HW07 - Global Sum
CS5500

Chelsi Gupta

October 13, 2018

1 Implementation Variations

I tried 4 different variations for sending the global_sum(of numbers at all processes) to all the processes in the cluster. These are:

1.1 global_sum_ring1

In this variant I generated a random number at each process and then circulated those numbers by passing the numbers from rank to (rank+1)%size one by one in a ring fashion. So each process(except 0) will receive data from its rank - 1, add it to its global sum and then pass the global sum to rank + 1. 0 starts by sending the data and at last when 0 receives the global sum then it sends it to all the other processes.

Compile and Run: 'mpirun –oversubscribe -np 8 python global_sum_ring1.py'

Time: The maximum time taken for one amongst the 8 processes was 0.083 seconds.

1.1.1 Code

```
import time
start_time = time.time()
from mpi4py import MPI
import math
import random
comm = MPI.COMM_WORLD

size = comm.Get_size()
rank = comm.Get_rank()
num = random.randint(1,5)
global_sum = num

if rank!=0:
    rdata = comm.recv(source=rank-1)
    global_sum += rdata
```

```
comm.send(global_sum, dest=(rank+1)%size)

if rank==0:
    rdata = comm.recv(source=size-1)
    for i in range(1,size,1):
        comm.send(rdata, dest=i)

if rank != 0:
    rdata = comm.recv(source=0)

print ('I am process ',rank,' had: ',num," global sum: ",rdata,' time: ',"  %s seconds " % (tim
```

1.1.2 Output

Figure 1: Ring 1

1.2 global_sum_ring2

In this variant I generated a random number at each process and then circulated those numbers by passing the numbers from rank to (rank+1)%size in a ring fashion. So each process(except 0) will receive data from its rank - 1, add it to its global sum and then pass the previously received data to rank + 1,the process goes on and after 'size -1' steps like these each process has the global sum. Here each process has to maintain two data with them one is the global sum and other is the data received. All the processes send at once here and do not have to wait for receiving data from their previous process.

Compile and Run: 'mpirun –oversubscribe -np 8 python global_sum_ring2.py'

Time: The maximum time taken for one amongst the 8 processes was 0.085 seconds.

1.2.1 Code

```
import time
start_time = time.time()
from mpi4py import MPI
import math
import random
comm = MPI.COMM_WORLD

size = comm.Get_size()
rank = comm.Get_rank()
num = random.randint(1,5)
global_sum = num
rdata = num

for i in range(1,size,1):
    comm.send(rdata,dest=(rank+1)%size)
    if rank != 0:
        rdata = comm.recv(source=rank-1)
    else:
        rdata = comm.recv(source=size-1)
    global_sum += rdata

print ('I am process ',rank,' had: ',num," global sum: ",global_sum,' time: ',"  %s seconds " %
```

1.2.2 Output

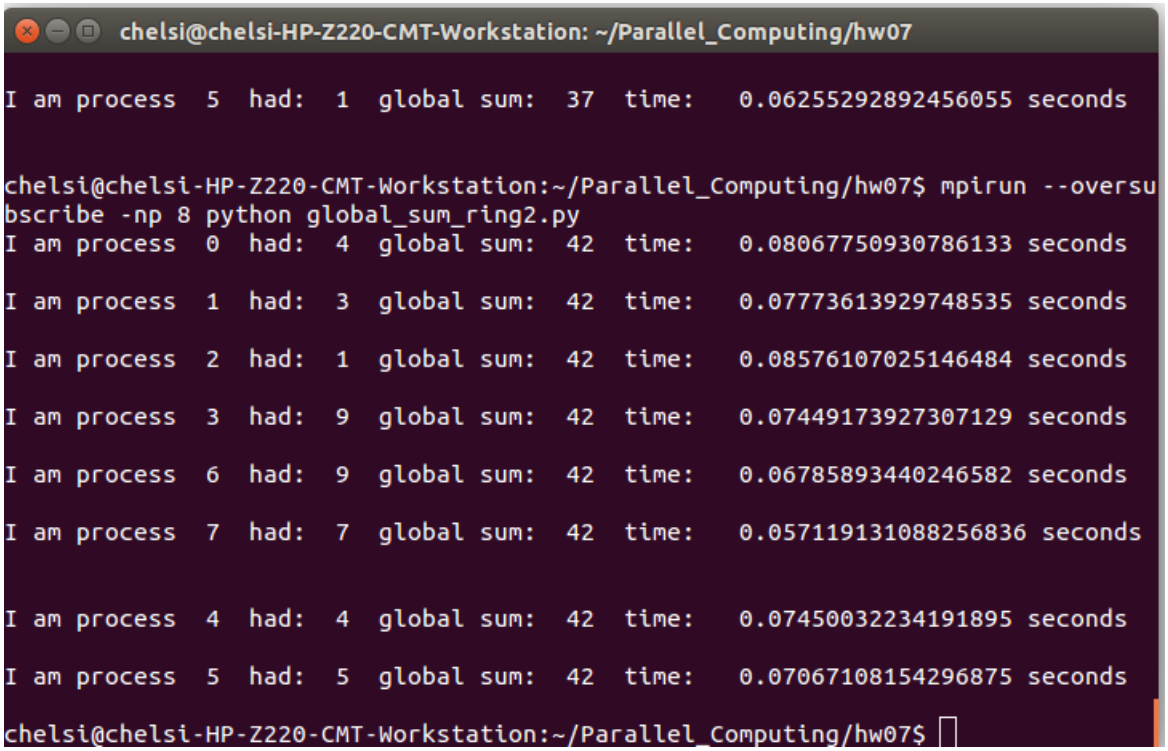

Figure 2: Ring 2

1.3 global_sum_cube1

In this variant I generated a random number at each process and made the processes to exchange data through cube exchange. Cube function is used to determine which process we have to exchange the data with. After determining the destination, we check if the rank of dest is greater than the current rank then the dest process sends the data and the current process receives it and vice-versa. So in this way at the end process 0 will have the global sum and we follow the reverse process by sending the data to the process which has a greater rank then the current process.

Compile and Run: 'mpirun –oversubscribe -np 8 python global_sum_cube1.py'

Time: The maximum time taken for one amongst the 8 processes was 0.126 seconds.

1.3.1 Code

```
import time
```

```
start_time = time.time()
from mpi4py import MPI
import math
import random
comm = MPI.COMM_WORLD

size = comm.Get_size()
rank = comm.Get_rank()
num = random.randint(1,5)
rdata = 0


def cube(i):
        size = comm.Get_size()
        rank = comm.Get_rank()
        mask = 1
        mask = mask << i
        dest = rank ^ mask
        #dest is the process we want to exchange the data with
        return dest


global_sum = num
k = int(math.log2(size))

for i in range(0,k,1):
    dest = cube(i)
    if dest < rank:
       comm.send(global_sum, dest=dest)
    else:
       rdata = comm.recv(source=dest)
       global_sum += rdata

for i in range(k-1,-1,-1):
    dest = cube(i)
    if dest > rank:
       comm.send(global_sum, dest=dest)
    else:
       rdata = comm.recv(source=dest)
       global_sum = rdata

print ('I am process ',rank,' had: ',num," global sum: ",global_sum,' time: '," %s seconds " %
```

1.3.2 Output

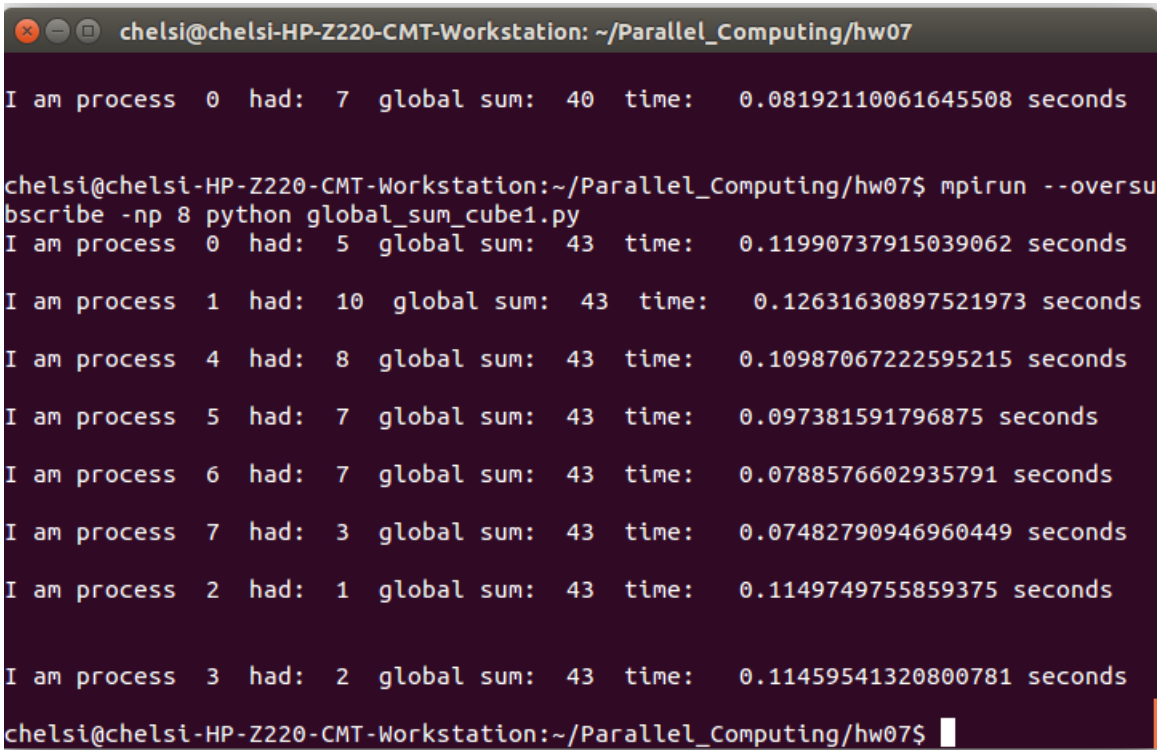

Figure 3: Cube 1

1.4 global_sum_cube1

In this variant I generated a random number at each process and made the processes to exchange data through cube exchange. Cube function is used to determine which process we have to exchange the data with. Through cube exchange each process will send and receive data from its cube exchanger. The maximum cube exchange we do here is log(size-1). After the log(size-1)th cube exchange each process will have the global sum.

Compile and Run: 'mpirun –oversubscribe -np 8 python global_sum_cube2.py'

Time: The maximum time taken for one amongst the 8 processes was 0.087 seconds.

1.4.1 Code

```
import time
start_time = time.time()
from mpi4py import MPI
import math
```

```
import random
comm = MPI.COMM_WORLD

size = comm.Get_size()
rank = comm.Get_rank()
num = random.randint(1,5)


def cube(i, sendData):
        size = comm.Get_size()
        rank = comm.Get_rank()
        mask = 1
        mask = mask << i
        dest = rank ^ mask
        #dest is the process we want to exchange the data with
        comm.send(sendData, dest=dest)
        recvData = comm.recv(source=dest)
        return recvData


global_sum = num

k = int(math.log2(size))
for i in range(0,k,1):
    rdata = cube(i,global_sum)
    global_sum += rdata

print ('I am process ',rank,' had: ',num," global sum: ",global_sum,' time: '," %s seconds " %
```

1.4.2 Output

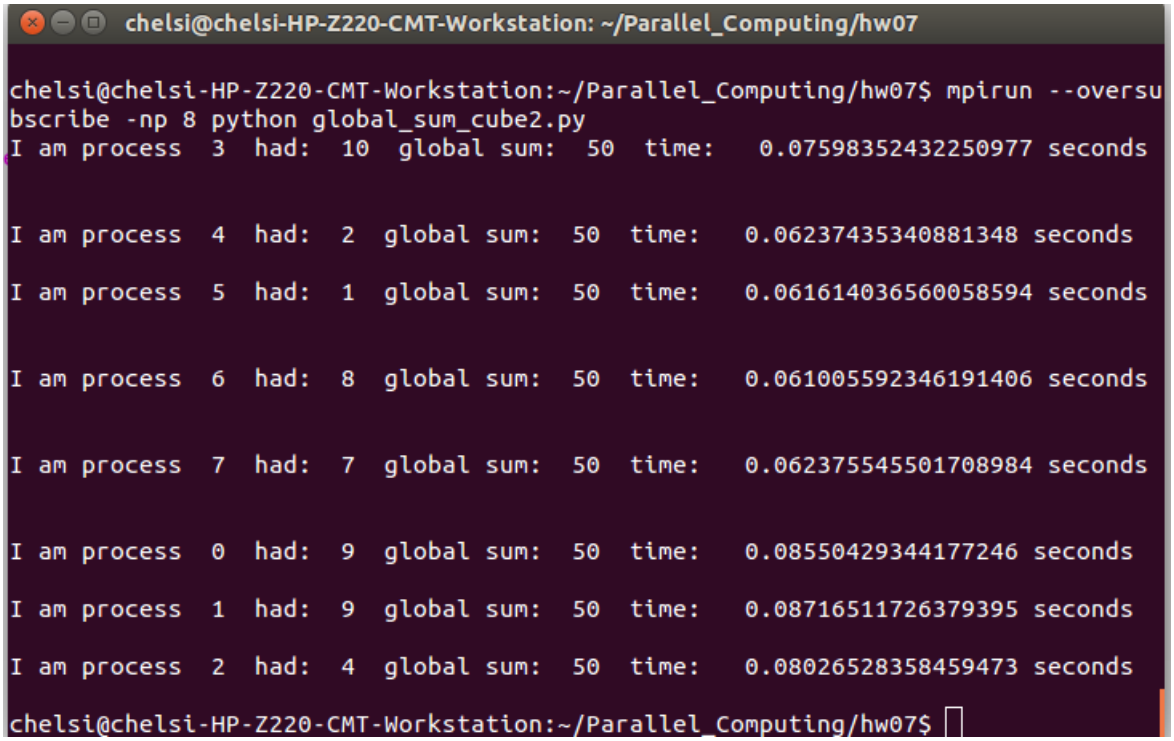

Figure 4: Cube 2