Journal of Advanced Computing Research Vol. 4, Issue 2 (2019) 10-12

Parallel Computing - A Case Study on MPI Application Programming Interface

Nur Adilah Zulkiflee [#1], Mohamed Faidz Mohamed Said [#2]

[#] *Faculty of Computer & Mathematical Sciences, Universiti Teknologi MARA*
70300 Seremban, Negeri Sembilan, MALAYSIA

[1] adilahzulkiflee@gmail.com

[2] faidzms@ieee.org

***Abstract*—Message passing interface (MPI) has been normally recognized as the best interface option for passing message in parallel computing backgrounds. MPI is a very general de-facto parallel programming application programming interface (API) for scattered memory systems. It is sometimes uncertain to decide the necessary network architecture because the passing interface mainly caters its location-based processes by addressing the required transport method to be applied. Overall this research paper would focus on the definition and the application of the passing interface. The objectives of this research are to study the MPI applications in the discipline of parallel computing in real world. The result shows that the passing interface has several advantages and is suitably used in parallel computing. Moreover, it has been applied for python and big data. However, on the other side, MPI has some disadvantages and these include being easy to make mistakes and hard to debug. For recommendation, it would be beneficial to experiment more specific studies on the performance sensitivity issues of the applications based on its memory latency as well as its parameters.**

***Keyword*: message passing interface, parallel computing**

I. Introduction

Application programming interface (API) is a typically defined technique of communication between numerous software mechanisms. It may be used for a web-based system, operating system, database system and computer hardware. API is a significant part of current programming specifically in parallel programming. They permit important new functionality such as communication and harmonization to be delivered to programmers without altering the original programming language [1].

Message passing interface (MPI) is a message passing library standard based on the agreement of the MPI Forum. According to [2], MPI has been extensively recognized as the message passing interface of choice in parallel computing situations. This passing interface encompasses the description of the syntax and semantics of library procedures and permits users to write portable programs in the main technical programming languages [3]. MPI has its own data types and it supports C, C++ and Fortran. In addition, MPI setting usually contains of at least a library for applying the API, a compiler and linker that support the library as well as a run time situation to launch a MPI program.

The aim of MPI is to create a convenient, effective and flexible standard for message passing that will be extensively used for writing message passing program accordingly (Fig. 1). It also allows users to control the passing of data between processes through well-defined subroutines. Next, the passing interface is sometimes indeterminate in the setup of the appropriate network architecture because it largely maintains the relevant run location by addressing the whatever transport method that are being used.

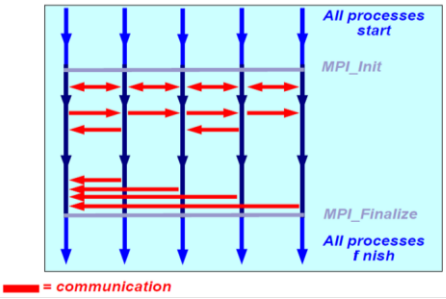

Fig. 1. Example of the MPI execution model

There are several advantages of MPI in parallel computing. This passing interface is a popular programming model. Moreover, it is flexible, straightforward and typically available. However, there are also disadvantages of this passing interface in parallel programming. These include the redesign of application, being easy to make mistakes and it is hard to debug.

II. Literature Review

Since its introduction, the MPI requirement has become the important standard for message-passing libraries in the world of parallel computers. In [3], the paper discusses about MPI for Fortran. The author makes a review of the passing interface abilities and the new presented structures for Python to improve communication presentation and good support traditional MPI-1 operations in a Python programming situation. MPI for Python was upgraded to support direct communication of anything transferring the single-segment buffer interface. This interface is a typical Python instrument provided by certain types permitting entree in the C side to a contiguous memory buffer containing the relevant data. Besides that, paper [4] states that Python and MPI scatterings can be optionally constructed as shared libraries in current operating systems supporting dynamic networking. Then, the Python translator can be simply permitted to run scripts in

simultaneous and support extension modules calling MPI functions.

Next, paper [2] focuses on the hard work to improve interfaces for programming parallel computing properties in certain message passing interface. This paper also studies how MPI raises out of the desires of the technical research group through a wide based review procedure. The development of the passing interface is contrasted with other similar standardization efforts. Anyway, Open MPI is one of the most popular applications and it is also the major programming pattern for parallel applications on scattered memory computers [5]. Java also is added to support Open MPI, revealing MPI functionality to Java programmers.

In [1], the authors explain about the first official condition of a non-trivial division of MPI, the main communication API in high presentation computing. MPI programs are often manually or automatically rearranged when ported to a different hardware stand, for instance by altering its original functions to specialized versions. Meanwhile, the author [6] compares the numerical difference between serial and MPI parallel computations and it is shown that MPI parallel computation clearly can give rise to a different passing solution from the serial computations.

III. Methodology

The first editions of MPI were made to work competently on multiprocessors which had very small work control and thus fixed progression models. The version is continuously pushing them to maintain a dynamic process model appropriate for use on groups or scattered systems. This process model would have influence on their performance. Therefore, the researcher [7] introduces a new application of the passing interface called FT-MPI that permits the semantics and related kinds of setbacks to be clearly controlled by an application via an altered MPI API (Fig. 2). In addition, a lot of simultaneous applications are written using MPI. However, the passing interface does not offer clear support for relocation. This process of relocation contains the transmitting of a process from one system to another during its implementation. Therefore, the author [8] proposes a solution that decreases the memory and I/O overhead in an application stage checkpoint-based relocation method.

```
rc = MPI_Bcast ( initial_work…);
if(rc==MPI_ERR_OTHER)reclaim_lost_work(…);
  while ( ! all_work_done) {
  if (work_allocated) {
    rc = MPI_Recv ( buf, ans_size, result_dt,
                    MPI_ANY_SOURCE,   MPI_ANY_TAG, comm, &status);
    if (rc==MPI_SUCCESS) {
                         handle_work (buf);
                         free_worker (status.MPI_SOURCE);
                         all_work_done--;
                         }
    else {
           reclaim_lost_work(status.MPI_SOURCE);
         if (no_surviving_workers) { /* ! do something ! */ }
         }
  } /* work allocated */
/* Get a new worker as we must have received a result or a death */
  rank=get_free_worker_and_allocate_work();
  if (rank) {
    rc = MPI_Send (… rank… );
    if (rc==MPI_OTHER_ERR) reclaim_lost_work (rank);
    if (no_surviving_workers) { /* ! do something ! */ }
  } /* if free worker */
} /* while work to do */
```

Fig. 2. Example of the FT-MPI master-worker code

Meanwhile, the researchers [9] write about a new technique called MPI Communication Management (MCM) for cloud situations. MCM is a communications management technique for MPI, based in the study of communication expectancies between processes. This technique describes the essential network topology and analyses parallel applications behaviour in the cloud which improve the application's messages latency time.

Besides that, the authors [10] describe a research of the hybrid MPI and OpenMP programming method functional to two pseudo application standards and two real life applications, and established advantages of the hybrid method for performance and reserve custom on three multicore-based simultaneous systems. As the latest high performance computing systems that are looking toward petascale and exascale, percore resources, for example, memory are predictable to become smaller. In addition, the authors [11] state that the study of hybrid MPI and threads is getting severe attention as are parallel mesh-based replication techniques. They believe, since the on-node performance of a hybrid system is better than its internode performance, designers will need to transform software to gain benefit of shared memory and other locality. However, such alteration can now occur more steadily after the initial port, and at a higher level than the specific thread organization.

The growing dimension of computational groups results in a rising possibility of disappointments, which in turn needs application check pointing in order to survive those failures. Original check pointing needs files to be copied from application memory into steady storage intermediate, which increases application completing time as it is commonly done in a separate step [12]. In MPI situations, difficulty leads to different communication procedures for different message sizes. The classification of these different behaviours is significant and valuable for software developers and network designers. Therefore, the researchers [13] introduce an automatic technique to obtain a classification of the communication behaviour of a particular MPI situation using LogP-based models. This technique automatically identifies the message sizes where the communication behaviour changes.

Meanwhile, the authors [14] present a runtime system for old MPI programs that allows the competent and clear out of core performance of scattered memory for simultaneous programs. The system called Big Data MPI (BDMPI), controls the semantics of MPI's API to organize the implementation of a huge number of MPI methods on much fewer compute nodes. BDMPI allows the effective improvement out of the core parallel scattered memory codes without a high engineering and algorithmic difficulties connected to several stages of blocking.

IV. Conclusions

In conclusion, the primary advantages of MPI is flexible and straightforward. This passing interface has been increasingly and widely used in numerous parallel computing platforms. Furthermore, the passing interface has been utilized in python and big data programs. Besides that, the passing interface background normally contains of at least a library applying the API, a compiler and linker that support the library and a run time situation to launch a MPI program. However, MPI also has some disadvantages such as easy to make mistakes and it is hard to debug. For recommendation, it would be beneficial to analyse more specific studies on the performance sensitivity of applications in terms of its memory latency and associated parameters.

References

[1] G. Li, R. Palmer, M. DeLisi, G. Gopalakrishnan, and R. M. Kirby, "Formal specification of MPI 2.0: Case study in specifying a practical concurrent programming API," *Scienceof computer programming*, pp. 65-81, 2011.

[2] R. Hempel and D. W. Walker, "The emergence of the MPI message passing standard for parallel computing," *Computer Standard & Interfaces*, vol. 21, pp. 51-62, 1999.

[3] L. Dalcín, R. Paz, M. Storti, and J. D'Elía, "MPI for Python: Performance improvements and MPI-2 extensions," *Journal of parallel and distributed computing*, pp. 655-662, 2007.

[4] L. Dalcín, R. Paz, and M. Storti, "MPI for Python " *Journal of Parallel and Distributed Computing*, vol. 65, pp. 1108-1115, 2005.

[5] O. Vega-Gisbert, J. E. Roman, and J. M. Squyres, "Design and implementation of Java bindings in Open MPI," *Parallel Computing*, vol. 59, pp. 1-20, 2016.

[6] S. B. Lee, "Numerical discrepancy between serial and MPI parallel computations," *International Journal of Naval Architecture and Ocean Engineering*, vol. 8, pp. 434-441, 2016.

[7] G. E. Fagg and J. J. Dongarra, "HARNESS fault tolerant MPI design, usage and performance issues," *Future generation computer system*, pp. 1127-1142, 2002.

[8] I. Cores, M. Rodriguez, P. Gonzalez, and M. J. Martín, "Reducing the overhead of an MPI application-level migration approach," *Parallel Computing*, vol. 54, pp. 72-82, 2016.

[9] L. Espinola, D. Franco, and E. Luque, "MCM: A new MPI Communication Management for Cloud Situation," *PROCEDIA - Computer Science*, no. 2303-2307, 2017.

[10] H. Jin, D. Jespersen, P. Mehrotra, R. Biswas, L. Huang, and B. Chapman, "High performance computing using MPI and OpenMP on multi-core," *Parallel Computing*, vol. 37, pp. 562-575, 2011.

[11] D. Ibanez, I. Dunn, and M. S.Shephard, "Hybrid MPI-thread parallelization of adaptive mesh operations," *Parallel Computing*, vol. 52, pp. 133-143, 2016.

[12] P. Dorozynski *et al.*, "Checkpointing of Parallel MPI Applications using MPI One-sided API with Support for Byte-addressable Non-volatile RAM," *Procedia Computer Science*, vol. 80, pp. 30-40, 2016.

[13] D. R. Martıneza, V. Blancob, J. C. Cabaleiroa, and F. F. R. T. F. Penaa, "Automatic Parameter Assessment of LogP-based Communication Models in MPI Situations," *procedia Computer Science*, vol. 1, pp. 2155-2164, 2012.

[14] D. LaSalle and G. Karypis, "MPI for Big Data New tricks for an old dog," *Parallel Computing*, vol. 40, pp. 754-767, 2014.

[15] N. A. Zulkiflee. (2017). Parallel Computing – A Case Study On MPI API. Available: http://youtube.com/watch?v=u-DYD2cLimA. [Accessed: 27-Nov-2017]