Zhiyue Qiu

Algorithm:

I use ‘push’ algorithm and parallel it.

This algorithm and implementation detail for each process goes like this:

Main Algorithm:

```
For each iteration:
    initialize send and receive buffer
    for each vertex:
        calculate the ‘push’ value
        for each adjacent vertex:
            store the ‘push’ key in send buffer
            store the ‘push’ value in send buffer

    MPI_Scatterv: send data to each processor

    for each received chunk of data:
        update key value

    for each PR:
        calculate norm_changed

    MPI_Reduce on norm_changed and determine if terminate the loop
End
```

Detailed implementation:

Myid == 0:
read in data, Send chunks to each processor. Chunk size is determined by NNZ of each vertex. Goal is to make sure each processor have similar number of NNZ.
NNZ might close to E/p, which is the data, each processor should get.
Also calculate and broadcast block_size, total number of vertices, edges to each processor.

Each processor allocate sending and receiving buffer. The size should be the num it is to receive and send. Each processor should know, how much I need to send to and how much I need to receive from. Also calculate the displacement for MPI_Scatterv, which can send to each processor different size of data.

Do the algorithm above

Zhiyue Qiu

Complexity analysis:

```
For each iteration:
    initialize send and receive buffer    -> O(E/p)
    for each vertex:                      -> O(V/p + E/p)
        calculate the ‘push’ value
        for each adjacent vertex:
            store the ‘push’ key in send buffer
            store the ‘push’ value in send buffer


    MPI_Scatterv: send data to each processor
        -> p * one to all communication: p * O(log p*Ts+ (p-1)/p*E*Tw)

    for each received chunk of data:      -> O(E/p)
        update key value

    for each PR:                          -> O(V/p)
        calculate norm_changed

    MPI_Reduce on norm_changed and determine if terminate the loop
                                          -> O(log p * Ts + Tw)
End
```

Serial complexity: (in each iteration) O(V+E)
p*Tp = O(E) + O(V) + p*lop*Ts+(p-1)*E*Tw

Overhead: Mainly from scatter
p*log p * Ts + (p-1)*E*Tw

It behaves better on A.graph, because A.graph is well formed, every chunk of nodes only send data to another chunck, so the overhead might become something like p * Ts + E * Tw.

B.graph and live.graph, worse on B, Better on live. The reason is that based on my observation, live.graph is not that mess. But B is purely random.

So A is the best case, every processor only need to communicate with one other processor. B is the worst, the real scatter p times. The live.graph between them.

Zhiyue Qiu

Timing result

A.graph

```
mpirun -np 1 ./pagerank /export/scratch/CSCI5451_S18/assignment-4/
A.graph A-parallel.out
Number of iterations: 38 average time: 1.904s

mpirun -np 2 ./pagerank /export/scratch/CSCI5451_S18/assignment-4/
A.graph A-parallel.out
Number of iterations: 38 average time: 1.094s

mpirun -np 4 ./pagerank /export/scratch/CSCI5451_S18/assignment-4/
A.graph A-parallel.out
Number of iterations: 38 average time: 0.760s

mpirun -np 8 ./pagerank /export/scratch/CSCI5451_S18/assignment-4/
A.graph A-parallel.out
Number of iterations: 38 average time: 0.580s

mpirun -np 16 ./pagerank /export/scratch/CSCI5451_S18/assignment-4/
A.graph A-parallel.out
Number of iterations: 38 average time: 0.484s

mpirun -np 32 ./pagerank /export/scratch/CSCI5451_S18/assignment-4/
A.graph A-parallel.out
Number of iterations: 38 average time: 0.477s
```

B.graph

```
mpirun -np 1 ./pagerank /export/scratch/CSCI5451_S18/assignment-4/
B.graph B-parallel.out
Number of iterations: 8 average time: 4.304s

mpirun -np 2 ./pagerank /export/scratch/CSCI5451_S18/assignment-4/
B.graph B-parallel.out
Number of iterations: 8 average time: 1.522s

mpirun -np 4 ./pagerank /export/scratch/CSCI5451_S18/assignment-4/
B.graph B-parallel.out
Number of iterations: 8 average time: 0.972s

mpirun -np 8 ./pagerank /export/scratch/CSCI5451_S18/assignment-4/
B.graph B-parallel.out
Number of iterations: 8 average time: 0.694s

mpirun -np 16 ./pagerank /export/scratch/CSCI5451_S18/assignment-4/
B.graph B-parallel.out
Number of iterations: 8 average time: 0.577s

mpirun -np 32 ./pagerank /export/scratch/CSCI5451_S18/assignment-4/
B.graph B-parallel.out
```

Zhiyue Qiu

```
Number of iterations: 8 average time: 0.628s
```

live.graph

```
mpirun -np 1 ./pagerank /export/scratch/CSCI5451_S18/assignment-4/
live-journal.graph live-parallel.out
Number of iterations: 62 average time: 2.208s

mpirun -np 2 ./pagerank /export/scratch/CSCI5451_S18/assignment-4/
live-journal.graph live-parallel.out
Number of iterations: 62 average time: 1.338s

mpirun -np 4 ./pagerank /export/scratch/CSCI5451_S18/assignment-4/
live-journal.graph live-parallel.out
Number of iterations: 62 average time: 0.996s

mpirun -np 8 ./pagerank /export/scratch/CSCI5451_S18/assignment-4/
live-journal.graph live-parallel.out
Number of iterations: 62 average time: 0.806s

mpirun -np 16 ./pagerank /export/scratch/CSCI5451_S18/assignment-4/
live-journal.graph live-parallel.out
Number of iterations: 62 average time: 0.664s

mpirun -np 32 ./pagerank /export/scratch/CSCI5451_S18/assignment-4/
live-journal.graph live-parallel.out
Number of iterations: 62 average time: 0.799s
```