PROGRAMMING ON SHARED ADDRESS PLATFORM
Assignment Report
Course CS430

(Parallel Programming)

System Specifications:

1. Debian System (Ubuntu 14.04)
2. GCC compiler version 4.8
3. 2 Cores System

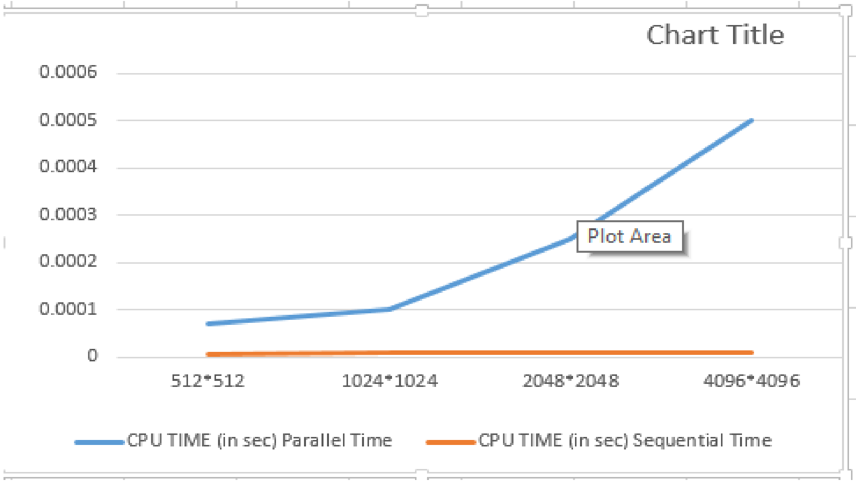
1) Array Sum using openMP

The array sum was done first sequentially and then using the openMp pragmas and the results were noted down. The results obtained are shown here in table and performance graph is also plotted.

Input Size	CPU TIME (in sec)	
	Parallel Time	Sequential Time
512*512	0.00007	.000005
1024*1024	0.00010	.000009
2048*2048	.00025	.00001
4096*4096	.00050	.00001

CONCLUSION:

a. The graph is shown here which clearly shows that as the size of matrix increases the CPU Time by sequential drastically increases while such doesn’t occur for parallel code. The computation is simple for matrix sum so sequential is running better.



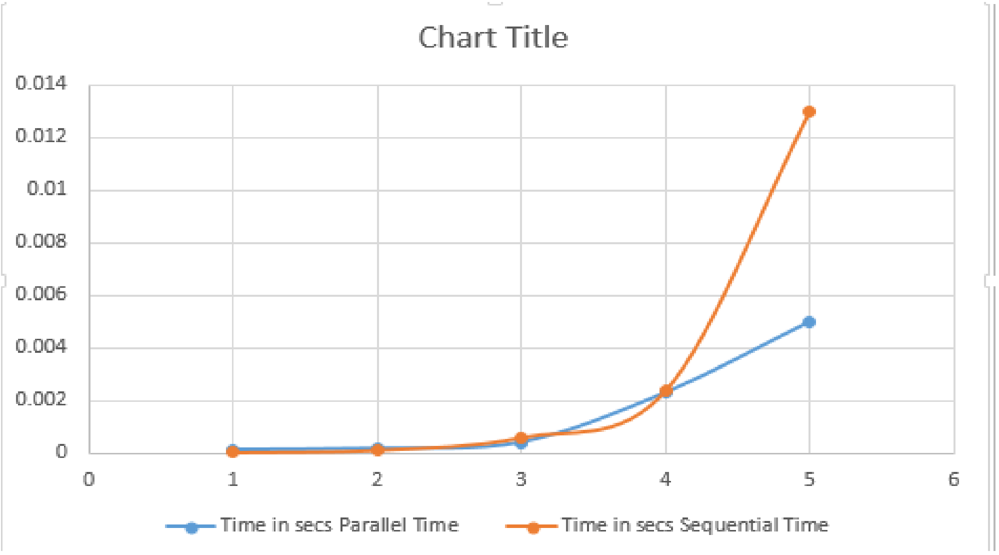
2) Matrix Transpose

The matrix transpose results were obtained by using sequential and then parallel. Results were compared to the output of sequential code.

Matrix Size	Processes	Time in secs	
		Parallel Time	Sequential Time
64*64	2 Processes	.00011	.000027
128*128		0.00016	.00012
256*256		.0004	.0006
512*512		.0023	.0024
1024*1024		.005	.013

CONCLUSION:

a. As the problem size increases the sequential codes become incompetent as compared to those parallel codes on computation timing aspect.



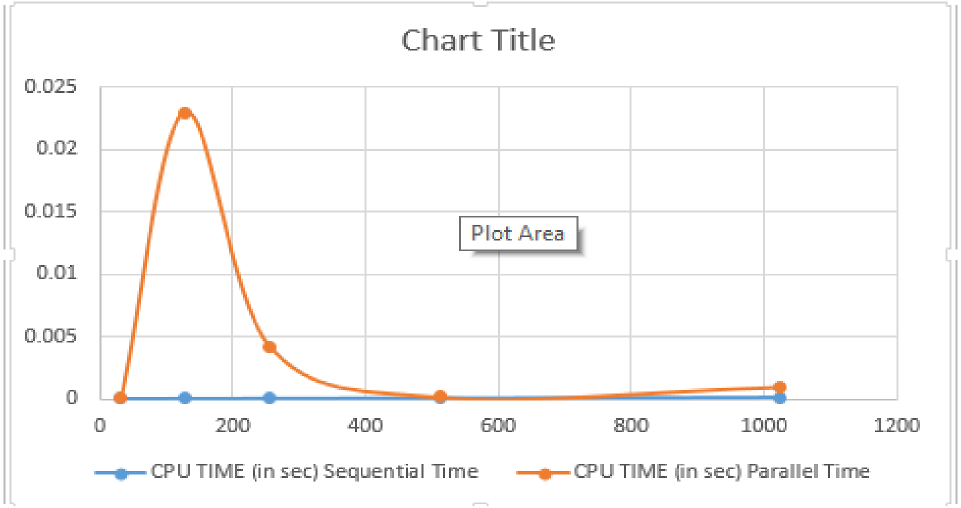
3) Prefix Sum

The sequential and parallel prefix sum is calculated and result is shown here.

Input Size	CPU TIME (in sec)	
	Sequential Time	Parallel Time
32	.000014	.0001
128	.000025	0.023
256	0.000039	.0042
512	.000061	.00017
1024	.000117	.00096

CONCLUSION:

For the problem with small sizes the sequential code is very fast but as increment in size is done parallel code becomes efficient (as shown in plot).

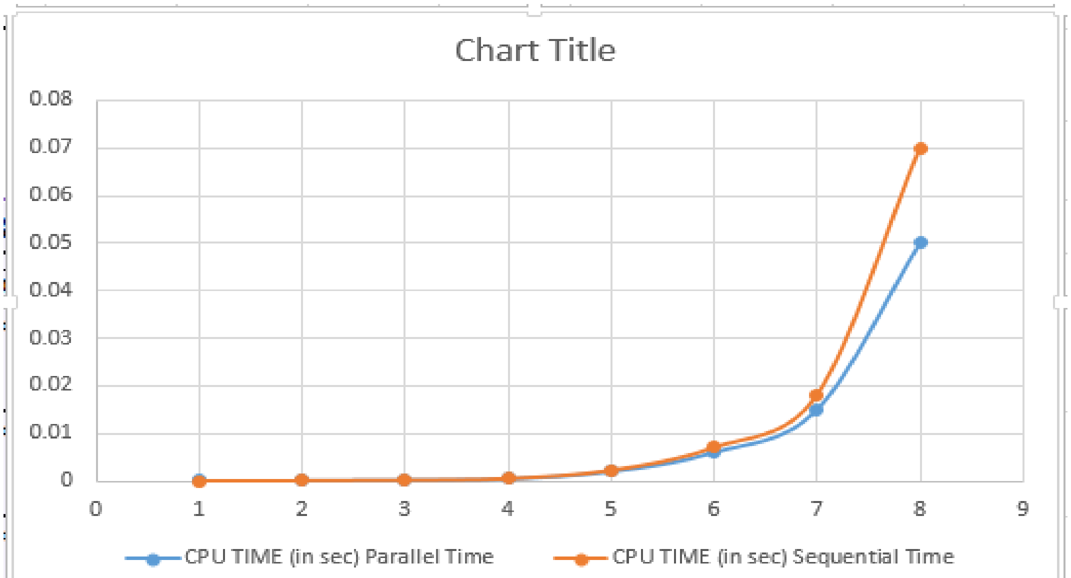


4) Matrix Vector Multiplication

Input Size	CPU TIME (in sec)	
	Parallel Time	Sequential Time
32*32	.000082	.00001
64*64	.00011	.000037
128*128	.00021	.00014
256*256	.00048	.00048
512*512	.0020	.0022
1024*1024	.006	.007
2048*2048	.015	.018
4096*4096	.05	.07

CONCLUSION:

As the problem size increases the sequential codes become incompetent as compared to those parallel codes on computation timing aspect (as shown in plot).



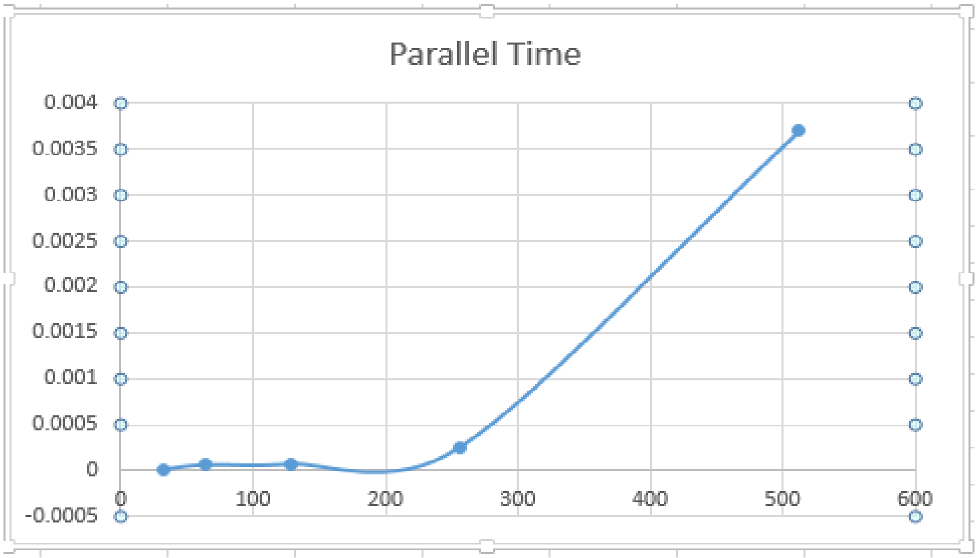
5) **Pointer Jumping :** Number of processor are set as 2 and at different size the comparison is done and graph is plot.

Input Size	Parallel Time
32	.000012
64	.00006
128	.00007

256	.00025
512	.0037

CONCLUSION:

As the problem size increases the processors gives better performance if time is checked only (as shown in plot).


Parallel Time