**Homework4 Report**

Yuanrui Zhang (yz545) & Yuchuan Li (yl645)

**Problem 1: Histogram**

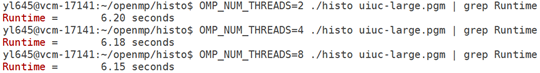
**Correctness:**



The only difference between my results and the validation.out file is the runtime part, not the result of histogram part, so these three versions are all correct.

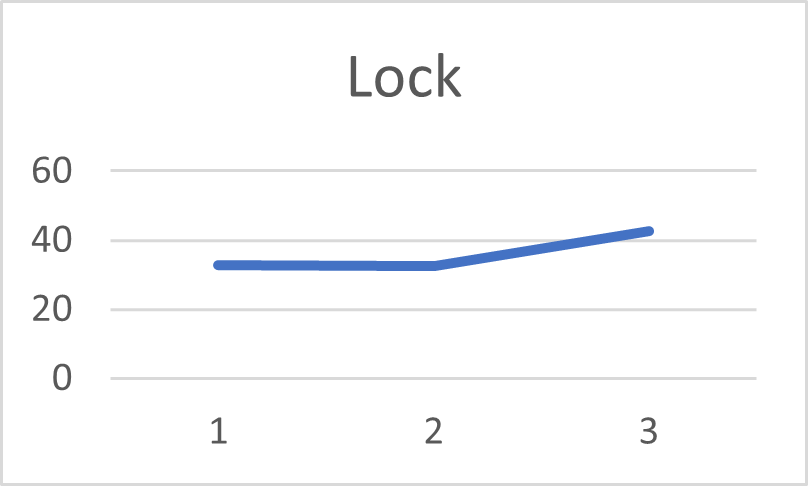
**Performance:**

Original Version:



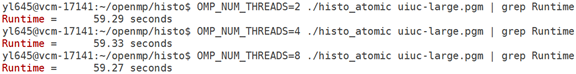
Lock Version:

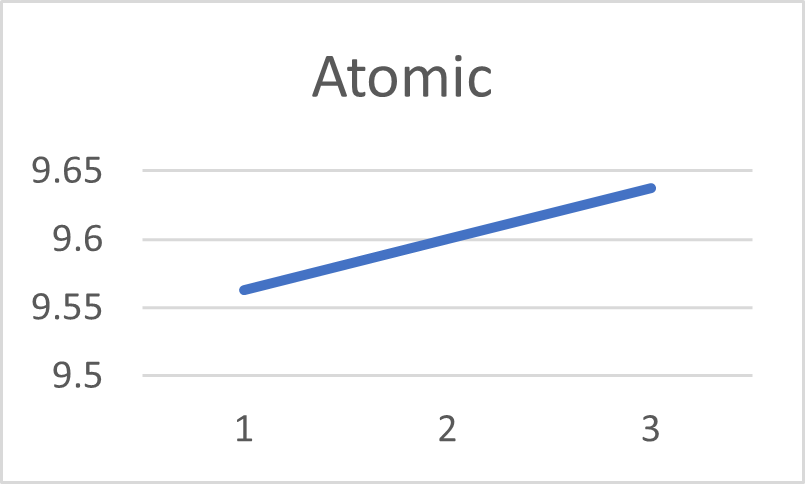




\*1 2 3 represents 2, 4, 8 processes

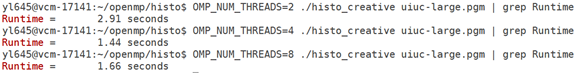
Atomic Version:

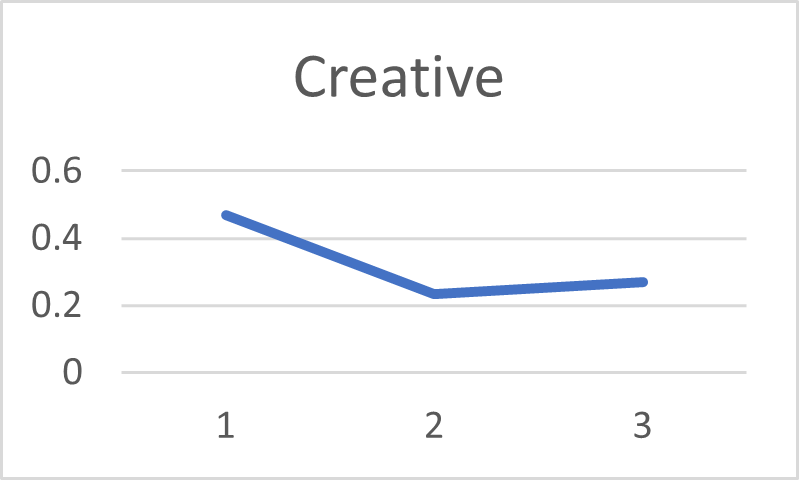




\*1 2 3 represents 2, 4, 8 processes

Creative Version:





\*1 2 3 represents 2, 4, 8 processes

**Creative Version Demonstration:**

The histo[x] is being updated all the time for the same x, so there will be race condition when executing parallelly. We can use the reduction for the array histo[] to update the value of histo[x] concurrently and sum them up in the end to get correct values for the whole array. To be more specific, this method will create a private version per process and update the private version concurrently.

**Analysis:**

Speed comparison: creative < original < atomic < lock

The lock version is significantly slower than other versions, this might be due to the cost of communication and the overhead of using locks. Even though I used an array of locks, which means that the granularity of locks is smaller, it still took much longer time than others.

There is not much speed-up for the lock version and atomic version, even worse. But the creative version has a boost when the number of processes increase.

**Problem 2: amgmk**

**1 Code changes**

1. MATVEC
2. Source file: csr\_matvec.c
3. Line number: 172
4. Code snippet:

Text

Description automatically generated

Text

Description automatically generated

1. OpenMP directive description

As for the first optimization, I set up an omp parallel and for directive with default as shared variable. And private variables are i, j, jj and temp because these variables have read/write conflict. I didn’t change the code other than that.

Similarly, I tried the same thing at the second spot, but slightly different in setting up private variables.

1. How do I know to add parallel code.

I used *perf report:*

*Text

Description automatically generated*

And from this report, I got to know that the most time-consuming part of code is:

Text

Description automatically generated

Thus, I built the OpenMP directive on top of this code.

1. Relax
2. Source file: relax.c
3. Line number: 76
4. Code snippet:

Text

Description automatically generated

1. OpenMP directive description

I set up an omp parallel and for directive with default as shared variable. And private variables are i, jj and res because these variables have read/write conflict. I also removed variable ii in order to decrease a private storage overhead.

1. Axpy
2. Source file: vector.c
3. Line number: 383
4. Code snippet:

Text

Description automatically generated

1. OpenMP directive description

I set up an omp parallel and for directive with default as shared variable. And private variable is only i. I didn’t change the code other than that.

**2 Performance Summary**

1. Sequential.

Run *./AMGMk* 5 times, get averagely:

Total Wall time = 2.86 seconds.

1. Parallelize with 1 thread

Run *OMP\_NUM\_THREADS=1 ./AMGMk* 5 times, get averagely,

Total Wall time = 2.87 seconds

1. Parallelize with 2 thread

Run *OMP\_NUM\_THREADS=2 ./AMGMk* 5 times, get averagely,

Total Wall time = 1.44 seconds

1. Parallelize with 4 thread

Run *OMP\_NUM\_THREADS=4 ./AMGMk* 5 times, get averagely,

Total Wall time = 0.74 seconds

1. Parallelize with 8 thread

Run *OMP\_NUM\_THREADS=8 ./AMGMk* 5 times, get averagely,

Total Wall time = 0.42 seconds

In conclusion, as the number of threads goes up, the performance increases, but not strictly proportionally.