Question – 2

Assignment – 4

Develop a parallel histogramming program using C/C++ and OpenMPI. A histogram is used to summarize the distribution of data values in a data set. The most common form of histogramming splits the data range into equal-sized bins. For each bin, the number of data values in the data set that falls into that class are totaled. Your input to this program will be integers in the range 1-100,000 (use a random number generator that first generates the numbers). Your input data set should contain 8 million integers. You will vary the number of bins. You need to figure out how to assign bins to OpenMPI processes. You are suggested to use the sample batch script provided on Canvas for specifying your OpenMPI configuration and running your program (you will need to change some of the job parameters).

a) Assume there are 128 bins. Perform binning across nodes and processes using OpenMPI, and then perform a reduction on the lead node, combining your partial results. Run this on 2, 4 and 8 nodes on Explorer. Your program should print out the number of values that fall into each bin. Compare the performance between running this on 2, 4 and 8 nodes. Comment on the differences.
b) For this part, assume you have 32 bins. Perform binning on each process using OpenMPI, and then perform a reduction on the lead node, combining your partial results. Run this on 2 and 4 nodes on Explorer. Your program should print out the number of values that fall into each bin. Compare the performance between running this on 2 and 4 nodes. Comment on the differences.
c) Compare the performance measured in parts a.) and b.). Try to explain why one is faster than the other and run additional experiments to support your claims.

Answer:

In Part (a), We first use 2 nodes and this is the output we got:

```
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=16
# ...
srun ~/MPI/parallel_histogram 128
```

```
Job Name: DavesJob
Running on 2 nodes with 16 tasks per node
Total tasks: 32
Final Histogram (128 bins):
Bin 0: 62441
Bin 1: 62319
Bin 2: 61981
Bin 3: 62791
Bin 4: 62596
Bin 5: 62519
Bin 6: 62681
Bin 7: 62687
Bin 8: 62259
Bin 9: 62397
Bin 10: 62149
Bin 11: 62676
Bin 12: 62678
Bin 13: 62581
Bin 14: 62550
Bin 15: 62430
Bin 16: 62058
Bin 17: 62091
Bin 18: 62565
Bin 19: 62213
Bin 20: 62546
Bin 21: 62621
Bin 22: 62697
Bin 23: 62181
Bin 24: 62660
Bin 25: 62072
Bin 26: 62576
Bin 27: 62875
Bin 28: 62684
Bin 29: 62484
Bin 30: 62336
```

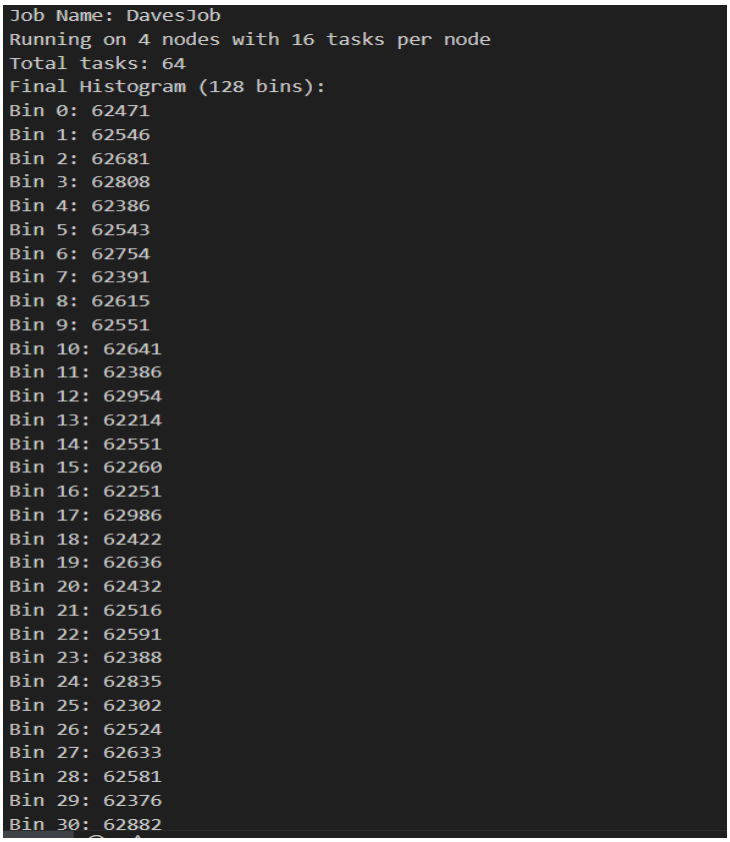
```
Bin 31: 62394
Bin 32: 62336
Bin 33: 62537
Bin 34: 62735
Bin 35: 62374
Bin 36: 62814
Bin 37: 62339
Bin 38: 62466
Bin 39: 62551
Bin 40: 62658
Bin 41: 62313
Bin 42: 62521
Bin 43: 62519
Bin 44: 62687
Bin 45: 62636
Bin 46: 61466
Bin 47: 62460
Bin 48: 62928
Bin 49: 62146
Bin 50: 62261
Bin 51: 62961
Bin 52: 62340
Bin 53: 62451
Bin 54: 62497
Bin 55: 62009
Bin 56: 62408
Bin 57: 62488
Bin 58: 62272
Bin 59: 61758
Bin 60: 62569
Bin 61: 62293
Bin 62: 62385
Bin 63: 62580
Bin 64: 62963
```

```
Bin 65: 62787
Bin 66: 62755
Bin 67: 62429
Bin 68: 62269
Bin 69: 62679
Bin 70: 63073
Bin 71: 62488
Bin 72: 62503
Bin 73: 62853
Bin 74: 62911
Bin 75: 62626
Bin 76: 63058
Bin 77: 62268
Bin 78: 62434
Bin 79: 62573
Bin 80: 62052
Bin 81: 62574
Bin 82: 62356
Bin 83: 62709
Bin 84: 62525
Bin 85: 62557
Bin 86: 62845
Bin 87: 62847
Bin 88: 62796
Bin 89: 62150
Bin 90: 62847
Bin 91: 62436
Bin 92: 62556
Bin 93: 62759
Bin 94: 62220
Bin 95: 62660
Bin 96: 62229
Bin 97: 62602
Bin 98: 62257
Bin 99: 62298
```

```
Bin 100: 63014
Bin 101: 62315
Bin 102: 62263
Bin 103: 62406
Bin 104: 62974
Bin 105: 62189
Bin 106: 62522
Bin 107: 62324
Bin 108: 62744
Bin 109: 62155
Bin 110: 62302
Bin 111: 63167
Bin 112: 62977
Bin 113: 62442
Bin 114: 62156
Bin 115: 62582
Bin 116: 62633
Bin 117: 62402
Bin 118: 62280
Bin 119: 62574
Bin 120: 62760
Bin 121: 62467
Bin 122: 62731
Bin 123: 62127
Bin 124: 62323
Bin 125: 62741
Bin 126: 62412
Bin 127: 62528
Total values tallied: 8000000 (expected ~8000000)
Time elapsed: 0.0748538 seconds
```

(a.2): We use 4 nodes and this is the output we get:

#SBATCH --nodes=4
#SBATCH --ntasks-per-node=16
...
srun ~/MPI/parallel_histogram 128



```
Bin 31: 62656
Bin 32: 62664
Bin 33: 62314
Bin 34: 62388
Bin 35: 62218
Bin 36: 62685
Bin 37: 62236
Bin 38: 62161
Bin 39: 62724
Bin 40: 62574
Bin 41: 62624
Bin 42: 62735
Bin 43: 62495
Bin 44: 62682
Bin 45: 62505
Bin 46: 62255
Bin 47: 62447
Bin 48: 62465
Bin 49: 62578
Bin 50: 62383
Bin 51: 62074
Bin 52: 62937
Bin 53: 61788
Bin 54: 62725
Bin 55: 62423
Bin 56: 62244
Bin 57: 62453
Bin 58: 62727
Bin 59: 62284
Bin 60: 62668
Bin 61: 62260
Bin 62: 62231
Bin 63: 62901
Bin 64: 62413
Bin 65: 62338
```

```
Bin 66: 62764
Bin 67: 62526
Bin 68: 62720
Bin 69: 62532
Bin 70: 61973
Bin 71: 62344
Bin 72: 62666
Bin 73: 62818
Bin 74: 62392
Bin 75: 62238
Bin 76: 62717
Bin 77: 62703
Bin 78: 62474
Bin 79: 62780
Bin 80: 62566
Bin 81: 62615
Bin 82: 61907
Bin 83: 62339
Bin 84: 62243
Bin 85: 62376
Bin 86: 62445
Bin 87: 62282
Bin 88: 62558
Bin 89: 62574
Bin 90: 62665
Bin 91: 62246
Bin 92: 62721
Bin 93: 62340
Bin 94: 62148
Bin 95: 62158
Bin 96: 62605
Bin 97: 62639
Bin 98: 61885
Bin 99: 62573
Bin 100: 63011
```

```
Bin 101: 62379
Bin 102: 62155
Bin 103: 62530
Bin 104: 62452
Bin 105: 62606
Bin 106: 62243
Bin 107: 62160
Bin 108: 62810
Bin 109: 62670
Bin 110: 62869
Bin 111: 63054
Bin 112: 62987
Bin 113: 61851
Bin 114: 62619
Bin 115: 62395
Bin 116: 63052
Bin 117: 61997
Bin 118: 62486
Bin 119: 62322
Bin 120: 62851
Bin 121: 62641
Bin 122: 62260
Bin 123: 62610
Bin 124: 62782
Bin 125: 62684
Bin 126: 62332
Bin 127: 62298
Total values tallied: 8000000 (expected ~8000000)
Time elapsed: 0.0539427 seconds
```

(a.3): We use 8 outputs and this is the output we get:
#SBATCH --nodes=8
#SBATCH --ntasks-per-node=16
...
srun ~/MPI/parallel_histogram 128

```
Job Name: DavesJob
Running on 8 nodes with 16 tasks per node
Total tasks: 128
Final Histogram (128 bins):
Bin 0: 62444
Bin 1: 61983
Bin 2: 62319
Bin 3: 62641
Bin 4: 62466
Bin 5: 62388
Bin 6: 62359
Bin 7: 62650
Bin 8: 62356
Bin 9: 62027
Bin 10: 62235
Bin 11: 62306
Bin 12: 62331
Bin 13: 62602
Bin 14: 62530
Bin 15: 62616
Bin 16: 62586
Bin 17: 63031
Bin 18: 62298
Bin 19: 62441
Bin 20: 62793
Bin 21: 62363
Bin 22: 62240
Bin 23: 62209
Bin 24: 62755
Bin 25: 62413
Bin 26: 62333
Bin 27: 62480
Bin 28: 62482
Bin 29: 62837
Bin 30: 62318
```

```
Bin 31: 62456
Bin 32: 62914
Bin 33: 63153
Bin 34: 62457
Bin 35: 62081
Bin 36: 62357
Bin 37: 62724
Bin 38: 62832
Bin 39: 62409
Bin 40: 62784
Bin 41: 62671
Bin 42: 62333
Bin 43: 62515
Bin 44: 63073
Bin 45: 62528
Bin 46: 62670
Bin 47: 62720
Bin 48: 62761
Bin 49: 62435
Bin 50: 61982
Bin 51: 62649
Bin 52: 62410
Bin 53: 62864
Bin 54: 62275
Bin 55: 62377
Bin 56: 62422
Bin 57: 62841
Bin 58: 62481
Bin 59: 62647
Bin 60: 62829
Bin 61: 62531
Bin 62: 62482
Bin 63: 62703
Bin 64: 62729
```

```
Bin 65: 62965
Bin 66: 62454
Bin 67: 62383
Bin 68: 61934
Bin 69: 62527
Bin 70: 62839
Bin 71: 62764
Bin 72: 62440
Bin 73: 62630
Bin 74: 62631
Bin 75: 62298
Bin 76: 62711
Bin 77: 62451
Bin 78: 62315
Bin 79: 62554
Bin 80: 62830
Bin 81: 62583
Bin 82: 62688
Bin 83: 62216
Bin 84: 62201
Bin 85: 62375
Bin 86: 62846
Bin 87: 62296
Bin 88: 63242
Bin 89: 62442
Bin 90: 62325
Bin 91: 62412
Bin 92: 62258
Bin 93: 62605
Bin 94: 62358
Bin 95: 62600
Bin 96: 62470
Bin 97: 63068
Bin 98: 62785
Bin 99: 62148
```

```
Bin 100: 62308
Bin 101: 61942
Bin 102: 62304
Bin 103: 62884
Bin 104: 62323
Bin 105: 62975
Bin 106: 62382
Bin 107: 62698
Bin 108: 61821
Bin 109: 62566
Bin 110: 62437
Bin 111: 62737
Bin 112: 62456
Bin 113: 62526
Bin 114: 62711
Bin 115: 62570
Bin 116: 62559
Bin 117: 62485
Bin 118: 62468
Bin 119: 62453
Bin 120: 62151
Bin 121: 62282
Bin 122: 62345
Bin 123: 62100
Bin 124: 61911
Bin 125: 62352
Bin 126: 62581
Bin 127: 62102
Total values tallied: 8000000 (expected ~8000000)
Time elapsed: 0.0343269 seconds
```

Configuration	Total MPI processes	Runtime (seconds)
2 nodes	16 ranks/node ⇒ 32 total	0.0748538 seconds
4 nodes	16 ranks/node ⇒ 64 total	0.0539427 seconds
8 nodes	16 ranks/node ⇒ 128 total	0.034269 seconds

Observed differences:

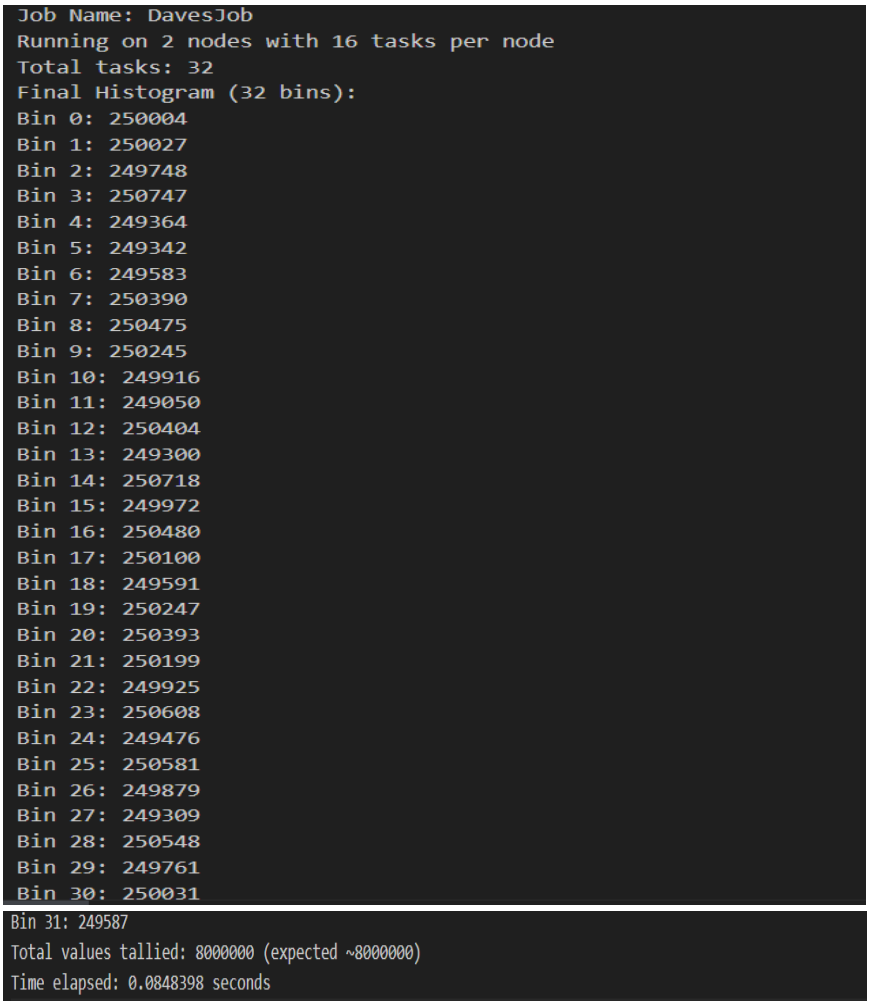
- From 2 → 4 Nodes, the time dropped from 0.0749 s to 0.0539 s and its speedup is roughly 1.39. From 4 → 8 Nodes, the time dropped from 0.0539 s to 0.0343 s and its speedup is roughly 1.57.

Commentary on the differences:

- **Faster computation with more nodes**: The local part of the histogram generation is quicker since each node (and its MPI processes) manages a smaller subset of the 8 million numbers.
- **Communication overhead**: The percentage of time spent on synchronization or collective operations (like MPI_Reduce) increases as more processes divide the work. Therefore, a 2× speedup is not always achieved by doubling the number of nodes.
- **Short Overall Runtimes**: You have very modest absolute times (0.07 s → 0.03 s). This implies that either the code is well optimized or that just a small portion of the work is being captured by the time measurement (or perhaps overhead is overwhelmed by incredibly rapid local operations). Even small setup or communication overhead can negate optimum scalability gains at such short times.
- **Network Effects**: Although extra nodes might occasionally result in more intricate communication patterns, the overhead is usually minimal for a single histogram decrease.

(b): In this, we change the number of bins from 128 to 32 from the code and the number of nodes to 2, we get:

```
#SBATCH --nodes=2
srun ~/MPI/parallel_histogram 32
```



(b.2): We change the number of nodes from 2 to 4. Here is the output we got:

#SBATCH --nodes=4
srun ~/MPI/parallel_histogram 32

```
Job Name: DavesJob
Running on 4 nodes with 16 tasks per node
Total tasks: 64
Final Histogram (32 bins):
Bin 0: 250094
Bin 1: 251193
Bin 2: 249689
Bin 3: 249376
Bin 4: 250108
Bin 5: 249915
Bin 6: 249943
Bin 7: 250130
Bin 8: 250121
Bin 9: 250056
Bin 10: 250017
Bin 11: 249719
Bin 12: 249814
Bin 13: 249425
Bin 14: 249820
Bin 15: 249729
Bin 16: 250142
Bin 17: 249744
Bin 18: 250541
Bin 19: 249163
Bin 20: 250133
Bin 21: 250285
Bin 22: 249893
Bin 23: 250823
Bin 24: 250324
Bin 25: 249936
Bin 26: 250498
Bin 27: 250314
Bin 28: 249576
Bin 29: 249425
Bin 30: 249775
Bin 31: 250279
Total values tallied: 8000000 (expected ~8000000)
Time elapsed: 0.0376092 seconds
```

Observed Differences:

- From 2 → 4 nodes, the runtime drops from 0.0848398 seconds to 0.0376092 seconds. So the speedup is more than doubled (2.26 times). This is greater than ideal doubling which means that distributing work across 4 nodes is quite effective in reducing runtime.
- At four nodes, the random data generation and local histogram computations are accelerated since each node (and its processes) manages less random numbers.

Commentary on the differences:

- Clear Runtime increase: The code scales well when the workload is distributed over multiple nodes, as evidenced by the substantial runtime increase from ~0.085s to ~0.038s.
- Histogram Size: The communication overhead for the final MPI_Reduce is minimal because there are just 32 bins. The fact that the majority of the work is done locally,

which spreads well over many ranks, may be the reason you see such a significant speedup.

(c): Here we are going to compare the difference between the execution time in 2 nodes, 4 nodes and 8 nodes in 128 and 32 bins respectively.

- On 2 nodes:
    - 128 bins; the execution time is 0.0749 seconds.
    - 32 bins; the execution time is 0.0848 seconds

Comments: The run with 32 bins should more execution time than 128 bins which is unusual and tangential to the theoretical trend.

- On 4 nodes:
    - 128 bins; the execution time is 0.0539 seconds
    - 32 bins; the execution time is 0.0376 seconds

Comments: Here 32 bins runs faster than 128 bins, which actually satisfies the condition that decrement in number of bins is equal to decrement in execution time

- On 8 nodes:
    - 128 bins; the execution time is 0.0343 seconds
    - 32 bins; the execution time is 0.034791 seconds

Comments: This falls well within the normal range of run-to-run fluctuation for a work that takes about 0.03 seconds to finish. In essence, both runs are so quick at 8 nodes that the overhead variations are insignificant

In the last part of question (c), we have to run additional experiments to support the claim:
So we will take number of nodes as 16 and 32 and run both 128 bins and 32 bins

If we ran with 16 nodes and 128 bins then we get:

```
Job Name: DavesJob
Running on 16 nodes with 16 tasks per node
Total tasks: 256
Final Histogram (128 bins):
Bin 0: 62449
Bin 1: 62350
Bin 2: 62370
Bin 3: 62001
Bin 4: 62685
Bin 5: 62640
Bin 6: 62415
Bin 7: 62318
Bin 8: 62294
Bin 9: 62380
Bin 10: 62363
Bin 11: 62553
Bin 12: 62165
Bin 13: 62264
Bin 14: 62817
Bin 15: 62601
Bin 16: 62837
Bin 17: 62661
Bin 18: 62075
Bin 19: 62594
Bin 20: 62927
Bin 21: 62767
Bin 22: 62697
Bin 23: 62771
Bin 24: 62266
Bin 25: 62727
Bin 26: 62754
Bin 27: 62875
Bin 28: 62756
Bin 29: 62055
Bin 30: 62350
Bin 31: 62665
Bin 32: 62744
Bin 33: 62243
Bin 34: 61954
Bin 35: 62488
Bin 36: 62323
Bin 37: 62125
Bin 38: 62338
Bin 39: 62341
Bin 40: 62696
Bin 41: 62013
Bin 42: 62787
Bin 43: 62263
Bin 44: 62938
Bin 45: 62699
Bin 46: 62375
Bin 47: 62195
Bin 48: 62543
Bin 49: 62587
Bin 50: 62138
Bin 51: 62565
Bin 52: 62583
Bin 53: 62473
Bin 54: 62525
Bin 55: 62319
Bin 56: 62918
Bin 57: 62575
Bin 58: 62342
Bin 59: 61895
Bin 60: 62329
Bin 61: 62303
Bin 62: 62060
Bin 63: 62699
Bin 64: 62753
Bin 65: 63013
```

```
Bin 66: 63091
Bin 67: 62677
Bin 68: 62457
Bin 69: 62448
Bin 70: 62388
Bin 71: 62435
Bin 72: 62933
Bin 73: 62512
Bin 74: 62341
Bin 75: 62157
Bin 76: 62257
Bin 77: 62973
Bin 78: 62244
Bin 79: 62047
Bin 80: 62903
Bin 81: 62255
Bin 82: 62723
Bin 83: 62636
Bin 84: 62166
Bin 85: 62691
Bin 86: 62928
Bin 87: 62133
Bin 88: 62933
Bin 89: 62639
Bin 90: 62819
Bin 91: 62217
Bin 92: 62240
Bin 93: 62546
Bin 94: 62594
Bin 95: 62503
Bin 96: 62631
Bin 97: 62202
Bin 98: 62355
Bin 99: 62116
Bin 100: 62853
Bin 101: 62618
Bin 102: 62830
Bin 103: 62397
Bin 104: 62512
Bin 105: 62394
Bin 106: 62517
Bin 107: 63133
Bin 108: 63111
Bin 109: 62415
Bin 110: 62462
Bin 111: 62522
Bin 112: 62623
Bin 113: 62441
Bin 114: 62384
Bin 115: 62286
Bin 116: 62320
Bin 117: 62447
Bin 118: 62818
Bin 119: 62495
Bin 120: 62721
Bin 121: 62320
Bin 122: 62576
Bin 123: 62104
Bin 124: 62337
Bin 125: 62926
Bin 126: 62295
Bin 127: 62369
Total values tallied: 8000000 (expected ~8000000)
Time elapsed: 0.0226543 seconds
```

The execution time we got here is 0.0226543 seconds.

Next we run 16 nodes on 32 bins then we get:

```
Job Name: DavesJob
Running on 16 nodes with 16 tasks per node
Total tasks: 256
Final Histogram (32 bins):
Bin 0: 249792
Bin 1: 250050
Bin 2: 249442
Bin 3: 249665
Bin 4: 250364
Bin 5: 249891
Bin 6: 250120
Bin 7: 249844
Bin 8: 248918
Bin 9: 250843
Bin 10: 250043
Bin 11: 249653
Bin 12: 250582
Bin 13: 250129
Bin 14: 250472
Bin 15: 250574
Bin 16: 250665
Bin 17: 249375
Bin 18: 250269
Bin 19: 249277
Bin 20: 249865
Bin 21: 250214
Bin 22: 250384
Bin 23: 249571
Bin 24: 250405
Bin 25: 250178
Bin 26: 250209
Bin 27: 250231
Bin 28: 249687
Bin 29: 249656
Bin 30: 249315
Bin 31: 250317
Total values tallied: 8000000 (expected ~8000000)
Time elapsed: 0.0225777 seconds
```

The execution time is 0.0225777 seconds.

- On 16 nodes
    - 128 bins we got the execution time as 0.0226543 seconds
    - 32 bins we got the execution time as 0.0225777 seconds

By changing the parameters it is observed that:

- The execution time is not greatly impacted by the increase in bins, which presumably increases data processing both within and between nodes. At this scale, the communication overhead brought on by the increased data handling is probably not a

substantial bottleneck. This may be considered a good sign of how well your distributed arrangement communicates.

- With fewer bins (32) you often get lower (or at least not higher) runtimes, because there is less overhead in local increments and the final MPI_Reduce.
- This 16-node result (both ~0.0226 s) demonstrates that the difference in bin counts becomes insignificant as the workload is widely distributed over numerous nodes. Bin count shouldn't be the deciding criterion because the code is finished too soon. Random HPC effects and MPI setup costs outweigh any potential benefits of employing fewer bins.