CIS520 - Project 4: Performance Analysis

Members: Benjamin Cooper, Bethany Weddle, Mir Fahim Anwar
Date: May 14, 2020

Pthread Observation and Description:
In the observation of runtimes, it appears that the more cores the better, although having more threads does not make for faster runtime all the time. An accident test was run with 444 threads with 4 cores, but the runtime was not far from the result of 4 threads with 4 cores. **Bold = (Best Column)**, **Bold & Underline = (Best Row)**, Purple = Shared

Pthreads was the simplest to understand and implement. The pthread_create and join made it really simple to start the parallelism and join all the threads to work together. We accounted for synchronization by having a start and end point for the threads that is calculated by the MAX_THREADS, and the id of the thread that is passed by pthread_create method to the FindSums method. The results do not outprint in order of line numbers, but the results are correct and consistent.

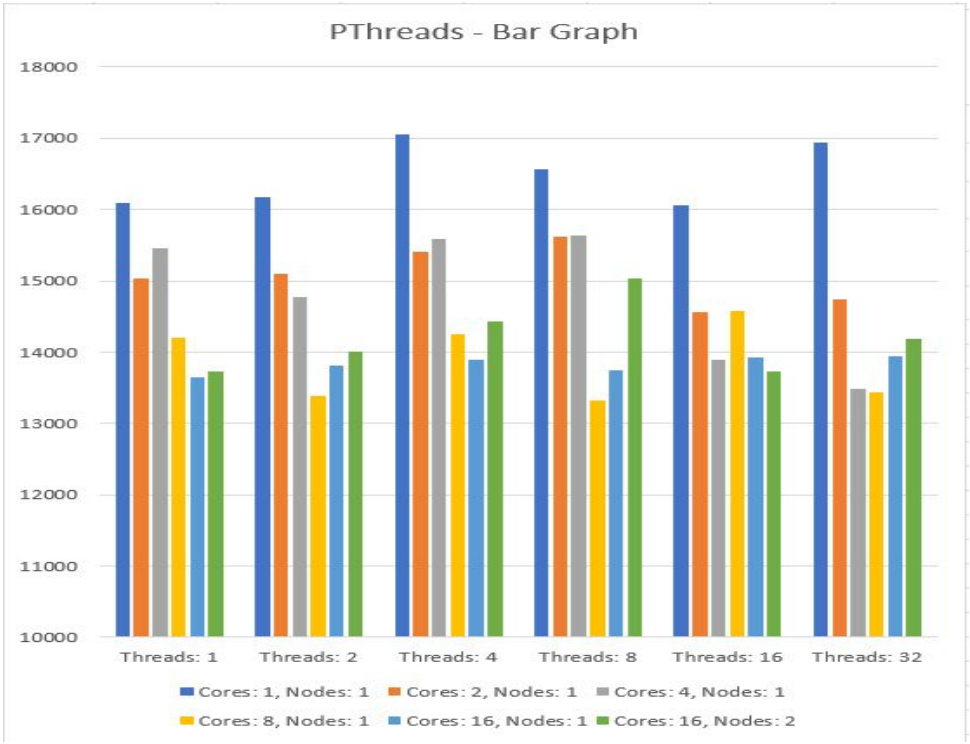
Pthread Runtime (ms):

Pthread	Cores: 1, Nodes: 1	Cores: 2, Nodes: 1	Cores: 4, Nodes: 1	Cores: 8, Nodes: 1	Cores: 16, Nodes: 1	Cores: 16, Nodes: 2
Threads: 1	16088.589	15040.830	15454.845	14206.231	**13646.415**	**13723.885**
Threads: 2	16169.847	15102.817	14776.748	**13387.157**	13809.272	14003.610
Threads: 4	17047.464	15399.775	15590.479	14258.153	**13894.574**	14431.968
Threads: 8	16571.219	15621.84	15643.283	13316.115	13740.383	15037.787
Threads: 16	**16059.787**	**14566.352**	13899.864	14581.026	13916.952	**13735.689**
Threads: 32	16938.594	14734.625	**13481.28**	**13437.882**	13945.842	14181.025

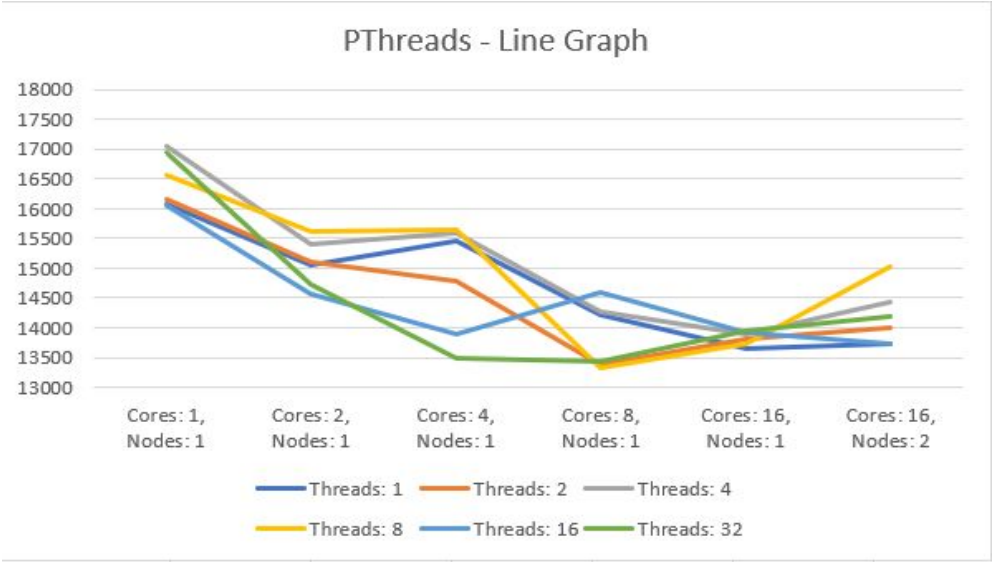
First 50 Lines Output:

line 0-1: -32923
line 1-2: 144845
line 2-3: -54368
line 3-4: 71061
line 4-5: -2101
line 5-6: 2187
line 25-26: -150417
line 26-27: 141415
line 27-28: -3493
line 28-29: -127309
line 29-30: -12523
line 6-7: 3137
line 7-8: -6344
line 8-9: 4350
line 9-10: -4558
line 10-11: -3039
line 30-31: 145174
line 31-32: -28979
line 32-33: -104102
line 33-34: 117276
line 34-35: 15487
line 35-36: -136449
line 36-37: 144032
line 37-38: 1779
line 11-12: -135433
line 12-13: 135721
line 13-14: -154007
line 38-39: -156143
line 39-40: 135656
line 40-41: -139426
line 41-42: -5371
line 42-43: -2978
line 43-44: 72240
line 44-45: 63548
line 45-46: -101234
line 46-47: 87363
line 47-48: 47957
line 48-49: -6798
line 49-50: 3632
line 14-15: 164651
line 15-16: -156641
line 16-17: 4546
line 17-18: -14809
line 18-19: 26528
line 19-20: -6322
line 20-21: 138054
line 21-22: -35903
line 22-23: 14261
line 23-24: -130131
line 24-25: 159709

Y-Axis Represents Milliseconds



Here, we clearly see that the parallel use of multiple CPUs consistently resulted in better runtimes compared to single-core execution. However, this effect does not scale for larger numbers of cores. For instance, our results show that there is no consistent difference between 8 and 16 core executions.



This graph shows there is a clear downward trend in runtime for increasing numbers of cores, up to about 8.

OpenMP Observation and Description:
In runtimes again appear to not be overly affected by the number of threads, but with the increase of cores, the runtimes become smaller. It also appears that OpenMP is slower than Pthreads in its execution of the program. **Bold = (Best Column)**, **Bold & Underline = (Best Row)**, Purple = Shared

OpenMP's code went through multiple revisions of trying to read the file in parallel as well as finding the sum differences. This did not work out because the threads kept interrupting each other to do the work, and the data would be inconsistent. So, we changed it to read the file into memory, and then traverse through an array of sums to show the differences. By OpenMP made parallel simple to start with the #pragma omp parallel, although it took **many** trial and error before we found the solution to synchronizing the threads.

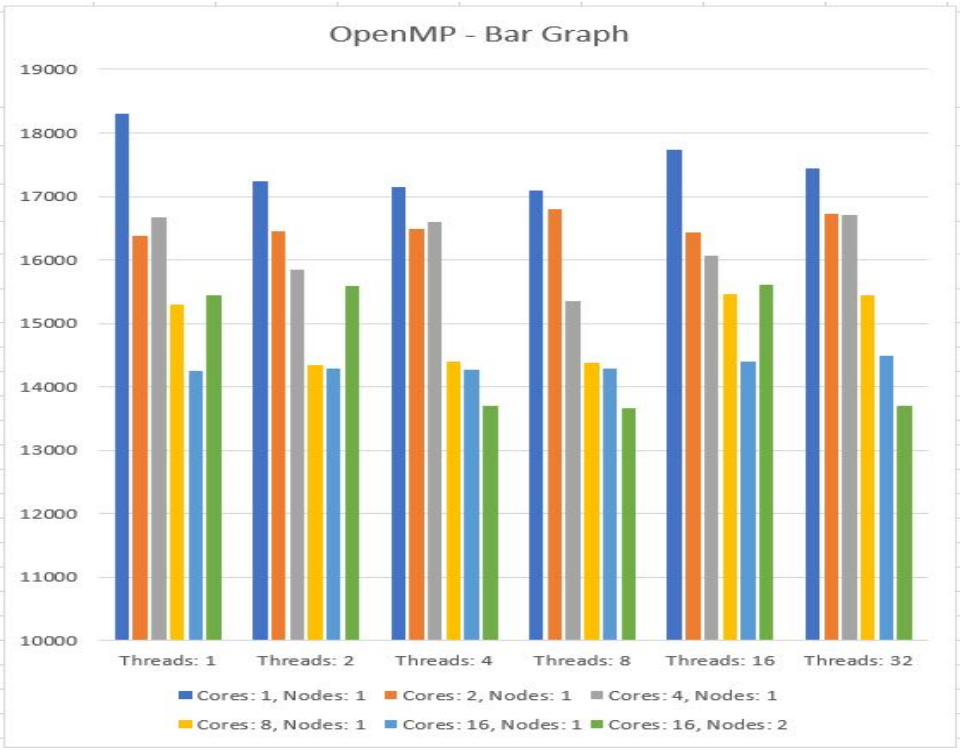
OpenMP Runtime (ms):

OpenMP	Cores: 1, Nodes: 1	Cores: 2, Nodes: 1	Cores: 4, Nodes: 1	Cores: 8, Nodes: 1	Cores: 16, Nodes: 1	Cores: 16, Nodes: 2
Threads: 1	18309.102	**16383.792**	16677.015	15299.809	**14244.468**	15444.925
Threads: 2	17240.266	16451.943	15853.849	**14347.395**	**14279.383**	15593.810
Threads: 4	17155.521	16494.000	16599.727	14395.587	14271.495	**13702.234**
Threads: 8	**17097.161**	16805.256	**15350.794**	14372.312	14293.426	**13662.298**
Threads: 16	17727.337	16423.814	16056.780	15461.847	**14398.244**	15600.227
Threads: 32	17437.464	16720.983	16697.874	15437.794	14479.559	**13694.473**

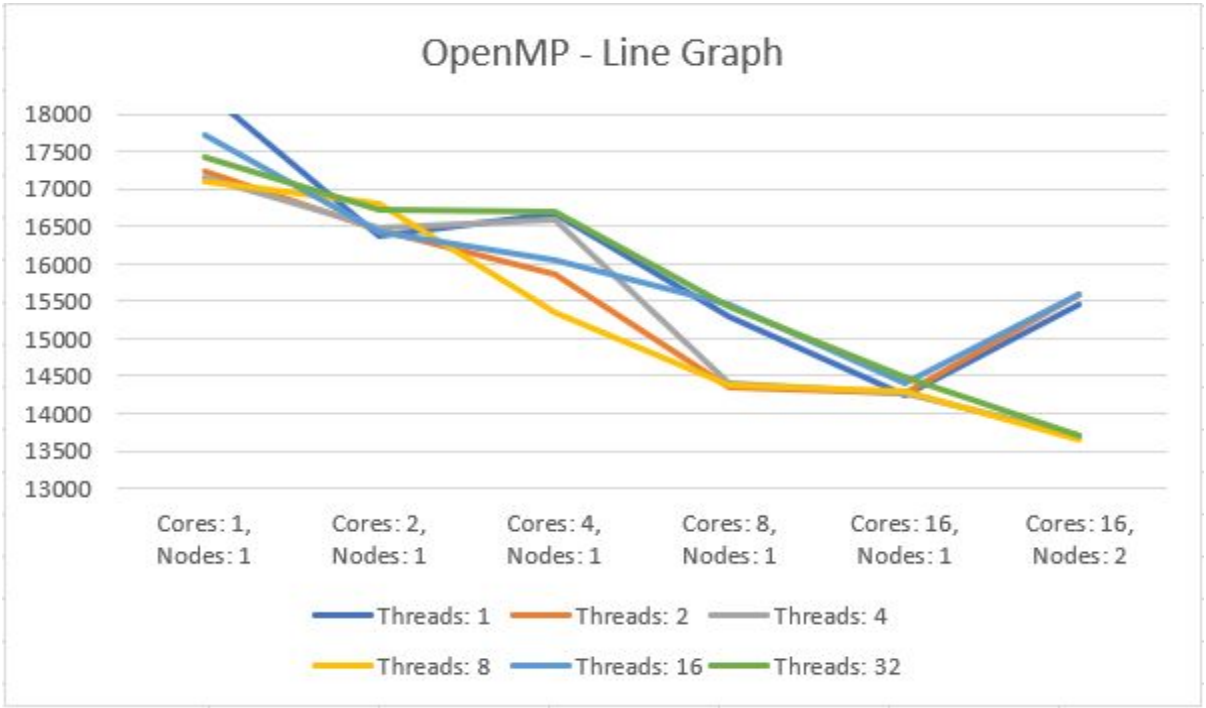
First 50 Lines Output:

line 0-1: 32923
line 1-2: -144845
line 2-3: 54368
line 3-4: -71061
line 4-5: 2101
line 5-6: -2187
line 6-7: -3137
line 7-8: 6344
line 8-9: -4350
line 9-10: 4558
line 10-11: 3039
line 11-12: 135433
line 12-13: -135721
line 13-14: 154007
line 14-15: -164651
line 15-16: 156641
line 16-17: -4546
line 17-18: 14809
line 18-19: -26528
line 19-20: 6322
line 20-21: -138054
line 21-22: 35903
line 22-23: -14261
line 23-24: 130131
line 24-25: -159709
line 25-26: 150417
line 26-27: -141415
line 27-28: 3493
line 28-29: 127309
line 29-30: 12523
line 30-31: -145174
line 31-32: 28979
line 32-33: 104102
line 33-34: -117276
line 34-35: -15487
line 35-36: 136449
line 36-37: -144032
line 37-38: -1779
line 38-39: 156143
line 39-40: -135656
line 40-41: 139426
line 41-42: 5371
line 42-43: 2978
line 43-44: -72240
line 44-45: -63548
line 45-46: 101234
line 46-47: -87363
line 47-48: -47957
line 48-49: 6798
line 49-50: -3632

Y-Axis Represents Milliseconds



This graph shows the same general trends as the one for Pthreads.



This graph shows that the running times do trend downward as the cores increase. Although using multiple nodes sometimes results in worse performance.

MPI Observation and Description:

MPI was the most frustrating to implement because running it on the head node appeared to display the correct differences for the lines, although once submitting jobs on beocat, the output would contain the data displaying multiple times. This was causing the jobs to take exceptionally longer times than we expected. It turns out we were using “mpirun” in the slurm launch script, which caused slurm to run n^2 copies of the process instead of n. Correcting this small syntax error, allowed us to run our MPI program with varying core/node configurations.

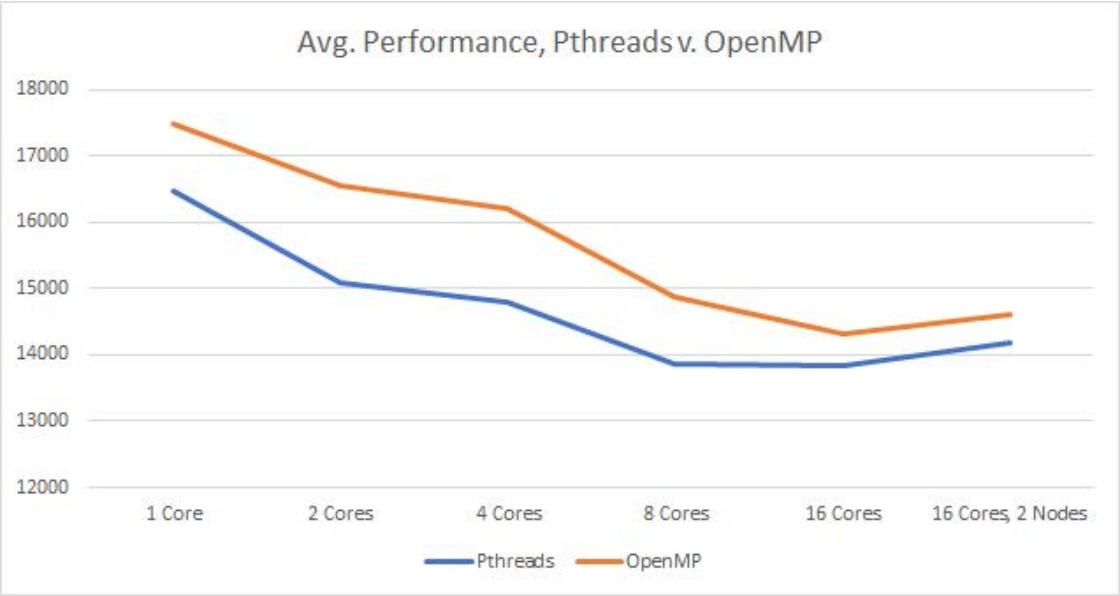
MPI does not explicitly use threads in the same way as pthread and openMP do. Parallelism is implemented by running copies of a single binary as separate processes, usually on separate CPU cores. So for the MPI jobs, we specified the cores and the number of nodes among which those cores are distributed to get the following results.

MPI Runtime (ms):

MPI	Cores: 1 Node: 1	Cores: 2 Node: 1	Cores: 4 Node: 1	Cores: 8 Node: 1	Cores: 16 Node: 1,2	Cores: 32 Node: 2	Cores: 64 Node: 4
	39048.937	38613.447	36648.339	37586.933	38702.484	43460.442	45356.267
					38883.399		

First 50 Lines 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: 363



This graph shows how the pthread program is consistently faster on average than openMP for all thread and cores combinations.

From this graph, we can see that our implementation of MPI failed to successfully use parallelism to speed up processing. The runtimes remained fairly consistent across multiple core/node configurations. We believe the slight upward trend in runtime at higher core counts is due to the overhead incurred by coordinating multiple processes.

Pthread Code:

```
#include <stdlib.h>
#include <stdio.h>
#include <stdint.h>
#include <sys/time.h>
#include <pthread.h>
#define MAX_THREADS 32
#define BUFFSIZE 1000000
#define FILENAME "/homes/dan/625/wiki_dump.txt"

int sums[BUFFSIZE];
int count = 0;

void *FindSums(void *arg)
{
    int j;
    int id = (uintptr_t)arg;
    int start = id  * (BUFFSIZE / MAX_THREADS);
    int end = start + (BUFFSIZE /MAX_THREADS);


    // j needs to start where thread left off from
    for(j = start; j <  end; j++){
        printf("tid-%d  line %d-%d: %d\n", pthread_self(), j,j+1,(sums[j+1]-sums[j]));
    }

    pthread_exit((void*) arg);
}

int main()
{
    struct timeval start, end;
    double elapsedTime;
    int numSlots, myVersion = 2; // pthreads = 1, openmp = 2, mpi = 3

    FILE *fp;
    char c = 0;  // To store a character read from file
    int sum = 0;
    int i = 0;

    int t, rc;
    pthread_t threads[MAX_THREADS];
    pthread_attr_t attr;
    void *status;
    /* Initialize and set thread detached attribute */
    pthread_attr_init(&attr);
    pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_JOINABLE);


    gettimeofday(&start, NULL);
    // Open the file
    fp = fopen(FILENAME, "r");
```

```
    // Check if file exists
    if (fp == NULL)
    {
       printf("Could not open file"); // breaks here
       return 0;
    }

    // Extract characters from file and store in character c
       for(i=0; i <= BUFFSIZE; i++) {
      c = getc(fp);
      while(c != '\n') {
            if(c == EOF) {
              break;
            }
        sum += (int)c;
        c = getc(fp);
          }
      sums[i] = sum;
      count++;
      sum = 0;
       }

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

    pthread_attr_destroy(&attr);
    for(t=0; t< MAX_THREADS; t++)
    {
      rc = pthread_join(threads[t], &status);
      if (rc)
      {
        printf("ERROR; return code from pthread_join() is %d\n", rc);
        exit(-1);
      }
    }

   gettimeofday(&end, NULL);

   elapsedTime = (end.tv_sec - start.tv_sec) * 1000.0; //sec to ms
   elapsedTime += (end.tv_usec - start.tv_usec) / 1000.0; // us to ms
   printf("DATA, %d, %s, %f, %d\n", myVersion, getenv("NSLOTS"),  elapsedTime, MAX_THREADS );

   fclose(fp);
   return 0;
}
```

OpenMP Code:

```
#include <stdlib.h>
#include <stdio.h>
#include <sys/time.h>
#include <omp.h>
#define MAX_THREADS 32
#define NUMLINES 1000000
#define FILENAME "/homes/dan/625/wiki_dump.txt"

int main() {
  struct timeval start, end;
  double elapsedTime;
  int numSlots, myVersion = 1; // omp = 1, pthread = 2, mpi = 3

  FILE * fp;
  int count = 0; // tracks total number of lines read
  char c = 0; // stores the char read from file
  int sum = 0; // the sum of a line's chars
  int * sums = malloc(NUMLINES * sizeof(int)); // a buffer to hold line sums
  int i = 0;
  int j = 0;

  omp_set_num_threads(MAX_THREADS);

  // Open the file
  fp = fopen(FILENAME, "r");

  // Check if file exists
  if (fp == NULL) {
    printf("Failed to open file");
    return -1;
  }
  printf("File opened successfully\n");

  gettimeofday(&start, NULL);
  // loop which reads characters from the file, stopping when EOF is reached or buffer is full
  while(c = getc(fp), c != EOF && i < NUMLINES) {
    if (c == '\n') {
      sums[i] = sum; // save line sum to sums buffer
      i++;
      sum = 0; // reset value for next line
    } else {
      sum += (int)c; // add each char to line sum
    }
  }
  count += i; // count tracks total number of lines across all loops

  #pragma omp parallel
  {
    #pragma omp for ordered
    for (j = 0; j < count-1; j++) {
      #pragma omp ordered
      printf("(thread %d) line %d-%d: %d\n", omp_get_thread_num(), j, j + 1, (sums[j + 1] - sums[j]));
    }
  }

  gettimeofday(&end, NULL);

  elapsedTime = (end.tv_sec - start.tv_sec) * 1000.0; //sec to ms
  elapsedTime += (end.tv_usec - start.tv_usec) / 1000.0; // us to ms
  printf("DATA, %d, %s, %f, %d\n", myVersion, getenv("NSLOTS"),  elapsedTime, MAX_THREADS);

  fclose(fp);
  return 0;
}
```

MPI Code:

```
#include <stdlib.h>
#include <stdio.h>
#include <mpi.h>
#include <sys/time.h>

#define BUFFSIZE 1000000
#define FILENAME "/homes/dan/625/wiki_dump.txt"

int num_proc;
int sums[BUFFSIZE];

void ReadFile()
{
    FILE *fp;
    int i = 0;
    int sum = 0;
    char c = 0;

    // Open the file
    fp = fopen(FILENAME, "r");

    // Check if file exists
    if (fp == NULL)
    {
       printf("Could not open file"); // breaks here
    }

    // Extract characters from file and store in character c
       for(i=0; i <= BUFFSIZE; i++) {
         c = getc(fp);
         while(c != '\n') {
            if(c == EOF) {
              break;
            }
           sum += (int)c;
           c = getc(fp);
          }
         sums[i] = sum;
         sum = 0;
       }

   fclose(fp);
}

void FindSums(int *rank)
{

    int j;
    int id = *rank;
    int start = id * (BUFFSIZE / num_proc);
    int end = start + (BUFFSIZE / num_proc);
```

```
    for(j = start; j < end; j++)
    {
       printf("%d-%d: %d\n",j,j+1,sums[j+1]-sums[j]);
    }
}


int main(int argc, char* argv[])
{
    struct timeval start, end;
    double elapsedTime;
    int numSlots, myVersion = 3;

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


    gettimeofday(&start, NULL);
    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);

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

    num_proc = numtasks;

    MPI_Bcast(sums, BUFFSIZE, MPI_INT, 0, MPI_COMM_WORLD);
    FindSums(&rank);
    MPI_Barrier(MPI_COMM_WORLD);

    MPI_Finalize();
    if(rank == 0) {
        gettimeofday(&end, NULL);
       elapsedTime = (end.tv_sec - start.tv_sec) * 1000.0; //sec to ms
       elapsedTime += (end.tv_usec - start.tv_usec) / 1000.0; // us to ms
       printf("DATA, %d, %s, %f, %d\n", myVersion, getenv("SLURM_CPUS_ON_NODE"),
elapsedTime, num_proc);
    }

    //MPI_Finalize();
    return 0;
}
```

Links:

https://github.com/ChristianJHughes/pintos-project4/blob/master/sample_code/proj4_code_beta/MPI_C_SAMPLE.c
https://www.openmp.org/wp-content/uploads/OpenMP3.0-SummarySpec.pdf
https://github.com/rpwilliams/CIS520_Proj4/blob/master/src/3way-mpi/lcs-mpi.c
https://support.beocat.ksu.edu/BeocatDocs/index.php
https://linux.die.net/man/
https://stackoverflow.com/questions/51414697/order-of-threads-executing-in-a-multiple-thread-program
https://stackoverflow.com/questions/42887315/avoiding-race-condition-using-int-to-void-casting
https://stackoverflow.com/questions/1845482/what-is-uintptr-t-data-typehttps://stackoverflow.com/questions/12490347/how-to-force-openmp-to-run-iterations-in-specific-order
https://princetonuniversity.github.io/PUbootcamp/sessions/parallel-programming/Intro_PP_bootcamp_2018.pdf
https://github.com/wesleykendall/mpitutorial/blob/gh-pages/tutorials/mpi-send-and-receive/code/ring.c
https://mpitutorial.com/tutorials/
https://stackoverflow.com/questions/8841069/critical-section-in-mpi