ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 3, Issue 10, October 2015

Survey on Pure Parallel Programming Models: OpenMP and MPI

Prof. Vinayak D. Shinde, Sampada Patil

HOD & Assistant Professor, Dept. of Computer engineering, Shree L. R. Tiwari College of Engineering,, Thane, India

Lecturer, Dept. of Information Technology, Thakur Polytechnic, Kandivali, India

ABSTRACT*:* This paper presents all pure parallel programming models. shared and distributed memory approaches are also reviewed in this paper. This paper introduces the contribution of Open standards such as Open Multi Processing (OpenMP), Message Passing Interface (MPI) in parallel computing. This survey is accomplished with study of different programming languages according to parallel models.

KEYWORDS: Distributed Programming, parallel Models, MPI, openMP

I. INTRODUCTION

Programming models are abstract so that a limited variety of machine architectures implement the same model. Programming models simplify developers work by maintaining a clean separation between the software solution and the hardware implementation [2].

There are two general categories of programming models for parallel computing:

1. shared memory models
2. distributed memory models

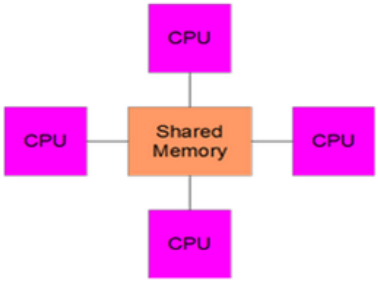

Fig 1: Shared Memory Architecture

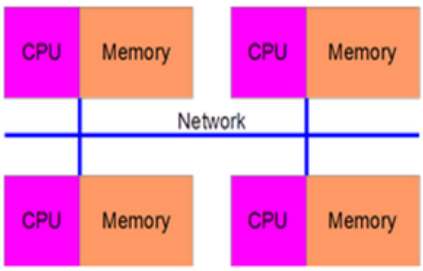

Fig 2: Distributed Memory Architecture

OpenMP used in shared memory architecture, other is MPI used in distributed memory systems. These two models are termed as Pure Parallel Models [1].

II. PURE PARALLEL PROGRAMMING MODELS

In this paper, parallel programming models using a pure shared or distributed memory approach is considered. We consider the shared memory OpenMP and distributed memory MPI models [1]. In the previous section, we mentioned that

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 3, Issue 10, October 2015

pure parallel programming models are divided into two types: shared memory model and distributed memory model. OpenMP and MPI are used to implement these models. Differences between these two models are given in Table 1.

Parameters	MPI	OpenMP
Portable	Yes	Yes
Scalable	Yes	Partially
Supports data parallelism	No	Yes
Supports incremental parallelization	No	Yes
Serial functionality intact	No	Yes
Correctness verifiable	No	Yes

Table 1: Comparison between OpenMP and MPI

Shared Memory OpenMP

OpenMP is a shared memory Application Programming Interface (API) whose aim is to ease Shared Memory parallel Programming. The OpenMP multithreading interface is designed to support HPC programs. Also it is portable across Shared Memory Architectures. OpenMP is implemented as a combination of a set of pragmas, compiler directives and a runtime providing both management of the thread pool anda set of a library routines . The compiler is instructed by these directives to create threads, perform synchronization operations, and manage shared memory. Therefore ,to understand and process these directives , OpenMP does require specialized compiler support. At present , an increasing number of OpenMP versions for Fortran, C and C++ are available in free.

Since the use of threads in OpenMP was designed specifically for parallel applications, it is highly structured. The switch between sequential and parallel sections of code follows the fork/join model [3]. This is a block-structured approach for introducing concurrency. A single thread of control splits into some number of independent threads (the fork). All the threads resume sequential execution , when they complete the execution of their specific task. A fork/join block corresponds to a parallel region. These parallel region is defined using PARRALLEL and END PARALLEL DIRECTIVES.

The parallel region enables a single task to be replicated across a set of threads. The distribution of different tasks across a set of threads is very common in parallel programs. A set of directives enable each thread to execute a different task. Th4is procedure is called work sharing. Therefore , OpenMP is specially suited for the loop parallel program structure pattern.

Application-oriented synchronization primitives which make easier to write parallel programs are provided by OpenMP. If we include these primitives as basic OpenMP operations, it is possible to generate efficient code more easily than using Pthreads [1] .

The OpenMP 3.0 version was released in May 2008 [4]. The major change in this version was the support for explicit tasks. Explicit tasks ease the parallelization of applications where units of work are generateddynamically, as in recursive structures or n while loops. This new characteristics is very powerful. It is possible to handle graph algorithms and dynamic data structures by supporting while loops and other iterative control structures.

The OpenMP parallelization model

From the operating system point o view, while the user's job simply consists in inserting suitable parallelization directives into the code , openMP functionality is based on the use of thread. The sequential functionality of the code sould not be influenced y these directives. This is supported by taking the form of Fortran comments and C/C++ preprocessor programs. The code blocks marked by OpenMP directives are transformed into threaded code by compiler which is aware of an OpenMP. At run time , the user can decide what resources shold be made available to the parallel parts of his executable and hoe they are organized or scheduled. This is illustrated in following figure:

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 3, Issue 10, October 2015

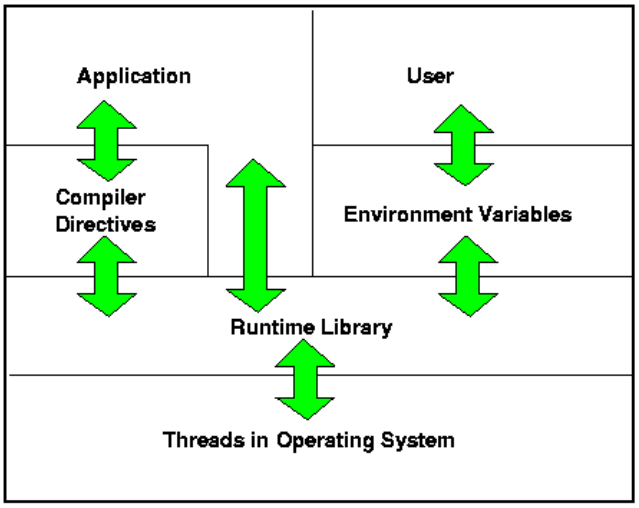

Fig 3 : OpenMP Architecture

OpenMP Shared-Memory programming

A fork-join model of parallelism is provided, where the programmer specifies parallelism in code using OpenMP directives. The OpenMP directives are embedded in the code either as special comments in FORTRAN or as programs in C and C++. Parallel regions and work sharing constructs enable the programmer to express parallelism at the level of structured blocks within the program, such as loops and program sections. In OpenMP, work sharing constructs include the OMP DO loops in Fortran and OMP FOR loops in C/C++ and OpenMP sections. OpenMP directives for work-sharing constructs may include a scheduling clause, defining the way work will be mapped onto parallel threads. Also OpenMP loops may include a reduction clause [5].

Message Passing

One of the parallel programming model is Message Passing. In Message Passing model, communication between processes is done by interchanging messages. Message Passing is a natural model for distributed memory architecture, the communication cannot be achieved by sharing variables. There are more or less pure realizations of this model such as ARMCI. ARMCI stands for Aggregate Remote Memory Copy Interface which allows a programming allows a programming approach between message passing and shared memory. However, over time, a standard has evolved and dominated for this model. That standard is nothing but the Message Passing Interface (MPI)

MPI

MPI is a specification for message passing operations. MPI is not a language. It is a library. MPI specifies the names, calling sequences, and results of the subroutines or functions which is to be called from FORTRAN, C, C++ programs. This programs can be compiled with ordinary compilers but must be linked with the MPI Library.

MPI applications can be run on different nodes of computational Grid implementing well-established middlewares such as Glous. MPI addressed the message-passing model. In this model, the processes executed in parallel have separate memory address space. When part of the address space of one process is copied into the address space of another process then communication occurs. This operation is cooperative. In this operation the first process executes a send operation and second process executes receive operation. This is illustrated in following figure

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 3, Issue 10, October 2015

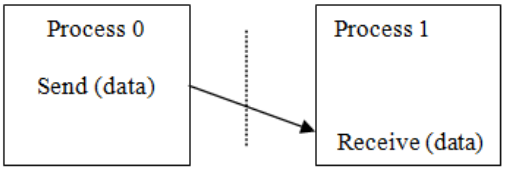

Fig 4 : MPI Point – To – point Communication

Like pthreads [1] in MPI, the programmer does the workload partitioning and task mapping. Programmers have to manage what tasks are to be computed by each process. In MPI, communication models comprises collective, point – to – point, one-sided and parallel I/O operations. Point-To-Point operations (MPI send and MPI recv pair) facilitate communication between processes. Collective operations (MPI_Bcast) ease communications involving more than two processes. Regular MPI send/receive communication uses a two-sided model. This mean that matching operations by sender and receiver are required. Therefore, to manage the matching of sends and receives , some amount of synchronization is needed.

In MPI , there are four levels of thread safety that a user must select explicitly:

1. MPI THREAD SINGLE
   The process has only one thread of execution.
2. MPI THREAD FUNNELED
   The process may be multithreaded, but only the thread that initialized MPI can make MPI calls.
3. MPI THREAD SERIALIZED
   The process may be multithreaded, but only one thread at a time can take MPI calls.
4. MPI THREAD MULTIPLE
   The process may be multithreaded and multiple threads can call MPI functions simultaneously

MPI Communication Protocols

MPI libraries are very complex and in order to be able to correctly interpret the gathered data. Indeed every MPI library exposes several "Knobs" which can be used to tune the performance of a particular application on the underlying target platform [6]. "Eager Limit" is the most relevant threshold for point-to-point communication. The eager protocol is not standardized by the MPI specification, however it is an implementation technique utilized by all MPI implementations. Every message exchanged between peer processes is subject to this protocol. MPI libraries typically use (at least) two algorithms, eager and rendezvous. When the size of the transmitted message is smaller than the specified threshold value, the message (together with an MPI header) is eagerly sent to the receiver. For larger messages the rendezvous protocol is utilized instead, i.e. the sender process sends a readyto-send message (RTS) to the receiver and blocks waiting for the acknowledgment, the clear-to-send (CTS), from the matching receive. The eager protocol is useful when latency is important because it avoids CTS/RTS round-trip overhead. However it requires additional buffering at the receiver side. Rendezvous protocols are typically used when resource consumption is critical [7].

III. CONCLUSION

The characteristics of OpenMP allow for a high abstraction level, making it well suited for developing HPC applications in shared memory systems.

MPI is well suited for applications where portability, both in space and in time is important. MPI is also an excellent choice for task-parallel computations and for applications where the data structures are dynamic, such as unstructured mesh computations.

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 3, Issue 10, October 2015

REFERENCES

[1] Chougule Meenal D et al, / (IJCSIT) International Journal of Computer Science and Information Technologies, Vol. 5 (4) , 2014, 5268-5271
[2] programming models, https://scs.senecac.on.ca/~gpu621/pages/content/platf_p.html#pro
[3] G.R. Andrews, Foundations of Multithreaded, Parallel, and Distributed Programming. Addison Wesley, 1999.
[4] OpenMP 3.0 Specification, http://www.openmp.org/mp-documents/ spec30.pdf, Oct. 2011.
[5] Basumallik, Ayon Ph.D., Purdue University. Compiling SharedMemory Applications for Distributed-Memory Systems. Major Professor: Rudolf Eigenmann , December, 2007.
[6] S. Pellegrini, T. Fahringer, H. Jordan, and H. Moritsch, "Automatic tuning of mpi runtime parameter settings by using machine learning," in Proceedings of the 7th ACM international conference on Computing frontiers, ser. CF '10. New York, NY, USA: ACM, 2010, pp. 115–116.
[7] M. Chaarawi, J. M. Squyres, E. Gabriel, and S. Feki, "A tool for optimizing runtime parameters of open mpi," in Proceedings of the 15th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface. Berlin, Heidelberg: SpringerVerlag, 2008, pp. 210–217.