Parallel Computing Labtainer

Disclaimer: This software is provided as is. Use at your own risk. The views expressed herein do not necessarily reflect those of the DoD.

Description: This lab explores the Message Passing Interface (MPI) standard, used widely in High-Performance Computing (HPC) [5]. This lab will use the MPICH implementation of MPI to run a variety of parallel programs on a single machine [1].

Motivation: Since the performance of sequential processors has plateaued, computer scientists must leverage parallelism to accelerate their applications [2, 3, 4]. MPI allows software to achieve high throughput on chip multiprocessors (i.e., multi-core CPUs) and HPC clusters.

Note: Due to the simple nature of these introductory MPI programs, you might not achieve significant speedup due to the high communication cost relative to the low computation cost. That being said, make sure that your VM has been allocated at least 2 processors.

Part 1: Hello World (hello.c)

Step 1: Compile and run the parallel Hello World program [5] with the following commands:

```
mpicc -o hello hello.c
mpirun -np 4 ./hello
```

Step 2: You will see the string "Hello World" appear four times, once for each process:

```
Hello world
Hello world
Hello world
Hello world
```

Part 2: Hello World with Process ID (procid.c)

Step 1: Comiple and run the parallel Hello World with Process ID program [5]:

```
mpicc -o procid procid.c
mpirun -np 4 ./procid
```

Step 2: You will see the string "Hello World" appear four times, but you will also see the rank of each process. The master process has rank 0, and the slave processes have rank 1, 2, and 3:

```
Hello world! I'm process 2 out of 4 processes
Hello world! I'm process 3 out of 4 processes
Hello world! I'm process 1 out of 4 processes
Hello world! I'm process 0 out of 4 processes
```

Note that since the four processes run in parallel, the output can appear in any order, e.g., 2-3-1-0 as shown above or any other combination, e.g., 0-1-2-3, 0-1-3-2, etc.

Part 3: Array Summation (arraysum.c)

Step 1: Compile and run the parallel Array Summation program [5]:

```
mpicc –o arraysum arraysum.c
mpirun –np 4 ./arraysum
```

Step 2: You will be prompted to enter an array size. Enter 100. You will see the partial sums computed by each of the four processes, as well as the grand total of 5050, which is the sum of the integers from 1 to 100:

```
please enter the number of numbers to sum:
100
sum 351 calculated by root process
Partial sum 2124 returned from process 3
Partial sum 1600 returned from process 2
Partial sum 975 returned from process 1
The grand total is: 5050
```

Part 4: Array Search (search.c)

Step 1: Compile and run the parallel Array Search program with the following commands:

```
mpicc –o search search.c
mpirun –np 4 ./search
```

Step 2: You will be prompted to enter an array size and a query. Enter 100 for the size and 7 for the query. You will see that the master node (rank 0) will report finding the query, since the first 25 array elements belong to the master:

```
please enter the number of numbers to search:
100
please enter the number to search for:
7
FOUND 7 in 0 at 6!
```

Step 3: Run it again, but this time change the query from 7 to 40. You will see that the first slave (rank 1) reports finding the query, since the second 25 array elements belong to slave 1:

```
mpirun -np 4 ./search
please enter the number of numbers to search:
100
please enter the number to search for:
40
FOUND 40 in 1 at 13!
```

Step 4: Run it again, but this time change the query to 65. You will see that the second slave (rank 2) reports finding the query, since the third 25 array elements belong to slave 2:

```
mpirun -np 4 ./search
please enter the number of numbers to search:
100
please enter the number to search for:
65
FOUND 65 in 2 at 13!
```

Step 5: Run it again, but this time change the query to 90. You will see that the third slave (rank 3) reports finding the query, since the fourth 25 array elements belong to slave 3:

```
mpirun -np 4 ./search
please enter the number of numbers to search:
100
please enter the number to search for:
90
FOUND 90 in 3 at 13!
```

Part 5: Image Processing (image.c)

Step 1: Compile and run the parallel Image Processing program with the following commands:

```
mpicc -o image image.c
mpirun -np 4 ./image lil_sun.ppm
```

Step 2: You will see the following output as the color of each pixel of lil_sun.ppm is inverted:

```
About to receive on slave 3
About to receive on slave 1
w=88 h=95 max=255
malloc succeeded. this is a debug message.
About to receive on slave 2
about to send on 0
about to send on 0
about to send on 0
about to receive on 0
about to send on  3
about to send on  1
about to send on  2
about to receive on 0
about to receive on 0
```

Step 3: Visually confirm that the colors of the output file, inverted.ppm, differ from the colors of the input file, lil_sun.ppm.

References:

[1] Abdelahim Amer et al., *MPICH User's Guide, Version 3.4.2*, Mathematics and Computer Science Division, Argonne National Laboratory, 28 May 2021.
URL: https://www.mpich.org/static/downloads/3.4.2/mpich-3.4.2-userguide.pdf

[2] Krste Asanovic et al. "A View of the Parallel Computing Landscape." *Communications of the ACM*, vol. 52, no. 10, pp. 56-67, October 2009.
URL: https://dl.acm.org/doi/10.1145/1562764.1562783

[3] Mark Oskin. "The Revolution Inside the Box." *Communications of the ACM*, vol 51, no. 7, pp. 70-78, July 2008. URL: https://dl.acm.org/doi/abs/10.1145/1364782.1364799

[4] Thomas Rauber and Gudula Rünger. *Parallel Programming for Multicore and Cluster Systems, Second Edition*, Springer-Verlag Berlin Heidelberg, 2013, 516 pages.
URL: https://link.springer.com/book/10.1007/978-3-642-37801-0

[5] Daniel Thomasset and Michael Grobe. "Introduction to the Message Passing Interface (MPI) Using C." Online Tutorial Document, Academic Computing Services, The University of Kansas, November 2013. URL: http://condor.cc.ku.edu/~grobe/docs/intro-MPI-C.shtml