numba-mpi

Numba @njittable MPI wrappers tested on Linux macOS and Windows

S. Arabas¹, O. Bulenok¹, K. Derlatka¹, M. Manna¹ & D. Zwicker²

FOSDEM’23 HPC, Big Data, and Data Science Devroom @ ULB (Feb 5 2023)

1: Jagiellonian University, Kraków, Poland
2: Max Planck Institute for Dynamics and Self-Organization, Göttingen, Germany
Python & HPC?
<table>
<thead>
<tr>
<th>Perkel 2019 (Nature)</th>
<th>doi:10.1038/d41586-019-02310-3</th>
</tr>
</thead>
<tbody>
<tr>
<td>“In scripting languages such as Python, users type code into an interactive editor line by line”</td>
<td></td>
</tr>
<tr>
<td>Perkel 2019 (Nature)</td>
<td></td>
</tr>
<tr>
<td>----------------------</td>
<td></td>
</tr>
<tr>
<td>doi:10.1038/d41586-019-02310-3</td>
<td></td>
</tr>
<tr>
<td>“In scripting languages such as Python, users type code into an interactive editor line by line”</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Perkel 2020 (Nature)</th>
</tr>
</thead>
<tbody>
<tr>
<td>doi:10.1038/d41586-020-03382-2</td>
</tr>
<tr>
<td>“level of computational performance that Python simply couldn’t deliver”</td>
</tr>
</tbody>
</table>
“In scripting languages such as Python, users type code into an interactive editor line by line”

“level of computational performance that Python simply couldn’t deliver”

“NumPy ... runs on machines ranging from embedded devices to the world’s largest supercomputers, with performance approaching that of compiled languages”
In scripting languages such as Python, users type code into an interactive editor line by line.

level of computational performance that Python simply couldn’t deliver

astronomers should avoid interpreted scripting languages such as Python ... in principle, Numba and NumPy can lead to an enormous increase in speed ... ... reconsider teaching Python to university students.

NumPy ... runs on machines ranging from embedded devices to the world’s largest supercomputers, with performance approaching that of compiled languages
<table>
<thead>
<tr>
<th>Author</th>
<th>Year</th>
<th>Source</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Perkel 2019 (Nature)</td>
<td>doi:10.1038/d41586-019-02310-3</td>
<td>“In scripting languages such as Python, users type code into an interactive editor line by line”</td>
<td></td>
</tr>
<tr>
<td>Perkel 2020 (Nature)</td>
<td>doi:10.1038/d41586-020-03382-2</td>
<td>“level of computational performance that Python simply couldn’t deliver”</td>
<td></td>
</tr>
<tr>
<td>Zwart 2020 (Nature Astronomy)</td>
<td>doi:10.1038/s41550-020-1208-y</td>
<td>“astronomers should avoid interpreted scripting languages such as Python ... in principle, Numba and NumPy can lead to an enormous increase in speed ... ... reconsider teaching Python to university students.”</td>
<td></td>
</tr>
<tr>
<td>Harris et al. 2020 (Nature)</td>
<td>doi:10.1038/s41586-020-2649-2</td>
<td>“NumPy ... runs on machines ranging from embedded devices to the world’s largest supercomputers, with performance approaching that of compiled languages”</td>
<td></td>
</tr>
<tr>
<td>Virtanen et al. 2020 (Nature Methods)</td>
<td>doi:10.1038/s41592-019-0686-2</td>
<td>“implementing new functionality, Python is the still the language of choice ... [SciPy] full test suite passed with the PyPy JIT compiler at the 1.0 release point”</td>
<td></td>
</tr>
<tr>
<td>Author(s)</td>
<td>Year</td>
<td>Title</td>
<td>DOI</td>
</tr>
<tr>
<td>-----------------------------------------------</td>
<td>------</td>
<td>--------------------------------------------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>Perkel 2019 (Nature)</td>
<td>2019</td>
<td>“In scripting languages such as Python, users type code into an interactive editor line by line”</td>
<td>doi:10.1038/d41586-019-02310-3</td>
</tr>
<tr>
<td>Perkel 2020 (Nature)</td>
<td>2020</td>
<td>“level of computational performance that Python simply couldn’t deliver”</td>
<td>doi:10.1038/d41586-020-03382-2</td>
</tr>
<tr>
<td>Zwart 2020 (Nature Astronomy)</td>
<td>2020</td>
<td>“astronomers should avoid interpreted scripting languages such as Python ... in principle, Numba and NumPy can lead to an enormous increase in speed ... reconsider teaching Python to university students.”</td>
<td>doi:10.1038/s41550-020-1208-y</td>
</tr>
<tr>
<td>Harris et al. 2020 (Nature)</td>
<td>2020</td>
<td>“NumPy ... runs on machines ranging from embedded devices to the world’s largest supercomputers, with performance approaching that of compiled languages”</td>
<td>doi:10.1038/s41586-020-2649-2</td>
</tr>
<tr>
<td>Virtanen et al. 2020 (Nature Methods)</td>
<td>2020</td>
<td>“implementing new functionality, Python is the still the language of choice ... [SciPy] full test suite passed with the PyPy JIT compiler at the 1.0 release point”</td>
<td>doi:10.1038/s41592-019-0686-2</td>
</tr>
</tbody>
</table>
• as a language, lacks support for multi-dimensional arrays or number-crunching facilities
- as a language, lacks support for multi-dimensional arrays or number-crunching facilities
- leaving it to be provided by external packages
• as a language, lacks support for multi-dimensional arrays or number-crunching facilities
• leaving it to be provided by external packages
• CPython interpreter is not the only Python implementation
as a language, lacks support for multi-dimensional arrays or number-crunching facilities

leaving it to be provided by external packages

CPython interpreter is not the only Python implementation

solutions exist streamlining just-in-time compilation for Python code
• as a language, lacks support for multi-dimensional arrays or number-crunching facilities

• leaving it to be provided by external packages

• CPython interpreter is not the only Python implementation

• solutions exist streamlining just-in-time compilation for Python code

• NumPy is not the only implementation of the NumPy API
- As a language, lacks support for multi-dimensional arrays or number-crunching facilities
- Leaving it to be provided by external packages
- CPython interpreter is not the only Python implementation
- Solutions exist streamlining just-in-time compilation for Python code
- NumPy is not the only implementation of the NumPy API
- Alternatives embedded in JIT/GPU frameworks leverage typing & concurrency
as a language, lacks support for multi-dimensional arrays or number-crunching facilities
leaving it to be provided by external packages
CPython interpreter is not the only Python implementation
solutions exist streamlining just-in-time compilation for Python code
NumPy is not the only implementation of the NumPy API
alternatives embedded in JIT/GPU frameworks leverage typing & concurrency
Python lets you glue (and package) together these technologies
JIT-compiled Python & NumPy API
Numba is an open source JIT compiler that translates a subset of Python and NumPy code into fast machine code ...
Numba is an open source JIT compiler that translates a subset of Python and NumPy code into fast machine code ...
NWP-related prototype problem

- hydrodynamics (transport)
- thermodynamics (phase changes)

**time evolution:**
- hydrodynamics (transport)
- thermodynamics (phase changes)
NWP-related prototype problem

- hydrodynamics (transport)
- thermodynamics (phase changes)

Time evolution:

numerical solution for transport-only PDE (2D)
NWP-related prototype problem

- hydrodynamics (transport)
- thermodynamics (phase changes)

time evolution:

numerical solution for transport-only PDE (2D)

(t/dt=157)
NWP-related prototype problem

- hydrodynamics (transport)
- thermodynamics (phase changes)

Numerical solution for transport-only PDE (2D)

(time evolution:)
- hydrodynamics (transport)
- thermodynamics (phase changes)
NWP-related prototype problem

- hydrodynamics (transport)
- thermodynamics (phase changes)

Time evolution:

numerical solution for transport-only PDE (2D)

\[(t/dt=471)\]
NWP-related prototype problem

- hydrodynamics (transport)
- thermodynamics (phase changes)

time evolution:

numerical solution for transport-only PDE (2D)

(t/dt=628)
example performance comparison: Bartman et al. 2022 (JOSS) doi:10.21105/joss.03896

(a) 10^0
(b) 10^-1

wall time per timestep [s]

PyMPDATA: Numba JIT disabled
libmpdata++
PyMPDATA: dynamic grid
PyMPDATA: static grid

number of threads
10 3

walltime per timestep [s]
libmpdata++ (3 threads)
PyMPDATA: dynamic grid (3 threads)

PyMPDATA \n \rightarrow \ Numba \ (\text{loop-based code, tricky for NumPy/CPython})
libmpdata++ \n \rightarrow \ Blitz++ \ (\text{OOP code; } 5\times \text{slower than F77 for small domains, on par for larger ones})
what if we need MPI?
Message Passing Interface

Message Passing Interface (MPI) is a standardized and portable message-passing standard designed to function on parallel computing architectures.\[1\] The MPI standard defines the syntax and semantics of library routines that are useful to a wide range of users writing portable message-passing programs in C, C++, and Fortran. There are several open-source MPI implementations, which fostered the development of a parallel software industry, and encouraged development of portable and scalable large-scale parallel applications.
despite the immense expansion of parallel computation both in the number of machines available as well as in the number of cores per parallel machine since then, no other parallel programming paradigm has replaced MPI – even though it is universally acknowledged that MPI is a rather crude way of programming these machines and that MPI might not be successful for machines much larger than the ones available today.

„despite the immense expansion of parallel computation both in the number of machines available as well as in the number of cores per parallel machine since then, no other parallel programming paradigm has replaced MPI – even though it is universally acknowledged that MPI is a rather crude way of programming these machines and that MPI might not be successful for machines much larger than the ones available today”
import numba
from mpi4py.MPI import COMM_WORLD

def number_crunching():
    rank = COMM_WORLD.Get_rank()

numba.njit(number_crunching)()
import numba
from mpi4py.MPI import COMM_WORLD

def number_crunching():
    rank = COMM_WORLD.Get_rank()

numba.njit(number_crunching)()
• Numba is the way to make Python code HPC ready
• MPI is the way to do distributed-memory parallelism for HPC
• Numba is the way to make Python code HPC ready
• MPI is the way to do distributed-memory parallelism for HPC
• ... so how come one cannot use MPI with Numba JIT-compiled code?!

\[ \text{python} + \text{Numba} + \text{MPI} = \text{?!} \]

1 https://github.com/numba/numba/issues/4115#issuecomment-642474009
2 https://bitbucket.org/mpi4py/mpi4py/issues/164
- Numba is the way to make Python code HPC ready
- MPI is the way to do distributed-memory parallelism for HPC
- ... so how come one cannot use MPI with Numba JIT-compiled code?!

![](python + Numba + MPI = ?)

- let's DuckDuckGo it...

---

1. https://github.com/numba/numba/issues/4115#issuecomment-642474009
2. https://bitbucket.org/mpi4py/mpi4py/issues/164
• Numba is the way to make Python code HPC ready
• MPI is the way to do distributed-memory parallelism for HPC
• ... so how come one cannot use MPI with Numba JIT-compiled code?!

[python] + [Numba] + [MPI] =

• let’s DuckDuckGo it... nope, Qwant...
● Numba is the way to make Python code HPC ready
● MPI is the way to do distributed-memory parallelism for HPC
● ... so how come one cannot use MPI with Numba JIT-compiled code?!

![Python + Numba + MPI = ?!](image)

● let’s DuckDuckGo it... nope, Qwant... nope, let’s Google...

---

1: [https://github.com/numba/numba/issues/4115#issuecomment-642474009](https://github.com/numba/numba/issues/4115#issuecomment-642474009)
2: [https://bitbucket.org/mpi4py/mpi4py/issues/164](https://bitbucket.org/mpi4py/mpi4py/issues/164)
- Numba is the way to make Python code HPC ready
- MPI is the way to do distributed-memory parallelism for HPC
- ... so how come one cannot use MPI with Numba JIT-compiled code?!

let's DuckDuckGo it... nope, Qwant... nope, let’s Google... nothing!

\[\text{python} + \text{Numba} + \text{MPI} = ?! \]

1. [https://github.com/numba/numba/issues/4115#issuecomment-642474009](https://github.com/numba/numba/issues/4115#issuecomment-642474009)
2. [https://bitbucket.org/mpi4py/mpi4py/issues/164](https://bitbucket.org/mpi4py/mpi4py/issues/164)
• Numba is the way to make Python code HPC ready
• MPI is the way to do distributed-memory parallelism for HPC
• ... so how come one cannot use MPI with Numba JIT-compiled code?!

python™ + Numba + MPI = ?!

• let’s DuckDuckGo it... nope, Qwant... nope, let’s Google... nothing!

• let’s ask on Numba¹ and mpi4py² forums (June 2020)...

  ~> "You will not be able to use mpi4py’s Cython code, it was not designed for such low-level usage..."

¹https://github.com/numba/numba/issues/4115#issuecomment-642474009
²https://bitbucket.org/mpi4py/mpi4py/issues/164
• Numba is the way to make Python code HPC ready
• MPI is the way to do distributed-memory parallelism for HPC
• ... so how come one cannot use MPI with Numba JIT-compiled code?!

let's DuckDuckGo it... nope, Qwant... nope, let's Google... nothing!

let's ask on Numba¹ and mpi4py² forums (June 2020)...

⇝ "You will not be able to use mpi4py’s Cython code, it was not designed for such low-level usage..."

• but it must be gluable!

¹https://github.com/numba/numba/issues/4115#issuecomment-642474009
²https://bitbucket.org/mpi4py/mpi4py/issues/164
• Numba is the way to make Python code HPC ready
• MPI is the way to do distributed-memory parallelism for HPC
• ... so how come one cannot use MPI with Numba JIT-compiled code?!

let’s DuckDuckGo it... nope, Qwant... nope, let’s Google... nothing!

let’s ask on Numba\(^1\) and mpi4py\(^2\) forums (June 2020)...

\[ \text{"You will not be able to use mpi4py’s Cython code, it was not designed for such low-level usage..."} \]

but it must be gluable!

30 months, 120 commits and 50 PRs from 5 contributors later... (unplanned side project!)

---
\(^1\) [https://github.com/numba/numba/issues/4115#issuecomment-642474009](https://github.com/numba/numba/issues/4115#issuecomment-642474009)
\(^2\) [https://bitbucket.org/mpi4py/mpi4py/issues/164](https://bitbucket.org/mpi4py/mpi4py/issues/164)
introducing: numba-mpi
Hello world example:

```python
import numba, numba_mpi, numpy

@numba.njit()
def hello():
    print(numba_mpi.rank())
    print(numba_mpi.size())
    src = numpy.array([1., 2., 3., 4., 5.])
    dst_tst = numpy.empty_like(src)
    if numba_mpi.rank() == 0:
        numba_mpi.send(src, dest=1, tag=11)
```

numba-mpi: implementation
"""MPI_Send() implementation"""

import ctypes

import numba

import numpy as np

from numba_mpi.common import _MPI_Comm_World_ptr, libmpi, send_recv_args

from numba_mpi.utils import _mpi_addr, _mpi_dtype

_MPI_Send = libmpi.MPI_Send
_MPI_Send.restype = ctypes.c_int
_MPI_Send.argtypes = send_recv_args

@numba.njit
def send(data, dest, tag):
    """wrapper for MPI_Send. Returns integer status code (0 == MPI_SUCCESS)"""
    data = np.ascontiguousarray(data)
    status = _MPI_Send(data.ctypes.data, data.size, _mpi_dtype(data), dest, tag, _mpi_addr(_MPI_Comm_World_ptr))
    # The following no-op prevents numba from too aggressive optimizations
    # This looks like a bug in numba (tested for version 0.55)

    data[0]
    # pylint: disable=pointless-statement

    return status
"""MPI_Send() implementation"""

import ctypes

import numba
import numpy as np

from numba_mpi.common import _MPI_Comm_World_ptr, libmpi, send_recv_args
from numba_mpi.utils import _mpi_addr, _mpi_dtype

_MPI_Send = libmpi.MPI_Send
_MPI_Send.restype = ctypes.c_int
_MPI_Send.argtypes = send_recv_args

@numba.njit
def send(data, dest, tag):
    """wrapper for MPI_Send. Returns integer status code (0 == MPI_SUCCESS)"""
    data = np.ascontiguousarray(data)
    status = _MPI_Send(
        data.ctypes.data,
        data.size,
        _mpi_dtype(data),
        dest,
        tag,
        _mpi_addr(_MPI_Comm_World_ptr),
    )
    data[0]  # pylint: disable=pointless-statement

    return status
"""MPI_Send() implementation"""

import ctypes

import numba
import numpy as np

from numba_mpi.common import _MPI_Comm_World_ptr, libmpi, send_recv_args
from numba_mpi.utils import _mpi_addr, _mpi_dtype

_MPI_Send = libmpi.MPI_Send
_MPI_Send.restype = ctypes.c_int
_MPI_Send.argtypes = send_recv_args

@numba.njit
def send(data, dest, tag):
    """wrapper for MPI_Send. Returns integer status code (0 == MPI_SUCCESS)"""
    data = np.ascontiguousarray(data)
    status = _MPI_Send(data.ctypes.data, data.size, _mpi_dtype(data),
                       dest, tag, _mpi_addr(_MPI_Comm_World_ptr))
    return status
import ctypes
import numba
import numpy as np

from numba_mpi.common import _MPI_Comm_World_ptr, libmpi, send_recv_args
from numba_mpi.utils import _mpi_addr, _mpi_dtype

_MPI_Send = libmpi.MPI_Send
_MPI_Send.restype = ctypes.c_int
_MPI_Send.argtypes = send_recv_args

@numba.njit
def send(data, dest, tag):
    """wrapper for MPI_Send. Returns integer status code (0 == MPI_SUCCESS)""
    data = np.ascontiguousarray(data)
"MPI_Send() implementation"
import ctypes

import numba
import numpy as np

from numba_mpi.common import _MPI_Comm_World_ptr, libmpi, send_recv_args
from numba_mpi.utils import _mpi_addr, _mpi_dtype

_MPI_Send = libmpi.MPI_Send
_MPI_Send.restype = ctypes.c_int
_MPI_Send.argtypes = send_recv_args

@numba.njit
def send(data, dest, tag):
    """wrapper for MPI_Send. Returns integer status code (0 == MPI_SUCCESS)""
    data = np.ascontiguousarray(data)
    status = _MPI_Send(
        data.ctypes.data,
        data.size,
        _mpi_dtype(data),
        dest,
        tag,
        _mpi_addr(_MPI_Comm_World_ptr),
    )
    return status
"""MPI_Send() implementation"""
import ctypes

import numba
import numpy as np

from numba_mpi.common import _MPI_Comm_World_ptr, libmpi, send_recv_args
from numba_mpi.utils import _mpi_addr, _mpi_dtype

_MPI_Send = libmpi.MPI_Send
_MPI_Send.restype = ctypes.c_int
_MPI_Send.argtypes = send_recv_args

@numba.njit
def send(data, dest, tag):
    """wrapper for MPI_Send. Returns integer status code (0 == MPI_SUCCESS)"""
    data = np.ascontiguousarray(data)
    status = _MPI_Send(
        data.ctypes.data,
        data.size,
        _mpi_dtype(data),
        dest,
        tag,
        _mpi_addr(_MPI_Comm_World_ptr),
    )
    # The following no-op prevents numba from too aggressive optimizations
    # This looks like a bug in numba (tested for version 0.55)
    # pylint: disable=pointless-statement
    data[0]
    return status
 2
 3
 4
 5
 6
 7
 8
 9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
numba-mpi: hacks :(  

------------------
48 @numba.extending.overload(_mpi_addr)
49 def _mpi_addr_njit(ptr):
50     def impl(ptr):
51         return numba.carray(
52             # pylint: disable-next=no-value-for-parameter
53             _address_as_void_pointer(ptr),
54             shape=(1,),
55             dtype=np.intp,
56             )[0]
57
58     return impl
59
60
61 # https://stackoverflow.com/questions/61509903/how-to-pass-array-pointer-to-numba-function
62 @numba.extending.intrinsic
63 def _address_as_void_pointer(_, src):
64     """returns a void pointer from a given memory address""
65     sig = types.voidptr(src)
66     def codegen(_, builder, ___, args):
67         return builder.inttoptr(args[0], cgutils.voidptr_t)
68     return sig, codegen
numba-mpi: CI, OSes, MPI impls
build:
  needs: [pylint, precommit, pdoc]
strategy:
matrix:
  platform: [ubuntu-latest, macos-latest, windows-latest]
  python-version: ['3.7', '3.8', '3.9', '3.10']
  mpi: [ 'mpich', 'openmpi', 'msmpi', 'intelmpi']
exclude:
  - platform: macos-latest
    mpi: msmpi
  - platform: macos-latest
    mpi: intelmpi
  - platform: ubuntu-latest
    mpi: msmpi
  - platform: windows-latest
    mpi: mpich
  - platform: windows-latest
    mpi: intelmpi
  - platform: ubuntu-latest
    python-version: 3.7
    mpi: mpich
  - platform: ubuntu-latest
    python-version: 3.8
    mpi: mpich
  - platform: ubuntu-latest
    python-version: 3.9
    mpi: mpich
runs-on: ${{ matrix.platform }}
steps:
  - uses: actions/checkout@v2
  - uses: actions/setup-python@v1
    with:
      python-version: ${{ matrix.python-version }}
  - uses: mpi4py/setup-mpi@v1
    with:
      mpi: ${{ matrix.mpi }}
  - run: pip install -e .
  - run: pip install pytest
  - run: python -W e -c "import mpi4py"
  - run: python -W e -c "import numba_mpi"
  - run: mpiexec -n 2 pytest -p no:unraisableexception -Ww
build:
  needs: [pylint, precommit, pdoc]
strategy:
  matrix:
    platform: [ubuntu-latest, macos-latest, windows-latest]
    python-version: ['3.7', '3.8', '3.9', '3.10']
    mpi: ['mpich', 'openmpi', 'msmpi', 'intelmpi']
  exclude:
    - platform: macos-latest
      mpi: msmpi
    - platform: ubuntu-latest
      mpi: intelmpi
    - platform: windows-latest
      mpi: mpich
    - platform: windows-latest
      mpi: openmpi
    - platform: windows-latest
      mpi: intelmpi

# https://github.com/numba-mpi/numba-mpi/issues/69
# (libfabric EFA provider is operating in a condition that could result in memory corruption or other system errors.)

runs-on: ${{ matrix.platform }}
steps:
  - uses: actions/checkout@v2
  - uses: actions/setup-python@v1
    with:
      python-version: ${{ matrix.python-version }}
  - uses: mpi4py/setup-mpi@v1
    with:
      mpi: ${{ matrix.mpi }}
  - run: pip install -e .
  - run: pip install pytest
  - run: python -We -c "import mpi4py"
  - run: python -We -c "import numba_mpi"
  - run: mpiexec -n 2 pytest -p no:unraisableexception -W e
build:
  needs: [pylint, precommit, pdoc]
strategy:
matrix:
  platform: [ubuntu-latest, macos-latest, windows-latest]
  python-version: ['3.7', '3.8', '3.9', '3.10']
  mpi: ['mpich', 'openmpi', 'msmpi', 'intelmpi']
exclude:
  - platform: macos-latest
    mpi: msmpi
  - platform: macos-latest
    mpi: intelmpi
  - platform: ubuntu-latest
    mpi: msmpi
  - platform: windows-latest
    mpi: mpich
  - platform: windows-latest
    mpi: intelmpi
  - platform: windows-latest
    mpi: openmpi
  - platform: windows-latest
    mpi: intelmpi

# https://github.com/numba-mpi/numba-mpi/issues/69
# (libfabric EFA provider is operating in a condition that
# could result in memory corruption or other system errors.)
  - platform: ubuntu-latest
    python-version: 3.7
    mpi: mpich
  - platform: ubuntu-latest
    python-version: 3.8
    mpi: mpich
  - platform: ubuntu-latest
    python-version: 3.9
    mpi: mpich

runs-on: ${{ matrix.platform }}
steps:
  - uses: actions/checkout@v2
  - uses: actions/setup-python@v1
    with:
      python-version: ${{ matrix.python-version }}
  - uses: mpi4py/setup-mpi@v1
    with:
      mpi: ${{ matrix.mpi }}
  - run: pip install -e .
  - run: pip install pytest
  - run: python -W E -c "import mpi4py"
  - run: python -W E -c "import numba_mpi"
  - run: mpiexec -n 2 pytest -p no:unraisableexception -W E

---

**kudos to mpi4py team**

for providing setup-mpi GitHub Action

this has saved us a lot of time!

**OSes and MPI implementations tested**

<table>
<thead>
<tr>
<th>MPI Implementation</th>
<th>Linux</th>
<th>macOS</th>
<th>Windows</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenMPI</td>
<td>+</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MPICH</td>
<td>+</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>IntelMPI</td>
<td>+</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MSmpi</td>
<td></td>
<td></td>
<td>+</td>
</tr>
</tbody>
</table>
build:
  needs: [pylint, precommit, pdoc]

strategy:
matrix:
  platform: [ubuntu-latest, macos-latest, windows-latest]
  python-version: ['3.7', '3.8', '3.9', '3.10']
  mpi: ['mpich', 'openmpi', 'msmpi', 'intelmpi']

exclude:
  - platform: macos-latest
    mpi: msmpi
  - platform: macos-latest
    mpi: intelmpi
  - platform: ubuntu-latest
    mpi: msmpi
  - platform: windows-latest
    mpi: mpich
  - platform: windows-latest
    mpi: openmpi
  - platform: windows-latest
    mpi: intelmpi

- platform: ubuntu-latest
  python-version: 3.7
  mpi: mpich
- platform: ubuntu-latest
  python-version: 3.8
  mpi: mpich
- platform: ubuntu-latest
  python-version: 3.9
  mpi: mpich

# kudos to mpi4py team
for providing setup-mpi GitHub Action
this has saved us a lot of time!

OSes and MPI implementations tested

<table>
<thead>
<tr>
<th></th>
<th>Linux</th>
<th>macOS</th>
<th>Windows</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenMPI</td>
<td>+</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>MPICH</td>
<td>+</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>IntelMPI</td>
<td>+</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MSMPI</td>
<td></td>
<td></td>
<td>+</td>
</tr>
</tbody>
</table>

caveat

MPICH v4 fails on Ubuntu for Python <3.10
"libfabric EFA provider is operating in a condition that could result in memory corruption" ↝ SIGABRT
numba-mpi: sample unit test
```python
import numba
import numpy as np
import pytest

import numba_mpi as mpi
from tests.common import MPI_SUCCESS, data_types
from tests.utils import get_random_array

@numba.njit()
def jit_bcast(data, root):
    return mpi.bcast(data, root)

@pytest.mark.parametrize("bcast", (jit_bcast.py_func, jit_bcast))
@pytest.mark.parametrize("data_type", data_types)
def test_bcast_np_array(data_type, bcast):
    root = 0
    data = np.empty(5, data_type).astype(dtype=data_type)
    datatobcast = get_random_array(5, data_type).astype(dtype=data_type)
    if mpi.rank() == root:
        data = datatobcast
    status = bcast(data, root)

    assert status == MPI_SUCCESS
    np.testing.assert_equal(data, datatobcast)
```

sample integration test scheme
Caveat using HDF5/MPI-IO (h5py) for concurrent file access from different MPI ranks... implies insurmountable trouble setting up CI test env on Windows (help welcome!)
caveat using HDF5/MPI-IO (h5py) for concurrent file access from different MPI ranks implies insurmountable trouble setting up CI test env on Windows (help welcome!)

(https://github.com/atmos-cloud-sim-uj/PySuperDropletLES/blob/main/tests/test_2d.py)
using HDF5/MPI-IO (h5py) for concurrent file access from different MPI ranks
... implies insurmountable trouble setting up CI test env on Windows (help welcome!)
py-pde: independent use case
py-pde is a Python package for solving partial differential equations (PDEs).
py-pde is a Python package for solving partial differential equations (PDEs).

Focus:

- Finite differencing and simple grids
- PDEs defined by mathematical expressions (supplied as strings)
**py-pde** is a Python package for solving partial differential equations (PDEs).

**Focus:**
- Finite differencing and simple grids
- PDEs defined by mathematical expressions (supplied as strings)

**Solution strategy:**
- Partition the grid onto different nodes using **numba-mpi**
- On each node, parse expressions using **sympy** and compile the result using **numba**
- Iterate the PDE, exchanging boundary information between nodes using **numba-mpi**
take-home messages
Python:

- common mismatch: language vs. ecosystem (e.g., arrays, number-crunching)
- has a range of **gluable HPC solutions** (JIT, GPU, multi-threading, MPI, ...)

numba-mpi:

- enables one to glue MPI with LLVM JIT-compiled Python code
- CI-tested on Linux, macOS, Windows; MPICH, OpenMPI, Intel MPI, & MS MPI
- developed aiming for 100% unit test coverage
- already a dependency of two PDE-solver projects: py-pde & PySuperDropletLES
Python:
- common mismatch: **language vs. ecosystem** (e.g., arrays, number-crunching)
- has a range of **gluable HPC solutions** (JIT, GPU, multi-threading, MPI, ...)

\[ \text{python}^\text{™} + \text{Numba} + \text{MPI} = \text{👍} \]

**numba-mpi**
- enables one to glue MPI with LLVM JIT-compiled Python code
- **CI-tested** on Linux, macOS, Windows; MPICH, OpenMPI, Intel MPI, & MS MPI
- developed aiming for **100% unit test coverage** (of the wrapping logic)
- already a dependency of two PDE-solver projects: py-pde & PySuperDropletLES
**numba-mpi sites:**

- [github.com/numba-mpi](https://github.com/numba-mpi) (contributors: slayoo, xann16, david-zwicker, Delcior, abulenok)
- [pypi.org/p/numba-mpi](https://pypi.org/p/numba-mpi)
- [anaconda.org/conda-forge/numba-mpi](https://anaconda.org/conda-forge/numba-mpi)

**Contributions welcome:**

- packaging
- h5py for Windows (HDF5) with support for MPI-IO
- MPICH ≥ 4.0 with Python < 3.10 on Linux (libfabric EFA provider issue)

**numba-mpi contribs:**

- adding support for other functions from the MPI API
- dropping dependency on mpi4py
- benchmarking performance

**Funding:**

(grant no. 2020/39/D/ST10/01220)
numba-mpi sites:
- github.com/numba-mpi (contributors: slayoo, xann16, david-zwicker, Delcior, abulenok)
- pypi.org/p/numba-mpi
- anaconda.org/conda-forge/numba-mpi

contributions welcome:
- packaging h5py for Windows (HDF5) with support for MPI-IO
- MPICH $\geq 4.0$ with Python $< 3.10$ on Linux (libfabric EFA provider issue)
- numba-mpi contribs:
  - logo
  - adding support for other functions from the MPI API
  - dropping dependency on mpi4py
  - benchmarking performance
numba-mpi sites:

- [github.com/numba-mpi](https://github.com/numba-mpi) (contributors: slayoo, xann16, david-zwicker, Delcior, abulenok)
- [pypi.org/p/numba-mpi](https://pypi.org/p/numba-mpi)
- [anaconda.org/conda-forge/numba-mpi](https://anaconda.org/conda-forge/numba-mpi)

contributions welcome:

- packaging **h5py for Windows** (HDF5) with support for MPI-I0
- **MPICH ≥ 4.0** with Python < 3.10 on Linux *(libfabric EFA provider issue)*

numba-mpi contribs:

- **logo**
- adding support for other functions from the MPI API
- dropping dependency on mpi4py
- benchmarking performance

funding:

[National Science Centre Poland](https://www.ncn.gov.pl/en)  
(grant no. 2020/39/D/ST10/01220)