The ASAP Python Toolbox Documentation

Release 0.6.2.post46

Kevin Paul, John Dennis, Sheri Mickelson, Haiying Xu

Jan 26, 2021

CONTENTS

Contents:

CHAPTER

ONE

THE ASAP PYTHON TOOLBOX

The ASAP Python Toolbox is a collection of stand-alone tools for doing simple tasks, from managing print messages with a set verbosity level, to keeping timing information, to managing simple MPI communication.

Send questions and comments to Kevin Paul (kpaul@ucar.edu).

1.1 Overview

The ASAP (Application Scalability And Performance) group at the National Center for Atmospheric Research maintains this collection of simple Python tools for managing tasks commonly used with its Python software. The modules contained in this package include:

vprinter For managing print messages with verbosity-level specification

timekeeper For managing multiple "stop watches" for timing metrics

partition For various data partitioning algorithms

simplecomm For simple MPI communication

Only the simplecomm module depends on anything beyond the basic built-in Python packages.

1.2 Dependencies

All of the ASAP Python Toolbox tools are written to work with Python 2.6+ (including Python 3+). The vprinter, timekeeper, and partition modules are pure Python. The simplecomm module depends on mpi4py (>=1.3).

This implies the dependency:

- mpi4py depends on numpy (>-1.4) and MPI

1.3 Easy Installation

The easiest way to install the ASAP Python Toolbox is from the Python Package Index (PyPI) with the pip package manager:

```
$  pip install [--user] asaptools
```

The optional '–user' argument can be used to install the package in the local user's directory, which is useful if the user doesn't have root privileges.

1.4 Obtaining the Source Code

Currently, the most up-to-date source code is available via git from the site:

```
https://github.com/NCAR/ASAPPyTools
```

Check out the most recent tag. The source is available in read-only mode to everyone, but special permissions can be given to those to make changes to the source.

1.5 Building & Installation

Installation of the ASAP Python Toolbox is very simple. After checking out the source from the above svn link, via:

```
$  git clone https://github.com/NCAR/ASAPPyTools
```

change into the top-level source directory, check out the most recent tag, and run the Python distutils setup. On unix, this involves:

```
$  cd ASAPPyTools
$  python setup.py install [--prefix-/path/to/install/location]
```

The prefix is optional, as the default prefix is typically /usr/local on linux machines. However, you must have permissions to write to the prefix location, so you may want to choose a prefix location where you have write permissions. Like most distutils installations, you can alternatively install the pyTools with the –user option, which will automatically select (and create if it does not exist) the $HOME/.local directory in which to install. To do this, type (on unix machines):

```
$  python setup.py install --user
```

This can be handy since the site-packages directory will be common for all user installs, and therefore only needs to be added to the PYTHONPATH once.

1.6 Instructions & Use

For instructions on how to use the ASAP Python Toolbox, see the documentation.

CHAPTER

TWO

ASAPTOOLS PACKAGE

The ASAP Python Toolbox

The ASAP Python Toolbox is a collection of stand-alone tools for doing simple tasks, from managing print messages with a set verbosity level, to keeping timing information, to managing simple MPI communication.

Send questions and comments to Kevin Paul (kpaul@ucar.edu).

2.1 Submodules

2.1.1 asaptools.vprinter module

A module containing the VPrinter class.

This module contains the VPrinter class that enables clean printing to standard out (or a string) with verbosity-level print management.

class asaptools.vprinter.**VPrinter**(*header=''*, *verbosity=1*)

Bases: object

A Verbosity-enabled Printing Class.

The VPrinter is designed to print messages to standard out, or optionally a string, as determined by a pre-set verbosity-level and/or on which parallel rank the VPrinter is instantiated.

header
A string to prepend to any print messages before they are printed

Type str

verbosity
The verbosity level to use when determining if a message should be printed

Type int

to_str(**args*, ***kwargs*)
Concatenates string representations of the input arguments.

This takes a list of arguments of any length, converts each argument to a string representation, and concatenates them into a single string.

Parameters **args** (*list*) – A list of arguments supplied to the function. All of these arguments will be concatenated together.

Keyword Arguments **kwargs** (*dict*) – The dictionary of keyword arguments passed to the function.

Returns

A single string with the arguments given converted to strings and concatenated together (in order). If the keyword 'header==True' is supplied, then the 'header' string is prepended to the string before being output.

Return type str

Raises **TypeError** – If the 'header' keyword argument is supplied and is not a bool

2.1.2 asaptools.timekeeper module

A module containing the TimeKeeper class.

This module contains is a simple class to act as a time keeper for internal performance monitoring (namely, timing given processes).

class asaptools.timekeeper.**TimeKeeper**(*time=<built-in function time>*)
Bases: object

Class to keep timing recordings, start/stop/reset timers.

_time
The method to use for getting the time (e.g., time.time)

_start_times
A dictionary of start times for each named timer

Type dict

_accumulated_times
A dictionary of the total accumulated times for each named timer

Type dict

_added_order
A list containing the name of each timer, in the order it was added to the TimeKeeper

Type list

get_all_times()
Returns the dictionary of accumulated times on the local processor.

Returns The dictionary of accumulated times

Return type dict

get_names()
Method to return the clock names in the order in which they were added.

Returns The list of timer names in the order they were added

Return type list

get_time(*name*)
Returns the accumulated time of the given timer.

If the given timer name has never been created, it is created and the accumulated time is set to zero before returning.

Parameters **name** – The name or ID of the timer to stop

Returns

The accumulated time of the named timer (or 0.0 if the named timer has never been created before).

Return type float

reset(*name*)
Method to reset a timer associated with a given name.

If the name has never been used before, the timer is created and the accumulated time is set to 0. If the timer has been used before, the accumulated time is set to 0.

Parameters **name** – The name or ID of the timer to reset

start(*name*)
Method to start a timer associated with a given name.

If the name has never been used before, the timer is created and the accumulated time is set to 0.

Parameters **name** – The name or ID of the timer to start

stop(*name*)
Stop the timing and add the accumulated time to the timer.

Method to stop a timer associated with a given name, and adds the accumulated time to the timer when stopped. If the given timer name has never been used before (either by calling reset() or start()), the timer is created and the accumulated time is set to 0.

Parameters **name** – The name or ID of the timer to stop

2.1.3 asaptools.partition module

A module for data partitioning functions.

This provides a collection of 'partitioning' functions. A partitioning function is a three-argument function that takes, as the first argument, a given data object and, as the second argument, an index into that object and, as the third argument, a maximum index. The operation of the partitioning function is to return a subset of the data corresponding to the given index.

By design, partitioning functions should keep the data "unchanged" except for subselecting parts of the data.

class asaptools.partition.**Duplicate**
Bases: *asaptools.partition.PartitionFunction*

Return a copy of the original input data in each partition.

class asaptools.partition.**EqualLength**
Bases: *asaptools.partition.PartitionFunction*

Partition an indexable object by striding through the data.

The initial object is "chopped" along its length into roughly equal length sublists. If the partition size is greater than the length of the input data, then it will return an empty list for 'empty' partitions. If the data is not indexable, then it will return the data for index=0 only, and an empty list otherwise.

class asaptools.partition.**EqualStride**
Bases: *asaptools.partition.PartitionFunction*

Partition an object by chopping the data into roughly equal lengths.

This returns a sublist of an indexable object by "striding" through the data in steps equal to the partition size. If the partition size is greater than the length of the input data, then it will return an empty list for "empty" partitions. If the data is not indexable, then it will return the data for index=0 only, and an empty list otherwise.

class asaptools.partition.**PartitionFunction**
Bases: object

The abstract base-class for all Partitioning Function objects.

A PartitionFunction object is one with a __call__ method that takes three arguments. The first argument is the data to be partitioned, the second argument is the index of the partition (or part) requested, and third argument is the number of partitions to assume when dividing the data.

class asaptools.partition.**SortedStride**
Bases: *asaptools.partition.PartitionFunction*

Partition an indexable list of pairs by striding through sorted data.

The first index of each pair is assumed to be an item of data (which will be partitioned), and the second index in each pair is assumed to be a numeric weight. The pairs are first sorted by weight, and then partitions are returned by striding through the sorted data.

The results are partitions of roughly equal length and roughly equal total weight. However, equal length is prioritized over total weight.

class asaptools.partition.**WeightBalanced**
Bases: *asaptools.partition.PartitionFunction*

Partition an indexable list of pairs by balancing the total weight.

The first index of each pair is assumed to be an item of data (which will be partitioned), and the second index in each pair is assumed to be a numeric weight. The data items are grouped via a "greedy" binning algorithm into partitions of roughly equal total weight.

The results are partitions of roughly equal length and roughly equal total weight. However, equal total weight is prioritized over length.

2.1.4 asaptools.simplecomm module

A module for simple MPI communication.

The SimpleComm class is designed to provide a simplified MPI-based communication strategy using the MPI4Py module.

To accomplish this task, the SimpleComm object provides a single communication pattern with a simple, light-weight API. The communication pattern is a common 'manager'/'worker' pattern, with the 0th rank assumed to be the 'manager' rank. The SimpleComm API provides a way of sending data out from the 'manager' rank to the 'worker' ranks, and for collecting the data from the 'worker' ranks back on the 'manager' rank.

PARTITIONING:

Within the SimpleComm paradigm, the 'manager' rank is assumed to be responsible for partition (or distributing) the necessary work to the 'worker' ranks. The *partition* mathod provides this functionality. Using a *partition function*, the *partition* method takes data known on the 'manager' rank and gives each 'worker' rank a part of the data according to the algorithm of the partition function.

The *partition* method is *synchronous*, meaning that every rank (from the 'manager' rank to all of the 'worker' ranks) must be in synch when the method is called. This means that every rank must participate in the call, and every rank will wait until all of the data has been partitioned before continuing. Remember, whenever the 'manager' rank speaks, all of the 'worker' ranks listen! And they continue to listen until dismissed by the 'manager' rank.

Additionally, the 'manager' rank can be considered *involved* or *uninvolved* in the partition process. If the 'manager' rank is *involved*, then the master will take a part of the data for itself. If the 'manager' is *uninvolved*, then the data will be partitioned only across the 'worker' ranks.

Partitioning is a *synchronous* communication call that implements a *static partitioning* algorithm.

RATIONING:

An alternative approach to the *partitioning* communication method is the *rationing* communication method. This method involves the individual 'worker' ranks requesting data to work on. In this approach, each 'worker' rank, when the 'worker' rank is ready, asks the 'manager' rank for a new piece of data on which to work. The 'manager' rank receives the request and gives the next piece of data for processing out to the requesting 'worker' rank. It doesn't matter what order the ranks request data, and they do not all have to request data at the same time. However, it is critical to understand that if a 'worker' requests data when the 'manager' rank does not listen for the request, or the 'manager' expects a 'worker' to request work but the 'worker' never makes the request, the entire process will hang and wait forever!

Rationing is an *asynchronous* communication call that allows the 'manager' to implement a *dynamic partitioning* algorithm.

COLLECTING:

Once each 'worker' has received its assigned part of the data, the 'worker' will perform some work pertaining to the data it received. In such a case, the 'worker' may (though not necessarily) return one or more results back to the 'manager'. The *collect* method provides this functionality.

The *collect* method is *asynchronous*, meaning that each slave can send its data back to the master at any time and in any order. Since the 'manager' rank does not care where the data came from, the 'manager' rank simply receives the result from the 'worker' rank and processes it. Hence, all that matters is that for every *collect* call made by all of the 'worker' ranks, a *collect* call must also be made by the 'manager' rank.

The *collect* method is a *handshake* method, meaning that while the 'manager' rank doesn't care which 'worker' rank sends it data, the 'manager' rank does acknowledge the 'worker' rank and record the 'worker' rank's identity.

REDUCING:

In general, it is assumed that each 'worker' rank works independently from the other 'worker' ranks. However, it may be occasionally necessary for the 'worker' ranks to know something about the work being done on (or the data given to) each of the other ranks. The only allowed communication of this type is provided by the *allreduce* method.

The *allreduce* method allows for *reductions* of the data distributed across all of the ranks to be made available to every rank. Reductions are operations such as 'max', 'min', 'sum', and 'prod', which compute and distribute to the ranks the 'maximum', 'minimum', 'sum', or 'product' of the data distributed across the ranks. Since the *reduction* computes a reduced quantity of data distributed across all ranks, the *allreduce* method is a *synchronous* method (i.e., all ranks must participate in the call, including the 'manager').

DIVIDING:

It can be occasionally useful to subdivide the 'worker' ranks into different groups to perform different tasks in each group. When this is necessary, the 'manager' rank will assign itself and each 'worker' rank a *color* ID. Then, the 'manager' will assign each rank (including itself) to 2 new groups:

1. **Each rank with the same color ID will be assigned to the same group, and** within this new *color* group, each rank will be given a new rank ID ranging from 0 (identifying the color group's 'manager' rank) to the number of 'worker' ranks in the color group. This is called the *monocolor* grouping.
2. **Each rank with the same new rank ID across all color groups will be assigned** to the same group. Hence, all ranks with rank ID 0 (but different color IDs) will be in the same group, all ranks with rank ID 1 (but different color IDs) will be the in another group, etc. This is called the *multicolor* grouping. NOTE: This grouping will look like grouping (1) except with the rank ID and the color ID swapped.

The *divide* method provides this functionality, and it returns 2 new SimpleComm objects for each of the 2 groupings described above. This means that within each group, the same *partition*, *collecting*, and *reducing* operations can be performed in the same way as described above for the *global* group.

class asaptools.simplecomm.**SimpleComm**

Bases: object

Simple Communicator for serial operation.

_numpy

Reference to the Numpy module, if found

_color
The color associated with the communicator, if colored

_group
The group ID associated with the communicator's color

allreduce(*data*, *op*)
Perform an MPI AllReduction operation.

The data is "reduced" across all ranks in the communicator, and the result is returned to all ranks in the communicator. (Reduce operations such as 'sum', 'prod', 'min', and 'max' are allowed.)

This call must be made by all ranks.

Parameters

- **data** – The data to be reduced
- **op** (*str*) – A string identifier for a reduce operation (any string found in the OPERATORS list)

Returns The single value constituting the reduction of the input data. (The same value is returned on all ranks in this communicator.)

collect(*data=None*, *tag=0*)
Send data from a 'worker' rank to the 'manager' rank.

If the calling MPI process is the 'manager' rank, then it will receive and return the data sent from the 'worker'. If the calling MPI process is a 'worker' rank, then it will send the data to the 'manager' rank.

For each call to this function on a given 'worker' rank, there must be a matching call to this function made on the 'manager' rank.

NOTE: This method cannot be used for communication between the 'manager' rank and itself. Attempting this will cause the code to hang.

Keyword Arguments

- **data** – The data to be collected asynchronously on the manager rank.
- **tag** (*int*) – A user-defined integer tag to uniquely specify this communication message

Returns On the 'manager' rank, a tuple containing the source rank ID and the data collected. None on all other ranks.

Raises **RuntimeError** – If executed during a serial or 1-rank parallel run

divide(*group*)
Divide this communicator's ranks into groups.

Creates and returns two (2) kinds of groups:

1. **groups with ranks of the same color ID but different rank IDs** (called a "monocolor" group), and
2. **groups with ranks of the same rank ID but different color IDs** (called a "multicolor" group).

Parameters **group** – A unique group ID to which will be assigned an integer color ID ranging from 0 to the number of group ID's supplied across all ranks

Returns

A tuple containing (first) the "monocolor" SimpleComm for ranks with the same color ID (but different rank IDs) and (second) the "multicolor" SimpleComm for ranks with the same rank ID (but different color IDs)

Return type tuple

Raises **RuntimeError** – If executed during a serial or 1-rank parallel run

get_color()
Get the integer color ID of this MPI process in this communicator.

By default, a communicator's color is None, but a communicator can be divided into color groups using the 'divide' method.

This call can be made independently from other ranks.

Returns The color of this MPI communicator

Return type int

get_group()
Get the group ID of this MPI communicator.

The group ID is the argument passed to the 'divide' method, and it represents a unique identifier for all ranks in the same color group. It can be any type of object (e.g., a string name).

This call can be made independently from other ranks.

Returns The group ID of this communicator

get_rank()
Get the integer rank ID of this MPI process in this communicator.

This call can be made independently from other ranks.

Returns The integer rank ID of this MPI process

Return type int

get_size()
Get the integer number of ranks in this communicator.

The size includes the 'manager' rank.

Returns The integer number of ranks in this communicator.

Return type int

is_manager()
Check if this MPI process is on the 'manager' rank (i.e., rank 0).

This call can be made independently from other ranks.

Returns

True if this MPI process is on the master rank (or rank 0). False otherwise.

Return type bool

partition(*data=None*, *func=None*, *involved=False*, *tag=0*)
Partition and send data from the 'manager' rank to 'worker' ranks.

By default, the data is partitioned using an "equal stride" across the data, with the stride equal to the number of ranks involved in the partitioning. If a partition function is supplied via the *func* argument, then the data will be partitioned across the 'worker' ranks, giving each 'worker' rank a different part of the data according to the algorithm used by partition function supplied.

If the *involved* argument is True, then a part of the data (as determined by the given partition function, if supplied) will be returned on the 'manager' rank. Otherwise, ('involved' argument is False) the data will be partitioned only across the 'worker' ranks.

This call must be made by all ranks.

Keyword Arguments

- **data** – The data to be partitioned across the ranks in the communicator.
- **func** – A PartitionFunction object/function that returns a part of the data given the index and assumed size of the partition.
- **involved** (*bool*) – True if a part of the data should be given to the 'manager' rank in addition to the 'worker' ranks. False otherwise.
- **tag** (*int*) – A user-defined integer tag to uniquely specify this communication message.

Returns A (possibly partitioned) subset (i.e., part) of the data. Depending on the PartitionFunction used (or if it is used at all), this method may return a different part on each rank.

ration (*data=None*, *tag=0*)

Send a single piece of data from the 'manager' rank to a 'worker' rank.

If this method is called on a 'worker' rank, the worker will send a "request" for data to the 'manager' rank. When the 'manager' receives this request, the 'manager' rank sends a single piece of data back to the requesting 'worker' rank.

For each call to this function on a given 'worker' rank, there must be a matching call to this function made on the 'manager' rank.

NOTE: This method cannot be used for communication between the 'manager' rank and itself. Attempting this will cause the code to hang.

Keyword Arguments

- **data** – The data to be asynchronously sent to the 'worker' rank
- **tag** (*int*) – A user-defined integer tag to uniquely specify this communication message

Returns On the 'worker' rank, the data sent by the manager. On the 'manager' rank, None.

Raises **RuntimeError** – If executed during a serial or 1-rank parallel run

sync()

Synchronize all MPI processes at the point of this call.

Immediately after this method is called, you can guarantee that all ranks in this communicator will be synchronized.

This call must be made by all ranks.

class asaptools.simplecomm.**SimpleCommMPI**

Bases: *asaptools.simplecomm.SimpleComm*

Simple Communicator using MPI.

PART_TAG

Partition Tag Identifier

RATN_TAG

Ration Tag Identifier

CLCT_TAG

Collect Tag Identifier

REQ_TAG

Request Identifier

MSG_TAG

Message Identifer

ACK_TAG
Acknowledgement Identifier

PYT_TAG
Python send/recv Identifier

NPY_TAG
Numpy send/recv Identifier

_mpi
A reference to the mpi4py.MPI module

_comm
A reference to the mpi4py.MPI communicator

ACK_TAG = 3

CLCT_TAG = 3

MSG_TAG = 2

NPY_TAG = 5

PART_TAG = 1

PYT_TAG = 4

RATN_TAG = 2

REQ_TAG = 1

allreduce(*data*, *op*)
Perform an MPI AllReduction operation.

The data is "reduced" across all ranks in the communicator, and the result is returned to all ranks in the communicator. (Reduce operations such as 'sum', 'prod', 'min', and 'max' are allowed.)

This call must be made by all ranks.

Parameters

- **data** – The data to be reduced
- **op** (*str*) – A string identifier for a reduce operation (any string found in the OPERATORS list)

Returns The single value constituting the reduction of the input data. (The same value is returned on all ranks in this communicator.)

collect(*data=None*, *tag=0*)
Send data from a 'worker' rank to the 'manager' rank.

If the calling MPI process is the 'manager' rank, then it will receive and return the data sent from the 'worker'. If the calling MPI process is a 'worker' rank, then it will send the data to the 'manager' rank.

For each call to this function on a given 'worker' rank, there must be a matching call to this function made on the 'manager' rank.

NOTE: This method cannot be used for communication between the 'manager' rank and itself. Attempting this will cause the code to hang.

Keyword Arguments

- **data** – The data to be collected asynchronously on the 'manager' rank.
- **tag** (*int*) – A user-defined integer tag to uniquely specify this communication message

Returns

On the 'manager' rank, a tuple containing the source rank ID and the the data collected. None on all other ranks.

Return type tuple

Raises **RuntimeError** – If executed during a serial or 1-rank parallel run

divide(*group*)

Divide this communicator's ranks into groups.

Creates and returns two (2) kinds of groups:

(1) groups with ranks of the same color ID but different rank IDs (called a "monocolor" group), and

(2) groups with ranks of the same rank ID but different color IDs (called a "multicolor" group).

Parameters **group** – A unique group ID to which will be assigned an integer color ID ranging from 0 to the number of group ID's supplied across all ranks

Returns

A tuple containing (first) the "monocolor" SimpleComm for ranks with the same color ID (but different rank IDs) and (second) the "multicolor" SimpleComm for ranks with the same rank ID (but different color IDs)

Return type tuple

Raises **RuntimeError** – If executed during a serial or 1-rank parallel run

get_rank()

Get the integer rank ID of this MPI process in this communicator.

This call can be made independently from other ranks.

Returns The integer rank ID of this MPI process

Return type int

get_size()

Get the integer number of ranks in this communicator.

The size includes the 'manager' rank.

Returns The integer number of ranks in this communicator.

Return type int

partition(*data=None*, *func=None*, *involved=False*, *tag=0*)

Partition and send data from the 'manager' rank to 'worker' ranks.

By default, the data is partitioned using an "equal stride" across the data, with the stride equal to the number of ranks involved in the partitioning. If a partition function is supplied via the 'func' argument, then the data will be partitioned across the 'worker' ranks, giving each 'worker' rank a different part of the data according to the algorithm used by partition function supplied.

If the 'involved' argument is True, then a part of the data (as determined by the given partition function, if supplied) will be returned on the 'manager' rank. Otherwise, ('involved' argument is False) the data will be partitioned only across the 'worker' ranks.

This call must be made by all ranks.

Keyword Arguments

- **data** – The data to be partitioned across the ranks in the communicator.

- **func** – A PartitionFunction object/function that returns a part of the data given the index and assumed size of the partition.
- **involved** (*bool*) – True, if a part of the data should be given to the 'manager' rank in addition to the 'worker' ranks. False, otherwise.
- **tag** (*int*) – A user-defined integer tag to uniquely specify this communication message

Returns A (possibly partitioned) subset (i.e., part) of the data. Depending on the PartitionFunction used (or if it is used at all), this method may return a different part on each rank.

ration(*data=None*, *tag=0*)

Send a single piece of data from the 'manager' rank to a 'worker' rank.

If this method is called on a 'worker' rank, the worker will send a "request" for data to the 'manager' rank. When the 'manager' receives this request, the 'manager' rank sends a single piece of data back to the requesting 'worker' rank.

For each call to this function on a given 'worker' rank, there must be a matching call to this function made on the 'manager' rank.

NOTE: This method cannot be used for communication between the 'manager' rank and itself. Attempting this will cause the code to hang.

Keyword Arguments

- **data** – The data to be asynchronously sent to the 'worker' rank
- **tag** (*int*) – A user-defined integer tag to uniquely specify this communication message

Returns On the 'worker' rank, the data sent by the manager. On the 'manager' rank, None.

Raises **RuntimeError** – If executed during a serial or 1-rank parallel run

sync()

Synchronize all MPI processes at the point of this call.

Immediately after this method is called, you can guarantee that all ranks in this communicator will be synchronized.

This call must be made by all ranks.

asaptools.simplecomm.**create_comm**(*serial=False*)

This is a factory function for creating SimpleComm objects.

Depending on the argument given, it returns an instance of a serial or parallel SimpleComm object.

Keyword Arguments **serial** (*bool*) – A boolean flag with True indicating the desire for a serial SimpleComm instance, and False incidicating the desire for a parallel SimpleComm instance.

Returns

An instance of a SimpleComm object, either serial (if serial == True) or parallel (if serial == False)

Return type *SimpleComm*

Raises **TypeError** – if the serial argument is not a bool.

Examples

```
>>> sercomm = create_comm(serial=True)
>>> type(sercomm)
<class 'simplecomm.SimpleComm'>
```

```
>>> parcomm = create_comm()
>>> type(parcomm)
<class 'simplecomm.SimpleCommMPI'>
```

CHAPTER

THREE

CHANGE LOG

3.1 Version 0.7.0

- Big refactor to use GitHub workflows (adding testing for Python version 3.7 and 3.8)
- Modernizing the package structure

3.2 Version 0.6.0

- Allowing for support of all Python 2.6+ (including Python 3+)

3.3 Version 0.5.4

- Bugfix: Special catch for dtype='c' (C-type char arrays) in check for Numpy arrays being bufferable

3.4 Version 0.5.3

- Updates just for PyPI release

3.5 Version 0.5.2

- Improved testing for send/recv data types
- Backwards compatability with mpi4py version 1.3.1

3.6 Version 0.5.1

- Checking dtype of Numpy NDArrays before determing if buffered send/recv calls can be used.

3.7 Version 0.5.0

- Now requires Python >=2.7 and <3.0
- Using more advanced features of Python 2.7 (over 2.6)
- Changed Numpy NDArray type-checking to allow for masked arrays, instead of just NDArrays

3.8 Version 0.4.2

- Update setup script to setuptools (instead of distutils)

3.9 Version 0.4.1

- Bugfixes

3.10 Version 0.4

- Updating install to include LICENSE
- Restructured source directory
- Upload to PyPI

3.11 Version 0.3

- Repackaging the pyTools repo into a Python package with installation software and Sphinx-style documentation

CHAPTER

FOUR

PRODUCT LICENSE

Apache License

Version 2.0, January 2004

http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

   "License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

   "Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

   "Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

   "You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

   "Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

   "Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

   "Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

   "Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

   "Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its

representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

   (a) You must give any other recipients of the Work or Derivative Works a copy of this License; and

   (b) You must cause any modified files to carry prominent notices stating that You changed the files; and

   (c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

   (d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

   You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

> To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "[]" replaced with your own identifying information. (Don't include the brackets!) The text should be enclosed in the appropriate comment syntax for the file format. We also recommend that a file or class name and description of purpose be included on the same "printed page" as the copyright notice for easier identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

> http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

a

INDEX

Symbols

A

C

D

E

G

H

I

M

N

P

R

S

T

V

W