Improving Your Life with mpi4py

Or

How I Learned to Stop Worrying and Love Parallelization

Jacob Seiler

Overview

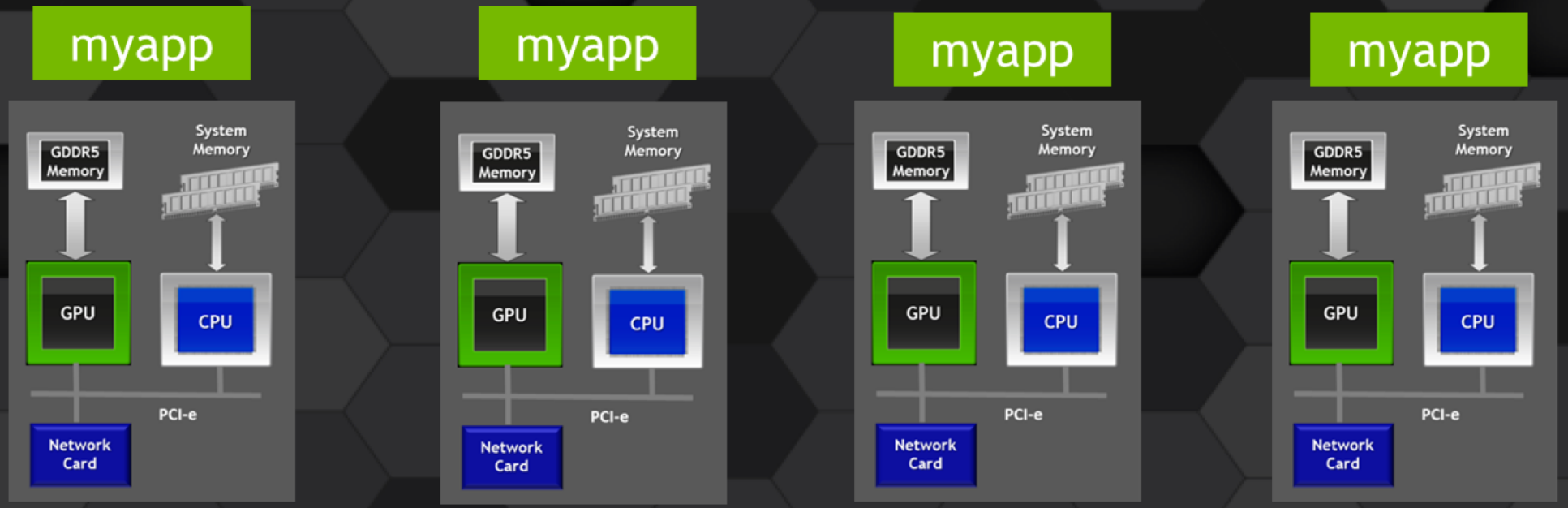
- What is MPI?
- When should I use MPI?
- Examples!

What is MPI?

- MPI stands for **M**essenger **P**assing **I**nterface.
- Provides a standard design for passing messages across multiple processors.
- Can be physically one machine (laptop) or separate discrete machines (supercomputer).
- mpi4py provides bindings for using MPI in Python. Same underlying machinery but has a few extra perks (e.g., communicating arrays across processors).

What is MPI?

```
mpirun -np 4 ./myapp <args>
```



When to use MPI?

- ~~Always.~~

- When multiple, similar operations are being performed (look for “for” loops).

- There are two broad classes of problems that can be solved with MPI:
  - Embarrassingly Parallel: In this case the operations are completely independent of each other. E.g., Running a program on 64 different initial conditions.
  - Communication Intensive: This class requires heavy communication between processors. E.g., N-body simulation where processors need to communicate their updated position at the end of each time step.

mpi4py: The Basics

Terminal Shell Edit View Window Help

...ktop/code-review/code-review/2018_05_25 — jseiler@farnarkle1:~/genesis/genesis/utils — vim hello_world.py

```
#!/usr/bin/env python

from __future__ import print_function

from mpi4py import MPI

comm = MPI.COMM_WORLD # Communicator for the processes.
rank = comm.Get_rank() # What number process is this one?
size = comm.Get_size() # How many processes in total are there?

print("I am Rank {0} of {1}: Hello World!".format(rank, size))
```

```
SA108-JSEIL-MAC:2018_05_25 100921091$ mpirun -np 4 python hello_world.py
I am Rank 2 of 4: Hello World!
I am Rank 3 of 4: Hello World!
I am Rank 0 of 4: Hello World!
I am Rank 1 of 4: Hello World!
SA108-JSEIL-MAC:2018_05_25 100921091$
```

mpi4py: Send/Receive

Terminal Shell Edit View Window Help

...sktop/code-review/code-review/2018_05_25 — jseiler@farnarkle1:~/genesis/genesis/utils — vim send_recv.py

```
#!/usr/bin/env python

from __future__ import print_function

import numpy as np
from mpi4py import MPI

comm = MPI.COMM_WORLD # Communicator for the processes.
rank = comm.Get_rank() # What number process is this one?
size = comm.Get_size() # How many processes in total are there?

if rank == 0: # Only rank 0 will have the data.
    data = np.arange(10)
    comm.send(data, dest=1, tag=11)
else: # All the other ranks will receive their data from rank 0.
    data = comm.recv(source=0, tag=11)

print("I am rank {0} and my data is {1}".format(rank, data))
```

```
SA108-JSEIL-MAC:2018_05_25 100921091$ mpirun -np 2 python send_recv.py
I am rank 0 and my data is [0 1 2 3 4 5 6 7 8 9]
I am rank 1 and my data is [0 1 2 3 4 5 6 7 8 9]
SA108-JSEIL-MAC:2018_05_25 100921091$
```

Send/Receive

- Be aware that ‘comm.send()’ and ‘comm.recv()’ are **blocking** communications. They will wait until the other process has received/sent the appropriate message.

- This means that programs can **hang** as they wait for a message that will never come!

- Test: See what happens when you change ‘tag=11’ to ‘tag=12’ for **either** ‘comm.send()’ or ‘comm.recv()’.

mpi4py: Send/Receive

```
# First determine the range of numbers this process will handle.
lower_range = N/size * rank
upper_range = N/size * (rank+1)
data = np.arange(lower_range, upper_range)
local_mean = np.mean(data)

print("I am rank {0} and my local mean is {1}".format(rank, local_mean))

# Then pass all the values to rank 0 to find the global mean.
if rank == 0:
    mean_array = np.empty(size)
    mean_array[0] = local_mean

    for i in range(1,size):
        mean_array[i] = comm.recv(source=i, tag=11)

    print("I am rank {0} and the mean from 0 to {1} is {2}".format(rank,
          N, np.mean(mean_array)))

else:
    comm.send(local_mean, dest=0, tag=11)
```

```
SA108-JSEIL-MAC:2018_05_25 100921091$ time mpirun -np 4 python3 send_recv_mean.py
I am rank 1 and my local mean is 374999999.5
I am rank 3 and my local mean is 874999999.5
I am rank 2 and my local mean is 624999999.5
I am rank 0 and my local mean is 124999999.5
I am rank 0 and the mean from 0 to 1000000000.0 is 499999999.5
```

mpi4py: Send/Receive

```
# First determine the range of numbers this process will handle.
lower_range = N/size * rank
```

For N = 1e9:

Time on 1 Processor: 5.922s

Time on 4 Processors: 2.311s

If you did this for 1000 time steps, MPI would save you 1 hour!

```
I am rank 0 and my local mean is 124999999.5
I am rank 0 and the mean from 0 to 1000000000.0 is 499999999.5
```

Manually Send/Receive is tedious…

- Manually managing all the send/receive calls can be tedious.
- Can be dangerous if you have many communications; need to ensure all send/receives are being answered.
- For collective operations (e.g., summing across processes) it's much much easier…

mpi4py: Send/Receive

```
# First determine the range of numbers this process will handle.
lower_range = int(N/size * rank)
upper_range = int(N/size * (rank+1))
data = np.arange(lower_range, upper_range)
local_mean = np.mean(data)

print("I am rank {0} and my local mean is {1}".format(rank, local_mean))

global_sum = comm.reduce(local_mean, op=MPI.SUM)
if rank == 0:
    print("I am rank {0} and the mean from 0 to {1} is {2}".format(rank,
          N, global_sum / size))
```

```
SA108-JSEIL-MAC:2018_05_25 100921091$ time mpirun -np 4 python send_recv_mean_easy.py
I am rank 3 and my local mean is 874999999.5
I am rank 2 and my local mean is 624999999.5
I am rank 1 and my local mean is 374999999.5
I am rank 0 and my local mean is 124999999.5
I am rank 0 and the mean from 0 to 1000000000.0 is 499999999.5
```

mpi4py: Send/Receive

```
N = 100
num_points = 10

data = np.array(random.sample(range(0, N), num_points))

print("I am rank {0} and my data is {1}".format(rank, data))

mean_data = np.zeros_like(data)

comm.Reduce([data, MPI.DOUBLE], [mean_data, MPI.DOUBLE], op=MPI.SUM, root = 0)

if rank == 0:
    print("I am rank {0} and the mean is {1}".format(rank, mean_data / size))
```

```
SA108-JSEIL-MAC:2018_05_25 100921091$ mpirun -np 4 python3 reduce_array.py
I am rank 0 and my data is [35 89 94 23 81 82 73 20 87 62]
I am rank 1 and my data is [87 94 43 78 52 98 84 88 73 66]
I am rank 2 and my data is [54 57 86 31 38 66 11 96 15 49]
I am rank 3 and my data is [52 20 99 59 49  7 73 58 78 23]
I am rank 0 and the mean is [57.   65.   80.5  47.75 55.   63.25 60.25 65.5  63.25 50.  ]
```

mpi4py: A more real example...

```
def my_example(datadir="./data", firstfile=0, lastfile=11):

    # If there aren't any data files, create some data.
    fname = "{0}/data_{1}".format(datadir, firstfile)
    if rank == 0 and not os.path.isfile(fname):
        print("Creating data files to read from.")
        create_data(datadir=datadir, firstfile=firstfile, lastfile=lastfile)
        print("Done!")

    comm.Barrier()  # Rank 0 may still be creating data so wait here.

    sum_thistask = 0  # Initialize.
    N_thistask = 0

    # Now each Task will get its own set of files to read in.
    # This loop ensures each file is only read one.
    for filenr in range(firstfile + rank, lastfile + 1, size):

        fname = "{0}/data_{1}".format(datadir, filenr)
        data_thisfile = np.loadtxt(fname)

        # Sum up the data from this file.
        sum_thistask += sum(data_thisfile)
        N_thistask += len(data_thisfile)

    # Then after all files have been read, reduce everything onto rank 0.
    global_sum = comm.reduce(sum_thistask, op=MPI.SUM)
    global_N = comm.reduce(N_thistask, op=MPI.SUM)

    print("I am rank {0} and I processed a total of {1} values.".format(rank,
          N_thistask))

    if rank == 0:
        print("I am rank {0} and {1} total values were processed with a sum "
              "of {2} and a mean of {3}".format(rank, global_N, global_sum,
                                                global_sum / global_N))
```

mpi4py: A more real example...

```
SA108-JSEIL-MAC:2018_05_25 100921091$ time mpirun -np 4 python3 all_examples.py
I am rank 3 and I processed a total of 1674225 values.
I am rank 1 and I processed a total of 1643769 values.
I am rank 2 and I processed a total of 1724562 values.
I am rank 0 and I processed a total of 1607437 values.
I am rank 0 and 6649993 total values were processed with a sum of 997048361483.0 and a mean of 149932.24225694675
```

```
def my_example(datadir="./data", firstfile=0, lastfile=11):
    comm.Barrier()  # Rank 0 may still be creating data so wait here.

    sum_thistask = 0  # Initialize.
    N_thistask = 0

    # Now each Task will get its own set of files to read in.
    # This loop ensures each file is only read one.
    for filenr in range(firstfile + rank, lastfile + 1, size):

        fname = "{0}/data_{1}".format(datadir, filenr)
        data_thisfile = np.loadtxt(fname)

        # Sum up the data from this file.
        sum_thistask += sum(data_thisfile)
        N_thistask += len(data_thisfile)

    # Then after all files have been read, reduce everything onto rank 0.
    global_sum = comm.reduce(sum_thistask, op=MPI.SUM)
    global_N = comm.reduce(N_thistask, op=MPI.SUM)

    print("I am rank {0} and I processed a total of {1} values.".format(rank,
          N_thistask))

    if rank == 0:
        print("I am rank {0} and {1} total values were processed with a sum "
              "of {2} and a mean of {3}".format(rank, global_N, global_sum,
                                                global_sum / global_N))
```

mpi4py: A more real example...

```
def my_example(datadir="./data", firstfile=0, lastfile=11):
```

```
SA108-JSEIL-MAC:2018_05_25 100921091$ time mpirun -np 4 python3 all_examples.py
I am rank 3 and I processed a total of 1674225 values.
I am
I am
I am
I am
```

For N = ~6e6 data points across 12 files:

Time on 1 Processor: 24.869s

Time on 4 Processors: 7.555s

```
print( I am rank {0} and {1} total values were processed with a sum
    "of {2} and a mean of {3}".format(rank, global_N, global_sum,
                                      global_sum / global_N))
```

Summary

- MPI can vastly speed your code up, allowing you to perform many operations simultaneously.
- It's critical to identify **if** your code can be parallelized, and the level of effort required (e.g., embarrassingly parallel vs intensive communication).
- Give consideration to time spent parallelizing vs speedup. Some codes are only ever run once; could the time spent parallelizing be spent actually running the code?

Read the Docs!

- http://mpi4py.scipy.org/docs/usrman/tutorial.html

- Basically a summary of everything I showed!