© 2017 IJSRSET | Volume 3 | Issue 6 | Print ISSN: 2395-1990 | Online ISSN : 2394-4099
Themed Section: Engineering and Technology

A Parallel Hybrid Approach With MPI And OpenMP

Janak Thakkar*[1], Dr. Mehul Parikh[2], Bakul Panchal[3]

*[1]IT Department, R C Technical Institute, Ahmedabad, Gujarat, India

[2]Associate Professor, IT Department, LDCE, Ahmedabad, Gujarat, India

[3]Assistant Professor IT Department, LDCE, Ahmedabad, Gujarat, India

ABSTRACT

Parallel Hybrid Computing has been an emerging research field in Parallel Computing and Multi Core and Many Core Programming in recent years. It is the hybrid approach to make the combination of MPI (Message Passing Interface) and OpenMP(Open Multi-Processor) programming structure to enhance the speed and running time of the program. In the last few years, hybrid approach became the eye catching approach for the large and iterative programs to enhance the performance and reduce the delay. With PHC we can get the advantage of shared memory paradigm with the distributed memory system. with this hybrid approach we can enhance the performance of the sorting algorithm, iterative programs like Pi and Prime Numbers, and all computing algorithms. Paper will mainly focus on the comparison of the Pure MPI, OpenMP and the Hybrid Version of both approaches.

Keywords: OpenMP, MPI, Parallel Hybrid Computing, Threads and MPI, HPC

I. INTRODUCTION

Parallel Hybrid Computing in the need of today as it combines the MPI (Message Passing Interface) and OpenMP(Open Multi-Processor) programming structure. with that hybrid approach we can increase the speed and running time of the program. Often, hybrid MPI+OpenMP programming denotes a programming style with OpenMP shared memory parallelization inside the MPI processes (i.e., each MPI process itself has several OpenMP threads) and communicating with MPI between the MPI processes, but only outside of parallel regions. For example, the MPI parallelization is based on a domain decomposition, the MPI communication mainly exchanges the halo information after each iteration of the outer numerical loop, and these numerical iterations itself are parallelized with OpenMP, i.e., (inner) loops inside of the MPI processes are parallelized with OpenMP work-sharing directives. However, this scheme is only one style in a set of different hybrid programming methods. This hybrid programming scheme will be named master only in the following classification, which is based on the question, when and by which thread(s) the messages are sent between the MPI processes. We pinpoint cases where a hybrid programming model can indeed be the superior solution because of reduced communication needs and memory consumption, or improved load balance. We will also represent the proposed hybrid model for the MPI+OpenMP programming structure.

The remaining document is organised as follow: organization of this document is as follows. In Section 2, we will discuss Pure MPI and Pure OpenMP structure. In section 3, we will represent basic comparison of MPI and OpenMP Programming Structure. In section 4, we will represent the Hybrid Model to implement the MPI+OpenMP Structure. In section 5, We will show implementation and result of the hybrid structure. Finally, conclusion is drawn in section 6.

II. PURE MPI AND PURE OPENMP

Message Passing Interface (MPI) is a standardized and portable message-passing system designed by a group of researchers from academia and industry to function on a wide variety of parallel computing architectures. The standard defines the syntax and semantics of a core of library routines useful to a wide range of users writing portable message-passing programs in C, C++, and Fortran. MPI is a communication protocol for programming parallel computers. Both point-to-point and collective communication are supported. MPI "is a message-passing application programme interface, together with protocol and semantic specifications for how its features must behave in any implementation." MPI's goals are high performance, scalability, and portability. MPI remains the dominant model used in high-performance computing today. Although MPI belongs in layers 5 and higher of the OSI Reference Model, implementations may cover most layers, with sockets and Transmission Control Protocol (TCP) used in the transport layer. Most MPI implementations consist of a specific set of routines directly callable from C, C++, Fortran (i.e., an API) and any language able to interface with such libraries, including C#, Java or Python.

Open MP is an implementation of multithreading, a method of parallelizing whereby a master *thread* (a series of instructions executed consecutively) *forks* a specified number of slave *threads* and the system divides a task among them. The threads then run concurrently, with the runtime environment allocating threads to different processors. The section of code that is meant to run in parallel is marked accordingly, with a compiler directive that will cause the threads to form before the section is executed. Each thread has an *id* attached to it which can beobtained using a function (called omp_get_thread_num()). The thread id is an integer, and the master thread has an id of *0*. After the execution of the parallelized code, the threads *join* back into the master thread, which continues onward to the end of the program. The runtime environment allocates threads to processors depending on usage, machine load and other factors. The runtime environment can assign the number of threads based on environment variables, or the code can do so using functions. The OpenMP functions are included in a header file labelled omp.h in C/C++.

III. MPI VS OPENMP

MPI and OpenMP structures has some significant potential merits and som limitations which are discussed here. Pure MPI lead us to the Portable to distributed and shared memory mechanism with inter process communications. The MPI structure can be scaled beyond one node. Also we don't required the data placement for the MPI programs. But with this distinct advantages MPI has some limitations. As MPI has explicit communication within the processes it reflects high latency and low bandwidth utilization. it has been very difficult to provide load balancing in MPI. OpenMP is the easiest way to implement parallelism within the node. Because of the intra node

communication, OpenMP provides low latency and high bandwidth for the communication. The best feature that OpenMP provides is the Dynamic load balancing within the threads which makes the communication more faster. The only disadvantage of the OpenMP is it can work only on the shared memory node or the machine and cannot scaled beyond the node.

IV. MPI+OPENMP HYBRID MODEL

As we know computer cluster basics, we can employ both shared and distributed memory architectures, by utilizing this advantage of the computer cluster we can make the combine the use of the distributed and shared memory together. The shared memory component is usually a cache coherent SMP node. Processors on a given SMP node can address that node's memory as global. The distributed memory component is the networking of multiple SMP nodes. SMPs know only about their own memory -not the memory on another SMP. Therefore, network communications are required to move data from one SMP to another SMP. Fig 1 shows the basic approach to combine the MPI and OpenMP structures and the communication process.

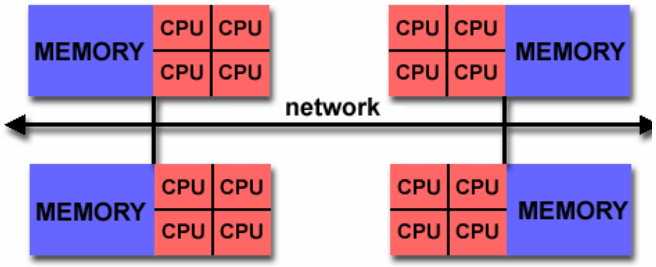

Figure 1. Hybrid Distributed - Shared Memory Model

In the process of converting sequential code, first we will make it parallel with MPI, then OpenMP will be incorporated into it. From MPI code, add OpenMP. From OpenMP code, treat as serial code. Simplest and least error-prone way is to use MPI outside parallel region, and allow only master thread to communicate between MPI tasks. Fig 2 represents exact process of the hybrid programming. Figure show that Rank 0 (Parent Process) is created and it will gather the data coming from the all processes. After the parent more child processes will be created which will perform the parallel programming with thread based on the OpenMP structure.

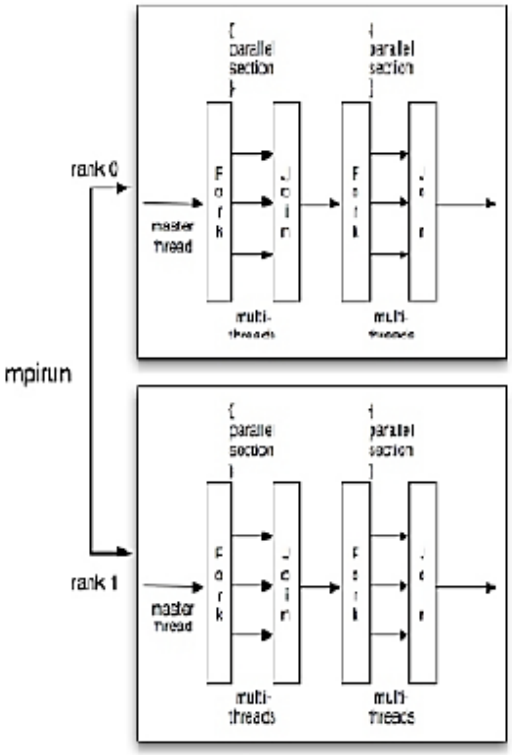

Figure 2. Hybrid Programming Model

To start with the Hybrid Programming Model, Start with MPI initialization, Create OMP parallel regions within MPI task (process), Serial regions are the master thread or MPI task, MPI rank is known to all threads, Call MPI library in serial and parallel regions and in last Finalize MPI. Program structure of the Hybrid Programming Structure is as follow.

Program Hybrid

call MPI_INIT (ierr)

```
call MPI_COMM_RANK (…) // Allocate the rank to the processes
call MPI_COMM_SIZE (…) // defining the size of the processes
… some computation and MPI communication
… start OpenMP within node
!$OMP PARALLEL DO PRIVATE(i) // OpenMP code will be implemented from here
!$OMP& SHARED(n)
do i=1,n
… computation
enddo
!$OMP END PARALLEL DO // OpenMP code will be end here some computation and MPI communication
call MPI_FINALIZE (ierr)
end
```

V. IMPLEMENTATION AND RESULT

A generalised program of Pi calculation is implemented on with pure MPI, with pure OpenMP and the hybrid version of MPI+OpenMP structure. We used CDAC super computer to implement this programs with multiple Cores, Threads and Processes. The programs implemented for 1 cr. steps and 10 cr. calculations.

Program	Pi Cal. (1 cr.)	Pi cal (10 cr.)
MPI 4 Processes	1.470000	2.490000
OMP 4 Threads	6.490000	10.540000
MPI+OMP Hybrid Version	0.170000	0.490000

Figure 3. Result of Pi Calculation Program

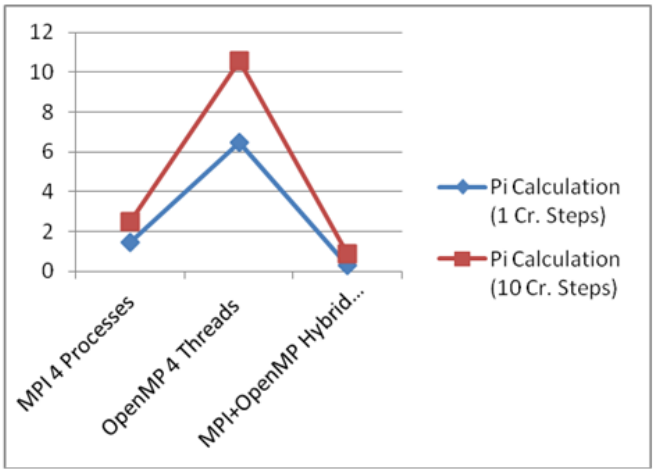

Figure 4. Execution time for the Pi Calculation

VI.CONCLUSION

Comparison of the all three programming model shows the significant advantages of the Hybrid Model over the MPI and OpenMP concept. Result shows that the implementation of both concepts enhance the speed of the program and reduce the delay within the processes. With the hybrid approach the Computer systems is fully utilised with Distributed Memory and Shared Memory system. (multi-core nodes connected via a network). Also we have the advantages of the both the MPI and OpenMP programming model separately. Hybrid code have the advantages of Intra node thread communication and inter process node communication.

As a part of future work many Sorting algorithm, Searching algorithms and iterative programs can be implemented with this hybrid approach.

VII. REFERENCES

[1] Rolf Rabenseifner, Gerhard Welleiny "Communication and Optimization Aspects of Parallel Programming" International Journal of High Performance Computing Applications, 2002. c Sage Publications.

[2] Frank Cappello and Daniel Etiemble, MPI versus MPI+OpenMP on the IBM SP for the NAS bench-marks, in Proc. Supercomputing'00, Dallas, TX

[3] Steve LantzSenior "Hybrid Programmming with MPI and OpenMP" Workshop: Introduction to Parallel Computing on Ranger, May 24, 2011

[4] Jorge Silva, Ana Aguiar, Fernando Silva " Parallel Computing Hybrid Approach for Feature Selection" , 2015 IEEE 18th International Conference on Computational Science and Engineering

[5] Ying Xu; Tie Zhang "A hybrid open MP/MPI parallel computing model design on the SM cluster",2015 6th International Conference on Power Electronics Systems and Applications (PESA)

[6] Rolf Rabenseifner, Georg Hager, Gabriele Jost "Hybrid MPI/OpenMP Parallel Programming on Clusters of Multi-Core SMP Nodes" 1066-6192/09 $25.00 © 2009 IEEE

[7] James Dinan, Pavan Balaji, Ewing Lusk"Hybrid Parallel Programming with MPI andUnified Parallel C" Conference Paper · January 2010