I J C T A, 7(2) December 2014, pp. 99-108
© International Science Press

Dynamic Task Scheduling in Highly Communicative Task Graph by Critical Path Selection and Partial Task Replication

T. Sajuraj[*] and K. Muneeswaran[**]

ABSTRACT

Scheduling is mapping of task to processor by reducing the job completion time. The aim of this paper is to introduce the scheduling approach for highly communicating task graph with partial replication strategy improving the speedup factor of deploying the parallel processor. Task nodes are mapped on to the processor using critical path, nodes selection and processor assignment algorithm. Some of the critical tasks are duplicated on more than one processor using task work with replication algorithm. Our algorithm provides better job completion time than previous work in the literature.

Keywords: Scheduling, Task Graph, Critical path.

1. INTRODUCTION

The demand for the computing has increased tremendously due to the large number of computers that are networked and the need for processing a enormous number of tasks. Task scheduling is one of the critical issues in multitasks and multiprocessor environment. Parallel programming enables the assignment of task both at spatial and temporal levels. This task scheduling becomes an NP-hard problem [1] which leads to the explorations of many ways of solving the problem with its heuristic solution for its optimality. The huge computational requirements are the challenges in today's mass multimedia-based processing applications. Parallel processing is one of the approaches to meet these requirements. However, designing a parallel algorithm is not that simple as that of the single processing environment. Lot of dependencies among the sub-tasks in the major tasks pose the problem in the realization of the parallel or distributed environments. The main issues are splitting the task into sub-tasks, studying its dependencies, communication across the sub-tasks, synchronization and scheduling to the resources. An inappropriate scheduling exhibits poor performance issues.

The primary objective of the scheduling in multi-processing environment is to minimize the job completion time and improve the turnaround time by appropriately allocating the sub-tasks to the parallel or distributed processors. Majority of the algorithms are based on static scheduling [1] [6], where the requirements regarding the characteristics of the running program must be known before the tasks being executed. However in static scheduling the resources may be underutilized. On the other hand the dynamic scheduling algorithm tries to adapt to the computing requirement with an objective of improving the speed and utilizing the resources to the maximum possible extent.

The work proposed in [3] identifies the dynamic critical path and the result obtained was compared with pre-existing schedulers for deploying multiprocessors having memory constraints. Task replication is sometimes made in the creation of concurrency graph, which requires the identification of the sub-task to

[*] Assistant Professor, C. S. I. Institute of Technology, Thovalai, Nagercoil, India, *E-mail: tsajuraj@gmail.com*

[**] Sr. Professor, Mepco Schlenk Engg. College, Sivakasi, India, *E-mail: kmuni@mepcoeng.ac.in*

be duplicated. The process involves in finding the maximal length of the task (critical path) and assigning the sub-tasks in that path to appropriate resources ensuring the right starting time of the task. In our proposed work, the features of both static and dynamic scheduling are made use of for high interactive/dependent tasks, which assigns the task to processors in a dynamic manner.

2. PROBLEM DEFINITIONS AND BACK-GROUND

The task graph $G = (V, E, W, C)$ represents the task set of tasks T. Here, V is the finite set of vertices representing the sub-tasks in the system. Each element in the set W represents the computational requirements of the sub-task V_i in V. E is the finite set of edges (E_{ij}) linking the sub-task T_i and sub-task T_j, where the sub-task T_j' s input is depending on the output of the sub-task T_i. A non-negative edge cost C_{ij} is the communication cost associated with the delay incurred in the transmission and propagation between the sub-tasks i and j.

P is the set of processors which will be mapped to the sub-tasks in the graph G. The entry point of the graph has the starting time of the task T. We have considered a non-preemptive environment with dedicated environment. The start time of node, 'n' is represented as st(n) and pr(n) is the assigning time of the processor to that node. st(n, p) represents the start time of the node 'n' on the processor 'p', and the completion time of the node 'n' by the processor 'p' is computed as.

$$comp(n, p) = st\,(n, p) + W(n, p) \tag{1}$$

The processing of a node can be done iff the processing element is free and all its previous or predecessor node's processing is completed. To schedule a task graph G onto a target system with a set of processors P, each node $n \in V$ must be associated with a start time and assigned to a processor 'p' where $p \in P$. In order to proceed with our work we assume the following about the system: (i) All the processors in the system are available for execution (ii) The tasks cannot be preempted (iii) The communication cost for executing in the current processor is taken as zero (iv) There is no overhead of communication cost other than transmission and propagation (v) There is no contention for the resources (vi) All the processors are connected.

Also, the processing cannot start unless the execution in the parent node of the current node is not completed in the task graph. The earliest start time can be only after the communication from the previous node to the current node is completed, if the previous node is associated with another processor. The following sections describe our proposed work in details.

3. THE PROPOSED SCHEDULING APPROACH

A proposed scheduling methodology is suggested for task graphs multiprocessor systems. The suggested methodology is intended for highly communicating task graphs.

For every task, the worst case execution time [3] is known a priori in static scheduling. It is greater than the actual execution time. Since the exact completion time of any task can only be known at runtime, static scheduling relies on resource reclaim within each processor. Resource reclaim at the time of completion may be performed on individual processor for the improvement of schedule length. 'Resources reclaim' means that if a task is ready for execution in the scheduled processor before the start time is assigned by the static scheduler, then it can start execution early. The selective replication strategies [7] and [8] tries meeting the memory constraints on each processor by properly choosing nodes that are anticipated to contribute more to the performance improvement during an online scheduling. A dynamic schedulers [1] schedule tasks to processors using the information available at runtime. In order to explore the benefits of the dynamic scheduling, we create different task graphs using Gaussian elimination approach by specifying the number of nodes in the task graph and processors, which is explained in the next section.

3.1. Gaussian Elimination Approach for Task Graph Creation

A task graph is created for Gaussian Elimination application which is a communication-intensive task graph and it is scheduled on set of heterogeneous processors. The simultaneous equation of the form $AX = C$ is solved by the Gaussian Elimination method to create the task graph. It consists of two steps: (i) Forward Elimination and (ii) Back Substitution. In the forward elimination, the coefficient matrix is transformed into an upper triangular matrix. The total number of nodes N in the task graph is calculated using the expression given in Equation (2) by specifying the order of the matrix m.

$$N = (m^2 + 3m)/2 \tag{2}$$

The goal of the static scheduling is to minimize the job completion time and is compliant with the code memory constraint of each processor. In order to implement the scheduling, we propose the following three algorithms as: (i) Critical path selection algorithm, (ii) Node selection algorithm, (iii) Processor Assign algorithm. First we create the task graph using Gaussian Elimination approach for any arbitrary size.

Algorithm genGraphUsingGaussian(M)

Input: *M - order of task graph G*

Output: *Graph G with node cost and edge cost with N number of nodes*

```
{
    N=(m*m+3*m)/2
    for each node n in N of graph G
    {
        c_n = genRandomCost( a,b) //Computation cost of node n in the range a,b
        S_n = false //S_n is the scheduled status of the node n
    }
    G = establishLink(N)
    //produces a Graph with a set of edges E using Gaussian Elimination method
    for each edge e in E with nodes u, v of G
    {
        C_u,v = genRandomCost( p,q) //Communication cost between u, v in the range p,q
    }
}
```

The following algorithms describes the computation of the cost associated with the critical path.

Algorithm computeCost(p)

Input: path p

Output*: path cost c*

```
{
    c = 0
    for each edge (u,v) in path p
    {
        //c_u is the computation cost of node
        //c_u,v is the communication cost of the edge u,v
        c = c + c_u + c_u,v
```

```
    }
    c = c + c_v
    return c
}
```

Algorithm getCritcalCost(G)

Input: Task Graph G with start node s and end node e

Output: Critical Cost C_C

```
{
    Initialize c = c_c = 0
    P = findAllPath (G, s, e)
    for each path p in P
        c_p = computeCost(p)
    C_c = max(c_p)
}
```

The critical path plays an important role in scheduling, and it determines the partial length of a task graph. Along with the critical path, the nodes of task graph can be added dynamically based on its predecessor relation. A node can be selected iff the predecessor node(s) or previous node(s) is completed. During each scheduling, the critical path may vary which results in the change of intermediate nodes dynamically between start and end nodes. The following algorithms *selectNode* and *assignProcessor* describe the selection of the node and assignment of the processor for the job.

Algorithm selectNode (p)

Input: *Critical path p*

Output : Ordered list of nodes - L^O

```
{
    repeat
    {
        for each node n in critical path p
          If (isAvailable(pred(p))
              add(L^O,pred(n))
          else add(L^O, n)
        }
    until all nodes are visited
}
```

After identifying the order of the node, a method is needed to select a processor for scheduling. The scheduled nodes do not have a fixed start times. In order to select a node, its previous node(s) or predecessor node(s) has to be completed. The completed sub task is placed in a list L^{WO}. If the parent node and the child node are executed by the same processor, then the communication cost is considered negligible, otherwise the communication cost is considered. The following algorithm describes processor assignment.

Algorithm assignProcessor(nop, Orderlist)

Input : *nop - number of processors, L^O in G with start and end node*

Output : *Task completed*

```
{
        tc= tps= 0;
        //tc - computational time, tps - processor start time
        for each node n in the L^O
        {
                if (isCompleted( pred(n))
                {
                  if (!isBusy(pr)
                  {
                            assignProcessor(pr,n) //pr - free processor
                            if(assign(pr,n) = = assign(pr, pred(n))
                                    c_u,v = 0
                  }
                  else
                            wait
        }
        else
                  wait //Until the predecessor job nodes are completed
        }
}
```

Till this point, we have discussed the scheduling of the free processor for the task in the graph by selecting the critical path and scheduling the processor to each node for the ordering for execution by means of looking at the completion status of the predecessor node. By this process, some processor may be waiting for the predecessor sub task's completion. After completion of those subtasks only, the other tasks can run. We will show how some of the tasks can be duplicated and idle time of the processors could be explored.

3.2. Task Replication

If we select some of the tasks which are required for the maintaining the sequence of the tasks (dependency) for executing the current sub-sequence of the tasks and duplicate in another path with an ordered list of nodes, the parallel processing power available in the computing system could be explored to improve the performance. By this, the idle time of the processor can be utilized in an efficient manner by executing the duplicated task. Hence concurrency graph is generated and selective nodes are duplicated in the concurrency graph. During the task replication phase, it identifies the concurrent nodes which are scheduled on different processors. A concurrency graph G_C *is* constructed from Gaussian Elimination graph with vertex set *V(G)* so that for each edge u,v in $E(G_C)$, the corresponding two vertices *u* and *v* are not reachable from each other in *G*, and they are not mutually exclusive to each other. The following algorithm narrates the process.

```
Algorithm duplicateTaskAndProcess(nop, L^O,G_C)
Input : nop, Ordered list and concurrent Task Graph G_C
Output : Task completed
{
        t_c=t_ps=0;
        for each node n in the L^O
        {
                if ( ! isCompleted( pred(n))
                {
                        If (checkDuplicate(G_C))
                                duplicateAndAssignProcessor(pr_1, n, pred(n))
                                //pr_1 is the free processor
                }
                elseif (!isBusy(pr)
                {
                        assignProcessor(pr,n) //pr - free processor
                        if(assign(pr,n) = = assign(pr, pred(n))
                                c_u,v = 0
                }
                else
                        wait
        }
}
```

4. EXPERIMENTAL SETUP

Having identified the ways to make use of the free processor for the performance improvement, the next step is to set up simulation for evaluating the work. Figure 1 shows the task graphs with different levels of Gaussian Tree such as 2, 3, 4 and 5 levels. The dependencies of the tasks on the completions are shown as directed arcs.

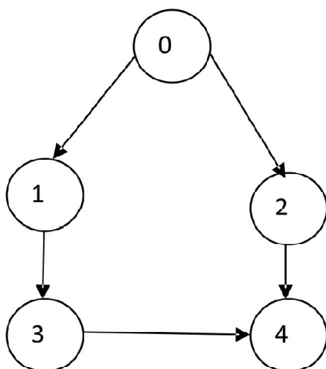

Fig. (a)

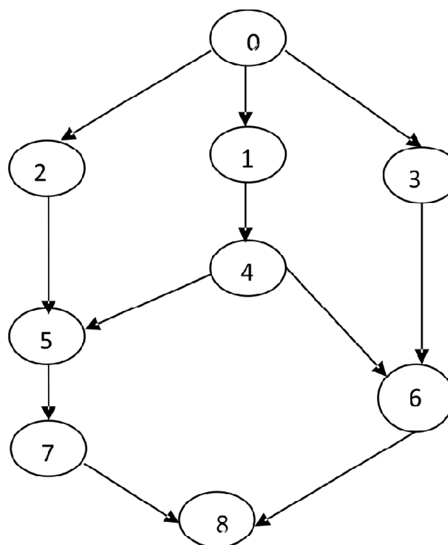

Fig. (b)

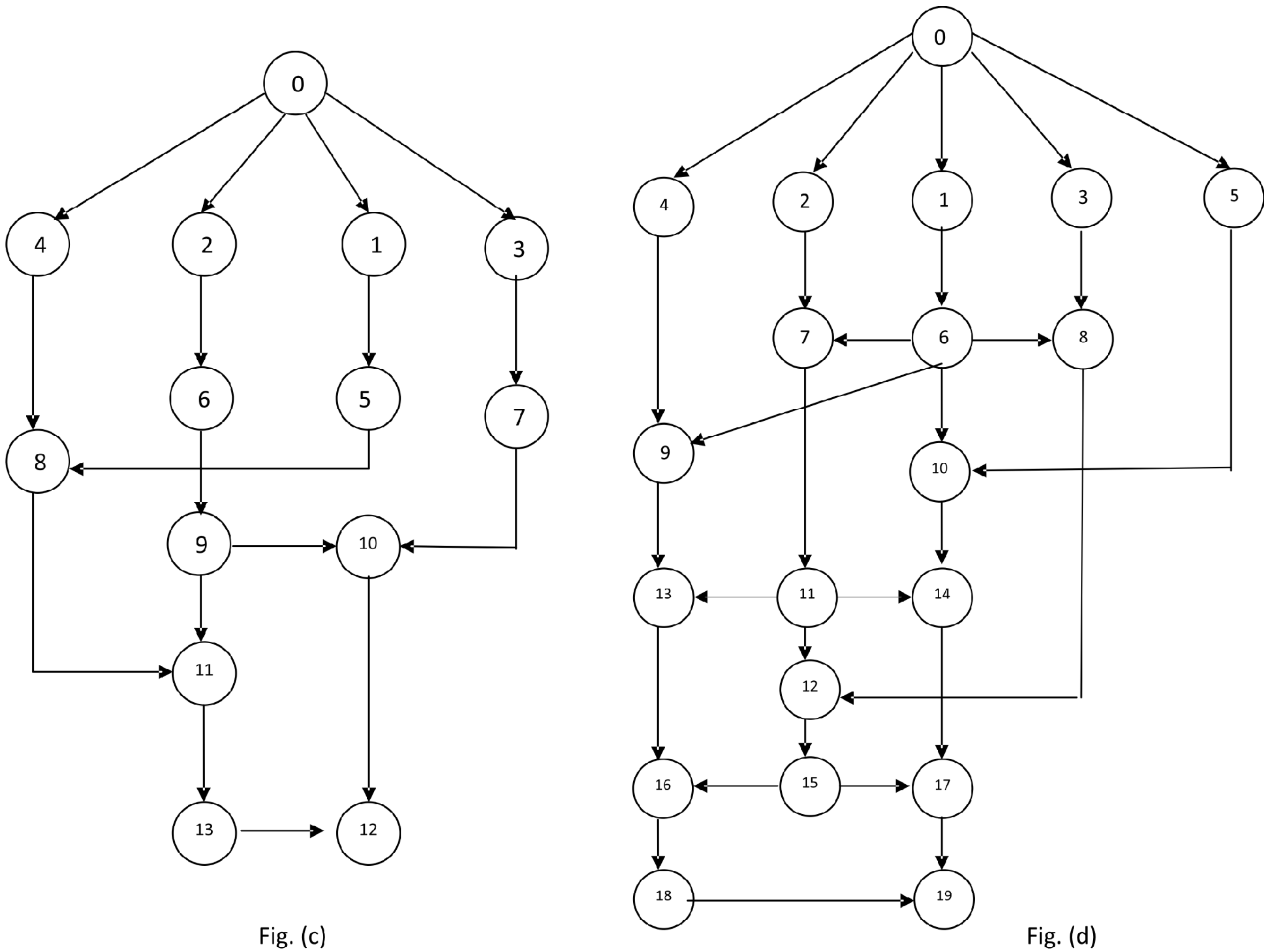

Figure 1: Task graphs with different levels of Gaussian Tree
(a) 2 levels, (b) 3 levels, (c) 4 levels, (d) 5 levels

5. RESULTS AND DISCUSSION

As the tasks are assigned to the processor, the time taken for the completion of the tasks is noted and tabulated. By exploring the parallel processing capabilities of the systems, the performance of deploying the single processor and multiprocessor environments are analyzed. One of the performance metric is speedup factor (S) which is defined as the ratio between the job completion time under single processor environment to the time under the multiple processors environment which is shown in Equation 3 where t_u is the time taken under the single processor environment and t_m is the time taken under multiprocessor environment. Also the efficiency of the deploying the multiple processors can be computed as shown in Equation 4, where the N is the number of processors. Table 1-7 show the performance of the proposed system at different experimental setup.

$$S = \frac{t_u}{t_m} \quad (3)$$

$$\eta = \frac{S}{N} * 100 \quad (4)$$

Table 1
Performance of the Multiprocessor Scheduling (Varying Levels of Task Graph and Number of Processors

#Levels	#Nodes	Time taken in milli seconds														
		#P=1	#P=2		#P=3			#P=4				#P=5				
		P0	P0	P1	P0	P1	P2	P0	P1	P2	P3	P0	P1	P2	P3	P4
2	5	20	11	16	8	12	13	8	9	10	12	7	8	9	9	10
3	9	55	33	36	27	22	28	17	25	24	17	17	16	23	18	17
4	14	87	51	52	36	40	37	33	31	27	29	27	26	26	26	23
5	20	99	76	58	44	52	48	43	41	43	33	35	27	33	33	35

Table 2
Outputs without Replication

#Levels	#Nodes	Time taken in milli seconds														
		#P=1	#P=2		#P=3			#P=4				#P=5				
		P0	P0	P1	P0	P1	P2	P0	P1	P2	P3	P0	P1	P2	P3	P4
2	5	26	18	24	12	18	24	10	11	18	24	7	10	11	18	24
3	9	62	50	55	42	40	35	42	30	29	38	30	23	42	27	22
4	14	73	64	55	45	45	41	25	43	27	36	42	25	32	23	27
5	20	99	70	66	57	62	60	39	42	36	47	35	30	27	31	25

Table 3
Outputs with Replication

#Levels	#Nodes	Time taken in milli seconds														
		#P=1	#P=2		#P=3			#P=4				#P=5				
		P0	P0	P1	P0	P1	P2	P0	P1	P2	P3	P0	P1	P2	P3	P4
2	5	26	16	22	11	16	22	22	16	10	9	7	9	10	16	22
3	9	62	46	49	36	36	39	27	27	38	34	19	32	20	23	29
4	14	73	48	58	41	41	43	29	28	30	41	22	22	19	35	25
5	20	99	64	59	50	52	51	31	35	40	42	31	29	25	35	23

Table 4
Critical Cost without Replication

#Levels	#Nodes	Time taken in milli seconds				
		#P=1	#P=2	#P=3	#P=4	#P=5
2	5	26	24	24	24	24
3	9	62	55	42	42	42
4	14	73	64	45	43	42
5	20	99	70	62	47	35

Table 5
Critical Cost with Replication

#Levels	#Nodes	Time taken in milli seconds				
		#P=1	#P=2	#P=3	#P=4	#P=5
2	5	26	22	22	22	22
3	9	62	49	39	38	32
4	14	73	58	43	41	35
5	20	99	64	52	42	35

Table 6
Speed up without Replication

#Levels	#Nodes	Time taken in milli seconds				
		#P=1	#P=2	#P=3	#P=4	#P=5
		S	S	S	S	S
2	5	1	1.08	1.08	1.08	1.08
3	9	1	1.12	1.47	1.47	1.47
4	14	1	1.14	1.62	1.69	1.73
5	20	1	1.41	1.59	2.10	2.82

Table 7
Speed up with Replication

#Levels	#Nodes	Time taken in milli seconds				
		#P=1	#P=2	#P=3	#P=4	#P=5
		S	S	S	S	S
2	5	1	1.18	1.18	1.18	1.18
3	9	1	1.26	1.58	1.63	1.93
4	14	1	1.25	1.69	1.78	2.08
5	20	1	1.54	1.90	2.35	2.82

Gaussian Elimination Task Graphs of various sizes are taken for experiment. The proposed algorithm for highly communicating task graph is compared and studied for various levels, and the number of processors and the time taken by them are also studied. Task graphs are created by varying the number of nodes and dimension of Gaussian elimination matrix. Computation cost and communication cost is selected randomly by using function. For the experiments, the matrix sizes are varied. This scheduling gives better performance compared to the static and dynamic scheduling. It is observed that the increase in the number of processors reduces the job completion time. Also the speedup faction with selected tasks duplicated improves as the idle processor time is explored.

6. CONCLUSION

The proposed algorithm is better for large number of processors. The performance of our proposed algorithm has been observed experimentally by using Gaussian elimination task graph. The proposed scheduling phase is better for large number of processors. In our proposed work, the generalized Gaussian elimination task graph is generated and the generated task graph is statically scheduled in the processor using the Dynamic Critical Path scheduling algorithm. Then, the generalized concurrency graph is generated for is exploring the parallel processing computing power and is shown to provide the promising results for the various test case scenarios compared to the existing work.

REFERENCES

[1] A. Feldmann, M.-Y. Kao, J. Sgall, and S.-H. Teng, "Optimal On-line Scheduling of Parallel Jobs with Dependencies," Proc. 25th Ann. ACM Symp. Theory of Computing (STOC '93), pp. 642-651, 1993.

[2] R. Gupta, D. Mosse, and R. Suchoza, "Real-Time Scheduling Using Compact Task Graphs," Proc. 16th Int'l Conf. Distributed Computing Systems (ICDCS '96), p. 55, 1996.

[3] Y. K. Kwok and I. Ahmad, "Dynamic Critical-Path Scheduling: An Effective Technique for Allocating Task Graphs to Multiprocessors," *IEEE Trans. Parallel and Distributed Systems*, Vol. 7, No. 5, pp. 506-521, 1996.

[4] R. Ernst and W. Ye, “Embedded Program Timing Analysis Based on Path Clustering and Architecture Classification,” Proc. IEEE/ ACM Int’l Conf. Computer-Aided Design, pp. 598-604, 1997.

[5] I. Ahmad and Y.-K. Kwok, “On Exploiting Task Replication in Parallel Program Scheduling,” *IEEE Trans. Parallel and Distributed Systems*, Vol. 9, No. 9, pp. 872-892, 1998.

[6] Y. K. Kwok and I. Ahmad, “Static Scheduling Algorithms for Allocating Directed Task Graphs to Multiprocessors,” *ACM Computing Surveys*, Vol. 31, No. 4, pp. 406-471, 1999.

[7] C.I. Park and T.Y. Choe, “An Optimal Scheduling Algorithm Based on Task Replication,” *IEEE Trans. Computers*, Vol. 51, No. 4, pp. 444-448, 2002.

[8] M.R. Guthaus, J.S. Ringenberg, D. Ernst, T.M. Austin, T. Mudge, and R.B. Brown, “MiBench: A Free Commercially Representative Embedded Benchmark Suite,” Proc. Fourth Ann. IEEE Workshop Workload Characterization (WWC), Dec. 2001.

[9] N. Fisher, J. Anderson, and S. Baruah, “Task Partitioning upon Memory-Constrained Multiprocessors,” Proc. 11th IEEE Int’l Conf. Embedded and Real-Time Computing Systems and Applications (RTCSA ’05), pp. 416-421, 2005.

[10] Pravanjan Choudhury, Rajeev Kumar and P.P. Chakrabarti, “Hybrid Scheduling of Dynamic Task Graphs with Selective Replication for Multi-processors under Memory and Time Constraints”, *IEEE Trans. Parallel and Distributed Systems*, Vol. 19, No. 7, July 2008.