```
running on as8.al-salam.loc
as8.al-salam.loc
as8.al-salam.loc
as8.al-salam.loc
as8.al-salam.loc
as8.al-salam.loc
as8.al-salam.loc
as8.al-salam.loc
as8.al-salam.loc
as7.al-salam.loc
as7.al-salam.loc
as7.al-salam.loc
as7.al-salam.loc
as7.al-salam.loc
as7.al-salam.loc
as7.al-salam.loc
as7.al-salam.loc
as7.al-salam.loc
as6.al-salam.loc
as6.al-salam.loc
as6.al-salam.loc
as6.al-salam.loc
as6.al-salam.loc
as6.al-salam.loc
as6.al-salam.loc
as6.al-salam.loc
as6.al-salam.loc
as5.al-salam.loc
as5.al-salam.loc
as5.al-salam.loc
as5.al-salam.loc
as5.al-salam.loc
as5.al-salam.loc
as5.al-salam.loc
as5.al-salam.loc
as5.al-salam.loc
hostname is as8.al-salam.loc
on launch cwd is /cluster/home/esly14
PBS_O_WORKDIR is /cluster/home/esly14/cs360/slc

Contents of main-out.c:

// File: main-out.c
// Authors: Edward Ly, Byron Roosa, George Crowson
// Last Updated: 5 May 2016
// Single-linkage clustering program on strings using MPI implementation.
// This particular version outputs the sorted tuples and merged counts to stdout or to output
file for comparison.

/*
    master:
        load tuples into arrays
        sort tuples
        iterate tuples:
            send target to client
            wait for client to finish
            receive indices of merged tuples from client
```

```
            update tuples to reflect merges

    client:
        define stride of the decomposition
        load tuples into arrays
        sort tuples
        iterate tuples:
            receive target from master
            find first relevant tuple
            iterate through tuples doing merging
            send master the indices of merged tuples
*/

#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <string.h>
#include <math.h>
#include <sys/time.h>
#include <mpi.h>

const bool EVENTS = false;
const bool ERRORS = false;
const bool TIMING = false;

const bool SHOW_INPUT    = false;
const bool SHOW_SORT     = false;
const bool SHOW_PROGRESS = true;
const bool SHOW_RESULTS  = true;

const int MAX_TUPLE_COUNT = 1.5e6, CLIENT_STOP = -1;

const int threshold = 2, length = 600;
const char* input_filename =
"/cluster/home/charliep/courses/cs360/single-linkage-clustering/Iceland2014.trim.contigs.good.uni
que.good.filter.unique.count.fasta";

// because I'm not passing MPI crap around everywhere
int process_count, rank, namelen;
int client_process_count, client_rank;
char processor_name[MPI_MAX_PROCESSOR_NAME];

typedef struct {
    int count;
    int merged_count;  //for debugging
    int merged_target; //for debugging
    char merged;
    char string[600];
} Tuple;



void event(char* message) {
    if(EVENTS) fprintf(stderr, "event/%d: %s\n", rank, message);
```

```
}
void error(char* message) {
    if(ERRORS) fprintf(stderr, "ERROR/%d: %s\n", rank, message);

    MPI_Finalize();
    exit(-1);
}



double start_time, end_time;

void start_watch() { start_time = MPI_Wtime(); }
void stop_watch () { end_time   = MPI_Wtime(); }

void show_watch(char* message) {
    if(TIMING) {
        double duration = end_time - start_time;
        fprintf(stderr, "event/%d: %.6f seconds to %s\n", rank, duration, message);
    }
}



void show_tuples(Tuple* tuples, int length) {
    for(int i = 0; i < length; i++) {
        if(tuples[i].merged == 'M') continue;

        fprintf(stdout, "%s %d\n", tuples[i].string, tuples[i].count);
    }
}

int get_tuples(Tuple tuples[]) {
    char stream[length + 1];
    int tuple_count = 0;

    {
        event("opening file");
        FILE* fstream = fopen(input_filename, "r");
        if(fstream == NULL) error("file was unable to be opened");

        event("reading file");
        while(fscanf(fstream, "%s %d", stream, &tuples[tuple_count].count) != EOF) {
            strcpy(tuples[tuple_count].string, stream);
            tuples[tuple_count].merged = 'N';
            tuples[tuple_count].merged_count = 0;
            tuples[tuple_count].merged_target = -1;

            //iterate to the next token
            tuple_count++;
            if(tuple_count > MAX_TUPLE_COUNT) break;
        }

        event("closing file");
        fclose(fstream);
```

```
    }

    if(SHOW_INPUT) {
        fprintf(stderr, "input/%d:\n", rank);
        show_tuples(tuples, tuple_count);
    }

    return tuple_count;
}

void sort_tuples(Tuple tuples[], int length) {
    int compare_tuples(const void *a, const void *b) {
        Tuple *source = (Tuple*) a;
        Tuple *target = (Tuple*) b;
        return (target->count - source->count);
    }

    qsort(tuples, length, sizeof(Tuple), compare_tuples);

    if(SHOW_SORT) {
        fprintf(stderr, "sort/%d:\n", rank);
        show_tuples(tuples, length);
    }
}


bool is_match(char* target, char* other, int len, int max) {
    int diff = 0;
    for(int i = 0; i < len; i++) {
        if(target[i] != '-' && other[i] != '.')
            diff += (target[i] != other[i]);

        if(diff > max) return false;
    }
    return true;
}


void server() {
    if(client_process_count == 0) error("must be ran with 2 or more processes");

    start_watch();
    Tuple* tuples;
    int tuple_count;
    {
        event("populating tuples");
        tuples = (Tuple*) malloc(MAX_TUPLE_COUNT * sizeof(Tuple));
        tuple_count = get_tuples(tuples);

        event("sorting tuples");
        sort_tuples(tuples, tuple_count);
    }
    stop_watch();
```

```
    show_watch("load and sort tuples");

    //declaring the array that stores matched indices
    int decompose_count = (tuple_count + client_process_count - 1) / client_process_count; //
    round up
    int match_indices[decompose_count]; //#@note #could be abstracted out

    start_watch();
    event("server listening to clients");
    MPI_Status status;
    int match_count, match_count_sum;
    for(int i = 0; i < tuple_count; i++) {
        if(SHOW_PROGRESS && (i % 10000 == 0)) fprintf(stderr, "tuple: %d\n", i);

        Tuple* t = &tuples[i];

        if(t->merged != 'N') continue;
        t->merged = 'T';

        //send target to clients
        for(int j = 1; j < process_count; j++)
            MPI_Send(&i, 1, MPI_INT, j, 1, MPI_COMM_WORLD);

        //get merged indices from clients
        match_count_sum = 0;
        for(int j = 1; j < process_count; j++) {
            MPI_Recv(&match_count, 1, MPI_INT, j, 1, MPI_COMM_WORLD, &status);
            MPI_Recv(&match_indices, match_count, MPI_INT, j, 1, MPI_COMM_WORLD, &status);

            if(match_count > 0) {
                t->merged = 'C';
                t->merged_count = match_count;

                //resolve matches
                for(int k = 0; k < match_count; k++) {
                    int index = match_indices[k];
                    match_count_sum += tuples[index].count;
                    tuples[index].merged = 'M';
                    tuples[index].merged_target = i;
                }
            }
        }

        //updating count to reflect the merges
        t->count += match_count_sum;
    }
    stop_watch();
    show_watch("run the SLC algorithm on all tuples");

    event("releasing clients");
    for(int i = 1; i < process_count; i++)
        MPI_Send(&CLIENT_STOP, 1, MPI_INT, i, 1, MPI_COMM_WORLD);
```

```
    if(SHOW_RESULTS) show_tuples(tuples, tuple_count);

    event("freeing memory");
    free(tuples);
}

void client() {
    start_watch();
    Tuple* tuples;
    int tuple_count;
    {
        event("populating tuples");
        tuples = (Tuple*) malloc(MAX_TUPLE_COUNT * sizeof(Tuple));
        tuple_count = get_tuples(tuples);

        event("sorting tuples");
        sort_tuples(tuples, tuple_count);
    }
    stop_watch();
    show_watch("load and sort tuples");

    //declaring the array that stores matched indices
    int decompose_count = (tuple_count + client_process_count - 1) / client_process_count; //
    round up
    int match_indices[decompose_count];

    event("client processing tuples");
    MPI_Status status;
    int match_count = 0;
    while(true) {
        //get target
        int target_index;
        MPI_Recv(&target_index, 1, MPI_INT, 0, 1, MPI_COMM_WORLD, &status);
        if(target_index == CLIENT_STOP) break;
        Tuple* t = &tuples[target_index];
        t->merged = 'T';

        //find first tuple that needs checked
        int group_index = target_index - (target_index % client_process_count);
        int start_index = group_index + client_rank;

        //iterate through tuples
        //@note #tuples with the same count that were searched previously will have already
        been matched
        match_count = 0;
        for(int i = start_index; i < tuple_count; i += client_process_count) {
            Tuple* s = &tuples[i];
            if(s->merged != 'N') continue;

            //check if index is a match
            if(is_match(t->string, s->string, length, threshold)) {
                match_indices[match_count] = i;
                match_count += 1;
```

```
                s->merged = 'M';
                t->merged = 'C';
            }
        }

        //returning matched indices
        MPI_Send(&match_count, 1, MPI_INT, 0, 1, MPI_COMM_WORLD);
        MPI_Send(&match_indices, match_count, MPI_INT, 0, 1, MPI_COMM_WORLD);
    }

    event("freeing memory");
    free(tuples);
}


int main(int argc, char* argv[]) {
    event("starting MPI");
    {
        MPI_Init(&argc, &argv);
        MPI_Comm_size(MPI_COMM_WORLD, &process_count);
        MPI_Comm_rank(MPI_COMM_WORLD, &rank);
        MPI_Get_processor_name(processor_name, &namelen);

        client_process_count = process_count - 1;
        client_rank = rank - 1;
    }

    if(rank == 0) {
        event("starting server");
        server();
    } else {
        event("starting client");
        client();
    }

    event("ending MPI");
    MPI_Finalize();
    exit(0);
}
```

Contents of main-out.qsub:

```
#!/bin/bash

#
# NB: Make sure you change the -N, -l, -M and module load commands as appropriate.
# The binary path and batch file will also need to be adjusted, basically check
# every line...
#

#PBS -N esl-slc-mainout
#PBS -q as_default
#PBS -l nodes=4:ppn=8
```

```
#PBS -m bea
#PBS -M esly14@earlham.edu

echo "running on `cat $PBS_NODEFILE`"
echo "hostname is `hostname`"
echo "on launch cwd is `pwd`"
echo "PBS_O_WORKDIR is `echo $PBS_O_WORKDIR`"

cd $PBS_O_WORKDIR
echo ""
echo "Contents of main-out.c:"
echo ""
cat main-out.c
echo ""
echo "Contents of main-out.qsub:"
echo ""
cat main-out.qsub
mpicc -O0 -g -Wall -lm -o main-out main-out.c
time mpirun -np 32 ./main-out > main-out.dat
echo ""
echo "Diff result:"
diff --ignore-all-space main-out.dat
~charliep/courses/cs360/single-linkage-clustering/Iceland2014-precluster-canon.dat | wc -l

Diff result:
279259
```