Theoretical Study based Analysis on the Facets of MPI in Parallel Computing

Blessy Boaz
Department of Computer Science,
Stella Maris College, Cathedral Road, Chennai-600086

Abstract: In the Big data era for scientific applications researchers deal with large volumes of data and so ascends the need for parallel programming. High performance computing frameworks increases the scalability, efficiency and the performance of scientific computing. Message Passing Interface (MPI), has been consistently successful for several decades and it adapts the changing needs of parallel programming. This study interprets the success and the challenges of MPI in parallel computing. The technique that is used to overcome the challenges of MPI and some of the hybrid parallel programming models. Finally the re-entry of MPI with SPARK as SPARK+MPI which uses the MPI libraries in Spark, fixing the gap between the Spark ecosystem and the native implementation with High performance computing.

***Keywords**: Parallel programming, SPARK, .Message Passing Interface (MPI), SPARK+MPI, Spark ecosystem.*

*****

I. INTRODUCTION

High performance computing ranges from personal computers to huge parallel processing systems. HPC is based on RISC processors, techniques learnt are transferred to the other systems. Programmers should know about the processors functionaltogether. Larger applications are solved by connecting high-speed network that functions as a network of workstations which is used as a large multicomputer.Problems that are large with added data interactions andexpensive compute budgets; thusneed many processors for scalable parallel processing systems.

In Parallel programming more than one CPU is used to complete a job. Execution order does not matter but the task must be independent. Processors can work on the segments of the problem called functional parallelism or it can work on the segments of data called as data parallelism. Parallel computing increases the performance and memory availability. Data parallelism is obtained more than functional parallelism.

Parallel programming is operationally challenging to achieve. There are some operational difficulties with parallel programming. They are load balancing, efficiency limitations in communication, dominance of serial time. Parallel computers are classified by their memory model: shared memory, distributed memory and hybrid. Shared memory model share the single address space. Distributed memory model is hard since it has multiple address space and it accesses remote data. Hybrid model is extremely difficult, but it offers greater flexibility.

An Overview of MPI

Message passing interface is a parallel programming model using low level programming languages such C and C++. There is no need to learn a new language in order to use MPI.With the existing program, MPI library is accessed at link time. On clusters, nodes have separate memory space which is distributed between processors; to access the distributedmemory; messages are being sent, processed and received over the network.

MPI is implemented on clusters of several computers with the same type and connection. If suppose shared memory machines are available thenOpenMP Parallel programming is used that is several processors share a single physical memory. In parallel programming model every process should initialize MPI in the beginningof parallelization and finalize MPI at the end. MPI communicator communicates with the subsets of processors.SPMD (Single/Shared Program Multiple Data) is the mode for MPI execution.A copy of the executableis given to each process and its rank determines which part of the problem to work on, otherwise it works independently.

II. LITERATURE SURVEY

This literature survey studies about the quick process initiation time, connection management using ACM algorithm over inifiniBand, I/O optimization techniques, scalability issues and the benefits of different hybrid parallel programming models. Process initiation and connection setup are the phases of fast and scalable process startup. The scalability issue in initiation phase is serialized process and in connection phase is high communication overhead. Parallel job launched by the process manager is process initiation and the initiated process requires process manager's assistance for peer-to-peer connections to communicate and perform computation, this is the connection setup phase. InfiniBand is a System Area Network (SAN) for interconnecting computing nodes and I/O nodes. InfiniBand provides Reliable connection (RC) as its transport services.

Fully connected processes are created in the parallel application if it follows the following steps: First step, MVAPICH is launched with a process launcher iterates over UNIX shell to start individual processes. Processes connect back to the launcher via a port. Each process has no global knowledge about the parallel program, except the rank of the process. Second step is the connection setup, each process creates N − 1 QPs, one for each peer process, for an N-process application. These processes interchange their local identifiers (LIDs) and QP identifiers (QP-IDs) for connection setup. Since each process is not connected to its peer processes, the data exchange has to rely on the connections that are created to the launcher in the first phase. Data about both IDs from each process is collected by the launcher, and the combined data is sent back to each process. Each process in turn sets up connections over InfiniBand with the received data. There are scalability bottlenecks, when it comes to large scale parallel applications.
[1] proposes to uses MPD with MPICH to gain efficient process initiation, as it has a large user base. The problem with large scale parallel applications can be solved by reducing the volume of data to communicate and parallelize communication for the exchange of QP IDs. Fast process initiation with MPD improves the scalability by exchanging the Queue Pair IDs over the ring of manager and by setting up Bootstrap channel within processes. Startup time is reduced drastically in this approach.

Adaptive connection management dynamically controls the creation of InfiniBand steadfast connection constructed on the communiqué frequency concerning MPI processes. Two ACM algorithms: on-demand and partial static algorithm have been designed and implemented in MVAPICH. ACM Algorithms used in unreliable datagram-based connection or reliable connection management-based mechanism this is implemented in MVAPICH over inifiniBand to support parallel programs. [2] proposed algorithm is evaluated to lessen the process initiation period, the number of dynamic connections, and the communication resource usage. Investigational outcomes show that the algorithm decreases the typical number of connections per process.

The strengths of shared memory and distributed memory programming models are efficiency, memory savings, ease of programming and scalability.Hybrid model selects a language/library for the instantiation. Multithreaded process per node in OpenMP threads exploits the multiple coresand nodes communicate using MPI. The two portable APIs of hybrid programming has to make some commitments to each other. The threads in a process remain runnable, if it is blocked by system call in case of OpenMP model whereas MPI will only block the calling thread and not the entire process. There are four levels of thread safety in MPI-2 standard; applications at run time will know the level supported by MPI Library. MPI calls are made by the master thread;hybrid programming does not require a full thread safety is the OpenMP style of programming. In order to examine the complete activities of parallel hybrid programs Jumpshottool is used.[3]Performed test on three different hardware/software machine/compiler environments, to demonstrate the practicality of OpenMP+MPI hybrid programming model and the Jumpshot tool. This hybrid programming model is available on the computers for application developers to create applications.

[4] Adds the improvement for collective, one-sided communication, multi-threaded programming, and support of performance analysis tools in MPI-3. MPI-1 the core, two-sided message passing, collective communication and computation is well established effective programming model, is the reason behind its success. MPI datatypes provide several important benefits. MPI communication context builds a way to modular software and an ecosystem of parallel software libraries by ensuring software components that communication was intercepted by the user's code or by another software component. Not all parallel computing models have such precision. The reason for the long-term success of MPI is its precise semantics, programs continue to work and give same results even after two decades of change in computer hardware. Despite its strength MPI has some weaknesses: MPI's specification as library, preventing close integration with the language and lack of support for distributed data structures are its weaknesses.

Myths about MPI: require p^2 buffering for p processes, not fault tolerant, no scalable startup, RMA complex rules, ordering of messages in the network. However these are myths about MPI there is no such requirement for buffering. Each process having its own internal buffer reduces code and complexity, fast scalable approach is a challenge. MPI forum strives to specify certain types of faults. Scalable startup mechanism is more complex to design, build and maintain. MPI RMA strives for precision, like any other one-sided programming model this has simple subsets that a programmer needs and so the rules are complex but complex to use than other programming model is a myth. MPI requires ordering in operations but no need for ordering of data moving over the network. In order to provide ordering transport layer can be used but it is not needed and so it makes efficient use of fast networks for data transfers.

MPI-3 faces greater challenges. Processor architecture changes due to the growing challenges of power consumption and heat dissipation in turn programming models also change rapidly. Remote Direct Memory Access challenges MPI in efficient use of hardware and programming model. Commodity network provided by Infiniband supports RDMA. MPI RMA model enforces shared memory operations. Systems at extreme scale likely to fail. MPI provides both static and dynamic analysis correctness as well as performance correctness.

[5] presents technical applications execution time spend on accessing data from files. In order to improve data access and boost the performance, a technique such as data prefetching and data layout is used but it is complex. Tools are used to streamline the optimization procedure. Many tools exist for I/O characterization and optimization. The proposed tool named IOSIG is to understand the parallel program's I/O behavior and the custom of I/O access information for optimization.

Data prefetching predicts the data that application needs and loads from storage devices to buffers before the application could request for data. If the prefetching method is attentive to the I/O features of an application and then uses it effectively it can improve the I/O performance. Next optimization technique is the data layout method; the three most popular methods are 1-DH, 1-DV and 2-D. First method is called as simple striping dispenses data across all storage devices. Second one is to store the data accessed by each I/O client process on one storage node. Third is the data accessed by each process is stored on a storage group. For the optimal performance of parallel I/O systems the knowledge of an applications data access patterns should be known.

IOSIG software's components are trace collector and trace analyser. The MPI-IO library is used in HPC applications for basic file operations. IOSIG trace collector captures MPI-IO calls and records their information. The Profiling MPI interface is used by the trace collector to capture the MPI-IO routines. Trace analyser uses the local and global access patterns and I/O signatures. The components of IOSIG works only at application level are an advantage for users without any privileges in large-scale machine. The software evaluation proves that IOSIG keeps the overhead at a minimal level with common resource requirements.

MPI in trans-petascale and exascale systems also addresses the scaling issues.In order to indicate the quantities indexed by rank, efficient data structures were developed in MPI implementation, sparse arrays and hash tables reduce the scalability issue. Next is the communication overhead issue, the algorithms for irregular all-gather operations and pipelining avoids delay by the long and short messages then there are hybrid collective algorithms for network congestion. [6] Involvedinthe new RMA model includes atomic remote memory operations, dynamic access to remote process memory and the memory ordering semantics used by applications; this is part of MPI-3 standard. Adaptive and dynamic algorithms were developed to provide good performance for latency and bandwidth-dominated communication, this is integrated into MPICH. Light weight thread and fine grain multithreading was supported by MPI. The enhancement of implementations of the MPI topology functions give the computational scientist a portable way to map an MPI program onto large parallel machine provides way for efficient communication. MPI datatypes can be used when memory motion is relatively expensive. Coverage analysis tool was developed for MPICH, which helps to identify the issues in MPICH. Hybrid programming model can also mix MPI with UPC or CoArray Fortran.

MPI parallel computing cluster communicate on public networks. [7] talks about the security over the network, to handle this Digital Signature Hashing Algorithm was developed. DSHA preserves communicated messages confidentiality. DSHA Algorithm is integrated with MPICH2 library. In MPICH2 there are four different layers: MPI-2, ADI-3, CH3 and the low level interfaces makes the scheme portable and flexible. When MPI sends and receives messages, message snooping could happen and to address this issue. Cryptographic algorithms are implemented in the TCP socket station CH3 layer of MPICH2. Programmers depend on exterior libraries to implement secure MPI applications. A block of message is taken to condense the data using hashing function and it is encrypting using the private key. DSHA algorithm works well for files with dynamic sized messages.

Distributed systems memory can be pooled on a global level by using PGAS approach, which simplifies nodes data exchange in parallel application development. [8] Presents a runtime environment (DART) implements the PGAS paradigm on big scales high-performance computing collections. UPC is a fully implemented PGAS language, besides the languages approaches can also be used in form of API or library. DART (DASH Runtime) establishes PGAS and handles allocation of memory and data movement. MPI based implementation of DART efficiency is measured in terms of Data Transfer Completion Time (DTCT) and the Data Transfer Initiation Time (DTIT) for blocking and non-blocking put/get calls. Performance of DART is compared to pure MPI as overheads are insignificant.

[9]presents a hybrid parallel program writingusing MPI and Charm++. A module for parallel application can be developed by programmers with these languages however it facilitates smooth interoperation. The challengesare enabling

interoperation and skills for handling the control flow and resourcesharing. To share resources in various situations it is important to understand the phases and modules in an application. Modules can either be time or space division in different phases of the application, in a hybrid division of resources. The data to be exchanged between the modules is local to each process, and memory pointer sharing is straightforward. The user has to set up a data transfer storehouse for interchange of data across components. It is evident that enabling interoperation can bring the prime of both worlds together for achieving good performance, high programmer productivity, and code reuse.

In Data analytics a prevalent framework called Apache Spark is being used.Fault tolerance and interoperability are the features of data analytics. Analytics operations in Spark are much slower than the native implementations. [10] proposes a system for integrating MPI with Spark. There is a huge gap in the efficacy of Spark and MPI. The nature of the breach is enlightened by optimizingand profiling Spark to well understand its performance.Spark+MPIchannels the breach, it is demonstratedand evaluated on real applications.In profiling Spark, it is observed that the Spark implementationsconsume a large amount of CPU resources and time spent in algorithm than Spark+MPI implementations. Spark implementations are making inefficientuse of CPU resources. For algorithmswith less iteration, it maybe efficient to stay in Spark to avoid the overheads ofSpark+MPI. MPI would be advantageous when finding larger chunks of work to do. Memory should be partitioned between Spark and MPI. Spark+MPIis to be release as an open-source software.

III. CONCLUSION

MPD with MPICH reduces Process startup time. ACM Algorithms in reliable connection management also lessen the start-up time, number of connections and the resource usage. Hybrid programming model with MPI and OpenMP combines its advantages and can be used as an application by developers. The performance degradation due to the time spend on data access can be minimised using optimization procedures. Other hybrid programming models like MPI and Charm++ can be used to achieve good performance and productivity. Spark and MPI is one another framework to lessenthe amount of resources used and attain good performance. Future work is to analyse the performance of a sorting algorithm on these parallel programming and hybrid models with the I/O optimization and other techniques used to improve the performance of the system.

REFERENCES

[1] Weikuan Yu, Jiesheng Wu, and Dhabaleswar K. Panda, “Fast and Scalable Startup of MPI Programs in InfiniBand Clusters” Springer-Verlag Berlin Heidelberg, 2004

[2] WeikuanYu ; Qi Gao ; D.K. Panda, “Adaptive Connection Management for Scalable MPI over InfiniBand” IEEE, 2006

[3] Ewing Lusk and Anthony Chan, “Early Experiments with the OpenMP/MPI Hybrid Programming Model” Springer-Verlag Berlin Heidelberg, 2008

[4] William Gropp, “MPI 3 and Beyond: Why MPI Is Successful and What Challenges It Faces” Springer-Verlag Berlin Heidelberg, 2012

[5] YanlongYin ;SurendraByna, “Boosting Application-specific parallel I/O optimization using IOSIG” 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, 2012

[6] Gropp, William Douglas, ”Enhancing the MPI Programming Model for PetaScale Systems”United States: N. p., 2013.

[7] M.Nagalakshmi, A.VeerabhadraRao, “A secured DSH Algorithm for MPI in Parallel Computing” IJCST Vol. 4, Issue 4, Oct - Dec 2013

[8] Huan Zhou, YousriMhedheb, “DART-MPI: An MPI-based Implementation of a PGAS Runtime System” eprint arXiv:1507.01773, 2015

[9] N Jain, A Bhatele,“Charm++ & MPI: Combining the Best of Both Worlds” IEEE, 2015

[10] Michael Anderson, ShadenSmith,”Bridging the Gap Between HPC and Big Data Frameworks” Journal Proceedings of the VLDB Endowment, 2017