Project #4 Analysis

Alexander Marney, Weston Harder, Aswini Patro

Introduction

For project #4, we were made to implement parallelization in 3 different ways on a program that would find the longest common substring between 2 strings sequentially within a text file. The 3 separate ways of parallelization we were made to user were OpenMP, Pthreads, and MPI, while the text file we were meant to analyze contained around 1 million Wikipedia entries, where each line was one entry.

Initially, our first task was to develop an algorithm to find the longest common substring between two strings. For our implementation we created a 2D array where each value was initialized to 1, with the two axis corresponding to one of the strings apiece. As common substrings were found, a diagonal string of integers was formed with increasing numbers. After the strings had been analyzed, we would find the longest string of increasing numbers and save the corresponding values to a string and return it to the calling function. After the longest common substring was found, it would be saved to an array of strings to be output at the end of the program. From here we were made to implement parallelization using the Beocat High Performance Computing Cluster at KSU, by scheduling jobs with varying data sizes and implementations.

Hardware

All iterations of our code were ran on Beocat, specifically the elf nodes. The hardware specifications for node groups are shown below:

Processors	2x 10-Core Xeon E5-2690 v2
Ram	96GB
Hard Drive	1x 250GB 7,200 RPM SATA
NICs	4x Intel I350
10GbE and QDR Infiniband	Mellanox Technologies MT27500 Family [ConnectX-3]

Processors	2x 10-Core Xeon E5-2690 v2
Ram	384GB
Hard Drive	1x 250GB 7,200 RPM SATA
NICs	4x Intel I350
10GbE and QDR Infiniband	Mellanox Technologies MT27500 Family [ConnectX-3]

Processors	2x 10-Core Xeon E5-2690v2
Ram	64GB
Hard Drive	1x 250GB 7,200 RPM SATA
NICs	4x Intel I350
10GbE and QDR Infiniband	Mellanox Technologies MT27500 Family [ConnectX-3]

Processors	2x 8-Core Xeon E5-2690
Ram	64GB
Hard Drive	1x 250GB 7,200 RPM SATA
NICs	4x Intel I350
10GbE and QDR Infiniband	Mellanox Technologies MT27500 Family [ConnectX-3]

Testing

For OpenMP and Pthreads, there were tests ran on 10k, 100k, 500k, and 1m line versions of the code. For each of these cases, the tests were run with 1, 2, 4, 8, and 16 cores. Due to time constraints, all versions were ran 1 time for our final results.

MPI was tested with only the 1m line version, with 1, 2, 4, 8, and 16 cores.

Time was tracked for every test, for the time to read, time to find LCS, and time to print results, after the times were gathered, they were then put into an excel spreadsheet to easier compare the data.

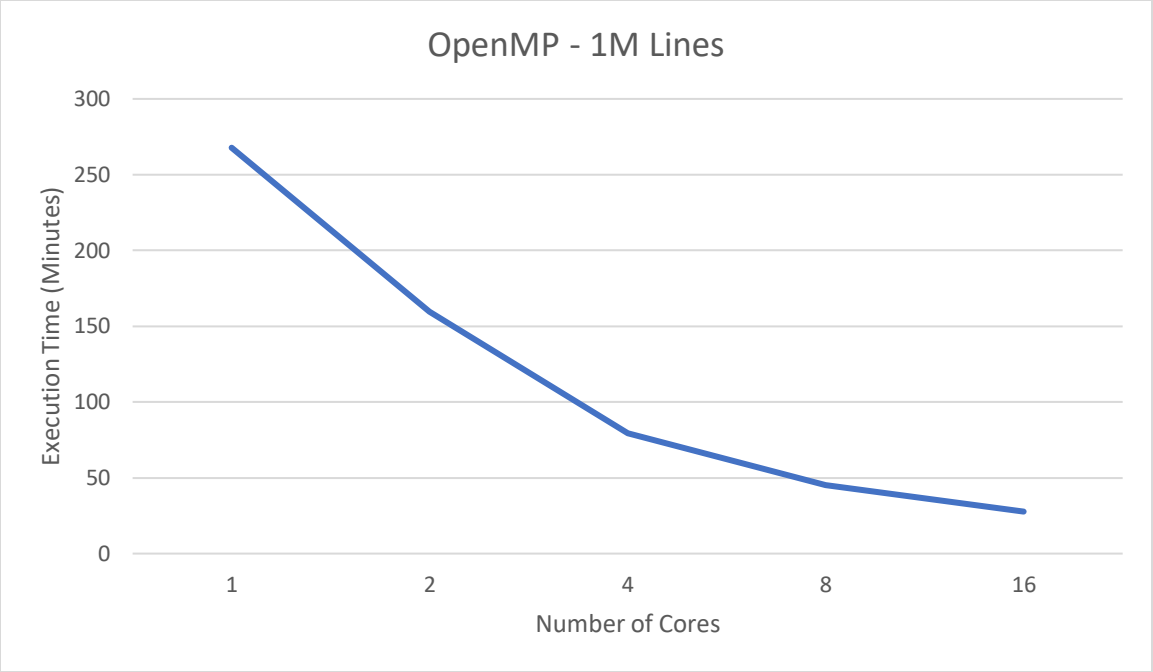
Results

OpenMP

Using OpenMP to parallelize our code resulted in the following execution times, shown in minutes:

OpenMP	1c	2c	4c	8c	16c
10k	2.51	1.343	0.7625	0.469	0.33
100k	26.5	13.9	8.5	4.53	3.15
500k	127.6	72.7	39.1	22.14	15.5
1m	267.8	159.63	79.2	45.32	27.7

The time to find the LCS of each line nearly linearly decreased by 50% each time the core count was doubled, to a point. Once the core counts reached 8 or above, the times stopped halving due to communication within the program and read/result times. However, the program still ran faster with 16 cores rather than 8. To better show the distribution between times, this is a graph of the 1 Million line tests with OpenMP:

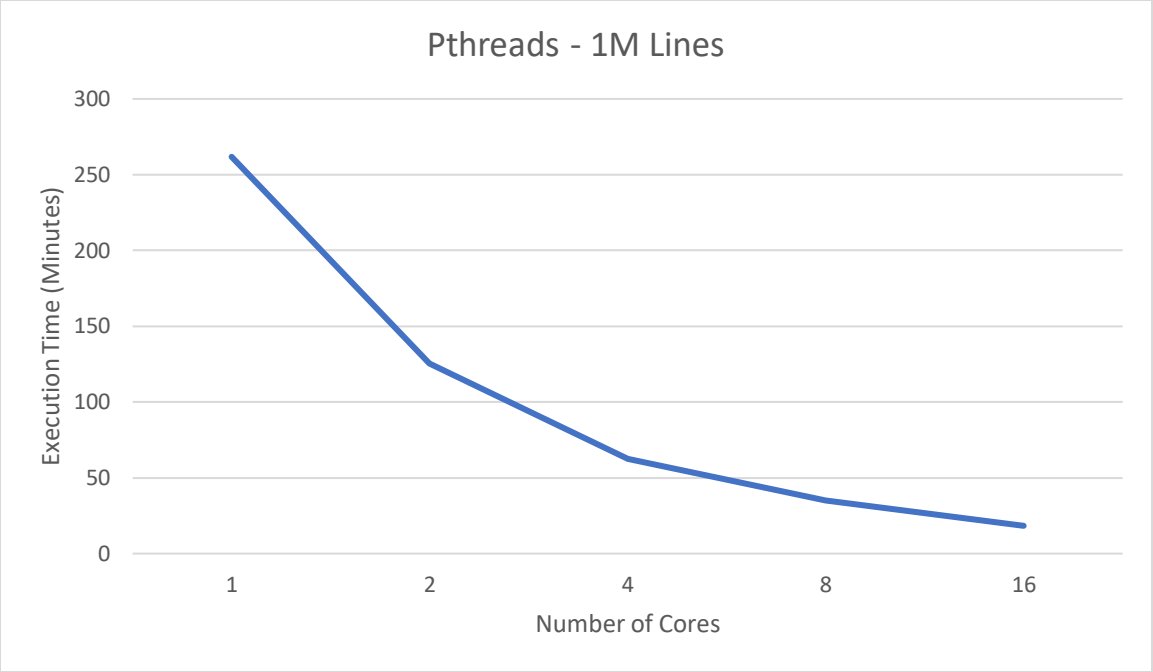


Pthreads

Using Pthreads to parallelize our code resulted in the following execution times, shown in minutes:

Pthread	1c	2c	4c	8c	16c
10k	2.8	1.23	0.642	0.3766	0.23
100k	27.4	14.2	6.4	3.8	2.3
500k	127.8	66.9	37.9	16.9	10.14
1m	261.8	125.2	62.3	34.85	18.33

Much like OpenMP, Pthreads execution times were reduced by 50% intially, before levelling off due to system communication and I/O times. To better show the distribution between times, this is a graph of the 1 Million line tests with Pthreads:

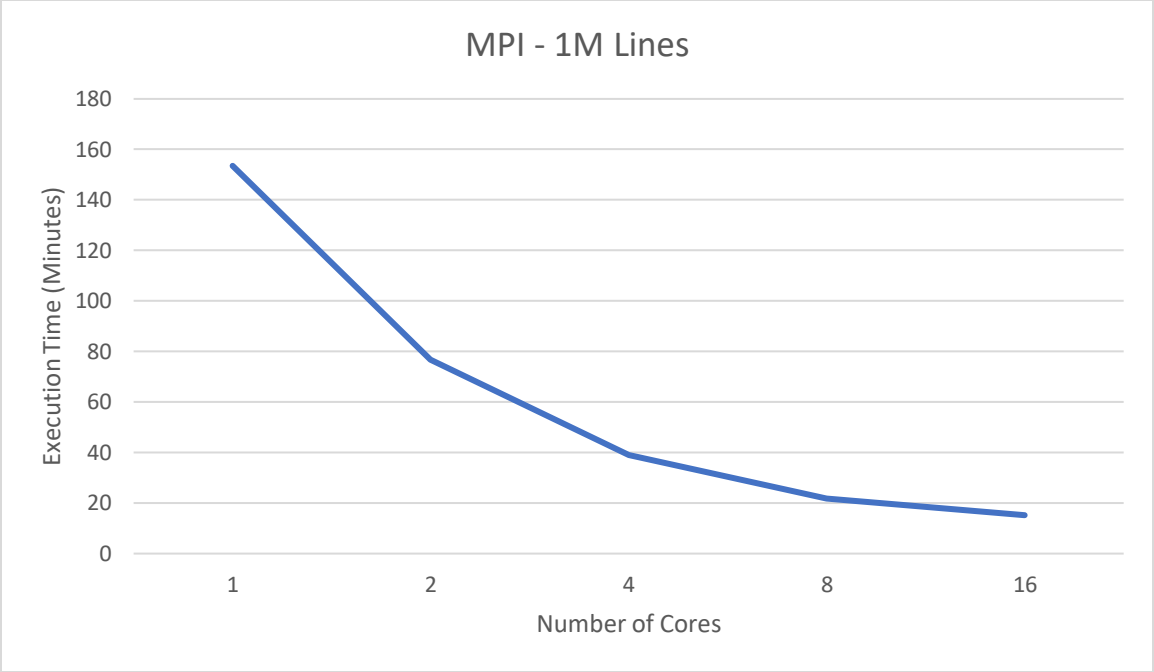


MPI

Using MPI to parallelize our program was done with some synchronization errors, while the LCS were calculated correctly, the final output file contained a number of empty strings comparable to the number of cores used. This resulted in the following execution times, shown in minutes:

MPI	1c	2c	4c	8c	16c
1m	153.5	76.75	39	21.7	15.2

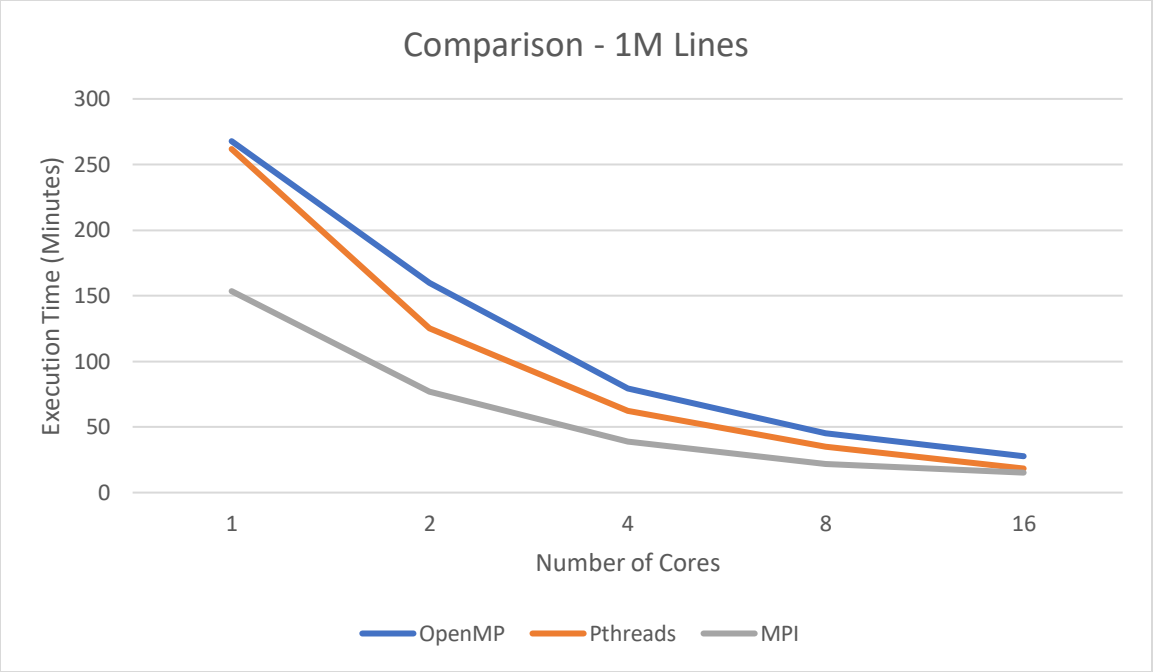
Although MPI ran faster than both OpenMP and Pthreads on all counts, it followed the same trend of execution time reduction with increasing core counts as they did. To better show the distribution between times, this is a graph of the 1 Million line tests with MPI:



Performance

Of the three methods, MPI was far faster than both OpenMP and Pthreads. The largest percentage difference was found at the lower number of cores, but in all categories it was by faster by at least 10%.

Pthreads and OpenMP both had similar performance initially, but as the core count increased Pthreads had a better curve, as the following graph illustrates.



Conclusion

We wrote a program to find the longest common substring between two given strings, and used that program to determine the LCS between up to 1 million separate Wikipedia entries contained in a text file. To do this we used 3 different libraries: OpenMP, Pthreads, and MPI.

OpenMP and Pthreads were both relatively easy to implement, though overall were the slowest of the three.

MPI was far more confusing, but it's program ran far faster than either OpenMP or Pthreads, at all core counts.

Overall, if there is an ability to use MPI, it should be used, but in scenarios where overall program run time already won't be extremely high it would be reasonable to forego the complexity of MPI's implementation for OpenMP or Pthreads.

Appendix A1: OpenMP Implementation.

```
/*
Command to compile:
gcc p4_omp.c -fopenmp -o p4_omp
*/

#include <omp.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/time.h>

#define NUM_THREADS 16 /* UPDATE IN .sh FILE TOO!!! Number of
threads/cores/CPUs */
#define NUM_LINES 1000000 /* Number of lines to read in */
#define LINE_LENGTH 2003 /* Max number of characters to store for each line
*/
#define FILENAME "/homes/dan/625/wiki_dump.txt" /* File to read in line by
line */
#define NUM_LINES_PER_THREAD (NUM_LINES / NUM_THREADS)

typedef unsigned long int uint32;
typedef unsigned int uint16;

uint32 actual_num_lines; /* Number of lines successfully read from file */
char data[NUM_LINES][LINE_LENGTH]; /* All data read in from file */
char lcs_data[NUM_LINES - 1][LINE_LENGTH]; /* longest common substrings */

void open_file(void);
uint32 lcs_dynamic(char*, const char*, uint32, const char*, uint32);
void thandle(int);

void open_file()
{
      int count;
      FILE *file;
      printf("Opening File\n");
      file = fopen(FILENAME, "r");
      if (file == NULL)
      {
            perror(FILENAME);
            return;
      }

      for(count = 0; count < NUM_LINES; count++)
      {
            if(fgets(data[count], LINE_LENGTH, file) == NULL)
            {
                  break;
            }
      }

      actual_num_lines = count;

      fclose(file);
```

```
}


void thandle(int tid) {
	char temp[LINE_LENGTH]; /* temporary string storage     */
	char ret[LINE_LENGTH];  /* formatted string with lcs    */
	char * s1;              /* pointer to string 1          */
	char * s2;              /* pointer to string 2          */
	uint32 len_lcs;         /* length of lcs                */
	uint32 line_number;     /* index into data for s1       */
	uint32 start;           /* index into data for first s1 */
	uint32 end;             /* index into data for last s2  */

	start = tid * NUM_LINES_PER_THREAD;
	end = start + NUM_LINES_PER_THREAD;

	/* Avoid reading past the end of data */
	if(end > (actual_num_lines - 1))
	{
		end = actual_num_lines - 1;
	}

	#pragma omp private (temp,ret,s1,s2,line_number,start,end)
	{
		for (line_number = start; line_number < end; line_number++)
		{
			s1 = data[line_number];
			s2 = data[line_number + 1];

			len_lcs = lcs_dynamic(ret, s1, strlen(s1), s2, strlen(s2));
			if (len_lcs > 0)
			{
				strcpy(lcs_data[line_number], ret);
			}
			else
			{
				strcpy(lcs_data[line_number], "No common
substring.");
			}

		}
	}

	return;
}

/*
Find the longest common substring between two strings using dynamic
programming
Return the length of the LCS
Write LCS to ret
*/
uint32 lcs_dynamic(char * ret, const char * a, uint32 a_size, const char * b,
uint32 b_size)
{
	uint32 i; /* loop index */
	uint32 a_idx; /* a index */
```

```
	uint32 b_idx; /* b index */

	uint32 a_idx_max = 0; /* a index of end of longest common substring */
	uint32 b_idx_max = 0; /* b index of end of longest common substring */
	uint32 max_length = 0; /* length of longest common substring */

	uint16 ** substring_lengths; /* uint16 limits max substring length to
about 65535 characters, which is more than we need */

	/* Allocate space for array */
	substring_lengths = (uint16 **)malloc(sizeof(uint16 *) * a_size);
	for (a_idx = 0; a_idx < a_size; a_idx++)
	{
		substring_lengths[a_idx] = (uint16 *)malloc(sizeof(uint16) *
b_size);
	}

	/* Compare each character in a with each character in b */
	for (a_idx = 0; a_idx < a_size; a_idx++)
	{
		for (b_idx = 0; b_idx < b_size; b_idx++)
		{
			/* If the characters match */
			if (a[a_idx] == b[b_idx])
			{
				/* If one of the characters is the starting character
for that string */
				if ((a_idx == 0)
					|| (b_idx == 0))
				{
					substring_lengths[a_idx][b_idx] = 1;
				}
				else
				{
substring_lengths[a_idx][b_idx] = substring_lengths[a_idx - 1][b_idx - 1] +
1;
				}

				/* Only override longest common substring if a longer
common substring is found */
				if (substring_lengths[a_idx][b_idx] > max_length)
				{
					max_length = substring_lengths[a_idx][b_idx];
					a_idx_max = a_idx;
					b_idx_max = b_idx;
				}
			}
			/* If the characters don't match */
			else
			{
				substring_lengths[a_idx][b_idx] = 0;
			}
		}
	}

	/* Copy longest common substring to ret */
	for (i = 0; i < max_length; i++)
```

```
	{
		ret[i] = a[a_idx_max - max_length + i + 1];
	}
	ret[max_length] = '\0';

	/* Free dynamically allocated memory for array */
	for (a_idx = 0; a_idx < a_size; a_idx++)
	{
		free(substring_lengths[a_idx]);
	}
	free(substring_lengths);

	return max_length;
}



int main(void)
{
	int i; /* Loop counter */

	struct timeval t1, t2, t3, t4;

	gettimeofday(&t1, NULL);
	open_file();
	gettimeofday(&t2, NULL);

	omp_set_num_threads(NUM_THREADS);

	#pragma omp parallel
	{
		thandle(omp_get_thread_num());
	}
	gettimeofday(&t3,NULL);

	for(i = 0; i < actual_num_lines - 1; i++)
	{
		printf("%3u - %3u: %s\n", i, i + 1, lcs_data[i]);
	}

	gettimeofday(&t4, NULL);
	double time = (t2.tv_sec - t1.tv_sec) * 1000.0;
	time += (t2.tv_usec - t1.tv_usec) / 1000.0;
	printf("Time to read data: %f\n", time);

	time = (t3.tv_sec - t2.tv_sec) * 1000.0;
	time += (t3.tv_usec - t2.tv_usec) / 1000.0;
	printf("Time to determine LCS: %f\n", time);

	time = (t4.tv_sec - t3.tv_sec) * 1000.0;
	time += (t4.tv_usec - t3.tv_usec) / 1000.0;
	printf("Time to print results: %f\n", time);
	printf("Program Completed. \n");

	return 0;
}
```

Appendix A2: First 100 lines of result.

```
  0 -    1: </title><text>\'\'\'aa
  1 -    2: </text> </page>

  2 -    3: }}</text> </page>

  3 -    4: <page> <title>a
  4 -    5: \n|foundation = 19
  5 -    6: <page> <title>abc_
  6 -    7: <page> <title>abc_
  7 -    8:  the [[australian broadcasting corporation]]
  8 -    9:  the [[australian broadcasting corporation]]
  9 -   10:  [[australian broadcasting corporation]]
 10 -   11: </title><text>{{infobox
 11 -   12: <page> <title>ab
 12 -   13: </title><text>\'\'\'ab
 13 -   14: </title><text>\'\'\'a
 14 -   15: <page> <title>ac
 15 -   16: <page> <title>acc
 16 -   17: \n\n{{disambig}}</text> </page>

 17 -   18: }}</text> </page>

 18 -   19: \n\n{{disambiguation}}</text> </page>

 19 -   20: </title><text>\'\'\'ac
 20 -   21: <page> <title>ac
 21 -   22: <page> <title>ac_
 22 -   23: <page> <title>ac_
 23 -   24: </text> </page>

 24 -   25: \'\'\' may refer to:\n
 25 -   26: \'\'\' may refer to:\n\n
 26 -   27: <page> <title>ad
 27 -   28: <page> <title>ad
 28 -   29: <page> <title>a
 29 -   30: \n\n{{disambig}}</text> </page>

 30 -   31: <page> <title>afc
 31 -   32: <page> <title>af
 32 -   33: </text> </page>

 33 -   34: <page> <title>a
 34 -   35: </title><text>{{infobox
 35 -   36: </title><text>{{
 36 -   37: <page> <title>a
 37 -   38: <page> <title>aid
 38 -   39: <page> <title>ai
 39 -   40: [[australian institute of
 40 -   41: <page> <title>a
 41 -   42: \n\n{{disambig}}</text> </page>

 42 -   43: ]]\n\n{{disambig}}</text> </page>
```

```
43 -  44: </text> </page>

44 -  45: n-stub}}</text> </page>

45 -  46: </text> </page>

46 -  47: </text> </page>

47 -  48: <page> <title>alar
48 -  49: <page> <title>al
49 -  50: <page> <title>alp
50 -  51: <page> <title>a
51 -  52: <page> <title>am
52 -  53: <page> <title>am
53 -  54: <page> <title>am
54 -  55: <page> <title>am
55 -  56: }}\n\n{{defaultsort:am
56 -  57: </title><text>{{unreferenced
57 -  58: class=\"wikitable\"
58 -  59: <page> <title>an
59 -  60: <page> <title>a
60 -  61: <page> <title>a
61 -  62: <page> <title>ap
62 -  63: <page> <title>ap
63 -  64: <page> <title>ap
64 -  65: <page> <title>ap
65 -  66: \n\n==external links==\n*
66 -  67: <page> <title>ap
67 -  68: </title><text>{{
68 -  69: {{cite web|url=http://www.
69 -  70: <page> <title>ar
70 -  71: <page> <title>a
71 -  72: ==\n{{reflist}}\n\n==
72 -  73: <page> <title>asa_
73 -  74: <page> <title>as
74 -  75: <page> <title>as
75 -  76: <page> <title>as
76 -  77: <page> <title>as
77 -  78: </text> </page>

78 -  79: <page> <title>as
79 -  80: american society for
80 -  81:  [[association fo
81 -  82: \'\'\' is a [[france|french]] [[association football]]
82 -  83: <page> <title>a
83 -  84: <page> <title>at
84 -  85: <page> <title>at
85 -  86: <page> <title>at
86 -  87: <page> <title>at
87 -  88: <page> <title>a
88 -  89: {{cite web|url=http://www.
89 -  90: <page> <title>a
90 -  91: <page> <title>a
91 -  92: ]]\n\n{{disambig}}</text> </page>

92 -  93: }}</text> </page>
```

```
 93 -  94: </title><text>{{
 94 -  95: {{unreferenced|date=
 95 -  96: <page> <title>a_b
 96 -  97: <page> <title>a_be
 97 -  98: <page> <title>a_be
 98 -  99:  2012}}\n{{infobox album
 99 - 100:  name        = a big
```

Appendix B1: Pthreads Implementation.

```
/*
Command to compile:
gcc p4_pth.c -o p4_pth -lpthread -mcmodel=medium
*/

#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/time.h>

#define NUM_THREADS 16
#define NUM_LINES 1000000 //reduced for testing purposes
#define LINE_LENGTH 2003 //reduced for testing purposes
#define FILENAME "/homes/dan/625/wiki_dump.txt" //file of interest
#define NUM_LINES_PER_THREAD (NUM_LINES / NUM_THREADS)

typedef unsigned long int uint32;
typedef unsigned int uint16;

uint16 actual_num_lines; /* Number of lines successfully read from file */
char data[NUM_LINES][LINE_LENGTH]; /* All data read in from file */
char lcs_data[NUM_LINES - 1][LINE_LENGTH]; /* longest common substrings */

void open_file(void);
uint32 lcs_dynamic(char*, const char*, uint32, const char*, uint32);
void thandle(int);

void open_file()
{
      int count;
      FILE *file;

      file = fopen(FILENAME, "r");
      if (file == NULL)
      {
            perror(FILENAME);
            return;
      }

      for(count = 0; count < NUM_LINES; count++)
      {
            if(fgets(data[count], LINE_LENGTH, file) == NULL)
            {
                  break;
```

```
            }
      }

      actual_num_lines = count;

      fclose(file);
}


void thandle(int tid) {
      char temp[LINE_LENGTH]; /* temporary string storage     */
      char ret[LINE_LENGTH];  /* formatted string with lcs    */
      char * s1;              /* pointer to string 1          */
      char * s2;              /* pointer to string 2          */
      uint32 len_lcs;         /* length of lcs                */
      uint32 line_number;     /* index into data for s1       */
      uint32 start;           /* index into data for first s1 */
      uint32 end;             /* index into data for last s2  */

      {
            start = tid * NUM_LINES_PER_THREAD;
            end = start + NUM_LINES_PER_THREAD;

            /* Avoid reading past the end of data */
            if(end > (actual_num_lines - 1))
            {
                  end = actual_num_lines - 1;
            }

            for (line_number = start; line_number < end; line_number++)
            {
                  s1 = data[line_number];
                  s2 = data[line_number + 1];
                  len_lcs = lcs_dynamic(ret, s1, strlen(s1), s2, strlen(s2));
                  if (len_lcs > 0)
                  {
                        strcpy(lcs_data[line_number], ret);
                  }
                  else
                  {
                        strcpy(lcs_data[line_number], "No common
substring.");
                  }

            }
      }

      return;
}


/*
Find the longest common substring between two strings using dynamic
programming
Return the length of the LCS
Write LCS to ret
*/
```

```
uint32 lcs_dynamic(char * ret, const char * a, uint32 a_size, const char * b,
uint32 b_size)
{
	uint32 i; /* loop index */
	uint32 a_idx; /* a index */
	uint32 b_idx; /* b index */

	uint32 a_idx_max = 0; /* a index of end of longest common substring */
	uint32 b_idx_max = 0; /* b index of end of longest common substring */
	uint32 max_length = 0; /* length of longest common substring */

	uint16 ** substring_lengths; /* uint16 limits max substring length to
about 65535 characters, which is more than we need */

	/* Allocate space for array */
	substring_lengths = (uint16 **)malloc(sizeof(uint16 *) * a_size);
	for (a_idx = 0; a_idx < a_size; a_idx++)
	{
		substring_lengths[a_idx] = (uint16 *)malloc(sizeof(uint16) *
b_size);
	}

	/* Compare each character in a with each character in b */
	for (a_idx = 0; a_idx < a_size; a_idx++)
	{
		for (b_idx = 0; b_idx < b_size; b_idx++)
		{
			/* If the characters match */
			if (a[a_idx] == b[b_idx])
			{
				/* If one of the characters is the starting character
for that string */
				if ((a_idx == 0)
					|| (b_idx == 0))
				{
					substring_lengths[a_idx][b_idx] = 1;
				}
				else
				{
					substring_lengths[a_idx][b_idx] =
substring_lengths[a_idx - 1][b_idx - 1] + 1;
				}

				/* Only override longest common substring if a longer
common substring is found */
				if (substring_lengths[a_idx][b_idx] > max_length)
				{
					max_length = substring_lengths[a_idx][b_idx];
					a_idx_max = a_idx;
					b_idx_max = b_idx;
				}
			}
			/* If the characters don't match */
			else
			{
				substring_lengths[a_idx][b_idx] = 0;
			}
```

```
            }
        }

        /* Copy longest common substring to ret */
        for (i = 0; i < max_length; i++)
        {
            ret[i] = a[a_idx_max - max_length + i + 1];
        }
        ret[max_length] = '\0';
        /* Free dynamically allocated memory for array */
        for (a_idx = 0; a_idx < a_size; a_idx++)
        {
            free(substring_lengths[a_idx]);
        }
        free(substring_lengths);

        return max_length;
}



int main(void)
{
        int i; /* Loop counter */
        void *status;
        int code;
        struct timeval t1, t2, t3, t4;
        pthread_t threads[NUM_THREADS];
        pthread_attr_t attr;

        pthread_attr_init(&attr);
        pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_JOINABLE);
        gettimeofday(&t1, NULL);
        open_file();
        gettimeofday(&t2, NULL);
          for (i = 0; i < NUM_THREADS; i++)
        {
            code = pthread_create(&threads[i], &attr, thandle,(void *) i);
            if(code)
            {
                printf("ERROR: error code from pthread_create(): %d\n",
code);
                exit(-1);
            }
        }
        pthread_attr_destroy(&attr);
        for (i = 0; i < NUM_THREADS; i++)
        {
            code = pthread_join(threads[i], &status);
            if (code)
            {
                printf("ERROR: error code from pthread_join(): %d\n",
code);
                exit(-1);
            }
        }
        gettimeofday(&t3, NULL);
```

```
	for(i = 0; i < actual_num_lines - 1; i++)
	{
		printf("%3u - %3u: %s\n", i, i + 1, lcs_data[i]);
	}
	gettimeofday(&t4, NULL);

	double time = (t2.tv_sec - t1.tv_sec) * 1000.0;
	time += (t2.tv_usec - t1.tv_usec) / 1000.0;
	printf("Time to read data: %f\n", time);

	time = (t3.tv_sec - t2.tv_sec) * 1000.0;
	time += (t3.tv_usec - t2.tv_usec) / 1000.0;
	printf("Time to determine LCS: %f\n", time);

	time = (t4.tv_sec - t3.tv_sec) * 1000.0;
	time+= (t4.tv_usec - t3.tv_usec) / 1000.0;
	printf("Time to print results: %f\n", time);

	printf("Program Completed. \n");
	return 0;
}
```

Appendix B2: First 100 lines of result.

```
  0 -   1: </title><text>\'\'\'aa
  1 -   2: </text> </page>

  2 -   3: }}</text> </page>

  3 -   4: <page> <title>a
  4 -   5: \n|foundation = 19
  5 -   6: <page> <title>abc_
  6 -   7: <page> <title>abc_
  7 -   8:  the [[australian broadcasting corporation]]
  8 -   9:  the [[australian broadcasting corporation]]
  9 -  10:  [[australian broadcasting corporation]]
 10 -  11: </title><text>{{infobox
 11 -  12: <page> <title>ab
 12 -  13: </title><text>\'\'\'ab
 13 -  14: </title><text>\'\'\'a
 14 -  15: <page> <title>ac
 15 -  16: <page> <title>acc
 16 -  17: \n\n{{disambig}}</text> </page>

 17 -  18: }}</text> </page>

 18 -  19: \n\n{{disambiguation}}</text> </page>

 19 -  20: </title><text>\'\'\'ac
 20 -  21: <page> <title>ac
 21 -  22: <page> <title>ac_
 22 -  23: <page> <title>ac_
 23 -  24: </text> </page>

 24 -  25: \'\'\' may refer to:\n
```

```
25 -  26: \'\'\' may refer to:\n\n
26 -  27: <page> <title>ad
27 -  28: <page> <title>ad
28 -  29: <page> <title>a
29 -  30: \n\n{{disambig}}</text> </page>

30 -  31: <page> <title>afc
31 -  32: <page> <title>af
32 -  33: </text> </page>

33 -  34: <page> <title>a
34 -  35: </title><text>{{infobox
35 -  36: </title><text>{{
36 -  37: <page> <title>a
37 -  38: <page> <title>aid
38 -  39: <page> <title>ai
39 -  40: [[australian institute of
40 -  41: <page> <title>a
41 -  42: \n\n{{disambig}}</text> </page>

42 -  43: ]]\n\n{{disambig}}</text> </page>

43 -  44: </text> </page>

44 -  45: n-stub}}</text> </page>

45 -  46: </text> </page>

46 -  47: </text> </page>

47 -  48: <page> <title>alar
48 -  49: <page> <title>al
49 -  50: <page> <title>alp
50 -  51: <page> <title>a
51 -  52: <page> <title>am
52 -  53: <page> <title>am
53 -  54: <page> <title>am
54 -  55: <page> <title>am
55 -  56: }}\n\n{{defaultsort:am
56 -  57: </title><text>{{unreferenced
57 -  58: class=\"wikitable\"
58 -  59: <page> <title>an
59 -  60: <page> <title>a
60 -  61: <page> <title>a
61 -  62: <page> <title>ap
62 -  63: <page> <title>ap
63 -  64: <page> <title>ap
64 -  65: <page> <title>ap
65 -  66: \n\n==external links==\n*
66 -  67: <page> <title>ap
67 -  68: </title><text>{{
68 -  69: {{cite web|url=http://www.
69 -  70: <page> <title>ar
70 -  71: <page> <title>a
71 -  72: ==\n{{reflist}}\n\n==
72 -  73: <page> <title>asa_
73 -  74: <page> <title>as
```

```
 74 -  75: <page> <title>as
 75 -  76: <page> <title>as
 76 -  77: <page> <title>as
 77 -  78: </text> </page>

 78 -  79: <page> <title>as
 79 -  80: american society for
 80 -  81:  [[association fo
 81 -  82: \'\'\' is a [[france|french]] [[association football]]
 82 -  83: <page> <title>a
 83 -  84: <page> <title>at
 84 -  85: <page> <title>at
 85 -  86: <page> <title>at
 86 -  87: <page> <title>at
 87 -  88: <page> <title>a
 88 -  89: {{cite web|url=http://www.
 89 -  90: <page> <title>a
 90 -  91: <page> <title>a
 91 -  92: ]]\n\n{{disambig}}</text> </page>

 92 -  93: }}</text> </page>

 93 -  94: </title><text>{{
 94 -  95: {{unreferenced|date=
 95 -  96: <page> <title>a_b
 96 -  97: <page> <title>a_be
 97 -  98: <page> <title>a_be
 98 -  99:  2012}}\n{{infobox album
 99 - 100:  name        = a big
```

Appendix C1: MPI Implementation.

```
/*
Command to compile (is probably):
mpicc p4_mpi.c -o p4_mpi -mcmodel=medium
*/

#include <mpi.h> /* include MPI */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/time.h>

#define NUM_LINES 1000000 /* Number of lines to read in */
#define LINE_LENGTH 2003 /* Max number of characters to store for each line
*/
#define FILENAME "/homes/dan/625/wiki_dump.txt" /* File to read in line by
line */

typedef unsigned long int uint32;
typedef unsigned int uint16;

uint32 NUM_LINES_PER_THREAD;
uint32 actual_num_lines; /* Number of lines successfully read from file */
char data[NUM_LINES][LINE_LENGTH]; /* All data read in from file */
```

```
char individual[NUM_LINES-1][LINE_LENGTH];
char lcs_data[NUM_LINES-1][LINE_LENGTH]; /* longest common substrings */

void open_file(void);
uint32 lcs_dynamic(char*, const char*, uint32, const char*, uint32);
void thandle(void*);

void open_file()
{
	int count;
	FILE *file;

	file = fopen(FILENAME, "r");
	if (file == NULL)
	{
		perror(FILENAME);
		return;
	}

	for (count = 0; count < NUM_LINES; count++)
	{
		if (fgets(data[count], LINE_LENGTH, file) == NULL)
		{
			break;
		}
	}

	actual_num_lines = count;

	fclose(file);
}


void thandle(void * rank) {
	int tid = *((int*) rank);
	char temp[LINE_LENGTH]; /* temporary string storage     */
	char ret[LINE_LENGTH];  /* formatted string with lcs    */
	char * s1;              /* pointer to string 1          */
	char * s2;              /* pointer to string 2          */
	uint32 len_lcs;         /* length of lcs                */
	uint32 line_number;     /* index into data for s1       */
	uint32 start;           /* index into data for first s1 */
	uint32 end;             /* index into data for last s2  */

	{
		start = tid * NUM_LINES_PER_THREAD;
		end = start + NUM_LINES_PER_THREAD;

		/* Avoid reading past the end of data */
		if(end > (actual_num_lines - 1))
		{
			end = actual_num_lines - 1;
		}

		for (line_number = start; line_number < end; line_number++)
		{
			s1 = data[line_number];
```

```
                  s2 = data[line_number + 1];
                  len_lcs = lcs_dynamic(ret, s1, strlen(s1), s2, strlen(s2));
                  if (len_lcs > 0)
                  {
                        strcpy(individual[line_number], ret);
                  }
                  else
                  {
                        strcpy(individual[line_number], "No common
substring.");
                  }

            }
      }

      return;
}

/*
Find the longest common substring between two strings using dynamic
programming
Return the length of the LCS
Write LCS to ret
*/
uint32 lcs_dynamic(char * ret, const char * a, uint32 a_size, const char * b,
uint32 b_size)
{
      uint32 i; /* loop index */
      uint32 a_idx; /* a index */
      uint32 b_idx; /* b index */

      uint32 a_idx_max = 0; /* a index of end of longest common substring */
      uint32 b_idx_max = 0; /* b index of end of longest common substring */
      uint32 max_length = 0; /* length of longest common substring */

      uint16 ** substring_lengths; /* uint16 limits max substring length to
about 65535 characters, which is more than we need */

                                          /* Allocate space for array
*/
      substring_lengths = (uint16 **)malloc(sizeof(uint16 *) * a_size);
      for (a_idx = 0; a_idx < a_size; a_idx++)
      {
            substring_lengths[a_idx] = (uint16 *)malloc(sizeof(uint16) *
b_size);
      }

      /* Compare each character in a with each character in b */
      for (a_idx = 0; a_idx < a_size; a_idx++)
      {
            for (b_idx = 0; b_idx < b_size; b_idx++)
            {
                  /* If the characters match */
                  if (a[a_idx] == b[b_idx])
                  {
                        /* If one of the characters is the starting character
for that string */
```

```
                        if ((a_idx == 0)
                              || (b_idx == 0))
                        {
                              substring_lengths[a_idx][b_idx] = 1;
                        }
                        else
                        {
                              substring_lengths[a_idx][b_idx] =
substring_lengths[a_idx - 1][b_idx - 1] + 1;
                        }

                        /* Only override longest common substring if a longer
common substring is found */
                        if (substring_lengths[a_idx][b_idx] > max_length)
                        {
                              max_length = substring_lengths[a_idx][b_idx];
                              a_idx_max = a_idx;
                              b_idx_max = b_idx;
                        }
                  }
                  /* If the characters don't match */
                  else
                  {
                        substring_lengths[a_idx][b_idx] = 0;
                  }
            }
      }

      /* Copy longest common substring to ret */
      for (i = 0; i < max_length; i++)
      {
            ret[i] = a[a_idx_max - max_length + i + 1];
      }
      ret[max_length] = '\0';

      /* Free dynamically allocated memory for array */
      for (a_idx = 0; a_idx < a_size; a_idx++)
      {
            free(substring_lengths[a_idx]);
      }
      free(substring_lengths);

      return max_length;
}



int main(int argc, char* argv[])
{
      int i; /* Loop counter */
      int ierr, processNum, rankNum; /* MPI variable */
      struct timeval t1, t2, t3, t4;

      gettimeofday(&t1, NULL);
      open_file();
      gettimeofday(&t2, NULL);
```

```
      ierr = MPI_Init(&argc, &argv); /* Double check arguements later, may
not be necessary*/
      ierr = MPI_Comm_size(MPI_COMM_WORLD, &processNum); /* Comm_size needs a
communicator input*/
      NUM_LINES_PER_THREAD= NUM_LINES/processNum;
      ierr = MPI_Comm_rank(MPI_COMM_WORLD, &rankNum);
        printf("Processes: %d\nRanks: %d\n", processNum, rankNum);
      MPI_Bcast(data, NUM_LINES_PER_THREAD, MPI_CHAR, 0, MPI_COMM_WORLD);
      thandle(&rankNum);
      MPI_Reduce(individual, lcs_data, NUM_LINES_PER_THREAD, MPI_CHAR,
MPI_SUM, 0, MPI_COMM_WORLD);
      ierr = MPI_Finalize(); /* Finish up MPI operations*/
      gettimeofday(&t3,NULL);
      for (i = 0; i < actual_num_lines - 1; i++)
      {
            printf("%3u - %3u: %s\n", i, i + 1, lcs_data[i]);
      }

      gettimeofday(&t4, NULL);
      double time = (t2.tv_sec - t1.tv_sec) * 1000.0;
      time += (t2.tv_usec - t1.tv_usec) / 1000.0;
      printf("Time to read data: %f\n", time);

      time = (t3.tv_sec - t2.tv_sec) * 1000.0;
      time += (t3.tv_usec - t2.tv_usec) / 1000.0;
      printf("Time to determine LCS: %f\n",time);

      time = (t4.tv_sec - t3.tv_sec) * 1000.0;
      time += (t4.tv_usec - t3.tv_usec) / 1000.0;
      printf("Time to print: %f\n", time);
      printf("Program Completed. \n");

      return 0;
}
```

Appendix C2: First 100 lines of result.

```
Processes: 1
Ranks: 0
  0 -    1: </title><text>\'\'\'aa
  1 -    2: </text> </page>

  2 -    3: }}</text> </page>

  3 -    4: <page> <title>a
  4 -    5: \n|foundation = 19
  5 -    6: <page> <title>abc_
  6 -    7: <page> <title>abc_
  7 -    8:  the [[australian broadcasting corporation]]
  8 -    9:  the [[australian broadcasting corporation]]
  9 -   10:  [[australian broadcasting corporation]]
 10 -   11: </title><text>{{infobox
 11 -   12: <page> <title>ab
```

```
12 -  13: </title><text>\'\'\'ab
13 -  14: </title><text>\'\'\'a
14 -  15: <page> <title>ac
15 -  16: <page> <title>acc
16 -  17: \n\n{{disambig}}</text> </page>

17 -  18: }}</text> </page>

18 -  19: \n\n{{disambiguation}}</text> </page>

19 -  20: </title><text>\'\'\'ac
20 -  21: <page> <title>ac
21 -  22: <page> <title>ac_
22 -  23: <page> <title>ac_
23 -  24: </text> </page>

24 -  25: \'\'\' may refer to:\n
25 -  26: \'\'\' may refer to:\n\n
26 -  27: <page> <title>ad
27 -  28: <page> <title>ad
28 -  29: <page> <title>a
29 -  30: \n\n{{disambig}}</text> </page>

30 -  31: <page> <title>afc
31 -  32: <page> <title>af
32 -  33: </text> </page>

33 -  34: <page> <title>a
34 -  35: </title><text>{{infobox
35 -  36: </title><text>{{
36 -  37: <page> <title>a
37 -  38: <page> <title>aid
38 -  39: <page> <title>ai
39 -  40: [[australian institute of
40 -  41: <page> <title>a
41 -  42: \n\n{{disambig}}</text> </page>

42 -  43: ]]\n\n{{disambig}}</text> </page>

43 -  44: </text> </page>

44 -  45: n-stub}}</text> </page>

45 -  46: </text> </page>

46 -  47: </text> </page>

47 -  48: <page> <title>alar
48 -  49: <page> <title>al
49 -  50: <page> <title>alp
50 -  51: <page> <title>a
51 -  52: <page> <title>am
52 -  53: <page> <title>am
53 -  54: <page> <title>am
54 -  55: <page> <title>am
55 -  56: }}\n\n{{defaultsort:am
56 -  57: </title><text>{{unreferenced
```

```
class=\"wikitable\"
<page> <title>an
<page> <title>a
<page> <title>a
<page> <title>ap
<page> <title>ap
<page> <title>ap
<page> <title>ap
\n\n==external links==\n*
<page> <title>ap
</title><text>{{
{{cite web|url=http://www.
<page> <title>ar
<page> <title>a
==\n{{reflist}}\n\n==
<page> <title>asa_
<page> <title>as
<page> <title>as
<page> <title>as
<page> <title>as
</text> </page>

<page> <title>as
american society for
 [[association fo
\'\'\' is a [[france|french]] [[association football]]
<page> <title>a
<page> <title>at
<page> <title>at
<page> <title>at
<page> <title>at
<page> <title>a
{{cite web|url=http://www.
<page> <title>a
<page> <title>a
]]\n\n{{disambig}}</text> </page>

}}</text> </page>

</title><text>{{
{{unreferenced|date=
<page> <title>a_b
<page> <title>a_be
<page> <title>a_be
 2012}}\n{{infobox album
 name        = a big
```