VOL. 8, NO. 1, JANUARY 2013 ISSN 1819-6608
ARPN Journal of Engineering and Applied Sciences


www.arpnjournals.com

PERFORMANCE EFFICIENT HETEROGENEOUS MULTI CORE SCHEDULING STRATEGY BASED ON GENETIC ALGORITHM

A. S. Radhamani[1] and E. Baburaj[2]
[1]Department of Computer Science and Engineering, Manonmanium Sundaranar University, Tirunelveli, Tamil Nadu, India
[2]Department of Computer Science and Engineering, Sun College of Engineering and Technology, Nagercoil, Tamil Nadu, India
E-mail: asradhamani@gmail.com

ABSTRACT

Multi-core processors offer a significant performance increase over single core processors. Therefore, they have the potential to enable computation-intensive real-time applications with stringent timing constraints that cannot be met on traditional single-core processors. However, with the number of cores on a single chip continuing to increase, it has been a great challenge to effectively manage the energy efficiency of multicore based systems. Power and temperature management are also two concerns that can increase exponentially with the addition of multiple cores. Design innovations in multicore processor architectures bring new optimization opportunities and challenges in the computing era. System performance will be further enhanced by addressing these challenges. In particular, the process (task) scheduler is one of the critical challenge is garnering great interest. High performance in a heterogeneous multicore system is essential which is achieved by effective scheduling, which remains a challenging problem. Further multi core technology opens research opportunities for energy reduction through efficient scheduling. There may be different hardware and software solutions for the above issue; hardware solutions are based on adjusting dynamic voltage per core, alternatively software approach includes, scheduling task among cores, in heterogeneous environment. Task scheduling in multicore architecture is an extremely difficult problem, because it requires a large combinatorial search space and also precedence constraints between the processes; for the effective utilization of multi core processor system, efficient assignment and scheduling of jobs is more important. Many of the existing algorithms are not focused on task scheduling and core utilization in heterogeneous multi core systems. This paper formulates task scheduling as an optimization problem and the results are compared with the earlier faster scheduler in use. Findings show that, in addition to its optimum solution for large scale problem, the Genetic Algorithm (GA) proposed here fits the heterogeneous multi core parallel scheduling problem of minimizing the completion time as well as in effective core utilization.

Keywords: heterogeneous multi core, task scheduling, genetic algorithm.

1. INTRODUCTION

The board trend in processor development has moved from dual-, tri-, quad-, hexa-, octo-core chips to ones with tens or even hundreds of cores. Besides, promising performance and efficiency gains of multicore processors in processing multimedia, recognition and networking applications can be achieved with the characteristics of multithreading, memory-on-chip, and special-purpose heterogeneous cores. There is also a challenge of improving energy-efficiency by focusing on performance-per-watt with advanced fine-grain or ultra fine-grain power management and dynamic voltage and frequency scaling (i.e., notebook computers and portable media players). However, many issues remain unsolved. In order to utilize a multicore processor at the maximum extent the applications run on the system must be scheduled.

Scheduling, in general, is concerned with allocation of resources to certain tasks to optimize few performance criterions, like the completion time, waiting time or cost production. In scheduling, task scheduling is measured as popular unsolved problem for heterogeneous multicore processors. The importance of scheduling has increased in recent years due to the extravagant development of new process and technologies. Scheduling, in multiprocessor architecture, can be defined as assigning the tasks of precedence constrained task graph onto a set of processors and determine the sequence of execution of the tasks at each processor. This multi core scheduling problem is known to be Non-deterministic Polynomial (NP) complete except in the case of large scale problems.

Because of, the classical algorithms are not dynamic, they cannot achieve the optimal scheduling for all situations, and therefore these algorithms cannot adapt themselves with all situations. This difficulty can be widely solved by Genetic algorithm. In the field artificial intelligence, Genetic Algorithm (GA) is a search heuristic that mimics the process of natural evolution. This heuristic is routinely used to generate useful solutions to optimization and search problems. Genetic algorithms belong to the larger class of Evolutionary Algorithms (EA), which generate solutions using procedure stimulated by natural evolution, such as inheritance, mutation, selection, and crossover to optimization problems. In most cases the methods are quite effective but not efficient enough, and some important aspects such as the time of transferring data between processes are ignored. Even in some researches, which almost all aspects are considered, only part of the problem is solved by genetic algorithm. The paper presents a scheduling problem in multi-core systems in a new intelligent method so as to improve the present scheduling for minimizing the total completion time of all process and maximizing their utilization.

2. RELATED WORK

Several research works has been carried out in the past decades, in the heuristic algorithms for job scheduling and generally, since scheduling problems is NP-hard i.e., as the problem size increases the time required to complete the problem also increases exponentially, therefore it is extremely significant and necessary to develop algorithms to fine solution to these problems. Some heuristic methods like branch and bound and prime and search [1], have been proposed earlier to solve this kind of problem. Also, the major set of heuristics for job scheduling onto multiprocessor architectures is based on list scheduling [2-8] [9]. However the time complexity increases exponentially for these conventional methods and becomes excessive for large problems. Then, the approximation schemes are often utilized to find an optimal solution. It has reported in [2, 6] that the critical path list scheduling heuristic is within 5% of the optimal solution 90% of the time when the communication cost is ignored, whereas in rare cases any list scheduling is within 50% of the optimum solution. General task scheduling for cloud applications with multicore processors are implemented in [10].

It is worth to mention that due to sever complexity of scheduling in multi core process; the proposed classical algorithms do not have high performance. Therefore the intelligent methods have been used to find the solution of scheduling problem. In these methods the genetic algorithm is widely used repeatedly [11-14]. The problem based on scheduling of independent jobs on non-identical parallel machines in order to minimize makespan is described in [15]. In [16], new Improved PSO (ImPSO) algorithm is used for solving job scheduling in multiprocessor architecture with the objective of minimizing the job finishing time and waiting time. Multiprocessor task scheduling problem can be stated as finding a schedule for a general task graph to execute on a multiprocessor system so that the schedule length can be minimized. Hybrid Flow Shop Scheduling with Multiprocessor Task (HFSM) problem is known to be NP-hard. In this study [17] an effective parallel greedy algorithm to solve HFSMT problem is presented.

3. PROBLEM DEFINITION

The job scheduling problem of multi processor architecture is scheduling problem to partition the tasks between different cores by accomplishing minimum completion time and minimum waiting time simultaneously. If M different cores M = {ci, i = 1, 2….m} and T different tasks T = {tj, j = 1, 2…n} are considered in a heterogeneous environment, every core works on different speeds and processing capabilities. It is assumed that the cores C1 is much faster than C2, C3 and so on. If the processing speed is V (i, j) then the execution time has calculated on the basis of size of the tasks by processing speed on different core. Completion Time (or) Processing Time = Task size/Speed of the core.

Maximum Completion Time

$$(C_{\max}) = \max_{i=1}^{m} \left(\sum_{j=1}^{n} (T(i, j)/V(i, j))\right) \quad (1)$$

To evolve good solutions and to implement natural selection, we need a measure for distinguishing good solutions from bad solutions. This can be mathematical model or a computer simulation and the measure is said to be objective function.

Therefore the devised objective function is described below.

$$\text{Min}\,(C_{\max}) = \max_{i=1}^{m} \left(\sum_{j=1}^{n} (T(i, j)/V(i, j))\right) \quad (2)$$

To maintain the concept of load balancing, effective core utilization is desired. The average utilization is calculated based on the individual performance of the core. The individual core utilization is given by

Utilization = Completion time/ Max.span

An initial population is the set consisting of 15 chromosomes. The chromosomes number, i.e., population size, is one of the important parameters of GA. The efficiency of the algorithm is highly dependent on their "quality" of members of the initial population (chromosomes) which are the parents of the next generations Genetic Algorithm was involved with the number of populations to be 100 and 800 generations. The crossover rate was 0.6 and the mutation rate was 0.01. Randomly the populations were generated and for various trails of the number of cores and task sizes, the completed fitness values of execution time is calculated.

4. CACHE FAIR THREAD SCHEDULING WITH WAIT FREE DATA STRUCTURE (CFTS WF)

At present there are different types of scheduling algorithms are available viz shortest job first, round robin, etc, each with its own merits and demerits. Scheduling algorithms to be of priority based since tasks of some category need an immediate attention while others can be listened later. All threads accessing the wait free data structure can complete its process within a restricted number of steps not considering the behavior of other threads. Therefore when batching CFTS scheduling with wait free data structure where all the threads are given a priority based on the either by the programmer (via system API) or the operating system. There are several scheduling queues are available each with its own priority. Initially the scheduler tries to schedule threads waiting with top priority queue then subsequently queue with low priority. Thus no queue left unattended. In CFTS, if a thread is ready to execute but the only available cores are not in its processor affinity set, then the thread is forced to wait. To satisfy this requirement, Cache Fair Thread Scheduling (CFTS) is integrated with Wait Free data structure to

access the internal scheduler and to achieve optimal efficiency in terms of energy, processing power, and line rate and network traffic stability. Hence CFTS-WF schedules such that maximum throughput, minimum response time, minimum waiting time and utmost CPU consumption is obtained [10].

5. GENETIC ALGORITHMS FOR SCHEDULING

Genetic algorithms are a kind of random search algorithms coming under evolutionary strategies which uses the natural selection and gene mechanism in nature for references. The key concept of genetic algorithm is based on natural genetic rules and it used random search space. GA was formulated by J. Holland with a key advantage of adopting population search and exchanging the information of individuals in population [11, 12 and 14]. The algorithm used to solve scheduling problem is as follows: The flowchart depicting the approach of genetic algorithm is as shown in Figure-1.

Step-1: Initialize the population to start the genetic algorithm Process. For initializing population, it is necessary to input number of processors, task size and core speed.

Step-2: With the generated populations the fitness function is calculated. For the defined problem, the fitness function can be obtained using the equation $f_i = \alpha \times e^{-\beta x Emax}$ where 'α and β' should be set to select an appropriate positive real number for ensuring the fitness of all good individuals to be positive in the solution space. And Emax is the objective function value.

Step-3: Perform selection process to select the best individual based on the fitness evaluated to participate in the next generation and eliminate the inferior. The job with the minimal execution time and waiting time is the best individual corresponding to a particular generation.

Step-4: Two crossover points are generated uniformly in the mated parents at random, and then the two

Step-5: In this step, mutation operation is performed to further create new offsprings, which is necessary for adding diversity to the solution set. Using turn over operation mutation is performed. Generally, mutation is adopted to avoid loss of information about the individuals the process of evolution. Mutation is performed by setting a random selected job to a random processor.

Step-6: Next stopping criteria must be tested. Stopping condition may be obtaining the best fitness value with minimum execution time and minimum waiting time for the given objective function. If the stopping criterion is satisfied then go to step-7 else go to step-2.

Step-7: From the completed generations optimum result is declared. Stop.

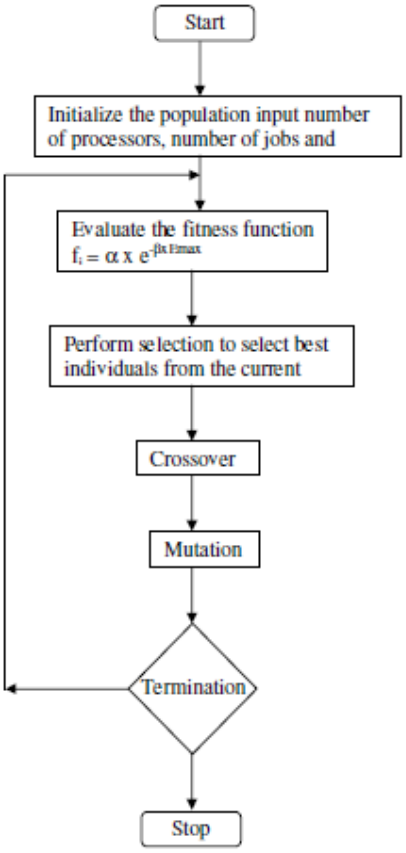

Figure-1. Flowchart for Genetic Algorithm

6. SCHEDULING HETEROGENEOUS MULTI CORES USING GENETIC ALGORITHM

High performance in a heterogeneous multicore system can be achieved by an efficient scheduling onto the process which minimizes the execution time and also average of response time. In this method genetic algorithm present the optimal scheduling directly from the final chromosome in final generation. A matrix X is composed of variables x (i, j) and it has following significance. The row I of the matrix X consists of n cores to process the different task based on the core speed. Rows are called "genes" (c1, c2, c3….cg…cn) and they represent cores to i, (ci) depend on size of the task and speed of the core.

A. Initial population

An initialization operator is applied first. An intermediate population of n "parent" individuals is created by the initialization operator. Then n independent removal of an individual from the old population is carried out [18] to produce "parents". Each individual being removed must be in linear relationship with the fitness of that individual. The average individuals which lie in the above range should have more copies in the new population; while below average individuals should have few to no copies present. The selection operator chooses two members of the present generation to participate in later crossover and mutation operations. Initialization

process raises the issue of fitness function. A requirement of the selection methods is that the probability f_i of an organism must be the best to be selected.

B. Fitness evaluation

The performance of an individual with respect to the current optimum so that different individuals can be compared to get a numerical value is called the fitness value. There exists a range of solutions for fitness from worst to best. The degree of success in the application of evolutionary algorithms depend significantly on the definition of a fitness that changes neither too rapidly nor too slowly wit the design parameters of the optimization problem. Performance evaluation of chromosomes is merely based on fitness function. After the generation of new population, fitness value of each chromosome is calculated (f_j). The higher the fitness value, the better the performance of the chromosome (i.e., parent). Hence, parents with higher fitness value have more chances to survive. Because the objective function is to minimize the execution time, fitness values can be obtained using the following function.$f_i = \alpha \times e^{-\beta x Emax}$, where α and β are positive real number and E_{max} is the objective function value (execution time) of the chromosome k.

C. Reproduction

Based on roulette-wheel selection, is a genetic operation for selecting potentially useful solutions for recombination. In fitness proportionate selection, as in all selection methods, the fitness function assigns fitness to possible solutions or chromosomes. This fitness level is used to associate probability of selection with each individual chromosome. Selection probability is given below if f_i is the fitness of individual i in the population,

$$P_i = \frac{f_i}{\sum_{j=1}^{N} f_j} \quad (3)$$

where N is the number of individuals in the population.

This could be imagined similar to a Roulette wheel in a casino. Based on their fitness value a proportion of the wheel is assigned to each of the possible selections. Normalization could be achieved by dividing the fitness of a selection by the total fitness of all the selections.

Table-1 shows the selection probability for 12 individuals, linear ranking and selective pressure of 2 together with the fitness value. Individual 1 is the fit individual and occupies the largest interval, whereas individual 11 as the second least fit individual has the smallest interval on the line. Individual 12, the least fit interval, has a fitness value of 0 and get no change for reproduction for selecting the mating population the appropriate number of uniformly distributed random numbers (uniform distributed between 0.0 and 1.0) is independently generated.

Table-1. Selection probability and fitness value.

Number of individual	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
Fitness value	2.2	2.0	1.8	1.6	1.4	1.2	1.0	0.8	0.6	0.4	0.2	0.0
Selection probability	0.2	0.19	0.17	0.15	0.13	0.11	0.09	0.07	0.06	0.03	0.02	0.0

D. Recombination (Crossover) operators

A new generation is formed by (a) selecting, according to fitness values, some of the parents and offspring and (b) rejecting others so as to keep the population size constant. Filter chromosomes have higher probabilities of being selected. Crossover is the major genetic operator which works on two chromosomes at a time and generates offspring by combining both chromosomes features. A simple way to achieve crossover would be to choose a random cut-point and generate the offspring by combining the segment of one parent to the left of cut-point with the segment of the other parent to the right of the cut-point. This method works well with the bit string representation. To a great extent the performance of genetic algorithm depends, based on the performance of the crossover operator used. In our method, crossover operator deals with genes and not chromosomes like most of the applications in literature. Each gene "i" consists of n cores to execute the various task on core i, ($x(i, j) \neq 0$, j = 1, 2…n); each chromosome contains m genes and each one consists in one feasible schedule. Crossover operator described here combines two genes of same chromosome, in proper order to obtain a new chromosome giving a better solution. Therefore crossover operation is carried out by moving one core to another for different task size and core speed to get a better solution. Figure-2 depicts the process of arithmetic cross over for different cores.

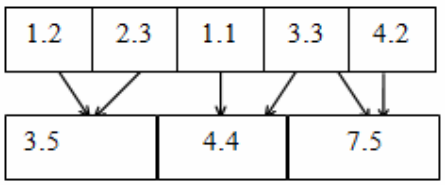

Figure-2. Arithmetic cross over.

The first gene to combine, gi, is the one that indicates the core with maximum execution time:

$$F(gi') = \max_{i=1}^{m} (f(gi)) \quad (4)$$

The core with the minimum processing time of gene gi is given by the equation below:

$$P(i', j') = \min_{i=1}^{m} P(i', j') \quad (5)$$

E. Mutation operator

Impulsive haphazard change in various chromosomes is produced by background mutation operator. Alteration of one or more genes is a simple method to achieve mutation. In genetic algorithms mutation place the following critical role either by (a) during the selection process replacement of genes lost from the population takes place by which they can be tried in a new context or (b) adopting the genes which were not available in the initial population. Crossover operator is used to combine existing genes in order to obtain new chromosomes, whereas mutation operator creates new chromosomes by causing small perturbation in genes. Therefore, it helps to maintain the diversity of the population and to extend the solution space. Because of the coding method used in this paper, mutation operator can be applied easily by alternating some elements x (i, j) of matrix X. every column of matrix X has only one element valued "1" (one job is processed only on one core). Mutation operation is carried out by alternating some elements x (i, j) of matrix X. Therefore here by alternating tasks randomly from one core to another mutation is done. This operation prevents from getting stuck on local suboptimal solutions and it is very helpful to maintain the richness of the population in dealing with large scale problems.

This operator replaces the value of the chosen gene with a uniform random value selected between the user-specified upper and lower bounds for that gene. This mutation operator can only be used for integer and float genes. For lower bound (task size) = 1 and upper bound (task size) = 5, the output produced based on the following equation (6).

$$Y = a + (b-a)\ \text{rand}(2, 1) \quad (6)$$

7. NUMERICAL APPLICATION

In this section, the scheduling problem of jobs on m heterogeneous cores is solved using GA described below.

The computational results are compared to the faster scheduler CFTS-WF. As a numerical example 20 tasks of different size are scheduled on 8 heterogeneous cores. The objective function is to minimize the completion time. The speed of the cores is given as (V_1, V_2, V_3, V_4, V_5, V_6, V_7, V_8) = (1, 2, 4, 5, 7, 9, 10, 12).

The scheduling task with GA is summarized in Table 2.

8. RESULTS AND DISCUSSIONS

We have used simulation to validate the genetic algorithm. The simulation was performed on simulation environment for parallel program execution that was developed using MATLAB programming language. Genetic Algorithm was invoked with the number of populations to be 100 and 800 generations. The crossover rate was 0.5 and the mutation rate was 0.01 randomly the populations were generated and for various trials of the number of cores and task sizes, the completed fitness values of execution time is calculated. Simulation results of genetic algorithm shows that when the task sizes are larger and close to neighborhood sizes the core utilization is higher than that of the smaller tasks. Figure-1 shows the result of scheduling with GA and CFTS-WF algorithm with respect to task size and Figure-2 shows the result of scheduling with genetic algorithm and fast scheduler (Cache Fair Thread Scheduler with Wait Free data structure) CFTS-WF based on number of processing cores. From Figure-2, it can be observed that as the number of core increases, the completion time has got reduced for GA than CFTS-WF. Figure-3 shows the variations in the task size with respect to number of cores for GA, and CFTS-WF scheduler. The execution time and waiting time is observed based on the number of tasks allocated to each cores. Also it shows the variation in execution time for the assigned size of tasks and speed of the core.

Table-2. Scheduling Task with GA

Scheduling with GA (iteration-1)		
Cores	**Scheduled tasks**	**Execution time**
C.1	Task2	15.5
C.2	Task3 Task5	14.8
C.3	Task7 Task6 Task4	16.0
C.4	Task13 Task8	15.0
C.5	Task12 Task10 Task20 Task1	15.4
C.6	Task11 Task17	15.8
C.7	Task15 Task19	16.2
C.8	Task14 Task16 Task18 Task9	16.0
Scheduling with GA (iteration-1I)		
Core	**Scheduled tasks**	**Execution time**
C.1	Task12	15.0
C.2	Task13 Task15	14.9
C.3	Task17 Task16 Task14	16.2
C.4	Task3 Task18	16.1
C.5	Task20 Task1 Task2 Task11	15.8
C.6	Task10 Task7	16.0
C.7	Task5 Task9	16.0
C.8	Task4 Task6 Task8 Task19	15.8

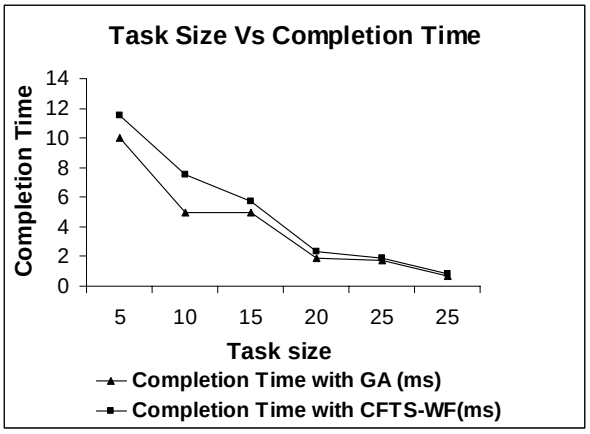

Figure-3. Task size Vs Completion time.

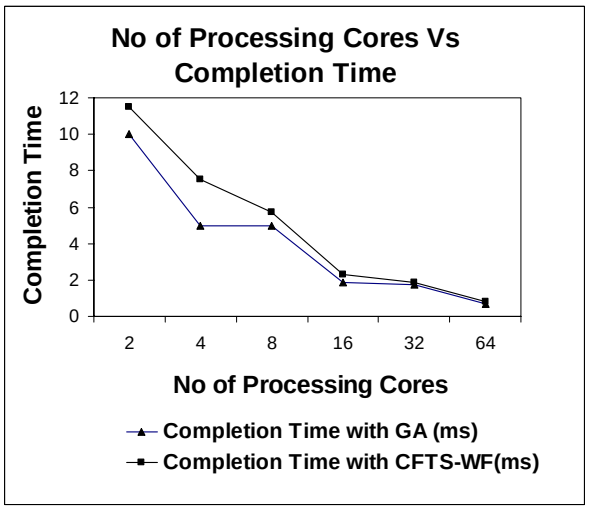

Figure-4. No. of processing cores Vs Completion time.

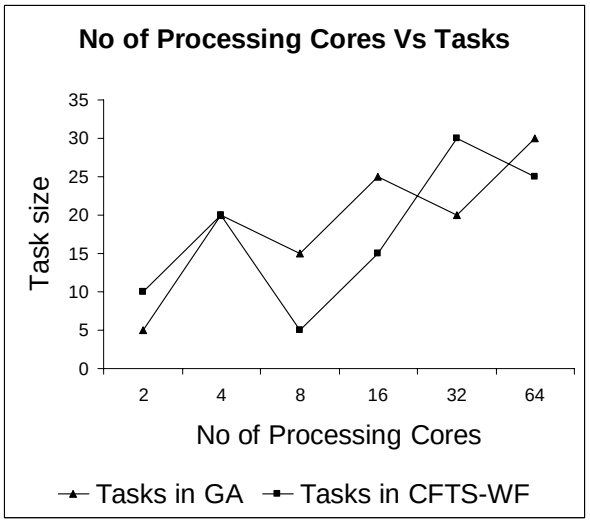

Figure-5. No. of processing cores Vs Tasks.

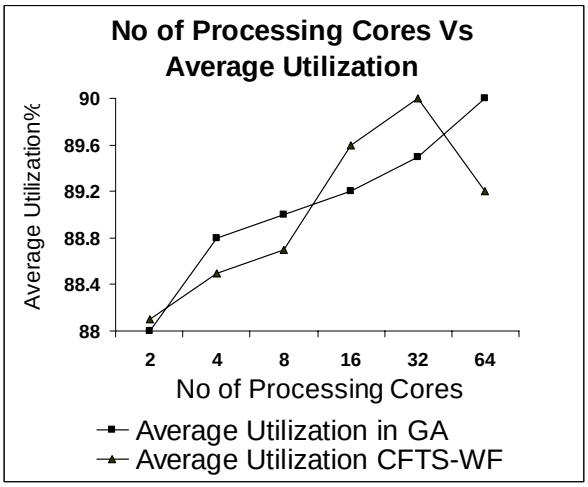

Figure-6. No. of processing cores Vs Average utilization.

9. CONCLUSIONS

Most of the scheduling problems are NP-hard. Mathematical optimization techniques can give an optimal solution for a reasonably sized problem, however, in the case of a large scale problem, their application is limited. Scheduling schemes without GA are suitable only for small scale problems and no scheduling schemes without GA guarantees optimum result in various problems. Research efforts have therefore concentrated on heuristic approaches. Among these approaches, GA outperforms others in view of its characteristic such as high adaptability, near optimization and easy realization. As many practical job shop and open shop scheduling problems can be simplified as heterogeneous multi core scheduling problems under certain conditions, the same has received a great deal of attention in the academic and engineering circle. There are many applications of GA to solve heterogeneous multi core scheduling problem; but, even though it is a common problem in the industry, only a small number of them deal with heterogeneous multi core systems. We therefore decided to concentrate our research effort on scheduling heterogeneous multi core with different parameters using GA so that better results may be produced.

REFERENCES

[1] L. Mitten. 1970. Brach and Bound Method: general formulation and properties. Operational Research. 18: 24-34.

[2] T.L. Adam, K.M. Chandy and J.R. Dicson. 1974. A Comparison of List Schedules of Parallel Processing Systems. Communication of the ACM. 17: 685-690, December.

[3] C.Y. Lee, J.J. Hwang, Y.C. Chow and F.D. Anger. 1998. Multiprocessor Scheduling with Interprocessor Communication Delays. Operations Research Letter. 7(3): 141-147, June.

[4] S. Selvakumar and C.S.R. Murthy. 1994. Scheduling Precedence Constrained Task Graphs with Non-Negligible Intertask Communication onto Multiprocessors. IEEE Trans. On Parallel and Distributed Computing. 5(3): 328-336, March.

[5] T. Yang and A. Gerasoulis. 1993. List Scheduling with and without Communication Delays. Parallel Computing. pp. 1321-1344.

[6] J. Baxter and J.H. Patel. 1989. The LAST Algorithm: A Heuristic- Based Static Task Allocation Algorithm. 1989 International Conference on parallel Processing. 2: 217-222.

[7] G.C. Sih and E.A. Lee. 1990. Scheduling to Account for Interprocessor Communication within Interconnection - Constrained Processor Network. 1990 International Conference on Parallel Processing. 1: 9-17.

[8] M.Y. Wu and D.D. Gajski. 1990. Hypertool: A Programming Aid for Message Passing Systems. IEEE Trans on Parallel and Distributed Computing. 1(3): 330-343, July.

[9] Ali Allahverdi, C.T. Ng, T.C.E. Cheng and Mikhail Y. Kovalyov. 2006. A Survey of Scheduling Problems with setup times or costs. European Journal of Operational Research (Elsevier).

[10] A.S. Radhamani and E. Baburaj. 2011. Implementation of Cache Fair Thread Scheduling for multi core processors using wait free data structures in cloud computing applications. Proceeding of 2011 World Congress on Information and Communication Technologies, © 2011, IEEE. pp. 604-609.

[11] S.M. Alaoui, O. Frieder and T.A. EIGhazawi. 2006. A parallel genetic algorithm for task mapping on parallel machines. In: Proceedings of the 11 IPPS/SPDP'99 Workshops Held in Conjuction with the 13th International Parallel Processing Symposium and 10th Symposium on Parallel and Distributed Processing, Springer, London, UK, 1999. 70 M.H. Shenassa, M. Mahmoodi (Editors). Journal of the Franklin Institute. 343: 361-371.

[12] A. Auyeung, I. Gondra and H.K. Dai. 2003. Integrating random ordering into multi-heuristic list scheduling genetic algorithm. In: Proceedings of the 3rd International Conference on Intelligent Systems Design and Applications, Springer, Berlin. pp. 447-458.

[13] Auyeung I. Gondra and H.K. Dai. 2003. Multi-heuristic list scheduling genetic algorithm for task scheduling. In: Proceedings of the 18th Annual ACM Symposium on Applied Computing, ACM Press. pp. 721-724.

[14] Y.H. Lee and C. Chen. 2003. A modified genetic algorithm for task scheduling in multiprocessor systems. The Ninth Workshop on Compiler Techniques for High-performance Computing.

[15] 2011. Savas _ Balin Yildiz Technical University, Department of Industrial Engineering, Barbaros Bulvari, 34349_ Istanbul, Turkey. Non – identical parallel machine scheduling using genetic algorithm. Journal of Expert Systems with Applications. 38: 6814-6821.

[16] K. Thanushkodi and K. Deeba. 2011. On Performance Comparisons of GA, PSO and proposed Improved PSO for Job Scheduling in Multiprocessor Architecture. IJCSNS International Journal of Computer Science and Network Security. 11(5): 27-34.

[17] Cengiz Kahraman, Orhan Engin, I' hsan Kaya and R. Elif Ozturk. 2010. Multiprocessor task scheduling in multistage hybrid flow – shops: A parallel greedy algorithm approach. Journal of Applied Soft Computing. 10: 1293-1300.

[18] Tomassini M. Parallel and Distributed Evolutionary Algorithms: A Review. In K.Miettinen, M. M"akel"a,p. Neittaanm" aki and J.Periaux (Editors). Evolutionary Algorithms in Engineering and Computer Science.