Utilization of Map-Reduce for Parallelization of Resource Scheduling using MPI: PRS

Likewin Thomas and Annappa B
Dept of Computer Science and Engineering, Centre for Wireless Sensor Network
National Institute of Technology Karnataka
Surathkal, Mangalore, India
likewinthomas@gmail.com, annappa@ieee.org

ABSTRACT
Scheduling for speculative parallelization is a *problem that remained unsolved despite its importance [2]*. In the previous work *scheduling* was done based on Fixed-Size Chunking (*FSC*) technique which needed several 'dry-runs' before an acceptable finalized chunk size that will be scheduled to each processors is found. There are many other *scheduling methods* which were originally designed for loops with no dependences, but they were primarily focused in the problem of load balancing. In this work we address the problem of scheduling *tasks* with and without dependences for speculative execution. We have found that a complexity between minimizing the number of re-executions and reducing overheads can be found if the size of the *scheduled block* of iterations is calculated at runtime. We introduce here a scheduling method called *Parallelization of Resource scheduling (PRS)* in which we first analyze the processing speed of each worker based on that further division of the actual task will be done. The result shows a 5% to 10% speedup improvement in real applications with dependences with respect to a carefully tuned *PRS* strategy.

Categories and Subject Descriptors
B.2.1 **[Design Style]:** Parallel; B.4.1 **[Data communication Devices]:** Processors, Receivers, Transmitters [Master]; D.1.3 **[Concurrent Programming]:** Distributed Programming, Parallel Programming; D.2.1 **[Requirement/ Specification]:** Tools- MPI; D.4.1 **[Process Management]:** Scheduling, Multiprocessing/ Multiprogramming/ Multitasking; D.4.4 **[Communication Management]:** Input/ Output, Message Sending, Network Communication; E.5 **[FILES]:** Sorting.

General Terms
Algorithm, Experimentation, Management, Performance, Standardization, Verification.

Keywords
Master-Worker, Parallel Computing, Map-Reduce, Scheduler, Fixed Chunk Size (FCS), PRS, Critical Path Analysis (CPA).

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
ICCCS'11, February 12–14, 2011, Rourkela, Odisha, India.
Copyright © 2011 ACM 978-1-4503-0464-1/11/02…$10.00.

1. INTRODUCTION
Many time-based applications require *predictable performance* and tasks in these applications have *deadlines* to be met [3]. In this work, we propose an efficient algorithm for scheduling of dynamically arriving real-time tasks (a-periodic tasks) in multiprocessor systems, a real-time task is characterized by its *deadline, resource requirements, and worst case computation time* on *p* processors [1], where *p* is the degree of parallelization of the task. We use this parallelism in tasks to meet their deadlines and, thus obtain better schedulability compared to non-parallelizable task scheduling algorithms. The comparison study shows that the schedulability of the proposed algorithm is better than that of the parallel algorithm for a wide variety of real time task parameters, hence based on the research work done on the resource requirement, the resource scheduling strategy is studied, which includes

- *Task priority,*
- *Resource assign rule,*
- *Sharing resource of parallel task and*
- *Cross maintenance.*

"An Efficient Dynamic Scheduling Algorithm for Multiprocessor Real-Time Systems [1]" is work which shows that each processor has its own dispatch queue. This organization, shown in Figure. 1, ensures that the processors will always find some tasks in the dispatch queues when they finish the execution of their current tasks hence ensuring proper schedulability is always a threat for scheduling process.

"Just-In-Time Scheduling for Loop-based Speculative Parallelization[2]" is work which shows that trading-off between minimizing the number of re-executions and reducing overheads can be found if the size of the scheduled block of iterations is calculated at runtime, hence much of the execution time will be wasted during runtime for trading off.

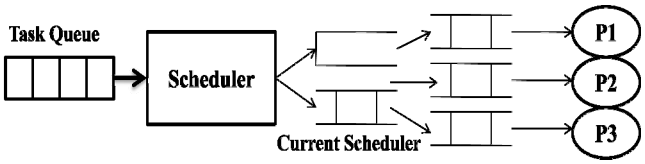

Figure 1 Dispatch Queues

1.1 PRS Algorithms is Centralized.
In a *centralized scheme*, all the tasks arrive at a central processor called the *scheduler (Master)*, which determines the scheduling credibility of each slaves registered with the master by performing the pre-scheduling task:

Map: Here it (master) decides which processor is capable of satisfying the credibility of the arrived task:

Reduce: Since the performance result of centralized scheme was founded to be more practically applicable when compared with the distributed scheme even after it having its own limitation this work is been framed on the skeleton of *centralized scheduling scheme [4]*.

2. MASTER SLAVE MODEL OF PARALLELIZATION: *Map-Reduce*

The parallelization paradigm that is used for optimization of resource scheduling is the *Master-Slave model*. This model is aimed at distributing the (objective function which in our term is called as task) evaluation of the individuals on several *slave* computing resources while a *master* resource executes the *optimization procedure* [5] by assigning the task to calculate the optimization of each processor. The *master-slave* model is shown in the. Figure 2

Figure 2. Master-Slave Model

It is very natural to use parallel computers to divide the task assigned and to solve expensive functions parallel. However, the key issue is that in most of cases we deal with heterogeneous resources but here, the *central computing resource i.e., the master,* based on the obtained solutions and gather information by performance of the assigned task to the registered resources *optimization steps (such as ranking* etc.) will be carried on. However, waiting for the slowest processor might take a long time. The main questions when solving problems on heterogeneous computing resources are

(a) How to efficiently search the *resources*? And
(b) How to use all of the resources so that none of them *stops working*?

This work provides the solution for the above said problems in heterogeneous systems; by providing proper architecture of master/slave model, where the master controls the entire operation of assigning the dummy/ common task in Map phase to find the computation capability of each resources and once that is obtained the master then decides/ come to know the processing speed of the available resources hence it then divides the work accordingly based on their processing speed and send it to the workers available in the cluster hence by doing so we are able to identify the slow running worker in the cluster and then assign less part of actual task to him.

3. PRS with Map-Reduce:

Analysis and Workloads Classification

In this section, we analyze the *Map-Reduce* working procedure, and give a classification of workloads on *Map-Reduce*.

3.1 PRS procedure analysis

Map tasks are the collection of independent tasks which use common input that are assigned to different nodes in a cluster in order to calculate the processing speed of that node. In the other hand, *Reduce tasks* depend on the output of *map* tasks. Here in the first phase the master finds the computation capability of the available resources in the cluster by the help of computation time taken by them, once that is found in map phase then the actual work is divided based on the computation capability of each resources and then the actual divided task is assigned in the Reduce phase for performing the desired operation; hence by doing so we are able to overcome the disadvantage obtained in FCS Method, the following figure clearly specifies the above description.

Figure 3. A *PRS* procedure analysis *Map-Reduce*

3.2 PRS: Schedule policies

Figure 4. Classification of workloads on *PRS*: Schedule policies

The tasks scheduler: master contains a CPU-bound queue where the tasks whose computation has to performed will be present in the queue for the completion of the *map*, once the *map* result are obtained the *master* will be able to decide which resource is better performing, based on that division of *chuck size* will be decided and proper scheduling will be done. *Reduce task* is then is performed by dividing the task into *chunk* based on the processing capability of the each resources in the cluster, hence if there is a resources which is either heavily loaded or slow running machine in the cluster and getting the result in most appropriate time.

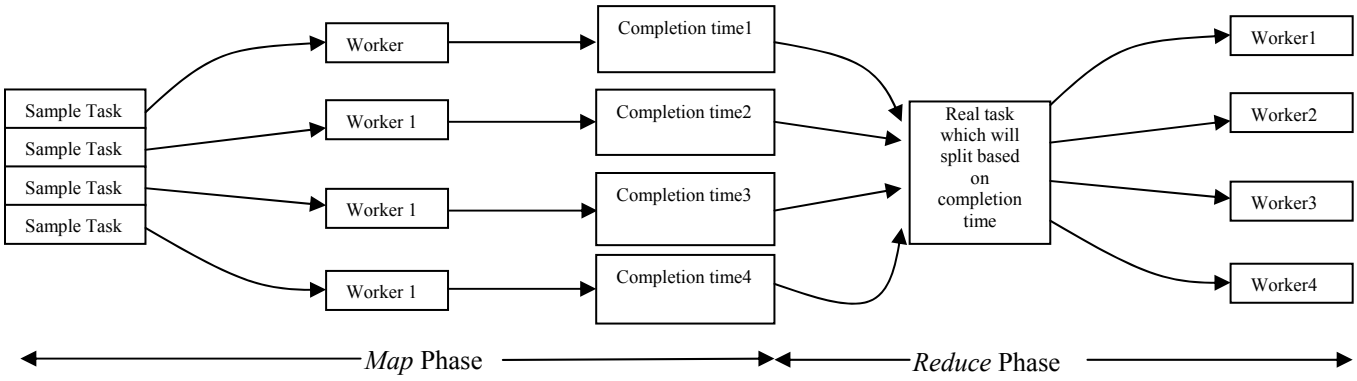

Figure 5. *Map-Reduce* in PRS

3.3 Map-Reduce in PRS

From the above diagram we are able describe the utilization of *map-Reduce* in *PRS*, in the initial phase:

In this phase decision will be taken which processor is better performer and who has to be given larger chunk size, here there is availability of same sample tasks which will be assigned to all the workers who are registered with the master in the cluster. Now the workers receive this sample task, then the earliest start time and earliest finish time will be recorded by the master.

Total time = Earliest Finish time - Earliest start time

Reduce Phase in *PRS:*

This is the phase where the actual computation begins now when the master has collected the time taken by each worker, and it has sorted the workers in the order of the priority of the performance, master will divide the actual run-time task on which the computation has to be done according to the order of the performance of the worker.

4. PARALLEL SOFTWARE FOR RESOURCE SCHEDULING

Algorithm

Step 1: Map Phase/Task: Send the dummy initial *task* to the entire worker/resources that are registered with the master and then wait for the workers to complete the *task* assigned, when the *task* gets completed *master* receives/collects the result by each worker along with the *finish time*.

MPI_Send(&sum,1,MPI_INT,k,1,MPI_COMM_WORLD);

Where:

Sum is the a task that is been assigned to the workers
1 is the size of the task
MPI_INT is the data type
K is an integer that is going to be initilized to the rank of the workers
MPI_COMM-WORLD is communicator

This is been received by each worker, when the operation is operated the time is set which is called as *earliest start time* and at the end of the operation the *earliest finish time* is set. Now then when we have the earliest start time and earliest finish time we can find the total time required for the operation.

Step 2: Then call sorting algorithm to sort the time in ascending order to find which worker has taken less time which is obtained in *map step* based on the computational time taken by the resources.

```
Do while(time[i] less then mid )// here the time [i] is the coparitive time of worker 1 and is compared to mid value time if yes then do following

Begin while::
        i++; // here I is increamented and is pointing to 2 time from right
        while(time[j] greater then mid)// here the time[j] is the time of worker 2 and is compared to the mid value weaather it is greater then that or not and if greater then do following

        Begin while::
                j--; // here j is decremented and is pointing to second from left of the array

                if(i less then & equal to j) // from above incrementation and decrementation we will reach a point where i will meet j then if i is less then j then swap as follow;

                Begin if::
                        y = time[i];
                        time[i]=time[j];
                        time[j]=y;
                        i++;
                        j--;                    end if
        End While
        while(i<=j)// if i is still less then or equal to j then call function recursively hence we get sorted list at the end hence

        Begin while::
                if(low<j)   sort(time,low,j);
                End if

        if(i<high  sort(time,i,high); End if
        return 0;End While
```

Where *i* pointing to the first element of the unsorted array of time and *j* is pointing to the last element of the array of time and then the comparison begin as explained in the above algorithm.

Step 3: Now based on the output of the previous sorting method the fastest processor will be identified, where the worker whose time has come first in the sorted list is considered to taken very less time hence we get the worker whose processing speed is high in the cluster and from the sorted list we are able to identify the processing speed of all of rest of workers in the cluster and once we have the worker ordered in the order of their processing speed then it is quite easy to divide the work which is actually going to be allotted hence the work will be divided based on their processing speed and following step clearly shows how the mathematical calculation is performed to get the percentage of amount to be divided to the workers.

```
        for(i=1;i<mpiSize;i++) // this implies for all the workers in the cluster

Begin for::
                percent=rem_per/size;// initially rem_per is initialized to 100 hence by dividing it with size (number of workers in the cluster we get the fixed chunk size)

                percent1[i]=percent+(percent/2);//hence percent of 1st worker in the list of processing speed sorted worker will be found where percent is already found in above calculation

                rem_per=rem_per-percent1[i];// Now rem_per is re-initialized by subtracting it with already alloted percentage to 1st worker
                size--;// size is decremented as the 1 worker is alloted the percentage
        following lines are used for converting the obtained percentage into a whole number which is clearly explained with an example below.

                percent2[i]=(float)percent1[i]/10;
                percentfinal3[i]=(int)percent2[i];
                percentfinal1[i]=percent2[i]-
        (int)percentfinal3[i];

                if(percentfinal1[i]>=0.5)
                 Begin if
percentfinal3[i]= percentfinal3[i]+1;
        End if
End for
```

Where rem_per is initialized to 100. and the fastest processor is alloted *percent+(percent/2)* where *percent* is *rem_per/size* and size is size of the cluster*;*

Example:

If the percent1[i] is 33 then the percent2[i] will be 3.3 that will be stored into an array of percent2[i]. Then percentfinal3[i] will be calculated by converting the obtained percent2[i] into integer and then percentfinal1[i] is an array which will obtain the decimal part of the float that is been obtained. Hence percentfinal1[i] will contain 0.3 of 3.3 which is the percent2[i].

This then will be compared with the 0.5 if it is less than 0.5 then the percentfinal3[i] which is 3 will remain as it is else if it is greater than 0.5 then percentfinal3[i] will be incremented/ added by 1. Hence we get the whole number based on which the array will be divided. Now the master will send this size of chunk to the respective worker who is stored in worker array based on the performance which is been calculated in the map-phase. Along with the chunk size master will have to send the starting point of the array to the worker so that the worker will be able to find from where he will have to start his portion of work of computation.

Hence for that:

```
last_per[i]=last_per[i-1]+percentfinal3[i-1];
```

where last_per[1] is initialized to 0 i.e., the starting point of the first worker will be 0 then the last_per[i] will be calculated for the workers who priority stands from 2 to last worker, hence the starting point of rest of the worker is found by adding the previous chunk size to the previous starting point of the worker.

Step 4: Now sending the allotment to each processor along with their starting point which is calculated

```
for(i=2;i<mpiSize;i++) Begin For::
        last_per[i]=last_per[i-1]+percentfinal3[i];
        End For
```

where last_per[i] is initialized to 0 for first processor and percentfinal3[i] is an array of allotement of each processor. Hence at the end of this step the starting task will be decided for each work.

Step 5: Now then based on percentage and starting point of the each worker, each worker will be allotted the work load.

4.1 Critical path analysis (CPA)

Heart of *PRS* algorithm uses *critical path analysis (CPA)* [12] to calculate a resource's processing capability those are registered with the *master*, which is done by the *pre- mapping* technique. Hence at the end of *CPA* the master will be able to know the processing capability of each processor registered with the master. The workers follow the *CPA* process step-by-step and provide a clear insight into how *resource schedules* are calculated by *Task Assignment* by the master.

Critical path analysis *(CPA)* is a mathematical procedure that calculates a resource's scheduling by taking each task in turn, it firstly calculates how quickly the task can be accomplished its early start and early finish time. The sample *CPA process* which is been adopted in this work is shown in the following diagram which clearly shows early start of a task and early finish.

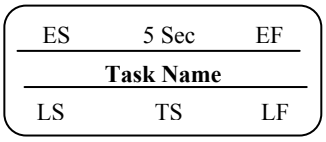

Figure 6. *CPA*

Where:
ES: Early Start,
EF: Early Finish,
LS: Latest Start,
LF: Latest Finish.

Hence from the CPA we are able to find the best path between the master and the worker in the cluster through which the work can assigned.

How CPA is adopted in PRS

Critical Path Analysis is a best performing and suggested tool since 1950 tool for finding the critical path of scheduling; hence it helps in scheduling and managing complex resources.Critical Path Analysis (CPA) or the Critical Path Method (CPM) was adopted in PRS to find all resources that must be analyzed before assigning the actual task begins so that the efficient path of allocation of resources can be found which helps in effective resource allocation. Hence, preparation of a schedule and of resource scheduling planning can be done. During execution of the task by the help of CPA we were able to monitor the working capability of the resources present in the cluster they also helped to overcome the fault by indicating where remedial action needs to be taken to get a task back on course. The benefit of using CPA within the PRS is that it helped us to develop and test our plan to ensure that it is robust. Critical Path Analysis formally identifies resources which must be allotted more percentage of divided tasks so that whole task can be completed on time without leaving any resources idle due to early completion compared to other resources. It also identifies which tasks can be delayed if resource needs to be reallocated to catch up on missed or overrunning tasks. A further benefit of Critical Path Analysis is that it helped us to identify the minimum length of time needed to complete a task hence by that we were able to find out where we need to run an accelerated project, it helps you to identify which task steps you should accelerate to complete the task within the available time.

5. RESULT AND ANALYSIS

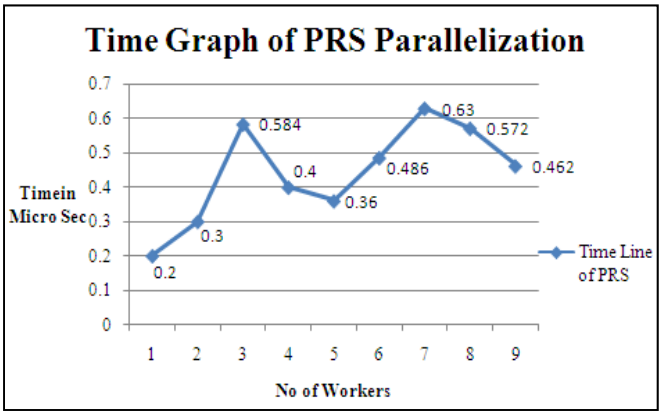

Figure 7. Time Graph of Parallelization of Resource Scheduling

The Figure 7 shows the time taken when the work *PRS* was run on 10 machines starting from 2. The result effectively shows how the time is properly distributed to achieve proper Resource Scheduling and Scheduling of the task. The figure 8 is the result which was obtained when a *FCS* parallelization algorithm was run for the same task which was run for the *PRS*, the result clearly differentiate the time taken when the no of machines were changed.

Figure 9 shows the comparison study for the normal parallelization with *PRS,* from the study we easily notice that the time taken by the proposed algorithm is much linear and better than that of the existing work.

Figure 10 shows the speed up of *5 to 10%* was obtained by *PRS* when run in comparison with that of *FCS*, this is the speed up obtained with the smaller chunk size hence by running it with larger size of data we are able to achieve more rate of speed up from the figure we are able to draw the speed-up generated by providing the proper Resource scheduling for the normal parallelization.

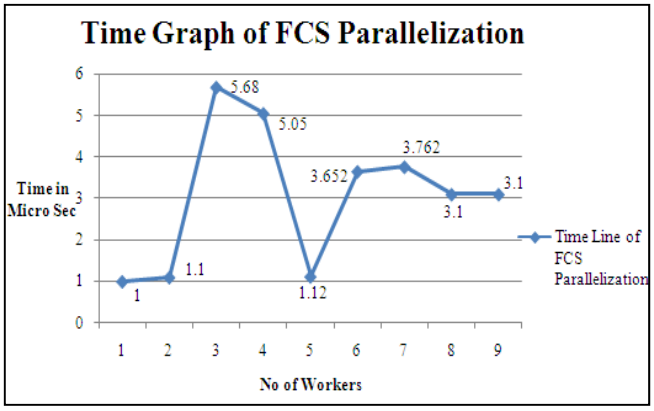

Figure 8. Time Graph of *FCS* Parallelization

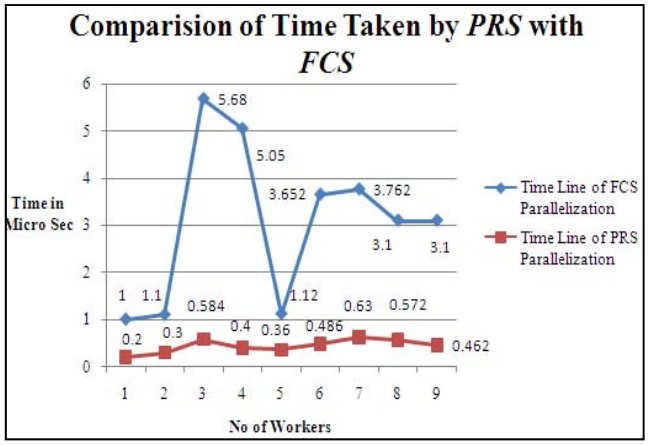

Figure 9. Comparison of *Time Taken* by *PRS* and *FCS*

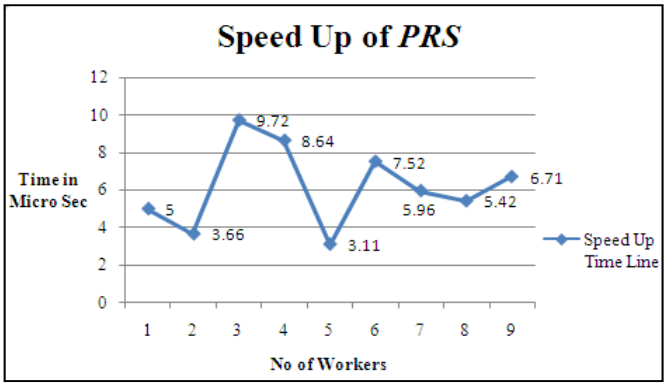

Figure 10. Speed Up of *PRS*

6. CONCLUSION

Meeting *deadlines* and *achieving high resource utilization* are the two main goals of task scheduling in real-time systems. Hence this work has been able to achieve these 2 goals with the high speed-up of 5%-10% which is suggested to be greater turn in the world of parallelization.

Parallelizable task scheduling considered in this work, is a solution which tries to meet the conflicting requirements of high schedulability with low overhead. In this work, we have designed a new algorithm based on parallelizable task model for dynamic scheduling of tasks in real-time multiprocessor systems. Speeding up the parallelization is a challenging task which is been effectively done in this work.

The comparison studies show that the success ratio offered by our algorithm is better than that of the other algorithm for a wide variety of task parameters. In our *future work*, we will try to implement proper fault tolerance and more optimization of Parallel Resource.

7. ACKNOWLEDGMENTS

Completion of a task is never a one man effort. It is often the result of valuable contribution of number of individual in a direct or indirect manner, which helps in shaping and achieving an objective. It is my pleasant duty to thank all those who have been helpful in various ways towards successful completion of this project. The credit of the successful completion of the project should go to the persons who rendered their consistent, constant source of knowledge, timely suggestions and instructions towards me.

First of all I wish to express earnest thanks and affection respect to my honorable *Prof and Head Of the Department Dr. Santhi Thilagan,* Dept of Computer Engineering for providing the required facility and encouraging for the successful completion of the project.

8. REFERENCES

[1] G.Manimaran and C.Siva Ram Murthy, Member, IEEE 1998 An Efficient Dynamic Scheduling Algorithm For Multiprocessor Real-Time Systems *IEEE Transactions on Parallel and Distributed Systems*, VOL. 9, NO. 3, MARCH 1998 PP No 312-319.

[2] Diego R. Llanos, David Orden, Bel´en Palop 2008 Just-In-Time Scheduling for Loop-based Speculative Parallelization *16th Euromicro Conference on Parallel, Distributed and Network-Based Processing* 0-7695-3089-3/08 © 2008 IEEE DOI 10.1109/PDP.2008.13 PP 334-342.

[3] Sanaz Mostaghim, J¨urgen Branke, Andrew Lewis, Hartmut Schmeck 2008 Parallel Multi-objective Optimization using Master-Slave Model on Heterogeneous Resources *IEEE Congress on Evolutionary Computation (CEC 2008)* 978-1-4244-1823-7/08 2008 PP 1981-1987.

[4] Chao Tian12, Haojie Zhou1, Yongqiang He 12, Li Zha 2009 A Dynamic Map-Reduce Scheduler for Heterogeneous *Workloads 2009 Eighth International Conference on Grid and Cooperative Computing* 978-0-7695-3766-5/09 © 2009 IEEE DOI 10.1109/GCC.2009.19 PP 218 -224.

[5] K.Somasundaram, S.Radhakrishnan 2008 Node Allocation In Grid Computing Using Optimal Resource Constraint (ORC) Scheduling *IJCSNS International Journal of Computer Science and Network Security*, VOL.8 No.6, June 2008 Manuscript received June 5, 2008. Manuscript revised June 20, 2008. PP 309-313.

[6] Ren Minglun, Zhu Weidong, Yang Shanlin Institute of computer network Systems 2000 Data Oriented Analysis of Workflow Optimization *Proceedings of the 3'" World Congress on Intelligent Control and Automation* June 28-July 2,2000, Hefei, P.R. China.

[7] Nathan R. Tallent and John M. Mellor-Crummey, Rice University 2009 *Identifying Performance Bottlenecks In Work-Stealing Computations* 0018-9162/09 2009 IEEE Published by the IEEE Computer Society.

[8] Changzheng Qu Liu Zhang Bo Zhang Mingjun Gao Department of Equipment Command and Management, Mechanical Engineering College 2009 Maintenance Resource Scheduling Modeling by Petri net *ICEMI'2009 The Ninth International Conference on Electronic Measurement & Instruments.*

[9] Argy Krikelis Apsex Microsystems Ltd. Brunel University "*Application-centric parallel multimedia software"* IEEE Concurrency.

[10] Christopher H. Nevison 1995 *Parallel computing in the Undergraduate Curriculum* Colgate University 0018-9162/95/ D 1995 IEEE December 1995.

[11] Slo-Li Chu, Tsung-Chuan Huang, Lan-Chi Lee 2001 *Computer Improving Workload Balance and Code Optimization in Processor-in-Memory Systems* 0-7695-1 153-8/010 2001 IEEE.

[12] Andy Jessop Project Learning International *Performing critical path Analysis*.

[13] Jeffrey Dean and Sanjay Ghemawat 2004 *Map-Reduce: Simplified Data Processing on Large Clusters* 0018-9162/95/ D OSDI IEEE 2004.