Use of DAG in Distributed Parallel Computing

Rafiqul Zaman Khan, Javed Ali
Department of Computer Science, Aligarh Muslim University, Aligarh

Abstract
In this paper, we present a systematic analysis of directed acyclic graph (DAG) in distributed parallel computing systems. Basic concepts of parallel computing have been discussed in detail. Many computational solutions can be expressed as directed acyclic graphs (DAGs) with weighted nodes. In parallel computing, a fundamental challenge is to efficiently map computing resources to the tasks, while preserving the precedence constraints among the tasks. Traditionally, such constraints are preserved by starting a task after all its preceding tasks are completed. However, for a class of DAG structured computations, a task can be partially executed with respect to each preceding task. We define such relationship between the tasks as weak dependency. This paper gives the basic idea about parallel computing by using DAG.

Keywords: Parallel Computing, Directed Acyclic Graph, Normalized Schedule Length, Data Dependencies etc.

1. Introduction
Parallel computing allows to indicate how different portions of the computation can be executed concurrently in a heterogeneous computing environment. The high performance of the existed systems may be achieved by a high degree of parallelism. It supports parallel as well as sequential execution of processes and automatic inter process communication and synchronization. Many scientific problems (Statistical Mechanics, Computational Fluid Dynamics, Modeling of Human Organs and Bones, Genetic Engineering, Global Weather and Environmental Modeling etc.) are so complex that solving them via simulation requires extraordinary powerful computers. Grand challenging scientific problems can be solved by using high performance parallel computing architectures. Task partitioning of parallel applications is highly critical for the prediction of the performance of distributed computing systems. A well known representation of parallel applications is a DAG (Directed Acyclic Graph) in which nodes represent application tasks. Directed arcs or edges represent inter task dependencies. Execution time of any algorithm is known as schedule length of that algorithm. The main problem in parallel computing is to find out the minimum schedule length of DAG computations. Normalized Schedule Length (NSL) of any algorithm shows that, variation of communication cost depends upon the overhead of the existing computing system. Minimization of inter-process communication cost and selection of network architecture are notable parameters to decide the degree of efficiency.

2. Types of parallel computing architectures
2.1 Flynn’s taxonomy: Michael J. Flynn created one of the earliest classification systems for parallel and sequential computers and programs. It’s the best known classification scheme for parallel computers. He classified programs and computers for different data partitioning streams. A process is a sequence of instructions (the instruction stream) that manipulates a sequence of operands (the data stream). Computer hardware may support a single instruction stream or multiple instruction streams for manipulating different data streams.

2.1.1 SISD (Single Instruction stream/Single Data stream): It’s equivalent to an entire sequential program. In SISD, virtual network machine fetches one sequence of instruction, data and instruction’s address from memory.

2.1.2 SIMD (Single Instruction stream/ Multiple Data stream): This category refers to computers with a single instruction stream but multiple data streams. These machines typically used by process arrays. A single processor fetches instructions and broadcast these instructions to a number of data units. These data units fetch data and perform operations on them. An appropriate programming language for such machines has a single flow of control by operating an entire array rather than individual array elements. It’s analogous to doing the same operation repeatedly over a large data set. This is commonly used in signal processing applications.

There are some special features of the SIMD.

All processors do the same thing or are idle.
1. It consists data partitioning and parallel processing phase.
2. It produces the best results for big and regular data sets.
Systolic array is the combination of SIMD and pipeline parallelism. It achieves very high speed by circulating data among processor before returning to memory.
2.1.3 MISD (Multiple Instruction /Single Data): It’s not totally clear that which type of machine fits in this category. One kind of MISD machine can be design for failing or safe operation. Several processors perform the same operation on the same data and check each other to be sure that any failure will be caught. The main issues in MISD are: branch prediction, instruction installation and flushing a pipeline. Another proposed MISD machine is a systolic array processor. In this category, stream of data are fetched from memory and passed to the array of processors. The individual processor performs its operation on the stream of accepted data. But they have no control over the fetching of data from memory. This combination is not very useful for practical implementations.

2.1.4 MIMD (Multiple Instructions and Multiple Data): In this stream several processor fetches their own instructions. Multiple instructions are executed upon a different set of data for computations of large tasks. To achieve maximum speed up, the processors must communicate in synchronized manner [3]. There are two types of streams under this category which are as follows.
1. MIMD (Distributed Memory): In this stream unique memory is associated with each processor of the distributed computing systems. So communication overhead is high due to the exchange of data amongst the processors.
2. MIMD (Shared Memory): This structure shares a single physical memory amongst the processors. Programs can share blocks of memory for the execution of parallel programs. In this stream, shared memory concept causes the problems of exclusive access, race condition, scalability and synchronization.

3. Important laws in parallel computing

3.1 Amdahl’s law: This law had been generated by Gene Amdahl in 1967. It provides an upper limit to the speedup which may be achieved by a number of parallel processors to execute a particular task. Asymptotic speedup is increased as the number of processors increases in high performance computing systems [1]. If the numbers of parallel processors in a parallel computing system are fixed then speedup is usually an increasing function of the problem size. This effect is known as Amdahl’s effect.

Suppose \( f \) be the sequential fraction of computations, where \( 0 \leq f \leq 1 \). The maximum speedup achieved by the parallel processing environment by using \( p \) processors is as follows:

\[
\Psi \leq \frac{1}{(f + (1 - f)p)}
\]

3.1.1 Limitations of Amdahl’s Law:
1. It does not provide overhead computation with the association of degree of parallelism (DOP).
2. It assumes the problem size as a constant and depicts how increasing processors can reduce computational overhead.

A small portion of the task which cannot be parallelized will limit the overall speedup achieved by parallelization. Any large mathematical or engineering problem will typically consist of several parallelizable parts and several sequential parts. This relationship is given by the equation:

\[
S = \frac{1}{(1 - \Psi)}
\]

Where \( S \) is the speedup of the task (as a factor of its original sequential runtime) and is a parallelizable fraction.

If a sequential portion of a task is 10% of the runtime, we can’t get more than 10× speedup regardless of how many processors are added. This rule puts an upper limit to add highest number of parallel execution units. When a task can’t be partitioned due to its sequential constraints then more efforts on this application has no effect on the schedule. Bearing of a child takes nine months regardless of the matter of assignment of a number of women [9]. Amdahl and Gustafson laws are related to each other because both laws give a speedup performance after partitioning given tasks into sub-tasks.

3.2 Gustafson’s law: It’s used to estimate the degree of parallelism over a serial execution. It allows that the problem size being an increasing function of the number of processors [5]. Speedup predicted by this law is termed as scaled speedup. According to Gustafson, maximum scaled speedup is given by,

\[
S(P) = P - \alpha(P - 1).
\]

Where \( P \), \( S \) and \( \alpha \) denotes the number of processors, speedup and non-parallelizable part of the process respectively. Gustafson law depends upon the sequential section of the program. On the other hand, Amdhal’s law does not depend upon the sequential section of parallel applications. Communication overhead is also ignored by this performance metric.

4. Performance parameters in parallel computing

There are many performance parameters for parallel computing. Some of them are listed as follows:
4.1 Throughput: It is measured in units of work accomplished per unit time. There are many possible throughput metrics which depends upon the definition of a unit of work. For a long process throughput may be one process per hour while for short processes it might be twenty processes per seconds. It is totally depends upon the underlying architecture and size of executing processes upon that architecture.

4.2 System utilization: It keeps the system as busy as possible. It may vary from zero to 100 percent approximately.

4.3 Turnaround time: It is the time taken by the job from its submission to completion. It’s the sum of the periods spent waiting to get into memory, waiting in ready queue, executing on the processor and spending time for input/output operations.

4.4 Waiting time: It is the amount of time spent to wait by a particular job in ready queue for getting a resource. In other words, waiting time for a job is estimated at the time taken from the job from its submission to get system for execution. Waiting time depends upon the parameters similar at turnaround time.

4.5 Response time: It is the amount of time to get first response but not the time that the process takes to output that response [4]. This time can be limited by the output devices of computing system.

4.6 Reliability: It is the ability of a system to perform failure free operation under stated conditions for a specified period of time.

5. Important objective of parallel computing

5.1 Allocation requirements: Since parallel computing may be used as distributed computing environment. Therefore, the following aspects play an important role in the performance of allocated resources.

5.2 Services: It’s considered that parallel computing is designed to address a single and multiple issues like minimum turnaround time as well as real time with fault tolerance.

5.3 Topology: Whether the job services are centralized or distributed or hierarchical in nature? Selection of appropriate topology is a challenging task for the achievement of better results.

5.4 Nature of the job: It can be predicted on the basis of load balancing and communication overhead of the tasks over the parallel computing architectures [8].

5.5 The effect of existing load: Existing load may cause the poor results.

5.6 Load balancing: If the tasks are spread over the parallel computing systems then load balancing strategy depends upon the nature and size of job and characteristics of processing elements.

5.7 Parallelism: Parallelism of the jobs may be considered on the basis of fine grain level or coarse grain level [6]. After jobs submission or before inserting jobs this parameter should be kept in mind.

5.8 Redundant resource selection: What should be the degree of redundancy in the form of task replication or resource replication? In case of failure what should be the criterion of node selection having task replica? How the system performance is affected by allocating the resources properly?

5.9 Efficiency: Efficiency of scheduling algorithm is computed as:

\[
\text{Efficiency} = \frac{\text{Scheduling Length of Single-processor System}}{\text{Scheduling Length on Multi-processor System} \times \text{No. of Processes}} \times 100
\]

5.10 Normalized Schedule Length: If makespan of an algorithm is the completion time of that algorithm then Normalized Schedule Length (NSL) of scheduling algorithm is defined as follows:

\[
\text{NSL} = \frac{\text{Makespan of Particular Algorithm}}{\text{Maximum of Sum of Computational Cost Along a Critical Path}}
\]

5.11 Resource management: Resource management is the key factor for target of maximum throughput. In management of resources generally includes resource inventories, fault isolation, resource monitoring, a variety of autonomic capabilities and service-level management activities. The most interesting aspect of the resource management area is the selection of the correct resource from the parallel computing resource alternatives.

5.12 Security: Prediction of the heterogeneous nature of resources and security policies is complicated and complex in a parallel computing environment. These computing resources are hosted in different security areas. So middleware solutions must address local security integration, secure identity mapping, secure access/authentication and trust management.

5.13 Reliability: Reliability is the ability of a system to perform its required functions under proposed conditions for a certain period of time. Heterogeneous resources, different needs of the implemented applications and distribution of users at the different places may generate insecure and unreliable circumstances [10]. Due to these problems it’s not possible to generate ideal parallel computing architectures for the execution of large and real time parallel processing problems.

5.14 Fault tolerance: A fault is a physical distortion, imperfection or fatal mistake that occurs within some hardware or software packages. An error is the deviation of the results from the ideal outputs. So failures of the system means error
and an error generate some faults. Due to these limitations parallel computing system must be fault tolerant. Therefore, if any type of problem is happening system must be capable to generate proper results.

6. NP Hard scheduling

Most of scheduling problems can be considered as optimization problems as we look for a schedule that optimizes a certain objective function. Computational complexity provides a mathematical framework that is able to explain why some problems are easier than others to solve[9]. It is accepted that more computational complexity means the problem is hard. Computational complexity depends upon the input size and the constraints imposed on it.

Many scheduling algorithms contain the sorting of at least one job, which require at most \(O(n \log n)\) time. These types of problem can be solved by exact methods in polynomial time. The class of all polynomial solvable problems is called class \(P\).

Another class of optimization problems is known as \(NP\)-hard \((NP\)-complete) problems. In \(NP\)-hard problems, no polynomial-time algorithms are known and it is generally assumed that these problems cannot be solved in polynomial time. Scheduling in a parallel computing environment is a \(NP\)-hard problem due to large amount of resources and jobs are to be scheduled. Heterogeneity of resources and jobs causes scheduling NP-hard.

6.1 Classes \(P\) and \(NP\) in parallel computing: An algorithm is a step-by-step procedure for solving a computational problem. For a given input, it generates the correct output after a finite number of steps. Time complexity or running time of an algorithm expresses the total number of elementary operations such as additions, multiplications and comparisons etc. An algorithm is said to be a polynomial or a polynomial-time algorithm, if it’s running time is bounded by a polynomial in the input size. For scheduling problems, typical values of the running time are e.g., \(O(n^2)\) and \(O(n \log n)\).

1. A problem is called a decision problem if its output range is \{yes, no\}.
2. \(P\) is the class of decision problems which are polynomials solvable.
3. \(NP\) is the class of decision problems with the property that it’s not solvable in polynomial time fashion.

7. DAG Description

A parallel program can be represented by a weighed Directed Acyclic Graph(DAG)[11], in which the vertex/node weights represent task processing time and the edge weights represent data dependencies as well as the communication time between tasks. The communication time is also referred as communication cost. Directed Acyclic Graph (DAG) is a directed graph that contains no cycles. A rooted tree is a special kind of DAG and a DAG is a special kind of directed graph. Directed Acyclic Graph(DAG) \(G = (V, E)\), where \(V\) is a set of \(v\) nodes/vertices and \(E\) is a set of \(e\) directed edges. The source node of an edge is called the parent node while the sink node is called the child node. A node with no parent is called an entry node and a node with no child is called an exit node.

7.1 DAG Applications

DAGs may be used to model different kinds of structure in mathematics and computer science, to model processes in which information flows in a consistent direction through a network of processors, as a space-efficient representation of a collection of sequences with overlapping subsequences, to represent a network of processing elements and etc. Examples of this include the following:

- In electronic circuit design, a combinational logic circuit is an acyclic system of logic gates that computes a function of an input, where the input and output of the function are represented as individual bits.
- Dataflow programming languages describe systems of values that are related to each other by a directed acyclic graph[10]. When one value changes, its successors are recalculated; each value is evaluated as a function of its predecessors in the DAG.
- In compilers, straight line code (that is, sequences of statements without loops or conditional branches) may be represented by a DAG describing the inputs and outputs of each of the arithmetic operations performed within the code; this representation allows the compiler to perform common sub expression elimination efficiently. This paper aims at building a dynamic scheduling model with DAGs[11]. In this model, an assigned processor which is called center scheduler, responsible for dynamically schedules the tasks. Based on the proposed dynamic scheduling model we present a new dynamic scheduling algorithm.

8. Conclusion

Analysis of task scheduling problem is still a challenging problem. DAG is used to minimize the cost of intercommunication factors. This paper represents meaningful analysis of distributed parallel computing architectures.
Scheduling in parallel computing is highly important due to the efficient use of high performance computing systems. Important laws related to parallel computing are also summarize. DAG based applications are also discussed in the paper.

References:

BIOGRAPHY AUTHORS:
Dr. Rafiqul Zaman Khan, is presently working as a Associate Professor in the Department of Computer Science at Aligarh Muslim University, Aligarh, India. He received his B.Sc Degree from M.J.P Rohilkhand University, Bareilly, M.Sc and M.C.A from A.M.U. and Ph.D (Computer Science) from Jamia Hamdard University. He has 19 years of Teaching Experience of various reputed International and National Universities viz King Fahad University of Petroleum & Minerals (KFUPM), K.S.A, Ittihad University, U.A.E, Pune University, Jamia Hamdard University and AMU, Aligarh. He worked as a Head of the Department of Computer Science at Poona College, University of Pune. He also worked as a Chairman of the Department of Computer Science, AMU, Aligarh. His Research Interest includes Parallel & Distributed Computing, Gesture Recognition, Expert Systems and Artificial Intelligence. Presently 06 students are doing PhD under his supervision. He has published about 46 research papers in International Journals/Conferences. Names of some Journals of repute in which recently his articles have been published are International Journal of Computer Applications (ISSN: 0975-8887), U.S.A, Journal of Computer and Information Science (ISSN: 1913-8989), Canada, International Journal of Human Computer Interaction (ISSN: 2180-1347), Malaysia, and Malaysian Journal of Computer Science(ISSN: 0127-9084), Malaysia. He is the Member of Advisory Board of International Journal of Emerging Technology and Advanced Engineering (IJETAE), Editorial Board of International Journal of Advances in Engineering & Technology (IJAET), International Journal of Computer Science Engineering and Technology (IJCSET), International Journal in Foundations of Computer Science & technology (IJFCST) and Journal of Information Technology, and Organizations (JITO).

Javed Ali is a research scholar in the Department of Computer Science, Aligarh Muslim University, Aligarh. His research interest include parallel computing in distributed systems. He did Bsc(Hons) in mathematics and MCA from Aligarh Muslim University, Aligarh. He published seven international research papers in reputed journals. He received state level scientist award by the government of India. He published 13 papers in the journal of international repute.