Homework 5 - MPI Histogram

Philip Nelson

30th November 2021

Histogram

This program uses MPI to calculate a distributed histogram. The directory Histogram contains the code to calculate a histogram from homework 2. Helpers contains auxiliary functions to support the main driver program like parsing commandline arguments, generating data, and printing to stdout. The main program is in the driver directory; it takes commandline arguments bin count, min measurement, max measurement, data count, and optionally a seed for the pseudo random number generator (default 100). It prints the bin maxes and bin counts to stdout.

The project includes a CMakeLists.txt. The following commands can be used to build and run the project on the center for high performance computing (CHPC).

```
module load cmake/3.21 gcc/9 openmpi
mkdir build
cd build
cmake -DCMAKE_BUILD_TYPE=Release ..
make -j
mpiexec -n 4 driver/run 10 0 10 100000

# bin_maxes: [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 ]
# bin_counts: [ 9873, 9980, 9920, 9932, 9997, 10136, 9909, 10030, 10187, 10036 ]
```

Results

The program `bench` runs the histogram program with the same command line arguments but performs three timing measurements:

1. total execution time - time starts at `MPI_Init` and ends after `MPI_Finalize`
2. application time - time starts after data generation and ends after the final histogram result is aggregated
3. histogram time - time starts before local histogram calculation and ends after local histogram calculation

The timing trials were performed on $50,000,000$, $100,000,000$ and $200,000,000$ elements using 1 to 64 processes with a 10 run average for each configuration. The trials were run on four nodes of kingspeak at the CHPC. The file `mpi_batch_script.sh` contains the complete configuration used to submit the job to `SLURM`. I used a python script to run the program multiple times with different configurations of commandline arguments. Each time it ran, output was redirected and appended to a file to collect results. After collecting results, I used python with pandas, and matplotlib to calculate the averages of the trials, speedup, and efficiency of each configuration and create plots. The results of the trials can be seen in Figure 1 below.

These plots reveal that the histogram calculation scales linearly, however; there is a lot of overhead associated with generating and distributing the dataset compared to the time that it takes to calculated the histogram. This results in poor overall scaling.

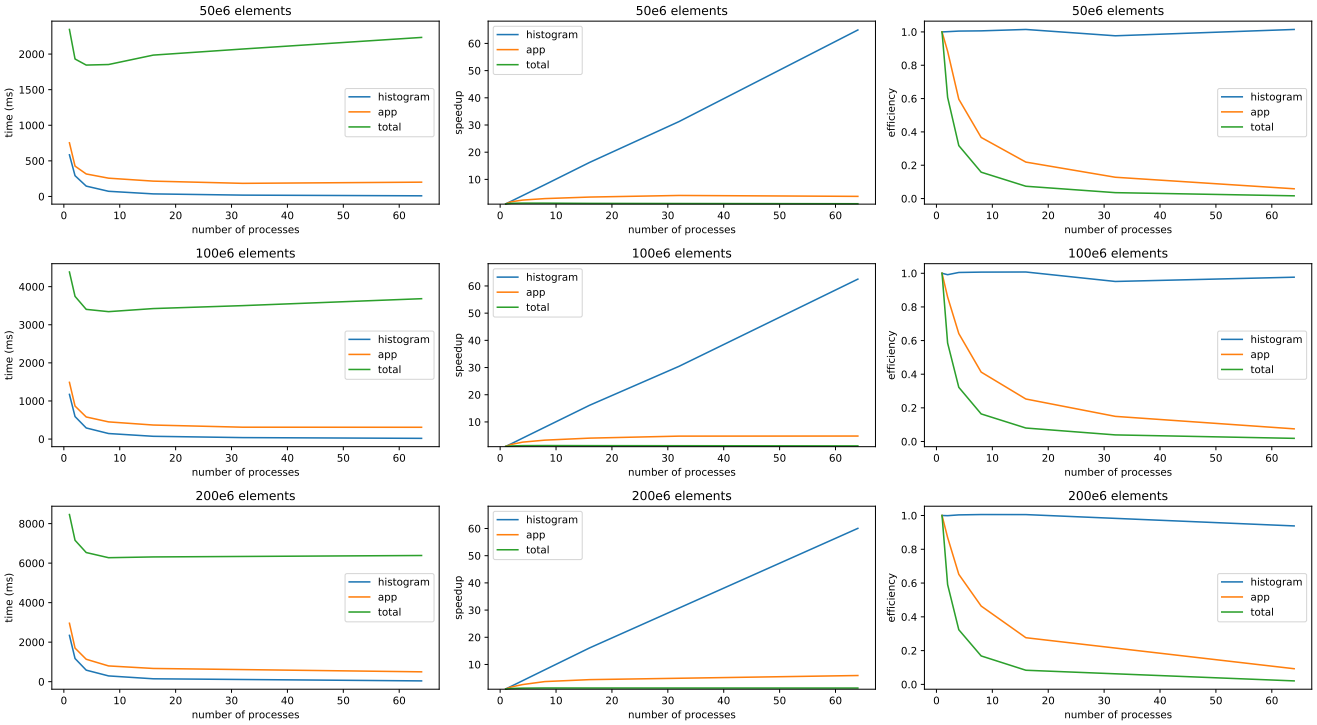

Figure 1: Timing, Speedup, and Efficiency