**CIS 520 – Spring 2020**
**Project #4 – One Program, Three ways**

## Group

Dominic Tassio <dmtassio@ksu.edu>
Sreenikhil Keshamoni <sreenikhil006@ksu.edu>
Blake Wewer <bewewer@ksu.edu>

tasks #of cores in the second line

## Introduction

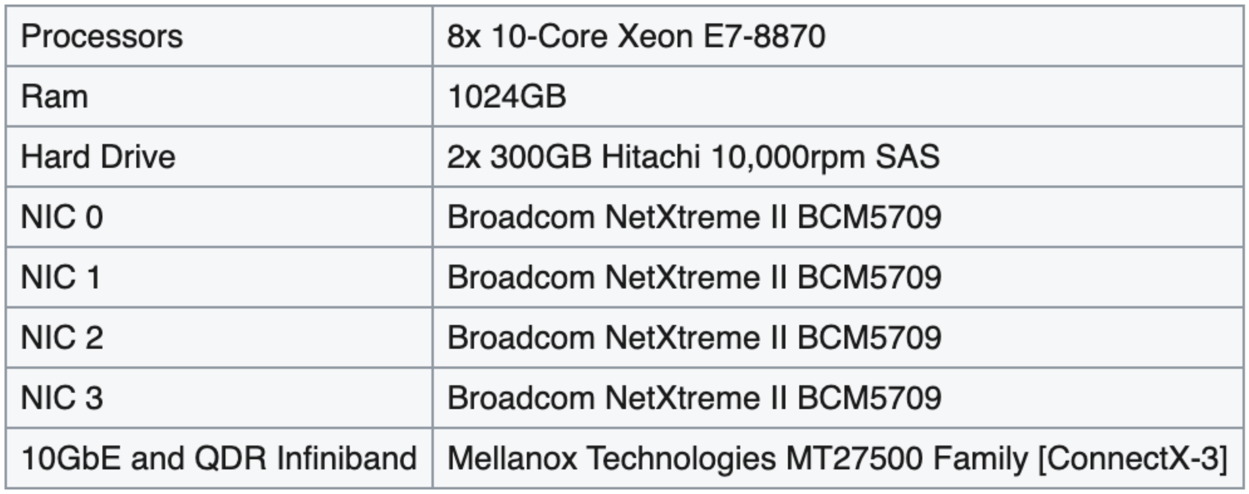
We were given a task to develop a parallelized program using the three standard tools Pthreads, OpenMP, and MPI. First, we had to create a sample program which computes the difference in sum of ASCII characters numeric values for each line for each pair of strings.

We use git line to read to each line of the file into an array of char pointers. Then we get the difference between in each like by first calculating the line and storing that in an integer array. Then iterated through the block of line where the difference is calculated.

Finally, we parallelized this program across the Beocat High Performance Computing Cluster here at Kansas State University using the tools Pthreads, OpenMP, and MPI.

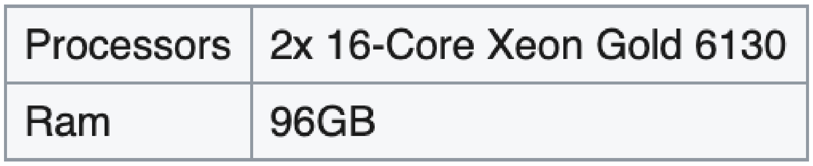
## Configuration

All the cores from 1-8 are ran on the MAGE node which has the below hardware configuration.



| | |
|---|---|
| Processors | 8x 10-Core Xeon E7-8870 |
| Ram | 1024GB |
| Hard Drive | 2x 300GB Hitachi 10,000rpm SAS |
| NIC 0 | Broadcom NetXtreme II BCM5709 |
| NIC 1 | Broadcom NetXtreme II BCM5709 |
| NIC 2 | Broadcom NetXtreme II BCM5709 |
| NIC 3 | Broadcom NetXtreme II BCM5709 |
| 10GbE and QDR Infiniband | Mellanox Technologies MT27500 Family [ConnectX-3] |

The 16 core jobs were run on WIZARD node which has the below hardware configuration.



| | |
|---|---|
| Processors | 2x 16-Core Xeon Gold 6130 |
| Ram | 96GB |

**Implementation**

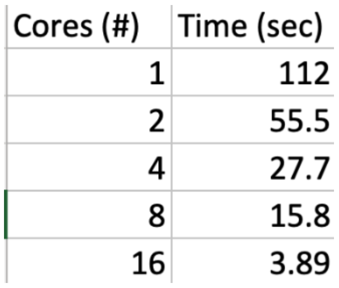
We used one million lines in all three of our approaches. We ran our tests on Beocat using 1,2,4,8 and 16 cores. For the openMP and pThreads each CPU was given 16 divided by the number of cores of memory. All the tests were run multiple times to check for consistency.

We kept track of time using getTimeofDay. We get the time before and after we read the file. We also get before and after time for the difference and we subtract the before and after to get the duration of time.

**Performance Analysis**

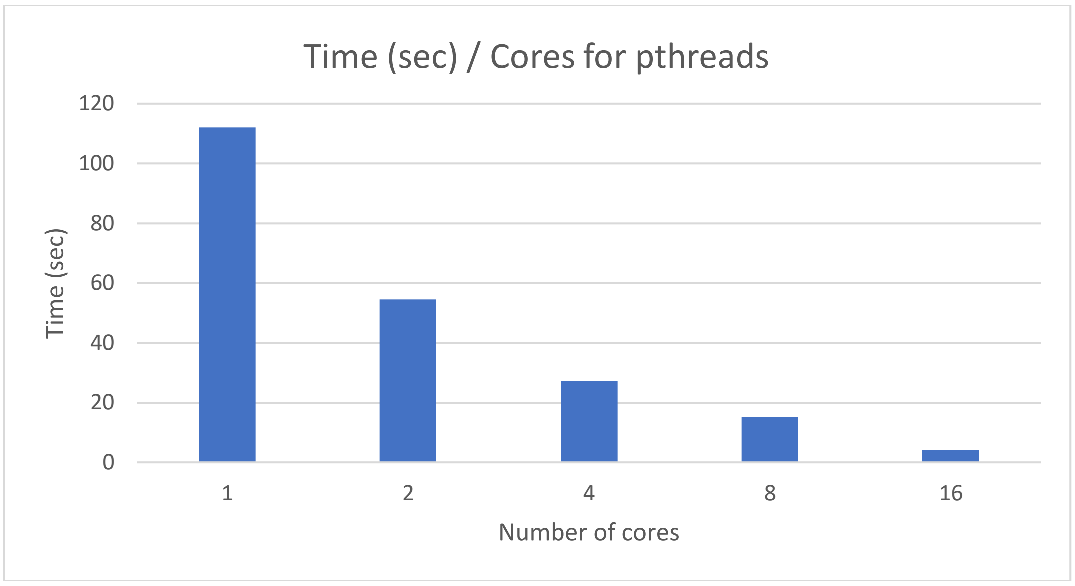
**OpenMP**:

As we expected, we can see as we add number of cores the time it takes to calculate the difference of ASCII characters in two given lines. The number of cores is inversely proportional to the time taken to calculate because the time is halved when we double the number of cores. We didn't have any race conditions while running openMP and pThreads. There is no communication between process. They independently read the data from the array and writes the difference to the differences array independently. By doing this, we compromised on computing some lines twice. We wanted to avoid threads talking to each other hence simplifying the computational process. This the same for pThreads.



| Cores (#) | Time (sec) |
|---|---|
| 1 | 112 |
| 2 | 55.5 |
| 4 | 27.7 |
| 8 | 15.8 |
| 16 | 3.89 |

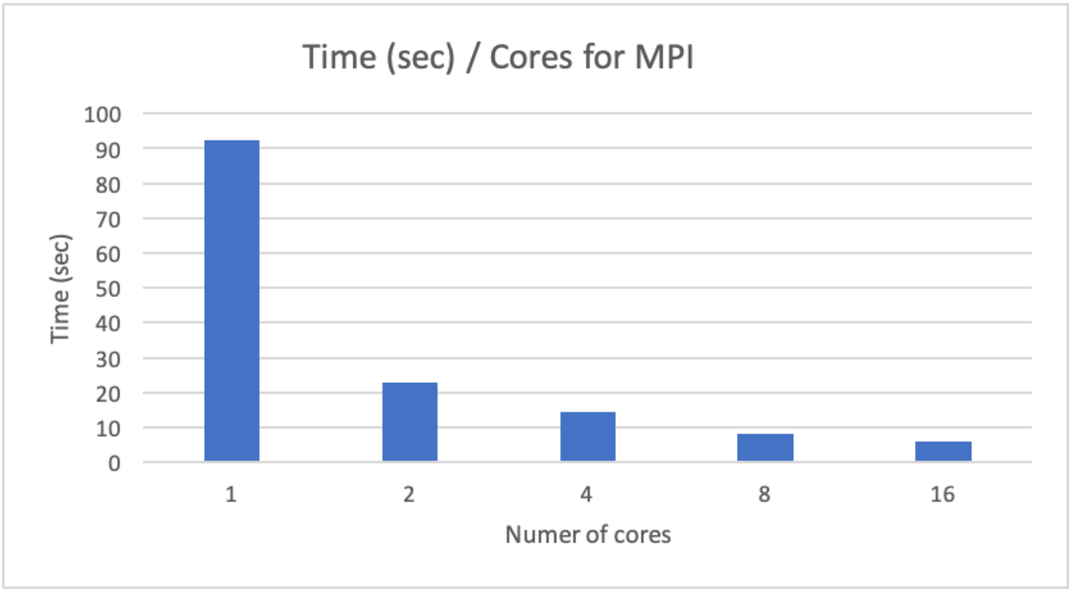
**pThreads:** explanation is the same as openMP.



| Cores (#) | Time (sec) |
|---:|---:|
| 1 | 112 |
| 2 | 54.6 |
| 4 | 27.4 |
| 8 | 15.3 |
| 16 | 4.1 |

**MPI:**

As we look at the graph below, we can see that the times takes to compute the difference is inversely proportional to the number of cores which means the times gained as we increased the cores is not the same every time we increased the number of cores. We haven't encountered any know race conditions. We used the MPI message passing functions like MPI_Bcast and MPI_COMM_WORLD which helps with synchronization. There is not a lot of communication while computing for the difference values. We didn't optimize the communication between process.



| Cores (#) | Time (sec) |
|---|---|
| 1 | 92.2 |
| 2 | 23.01 |
| 4 | 14.3 |
| 8 | 8.2 |
| 16 | 6.1 |

## Conclusion

We implemented our base code to compute the difference in sum of ASCII character numeric values for each line for each pair of strings. We used three different parallelization tools which are openMP, pThreads and MPI.

As we look across all three implementations, pThreads and openMP have similar running times. While MPI had the fastest running times in comparison with both pThreads and openMP.

## **Appendix A -** OpenMP source code

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

           #define NUM_THREADS 16
           #define NUM_LINES 1000000
```

```c
char *lines[NUM_LINES];
int line_diffs[NUM_LINES - 1];

void read_file();
void line_diff(size_t my_id);
int line_sum(size_t line);
void free_lines();
void print_diffs();

int main(void)
{
    struct timeval t1, t2, t3, t4;
    double elapsed_time;
    int my_version = 2;

    // Start time of reading file
    gettimeofday(&t1, NULL);

    read_file();

    // End time of reading file
    gettimeofday(&t2, NULL);

    omp_set_num_threads(NUM_THREADS);

    // Start time of parallel code
    gettimeofday(&t3, NULL);

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

    // End time of parallel code
    gettimeofday(&t4, NULL);

    // Calculate and print time to read file
    elapsed_time = (t2.tv_sec - t1.tv_sec) * 1000.0;
    elapsed_time += (t2.tv_usec - t1.tv_usec) / 1000.0;
    printf("Time to read file: %f\n", elapsed_time);

    // Calculate and print time to calculate line differences
    elapsed_time = (t4.tv_sec - t3.tv_sec) * 1000.0;
    elapsed_time += (t4.tv_usec - t3.tv_usec) / 1000.0;
    printf("Time to calculate difference: %f\n", elapsed_time);

    printf("DATA, %d, %s, %f\n", my_version, getenv("SLURM_CPUS_ON_NODE"),
elapsed_time);

    free_lines();
}

void read_file()
{
    FILE *fp = fopen("/homes/dan/625/wiki_dump.txt", "r");

    if (fp == NULL)
    {
        perror("[openmp/ERROR] [read_file/fopen]");
        exit(EXIT_FAILURE);
    }

    size_t i;
    for (i = 0; i < NUM_LINES; i++)
    {
        char *line = NULL;
        size_t len;

        if (getline(&line, &len, fp) == -1)
        {
            perror("[openmp/ERROR] [read_file/getline]");
            free(line);
            exit(EXIT_FAILURE);
        }

        lines[i] = line;
    }

    fclose(fp);
}

void line_diff(size_t id)
{
    size_t start, end, i;
    int previous_sum, sum, diff;
    #pragma omp private(id, start, end, i, previous_sum, sum, diff)
    {
        start = id * (NUM_LINES / NUM_THREADS);
        end = start + (NUM_LINES / NUM_THREADS);
        previous_sum = line_sum(start);

        printf("id = %zu | start = %zu | end = %zu\n", id, start, end);

        for (i = start + 1; i < end; i++)
        {
            sum = line_sum(i);
            diff = previous_sum - sum;

            line_diffs[i - 1] = diff;

            previous_sum = sum;
        }
    }
}

int line_sum(size_t line_num)
{
    char *line = lines[line_num];
    int result = 0;

    size_t i;
    for (i = 0; i < strlen(line); i++)
    {
        result += (int)line[i];
    }

    return result;
}

void free_lines()
{
    size_t i;
    for (i = 0; i < NUM_LINES; i++)
    {
        free(lines[i]);
    }
}
```

```
void print_diffs()
{
    size_t i;
    for (i = 0; i < NUM_LINES - 1; i++)
    {
        printf("%zu-%zu: %d\n", i, i + 1, line_diffs[i]);
    }
}
```

**Appendix B -** openMP batch file

#!/bin/bash -l //usr/bin/time -o /homes/dmtassio/proj4/CIS520P4/src/3way-openmp/time/time-$RANDOM.txt /homes/dmtassio/proj4/CIS520P4/src/3way-openmp/openmp-prod

**Appendix C -** Mass batch file

```
#!/bin/bash

for i in 1 2 4 8 16; do
    sbatch --time=04:00:00 --ntasks-per-node=$i --nodes=1 --mem-per-
cpu=$((16/$i))G --partition=killable.q openmp-batch.sh
done
```

**Appendix D -** pThreads source code

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

#define NUM_THREADS 16
#define NUM_LINES 1000000

char *lines[NUM_LINES];
int line_diffs[NUM_LINES - 1];

void read_file();
void line_diff(size_t my_id);
int line_sum(size_t line);
void free_lines();
void print_diffs();
```

```
int main(void)
{
    struct timeval t1, t2, t3, t4;
    double elapsed_time;
    int my_version = 3;

    int i, rc;
    pthread_t threads[NUM_THREADS];
    pthread_attr_t attr;
    void *status;

    pthread_attr_init(&attr);
    pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_JOINABLE);

    // Start time of reading file
    gettimeofday(&t1, NULL);

    read_file();

    // End time of reading file
    gettimeofday(&t2, NULL);

    // Start time of parallel code
    gettimeofday(&t3, NULL);

    for (i = 0; i < NUM_THREADS; i++)
    {
        rc = pthread_create(&threads[i], &attr, line_diff, (void *)i);

        if (rc)
        {
            printf("ERROR; return code from pthread_create() is %d\n", rc);
            exit(EXIT_FAILURE);
        }
    }

    pthread_attr_destroy(&attr);

    for (i = 0; i < NUM_THREADS; i++)
    {
        rc = pthread_join(threads[i], &status);

        if (rc)
```

```c
        {
            printf("ERROR; return code from pthread_join() is %d\n", rc);
            exit(EXIT_FAILURE);
        }
    }

    // End time of parallel code
    gettimeofday(&t4, NULL);

    // Calculate and print time to read file
    elapsed_time = (t2.tv_sec - t1.tv_sec) * 1000.0;
    elapsed_time += (t2.tv_usec - t1.tv_usec) / 1000.0;
    printf("Time to read file: %f\n", elapsed_time);

    // Calculate and print time to calculate line differences
    elapsed_time = (t4.tv_sec - t3.tv_sec) * 1000.0;
    elapsed_time += (t4.tv_usec - t3.tv_usec) / 1000.0;
    printf("Time to calculate difference: %f\n", elapsed_time);

    printf("DATA, %d, %s, %f\n", my_version, getenv("SLURM_CPUS_ON_NODE"),
elapsed_time);

    free_lines();
    pthread_exit(NULL);
}

void read_file()
{
    FILE *fp = fopen("/homes/dan/625/wiki_dump.txt", "r");

    if (fp == NULL)
    {
        perror("Failed: ");
        exit(EXIT_FAILURE);
    }

    size_t i;
    for (i = 0; i < NUM_LINES; i++)
    {
        char *line = NULL;
        size_t len;

        if (getline(&line, &len, fp) == -1)
        {
            perror("Failed: ");
            free(line);
            exit(EXIT_FAILURE);
        }

        lines[i] = line;
    }

    fclose(fp);
}

void line_diff(size_t id)
{
    size_t start, end, i;
    int previous_sum, sum, diff;

    start = id * (NUM_LINES / NUM_THREADS);
    end = start + (NUM_LINES / NUM_THREADS);
    previous_sum = line_sum(start);

    printf("id = %zu | start = %zu | end = %zu\n", id, start, end);

    for (i = start + 1; i < end; i++)
    {
        sum = line_sum(i);
        diff = previous_sum - sum;

        line_diffs[i - 1] = diff;

        previous_sum = sum;
    }

    pthread_exit(NULL);
}

int line_sum(size_t line_num)
{
    char *line = lines[line_num];
    int result = 0;

    size_t i;
    for (i = 0; i < strlen(line); i++)
    {
        result += (int)line[i];
    }

    return result;
}

void free_lines()
{
    size_t i;
    for (i = 0; i < NUM_LINES; i++)
    {
        free(lines[i]);
    }
}

void print_diffs()
{
    size_t i;
    for (i = 0; i < NUM_LINES - 1; i++)
    {
        printf("%zu-%zu: %d\n", i, i + 1, line_diffs[i]);
    }
}
```

**<u>Apendix E</u>** – pthreads batch file

```
#!/bin/bash
-l

//usr/bin/time -o /homes/dmtassio/proj4/CIS520P4/src/3way-pthread/time/time-
$RANDOM.txt /homes/dmtassio/proj4/CIS520P4/src/3way-pthread/pthread-prod
```

**<u>Appendix F</u>** – pthreads mass sbatch file

```
#!/bin/bash

for i in 1 2 4 8 16; do
    sbatch --time=04:00:00 --ntasks-per-node=$i --nodes=1 --mem-per-
cpu=$((16/$i))G --partition=killable.q pthread-batch.sh
Done
```

**<u>Appendix G</u>** – MPI source code

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

            #define NUM_ENTRIES 1000000
            #define LINE_LENGTH 2003

            int NUM_THREADS;
            char entries[NUM_ENTRIES][LINE_LENGTH];

            int results[NUM_ENTRIES];
            int local_results[NUM_ENTRIES];

            int calc_difference(char* line1, char* line2, int line1_length, int
            line2_length);
            void read_file();


            void main(int argc, char* argv[])
            {
                struct timeval t1, t2, t3, t4;
                double elapsedTime;

                int numSlots, myVersion = 4;

                int i, rc, numTasks, rank, count, dest, source, tag = 1;
                MPI_Status Status;

                rc = MPI_Init(&argc, &argv);

                if(rc != MPI_SUCCESS)
                {
                    printf("\n\nError starting MPI.\n\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);
```

```
    gettimeofday(&t1, NULL);
       if(rank == 0 )
    {
              read_file("/homes/dan/625/wiki_dump.txt");
        // read_file("src/base/test.txt");
       }
       gettimeofday(&t2, NULL);

    MPI_Bcast(entries, NUM_ENTRIES * LINE_LENGTH, MPI_CHAR, 0, MPI_COMM_WORLD);

    gettimeofday(&t3, NULL);

    int line1_counter = -1;
    int line2_counter = 0;
    char line1_array[1000];
    char line2_array[1000];
    char* line1 = "";
    char* line2 = entries[line2_counter];
    while(line2 != NULL)
    {
        //printf("Lines %d-%d: %d\n", line1_counter, line2_counter,
calc_difference(line1, line2, strlen(line1), strlen(line2)));
        local_results[line2_counter] = calc_difference(line1, line2,
strlen(line1), strlen(line2));
        line1_counter++;
        line2_counter++;
        line1 = entries[line1_counter];
        line2 = entries[line2_counter];
    }

    gettimeofday(&t4, NULL);
    MPI_Reduce(local_results, results, NUM_ENTRIES * LINE_LENGTH, MPI_CHAR,
MPI_SUM, 0, MPI_COMM_WORLD);

    elapsedTime = (t2.tv_sec - t1.tv_sec) * 1000.0; //sec to ms
       elapsedTime += (t2.tv_usec - t1.tv_usec) / 1000.0; // us to ms
       printf("Time to read file: %f\n", elapsedTime);

       elapsedTime = (t4.tv_sec - t3.tv_sec) * 1000.0; //sec to ms
       elapsedTime += (t4.tv_usec - t3.tv_usec) / 1000.0; // us to ms
       printf("Time to calculate difference: %f\n", elapsedTime);
    printf("DATA, %d, %s, %f\n", myVersion, getenv("SLURM_CPUS_ON_NODE"),
elapsedTime);


    // if(rank == 0)
    // {
    //     dest = 1;
    //     source = 1;
    //     rc = MPI_Send(&outmsg, 1, MPI_CHAR, dest, tag, MPI_COMM_WORLD);
    //     rc = MPI_Recv(&inmsg, 1, MPI_CHAR, source, tag, MPI_COMM_WORLD,
&Stat);
    // }
    // else if (rank == 1)
    // {
    //     dest = 0;
    //     source = 0;
    //     rc = MPI_Send(&outmsg, 1, MPI_CHAR, dest, tag, MPI_COMM_WORLD);
    //     rc = MPI_Recv(&inmsg, 1, MPI_CHAR, source, tag, MPI_COMM_WORLD,
&Stat);
    // }

    MPI_Finalize();
}

int calc_difference(char* line1, char* line2, int line1_length, int
line2_length)
{
    long sum1 = 0;
    long sum2 = 0;

    for(int i = 0; i < line1_length; i++)
        sum1 += line1[i];

    for(int i = 0; i < line2_length; i++)
        sum2 += line2[i];

    // printf("%ld - %ld\n", sum1, sum2);

    return (int)(sum1 - sum2);
}

void read_file(char* filename)
{
    FILE* fp = fopen(filename, "r");
    if(fp == NULL)
    {
        perror("Unable to open file!  File is - ");
        exit(1);
    }

    char line[LINE_LENGTH];

    int i = 0;
    while(fgets(line, LINE_LENGTH, fp) != NULL && i < NUM_ENTRIES)
    {
        strcpy(entries[i], line);
        i++;
    }

    fclose(fp);
}
```

## **Appendix H** – MPI batch file

```
#!/bin/bash
-l

module load OpenMPI

mpirun //usr/bin/time -o /homes/bewewer/proj4/CIS520P4/src/3way-
mpi/time/time-$RANDOM.txt /homes/bewewer/proj4/CIS520P4/src/3way-mpi/mpi-prod
```

## **Appendix I** – MPI mass sbatch file

```
#!/bin/bash

for i in 1 2 4 8 16
do
        sbatch --time=04:00:00 --ntasks-per-node=$i --nodes=1 --mem-per-
cpu=$((16/$i))G --partition=killable.q mpi-batch.sh
done
```