An Approach to Parallel Processing

Yashraj Rai
Computer Engineering
Ramrao Adik Institute of Technology
Navi Mumbai, India
yashraj62@yahoo.co.in

Puja Padiya
Computer Engineering
Ramrao Adik Institute of Technology
Navi Mumbai, India
puja.padiya@gmail.com

Abstract—Parallel processing offers enhanced speed of execution to the user and facilitated by different approaches like data parallelism and control parallelism. Graphic Processing Units provide faster execution due to dedicated hardware and tools. This paper presents two popular approaches and techniques for distributed computing and GPU computing, to assist a novice in parallel computing technique. The paper discusses environment needs to be setup for both the above approaches and as a case study demonstrate matrix multiplication algorithm using SIMD architecture.

Keywords-*Distributed Computing; High performance Computing; Message Passing Interface; parallel computing;*

*****

I. INTRODUCTION

The utilization of unused processors to execute more than one job in parallel lead to parallel processing. The first parallel processors were Massively Parallel Processors (MPP's) which were launched in mid 1980's. In the late 1980's clusters came to compete and eventually displace MPP's for many applications. Parallel computing today are possible using multi-core processors [1]. All the processors available today commercially are multi-core processors. The famous brand Intel develops three main processors, i3, i5 and i7. The i3 processor is a dual core processor and the i5 and i7 are quad core processors. To increase the level of parallelism, today we can use unutilized cores of Graphics Processing Unit (GPU) which are part of systems used in high graphics applications. The GPU can have 500 times more cores than the CPUs. There are many applications using parallel processing today. Some of the applications are:

- Scientific applications, like bioinformatics, high performance computing, weather modeling, flood prediction, etc [2].
- Commercial applications, like web and database servers, data mining and analysis for optimizing business and marketing decisions [2].
- Applications in Computer Systems, like intrusion detection in computer security, area of cryptography, etc [2].

We now discuss about the algorithm models, an algorithm model is typically a way of structuring a parallel algorithm by selecting decomposition and mapping technique and applying the appropriate strategy to minimize interactions [2]. There are two main types of algorithm models, data parallelism, and task parallelism.

Data parallelism, the tasks are statically or semi-statically mapped onto processes and each task performs similar operations on different data. It is a result of identical operations being applied concurrently on different data items.

Task parallelism, the interrelationships among the tasks are utilized to promote locality or to reduce interaction costs. It is a parallelism that is naturally expressed by independent tasks in a task-dependency graph [2].

We will be using data parallelism for our parallel computing environment. And we can perform parallel computing in two ways, (1) Parallel and Distributed Systems, by connecting systems over a high speed LAN and using processors of all the systems to execute a job, and (2) GPGPU, when GPU are used for general purpose tasks by using their cores for parallel processing.

Many decomposition techniques are also used in parallel processing, Data Decomposition is one of them, and we will be using this decomposition method in both our approach. Data Decomposition is a powerful and commonly used method for deriving concurrency in algorithms that operate on large data structures [2].

There are many architectures being used today in the implementation of parallel systems. We will be using Single Instruction, Multiple Data (SIMD) [2] architecture for our environment. We will be using this architecture in both the distributed and GPGPU environment. In SIMD, a single control unit dispatches instructions to each processing unit and the same instruction is executed synchronously by all processing units [2].

II. AN ENVIRONMENT FOR GENERAL PURPOSE GPU (GPGPU)

The GPU is a graphics processing unit whose primary task is to render graphics on the display screen. These units have very high number of cores which support parallel processing. Since high level of processing is required when a high resolution and high graphics image or video is played, these processors are dedicated for only one purpose. But tools and languages have been developed to use these processors for general purpose processing. We will be using Nvidia GPU, which is available in different versions in market. We will use the Nvidia GeForce graphics card for our GPGPU processing. Another famous graphics card available in market is AMD Radeon. Nvidia graphics card will be easier to work on as; Nvidia has developed a specific language, CUDA, which works only on Nvidia graphics card. The other language which is used for graphics programming is OpenCL. OpenCL is open source and supported by both the graphics card. OpenCL is managed by the Khronos group. One big advantage of OpenCL is, we don't need to write different code for CPU and GPU as we have to do in CUDA, in OpenCL the GPU code will also work for CPUs whereas in CUDA we need to write specific codes for GPU and CPU. We will be using Nvidia graphics card with CUDA programming in our GPGPU processing.

Hardware specification of our high performance system includes:

- Intel core i5 processor
- 8 GB of RAM
- NVIDIA GeForce 940M GPU with 4 GB memory

We have used the above mentioned NVIDIA's graphics card in our GPGPU environment. There are 384 cores in our system's graphics card. We have to achieve parallelism by assigning tasks to different processors of the GPU, which will be achieved by executing CUDA program, with proper set of instructions. There are many versions of the GeForce graphics card, and only few support the CUDA functionality. So to run CUDA in a GeForce card, we need to first check the compatibility from the NVIDIA website. Once the compatibility check is complete, we move forward to download the NVIDIA CUDA Toolkit, which contains the complete libraries of CUDA and OpenCL. It also contains drivers for running the CUDA program on the Nvidia graphics card. It is a GUI based toolkit, which makes the implementation very easy for the user. We can also customize the Toolkit interface. The Nvidia CUDA Toolkit is operating system specific; all the versions of operating systems are not supported. For example, Ubuntu's latest version is 16.04 but Nvidia toolkit is only supported in 12.04 and 14.04 versions of Ubuntu.

The CUDA program is written in Microsoft's Visual Basic, if we are using Microsoft Window as our operating system. Only CUDA enabled compiler is required if we are executing the programs in Linux operating system. We have opted for Linux based operating system, i.e. Ubuntu, so we have downloaded a supporting CUDA enabled compiler.

III. An Environment for Parallel and Distributed Computing

In this section we discuss about the parallel and distributed environment which was setup for testing parallel programming using Message Passing Interface (MPI). We have used shared memory to access data over the network when programming with MPI. The shared memory environment is created by using Network File System (NFS), which mounts a drive on the network, to be accessed by all the systems connected to that network.

The secure connection is provided by the Secure Shell (SSH) which connects two or more system in a client-server architecture using Openssh prompt in Linux based operating system [3]. The clients and servers are named at this time of the stage. They are labeled properly while using the commands in the configuration. We can increase the number of systems in the LAN to increase the level of parallelization. We had used Data decomposition in our experiment to divide the program in different systems.

The high performance computing using the parallel and distributed computing can be used to connect homogeneous or heterogeneous systems [2]. There are lots of tools available today for high performance computing. For example, MPI, PVM, BLACS, PICL, HPF, etc [4]. These tools are classified as portable tools or hardware specific tools. There is a lot of research going in standardizing the software tools available for parallel and distributed computing. We will be using MPI in our environment for high performance computing.

There are different types of communications that take place when we connect the systems using MPI.

Point-to-Point Communication, The point-to-point communication is the simplest form of communication between two systems. This communication method can be implemented using two simple functions given in MPI library send and receive. The send function is basically used to send data from one machine to another. E.g. - Sending data for processing from server to client, or sending the processed data from client to server. The receive function is used to receive data from another machine. Every send function should have a receive function of its own as the data which is sent and received has a unique identifier [5].

It can be further divided into Synchronous and Asynchronous communications.

- In Synchronous communication, after assigning a job to the client the server waits for the reply and then once the reply is received it starts its execution again.
- In Asynchronous communication, the server assigns the job to the client and continues its work, once the client completes the task; it sends the reply to the server, which is accessed by the server once it has finished its current task [5].

Group Communications, The group communications work when there are more than two systems involved in the distributed computing. These group communications can be further divided into one-to-many communication, many-to-one communication and many-to-many communication [5].

- One-to-Many communication, in this communication one system sends data to many receivers; this can also be referred to as broadcast or multicast of the information over the LAN. This communication is used when a server has multiple clients and the server has to send data to multiple clients at once.
- Many-to-One communication, in this communication many systems send data to one receiver. This is just the opposite of the One-to-Many communication. E.g. when the clients have finished processing, they send the result together to the server. This is a reduction operation [5].

For a parallel and distributed computing environment, our hardware requirement involves two systems with powerful processors and high memory and one high speed network to connect the systems. The software portion required for the environment involves high speed communication protocol, MPI compiler and a Linux based operating system. The flow of our system is shown in figure 1.

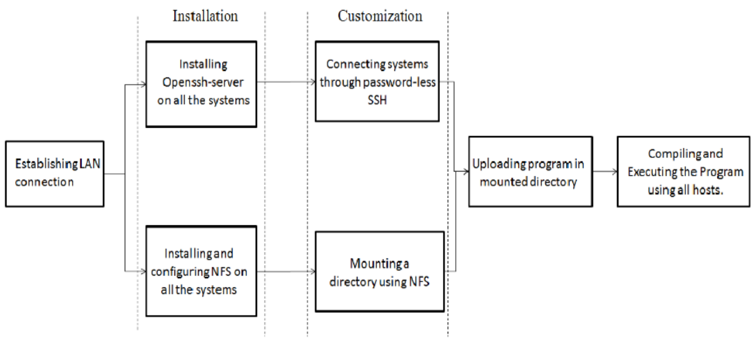
Installation
Customization
Installing Openssh-server on all the systems
Connecting systems through password-less SSH
Establishing LAN connection
Uploading program in mounted directory
Compiling and Executing the Program using all hosts.
Installing and configuring NFS on all the systems
Mounting a directory using NFS

Figure.1 Process Flow

A. *Hardware Support*

High performing systems with following hardware support were used:

- Inter core i3 processors @ 2 GHz
- 2 GB RAM

- Ethernet Network Adapter with bandwidth of 1Gbps
- Wi-Fi Network Adapter for portable systems or laptops

High speed network with the following properties are required for hardware support:

- High speed CAT 5 cable
- A Network Switch if more than 2 systems are connected
- Wi-Fi connection

We should use the best possible combination of tools so that the network speed is high and reliable.

B. *Software Support*

We had decided on working on Linux based platform, so, we selected Ubuntu as our operating system. Ubuntu is an open source platform used in smart phones, tablets and PC. Ubuntu is derived from a very famous Linux project, Debian [6]. Since Ubuntu is open source and software packages are easily available online, and it is easy to handle for new users, we preferred this operating system. The latest version available is 16.04, but we will be using a more stable version i.e. 15.04. So that we can easily get the solutions of any problems arising during the setup of the environment, as the support for the operating system is very dependable.

After deciding on the operating system, we move on to the programming language. As we have already decided, we will be using MPI in our distributed environment. To execute MPI programs we need MPI compiler which is supported in Ubuntu versions till 15.10, i.e. MPICH2. If we are using Ubuntu version 16.04, we will have to install MPICH. MPICH and MPICH2 can be downloaded by online. We should make sure that the MPICH2 versions on all the systems are same. If we fail to do this check then, while executing this program, we will get error while connecting to other systems in the cluster.

For accessing the resources over the LAN, we required shared memory. This shared memory was implemented by Network File System (NFS), the NFS allows remote hosts to mount file systems over a network and interact with those file systems as though they are mounted locally. This enables system administrators to consolidate resources onto centralized servers on the network [7].

We can assign the number of clients in a job and total number of processes needed for the job. These processes are divided equally to all the client systems mentioned during the execution step. The server assigns the tasks to the clients and the clients return the results after the job is completed.

All the instructions are given by the server; other systems perform their own tasks. Only the processes are utilized from the client, which occur at the background.

IV. Results and Analysis

We discuss about the results generated during the execution of the matrix multiplication test case.

The minimum hardware requirement for a system in the distributed environment is:

- Dual core processor @ 2GHz
- 2 GB RAM
- Ethernet/Wi-Fi adapter for network connectivity

The software requirement includes:

- A Linux based operating system, Ubuntu
- A compiler for MPI, Mpich2
- Openssh-server
- Network File System (NFS)

We saw some marginal improvement in the time of our setup of distributed system. We tried with different number of processes,

Number of processes – 2
Execution time in 1 system - 0.002512 seconds
Execution Time in cluster - 0.021693 seconds

Number of processes – 10
Execution time in 1 system - 0.060686 seconds
Execution Time in cluster - 0.058349 seconds

Number of processes – 50
Execution time in 1 system - 0.158426 seconds
Execution Time in cluster - 1.232802 seconds

We see that when the number of processes is 10, we get 0.002 seconds of improvement using the cluster of systems in distributed environment for execution of matrix multiplication.

V. Conclusion

We have implemented parallel processing using Message Passing Interface in distributed computing environment. We did a comparative study by executing a matrix multiplication program in a single system and in parallel using a cluster of systems connected over a LAN. Our study showed that program executed in a cluster took less time than in a single system.

References

[1] http://www.intel.com/pressroom/kits/upcrc/ParallelComputing_backgrounder.pdf
[2] Kumar, Vipin, et al. Introduction to parallel computing: design and analysis of algorithms. Vol. 400. Redwood City, CA: Benjamin/Cummings, 1994.
[3] http://mpitutorial.com/tutorials/running-an-mpi-cluster-within-a-lan
[4] http://people.eecs.berkeley.edu/~culler/machines/tools.html
[5] Hariri, Salim, et al. "A message passing interface for parallel and distributed computing." High Performance Distributed Computing, 1993., Proceedings the 2nd International Symposium on. IEEE, 1993.
[6] http://www.ubuntu.com/about/about-ubuntu
[7] https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/6/html/Storage_Administration_Guide/ch-nfs.html