CHAPTER 3

Review of Scheduling Schemes

In addition to designing an appropriate multiprocessor network, the efficient management of parallelism on an interconnection network involves optimizing conflicting performance indices, like the minimization of communication and scheduling overheads and uniform distribution of load among the processors (nodes). In such a system more than one nodes process the various jobs concurrently. Each job may consist of various tasks that could be executed independently. The number of tasks allocated to each processor has to be controlled in such a way that a high speed execution of processes may occur while maintaining high processor utilization. Due to the unevenly division of tasks (load), some processors may complete execution of their tasks before others and become idle. In a multiprocessor system, if some nodes remain idle while others are extremely busy, system performance will be degraded drastically. Therefore, scheduling of tasks becomes an important problem for multiprocessor system architectures and consequently it has a substantial effect on the system performance and utilization. It is required that all the processors should share the load evenly that would lead to complete the job in minimum possible time.

The scheduling problem is to maintain a balanced execution of all the tasks among the various available nodes in a multiprocessor network. A collection of independent tasks originate and mapped on the root processor.
scheduling policy assumes a set of processors and a set of tasks which are to be serviced by these processors according to a specific policy. This chapter studies the different methods and scheduling algorithms that have been proposed for scheduling the load on specific interconnection network topologies.

Scheduling may be performed at the local level or global level based on the information they use to make load balancing decisions [Zaki et al., 1997]. In a uni-processor system, scheduling is performed by the operating system on the basis of the time-slices of the processor known as local scheduling. Global scheduling, however, decides the processor in a multiprocessor system on which a process is to be executed [Sharma et al., 2008]. In the global schemes, the scheduling decision is made using global knowledge: i.e. all the processors take part in the synchronization and send their performance profiles to the scheduler. Scheduling algorithms can be classified as either static or dynamic. The static algorithm performs by a predetermined policy, whereas, the dynamic algorithm makes its decision at run time according to the status of the system [Zeng and Veeravalli, 2006].

3.1 Static Scheduling

In static scheduling, processes are assigned to processors during the execution. Information regarding the total mix of processes in the system as well as all the independent subtasks involved in a job or task is assumed to be available by the time the program object modules are linked into load modules. Hence, each executable image in a system has a static assignment to a particular processor, and each time that process image is submitted for execution, it is assigned to that processor. In other words, static scheduling requires partitioning the job into a set of independent tasks and then statically allocating them to processors so as to have maximum balance. Static scheduling has been well studied [Grosu and Chronopouls, 2002], [Zeng and Veeravalli, 2004], [Li and Kamede, 1998]. Houle et al. discussed the problems for static load balancing on trees, assuming that the total load is fixed [Houle et
al., 2002]. The goal of static load balancing method is to reduce the overall execution time of a concurrent program while minimizing the communication delays. Some examples of static algorithms presented by [Sharma et al., 2008] are:

- Round Robin and Randomized Algorithms
- Central Manager Algorithm
- Threshold Algorithm

A modified Round Robin scheduling for divisible load named Ordered Round-Robin (ORR) scheduling was proposed by [Yao et al., 2008]. The theoretical derivation and analysis are discussed. They have further proposed and designed the packetized version of the ORR algorithm named Packetized-ORR (P-ORR) to deal with variable length packets.

The static scheduling is generally performed in multi-computer systems where the load is distributed across different computers before execution using a priori known information and the load distribution remains unchanged at runtime [Nehra et al., 2007]. A general drawback that exists in all the static schemes is that the final selection of a host for process allocation is made when the process is created and can not be changed during process execution to make changes in the system load [Sharma et al., 2008]. The main objective of load balancing methods is to speedup the execution of applications on resources whose workload varies at run time in unpredictable way. Static scheduling however, avoids the run-time scheduling overhead. Therefore, in a multiprocessor environment with load changes on the nodes, a more dynamic approach is required. However, the definition between static and a dynamic job allocation algorithm is not very clear and different authors use slightly different definitions of static and dynamic algorithms. Recently, a hybrid scheduling approach has received some attention [Boeres et al., 2003], [Subrata et al., 2008]. A hybrid load balancer attempts to combine the quality of static and
dynamic job allocation algorithm, by minimizing their relative inherent
disadvantages. Static scheduling can be further divided into the following
categories:

- **Optimal versus Sub-optimal**: In static scheduling, it is assumed that all
  information governing the scheduling decisions that can include the
  characteristics of the jobs, the computing nodes, and the communication
  network are known in advance. An optimal scheduling decision is made
deterministically based on some criterion function. On the other hand, if
these problems are computationally not feasible, a sub-optimal or
probabilistic decision may be applied [Darbha and Agrawal, 1998],
[Park and Choe, 2002]. There are two ways to obtain the sub-optimal
solutions namely approximate and heuristic.

- **Approximate versus Heuristic**: In approximate scheduling, same formal
  computational model is used but, instead of searching the entire solution
  space for an optimal solution, one is satisfied when a good one is found.
  A primary intention of heuristic is to find a solution as fast as possible,
  if necessary, at the cost of quality. Heuristics are characterized by their
  essentially deterministic operation [Lee and Zomaya, 2008]. There are
various methods of task allocation on the processors, where the
scheduling decisions may be optimal or sub-optimal shown in Figure
3.1.

3.2 Dynamic Scheduling

In Dynamic Scheduling (DS), the load is distributed among the
processors during execution time in such a way that each processor would have
the same or nearly the same amount of work to do. This
redistribution/allocation of load is performed by transferring the tasks from the
over-loaded processors to the under-loaded processors with the aim of
obtaining the highest possible execution speed. DS schemes are widely
recognized as important techniques for the efficient utilization of resources in a multiprocessor system. The performance of such a system may be increased by increasing the utilization of CPU, memory or a combination of CPU and memory [Qin et al., 2003]. There has been a lot of research for dynamic load balancing (DLB) in traditional parallel and distributed systems literature for more than two decades [Ishfaq and Ghafoor, 1991], [LeMair and Reeves, 1993], [Zaki et al., 1997], [Watts and Taylor, 1998], [Anand et al., 1999], [Ciardo et al., 2001], [Dobber et al., 2005], [Zeng and Veeravalli, 2006], [Dobber et al., 2009]. Yagoubi and Slimani addressed the problem of load balancing in grid computing that works on grid having tree type architecture [Yagoubi and Slimani, 2006]. The important issues in DLB are:

- When to invoke a balancing operation.

- Who makes load balancing decision according to what information, and

- How to manage load migration between processors.

Besides, there are two important parameters when dynamic scheduling algorithms are implemented on parallel systems. The first is parallel systems generally use a regular point-to-point interconnection network, instead of random network configuration. Similarly, the load imbalance occurs mainly, because of the un-even and unpredictable nature of tasks. Dynamic approaches have a major drawback, they are very much sensitive to inaccuracies in performance prediction information that the algorithm uses for job allocation purposes. Due to this high sensitivity, they produce extremely poor results even when the information accuracy is only slightly less than 100 percent. Secondly, in parallel systems, it is very hard to achieve and maintained 100 percent accurate information [Subrata et al., 2008].
3.3 Types of Dynamic Scheduling Schemes

In a multiprocessor system, the effect of a scheduling operation is observed on all the co-operating processors. There are different models of DS which decides the effect and co-ordination across the processors [Banicescu and Velusamy, 2002], [Dobber et al., 2004], [attiya, 2004], [Corbalan et al., 20005], [Beaumont et al., 2008], [Chandra and Shenoy, 2008]. The various models of DS are shown in Figure 3.1. The different models are:

3.3.1 Centralized

Centralized load balancing policies are characterized by the use of a dedicated processor which also takes part in computation. This processor is also known as master processor or central scheduler responsible to make the entire load balancing decisions [Lin and Raghavendra, 1992]. The master processor gathers the global information about the state of the system and assigns tasks to individual node. In this way it can improve the resource utilization by applying sophisticated algorithms. However, for large system consisting of 100 or 1000 of nodes, the master processor becomes a bottleneck. Moreover, if the central processor fails, the whole system stops working.

3.3.2 Fully distributed

It is an alternative to centralized approach, in which the load balancing decisions are carried out by all the processors of the system. Each node executes a scheduling algorithm by exchanging information with other nodes. It is therefore very costly for each node to obtain and maintained the dynamic state information of the whole system [Shivaratri et al., 1992].
Figure 3.1: Scheduling taxonomy
Many variants of fully distributed schemes appear in the literature. However, there are numerous drawbacks associated with them [Ishfaq and Ghafoor, 1991]. The first is that for large systems (more than 100 processors), optimal scheduling decisions are difficult to make, even if the correct decisions are made it results in a high control overhead at heavy load conditions. The second drawback is that the fully distributed algorithms use partial information about the state of the system for suboptimal decisions [Subrata et al., 2008]. A reduced amount of information results in a smaller range of scheduling options. Other problem with fully distributed schemes is of communication delays, which may turn a correct scheduling decision into a wrong choice. Therefore, it may be concluded that, fully distributed algorithms is a better option for small to moderate systems.

3.3.3 Partially distributed

Partially distributed or sometimes called as semi-distributed algorithms are proposed as trade-off between centralized and fully distributed scheduling schemes [Ishfaq and Ghafoor, 1991]. The main idea is that the system is divided into different regions and thus the load balancing problem is divided into subtasks. Each region is generally managed by a single master processor using a centralized scheme. Master processors of each region may exchange information for balancing the load dynamically in the system.

3.3.4 Synchronous versus Asynchronous

The fully distributed and partially distributed schemes may further be categorized as synchronous and asynchronous based on the instant at which load balancing operations are made. In synchronous schemes all processors involved in load balancing carry out balancing operations instantly. Each processor can not proceed with normal computation until the load migrations demanded by the current operations have been completed. On the other hand, in asynchronous approach the running processor takes the load balancing
decision independently. Each processor performs the balancing action regardless what the other processor doing at that time. A number of synchronous and asynchronous load balancing algorithms have been discussed in the literature [Bahi et al., 2005].

3.4 Dynamic Load Balancing Strategies

There are number of approaches to solve non-uniform problems on multiprocessor systems based on the various models discussed above. Some of the most relevant strategies reported in the literature are discussed. These include:

3.4.1 Randomization

In this scheme, the destination processors for load transfer are chosen in a random fashion. These algorithms use local information to make movement decisions. A threshold value or sometimes called ideal load (IL) is calculated which decides, whether a processor is overloaded or underloaded. When a processor detects that there is load imbalance, a processor is randomly selected as a destination of load movement. In some algorithms instead of using only one threshold value, two thresholds values (L1 and L2) are used to decide about the overloaded and underloaded processors. A processor is considered overloaded when its load becomes greater than L1. Similarly, Underloaded processors are those whose load is smaller than L2.

3.4.2 Diffusion

In this scheme the destination node is selected from a pool of neighbor nodes. A neighbor node is one which has a direct link to the source node. One simple method for dynamic load balancing, is to select the neighbor node for load transfer, if it is underloaded. In this way, a local load balance is achieved by migrating the surplus load. The surplus load can be interpreted as diffusion through the processors towards a balance state. Diffusion algorithms assume
that a processor is able to send and receive load to/from all its neighbor nodes simultaneously. If there is no underloaded processor amongst the neighbor nodes, then the nodes on the next levels are selected. In this way the method is iterative to solve the problem of diffusion. LeMair and Reeves classified the diffusion algorithms into two groups [LeMair and Reeves, 1993]:

i) **Sender Initiated Diffusion (SID):** It is an approach which makes use of near neighbor load information to share out surplus load from heavily loaded processors to underloaded neighbor processors in the system. In other words, tasks from heavily loaded processors diffuse into lightly loaded areas in the system. Each processor acts independently and is limited to load information from within its domain, which consists of itself and its immediate neighbor. The underlying processor checks the load of its neighbor processors. If any of the neighbor processors has a load value smaller than the underlying processor’s load, such processors are considered underloaded processors. Once the underloaded processors are identified, the underlying processor evaluates the load difference between itself and each of its underloaded neighbors. Subsequently, a fixed portion of the corresponding load difference is sent to each one of the underloaded neighbors. The difference is calculated based on the average load of the underlying processor such as ideal load (IL). All processors inform their near neighbors of their load levels and update this information throughout program execution.

ii) **Receiver Initiated Diffusion (RID):** In RID, the underloaded processors are the active processors. These processors request load from the overloaded neighbor nodes in the system. The balancing process is initiated by any processor whose load becomes smaller than the prescribed threshold. However, upon receiving a load request, a processor will fulfill the request only up to an amount equal to half of its current load. The majority of overload in RID scheme lies on the underloaded processors.
3.4.3 Dimension Exchange Method (DEM)

It is a global, fully synchronous approach for load balancing. Load balancing is achieved in an iterative fashion by “folding” an N processor system into log₂ N dimensions and balancing one dimension at a time. This method was initially studied for hypercube topologies where processor neighbors are inspected by following each dimension of the hypercube [Cybenko, 1989]. The processors of a k-dimensional hypercube pair up with their neighbors in each dimension and exchange half the difference in their respective load. Then research has been reported on adapting a new and more efficient DE type algorithm named Generalized Dimension Exchange (GDE) strategy [Xu and Lau, 1992], [Xu and Lau, 1995]. Similarly, the Dimension Exchange on hypercube architecture with broken edges has been studied [Bahi et al., 2003]. This is the enhanced version of GDE and termed as Generalized Adaptive Exchange (GAE). A number of policies are further discussed that work on the principles of GAE [Bahi et al., 2005].

3.4.4 Gradient Model (GM)

In this scheme [LeMair and Reeves, 1993], load is restricted to being along the direction of the most lightly loaded processors in the system. The basic procedure is that underloaded processors inform other processors in the system of their state, as a result the overloaded processors respond. That is, an overloaded processor will send its excess load only to one lightly loaded neighbor processor at the end of one iteration of the load balancing algorithm. The scheme is based on the two threshold parameters L1 and L2. A processor is considered overloaded when its load becomes greater than L1, light if below the L2, and moderate otherwise. The Gradient Model scheme differs from the Dimension Exchange scheme in the sense that, in GM, the load information of the entire underlying domain is considered in deciding the destination processor, whereas in DEM only one processor is considered at each iteration.
In the Gradient model algorithm the first step is to determine the loading condition of each individual processor: light, moderate or heavy. The second step consists of establishing a system-wide gradient map to generate route between underloaded and overloaded processors. The gradient map is represented by the aggregate of all proximities. A node’s proximity is defined as the minimum distance from itself to the nearest lightly loaded node in the system. Initially, all the nodes have proximity of $w_{\text{max}}$, a constant equal to the diameter of the system. The proximity of a node is set to zero, if its state becomes light. Every node in the system calculates their proximities. A node’s proximity may not exceed $w_{\text{max}}$. A system is saturated, and does not require load balancing if all nodes report proximity of $w_{\text{max}}$. If the proximity of a node changes it must notify its near neighbors. Hence, balancing process is initiated by lightly loaded processors reporting proximity of zero. If a processor’s state is heavy and any of its neighbors report a proximity less than $w_{\text{max}}$, then it sends a unit of its load to the neighbor of lowest proximity. The proximity map therefore is used to perform the migration phase.

3.4.5 Hierarchical Balancing Method (HBM)

It is an asynchronous and decentralized approach of load balancing [LeMair and Reeves, 1993]. It classifies the multi-computer system into a hierarchy of balancing domain. Each domain has a particular level of load balancing at different levels. Specific processors are designated to control the balancing operations at different levels of the hierarchy. The balancing process at different level is invoked by the receipt of load update messages indicating an imbalance between lower level domains i.e. processor in charge of the balancing process at a level $l_i$, receives load information from lower level, $l_{i-1}$, domains. Global balancing is achieved by ascending the network and balancing the load between adjacent domains at network level in the hierarchy. This procedure is asynchronous, however, where balancing is invoked within a domain whenever an imbalance is detected by the domain’s specific processor.
Different imbalance thresholds can be set at different levels of the hierarchy. The HBM scheme distributes the load balancing responsibilities to all processors in the system. This scheme is effective to manage both the local load imbalance as well as excessive global imbalances.

3.4.6 Minimum Distance Scheduling (MDS)

This is another novel dynamic scheduling scheme for load balancing reported [Rafiq et al., 1999]. The algorithm operates on a minimum distance property which assures the minimization of the communication in distributing tasks among processors. In general, the performance of a multiprocessor system can be characterized by communication delay, distribution of load among the processors and scheduling overhead [Ravikanth et al., 1988], [Reddy, 1993]. Therefore, a close correspondence between the structures of the problem and the architecture of the processors is desired in order to minimize these overheads. When the problem graph topology is not known in priori, the mapping is done on the fly onto the processors. Thus, dynamic load balancing is essential for efficient utilization of highly parallel systems when solving non-uniform problems with unpredictable load estimates. The scheduling techniques may have certain constraints that may vary from application to application. The MDS scheme works to minimize the communication in distributing tasks among processors.

3.4.6.1 Minimum Distance Property: One of the important parameters for proper utilization of multiprocessor systems is the inter-processor communication costs which should be as small as possible. It necessitates some means to reduce these overheads. Therefore, to assign tasks on processors, a scheduling strategy must be designed which take care of minimization of execution and communication costs. Minimum distance is the property which assures the minimization of the communication in distributing subtasks and collecting partial results. Therefore, one of the method to sustain this property is to keep message path lengths to one hop. A scheduling scheme operates with
this property minimizes overhead and ensures the maximum possible speedup. The property may be formally stated as: “If T and \( T_1 \) are the two tasks from a task tree of a given problem such that T is the parent of \( T_1 \) and if P and \( P_1 \) are the processors on which T and \( T_1 \) are scheduled, then, P should be directly connected to \( P_1 \) in the network.” In the MDS algorithm, the adjacency matrix of the network is used to satisfy the minimum distance property [Ravikanth et al., 1988].

The general model of the dynamic load balancing is mainly based on the load balancing profitability determination at various sites in a multiprocessor Network [LeMair and Reeves, 1993]. Whenever, profitable, a scheduler is invoked which migrates tasks to achieve a more uniform distribution of load on processors. The donors (overloaded) and acceptors (underloaded) processors are identified based on a threshold value known as ideal load (IL). Each donor processor, during balancing, selects most suitable tasks (based on task dependencies) for migration thus maintaining minimum distance. Migration from donor processor is done to the directly connected acceptors. Thus, for every donor, there is a set of Minimum Distance Acceptors (MDA). Tasks are not allowed to migrate to acceptors which are outside this set. To perform the load balancing, the algorithm calculates ideal load (IL) value for each iteration, which is used by load balancer as a threshold to detect load imbalances and make load migration decisions. Mostly any load balancing algorithm considers the overall load at a processor. However, in this algorithm the load at a particular stage of the task structure is taken into consideration. The load imbalance factor for \( k^{th} \) stage, denoted as \( \text{LIF}_k \), is defined as

\[
\text{LIF}_k = \frac{\max \{\text{load}_k(P_j)\} - \text{(ideal\_load)}_k}{\text{(ideal\_load)}_k} \quad \ldots \ldots \quad (3.1)
\]

where,

\[
\text{(ideal\_load)}_k = \frac{\text{load}_k(P_0) + \text{load}_k(P_1) + \ldots + \text{load}_k(P_{N-1})}{N} \quad \ldots \ldots \quad (3.2)
\]

and \( \max \{\text{load}_k(P_i)\} \) denotes the maximum load pertaining to stage \( k \) on a
processor $P_i, 0 \leq i \leq N-1$, and $\text{Load}_k(P_i)$ stands for the load on processor $P_i$ due to $k^{th}$ stage. When implemented on Linearly Extensible Tree (LET), the MDS scheme shows that the network has good load balancing properties when considering problem structures having parallelism but non-uniform growth in various branches. The balancer uses the concept of balancing domains which reduces the overhead of the balancing process, but does not ensure a balanced load for entire system. This trade-off is illustrated in the scheduling strategies [Rafiq, 1995].

From the above review, it is apparent that myriad of multiprocessor scheduling strategies exist which can be applied to specific structure of programs and specific system architectures. An optimal scheduling can be made based on some objective functions to enhance the performance of overall system.

In general the following remarks can be highlighted.

- Static approaches are easier to implement and have minimal runtime overhead. However, these schemes are not applicable for parallel systems where computing resources and communication network/traffic are not known in advance.

- Dynamic approaches result better performance but at the cost of high overhead.

- De-centralized schemes are costlier than centralized because it is very difficult to obtain and maintained the dynamic state information of the whole system by individual nodes.

- The total number of iterations on SID and RID policies required to achieve the global balancing are application and topology dependent.
• RID strategy, on the other hand may be implemented easily to simpler topologies and can scale elegantly for large system.

• The efficiency of the DEM and HBM strategies depends heavily on the system interconnection topologies.

Motivated by the linear extensible properties of LET and its performance analysis when MDS scheduling schemes is applied on it, a new cube like interconnection topology named Linearly Extensible Cube (LEC) has been proposed and analyzed. The proposed architecture with topological properties is discussed in the next Chapter 4. A new dynamic scheduling scheme has been devised and implemented to evaluate the performance of the proposed LEC. The simulation results are discussed in Chapter 5.