Analysis of Different Parallel Programming Models

Kaushik Chandrasekaran
Indiana University Bloomington

Abstract

This paper gives a vision of programming models used for parallel computation and compares various features of different programming models. This paper provides the architecture of few programming models such as Bulk synchronous model, MPI Primitives and MapReduce. It also discuss about some of the drawbacks found in these programming models.

1. Introduction

There are a number of programming models and techniques to process data on large clusters. Data could be of different forms like raw data, graph structure of web documents etc. All these data cannot be processed on a single system as they require huge amount of memory and thus they are distributed across clusters of computers and are processed in parallel. Parallel processing of data is not an easy task and face a number of challenges like fault tolerance [4], how the data is to be distributed etc. This paper describes about the different models that can be used to process data and the challenges faced by these models. Some of the programming models that are used to process data include MapReduce [5], bulk synchronous programming [1], MPI primitives [3] etc. The following sections of the paper deals with understanding some of the techniques used for parallel processing of huge data.

2. Bulk Synchronous Programming Model

All models must satisfy three basic aspects of parallelism [2]. One is that it must provide good transportability with sequential computations. Second is developing programming languages that are appropriate for hosting parallel computations. The final one is developing compilers that produce highly efficient code. The bulk synchronous parallel model was proposed by Valiant that addresses all the above aspects. We all know that Scalability in parallel computing can be achieved using machine dependent software. However in order to achieve portability, Scalability must be sacrificed. The bulk synchronous parallel model is aimed at achieving both high scalability and efficiency without having to sacrifice one for the other. In the BSP model a parallel machine consists of a number of processes that perform various tasks. Computations in BSP are divided into supersteps. In each superstep a processor may perform the following operations:

- Local operations on the data
- Send or receive packets.

Each process in a BSP model has its own local memory. A packet sent in one superstep is delivered to the destination processor at the beginning of the next superstep. The communication time of an algorithm in BSP model is given by a simple cost function. The BSP model consists of the following parameters:

1. $P$ = The number of processors
2. $g$ which denotes the bandwidth per-processor basis
3. $L$ which denotes the latency to send a packet through a network

2.1 BSP Algorithm

Consider a program consisting of $S$ supersteps, then execution time for superstep $i$ is given by

$$W_i + gh_i + L$$

$W_i$ = Largest amount of work done, which is also referred to as work depth
$h_i$ = largest number of packets sent or received by any processor during the $i$th superstep
$L$ = Latency to send a packet through a network

The execution time of the entire program is given by:

$$W + gh + L$$

Thus the BSP model must follow the following principles:

1. Minimize the work depth
2. Minimize the maximum number of packets sent and received by any processor in each superstep
3. Minimize the total number of superstep in a program
Thus for an algorithm designer the BSP model provides a simple cost function to analyze the complexity of algorithms. The BSP model thus helps in providing both scalability and performance without any compromise of one. The BSP algorithm makes it simpler for the programmer to perform parallel operations.

3. Message Passing Interface

MPI stands for Message Passing Interface. MPI is a language-independent communications protocol used to program parallel computers. In other words it allows processes to communicate with each other by sending and receiving messages. The MPI supports both point to point as well as collective communication. MPI is basically run on homogeneous or heterogeneous parallel or distributed system. The Message Passing Interface consists of the following concepts:

3.1 Communicator

The Communicator is generally used to connect a group of processes. The Communicator basically gives an identifier to each process and arrange all the processes that are contained by it in an ordered topology. MPI defines an initial communicator MPI_COMM_WORLD that contains all the processes of the program as well as a unique communication context. MPI also allows communicators to be partitioned. This is done by MPI_COMM_SPLIT.

3.2 Point-to-Point Communication

MPI supports point-to-point communication where a process sends or receives a message from another process. (i.e. it allows two way communication between the processes). A process sends a message to another processor using the MPI_SEND. The MPI_SEND consists of the following parameters:

1. initial address of the send buffer
2. number of elements in the send buffer
3. Datatype of each send buffer element
4. Rank of destination
5. Message Tag
6. Communicator.

Similarly the receiving process receives the message using the MPI_RECV. The MPI_RECV contains the exact parameters as for MPI_SEND except it has a receive buffer instead of a send buffer and has one more parameter called status which is used to indicate the object status.

3.3 Collective Communication

Collective communication involves communication between groups of processes. An example of collective communication is MPI_Bcast call. The root process sends its information to all other processes using this call. The MPI_Bcast consists of the following parameters:

1. starting address of buffer
2. number of entries in buffer (integer)
3. data type of buffer (handle)
4. rank of broadcast root (integer)
5. Communicator

3.4 Derived Datatype

MPI functions require that you specify the type of data, which is sent between processors. Certain predefined MPI datatypes are MPI_INT, MPI_CHAR etc.

The Following snippets consist of an example implementation of MPI that was used to calculate the PageRank [9] values of different web pages.
4. MapReduce

MapReduce [6] is a programming model and an associated implementation for processing and generating large data sets. MapReduce is generally a software framework that was introduced by Google to process large amount of data on distributed or parallel computers. The MapReduce has a number of advantages compared to the other models used to process data. Computational Process takes place on data stored on both structured as well as an unstructured system. Basically MapReduce makes use of 2 functions. They are the map function and the reduce function. The map function takes in a series of key/value pair and gives zero or more output key/value pairs. The Reduce function is applied to all the computers in parallel and it returns a list of final output values. The map and reduce functions are similar to how they are used in functional programming (i.e. both the map function as well as the reduce function are written by the user). The main advantage of mapreduce is that it allows distributed processing of map and reduce operations.

4.1 Architecture Overview

Figure 3 gives the execution/architecture overview of the MapReduce model. Initially the input files are split and are distributed to the worker. Workers here refer to the computers that are involved in the parallel computation. There is one master, which assigns work to the idle workers. The work maybe a map operation or a reduce operation.

Map Operation

When a master assigns a worker with a Map operation the worker reads the corresponding input file and performs a mapping on it to generate an intermediate key/value pairs. These intermediate key/value pairs are then sorted in the local buffer.

Reduce Operation

The Reduce worker uses Remote procedure Calls to access the local buffer and reads the intermediate key/value pairs. After reading all the intermediate values the reduce worker sorts the key/value pairs because different keys may map to the same reduce worker. Once the sorting is done the Reduce worker iterates over the sorted intermediate data and for each unique intermediate key encountered, it passes the key and the corresponding values to the user’s Reduce function. The output of this function is appended along with the final output. After all the Map and Reduce operations are performed the master wakes up the user program.
5. Challenges and Drawbacks

All the above programming models have a number of drawbacks and challenges. Some of them are:

Worker Failure

In the case of MapReduce after a master has scheduled an operation to the worker and there is no response from the worker, it marks the worker has failed. This worker is then reset back to the idle state so that it becomes eligible for rescheduling. The workers are also reset back to the idle state after the completion of every map and reduce operation.

Fault Tolerance

Obtaining a proper fault tolerance mechanism for these programming models are really difficult. There are a number of approaches for fault tolerance in parallel computing [7,8]. Jakadeesan G et al in their paper explain about a classification based approach to fault tolerance [8]. They take into account the structural and behavioral characteristics of the message passed.

6. Conclusion

The main difference between the BSP model, MPI primitives and MapReduce is that MapReduce exploits a restricted programming model to parallelize the user program automatically and to provide transparent fault tolerance.

7. References


