Message Passing Interface for Python

Frozan Maqsoodi
Institute of Computer Science,
University of Tartu
frozan.maqsoodi@ut.ee

***Abstract*— The Message Passing Interface (MPI) is a message passing library standard based on MPI forum. The goal of MPI is to establish a portable, efficient and flexible standard for message passing that will be widely used for writing message passing programs. In this paper, the work related to MPI for Python is shown that allows users to exploit multiple processors by using the MPI implementations.**

I. INTRODUCTION

Passing messages in parallel or in a distributed computing environment used to be a trouble for programmers due to incompatible issues. This trouble has been removed after MPI Forum gained a huge acceptance. MPI (message passing interface) is a leading standard message-passing library used for parallel computers. It is defined for some high performance scientific programming languages but some great implementations of MPI is also available in Python, this paper will briefly discuss the new implementations of MPI for Python and its additional features. Also, Python as being a famous programming language offers the advantage of writing parallel programs using MPI.

Section II presents a brief introduction of MPI, Python and MPI for Python. Section III describes the package support and new changes in MPI for Python including language bindings in Python from C++. Section IV is the conclusion.

II. BACKGROUND

A. MPI

MPI (message passing interface) is a message-passing interface library specification [1]. The standard describes the syntax and semantics of library procedures. Users can write parallel programs in programming languages such as C, C++ or Fortran. There also several implementations available for high-performance computing and open source projects like MPICH (MPI mpich), Open MPI, and Intel MPI library. MPI forum is available, which contains all the MPI standard document and version specifications[1]. The main reason of developments of MPI as a standard is to enhance message portability and simplicity in parallel environments.
This is an open sources library, developed by the active contribution of scientists, programmers, application developers and parallel computing vendors. C and Fortran, the famous scientific programming languages are included as part of the MPI standard, allowing flexible binding with the interface, since MPI is an interface not a language. Some common goal behind the development of MPI is the following.

- Deliver efficient communication by reducing memory-to-memory copying, increased correspondence in communication and computation
- Provide implementation for varied or heterogeneous environment
- Convenient binding in C and Fortran for the interface
- Independent language semantics and ability for thread safety
- Implementation capability on multi vendor platform

B. Python

Python is a modern programming language. It is a powerful language with high-level data structures that allows simple, effective, and a dynamic approach to object-oriented programming. The modules and packages in Python are pertinent for program modularity and code reuse. The standard library and Python interpreter are available in source or binary, also freely distributed for various major platforms. It is easily extended with new functions and data types implemented in C/C++. The codes in Python are easily are developed and maintained. It also achieves a high level of integration with other libraries written in compiled languages [2].

C. MPI for Python

MPI4py (message passing interface for Python) provides an object-oriented approach to MPI. This was designed to use the MPI-2 bindings for C++ to Python so the users can use this module without learning any new interface. This module is a Python interface to MPI that supports all MPI calls. Many of the python standard libraries rely on disk storage to support data persistence, however, this can also be achieved by pickling and marshaling 1 with memory buffers. The following command is used to install MPI4py

```
$ [sudo] pip install mpi4py
$ [sudo] easy_install mpi4py
```

III. MPI FOR PYTHON PACKAGE SUPPORT AND CHANGES

A. New Features in MPI for Python

1) Object Serialization : Most of the data persistence supported in Python rely on disk storage. Pickling and marshaling works with memory buffers. Pickle is a module, which provides user-extensible facilities to serialize Python

objects using ASCII or binary formats[2]. The marshal module provides facilities to serialize built-in Python objects using binary formats specific to Python, but independent of machine architecture issues [2]. Pickle is written in Python which a little slower than cPickle (faster) written in C. MPI for Python thus can easily communicate any built-in Python object with the help of features provided by cPickle and marshal modules. The functionalities of cPickle and marshal are wrapped in two classes; Pickle and Marshal, which defines the *dump()* and *load()* methods optimized for serialization of Python objects on memory streams. Pickle modules offer great features for MPI4py to communicate any built-in or user-defined Python objects. This is convenient to handle binary representations of objects to communicate and restore during sending and receiving processes respectively [3].

2) Direct Communication of Memory Buffers: Pickling and un-pickling is a serialization approach that inflicts high processor usage and memory. The inflict increases when objects of large memory being communicated. And pickling of Python objects requires computer resources and processing to dispatch the right serialization method depending on the type of object. In addition, more memory is needed and many reallocations can happen when total amount is unknown. MPI4py supports direct communication of any object exporting the single-segment buffer interface [3]. This interface is a standard mechanism in Python provided by strings and numeric arrays, which allows access to C memory buffers such as address and lengths that also contains all relevant data. This features allows implementation of complex algorithms such as image processing, fast Fourier Transforms, finite difference schemes on structured Cartesian grids, directly in Python with the same compiling speed as in Fortran, C or C++ codes.

3) Nonblocking and Persistent Communications: Nonblocking is a communication mechanism embedded in MPI to support overlap communication and computation. The Comm class in MPI has two important methods: *Isend()* and *Irecv()* which sends and receives operations respectively. These methods returns a *Request* instance that identifies about the start of operation with their completion controlled by *Request* class using *Test()*, *Wait()*, and *Cancel()*. The *Send_init()* and *Recv_init()*, methods of Comm class creates persistent calls for send and receive operations respectively by returning instance of the Prequest class (sub class of Request class). The communication is begins by *Start()* method.

Minimal MPI4py example

```
from mpi4py import MPI
rank = MPI.COMM_WORLD.Get_rank()
size = MPI.COMM_WORLD.Get_size()
name = MPI.Get_processor_name()
print("Hello World" "I am process %d of %d
"%(rank, size, name))
```

The result of the script will display "Hello World I am process 0 of 1"

B. MPI for Python Package Support

MPI4py package is constructed on top of MPI standard 1 and 2 specifications. This package allows object oriented interface that follows MPI-2 C++ bindings. The package contains mpi4py.MPI submodule. It contains the main functions i.e. $get\_include()$, returns the directory in the package that contains header files. Extension modules uses this function to locate appropriate include directory. $get\_config()$, returns a dictionary with information about MPI. The Cython (C expansion of Python) based implementation of MPI for python was released in 2009. The latest version released in 2017 has the following changes compared to its previous versions.

- New features include the followings
  - mpi4py.futures: Based on the package concurrent.futures (Python standard library for asynchronous computations), this features executes asynchronous computations using MPI processes as a pool. mpi4py.futures uses concurrent.futures form Python 2.7 or Python 3 standard library.
  - mpi4py.run: This is a new feature in MPI for Python. This runs Python code on MPI4py and ends all execution that are not handled to prevent deadlocks.
  - mpi4py.bench: Run basic MPI benchmarks and tests
- Enhancements:
  - Lowercase, pickle based collective communication calls are now thread safe through the use of fine grained locking.
  - MPI module exposes a memory type which is lightweight variant of the built-in memory view type, but exposes both the legacy Python 2 and Python 3 buffer interface under a Python 2 runtime.
  - $MPI.Comm.Alltoallw()$ method now uses $count = 1$ and $displ = 0$ as defaults, assuming that messages are specified by user-defined data types. This is a routine method for collective communication and specifically distributes data from all to all
  - The MPI $Request.Wait[all]()$ methods now return True to match the interface of $Request.Test[all]()$. The Request.Wait[all] waits for all previously initiated requests to complete and Request.Test[all] tests for completion of any previously initiated requests
  - The Win (window) class implements the Python buffer interface.
- Backward-incompatible changes:
  - The buf argument of $MPI.Comm.recv()$ method is deprecated, passing anything but None emits a warning.

- The $MPI.Win.memory$ is replaced with $MPI.Win.tomemory()$
- Executing *python -m MPI4py* in the command line is equivalent to *python -m MPI4py.run*
- mpi4py.MPI no longer supports Python 2.6 and 3.2

C. MPI Language Binding

MPI is produced by MPI forum and is under continuous development[1]. Several research on MPI is going on in different areas such as performance optimization, interoperability with Open message passing. In regards to language binding, MPI forum has replaced the C++ API (application programming interface). There are better C++ MPI bindings developed outside MPI e.g Boost.MPI, which explicitly accepts language binding outside the MPI standard. Boost.MPI is basically a C++ library. Python can also access this library via Boost.Python library. with Boost.Python library, it is easy to export C++ to python where the python and C++ interface exactly remain the same. This allows very less or minimal changes in C++ classes in order to use them with Boost.Python. The bindings in Boost.Python is done in using C++ compiler and editor [4]. Python provides an API to interact Python and C together. And Boost.Python is a wrapper for Python and C API, such that when a pointer is passed between C and Python, the pointer will not hang if the object it is passed to is no longer available. In addition, Boost.Python also allows to perform Python operations on C++ in object oriented style. For example, using Boost.MPI, can easily simplify the Python/C API in the following way.

$$y = PySequence\_GetItem(object\_x, i);$$

can be equivalently done in Boost.Python as

$$y = object\_x[i];$$

Boost.Python also allows to easily export C++ classes into Python, without changing them. It is simply designed so that the users do not require to change any PyObject.

D. Features in MPI for Python Language Binding

[8]

- Convenient communication of any picklable Python object
- Fast communication of Python object exposing the Python buffer interface (NumPy arrays, builtin bytes/string/array objects)
- Process groups and communication domains
- Parallel input/output
- Dynamic process management
- One-sided operations

E. MPI4py Functionality

There are hundreds of functions available in MPI standards, not all of them are available in MPI4py. There is no need to call two functions after importing MPI in MPI4py, which is an important particularity. The functions are $MPI\_Init()$ and $MPI\_Finalize()$, importing MPI4py already triggers $MPI\_Init()$ and $MPI\_Finalize()$ is called when all python processes finish execution or exit [7]. The main variables or communicators that need to be initialized are:

- COMM_WORLD (has all processes involved)
- COMM_SELF (contains just the calling process)
- COMM_NULL (no communicator)

MPI scripts are executed using mpiexec command and by calling the library at the beginning, the first command (import MPI from MPI4py) imports the MPI library. And the second one executes a python script (mpiexec -n 4 python script.py) 4 represents the number of time the output is displayed or run program with 4 processors. The communication of generic python object should be all in lowercase case functions, in the communication (Comm) class like send(), recv(), bcast().

IV. CONCLUSIONS

MPI for Python provides a baseline for applying the message-passing model in parallel applications written in Python. By taking the usage of MPI implementations and retaining the syntax and standard of its specification. MPI for Python can communicate general Python objects including any Python object exposing buffer memory. Future work in MPI for Python is dedicated towards more implementations of MPI functionalities like data types decoding, attribute catching and interoperability with Fortran libraries.

REFERENCES

[1] Message Passing Interface Forum, Message Passing Interface (MPI) Forum Home Page, http://mpi-forum.org/docs/mpi-3.1/mpi31-report.pdf.

[2] MPI for Python, MPI for Python (MPI4py) Home Page, http://MPI4py.scipy.org/docs/MPI4py.pdf

[3] MPI for Python: Performance Improvements and MPI-2 Extensions, Home Page, https://docs.python.org/3/library/pickle.html#module-pickle

[4] Message Passing Interface Forum, Message Passing Interface (MPI) Forum Home Page, http://mpi-forum.org/

[5] Boost Python, Boost C++ Libraries, Home Page, http://www.boost.org/doc/libs/1_43_0/doc/html/mpi/python.html

[6] Yiannis Cotronis, Anthony Danalis, Dimitris Nikolopoulos, Jack Dongarra, Recent Advances in the Message Passing Interface: 18th European MPI Users Group Meeting, EuroMPI 2011, Santorini, Greece, September 18-21, 2011. Proceedings (Periodical styleSubmitted for publication), Page#294.

[7] MPI for Python, MPI Futures, Home Page, https://MPI4py.readthedocs.io/en/latest/MPI4py.futures.html

[8] Changes: MPI for Python, Author Lisandro Dalcin, Home Page, https://github.com/MPI4py/MPI4py/blob/master/DESCRIPTION.rst

[9] L. Dalcin, P. Kler, R. Paz, and A. Cosimo, Parallel Distributed Computing using Python, Advances in Water Resources, 34(9):1124-1139, 2011. http://dx.doi.org/10.1016/j.advwatres.2011.04.013

[10] L. Dalcin, R. Paz, M. Storti, and J. DElia, MPI for Python: performance improvements and MPI-2 extensions, Journal of Parallel and Distributed Computing, 68(5):655-662, 2008. http://dx.doi.org/10.1016/j.jpdc.2007.09.005

[11] L. Dalcin, R. Paz, and M. Storti, MPI for Python, Journal of Parallel and Distributed Computing, 65(9):1108-1115, 2005. http://dx.doi.org/10.1016/j.jpdc.2005.03.010