Parallel Programming using MPI - A Case Study on Hello World

Amira Adila bt Abdul Manab [#1], Mohamed Faidz Mohamed Said [#2]

[#] *Faculty of Computer and Mathematical Sciences, Universiti Teknologi MARA*

70300 Seremban, Negeri Sembilan, MALAYSIA

[1] amiraadila220195@gmail.com

[2] faidzms@ieee.org

***Abstract*—All modern information processing systems are equipped with multicore processors, and the majority of them also have graphics cards for carrying out vector calculations. Every aspiring coder should know the techniques of latitude and distributed computer programmes. This paper presents recent trends in programming and applications using central and whole graphics processing. Several working of the MPI standards are shown, including the open source LAM/MPI and MPICH implementations as well as Sun MPI, an example of a vendor-supplied MPI implementation. Different aspects and perspectives are investigated, such as supported MPI features, system architecture, network hardware, and operating system. Graph-oriented programming (GOP), a high-level abstraction for message passing applications based on MPI, are also included as a part of development research especially on the low-level approach taken by MPI. The paper concludes with an outlook on the future of the MPI standard and its implementations, and how they are influenced by recent trends in cluster computing.**

Keywords: **parallel programming, MPI, Hello World**

I. INTRODUCTION

During recent years, high execution computing has come to be a low-priced rather to many extra researchers in the medical network than ever before. The alignment of quality open source software program package deal and commodity ironware strongly influenced the present enormous reputation of Beowulf type cluster.

Among many parallel computational fashions, message-passing has demonstrated to be a powerful one. This paradigm is especially proper for allotted retentively architectures and is used in these days' engineering application related to model, simulation, layout, and sign processing. But, transportable message-passing parallel scheduling was incompatible in the past due to the numerous mismatched alternatives faced by developers.

Luckily, this issue really changed after the MPI discussion board released its popular specification. High performance computing is traditionally related to software program, but normally only a small part of the code is crucially sufficient to require the performance of compiled languages. The relaxation of the code is typically associated with memory control, erroneousness coping with, enter/output, and consumer interaction. Interpreted high-storey languages may be genuinely tremendous for this shape of exertions.

For enforcing general-intention numerical calculation, Matlab is the dominant interpreted programming language in the open source side. Musical octave and Scilab are well known, freely distributed software imparting compatibility with the Matlab language. Nowadays, MPI for Python, a brand new bundle package is used to take advantage of a couple of processors by using popular MPI "appearance and tactile assets" in Python handwriting.

Table1. MPI Communicator (MPI_Comm, MPI_COM_WORLD)

C Function Call	Function Purpose
int MPI_Init(int *argc, char **argv)	Initialize MPI
int MPI_Comm_size(MPI_Comm comm, int *size)	Determine number of processes within a communicator
int MPI_Comm_rank(MPI_Comm comm, int *rank)	Determine processor rank within a communicator
int MPI_Finalize()	Exit MPI (must be called last by all processors)
int MPI_Send (void *buf,int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm)	Send a message
int MPI_Recv (void *buf,int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Status *status)	Receive a message

- MPI Definition

Message Passing Interface (MPI) is a standardized and portable message-passing system designed by a group of researchers from academia and industry to deal with a wide diversity of parallel computing architectures. A program named Hello World is one of the most basic MPI computer programs being used.

II. HISTORY

The MPI attempt started within the summer of 1991 when a small group of researchers began discussions at a mountain

retreat in Austria. As a result of the dialogue, a Workshop on requirements for Message Passing in a dispensed memory environment was held on 29-30 April 1992 in Williamsburg, Virginia.

Attendees at Williamsburg mentioned the simple capabilities critical to a popular message-passing interface and mounted a running group to retain the standardization process. Jack Dongarra, Tony Hey, and David W. Walker put forward an initial draft concept, "MPI1" in November 1992. A formal meeting of the MPI operating organization was conducted in Minneapolis. Then, the MPI working organization met each six weeks at some stage in the first nine months of 1993. The draft MPI was presented at the Supercomputing '93 conference in November 1993. After a length of public remarks, which led to some adjustments in MPI version 1.0 of MPI was released in June 1994. These meetings and the email dialogue collectively constituted the MPI discussion board, and the membership has been open to all individuals of the high-performance-computing network.

III. REVIEW PAPER

There are numerous papers that have been reviewed to complete this research.

A. A Beowulf Cluster for Teaching and Learning

This paper aims to explore the effects of parallel computing on some programs in a Linux based Beowulf cluster. The research project analyses the performance of some selected parallel programs on the cluster in an effort to provide a parallel computing system for the practical study of parallel and distributed computing [1].

B. Communicating across parallel message-passing environments

The current implementation helps manner conversation among PVM, MPI, and PARIX. With handiest marginal extra attempt, the interface may be tailored to support other message-passing environments as properly. In this paper, it shows PLUS, a lightweight, extensible and efficient interface for the conversation among parallel message-passing models [2].

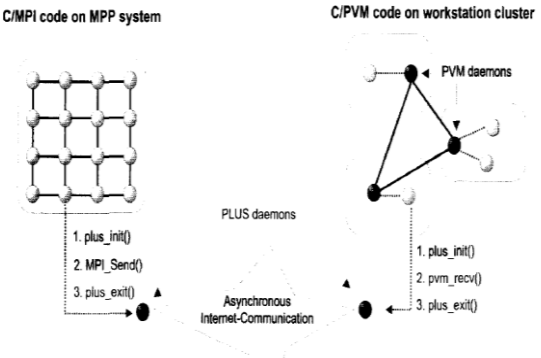

Fig. 1. Process communication with PLUS. plus_init ( ) gets the host names of the PLUS master daemons running in the corresponding (local) PLUS domains

C. Charliecloud: Unprivileged containers for user-defined software stacks in HPC

This project is to provide those offerings in a usable way while minimizing the risks: security, assist burden, missing functionality and overall performance. Charliecloud is presented which uses the Linux and mount namespaces to run industry-trendy Docker containers with no privileged operations or daemons on middle sources.

```
#define _GNU_SOURCE
#include <fcntl.h>
#include <sched.h>
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main(void)
{
uid_t euid = geteuid();
int fd;
printf("outside userns , uid=%d\n", euid);
unshare(CLONE_NEWUSER);
fd = open("/proc/self/uid_map", O_WRONLY);
dprintf(fd, "0 %d 1\n", euid);
close(fd);
printf("in userns , uid=%d\n", geteuid());
execlp("/bin/bash", "bash", NULL);
```

Fig. 2. Hello world implementation of a user namespace, available as examples/userns-hello.c in the Charliecloud source code [1]

This program shown in Fig. 2 creates the namespace with unshare(2), maps within-namespace UID 0 to the invoking user's EUID by writing uid_map, and then starts the root shell [3].

D. ePython: An implementation of Python for the many-core Epiphany coprocessor

In this paper, it presents the work on ePython, a subset of Python for the Epiphany and similar many-core co-processors. The end result of this work is to help developing Python at the Epiphany, which may be implemented to different similar architectures, that the community have already commenced to undertake and use to discover concepts of parallelism and HPC [4].

```
1 from parallel import ∗
23
print "Hello world from core "+ str ( coreid ( )
)+" of "+ str ( numcores ( ) )
```

The above listing illustrates a simple hello world Python code where each Epiphany core will display the message with its core ID and total number of Epiphany cores.

E. Creating Java to Native Code Interfaces with Janet Extension

As Java becomes the perfect environment for excessive performance computing, the interest arises in combining it with present code written in other languages. Portable Java interfaces to local code can be advanced in the use of the Java native Interface (JNI). The paper presents Janet - a particularly expressive Java language extension and preprocessing device that allows convenient integration of native code with Java packages [5].

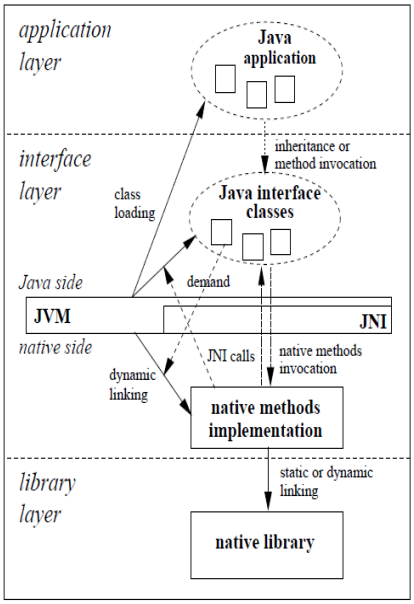

Fig. 3. Use of a native library in a Java application

F. MPI for Python

This report describes the MPI for Python package deal. MPI for Python gives bindings of the Message Passing Interface, well known for the Python programming language, allowing any Python program to take advantage of more than one processor [6].

```
// file: helloworld.i
%module hello world
%{
#include <mpi.h>
#include "helloworld.c"
}%
%include mpi4py/mpi4py.i
%mpi4py_typemap(Comm, MPI_Comm);
void sayhello(MPI_Comm comm);
```

Fig 4. SWIG interface file

MPI for Python provides an object oriented approach to message passing which grounds on the standard MPI-2 C++ bindings. The interface was designed with focus in translating MPI syntax and semantics of standard MPI-2 bindings for C++ to Python. Any user of the standard C/C++ MPI bindings should be able to use this module without the need of learning a new interface.

G. MYMPI - MPI programming in Python

In this paper, the discussion is on the incentive for creating the MYMPI module, along with differences between MYMPI and pyMPI, another MPI Python representative [7].

A Python MPI version of "Hello world" could be written as:

```
#!/usr/bin/env python
from mpi import *
import sys
# initialize mpi
# the initialization routine requires the
# command line arguments held in sys.argv
sys.argv = mpi_init(len(sys.argv),sys.argv)
# get the total number of processes in this parallel
# job and "myid", the identifier for each process
myid=mpi_comm_rank(MPI_COMM_WORLD)
numprocs=mpi_comm_size(MPI_COMM_WORLD)
# each process will print its identifier and the total number
# of processes
print "Hello from ",myid, "The total # of processors is ",numprocs
mpi_finalize()
If this job is run on 2 processors the output might be:
Hello form 0 The total # of processors is 2
Hello form 1 The total # of processors is 2
```

Fig. 5. Python MPI version of "Hello world"

The researchers developed a Python module MYMPI for creating parallel application using MPI. It can be used on a potpourri of parallel platforms and using different MPI libraries. It has been used in some applications and areas including biology, astronomy and geoscience. Future continuous characteristic will be added to the module as needed.

H. Implementation of parallel NetCDF in the ParFlow hydrological model: A code modernisation effort as part of a big data handling strategy.

- Implementation of a MPI-parallel data society to only one data flow per node (with several MPI tasks per node)
- Replacement of the ParFlow binary output module with a NetCDF4 I/O module
- ParFlow I/O optimisation with the profiling equipment
- Density of NetCDF output
- Execution of in-situ processing in ParFlow using VisIt on JURECA to reduce total processing time and type output data volumes [8]

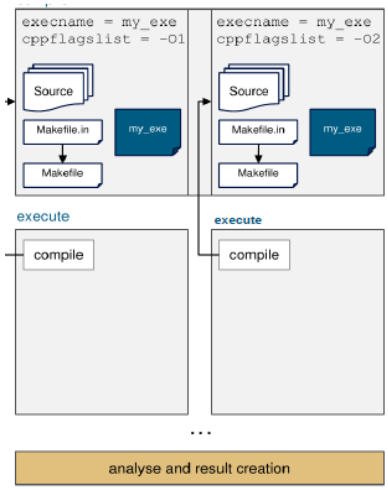

Fig. 6. JUBE2 "hello world" example and benchmark directory preservation. Every rectangle on the right side of the Fig. represents a subdirectory. For every parameter set permutation and the total amount of steps subdirectories are created that "auto-document" stderr, stdout and the explicit parameter set for the current step or test [8].

I. Improving ease of use in BLACS and PBLAS with Python

Researchers and engineers have to spend a considerable amount of term efficiently in developing and discharging codes in the environments. Developer would rather devote the time at using their computational applications. To alleviate this, they promote the reuse of robust software program library like the ones in the ACTS Collecting and the paper presents the work in a subset of the high-level language port, PyACTS [9]. It helps users prototype their codes using these libraries. Lastly, comparison is made between traditional programming practices to the proposed approach. It illustrates some examples of these interfaces and their performance. Additionally, the researchers also evaluate not only their performance but also how user friendlier they are compared to the original call.

```
% mpirun -np 3 pyMPI
>>> import mpi
>>> print 'Hello world',mpi.rank
Hello world 0
Hello world 2
Hello world 1
>>>
```

PyMPI works by building an exchange startup executable for Python, and by using all the installed base of Python code modules, which in turn enables PyMPI to use the same Python modules and rich functionality.

J. Architectural Skeletons: The Re-Usable Building-Blocks for Parallel Applications

The paper defines a model for recognizing and using parallel design patterns. The model provides many functionalities of MPI, plus the reimbursement of the patterns [10]. The following example illustrates a singleton module, which does nothing in excess of printing the drawstring Hello World. Being a single process entity, a singleton has no children. Rep is the illustration. When the representative codification is not filled, it simply becomes a singleton machine. The example uses the current specification language.

```
// My simple sequential program.
MyModule EXTENDS SingletonSkeleton
{
Rep {
printf ("Hello World\n");
}
}
```

To the best of the cognition, this skeleton in the cupboard--based approaching is the first of its kind that aims at providing a monetary standard model for a parallel computing pattern.

K. Adaptive parallel computing on heterogeneous networks with mpC

The paper presents a new advanced version of the mpC parallel language. The language was designed particularly for programming high-performance parallel computations on heterogeneous networks of computers [11].

Implementation of the function MPC_Printf by a process consists of transferring the message "Hello, world!" to the user's terminal from which the whole parallel program has been started up. Thus, the user will see N messages "Hello, world!" on this node - just one from each occupied process.

L. A high performance Java middleware for general purpose computing and capacity planning

In this paper, it presents Java Cá&Lá or simply JCL, a dispensed shared memory lightweight middleware for Java builders that separates business common sense from distribution troubles at some point of the improvement method [12]. It gathers numerous features provided one at a time inside the ultimate many years of middleware literature, allowing constructing disbursed or parallel packages with few transportable instructions and capable to run over extraordinary platforms, which include small ones. This paper describes JCL's features, compares and contrasts. It also shows JCL to different Java middleware systems, and reviews overall performance measurements of JCL programs in numerous wonderful eventualities.

```
1 public class HelloWorld {
2
3 public void print ( ) {
4 // Prints "Hello World!" in the console.
5 System.out.println ("Hello World!");
6 }
7 }
```

Fig. 7. JCL to different Java middleware systems - version of "Hello world"

Fig. 7 illustrates how JCL introduces distribution to an existing chronological code. At line four, the developer gets an example of the JCL and at line five the class "Hello World" is registered, so it becomes noticeable to the whole JCL cluster. At line six, the developer wishes a single execution of the technique "print" of the registered class. JCL "execute" technique requires the class nickname "Hello", the method to be executed "print" and the opinion of such a method or null if no arguments are required.

M. Overview of the MPI Standard and Implementations

This document first introduces the underlying paradigm, message passing, and explores a number of the challenges explicit message passing poses for developing parallel programs. The document concludes with an outlook at the future of the MPI well known and its implementations, and how they may be influenced with the aid of current tendencies in cluster computing [13].

The "Hello World" program is designed to run as techniques, process zero and manner. Every method makes use of the equal program, which then takes distinctive paths in line with the rank of the process (rank could be defined rapidly, for now simply consider it as a procedure identity). The rank is decided in line nine, after that, the if-statement makes the 2 tactics take exceptional execution paths: manner 0 sends a message "Hello World" to manner 1 (line 12), at

the same time as procedure 1 gets this message (line 14) and prints it on general out.

IV. METHODOLOGY

The following assumes that programmers are using MPICH on a cluster of machines discharge some variance of UNIX for which there is admission to all or some of the nodes via the mpirun order. It is also assumed that the nodes are executing the commands to compile, copy, and run the codes from a command layer, or in a terminal window.

Typically, running MPI codes will consist of three steps which are compile, copy and execute. For the compile, it is assumed that a user has computer code to compile in order to create an executable. This involves compilation of the codes with the appropriate compiling program, linked to the MPI program library. It is possible to pass all options through a standard f77 bidding, but MPICH provides support that appropriately links against the MPI libraries and sets the appropriate include and depository library way of life. After saving the above example file, a user can compile the program using the mpicc command. In order for the programme to run on each node, the executable must exist on each node. There are as many ways to ensure that the executable exists on all of the nodes as there are many options to put the cluster together in the first place. Only one method is shown to be used with the BCCD bootable cluster CD. This method will assume that an accounting (bccd) exists on all automobile with the same place directory (/home/bccd), that assay-mark is being done via ssh, and that public keystone have been shared for the account to allow for login and remote control execution of instrument without a password.

V. CONCLUSION

Conclusively, it is not possible to present all the prospective actions of multiprocessor computers to use or for programming computers in a mesh which works at a common task. Nevertheless, OpenMP and MPI are standards for parallel and distributed programming. In the era of multicore computers and computer meshing, these standards are essential for instruction programming. Typical course usually begins with POSIX yarn in C and some popular C++ threads libraries like Hike, which should be usually enough for basic teaching. Next step is to use multithreads in other popular languages like Java. Present architecture tends to integrate shared memory machines into a cluster. In that way, the clusters use heterogeneous calculation on a mixture OpenMP and MPI. These proficiencies are used not only for sophisticated scientific computing but in usual auction and shopping websites like eBay or Allegro mentioned in the listing of peak 500 supercomputing centres.

REFERENCES

[1] R. Priedhorsky and T. C. Randles, "Charliecloud: Unprivileged containers for user-defined software stacks," 2016.

[2] A. L. B. Almeida, "A high performance Java middleware for general purpose computing and capacity planning," 2016.

[3] M. Panczyk, "Improving computation efficiency by parallel programming," *Актуальні проблеми економіки,* no. 3, pp. 398-406, 2013.

[4] H. Xiong, D. Zhang, C. Martyniuk, V. Trudeau, and X. Xia, "Using Generalized Procrustes Analysis (GPA) for normalization of cDNA microarray data," *BMC Bioinformatics,* vol. 9, 2008.

[5] M. Åstrand, P. Mostad, and M. Rudemo, "Empirical Bayes models for multiple probe type microarrays at the probe level," *BMC Bioinformatics,* vol. 9, 2008.

[6] J. Hill *et al.*, "SPRINT: A new parallel framework for R," *BMC Bioinformatics,* journal article vol. 9, no. 1, p. 558, December 29 2008.

[7] S. Calza, D. Valentini, and Y. Pawitan, "Normalization of oligonucleotide arrays based on the least-variant set of genes," *BMC Bioinformatics,* vol. 9, 2008.

[8] A. Brazma *et al.*, "ArrayExpress – a public repository for microarray gene expression data at the EBI," *Nucl Acids Res,* vol. 31, 2003.

[9] G. Vera, R. Jansen, and R. Suppi, "R/parallel – speeding up bioinformatics analysis with R," *BMC Bioinformatics,* vol. 9, 2008.

[10] H. Schwender and K. Ickstadt, "Empirical Bayes analysis of single nucleotide polymorphisms," *BMC Bioinformatics,* vol. 9, 2008

[11] C. Zee, "Overview of the MPI standard and Implementations," *Universität Stuttgart, Alemania,* 2004.

[12] M. Dunning, N. Barbosa-Morais, A. Lynch, S. Tavaré, and M. Ritchie, "Statistical issues in the analysis of Illumina data," *BMC Bioinformatics,* vol. 9, 2008.

[13] J. Bull, M. D. Westhead, M. Kambites, and J. Obdrzálek, "Towards OpenMP for java," in *European Workshop on OpenMP (EWOMP 2000)*, 2000, vol. 39, p. 40.

[14] A. Lastovetsky, "Adaptive parallel computing on heterogeneous networks with mpC," *Parallel computing,* vol. 28, no. 10, pp. 1369-1407, 2002.

[15] A. Reinefeld, J. Gehring, and M. Brune, "Communicating across parallel message-passing environments," *Computer Standards & Interfaces,* vol. 20, no. 6-7, p. 427, 1999.

[16] L. Poorthuis, K. Goergen, W. Sharples, and S. Kollet, "Implementation of parallel NetCDF in the ParFlow hydrological model: A code modernisation effort as part of a big data handling strategy," in *NIC Symposium 2016*, 2016, no. FZJ-2016-03551: Jülich Supercomputing Center.

[17] D. Goswami, A. Singh, and B. R. Preiss, "Architectural Skeletons: The Re-Usable Building-Blocks for Parallel Applications," in *PDPTA*, 1999, pp. 1250-1256.

[18] L. A. Drummond, V. G. Ibarra, V. Migallón, and J. Penadés, "Improving Ease of Use in BLACS and PBLAS with Python," in *PARCO*, 2005, pp. 325-332.

[19] A. A. Datti, H. A. Umar, and J. Galadanci, "A Beowulf Cluster for Teaching and Learning," *Procedia Computer Science,* vol. 70, pp. 62-68, 2015.

[20] L. Dalcin, "MPI for Python," ed: Release, 2010.

[21] M. Bubak, D. Kurzyniec, and P. Luszczek, "Creating Java to native code interfaces with Janet extension," in *Proceedings of the First Worldwide SGI Users' Conference*, 2000, pp. 283-294.

[22] N. Brown, "ePython: An Implementation of Python for the Many-Core Epiphany Co-processor," in *Python for High-Performance and Scientific Computing (PyHPC), Workshop on*, 2016, pp. 59-66: IEEE.

[23] D. J. Becker, T. Sterling, D. Savarese, J. E. Dorband, U. A. Ranawak, and C. V. Packer, "BEOWULF: A parallel workstation for scientific computation," in *Proceedings, International Conference on Parallel Processing*, 1995, vol. 95, pp. 11-14.