BAG OF TASKS PARALLELISM

Timothy H. Kaiser, Ph.D.
tkaiser@mines.edu

Examples at:

- http://hpc.mines.edu/examples/

To just get mpi4py examples:

```
mkdir examples
cd examples
curl http://hpc.mines.edu/examples/examples/mpi/mpi4py/mpi4py.tgz | tar -xz
```

Data set for this program:

```
curl http://hpc.mines.edu/examples/laser.tgz | tar -xz
```

What if you have?

- A bunch of tasks to do
- They are all independent
- Similar, maybe just different input files
- Often called bag of task parallelism or embarrassingly parallel

SIMPLE.PY

- Starts MPI
- Splits the processors into two groups/communicators 0-(N-2) and (N-1)
- Processor (N-1) waits for “ready” from other processors, then sends work
- Rest of processors loop
  - send requests for work
  - do work (in this case generate plots)
  - send results

BAG OF TASKS

simple.py

This is a bag-of-tasks program. We define a manager task that distributes work to workers. Actually, the workers request input data. The manager sits in a loop calling Iprobe waiting for requests for work.
In this case the manager reads input. The input is a list of file names. It will send a entry from the list as requested. When the worker is done processing it will request a new file name from the manager. This continues until the manager runs out of files to process. The manager subroutine is just "manager"
The worker subroutine is "worker". It receives file names form the manager.
The files in this case are outputs from an optics program tracking a laser beam as it propagates through the atmosphere. The workers read in the data and then create an image of the data by calling the routine mkview.plotit. This should worker with arbitrary 2d files except the size in mkview.plotit is currently hard coded to 64 x 64.
We use the call to "Split" to create a separate communicator for the workers. This is not important in this example but could be if you wanted multiple workers to work together.

- MPI.COMM_WORLD
- Get_rank()
- Get_size()
- comm.gather
- comm.Send()
- comm.Recv()
- MPI.Status()
- comm.Iprobe()
- gotfrom=status.source
- MPI.Get_processor_name()
- MPI_COMM_WORLD.barrier()
- MPI.Finalize

To get the data...

```
curl http://hpc.mines.edu/examples/laser.tgz | tar -xz
```

- Do initialization

```
#!/usr/bin/env python
from mpi4py import MPI
import numpy
global numnodes,myid,mpi_err
global mpi_root
import sys
mpi_root=0
. . .
. . .
. . .
. . .
#
if __name__ == '__main__':
# do init
   global numnodes,myid,mpi_err
   comm=MPI.COMM_WORLD
   myid=comm.Get_rank()
   numnodes=comm.Get_size()
   name = MPI.Get_processor_name()
   print("hello from %d of %d on %s" % (myid,numnodes,name))
```

- Create a communicator that contains everyone but the one process

```
   num_used=numnodes-1
   mannum=0;
   MPI_COMM_WORLD=MPI.COMM_WORLD
   if(myid == mannum):
      group=0
   else:
      group=1
# Split will create a set of communicators.  All of the
# tasks with the same value of group will be in the same
# communicator.  In this case we get two sets one for the
# manager and one for the workers.  The manager's version
# of the communicator is not used.

   DEFINED_COMM=MPI_COMM_WORLD.Split(group,myid)
#
   new_id=DEFINED_COMM.Get_rank()
   worker_size=DEFINED_COMM.Get_size()
   print("old id = %d   new id = %d   worker size = %d" %
(myid,new_id,worker_size))
#
```

- Manager is not part of "worker group" so she manages

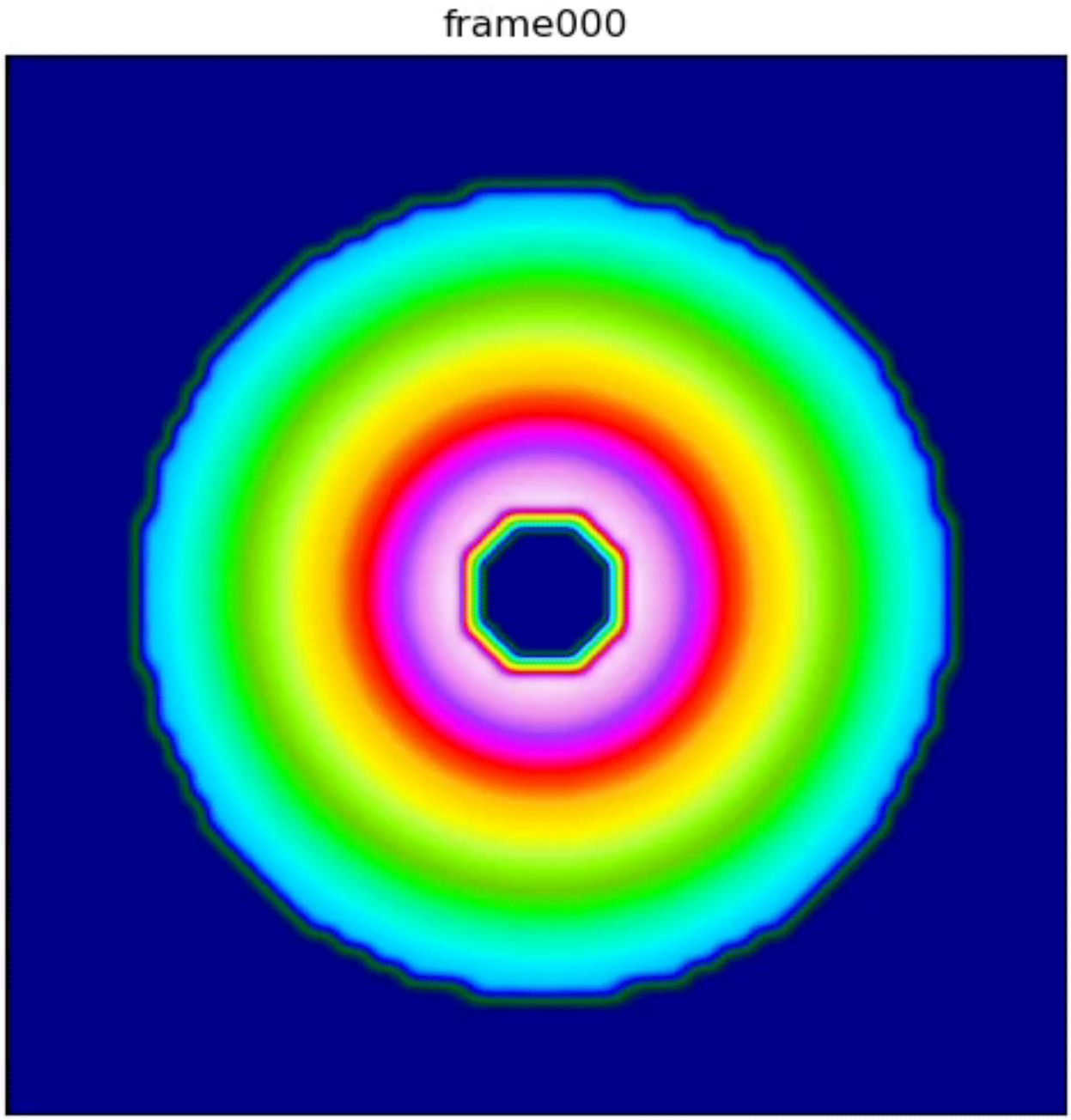
```
   if(group == 0):
      todo=1000
# if not part of the new group do management. #
      manager(num_used,todo)
      print("manager finished")
      #mpi_err =  MPI_Barrier(MPI_COMM_WORLD)
      MPI_COMM_WORLD.barrier()
      MPI.Finalize()
   else:
# part of the new group do work. #
      mannum=0;
      ts=MPI.Wtime()
      idid=worker(DEFINED_COMM,mannum)
      te=MPI.Wtime()
      print("worker (%d,%d) finished  did %d tasks in %8.2f
seconds" %(myid,new_id,idid,te-ts))
      MPI_COMM_WORLD.barrier()
      MPI.Finalize()
```

- Workers tell manager they are ready
- Get work
- Do work
- Send Results

```
def worker(THE_COMM_WORLD,managerid):
   import mkview
   x=0
   comm=MPI.COMM_WORLD
   send_msg = numpy.arange(1, dtype='i')
   recv_msg = numpy.zeros_like(send_msg)
   ic=0
   while(True) :
# send message says I am ready for data #
      send_msg[0]=x
      comm.Send([send_msg, MPI.INT], dest=managerid, tag=1234)
# get a message from the manager #
      buffer=numpy.array((1), dtype=str)
      comm.Recv([buffer,MPI.CHAR], source=managerid, tag=2345)
#     print(buffer)
      x=str(buffer).split()
      fname=x[0]
      x=int(x[1])
      if(x < 0):
         return ic
      print(THE_COMM_WORLD.Get_rank(),fname,x)
      ic=ic+1
      mkview.plotit(fname,x)
#
```

```
def manager(num_used,TODO):
    global numnodes,myid,mpi_err
    global mpi_root
    comm=MPI.COMM_WORLD
    send_msg = numpy.arange(1, dtype='i')
    recv_msg = numpy.zeros_like(send_msg)
    status = MPI.Status()
# our "data"
# Our worker is expecting a single word  followed
# by a manager appended integer
    data=sys.stdin.readlines()
    todo=len(data)
# counters
    igot=0
    isent=0
    while(isent < todo):
# wait for a request for work #
        flag=comm.Iprobe(source=MPI.ANY_SOURCE, tag=MPI.ANY_TAG,status=status)
        if(flag):
# where is it comming from #
            gotfrom=status.source
            sendto=gotfrom
            comm.Recv([recv_msg, MPI.INT], source=gotfrom, tag=1234)
            x=recv_msg[0]
            print("worker %d sent %d" % (gotfrom,x))
            if(x > -1):
                igot=igot+1
                print("igot "+str(igot))
            if(isent < TODO):
# send real data #
                d=data[isent]
                d=d.strip()
                send_msg=numpy.array([d+" "+str(isent)], dtype=str)
                comm.Send([send_msg, MPI.CHAR], dest=sendto, tag=2345)
                isent=isent+1
# tell everyone to quit #
    for i in range(1,num_used+1):
        send_msg=numpy.array(["stop -1000"], dtype=str)
        comm.Send([send_msg, MPI.CHAR], dest=i, tag=2345)
    return None
```

- Gets input from stdin
- Manager waits for ready
- Sends work
- Tells everyone to quit when work is finished


frame000