogeneous network of computers. The mapping is executed at runtime; therefore its efficiency is crucial for the total execution performance of mpC applications. The presented model of a heterogeneous network and the mapping algorithm were developed to keep balance between accuracy and efficiency.

Pasqua et. al(2002) discussed the continuous evolution of computing and networking technologies makes available to an increasing community parallel and distributed systems that can deliver impressive performance, along with a large amount of software, data and other resources. However, using such resources in an efficient way is generally assessed to be a very hard task. The development of efficient software requires a deep knowledge of the architectures at hand and a consistent programming effort to deal with architecture-related details, such as memory hierarchy access, process decomposition, scheduling and mapping, inter-processor and inter-system communication implementation, and load balancing.

Chow et. al(2002) explained research collocates which was the concept of load balancing and agent theory into one perspective. The discussion was followed by an introduction of multiagent cluster system test platform built from scratch and the design of a novel load-balancing algorithm. One limitation of this paper was that authors assume a static communication pattern. In reality, agents may change their communication patterns dynamically. Thus, a further research direction is to investigate the effectiveness of the Comet algorithm under such a more dynamic situation. Another avenue of further research was to incorporate language constructs in devising the load- balancing strategies and agent migration decisions.

Das et. al(2001) described a novel topology independent approach to solving the dynamic load balancing problem for adaptive meshes. Their thorough experimental investigation with an unstructured adaptive mesh application showed that the proposed SBN-based load balancer achieves a lower redistribution cost than that under the PLUM environment. This was possible by overlapping processing and data migration. However, the communication costs using SBN were significantly higher than those reported under PLUM. A vertex was usually redistributed to a processor that owns an adjacent vertex. While diffusive strategies were fairly common, scratch-remap techniques had also been used successfully to load balance adaptive mesh applications.

Heiss et. al(1995) proposed a new decentralized heuristic algorithm to the problem of dynamically distribute tasks of parallel programs in large multicomputer systems with multiprogramming. In contrast to other load balancing schemes it reflects communication patterns among the tasks of the parallel programs. Experiments based on simulations for 2D-grid and hypercube topologies confirmed the necessity to account for communication and migration overhead in order to achieve low response times. Especially for high loads, load balancing alone was not sufficient and the consideration of the additional forces can reduce the response times by up to 50%.
From the literature survey it concluded that research can still be done in the area of optimizing load balancing by parallel programming.

5. Performance Matrices

The performance of the load balance system is measured in terms of execution time speedup, efficiency, throughput etc.

Execution Time: This is the total time spend by the system to solve a specific problem.

$$\text{i.e. } t_p = t_{comm} + t_{comp}$$

 ISSN: 2321-8169
3360 - 3365

where t_{comm} is the communication time and t_{comp} is the computation time.

$$t_{comm} = t_{startup} + w\, t_{data}$$

where $t_{startup}$ is the time to spend in binding data at source end and unbinding data at receiving end. t_{data} is the time to send a single data from source to destion. w is the number of data elements.

Speedup : This is the ratio of time which is used to spend to execute a program on single processor (t_s) to execute the same program on multiple processors (t_p).

$$\text{i.e. } S_k = \frac{(t_s)}{(t_p)}$$

Efficiency : Speedup divided by the number of processors is called the efficiency of the system.

Throughput : The number of results produced per unit time is called the throughput of the system.

Unbalanced Load: It means the load is not distributed to all the available processors as shown in fig. 1. P1, P2, P3, P4 are the processors.

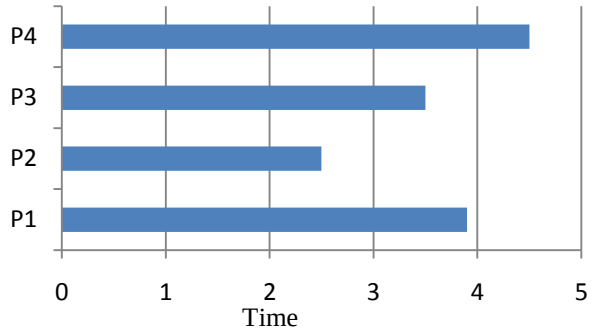


Fig. 1 : Unbalanced Load

Balanced Load : It means the load is equally distributed to all the available processors as shown in fig. 2. Here the load of P1, P2, P3, P4 processors is almost equal. It will help to reduce the execution time.

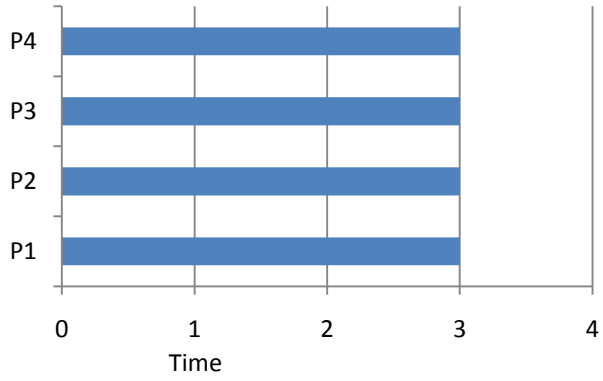


Fig. 2 : Balanced Load

6. Comparison of Static vs. Dynamic Load Balancing

Next table shows an analysis of static and dynamic load balancing issues :

Factors	Static Load Balancing	Dynamic Load Balancing
Nature	Work load is assigned to the processor at compile time	Work load is assigned to the processor at run time
Overhead involved	Overhead due to IPC is less	Overhead due to process redistribution is maximum.
Resource utilization	Resource utilization is low	Resource utilization is high
Processor thrashing	There is no process thrashing	There is considerable amount of process thrashing
Predictability	Easy to predict	Difficult to predict
Adaptability	It is less	It is high
Reliability and Response time	It is less	It is high
Stability	More stable	Less stable
Complexity	Complexity is low	Complexity is high
Cost	Less expensive	More expensive
Speedup factor	Low speedup	High speedup
Power Consumption	It's high	It's low

7. Conclusion-

In this paper various load balancing techniques have been studied. It is concluded that dynamic load balancing technique improves the system performance in terms of speedup, efficiency, throughput etc. In addition Dynamic decentralized load balancing provides an additional advantage science the master computer does not control all the activity of slave computers. Slave computers are controlled by sub-masters in decentralized load balancing. Analysis of work done by various researchers shows that work has been done with maximum 64 processors. In future work it is suggested to implement load balancing technique for more than 64 processors.

References

[1] Abramson David et al.2011. *Parameter Exploration in Science and Engineering Using Many - Task Computing* .IEEE Transactions on Parallel and Distributed Systems .22(6) : 960 – 973 .

[2] Adve S.V. et al. 2008. *Parallel Computing Research at Illinois: The UPCRC Agenda* . University of Illinois at Urbana-Champaign.

[3] Armando E. et al. 2005. *Dynamic Load Balancing in Parallel Processing on Non – Homogeneous Clusters*. Journal of Computer Science. 5(4) :272-278.

[4] Barker Kevin, Chernikov Andrey, Chrisochoides Nikos, and Pingali Keshav.2004. *A Load Balancing Framework for Adaptive and Asynchronous Applications*. IEEE

Transactions On Parallel And Distributed Systems .15(2) : 183 – 192.

[5] Barrett Richard F., Vaughan Courtenay T. and Heroux Michael A. 2012. *MiniGhost: A Miniapp for Exploring Boundary Exchange Strategies Using Stencil Computations in Scientific Parallel Computing* . Sandia Report, SAND2012-2437, Unlimited Release : 1-31.

[6] Bridgewater Jesse S.A., Boykin P. Oscar Boykin and Roychowdhury Vwani P.2007. *Balanced Overlay Networks (BON): An Overlay Technology for Decentralized Load Balancing*. IEEE Transactions on Parallel and Distributed Systems . 18(8) : 1122 – 1133.

[7] Bruneo Dario, Scarpa Marco and Puliafito Antonio. 2010. *Performance Evaluation of gLite Grids through GSPNs* . IEEE Transactions on Parallel and Distributed Systems. 21 (11) : 1611 – 1625.

[8] Chandra Pushpendra Kumar and Sahoo Bibhudatta. 2008. *Dynamic load distribution algorithm performance in heterogeneous distributed system for I/O- intensive task* . TENCON 2008. IEEE Region 10 Conference. : 1 – 5.

[9] Chow Ka-Po and Kwok Yu-Kwong. 2002.*On Load Balancing for Distributed Multiagent Computing* . IEEE Transactions on Parallel and Distributed Systems .13(8) : 787 – 801.

[10] Das Sajal K., Harvey Daniel J. and Biswas Rupak. 2001. *Parallel Processing of Adaptive Meshes with Load Balancing*. IEEE Transactions on Parallel and Distributed Systems. 12(12) : 1269 – 1280.

[11] Dhakal Sagar, Hayat Majeed M., PezoJorge E. a, Yang Cundong and Bader David A.2007. *Dynamic Load Balancing in Distributed Systems in the Presence of Delays: A Regeneration-Theory Approach*. IEEE Transactions on Parallel and Distributed Systems. 18(4) : 485 – 497 .

[12] Duatot Pierre-Francois et al. 2009. *Scheduling Parallel Task Graphs on (Almost) Homogeneous Multicluster Platforms*. IEEE Transactions on Parallel and Distributed Systems . 20(7) : 940 – 952 .

[13] Gottlieb, Allan; Almasi, George S. 1989. *Highly parallel computing*. Redwood City, Calif.: Benjamin/Cummings. ISBN 0-8053-0177-1.

[14] Guo Jiani and Bhuyan Laxmi Narayan.2006. *Load Balancing in a Cluster-Based Web Server for Multimedia Applications.* IEEE Transactions on Parallel and Distributed Systems . 17(11) : 1321 – 1334.

[15] Heiss Hans-Ulrich and Schmitz Michael. 1995. *Decentralized Dynamic Load Balancing: The Particles Approach*. Information Sciences. 84(1-2) : 1 – 6.

[16] Hennessy, John L., Patterson, David A., Larus, James R. 1999. *Computer organization and design : the hardware/software interface* (2. ed., 3rd print. ed.). San Francisco: Kaufmann. ISBN 1-55860-428-6.

[17] Hey Anthony J. G. et al. 1998.*Integrating Computation and Information Resources – An MPP Perspective* .IEEE Conf. : 174-184 .

[18] Krste Asanovic, et al. 2006. *The Landscape of Parallel Computing Research: A View from Berkeley*. University of California, Berkeley. Technical Report No. UCB/EECS-2006-183.

[19] Lastovetsky Alexey et al. 2002.*Adaptive Parallel Computing on Heterogeneous Networks with mpC*. ELSEVIER , Parallel Computing (28) : 1369-1407.

[20] Liao Ching-Jung, and Chung Yeh-Ching.1999. *Tree-Based Parallel Load-Balancing Methods for Solution-Adaptive Finite Element Graphs on Distributed Memory Multicomputers*. IEEE Transactions on Parallel and Distributed Systems. 10(4) : 360-370.

[21] Manekar Amit Kumar S et al. 2012. *A Pragmatic Study and Analysis of Load Balancing Techniques In Parallel Computing*. International Journal of Engineering Research and Applications .2(4) : 1914-1918. ISSN: 2248-9622.

[22] Min He et al. 2010. *Optimal Sorting Algorithms for a Simplified 2D Reconfigurable Pipelined Bus System* . IEEE Transactions on Parallel and Distributed Systems. 21(3) : 303 – 312.

[23] Pasqua D'Ambra et al. 2002 .*Advanced Environments for Parallel and Distributed Applications : A View of current Status* .ELSEVIER , Parallel Computing (28) : 1637-1662.

[24] Pinar Ali and Hendrickson Bruce . 2005. *Improving Load Balance with Flexibly Assignable Tasks*. IEEE Transactions On Parallel And Distributed Systems. 16(10) : 956 – 965.

[25] Radenski Atanas. 2011. *Shared Memory, Message Passing and Hybrid Merge Sorts for Standalone and Clustered SMPs* .The 2011 International Conference on Parallel and Distributed Processing Techniques and Applications. CSREA Press : 367 – 373.

[26] Reddy P. Venkata Subba. 2012. *Computer Architecture and Algorithms for High Performance Computing through Parallel and Distributed Processing* . Journal Of Computer Science And Engineering. 12(1) :5 – 8.

[27] Ros Alberto et al. 2010. *A Direct Coherence Protocol for Many-Core Chip Multiprocessors.* IEEE Transactions on Parallel and Distributed Systems. 21(12) :1779 – 1792 .

[28] Tiwari Rajesh, Raja Rohit, Behar Nishant , Mehta Kamal. 2009.*Concurrent Solution of SPMD problem USING MATLAB*. Journal of Advance Research in Computer Engineering: An international journal . 3(2) : 209-212.ISSN – 0974 - 4320.

[29] Traff Jesper Larsson , Gropp William D. and Thakur Rajeev.2010. *Self-Consistent MPI Performance Guidelines*. IEEE Transactions on Parallel and Distributed Systems. 21(5) : 698 – 709.

[30] Uehara Koh, Sato Shimpei, Miyoshi Takefumi, and Kise Kenji. 2009. *A Study of an Infrastructure for Research and Development of Many-Core Processors* . International Conf. on Parallel and Distributed Computing , Application and Technology : 414 – 419.

[31] Walters John Paul and Chaudhary Vipin.2009.*Replication-Based Fault Tolerance for MPI Applications* . IEEE Transactions on Parallel and Distributed Systems. 20(7) : 997 – 1010 .

[32] Warneke Daniel et al. 2011. *Exploiting Dynamic Resource Allocation for Efficient Parallel Data Processing in The Cloud*. IEEE Transactions on Parallel and Distributed Systems. 22(6) : 985-997.

[33] Yildirim Esma, Yin Dengpan, and Kosar Tevfik. 2011. *Predication of Optimal Parallelism Level in Wide Area Data Transfers.* IEEE Transactions on Parallel and Distributed Systems. 22(12) :2033- 2045.