CSC 333 Project 2

Aidan Pieper 4/25/17

Program Design

I decided that it wasn't worth it to parallelize hash generation because `SHA-256` is relatively fast to compute. Thus, the root node generates `n` hashes (as `unsigned long`s) using 10 character random strings and stores them in an array of length `n`. This array is then broadcast to all other nodes. If hash generation was parallelized then each core would have its own random seed. If I moved hashes around from different nodes I would have to keep track of the random seed that generated it somehow and I didn't want to do that (Thank you John for telling me that there is an easy way around this problem. However, this was my thinking process in the design stage).

To check collisions, I use two nested for loops similar to the `birthday_hash.c` starter code to check all pairs of hashes in the buffer of hashes. I parallelize the inner loop by using a cyclic distribution in the loop. I have a variable flag on each node that stores whether a collision is found. When a collision is found, the program breaks from either of the loops. After the inner loop has finished, I `MPI_Allreduce` the collision found variable to all other nodes. This means that if a collision is found, all other nodes get notified and stop their loops.

Timing is computed in a way similar to Project 1 and the textbook.

<u>Programs</u>:

`MPI_birthday_attack.c`: MPI code similar to `birthday_hash.c`. Program exits once a single hash collision has been found.

`MPI_birthday_attack_loop.c`: Modified code that runs forever and prints out number of hash collisions it finds periodically. I just let jupiter kill the program after 1 hour and looked at the output for the extra credit.

Performance Analysis

I ran my program five times for each `comm_sz` and number of hashes (keeping the bit window constant at 32 bits). I took the average of the time of the five runs.

- Bit window of 32 bits
- `comm_sz` of 1,2,4,8,16
- Number of hashes 77000, and 110000

See the PDF spreadsheet for speed and efficiency numbers.

<u>Speedup</u>

I don't trust my speedup numbers because I feel that some of my runs just got really lucky or unlucky. For example, one of my serial runs took 9 seconds while another took 60 seconds. In addition, my runs for `comm_sz` of 8 were all pretty fast. This produced a speedup of ~8 for 77000 hashes. I do not believe this at all.

<u>Efficiency</u>

I don't trust the efficiency numbers the same reasons explained above. I shouldn't be getting efficiencies over 1.

Extra Credit

The most hashes generated in an hour. I had to modify my original program slightly in order to make it run forever. For each run I used a `comm_sz` of 8.

1. 32 bit: 470 collisions
2. 33 bit: 250 collisions
3. 34 bit: 150 collisions
4. 36 bit: 30 collisions

times

comm_sz	77000	110000
1	33.1940992	11.831883
2	14.52120798	11.19908362
4	16.1924754	8.7405694
8	3.77269	11.1294932
16	14.1686086	14.0479048
Average time in seconds		

speedup

comm_sz	77000	110000
1	-	-
2	2.285904812	1.056504568
4	2.049970643	1.353674167
8	8.798522858	1.063110672
16	2.34279174	0.8422525044

efficiency

comm_sz	77000	110000
1	-	-
2	1.142952406	0.5282522839
4	0.5124926606	0.3384185417
8	1.099815357	0.1328888341
16	0.1464244838	0.05264078153