International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395 -0056
Volume: 03 Issue: 05 | May-2016 www.irjet.net p-ISSN: 2395-0072

Types of Scheduling Algorithms in Parallel Computing

Arun Seth, Vishaldeep Singh

Assistant Professor, Department Of Computer Science & Engineering, Guru Nanak Dev University, Regional Campus Gurdaspur, Punjab, India
Assistant Professor, Department Of Computer Science & Engineering, Guru Nanak Dev University Amritsar, Punjab, India

---

Abstract - *In this paper various scheduling techniques for parallel computing are discussed like Longest Processing Time, List scheduling and approximation techniques like heuristic algorithms are discussed. Then some other scheduling techniques, load balancing and thread scheduling for parallel computing is discussed. Then summary of various scheduling algorithms is presented in the end of this paper.*

Key Words: **Load balancing, Static scheduling, dynamic scheduling, Gang scheduling, Genetic algorithm.**

1. INTRODUCTION

In computing **Scheduling** refers to a set of policies which define the order of execution of processes. From all of the available resources of a computer system that needs scheduling before use, the CPU is one of the most critical of them. Multiprogramming is one of the basic and important scheduling technique. The scheduling in CPU is done to keep it as busy as possible. In parallel computing, multiple processors have to be scheduled, and it needs to manage the resources for all the processors. In managing the resources for multiple processors, it should be ensured that, there should not be any overlapping of the resources, and it should not give any conflicting results. So the scheduling in multiprocessors is more difficult than scheduling in a single processor unit.
In scheduling of multiple processors it should be ensured that any processor should not be overloaded and any processor should not be under loaded. So the overall system should be balanced. In multi-programmed memory systems, when there are multiple ready processes in the main memory, the scheduler must decide the order of execution of processes. But in parallel processing system, as there will be multiple processors, there will be multiple queues, so there is need of scheduling multiple queues simultaneously. The scheduling algorithms for multiple processors should be capable of scheduling all the queues optimally.

1.1 BATCH PROCESSING

The jobs were executed in a group in the order in which they were entered into the system.

1.2 MULTIPROGRAMMING

There are multiple processes in the memory and there was need to implement scheduling algorithms.

1.3 PARALLEL PROCESSING

There is execution of multiple processes simultaneously, so there is increasing demand of more processing speed. In which there were multiple processors and which were executing different instructions simultaneously. In it there is need of scheduling techniques to make them more efficient and get better results.

2. SCHEDULING IN PARALLEL COMPUTING

Symmetric Multi-processing (SMP), Massively Parallel Processing (MPP) units, Cluster computing and Non Uniform Memory Access (NUMA) are the. Symmetric Multi-Processor is a computer architecture in which multiple numbers of processors are connected via bus or crossbar to access the single shared main memory. This architecture can only be scaled up to a dozen processors due to limited amount of available memory bandwidth. Programming model implemented in SMPs is simple because all the processes can access entire system memory. Clusters computers are combination of multiple symmetric multi processor computer nodes to provide more processing power. **MPPs** are similar to clusters because they also support multiple connected nodes. But in MPP the memory bandwidth is designed in such a way that a large number of computer nodes can be added to it. Most powerful supercomputers are all MPP systems. The main limitation of MPP and cluster computing is its message passing. In **NUMA** architecture associated processors have their own local memory. Processors can access non-local memory through an interconnection network, which increases the delay in processing.

3. BASIC PARALLEL SCHDULING

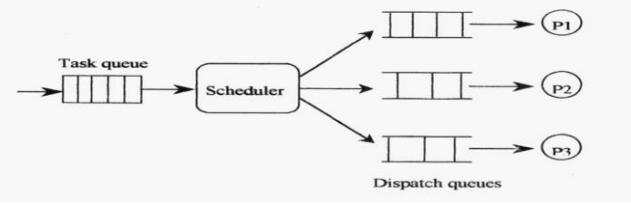

Fig -1: Basic Parallel Processing

4. CONSIDERATIONS IN PARALLEL SCHEDULING

The various parameters those needed to be considered while scheduling in parallel computing are as follows:

- Check which task should be executed by which processor.
- Minimize total turnaround time and total response time for all the processes.
- Balance the workload on all the available processors.
- Order of execution of various queues.
- Finally join the results of all the processors and to give the consistent overall result to the user.
- Maximum and optimal use of all the resources.

To consider all these issues in parallel scheduling, we need the scheduling algorithms explained in following sections.

5. SCHEDULING POLICIES IN PARALLE COMPUTING

Scheduling is difficult in Parallel Computing is difficult as compare to scheduling in serial computers. So in it, we need more sophisticated algorithms for scheduling.

Scheduling in parallel computing is NP-Complete problem.

The algorithms for parallel computing are:

- Longest Processing Time (LPT)
- Look-ahead optimized scheduling (LOS)
- List Scheduling

There are some Heuristic algorithms for parallel scheduling, which are:

- Opportunistic Load Balancing (OLB)
- User Directed Allocation (UDA)

The explanation to these algorithms is:

5.1 LONGEST PROCESSING TIME

In this technique the list of available processes is sorted in decreasing order of size, and largest sized process is allocated to processor which is least loaded.

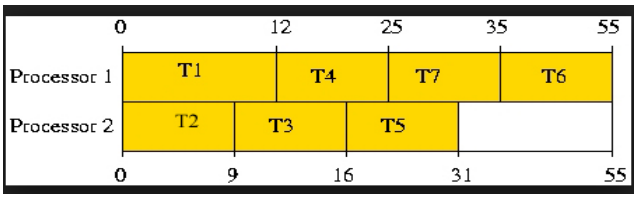

Fig – 2: Example of Longest Processing Time

Here T1 is the longest process and it is assigned to processor 1, because it is the least loaded processor in this case.

5.2 LOOKAHEAD OPTIMIZED SCHEDULING

In this technique each processor has its own ready queue and it looks for the future incoming requests, and it stores only those pages in its memory that may come in memory in near future.

5.3 LIST SCHEDULING

In this scheduling, it makes a list of processes according to their priority and assigned the highest priority process to the least loaded processor.

Then there are also some **Heuristic algorithms** of scheduling in parallel processing, which are:

- **Opportunistic Load Balancing:** In this scheduling, the next task is assigned to the next free processor.
- **User Directed Allocation:** In it user assigns the next task to the processor with lowest expected time to complete

6. TYPES OF MULTIPROCESSOR SCHEDULING

- Thread scheduling
- Load sharing/balancing

6.1 THREAD SCHEDULING

Issues in thread scheduling:

- Multi-threading is useful to decouple different activities
- It must ensure that critical activities must be performed prior to those activities that are non-critical.
- To know whether threads will interfere with other thread.
- Can abstract out separate descriptors for canonical behavioral classes.
- During execution fairness is not guaranteed between different threads.

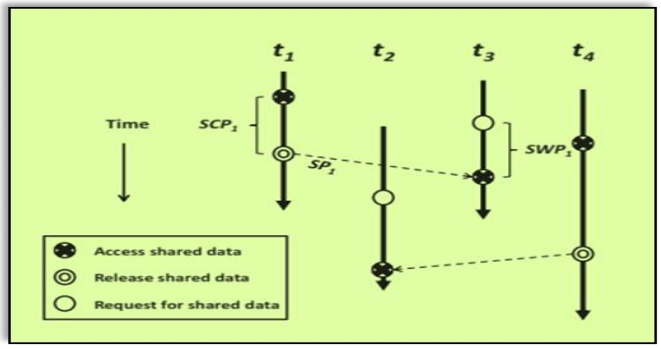

Fig -3: Precedence Relation in Thread Scheduling [2]

Application Model:

An application consists of several threads. Each thread has its own arrival time, its workload, and different precedence

relations with other threads. The precedence relationships are maintained by application designers through spinlocks. When a running thread tries to read the shared data, it must do busy wait until it has exhausted its time-slice or until another thread has released its access on the shared data. To describe the precedence relationships among threads, synchronization-waiting period (SWP) and synchronization-completing period (SCP) are used.
In the figure shown above, One SWP corresponds to one SCP, which belongs to two threads. For example, suppose there is an application which has 4 threads, which are t1 to t4, and their precedence relationships are illustrated. In this example, t1 must do busy wait for SWP units of time, until thread t3 executes for SCP11 units of time. In this work, it is assumed that when an application enters the sys-tem, OS is not aware of any existing precedence relationships among threads.

6.1.1 THREAD SCHEDULING ALGORITHM

For thread scheduling in multi-processing, the algorithm is Synchronization aware dynamic scheduling. Flow chart for above stated algorithm is:

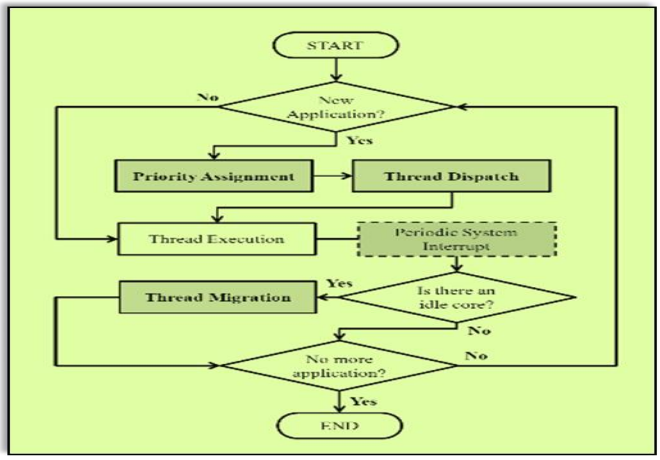

Fig -4: Flowchart of Synchronization aware Thread Scheduling [2]

This scheduling consists of three main procedures: priority assignments, thread dispatch, and thread migration. A periodic system interrupt is needed by scheduling algorithm to collect information of thread behaviors, including the number of SWPs and the number of SCPs. In the following sections, the strategies of three procedures employed in scheduling algorithm are discussed.

(a) Priority assignment: There are two steps in the priority assignment procedure, namely synchronization sampling and priority assignment game.

(i) **Synchronization sampling**: A thread is usually responsible for a specific job, so we have to group the threads based on their type of job.

(ii) **Priority assignment game**: With the given set of threads and their synchronization values, priority to different threads is assigned in a non cooperative way. When a new thread first enters the system, the scheduling algorithm assigns an initial priority to it, after that scheduling algorithm performs synchronization sampling for it.

(b) Thread dispatch: When a new thread enters the system, the operating system scheduler dispatches it to a processor with the shortest ready queue, that is, the core with the smallest number of threads in its ready queue. However, the priorities of each thread could be different, and the difference in time-slice between a high priority thread and a low-priority thread is very large.

(c) Thread migration: When the ready queue of a core becomes empty, we say that the core is idle. The OS scheduler uses a thread migration strategy to achieve better core utilization when it finds a core has become idle.

6.1.1 EXPERIMENTAL RESULTS & REAL WORLD IMPLEMENTATION

For its implementation there is a multi-core thread scheduling simulator using Borland C++ Builder 2009. The thread scheduler is implemented in Linux kernel version 2.6.5. The simulator includes the ready queue structures the scheduler with scheduling algorithm and our synchronization-aware scheduling algorithm (SA)

Frequency	1600 MHz
Supply Voltage	1.5 Volts
Typical Dynamic Power	8.9 Watts
Typical Leakage Power	2.9 Watts

Chart -1: Frequency and Power of UltraSparc T2 Core [2]

6.2 LOAD BALANCING

Let a distributed dataset which has 4 sites and data is evenly distributed on all sites. The different sites will run their piece of code on their own set of data and try to figure out the relationship between the data. It is possible that the data contained at sites 0, 2, and 3 may converge much faster than the data at site 1. If this is the case, the three sites which finished first will remain idle while site 1 finishes. When an attempt is made to balance the workload on different sites, then the parameters that need to be considered are: processing speed of that site and the cost associated with transfer of data from one site to another site.

There are two types of load balancing:

- Static Load balancing
- Dynamic Load balancing

6.2.1 STATIC LOAD BALANCING

The load is assigned to each processor before the execution of the processor. Once a load assigned to a processor would not be changed. In static load balancing, different processing models are: Client-Server model and another is distributed model.

Static Load Balancing is better when:

- Using homogeneous cluster
- Each processing site has an equal amount of work assigned.

Static Load Balancing performs badly, when:

- Non homogeneous clusters are used and speed difference between these clusters is not considered.
- work distribution is uneven

Now static load balancing with genetic algorithm is discussed.

Genetic algorithm for static scheduling: Genetic algorithms are generalized solution for selection problem. In GA, the individuals of a population of potential solutions to a problem having good genetic characteristics have greater survival and reproduction possibilities. Those candidates which would not be suitable for the system would not be appreciated by genetic algorithm. So from the genetic algorithm we can figure out the most promising candidate for scheduling.

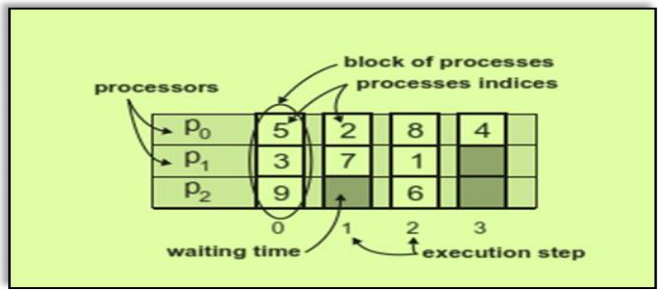

Fig -5: Block of Processors and Processes

In this scheduling algorithm, the genetic operator affects only the processes. They select the place where the process will be executed and at what execution time it will be executed.

Cross-Over: The genetic algorithm applies a proposed crossover operator to generate two new individuals into the new generation. It is a process in which firstly the information of genes of parent is taken, then in the second step the related information is swapped. In the stage of getting the genes, the processors of the parents are divided into two partitions: even processors and odd processors. So from the partition, the first processor would get the information from an even processor. The second new processor would get the information from an odd processor. [1]

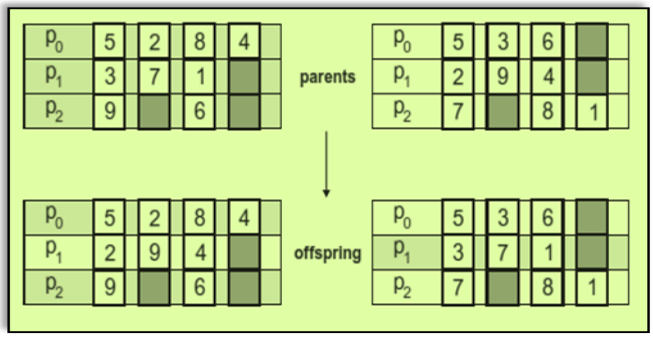

Fig -6: Crossover of Processor and Processes

In swapping of repeated information, duplicate jobs assigned to a processor would be searched, and the duplicate jobs at the same position in two different processors would be considered.

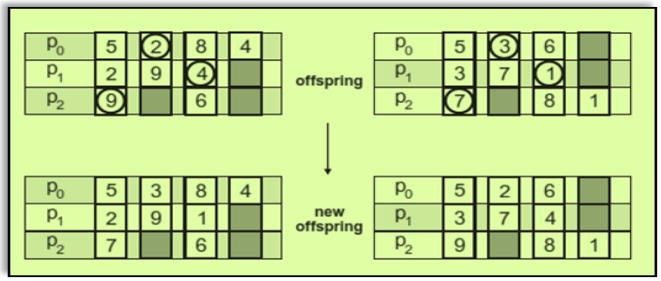

Fig -7: First step Cross-Over

The second step of the cross over is:

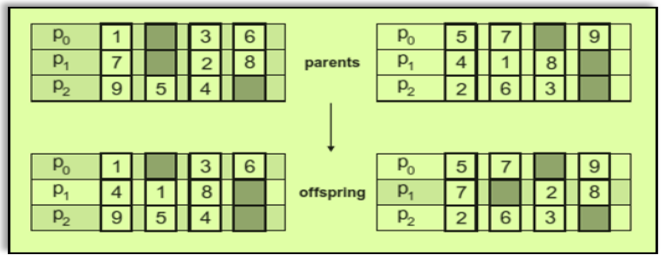

Fig -8: Second step Cross-Over

Mutation: The goal of the uniform mutation is to exchange two neighbor genes without violating precedence relationship in order to create an individual that could not have been produced by the crossover operator.[1]

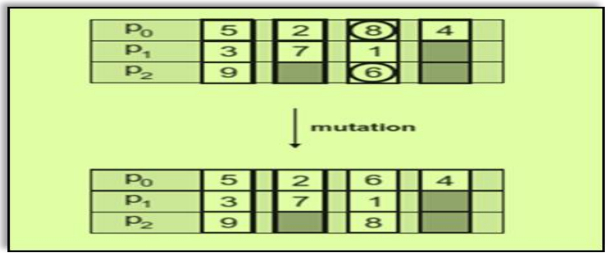

Fig -9: Mutation Process

6.2.2 DYNAMIC LOAD BALANCING

The dynamic Load balancing is of two further types:

- **Task-oriented:** When one processing site finishes its task, it is assigned another task.
- **Data-oriented:** When one processing site finishes its task before other sites, the site with the most work gives the idle site some of its data to process.

Here Dynamic Scheduling with the help of Gang scheduling is discussed.

6.3 GANG SCHEDULING

A local scheduler in Concurrent Gang is composed of two main parts: the Gang scheduler and the local task scheduler. The Gang Scheduler schedules the next thread indicated in the trace diagram at the arrival of a synchronization signal. The local task scheduler is responsible for scheduling local threads that do not need global coordination and it is similar to a **UNIX scheduler**. The Gang Scheduler has precedence over the local task scheduler. [5]

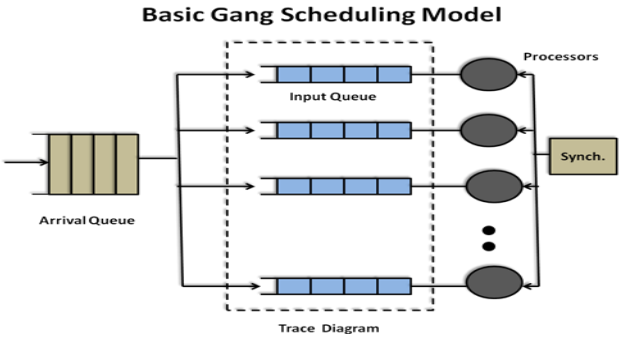

Fig -10: Basic Gang Scheduling Model

An advantage of space sharing is the elimination of multiprogramming, which eliminates the context switching overhead. However, an equally clear disadvantage is the time wasted when a CPU blocks and has nothing at all to do until it becomes ready again. Consequently, people have looked for algorithms that attempt to schedule in both time and space together, especially for processes that create multiple threads, which usually need to communicate with one another.

To see the kind of problem that can occur when the threads of a process (or processes of a job) are independently scheduled, consider a system with threads A0 and A1 belonging to process A and threads B0 and B1 belonging to process B. threads A0 and B0 are timeshared on CPU0; threads A1 and B1 are timeshared on CPU1. Threads A0 and A1 need to communicate often. The communication pattern is that A0 sends A1 a message, with A1 then sending back a reply to A0, followed by another such sequence. Suppose that luck has it that A0 and B 1 start first. [6]

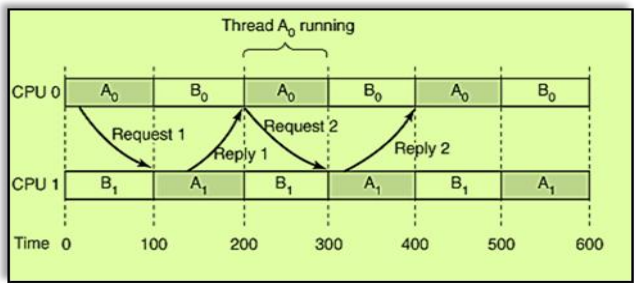

Fig -11: Thread communication in Gang scheduling

In time slice 0, A0 sends A1 a request, but A1 does not get it until it runs in time slice 1 starting at 100 msec. It sends the reply immediately, but A0 does not get the reply until it runs again at 200 msec. The net result is one request-reply sequence every 200 msec. So the given solution is not very good.

The solution to this problem is gang scheduling, which is an outgrowth of co-scheduling Gang scheduling has three parts:

- Groups of related threads are scheduled as a unit, a gang.
- All members of a gang run simultaneously, on different timeshared CPUs.
- All gang members start and end their time slices together.

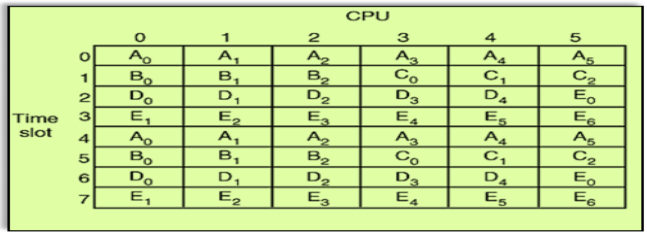

Fig -12: Gang scheduling with six CPUs

7. CONCLUSIONS

In this paper, we have studied some scheduling techniques for parallel computing. The scheduling techniques are threading scheduling, load balancing. In which thread scheduling is basically scheduling the different threads to different processor, which may be a problem, because it may happen that two threads of the same process are allocated to different processor, and if they need to communicate with each other, then this type of scheduling technique is not for these types of problems. And in load balancing, we have studied two types of load balancing, which are static load balancing and dynamic load balancing. Static scheduling is suitable where you have advance knowledge of the workload and all the iteration performed on the chunk of the data. But if the total workload and all the computations are not known, then the dynamic load balancing is more suitable. In it we can assign the load dynamically to that processor, which is least loaded.

REFERENCES

[1] Renata Mederiros de Carvalho, Rucardi Massa F. Lima "An efficient algorithm for static task scheduling in parallel applications" **IEEE 2011**

[2] Chic-Shen Lin, Pao-Ann Hsiung "Synchronization-Aware Dynamic Thread Scheduling for improving Performance and saving energy in Multi-Core systems" **IEEE 2012**

[3] http://www.fer.unizg.hr/_download/ repository/ Grudenickvalifikacijski.pdf

[4] Ravreet Kaur, Dr. Gurvinder Singh "Genetic Algorithm Solution for Scheduling multiprocessor environment" **IEEE 2012**

[5] Fabricio Alves Barbosa da Silva, Isaac D. Scherson "Improvements in Parallel Job Scheduling using Gang Service" **IEEE 2011**

[6] http://lovingod.host.sk/tanenbaum/ MULTIPLE-PROCESSOR-SYSTEMS.html