```
 +----------------------------------+
|         CIS 520                  |
| PROJECT 4: One Program, Three ways |
|     DESIGN DOCUMENT              |
+----------------------------------+
```

—- GROUP —-

Jarod DeWeese Kaleb Cox Weston Chan

— Analysis —

All 3 attempts seem to work fine. We unfortunately waited a bit too last minute to perform adequate analysis through Beocat. The multiple methods seemed to run roughly equally well for the small scale (about 500 lines) tests we were able to conduct through the headnode.

Appendix 1: Small scale output

```
0-1: 32923
1-2: -144845
2-3: 54368
3-4: -71061
4-5: 2101
5-6: -2187
6-7: -3137
7-8: 6344
8-9: -4350
9-10: 4558
10-11: 3039
11-12: 135433
12-13: -135721
13-14: 154007
14-15: -164651
15-16: 156641
16-17: -4546
17-18: 14809
18-19: -26528
19-20: 6322
20-21: -138054
21-22: 35903
22-23: -14261
23-24: 130131
24-25: -159709
25-26: 150417
26-27: -141415
27-28: 3493
```

28-29: 127309
29-30: 12523
30-31: -145174
31-32: 28979
32-33: 104102
33-34: -117276
34-35: -15487
35-36: 136449
36-37: -144032
37-38: -1779
38-39: 156143
39-40: -135656
40-41: 139426
41-42: 5371
42-43: 2978
43-44: -72240
44-45: -63548
45-46: 101234
46-47: -87363
47-48: -47957
48-49: 6798
49-50: -3632
50-51: 4848
51-52: 1142
52-53: 8649
53-54: -7922
54-55: 51154
55-56: 53536
56-57: -101903
57-58: -1960
58-59: -2559
59-60: -4652
60-61: 130896
61-62: -127862
62-63: 148807
63-64: -149376
64-65: 114949
65-66: -78167
66-67: -37373
67-68: 13089
68-69: 2828
69-70: -13441
70-71: 5746
71-72: -7857
72-73: 10427
73-74: 2989

```
74-75: -15303
75-76: 4043
76-77: 71966
77-78: 56251
78-79: -128003
79-80: 99641
80-81: -2498
81-82: -112661
82-83: 14361
83-84: 36885
84-85: -14814
85-86: -15297
86-87: 9776
87-88: 50498
88-89: -69617
89-90: 1034
90-91: 79714
91-92: 49706
92-93: -80504
93-94: -23558
94-95: -23559
95-96: 8407
96-97: -4460
97-98: 1658
98-99: 12984
```

Appendix 2: OpenMP

```
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define NUM_THREADS 10

#define NUMBER_LINES 100
#define MAX_LINE_SIZE 2005 //It seems the max line size is 2001, but that is only a guess

// Contains the output
char char_array[NUMBER_LINES][MAX_LINE_SIZE];
int line_sum[NUMBER_LINES - 1];

void init_arrays(){
    int i;

    for (i = 0; i < NUMBER_LINES - 1; i++){
        line_sum[i] = 0;
```

```
    }

    //Read in the main file into char_array
    FILE* file = fopen("/homes/dan/625/wiki_dump.txt", "r");
    // FILE* file = fopen("test_file.txt", "r");
    for(i = 0; i < NUMBER_LINES; i++) {
        if (file == NULL) {
            printf("File not Found\n");
            exit(-1);
        }
        fgets(char_array[i], MAX_LINE_SIZE, file);

        //printf("%d: %s", i, char_array[i]);
    }
    fclose(file);
    printf("\n");
}

void count_chunk(int id){
    char theChar;
    int i, j, sum, startLine, endLine;
    int local_line_sum[NUMBER_LINES / NUM_THREADS];

    #pragma omp private(id, theChar, i,j,sum,startLine,endLine,local_line_sum)
    {
        sum = 0;
        startLine = id * (NUMBER_LINES / NUM_THREADS);
        endLine = startLine + (NUMBER_LINES / NUM_THREADS);
        if (id == NUM_THREADS - 1) endLine = NUMBER_LINES;

        // Ready for parallelization

        printf("ID: %d\n", id);
        printf("Start line: %d\n", startLine);
        printf("End line: %d\n", endLine - 1);

        for(i = startLine; i < endLine; i++) {
            sum = 0;
            for(j = 0; j < MAX_LINE_SIZE; j++){
                theChar = char_array[i][j];
                if(theChar == '\0'){
                    // printf("Length: %d\n",j); //Used to test how long to make each str
                    break;
                }
                sum += theChar;
            }
```

```
            local_line_sum[i - startLine] = sum;
        }

        // Critical Section
        #pragma omp critical
        {
            for(i = startLine; i < endLine; i++) {
                if(i != 0) line_sum[i-1] -= local_line_sum[i-startLine];
                if(i != NUMBER_LINES-1) line_sum[i] += local_line_sum[i-startLine];
                // printf("\t%d: %d\n", i, local_line_sum[i-startLine]);
            }

            // printf("\n");
        }
    }
}

void print_results(){
    int i;

    for(i = 0; i < NUMBER_LINES - 1; i++){
        printf("%d-%d: %d\n",i,i+1,line_sum[i]);
    }
}

main(int argc, char *argv[]){
    if(NUM_THREADS > NUMBER_LINES) {
        printf("Number of chunks must be less than number of lines\n");
        return -1;
    }

    omp_set_num_threads(NUM_THREADS);

    init_arrays();

    #pragma omp parallel
    {
        count_chunk(omp_get_thread_num());
    }

    print_results();
}
```

Appendix 3: pthread

```
#include <pthread.h>
#include <stdio.h>
```

```c
#include <stdlib.h>
#include <string.h>

#define NUM_THREADS 10
#define NUMBER_LINES 100
#define MAX_LINE_SIZE 2005 //It seems the max line size is 2001, but that is only a guess

pthread_mutex_t mutexsum;                       // mutex for line_sum

// Contains the output
char char_array[NUMBER_LINES][MAX_LINE_SIZE];
int line_sum[NUMBER_LINES - 1];

void init_arrays(){
    int i;

    for (i = 0; i < NUMBER_LINES - 1; i++){
        line_sum[i] = 0;
    }

    //Read in the main file into char_array
    FILE* file = fopen("/homes/dan/625/wiki_dump.txt", "r");
    // FILE* file = fopen("test_file.txt", "r");
    for(i = 0; i < NUMBER_LINES; i++) {
        if (file == NULL) {
            printf("File not Found\n");
            exit(-1);
        }
        fgets(char_array[i], MAX_LINE_SIZE, file);

        //printf("%d: %s", i, char_array[i]);
    }
    fclose(file);
    printf("\n");
}

void count_chunk(int id){
    char theChar;
    int i, j, sum, startLine, endLine;
    int local_line_sum[NUMBER_LINES / NUM_THREADS];


    sum = 0;
    startLine = id * (NUMBER_LINES / NUM_THREADS);
    endLine = startLine + (NUMBER_LINES / NUM_THREADS);
    if (id == NUM_THREADS - 1) endLine = NUMBER_LINES;
```

```
    // Ready for parallelization

    printf("ID: %d\n", id);
    printf("Start line: %d\n", startLine);
    printf("End line: %d\n", endLine - 1);

    for(i = startLine; i < endLine; i++) {
        sum = 0;
        for(j = 0; j < MAX_LINE_SIZE; j++){
            theChar = char_array[i][j];
            if(theChar == '\0'){
                // printf("Length: %d\n",j); //Used to test how long to make each str
                break;
            }
            sum += theChar;
        }
        local_line_sum[i - startLine] = sum;
    }

    // Critical Section
    pthread_mutex_lock (&mutexsum);

    for(i = startLine; i < endLine; i++) {
        if(i != 0) line_sum[i-1] -= local_line_sum[i-startLine];
        if(i != NUMBER_LINES-1) line_sum[i] += local_line_sum[i-startLine];
        // printf("\t%d: %d\n", i, local_line_sum[i-startLine]);
    }
    printf("\n");

    pthread_mutex_unlock (&mutexsum);

    pthread_exit(NULL);

}

void print_results(){
    int i;

    for(i = 0; i < NUMBER_LINES - 1; i++){
        printf("%d-%d: %d\n",i,i+1,line_sum[i]);
    }
}

main(){
    int i, rc;
```

```
    pthread_t threads[NUM_THREADS];
    pthread_attr_t attr;
    void *status;


    /* Initialize and set thread detached attribute */
    pthread_attr_init(&attr);
    pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_JOINABLE);



    if(NUM_THREADS > NUMBER_LINES) {
        printf("Number of chunks must be less than number of lines\n");
        return -1;
    }

    init_arrays();


    for (i = 0; i < NUM_THREADS; i++ ) {
        rc = pthread_create(&threads[i], &attr, count_chunk, (void *)i);
        if (rc) {
            printf("ERROR; return code from pthread_create() is %d\n", rc);
            exit(-1);
        }
    }

 /* Free attribute and wait for the other threads */
    pthread_attr_destroy(&attr);
    for(i=0; i<NUM_THREADS; i++) {
        rc = pthread_join(threads[i], &status);
        if (rc) {
            printf("ERROR; return code from pthread_join() is %d\n", rc);
            exit(-1);
        }
    }

    print_results();

    pthread_mutex_destroy(&mutexsum);
    printf("Main: program completed. Exiting.\n");
    pthread_exit(NULL);

}
```

Appendix 4: MPI

```c
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

//#define NUM_THREADS 1
int NUM_THREADS;

#define NUMBER_LINES 100
#define MAX_LINE_SIZE 2005 //It seems the max line size is 2001, but that is only a guess

// Contains the output
char char_array[NUMBER_LINES][MAX_LINE_SIZE];
int line_sum[NUMBER_LINES];
int local_line_sum[NUMBER_LINES];

void init_arrays(){
    int i;

    for (i = 0; i < NUMBER_LINES - 1; i++){
        line_sum[i] = 0;
    }

    //Read in the main file into char_array
    FILE* file = fopen("/homes/dan/625/wiki_dump.txt", "r");
    // FILE* file = fopen("test_file.txt", "r");
    for(i = 0; i < NUMBER_LINES; i++) {
        if (file == NULL) {
            printf("File not Found\n");
            exit(-1);
        }
        fgets(char_array[i], MAX_LINE_SIZE, file);

        //printf("%d: %s", i, char_array[i]);
    }
    fclose(file);
    printf("\n");
}

void count_chunk(void *rank){
    char theChar;
    int i, j, sum, startLine, endLine;
    int id = *((int *) rank);

    sum = 0;
    startLine = id * (NUMBER_LINES / NUM_THREADS);
```

```
    endLine = startLine + (NUMBER_LINES / NUM_THREADS);
    if (id == NUM_THREADS - 1) endLine = NUMBER_LINES;

    // Ready for parallelization

    printf("ID: %d\n", id);
    printf("Start line: %d\n", startLine);
    printf("End line: %d\n", endLine - 1);

    for(i = startLine; i < endLine; i++) {
        sum = 0;
        for(j = 0; j < MAX_LINE_SIZE; j++){
            theChar = char_array[i][j];
            if(theChar == '\0'){
                // printf("Length: %d\n",j); //Used to test how long to make each str
                break;
            }
            sum += theChar;
        }
        local_line_sum[i] = sum;

    }

}

void print_results(){
    int i;

    for(i = 0; i < NUMBER_LINES - 1; i++){
        printf("%d-%d: %d\n",i,i+1,line_sum[i]-line_sum[i+1]); //The computation moves here
                                                               //It could instead go into count but w
        fflush(stdout);
    }
}

main(int argc, char *argv[]){

    int i, rc;
    int numtasks, rank;
    MPI_Status Status;

    rc = MPI_Init(&argc,&argv);
    if (rc != MPI_SUCCESS) {
        printf ("Error starting MPI program. Terminating.\n");
        MPI_Abort(MPI_COMM_WORLD, rc);
    }
```

```
    MPI_Comm_size(MPI_COMM_WORLD,&numtasks);
    MPI_Comm_rank(MPI_COMM_WORLD,&rank);

    NUM_THREADS = numtasks;
    printf("size = %d rank = %d\n", numtasks, rank);
    fflush(stdout);

    if ( rank == 0 ) {
            init_arrays();
    }
    MPI_Bcast(char_array, NUMBER_LINES * MAX_LINE_SIZE, MPI_CHAR, 0, MPI_COMM_WORLD);

    count_chunk(&rank);

    MPI_Reduce(local_line_sum, line_sum, NUMBER_LINES, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);


    if ( rank == 0 ) {
            print_results();
    }

    MPI_Finalize();
    return 0;
}
```

Appendix 5: The parallelized serial code

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define NUMBER_CHUNKS 1
#define NUMBER_LINES 500
#define MAX_LINE_SIZE 2005 //It seems the max line size is 2001, but that is only a guess

// Contains the output
char char_array[NUMBER_LINES][MAX_LINE_SIZE];
int line_sum[NUMBER_LINES - 1];

void init_arrays(){
    int i;

    for (i = 0; i < NUMBER_LINES - 1; i++){
        line_sum[i] = 0;
    }
```

```
    //Read in the main file into char_array
    FILE* file = fopen("/homes/dan/625/wiki_dump.txt", "r");
    // FILE* file = fopen("test_file.txt", "r");
    for(i = 0; i < NUMBER_LINES; i++) {
        if (file == NULL) {
            printf("File not Found\n");
            exit(-1);
        }
        fgets(char_array[i], MAX_LINE_SIZE, file);

        //printf("%d: %s", i, char_array[i]);
    }
    fclose(file);
    printf("\n");
}

void count_chunk(int id){
    char theChar;
    int i, j, sum, startLine, endLine;
    int local_line_sum[NUMBER_LINES / NUMBER_CHUNKS];


    sum = 0;
    startLine = id * (NUMBER_LINES / NUMBER_CHUNKS);
    endLine = startLine + (NUMBER_LINES / NUMBER_CHUNKS);
    if (id == NUMBER_CHUNKS - 1) endLine = NUMBER_LINES;

    // Ready for parallelization

    printf("ID: %d\n", id);
    printf("Start line: %d\n", startLine);
    printf("End line: %d\n", endLine - 1);

    for(i = startLine; i < endLine; i++) {
        sum = 0;
        for(j = 0; j < MAX_LINE_SIZE; j++){
            theChar = char_array[i][j];
            if(theChar == '\0'){
                // printf("Length: %d\n",j); //Used to test how long to make each str
                break;
            }
            sum += theChar;
        }
        local_line_sum[i - startLine] = sum;
    }
```

```
    // Critical Section
    for(i = startLine; i < endLine; i++) {
        if(i != 0) line_sum[i-1] -= local_line_sum[i-startLine];
        if(i != NUMBER_LINES-1) line_sum[i] += local_line_sum[i-startLine];
        // printf("\t%d: %d\n", i, local_line_sum[i-startLine]);
    }

    printf("\n");
}

void print_results(){
    int i;

    for(i = 0; i < NUMBER_LINES - 1; i++){
        printf("%d-%d: %d\n",i,i+1,line_sum[i]);
    }
}

main(int argc, char *argv[]){
    if(NUMBER_CHUNKS > NUMBER_LINES) {
        printf("Number of chunks must be less than number of lines\n");
        return -1;
    }
    int i;

    init_arrays();

    for(i = 0; i < NUMBER_CHUNKS; i++){
        count_chunk(i);
    }
    print_results();
}
```

Appendix 6: Sample script for OpenMP

```
#!/bin/bash

#SBATCH --ntasks=1
#SBATCH --nodes=1
SCRIPTPATH=$(dirname "$SCRIPT")
$SCRIPTPATH/omp_1_thread
```

Appendix 7: Sample script for Pthread

```
#!/bin/bash
```

```
#SBATCH --ntasks=1
#SBATCH --nodes=1
SCRIPTPATH=$(dirname "$SCRIPT")
$SCRIPTPATH/pthread_1_thread
```

Appendix 8: Sample script for MPI

```
#!/bin/bash

mpirun -npernode 1 mpi
```