Report for HPC Project 2

Name: Haojian Jin

Summary:

Source code Structure:

```
|---MPI.Helloworld
|             |---1. simplehelloworld
|             |                   |---hpc_mpi_simple_helloworld
|             |                   |---hpc_mpi_simple_helloworld.c
|             |                   |---Makefile
|             |---2. helloworldwithring
|             |                     |---hpc_mpi_helloworldwitharing
|             |                     |---hpc_mpi_helloworldwitharing.c
|             |                     |---Makefile
|             |---4. helloworldtime
|             |                 |---hpc_mpi_helloworld_with3roundtrips
|             |                 |---hpc_mpi_helloworld_with3roundtrips.c
|             |                 |---Makefile
|             |---5. helloworld_analyze
|             |                     |---hpc_mpi_helloworld_with3roundtrips.c
|             |                     |---Makefile
|             |                     |---hpc_mpi_helloworld_with3roundtrips
|             |                     |---mpihosts
|---MPI.Mandelbrot
|             |---domain_composition
|             |                  |---display.c
|             |                  |---display.h
|             |                  |---display.o
|             |                  |---Makefile
|             |                  |---mandel.c~
|             |                  |---mandel.c
|             |                  |---mandel.o
|             |                  |---p2-mandel
|             |---processor_farm
|             |              |---display.c
|             |              |---display.h
|             |              |---display.o
|             |              |---Makefile
|             |              |---mandel.c~
|             |              |---mandel.c
|             |              |---mandel.o
|             |              |---p2-mandel
```

Usage:

```
make
mpirun –np x ./executable file.
```

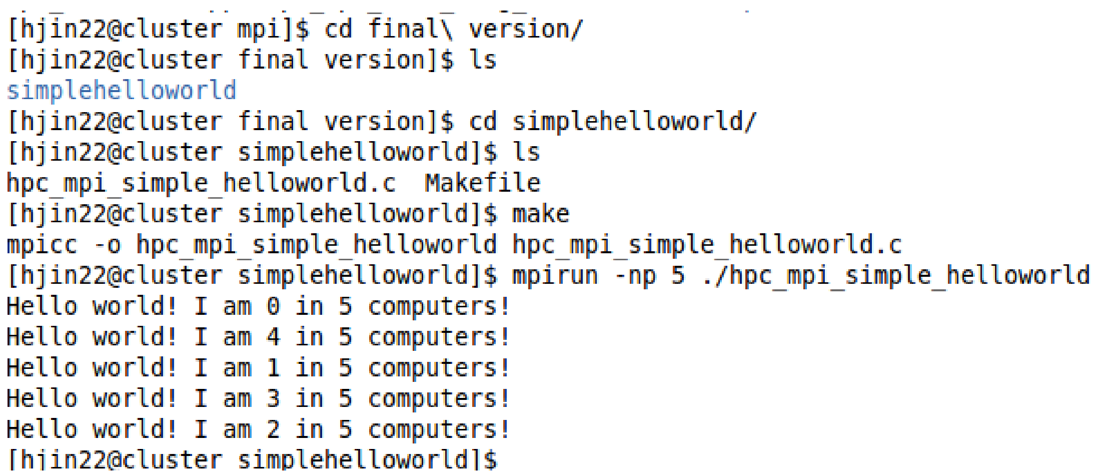
PS: Make sure remove the former executable files first.

Report for Question 1: An MPI Hello, world.

1. ***Basic "Hello, World in MPI version".***

   The source code is in "**1. simplehelloworld**" folder.

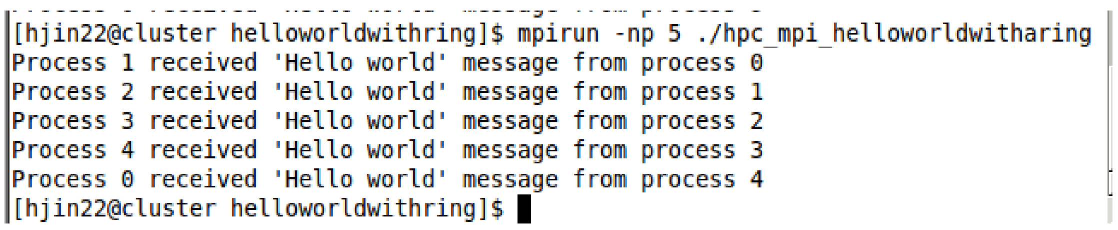
   Here is a screen shot of running result.



```
[hjin22@cluster mpi]$ cd final\ version/
[hjin22@cluster final version]$ ls
simplehelloworld
[hjin22@cluster final version]$ cd simplehelloworld/
[hjin22@cluster simplehelloworld]$ ls
hpc_mpi_simple_helloworld.c  Makefile
[hjin22@cluster simplehelloworld]$ make
mpicc -o hpc_mpi_simple_helloworld hpc_mpi_simple_helloworld.c
[hjin22@cluster simplehelloworld]$ mpirun -np 5 ./hpc_mpi_simple_helloworld
Hello world! I am 0 in 5 computers!
Hello world! I am 4 in 5 computers!
Hello world! I am 1 in 5 computers!
Hello world! I am 3 in 5 computers!
Hello world! I am 2 in 5 computers!
[hjin22@cluster simplehelloworld]$
```

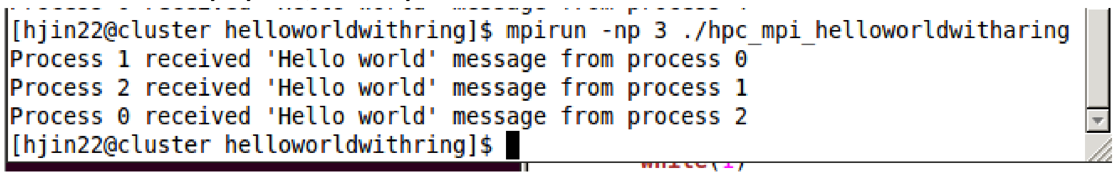
2. ***Modify the program to send the message around a ring of process.***

   The source code is in folder "**2. helloworldwithring**".



```
[hjin22@cluster helloworldwithring]$ mpirun -np 5 ./hpc_mpi_helloworldwitharing
Process 1 received 'Hello world' message from process 0
Process 2 received 'Hello world' message from process 1
Process 3 received 'Hello world' message from process 2
Process 4 received 'Hello world' message from process 3
Process 0 received 'Hello world' message from process 4
[hjin22@cluster helloworldwithring]$
```

3. ***More test with –np option, when np = 3;***



```
[hjin22@cluster helloworldwithring]$ mpirun -np 3 ./hpc_mpi_helloworldwitharing
Process 1 received 'Hello world' message from process 0
Process 2 received 'Hello world' message from process 1
Process 0 received 'Hello world' message from process 2
[hjin22@cluster helloworldwithring]$
```

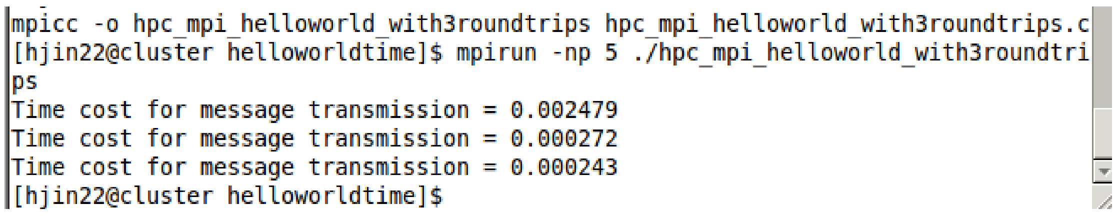
   More tests could be acquired by command:

   make

   mpirun -np 6 ./hpc_mpi_helloworldwitharing

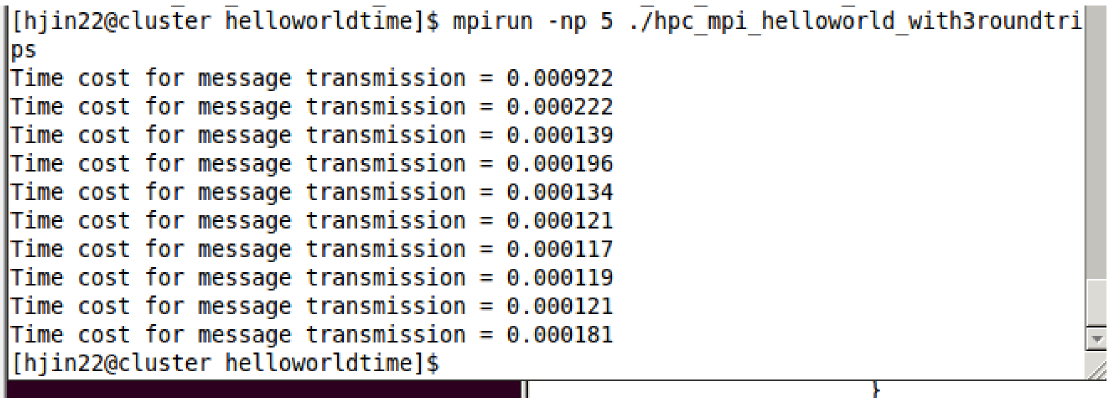
4. ***Code for time evaluation is in folder: "helloworldtime".***

   With 3 round-trips:



```
mpicc -o hpc_mpi_helloworld_with3roundtrips hpc_mpi_helloworld_with3roundtrips.c
[hjin22@cluster helloworldtime]$ mpirun -np 5 ./hpc_mpi_helloworld_with3roundtri
ps
Time cost for message transmission = 0.002479
Time cost for message transmission = 0.000272
Time cost for message transmission = 0.000243
[hjin22@cluster helloworldtime]$
```

With 10 round-trips:

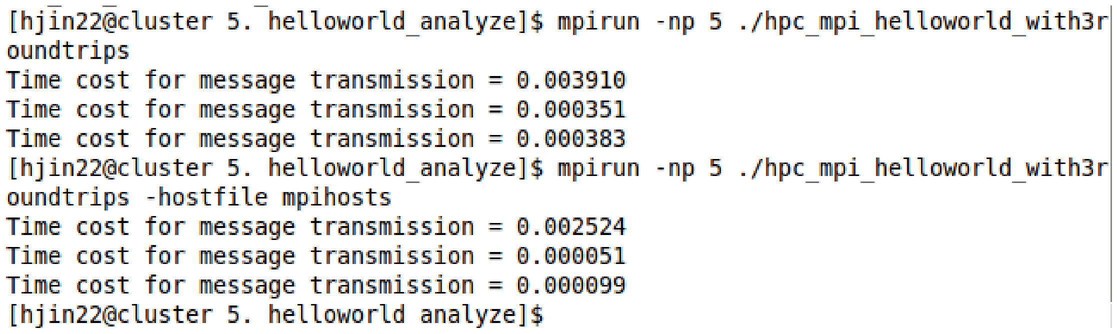


```
[hjin22@cluster helloworldtime]$ mpirun -np 5 ./hpc_mpi_helloworld_with3roundtri
ps
Time cost for message transmission = 0.000922
Time cost for message transmission = 0.000222
Time cost for message transmission = 0.000139
Time cost for message transmission = 0.000196
Time cost for message transmission = 0.000134
Time cost for message transmission = 0.000121
Time cost for message transmission = 0.000117
Time cost for message transmission = 0.000119
Time cost for message transmission = 0.000121
Time cost for message transmission = 0.000181
[hjin22@cluster helloworldtime]$
```

It takes nearly 0.0002 seconds to make message circle the ring.

5. *Vary the number of machines in mpihosts:*

Here is a screenshot of that running program.



```
[hjin22@cluster 5. helloworld_analyze]$ mpirun -np 5 ./hpc_mpi_helloworld_with3r
oundtrips
Time cost for message transmission = 0.003910
Time cost for message transmission = 0.000351
Time cost for message transmission = 0.000383
[hjin22@cluster 5. helloworld_analyze]$ mpirun -np 5 ./hpc_mpi_helloworld_with3r
oundtrips -hostfile mpihosts
Time cost for message transmission = 0.002524
Time cost for message transmission = 0.000051
Time cost for message transmission = 0.000099
[hjin22@cluster 5. helloworld analyze]$
```

To better illustrate the trends, I built a chart as follows (since the 1st time I got is not exact the message transmission time, I just ignored this time):

Node number	Processes Number	Time #1	Time #2	Time #3	Average time cost /message
1	1	0.000005	0.000001	0.000002	2.66667E-06
1	2	0.000007	0.000002	0.000002	1.83333E-06
1	3	0.000007	0.000020	0.000002	3.22222E-06
1	4	0.000010	0.000003	0.000003	1.33333E-06
1	5	0.000225	0.000173	0.000197	3.96667E-05
1	6	0.000302	0.000480	0.000329	6.17222E-05
2	4	0.000014	0.000003	0.000003	1.66667E-06
2	8	0.000288	0.000127	0.000230	0.000026875
2	16	0.000584	0.000437	0.000925	4.05417E-05
2	32	0.001464	0.001489	0.002831	0.00006025
2	64	0.007279	0.006499	0.010775	0.00012788

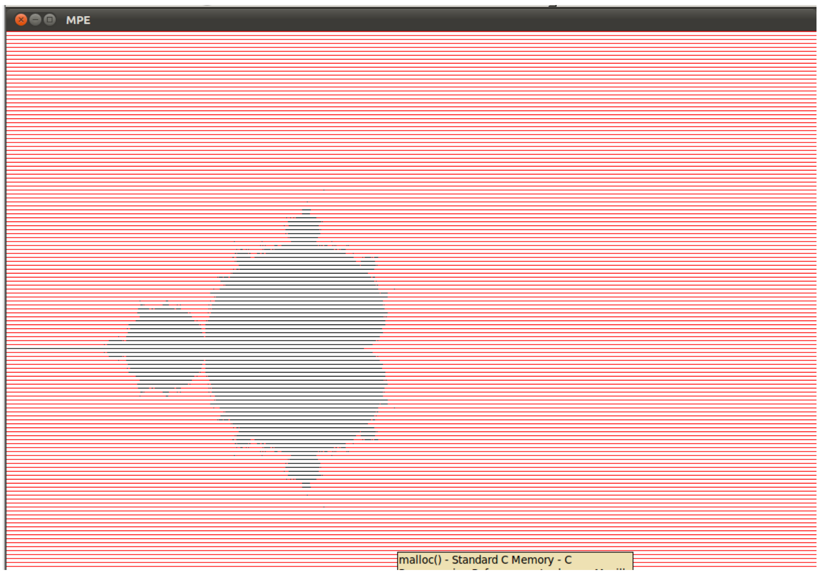
From the information in /proc/cpuinfo, we can learn that the cluster.cci.emory.edu has 4 cores. When we tested the code with node number, it was run on "cluster.cci.emory.edu". We can see the time cost has a significant increasing after the processes number exceeds 4.

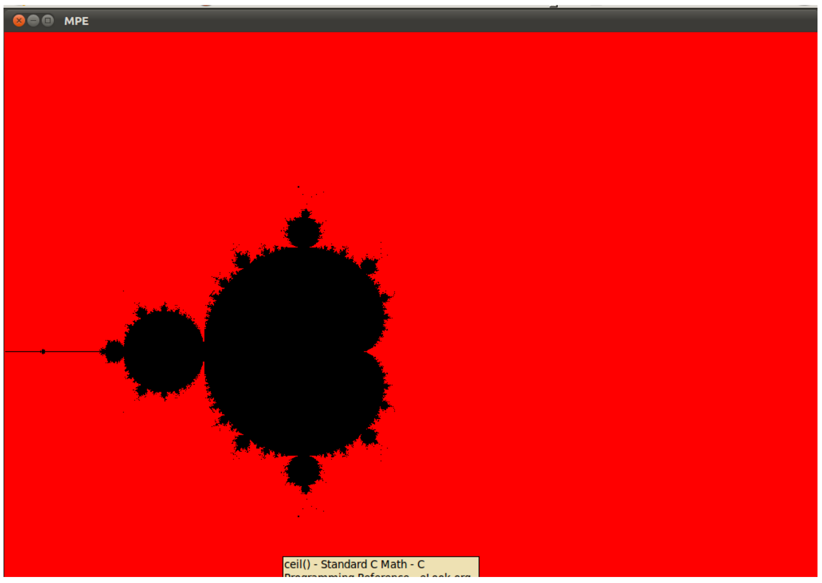
While we expand this experiment to more nodes, like n =2 or n=3 (since the table would be extremely long, I didn't put the data n=3 on that table.)

So we can have a conclusion like that:
The cost of contention between processes on a single node start to outweigh network traffic costs when the "processes/number of machines" > the number of cores machines have. In other words, it would outweigh when the processes is more than the cores these cluster have.

Report for Question 2: The Mandelbrot Set.

1. The code in domain-composition method is in "MPI.Mandelbrot/domain_composition"
   The working screenshots are as follows. The 1st pic is how this new code draw the pic.



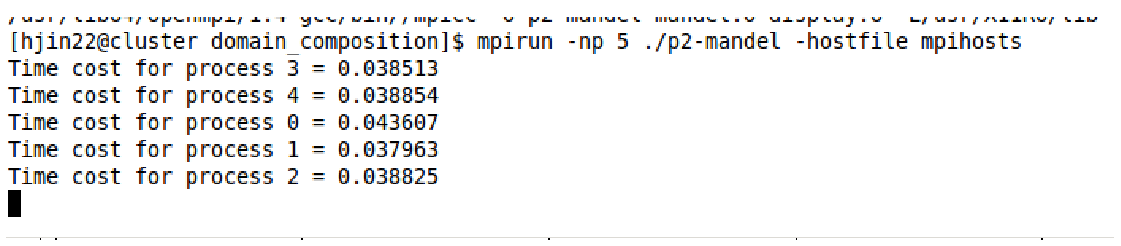


   For the problem, does order in which rows are computed make a difference?
   Yes. If we separated the task like the 1st process works on the top 100 rows, and 2nd process works on the next 100 rows …… It's easy to find that the later processes have a heavier workload compared with the former ones. When the later processes are still working, the former ones may be in idle.
   So how to balance the work load to different processes is the key problem. I used modulus

operation to make the work load balance.

Performance Review

Here is a screen shot during performance review section.



```
[hjin22@cluster domain_composition]$ mpirun -np 5 ./p2-mandel -hostfile mpihosts
Time cost for process 3 = 0.038513
Time cost for process 4 = 0.038854
Time cost for process 0 = 0.043607
Time cost for process 1 = 0.037963
Time cost for process 2 = 0.038825
```

We can get a similar data table like that:

Chart for Domain-Composition:

Node Number	Process Number	Longest processing time	Processes/Node
1	1	0.190550	1
1	4	0.056593	4
1	16	0.060688	16
1	32	0.036163	32
2	1	0.196408	0.5
2	4	0.050979	2
2	8	0.038118	4
2	16	0.026772	8
2	32	0.036151	16
2	64	0.058907	32
4	1	0.194155	0.25
4	4	0.048529	1
4	16	0.022379	4
4	32	0.026912	8

From this chart, we can see, the program has a best performance when it has a processes/node under 8, since the node on clusters has 8 cores (but the cluster.cci.emory.edu is a 4 cores machine, that's why the 3rd entry's performance is better.).

Chart for Processor Farm:

Node Number	Process Number	Longest processing time	Processes/Node
1	1	No result. (Master process doesn’t draw.)	1
1	2	0.197558	2
1	4	0.067205	4
1	16	0.062024	16
1	32	1.010694	32
2	1	No result. (Master process doesn’t draw.)	0.5
2	2	0.194218	1
2	4	0.073578	2
2	16	0.068836	8
2	32	0.102448	16
4	1	No result. (Master process doesn’t draw.)	0.25
4	4	0.111042	1
4	16	0.063231	4
4	32	0.075884	8

From this chart, we can see the program has best performance when processes/node is fewer than 4, which is the number of cores/node.

When the number of processes/nodes exceeds the number of cores/node, the program spent more time on traffic and some other activity, which results in a reduction in performance.