L. Radebe
HPC Week 2

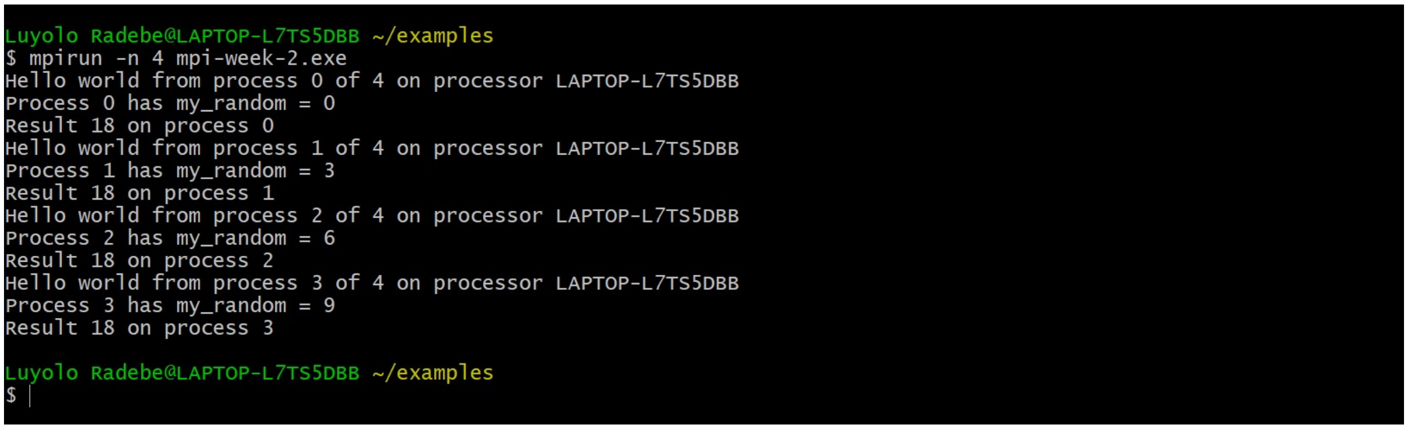
1.1.
Adapt the example on slide 14 to use the command MPI_Get_processor_name to print out on what processor your process runs.

The given mpi-week-2.c file is modified to include the additions shown below

```
MPI_Get_processor_name(name,&len);
printf("Hello world from process %d of %d on processor %s \n", proc_id, proc_size, name);
```

HPC/week2/mpi-week-2.c

Gives the following output on the localhost:


```
Luyolo Radebe@LAPTOP-L7TS5DBB ~/examples
$ mpirun -n 4 mpi-week-2.exe
Hello world from process 0 of 4 on processor LAPTOP-L7TS5DBB
Process 0 has my_random = 0
Result 18 on process 0
Hello world from process 1 of 4 on processor LAPTOP-L7TS5DBB
Process 1 has my_random = 3
Result 18 on process 1
Hello world from process 2 of 4 on processor LAPTOP-L7TS5DBB
Process 2 has my_random = 6
Result 18 on process 2
Hello world from process 3 of 4 on processor LAPTOP-L7TS5DBB
Process 3 has my_random = 9
Result 18 on process 3

Luyolo Radebe@LAPTOP-L7TS5DBB ~/examples
$ |
```

1.2.
Confirm that you are able to run a program that uses two different nodes on the CHPC.

A PBS job script is written to make use of just above the minimum amount of resources allowable for a single node in a 2 node job with additional instructions shown below:

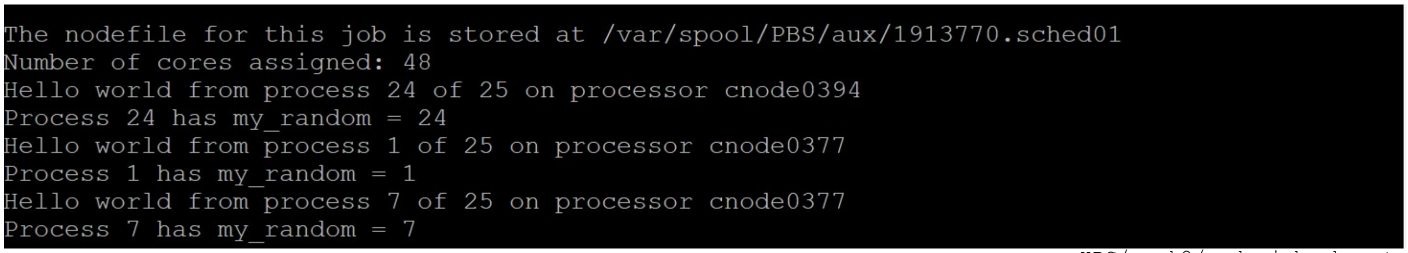
```
module add chpc/openmpi/1.10.2/gcc-5.1.0

np=$(wc -l < ${PBS_NODEFILE})

cd ${PBS_O_WORKDIR}
echo "The nodefile for this job is stored at ${PBS_NODEFILE}"
echo "Number of cores assigned: ${np}"
mpirun -np 25 mpi-week-2
```

HPC/week2/node_job.job

The job script gives the following output file:


```
The nodefile for this job is stored at /var/spool/PBS/aux/1913770.sched01
Number of cores assigned: 48
Hello world from process 24 of 25 on processor cnode0394
Process 24 has my_random = 24
Hello world from process 1 of 25 on processor cnode0377
Process 1 has my_random = 1
Hello world from process 7 of 25 on processor cnode0377
Process 7 has my_random = 7
```

HPC/week2/node_job.sh.out

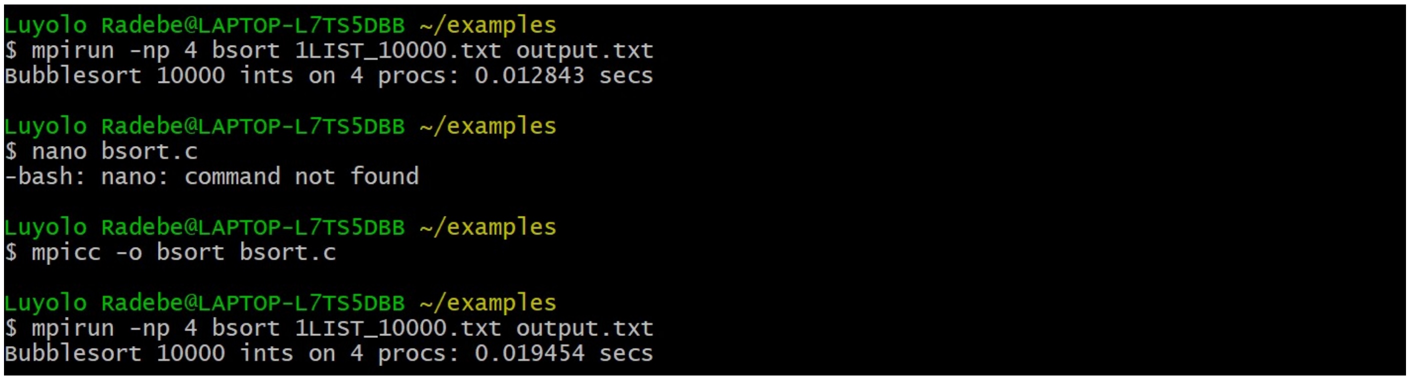
2.1.
For the Bubble sort example that was briefly reviewed, create a larger dataset and test the scaling performance on the CHPC.

1 000 Medium example	100 000 Large example
2 procs: 0.077110 secs	2 procs: >220 000 secs
4 procs: 0.024072 secs	4 procs: >220 000 secs
6 procs: 0.011686 secs	6 procs: 1738.595156 secs
8 procs: 0.007135 secs	8 procs: 1009.508301 secs
10 procs: 0.005038 secs	10 procs: 664.399389 secs

2.2.
Can you improve the code to get better scaling?

2.2.1. Static vs Static Inline
Produces a 33% increase in speed when using 4 processes

$$percent := \frac{0{,}012843 - 0{,}019454}{0{,}019454} = -33{,}9827\ \%$$



```
Luyolo Radebe@LAPTOP-L7TS5DBB ~/examples
$ mpirun -np 4 bsort 1LIST_10000.txt output.txt
Bubblesort 10000 ints on 4 procs: 0.012843 secs

Luyolo Radebe@LAPTOP-L7TS5DBB ~/examples
$ nano bsort.c
-bash: nano: command not found

Luyolo Radebe@LAPTOP-L7TS5DBB ~/examples
$ mpicc -o bsort bsort.c

Luyolo Radebe@LAPTOP-L7TS5DBB ~/examples
$ mpirun -np 4 bsort 1LIST_10000.txt output.txt
Bubblesort 10000 ints on 4 procs: 0.019454 secs
```

2.2.2. Exiting code after no more swaps are possible
The following adaptations can be made to the bubble sort code to improve on runtime by aborting the bubble sort once no more swaps can be made.

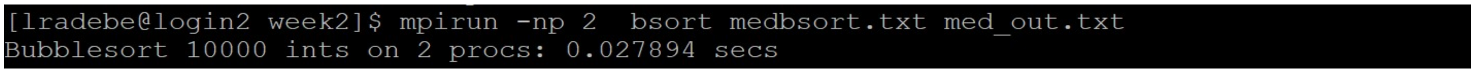
This speeds up the original code by 63% when using 2 processes

$$speedup := \frac{0{,}027894 - 0{,}07711}{0{,}07711} = -63{,}8257\ \%$$

The program output can be seen below.

```
void bubblesort(int * v, int n)
{
 int i, j;
 int flag;
 for (i = n-2; i >= 0; i--)
 {
  flag = 0;
  for (j = 0; j <= i; j++)
  {
  if (v[j] > v[j+1])
   {
   flag = 1;
    swap(v, j, j+1);
  }
  }
  if(flag == 0)
  {
  return;
  }
 }
}
```

HPC/week2/bsort.c



```
[lradebe@login2 week2]$ mpirun -np 2  bsort medbsort.txt med_out.txt
Bubblesort 10000 ints on 2 procs: 0.027894 secs
```

3) Write an MPI parallelised program that takes in a very large integer number N and then print out all the prime numbers between 1 and N. The work should be distributed between each process. Also test your scaling performance on the CHPC

4) Implement the parallel vector norm calculation mentioned on slide 19. Modify this example to calculate the 2Norm relative error: err = ||x - xref||2 / ||xref||2) - where x is some complex vector and xrefthe complex vector that is the reference result. Use initiative to generate data.

5) Implement the ping-pong example (in C only) on the CHPC and measure the quantities b and tl.