Sri Lanka Institute of Information Technology

A Comparative Study of Parallelization Approaches using OpenMP, MPI, and OneAPI for String Search and Image Processing

Methmi Nugawela

Year 4, Semester 2

December 18, 2022

Parallel Computing – SE4060

Contents

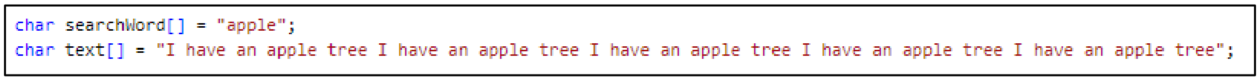
1. Recursion - String Search 2
  1.1. Description of Solution 2
  1.2. General Parallelization Strategy 2
  1.3. Research referenced 3
  1.4. OpenMP 3
    Timing Graphs 3
    Specific Parallelization Strategies Used and Reflection 4
    Limitations 4
  1.5. MPI 5
    Timing Graphs 5
    Specific Parallelization Strategies Used and Reflection 6
2. Data Manipulation - Image Processing 6
  2.1. Description of Solution 6
  2.2. General Parallelization Strategy 6
  2.3. Research Referenced 7
  2.4. OpenMP 8
    Timing Graphs 8
    Specific Parallelization Strategies Used and Reflection 8
    Limitations 9
  2.5. MPI 9
    Specific Parallelization Strategies Used and Reflection 9
    Limitations 9
  3.1. One API 9
    Parallelization Strategies Used and Reflection 9
    Limitations 9

1. Recursion - String Search

1.1. Description of Solution

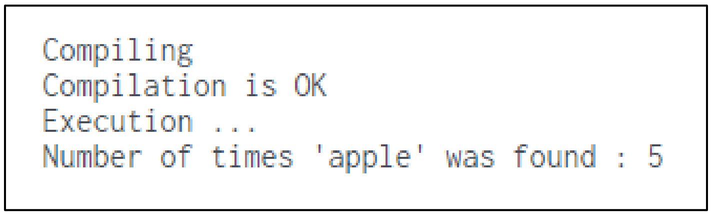
The program searches for a given search word within a sentence and outputs the number of times the word occurred. The matchWord( ) function created, uses strtok( ) to extract a word split by the nearest space and recursively compares the search word to the extracted word and increments a private counter for number of matches which is then returned at the end of the function when there are no more words left to extract. Then each recursion's private value for the number of matches are added to the previous recursions match count and the outermost function call returns the total number of matches collected from all recursions.

Sample Input and Output



```
char searchWord[] = "apple";
char text[] = "I have an apple tree I have an apple tree I have an apple tree I have an apple tree I have an apple tree";
```

Figure 1: Sample Input



```
Compiling
Compilation is OK
Execution ...
Number of times 'apple' was found : 5
```

Figure 2: Sample Output

1.2. General Parallelization Strategy

The divide and conquer algorithm is used to divide the main problem (text[ ]) into smaller sub-problems (sub strings) according to the number of nodes/processors available which is then split across all available nodes/processors which return a value for the number of matches each node/processor found in their substring.

The main string was split by dividing the total number of characters in the text string array by the number of nodes and checking if the character in that position is a space or immediately next to a space, which would signify that this character is the end of a complete word. This prevented the substrings from having partial words which will go undetected as a matching word by the matchWord( ) function. Searching is then performed on each substring.

‘t’	‘ ’	‘a’	or	‘t’	‘s	‘ ‘	or	‘ ‘	‘s’	‘t’

Figure 3: The characters that are chosen to be used as a split pint are shown shaded

1.3.Research referenced

Raju, S. Viswanadha, K. K. V. V. S. Reddy, and Chinta Someswara Rao. "Parallel string matching with linear array, butterfly and divide and conquer models." Annals of Data Science 5.2 (2018): 181-207.

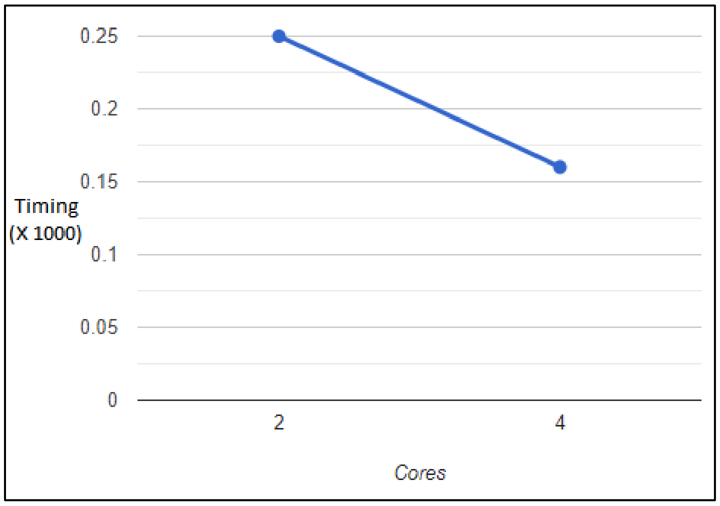
Bentley, Jon L., and Robert Sedgewick. "Fast algorithms for sorting and searching strings." Proceedings of the eighth annual ACM-SIAM symposium on Discrete algorithms. 1997.

1.4.OpenMP

Timing Graphs

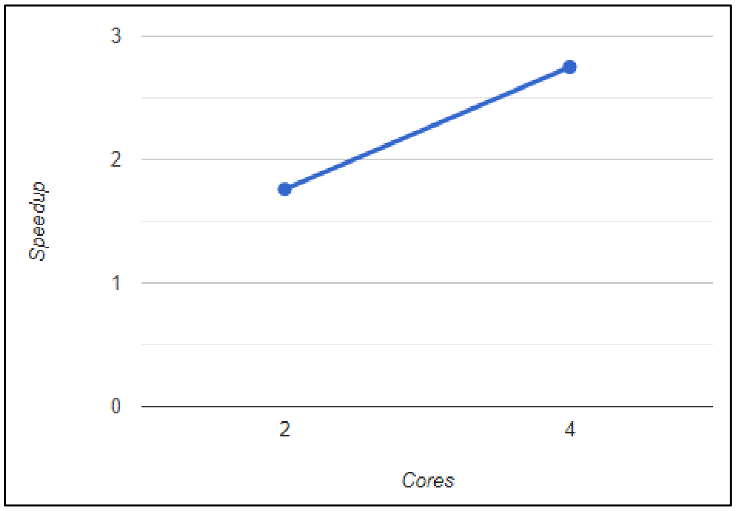
Cores vs Timing

Each node has two cores.



Cores vs Speedup

Each node has two cores.



Specific Parallelization Strategies Used and Reflection

The two sections identified with code that can be executed concurrently were the for loop for splitting and the recursive matchWord() function. In addition to the strategy mentioned in section 1.2, threads were used to parallelize the iterations of the for loops run to split the strings. #pragma omp parallel for collapse(2) was used to parallelize the two nested for loops by running each iteration on a separate thread. collapse(2) was used to flatten the two nested loops so that each iteration would have an individual thread and not just the outer loop. Towards the end of the loop, #pragma omp critical was used when adding up the match totals of all substrings to prevent a data race #pragma omp task was used to execute the recursive tasks parallelly and then all parallel threads of the tasks were synchronized with #pragma omp taskwait before returning the number of matches found.

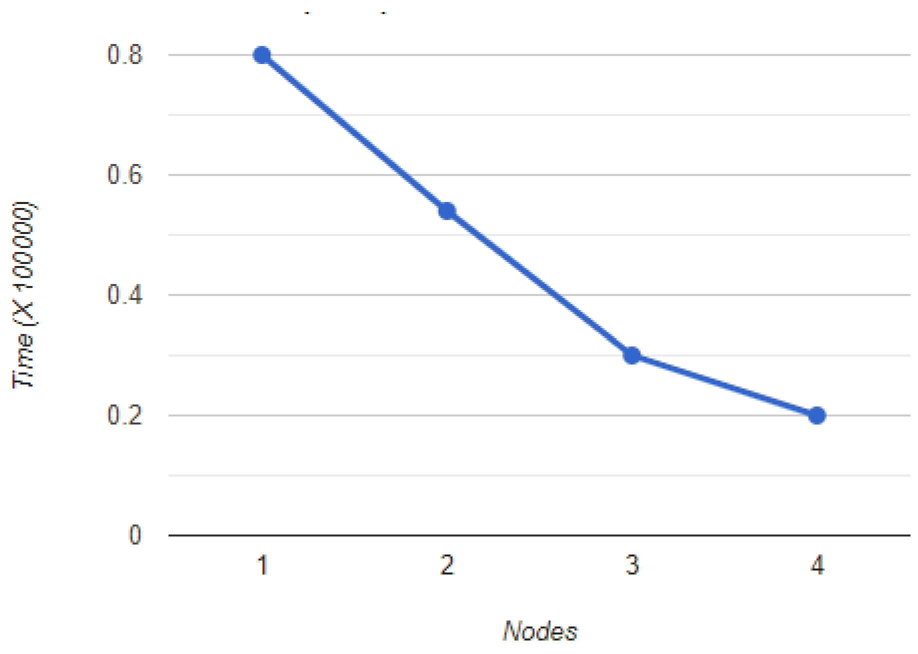
Limitations

The method of spitting the problem into substrings depends on the number of processors present. If the number of characters in text string is not divisible by the number of processors present, the workload will have to be split unevenly. Furthermore, the process of making sure words are not split into sub strings involve assigning characters of the same word to one processor even when it has reached its maximum required number of characters to split the string evenly leading to more uneven splitting. Furthermore, the #pragma omp critical command only allowing one thread through could significantly slow down the process when many threads are involved.

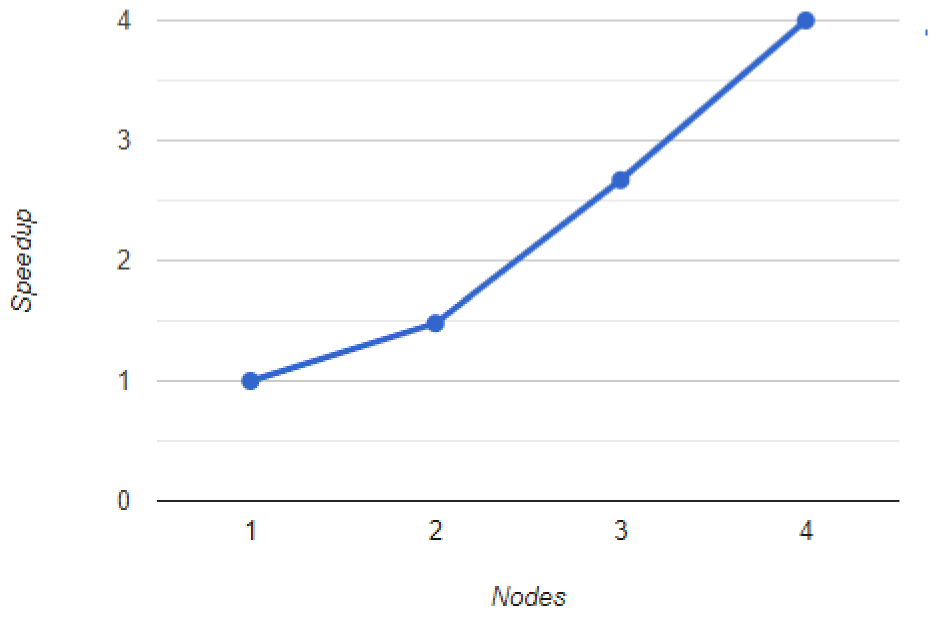
1.5. MPI

Timing Graphs

Nodes vs Timing



Nodes vs Speedup



Specific Parallelization Strategies Used and Reflection

The strategy mentioned in section 1.2 was used to split the text into sub strings to be sent to each node. Then, MPI_Bsend( ) is invoked ina aloop to send each split string to each node. MPI_Bsend( ) was used over MPI_Ssend( ) since the sending node would also be receiving a string and therefore having MPI_Ssend( ) would cause a deadlock in the sending node. After each node receives their substring, they each perform the recursive matchWord() function on it. The number of matches found on each node is sent to the master node and totaled with MPI_Reduce( ) used with the MPI_SUM( ) operation. The master node then prints the total received.

2. Data Manipulation - Image Processing

2.1. Description of Solution

A PNG image with values for RGBA channels is input and each pixel's value in each of the RGB channels is used to calculate the grayscale value of that pixel, the output pixel values are then encoded as a PNG image, which is a gray scale version of input image.

Sample Input and Output

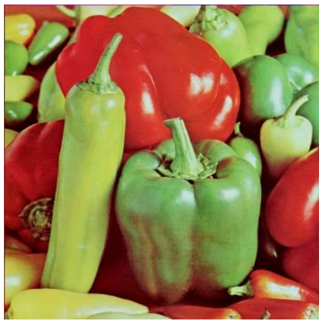


Figure 4: Sample Input

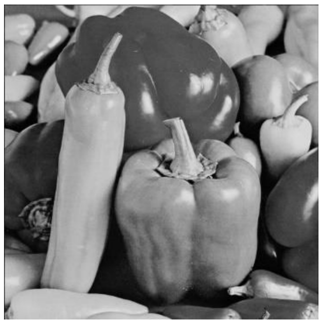


Figure 5: Sample Output

2.2. General Parallelization Strategy

As the decoded image has 4 channels and only 3 of them are needed for gray-scaling, a new array is created and the RGB values for each pixel in the array test_img is copied to rgb_img, this is done via two nested for loops. Another array called output_img is created to the size of test_img array to store grayscale image. This rgb_img array is what is accessed when the grayscale function is called on each pixel. Next, to access each of values for RGB for each pixel, another two for loops are nested to navigate through the rows and columns of the rgb_img.

The channel values are stored in a 1-dimensional array as shown below, test_img and output_img will have 4 elements per pixel and rgb_img will have 3 elements per pixel:

r	g	b	a	r	g	b	a	r	g	b	a
Pixel 1 channel values				Pixel 2 channel values				Pixel 3 channel values			

The set of array values belonging to the current pixel is located on the rgb_img array by

(row)*width*3 + (column)*3

where width is the width of the image and its multiplied by 4 due to rgb_img have 3 (rgb) values for each pixel. This pixel index value is sent to the grayscale function at each iteration which calculates the grayscale values from the set of array values belonging to the pixel index value received. The calculated grayscale value is output and is stored for each channel element value at the pixel index in output array.

2.3. Research Referenced

Asaduzzaman, Abu, Angel Martinez, and Aras Sepehri. "A time-efficient image processing algorithm for multicore/manycore parallel computing." SoutheastCon 2015. IEEE, 2015.

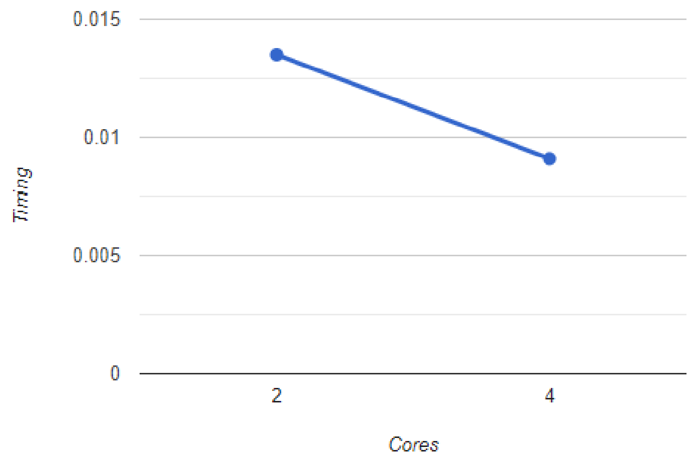
Rosenfeld, Azriel. "Parallel image processing using cellular arrays." Computer 16.01 (1983): 14-20.

2.4. OpenMP

Timing Graphs

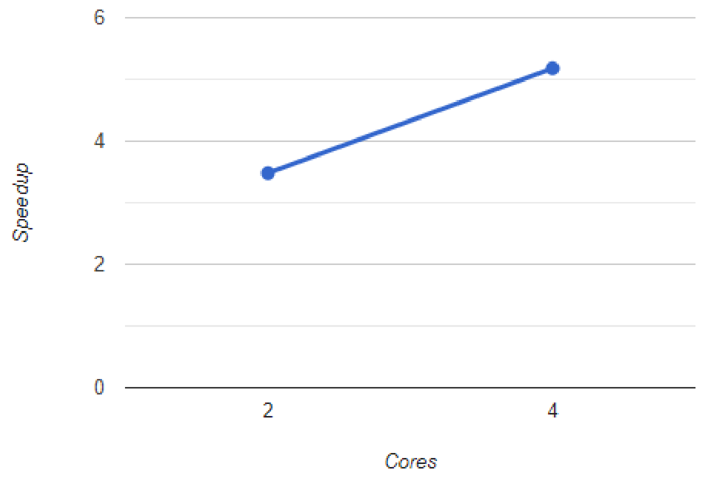
Cores vs Timing

Each node has two cores.



Cores vs Speedup

Each node has two cores.



Specific Parallelization Strategies Used and Reflection

The program used 2 pairs of nested for loops one t split the array and another to access each pixel's RGB values ad store grayscale value in output_img. Therefore, for each for loop, #pragma omp parallel for collapse(2) was used to parallelize the two nested for loops by running each iteration on a separate thread. collapse(2) was used to flatten the two nested loops so that each iteration would have an individual thread and not just the outer loop. No critical sections were identified due to each iteration being designed to only access one pixel, thus each thread would be specifically accessing a single pixel's range of elements only.

Limitations

Having to convert the 1D array with 4 elements per pixel to 3 elements per pixel took time.

2.5. MPI

Specific Parallelization Strategies Used and Reflection

The method explained in section 2.2 was used and parallelized by splitting the test_img among the nodes present into sub_img arrays. Then rgb_img and output) img created were broadcasted to the whole cluster for their reference. The image splitting was carried out by the scatter command as it handled both send and received. Finally MPI_Ibarrier( ) synchronized all nodes in cluster.

Limitations

3. Having to convert the 1D array with 4 elements per pixel to 3 elements per pixel took time.

3.1. One API

Parallelization Strategies Used and Reflection

A gpu_selector was chosen for the queue and each iteration in the parallel for running on the queue was treated as an access to an individual element in the rgb_img. Therefore, to find out which channel element in the pixel group the iteration is in, the modulus of 4 for the iteration value (i) is taken as there are 4 elements in a pixel group, and the remainder was used to determine the channel (0: red, 1: green. 2: blue, 3: alpha). This was determined via a switch case. When the channel pixel location was verified, the pixel access location of its other channel values by checking to the let and right of the pixel value in the iteration.

Limitations

Since recursion cannot occur within a OneAPI kernel, methods like sorting need workarounds. The sorting method was required the grayscale image.