Towards Python MPI for Artificial Intelligence and Deep Learning Research

Gregor von Laszewski, Fidel Leal, Shannon Kerr, Erin Seliger, Cooper Young, Agness Lungu

Abstract

This is a pandoc test. For more information, please contact: laszewski@gmail.com

Contents

1 Preface

Add a preface

- Who we are and how this activity came about.

- Notation. We can copy what I have done in other books, but keep it simple e.g. we do not have to worry about epub
  - code
  - use of `verbatim` inline and in block
  - use of LaTex formulas
  - use of markdown
    * spaces before and after headline, itemize lists, e.g. online editors do that not correctly
    * hyperlinks
    * citation

2 Overview

TODO: Gregor improves this section

After reflecting on the meeting I decided that in order to increase your python knowledge and to also lead you towards research we will be developing initially a tutorial that teaches students how to use MPI (Message Passing Interface). We do this with something that is called mpi4py. (later on we will use this to coordinate AI algorithms) We will develop multiple sections to this tutorial and each of you will each week work on a chapter so it can be redone by others. You can also work together and share each others ideas and thoughts openly as well as ask questions. We will do the following structured tasks (you will need to know what plagiarism is and when and how you need to cite):

2.1 1. Organizing Collaborative Research Teams

2.1.1 1.1 Communication (Gregor)

TODO Intro paragraph

2.1.1.1 1.1.1 Video Conferencing (Shannon, Fidel) Due to COVID restrictions and the summer semester, maintaing a virtual environment is esential to the lab group. Because of this, the team has resorted to video conferencing to keep group meetings intimate and proactive.

2.1.1.1.1 1.1.2 Google Meet (Shannon, Gregor)

- What is Google Meet?

  Google Meet is an online platform that allows groups of people to host video and audio conference calls. It has evolved from google hangout.

- What are good features?

  Some good features of Google meet include that it has a button for closed captioning or subtitles in the case that you're having trouble hearing the speaker.
  It is also compatible across devices so it is very easy and accessible for all group members. The sound quality is very good.

- What are not so good features?

  One disadvantage of Google meet is that there is not a feature where users can allow other group members to take control of the screen currently being shared. The best feature of google hangout to share each other's screens all the time has not been implemented. Instead just like other video conference meetings a single user shares the screen.

- Why do we use google meet and not zoom?

  Google has many other features such as google drive and google docs and google presentations. These tools can be used collaboratively. It is most natural to just use the same account name to also use google meets reducing the number of systems one uses for communication in a team.

2.1.1.1.2 1.1.3 Zoom (Fidel, Gregor)

- What is zoom?

  Zoom is a cloud-based communications platform that provides support for one-on-one, group meetings, and webinars.

- What are good features?

  Some of Zoom's most popular features include live chat, screen sharing, a whiteboard, and virtual reactions for meeting participants. Additionally, the ability to record meetings to the cloud or personal devices, create breakout rooms and seamlessly move
  between them, remotely control a participant's screen and the functionality to share files in the meeting make Zoom into an attractive collaborative tool. Zoom allows users to join any session through an established meeting URL. Participants do not need to be signed in or even have a Zoom account. Additionally, people joining from places with limited Internet access can call into the meeting's audio channel using dedicated telephone numbers.

- What are not so good features?

  Free account holders can host unlimited one-on-one meetings (meeting durations up to 24 hrs). IIn contrast to Google meet, there is a duration limit of 40 mins for meetings with three or more participants, making it difficult even for small teams to take advantage of the platform's features. Additionally, video quality can be unstable, and the overall platform performance can quickly deteriorate over limited bandwidth connections.

- How can someone take control of a remote desktop?

  To take control of a remote desktop, the remote user must activate screen sharing. Once the screen sharing is activated, we need to click the `View Options` dropdown menu (usually at the top of the screen) and click on `Request remote control`. The remote user will then get a prompt to approve the remote control request.

  Institutional accounts may have the remote control functionality disabled by their account's administrator. For further details, refer to the Zoom support pages.

2.1.1.1.3 1.1.4 Others (Agness)

- What do you find useful and a reserach team may consider? The answer could be nothing ;-)
- A shared google doc for documented steps we take when figuring stuff out. A shared database of information
- A dash board of weekly takes

2.1.1.2 1.2 Chat (Erin, Cooper) The main way in which our research team stays in touch over the course of the week is through Slack. We have a couple of different channels (general, python, random, etc.) we use to communicate. This allows everyone in the group to contribute to and benefit from the answers to specific questions/problems that others may have already run in to along the way.

2.1.1.2.1 1.2.1 Slack (Cooper)

- What is . . . ?

Slack is a communication software that is used for certain groups to send and receive messagese up to a group capacity or even directly with someone. * What are good features?

It offers a pleasant GUI Quality connection rate Easy to send photos

- What are not so good features?

Difficult to see all notifications If involved in a lot of group works, hard to manage all the workspaces.

2.1.2 Creating Text (Gregor)

TODO Introduction

2.1.2.1 Dictation (Windows 10 try out, Agness) Sometimes it is conveneinet to directly dictate the text for a manual or tutorial into an editor. On MacOS and Windows you will find useful tools for this. A Voice to text recorder may also help you in case you have a recoded video of yourself to generate transcripts. Disadvantag for a lot of non Native english speakers is that the accuraccy may be limited and that not using them leads to unacceptable results. Some of them can be trained. Try it out and let us know if they work for you and if you are satisfied with their accuracy. Examples include:

- Apple Dictation (free for Apple devices) You can directly dictate into various applicatipns that help you improving your text such as Grammarly and MS Word if you have installed them
- Windows 10 Voice Recognition (free for Windows users). You can directly record into MS Word so you get a free grammar checcker > Agnes: I tried windows voice recognition and it did not work on my computer
- Google Docs offers buidl in voice typing for dictating Google voice has a very good recognition success rate it is fast but has at times difficultis recognizing names. In case it makes mistakes, you will have to clean it up after dictation.
- TODO: can we dictate into MS word?

2.1.2.2 Grammar Checkers (Gregor, review Erin) When developing content for tutorials and documentation it is important to check their correctness with a grammar checcker. We have made the best experience with Grammarly followed by MS word. The best way to use them is to copy and paste small sections into them from your document and then check them. After you are satisfied copy the contents back to the original one, while overwriting the old text.

- Grammarly
  - Pro: works well, is available for free, the free version is good enough
  - Cons: not everything is done correct. In some text it shows false errors, but it's still very good
- Word
  - Pros: those familiar with word will know it
  - Cons: sometimes Grammarly performs better, Copying text back and forth can introduce errors when it comes to using quotes and other symbols. Thus you must check all symbols after you copy it into a markdown document.

Recommendation: use Grammarly

2.1.2.3 Editors You will need likely multiple two editors as part of your research activities. This is motivated by the fact that we do lots of development on yourl local machine, but also do remote development via terminal access to a remote computer that does not have a GUI. In case you only want t learn one editor to do all of this, just use emacs. We have listed bellow some editors and youmay want to chose

- emacs vs vi/vim for terminal editing
- pycharm vs MS code for fancyful python code development

There is a third option that we will use and that is jupyter which allows us to interactively develp python code. Jupyter is important as it is used by many data scientist due to its ad hoc interactive mode while programming. However, also pycharm and vscode prode options to view and run jupyter notebooks. However, not everything that works in jupyter is working on these platforms.

Recommendation: Use emacs and pyCharm

Terminal Editors

- emacs

  - Pro: terminal, established, very good markdown support, block format with ESC-q, keyboard shortcuts also used in bash and ther shells, has a python and markdown mode
  - Cons: some users have a hard time remembering keyboard shortcuts, the editor may get stuck in some unkown mode that you activated by accident. ALL this is easy to fix by remembering CRTL-X-s (save) and CTRL-G (get out of a strange mode)
- vi
  - use vim instead, vi is availabel on all Linux machines, but has rather archaic editing controls.
- vim
  - Pros: like vi, but with cursor control
  - Cons: often awkward to use

IDEs: * Pycharm * Pro: best Python editor * Cons: needs lots of resources, steep learning curve * Code (MS) * Pro: ofetn used on Raspberry * Cons: Pycharm seems to have more features from the start

2.1.3 1.1 Documentation Development with Markdown ()

2.1.3.1 1.1.2 Markdown () TODO

- What is markdown?
- Why do we use it?
- What a re good editors for markdon?
- Pointer to Gregor book
- Collaborative editing with HackMD.io

2.1.4 1.1 TODO

It is important to communicate quickly some tasks in the document that we write as a team. In order to do this we wuse the keyword TODO, followed usually by an explanation if needed. As a TODO can be hopefully resolved quickly it shoudl be able yp complete them in 1-2 hours. Any TODO that may take longer we also add to our GitHub project in order for it to be recorded and if we identify or delays in its execution we can assign additional team members to help on this tasks.

Once a team member has identified a TODO item, the team member can simple put his name behwind it, as well as the data and time so others know you work on it. You can aslo communicate on slack about the task you do if you run into issues or have questions.

2.1.5 1.2 Git and GitHub

Git is a distributed version control system to support working on project in teams while allowing different team members to contribute and to currate the contribution through reviews.

GitHub is a service offered for free with the limitation that the repositories shoudl not be larger than 1GB and the individual files must be smaller then 200MB. Github is very popular for OpenSource projects and through its free offering allows community building around OpenSource Projects.

2.1.5.1 Git from the commandline

Git can easily be installed on all platforms including

- macOS: You will need to install Xcode which includes not only git but user Linux programs such as Makefile
- Ubuntu: You can install it via `apt install git`
- Windows: You can install it via *Git Bash* which is distributed from https://git-scm.com/downloads. When installing read carefully the available options. We recommend you install a desktop shortcut.

2.1.6 Git form IDEs

Pycharm is one of the best editors for Python. It does provide build in support to interact with GitHub. This document here and I already went to some of your contributions and made improvements or two then OK

2.1.7 GitHub from a GUI

Some may fancy using a Graphical user interface to interact with GitHub. However, in many cases, the terminal access is simpler. However, if you like to browse the repositories and see the commit tree, these GUI interfaces are useful. Several such interfaces are available at:

- https://git-scm.com/downloads/guis

2.1.7.1 1.2.2 GitHub Commands

- What are the most important commands?
  - pull TODO
  - git add FILENAME TODO
  - commit -a TODO
  - push TODO
  - checkout branchanme TODO
- What is a branch and how do we use it form the commandline?

  TODO
- What if you committed something you did not want to and pushed it?

That is realy bad and you need to contact Gregor THis will take hours to fix, so be careful. So make sure you code does not contain passwords in plaintext.

Also do never ever use the git command `git add .` as that adds all files and you could have files that you do not want to commit. instead **always** use `git add FILENAME`, where FILENAME is the file you like to add

2.1.7.2 1.2.1 Task Management

- Our tasks in Github TODO
- Our issues in GitHub TODO

2.1.7.3 1.2.2 Code Management Our cod is managed as opens ource code in github.

- Our code in GitHub

  To check out use

  ```
  git clone git@github.com:cloudmesh/cloudmesh-mpi.git
  ```

  or

  ```
  git clone https://github.com/cloudmesh/cloudmesh-mpi.git
  ```

2.1.7.4 1.2.3 Github Actions We have not yet used them

TODO: provide description what they are

All: if yo see a TODO, and want to do it (e.g. have 1-2 hours time, put your name to it so others know you will work on it. Do not assign a TODO to you if you do not have time and will do it in a month from now, Research tasks need to be done immediately. HOwever we will also assign some longer term tasks to you so you can work ahead and in parallel, if your task is not done it will be assigned to some one else to mitigate that the time delay does not block the project.)

All: add tasks in github so we can assign todos and monitor progress

Gregor: Describe In detail how this is done

3 Introduction to MPI

- What is MPI and why do you want to use it
- What are some example MPI functionalities and usage patterns (send receive, embarrassing parallel

3.1 Installation of mpi4py on Windows

1. Look up msmpi and click the second link to download and install `msmpisetup.exe` and `msmpisdk.msi`
2. Open the system control panel
3. Click on Advanced system settings and then Environment Variables
4. Under the user variables box click on Path
5. Click New in order to add `C:\Program Files (x86)\Microsoft SDKs\MPI` and `C:\Program Files\Microsoft MPI\Bin` to Path
6. Close any open bash windows and then open a new one
7. Type the command `which mpiexec`
8. Install mpi4py with `pip install mpi4py`
9. In order to verify that the installation worked type `mpiexec -n 4 python mpi4py.bench helloworld`

3.2 Installing mpi4py in a Raspberry Pi

1. Activate our virtual environment: `source ~/ENV3/bin/activate`
2. Install Open MPI in your pi by entering the following command: `sudo apt-get install openmpi-bin`.

   After installation is complete you can check if it was successful by using `mpicc --showme:version`.
3. Enter `pip install mpi4py` to download and install mpi4py.
4. The installation can be tested with `mpiexec -n 4 python -m mpi4py.bench helloworld` (depending on the number of cores/nodes available to you, it may be necessary to reduce the number of copies that follow the -n option) In a PI4, the above test returned:

```
(ENV3) pi@red:~ $ mpiexec -n 4 python -m mpi4py.bench helloworld
Hello, World! I am process 0 of 4 on red.
Hello, World! I am process 1 of 4 on red.
Hello, World! I am process 2 of 4 on red.
Hello, World! I am process 3 of 4 on red.
```

3.3 Installing mpi4py in MacOS

A similar process can be followed to install mpi4py in MacOS. In this case, we can use Homebrew to get Open MPI by entering: `brew install open-mpi`.

Once Open MPI is working, steps 3 and 4 from the above pi4 installation can be followed in order to download and install mpi4py.

3.4 Hello World

To test if it works a build in test program is available.

To run it on on a single host with n cores (lest assume you have 2 cores), you can use:

```
mpiexec -n 4 python -m mpi4py.bench helloworld
Hello, World! I am process 0 of 5 on localhost.
Hello, World! I am process 1 of 5 on localhost.
Hello, World! I am process 2 of 5 on localhost.
Hello, World! I am process 3 of 5 on localhost.
```

Note that the messages can be in different order.

To run it on mulitple hosts with each having n cores please create a hostfile as follows:

TODO:

3.5 Machine file, hostfile, rankfile

Run sudo apt-get install -y python-mpi4py on all nodes.

Test the installation: mpiexec -n 5 python -m mpi4py helloworld

THIS CAN BE DONE BEST WITH CLOUDMESH

FIRTS TEST BY HAND

TODO: VERIFY

```
mpirun.openmpi \
  -np 2 \
  -machinefile /home/pi/mpi_testing/machinefile \
  python helloworld.py
```

The machinefile contains the ipaddresses

```
pi@192. ....
yout add teh ip addresses
```

TODO: learn about and evaluate and test if we can do

```
mpirun -r my_rankfile --report-bindings ...

Where the rankfile contains:
rank 0=compute17 slot=1:0
rank 1=compute17 slot=1:1
rank 2=compute18 slot=1:0
rank 3=compute18 slot=1:1
```

3.6 MPI Functionality examples

3.7 MPI Collective Communication functionality examples

3.7.1 Broadcast `comm.bcast()`

In this example, we broadcast a two-entry Python dictionary from a root process to the rest of the processes in our communicator group.

```
from mpi4py import MPI

# Set up the MPI Communicator
comm = MPI.COMM_WORLD

# Get the rank of the current process in the communicator group
rank = comm.Get_rank()

if rank == 0:  # Process with rank 0 gets the data to be broadcast
    data = {'size' : [1,3,8],
            'name' : ['disk1', 'disk2', 'disk3']}
else:  # Other processes' data is empty
    data = None

# Print data in each process
print("before broadcast, data on rank %d is "%comm.rank, data)

# Data from process with rank 0 is broadcast to other processes in our
# communicator group
data = comm.bcast(data, root=0)

# Print data in each process after broadcast
print("after broadcast, data on rank %d is "%comm.rank, data)
```

After running `mpiexec -n 4 python bcast.py` we get the following:

```
before broadcast, data on rank 0 is
  {'size': [1, 3, 8], 'name': ['disk1', 'disk2', 'disk3']}
before broadcast, data on rank 1 is  None
before broadcast, data on rank 2 is  None
before broadcast, data on rank 3 is  None
after broadcast, data on rank 0 is
  {'size': [1, 3, 8], 'name': ['disk1', 'disk2', 'disk3']}
after broadcast, data on rank 1 is
  {'size': [1, 3, 8], 'name': ['disk1', 'disk2', 'disk3']}
after broadcast, data on rank 2 is
  {'size': [1, 3, 8], 'name': ['disk1', 'disk2', 'disk3']}
after broadcast, data on rank 3 is
  {'size': [1, 3, 8], 'name': ['disk1', 'disk2', 'disk3']}
```

As we can see, the process with rank 1, received the data broadcast from rank 0.

3.7.1.1 Scatter `comm.scatter()`

In this example, with scatter the members of a list among the processes in the communicator group.

```
from mpi4py import MPI

# Communicator
comm = MPI.COMM_WORLD

# Number of processes in the communicator group
size = comm.Get_size()

# Get the rank of the current process in the communicator group
rank = comm.Get_rank()

# Process with rank 0 gets a list with the data to be scattered
if rank == 0:
    data = [(i+1)**2 for i in range(size)]
else:
    data = None

# Print data in each process
print("before scattering, data on rank %d is "%comm.rank, data)

# Scattering occurs
data = comm.scatter(data, root=0)

# Print data in each process after scattering
print("data for rank %d is "%comm.rank, data)
```

Executing `mpiexec -n 4 python scatter.py` yields:

```
before scattering, data on rank 2 is  None
before scattering, data on rank 3 is  None
before scattering, data on rank 0 is  [1, 4, 9, 16]
before scattering, data on rank 1 is  None
data for rank 2 is  9
data for rank 1 is  4
data for rank 3 is  16
data for rank 0 is  1
```

The members of the list from process 0 have been successfully scattered among the rest of the processes in the communicator group.

3.7.1.2 Gather `comm.gather()` In this example, data from each process in the communicator group is gathered in the process with rank 0.

```
from mpi4py import MPI

# Communicator
comm = MPI.COMM_WORLD

# Number of processes in the communicator group
size = comm.Get_size()

# Get the rank of the current process in the communicator group
rank = comm.Get_rank()

# Each process gets different data, depending on its rank number
data = (rank+1)**2

# Print data in each process
print("before gathering, data on rank %d is "%comm.rank, data)

# Gathering occurs
data = comm.gather(data, root=0)

# Process 0 prints out the gathered data, rest of the processes
# print their data as well
if rank == 0:
    print("after gathering, process 0's data is ", data)
else:
    print("after gathering, data in rank %d is "%comm.rank, data)
```

Executing `mpiexec -n 4 python gather.py` yields:

```
before gathering, data on rank 2 is  9
before gathering, data on rank 3 is  16
before gathering, data on rank 0 is  1
before gathering, data on rank 1 is  4
after gathering, data in rank 2 is  None
after gathering, data in rank 1 is  None
after gathering, data in rank 3 is  None
after gathering, process 0's data is  [1, 4, 9, 16]
```

The data from processes with rank `1` to `size - 1` have been successfully gathered in process 0.

3.7.1.3 Broadcasting buffer-like objects `comm.Bcast()` In this example, we broadcast a NumPy array from process 0 to the rest of the processes in the communicator group.

```
from mpi4py import MPI
import numpy as np

# Communicator
comm = MPI.COMM_WORLD

# Get the rank of the current process in the communicator group
rank = comm.Get_rank()

# Rank 0 gets a NumPy array containing values from 0 to 9
if rank == 0:
    data = np.arange(0,10,1, dtype='i')

# Rest of the processes get an empty buffer
else:
    data = np.zeros(10, dtype='i')

# Print data in each process
print("before broadcasting, data for rank %d is: "%comm.rank, data)

# Broadcast occurs
comm.Bcast(data, root=0)

# Print data in each process after broadcast
print("after broadcasting, data for rank %d is: "%comm.rank, data)
```

Executing `mpiexec -n 4 python npbcast.py` yields:

```
before broadcasting, data for rank 1 is:  [0 0 0 0 0 0 0 0 0 0]
before broadcasting, data for rank 2 is:  [0 0 0 0 0 0 0 0 0 0]
before broadcasting, data for rank 3 is:  [0 0 0 0 0 0 0 0 0 0]
before broadcasting, data for rank 0 is:  [0 1 2 3 4 5 6 7 8 9]
after broadcasting, data for rank 0 is:  [0 1 2 3 4 5 6 7 8 9]
after broadcasting, data for rank 2 is:  [0 1 2 3 4 5 6 7 8 9]
after broadcasting, data for rank 3 is:  [0 1 2 3 4 5 6 7 8 9]
after broadcasting, data for rank 1 is:  [0 1 2 3 4 5 6 7 8 9]
```

As we can see, the values in the array at process with rank 0 have been broadcast to the rest of the processes in the communicator group.

3.7.1.4 Scattering buffer-like objects `comm.Scatter()` In this example, we scatter a NumPy array among the processes in the communicator group.

This is fake

```
from mpi4py import MPI
import numpy as np

# Communicator
comm = MPI.COMM_WORLD

# Number of processes in the communicator group
size = comm.Get_size()

# Get the rank of the current process in the communicator group
rank = comm.Get_rank()

# Data to be sent
sendbuf = None

# Process with rank 0 populates sendbuf with a 2-D array,
# based on the number of processes in our communicator group
if rank == 0:
    sendbuf = np.zeros([size, 10], dtype='i')
    sendbuf.T[:,:] = range(size)

    # Print the content of sendbuf before scattering
    print('sendbuf in 0: ', sendbuf)

# Each process getd a buffer (initially containing just zeros)
# to store scattered data.
recvbuf = np.zeros(10, dtype='i')

# Print the content of recvbuf in each process before scattering
print('recvbuf in  %d: '%rank, recvbuf)

# Scattering occurs
comm.Scatter(sendbuf, recvbuf, root=0)

# Print the content of sendbuf in each process after scattering
print('Buffer in process %d contains: '%rank, recvbuf)
```

Executing `mpiexec -n 4 python npscatter.py` yields:

```
recvbuf in  1:  [0 0 0 0 0 0 0 0 0 0]
recvbuf in  2:  [0 0 0 0 0 0 0 0 0 0]
recvbuf in  3:  [0 0 0 0 0 0 0 0 0 0]
sendbuf in 0:  [[0 0 0 0 0 0 0 0 0 0]
                [1 1 1 1 1 1 1 1 1 1]
                [2 2 2 2 2 2 2 2 2 2]
                [3 3 3 3 3 3 3 3 3 3]]
recvbuf in  0:  [0 0 0 0 0 0 0 0 0 0]
Buffer in process 2 contains:  [2 2 2 2 2 2 2 2 2 2]
Buffer in process 0 contains:  [0 0 0 0 0 0 0 0 0 0]
Buffer in process 3 contains:  [3 3 3 3 3 3 3 3 3 3]
Buffer in process 1 contains:  [1 1 1 1 1 1 1 1 1 1]
```

As we can see, the values in the 2-D array at process with rank 0, have been scattered among all our processes in the communicator group, based on their rank value.

3.7.1.5 Gathering buffer-like objects `comm.Gather()` In this example, we gather a NumPy array from the processes in the communicator group into a 2-D array in process with rank 0.

```
from mpi4py import MPI
import numpy as np

# Communicator group
comm = MPI.COMM_WORLD

# Number of processes in the communicator group
size = comm.Get_size()

# Get the rank of the current process in the communicator group
rank = comm.Get_rank()

# Each process gets an array with data based on its rank.
sendbuf = np.zeros(10, dtype='i') + rank

# Print the data in sendbuf before gathering
print('Buffer in process %d before gathering: '%rank, sendbuf)

# Variable to store gathered data
recvbuf = None

# Process with rank 0 initializes recvbuf to a 2-D array conatining
# only zeros. The size of the array is determined by the number of
# processes in the communicator group
if rank == 0:
    recvbuf = np.zeros([size,10], dtype='i')

    # Print recvbuf
    print('recvbuf in process 0 before gathering: ', recvbuf)

# Gathering occurs
comm.Gather(sendbuf, recvbuf, root=0)

# Print recvbuf in process with rank 0 after gathering
if rank == 0:
        print('recvbuf in process 0 after gathering: \n', recvbuf)
```

Executing `mpiexec -n 4 python npgather.py` yields:

```
Buffer in process 2 before gathering:  [2 2 2 2 2 2 2 2 2 2]
Buffer in process 3 before gathering:  [3 3 3 3 3 3 3 3 3 3]
Buffer in process 0 before gathering:  [0 0 0 0 0 0 0 0 0 0]
Buffer in process 1 before gathering:  [1 1 1 1 1 1 1 1 1 1]
recvbuf in process 0 before gathering:
 [[0 0 0 0 0 0 0 0 0 0]
  [0 0 0 0 0 0 0 0 0 0]
  [0 0 0 0 0 0 0 0 0 0]
  [0 0 0 0 0 0 0 0 0 0]]
recvbuf in process 0 after gathering:
 [[0 0 0 0 0 0 0 0 0 0]
  [1 1 1 1 1 1 1 1 1 1]
  [2 2 2 2 2 2 2 2 2 2]
  [3 3 3 3 3 3 3 3 3 3]]
```

The values contained in the buffers from the different processes in the group have been gathered in the 2-D array in process with rank 0.

3.7.1.6 send receive

TODO

3.7.1.7 Dynamic Process Management

TODO

3.7.1.8 task processing (spwan, pull, . . .)

TODO: Cooper

3.7.1.8.1 Futures

https://mpi4py.readthedocs.io/en/stable/mpi4py.futures.html

3.7.1.9 examples for other collective communication methods

TODO

3.8 MPI-IO

TODO: Agnes

3.8.1 Collective I/O with NumPy arrays

TODO: Agnes

3.8.2 Non-contiguous Collective I/O with NumPy arrays and datatypes

TODO: Agnes

3.9 Monte Carlo calculation of Pi

TODO: Shannon, improve

TODO WHAT IS THE PROBLEM GOAL

We start with the Mathematical formulation of the Monte Carlo calulation of pi. For each quadrant of the unit square, the area is pi. Therefore, the ratio of the area outside of the circle is pi over four. With this in mind, we can use the Monte Carlo Method for the calculation of pi.

TODO: Drawing

TODO: HOW AND WHY DO WE NEED MULTIPLE COMPUTERS

3.9.1 Program

TODO: Shannon

- Example program to run Montecarlo on multiple hosts
- Benchmarking of the code
  - cloudmesh.common (not thread safe, but still can be used, research how to use it in multiple threads)
  - other strategies to benchmark, you research (only if really needed
- Use numba to speed up the code
  - describe how to install
- showcase basic usage on our monte carlo function
- display results with matplotlib

3.10 GPU Programming with MPI

Only possibly for someone with GPU (contact me if you do) Once we are finished with MPI we will use and look at python dask and other frameworks as well as rest services to interface with the mpi programs. This way we will be able to expose the cluster to anyone and they do not even know they use a cluster while exposing this as a single function ... (edited)

The github repo is used by all of you to have write access and contribute to the research effort easily and in parallel. You will get out of this as much as you put in. Thus it is important to set several dedicated hours aside (ideally each week) and contribute your work to others.

It is difficult to asses how long the above task takes as we just get started and we need to learn first how we work together as a team. If I were to do this alone it may take a week, but as you are less experienced it would likely take longer. However to decrease the time needed we can split up work and each of you will work on a dedicated topic (but you can still work in smaller teams if you desire). We will start assigning tasks in github once this is all set up.

Idea:

WE WILL NO LONGER USE HACKMD AS GITHUB IS BETTER FOR INTEGRATION WITH CODE

- tutorial about hackmd.io https://hackmd.io/t9SkKiSLR5qW9RUUA_CT-A
- Github vs hackmd

We will use initially hackmd so we avoid issues with github and we can learn github once we do more coding.

4 Appendix

4.1 Hardware of current students

- Fidel Leal,
  - Equipment
    - MacBook Pro 2015, 16GB RAM i7, SSD 512GB
    - PC, 64-bit, 8GB RAM, i5, SSD <240GB, speed>
      - Windows 10 Education
    - Editor: Pycharm, vim
- Shannon Kerr,
  - Equipment
    - Dell Inspiron, i5, 8GB, 1.6GHz 5482,
    - HDD 1TB 5.4K 6GB/s
    - Windows 64bit Home

        * Editor: Vim
- Erin Seliger
  - Equipment
    * Windows hp 2020, 16GB RAM, i7, 64-bit operating system
    * Windows Pro 64bit
    * Editor: Vim
- Cooper Young
  - Equipment
    * Dell Inspiron 7000, i7 2 Ghz, 16GB RAM, Intel Optane 512GB SSD
    * Windows 10 Education 64bit
    * Vim, Pycharm, Pico
- Agness Lungu
  - Equipment
    * Lenovo V570, 16GB RAM, intel(R) core(TM) i5-2430M, 64-bit operating system,
    * Windows 10 Education
    * editor: Vim, Pycharm