Parallel Implementation of 2D FFT

High Performance Computing Project

Prithvi Patel: 201501230
Swastika Nayak: 201501423

Introduction

- Parallelization of 2D FFT on a grayscale image using Radix-2 Cooley Tukey algorithm.
- Two parallelization techniques are used for the parallelizing the algorithm.
- 2D FFT has many applications in Digital image processing as it converts the image into its sine and cosine components that is the spatial frequency domain.
- The algorithm is limited here to images of size in powers of 2.

Serial Implementation and Complexity

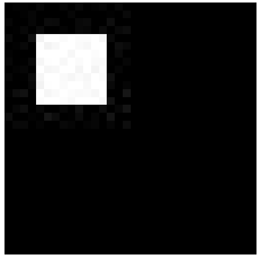
The algorithm has been divided into following tasks

1. Convert Image into Complex array
2. 1D FFT on each row
3. Transpose of matrix
4. 1D FFT on each row of transposed matrix (actually to perform FFT on each column of original)
5. Transpose of matrix
6. Compute Magnitude of the complex 2D array
7. Convert Magnitude to grayscale image

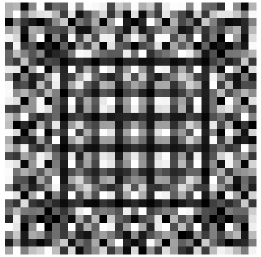
Complexity: $O(N^2 \log_2 N)$, where $N \log_2 N$ is the complexity of calculating the FFT of one dimension.

Input Output

Input:



Output:



The magnitude of complex array was verified with the results obtained in MATLAB.

The output image can be normalized to get accurate results. We have let the machine assign the values to the calculated magnitude of each pixel.

Serial Pseudocode

```
FFT_2D (complex 2D array(NXN))
{
for (i=0,N)
        FFT(Row(i));
}

FFT(Row(i))
{
Perform Bit reversal;
z = Row(i);
stride_j=1
for k = (1,logN) {
        stride_i = 2*stride_j;
        for i = (0,(N/stride_i)){
                twiddle_index = 0;
                j = i*stride_i;
                while  (twiddle_index < stride_j){
                        j2 = j+stride_j;
                        u =w[(twiddle_index*N/stride_i)];
                        t = multiply(u, z[j2]);
                        z[j2] = z[j] - t;
                        z[j] = z[j] + t;
                        j++;
                        twiddle_index++;
                }
        }
        stride_j *= 2;
}
```

Scope of parallelization

Theoretical Speedup:

Using the profiling using gprof, Flat profile reveals that FFT takes up 84.63% and calculating transpose takes up 6.35% of execution time. The parallelizable fraction as obtained using timers is 0.8979.

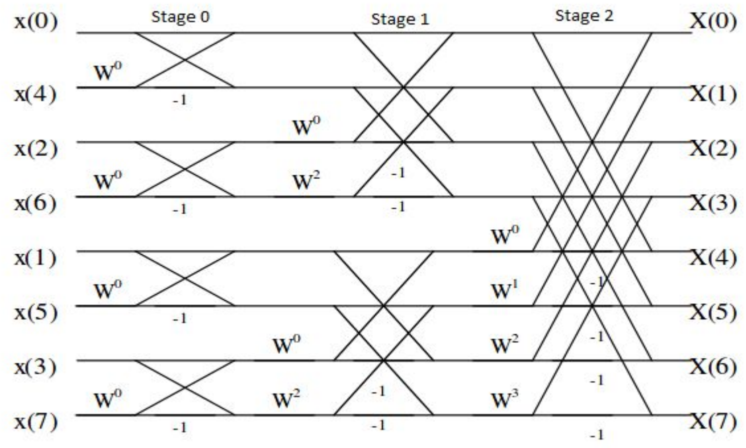
Theoretical Speedup = 3.0767 (Using Amdahl's Law for 4 cores)

Parallelization Strategies:

To exploit spatial locality, block method is used for transposing the matrix, however it is not as efficient as there are cache misses along the columns.

FFT calculation has been parallelized in ways as shown in next slide.

Parallelization Strategies

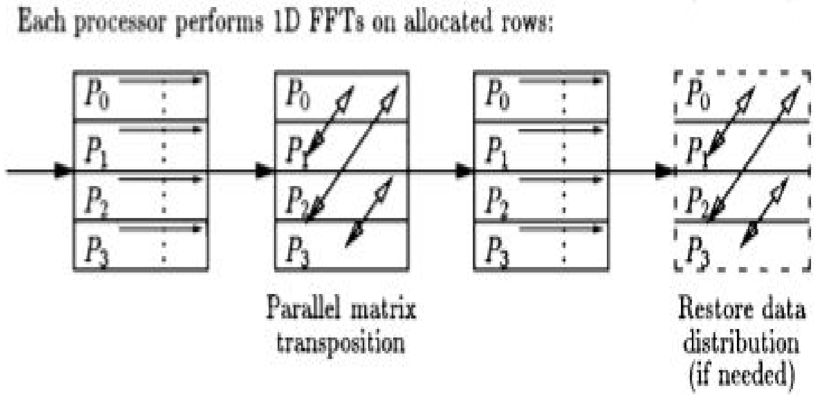


Each butterfly operation calculated in 1D FFT is parallelized. That is for above case, assign 4 processors to 0th stage, 2 to 1st stage and so on. The second for loop in above pseudocode of FFT has been distributed to processors that is computational work has been mapped

Complexity:: $O(N^2\ (\log_2(N/p) + \log_2 p)\ /p)$

overhead $\propto \log_2 p$

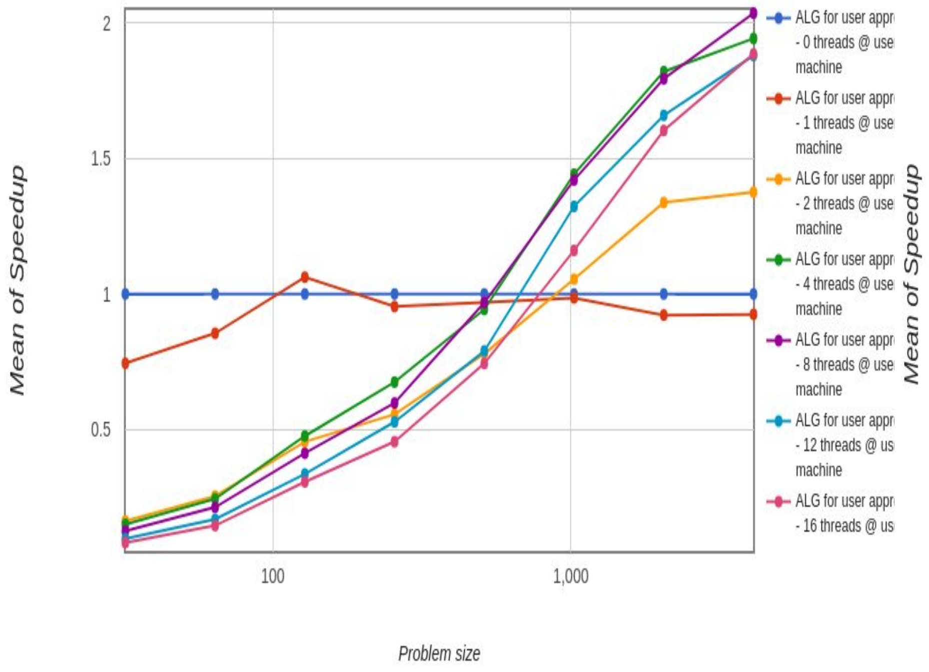
In the second approach the data has been decomposed. The processors operate on independent rows of the image and thus there is no synchronization issue as was there in first approach where stages are dependent.


Each processor performs 1D FFTs on allocated rows:
P0
P1
P2
P3
Parallel matrix transposition
Restore data distribution (if needed)

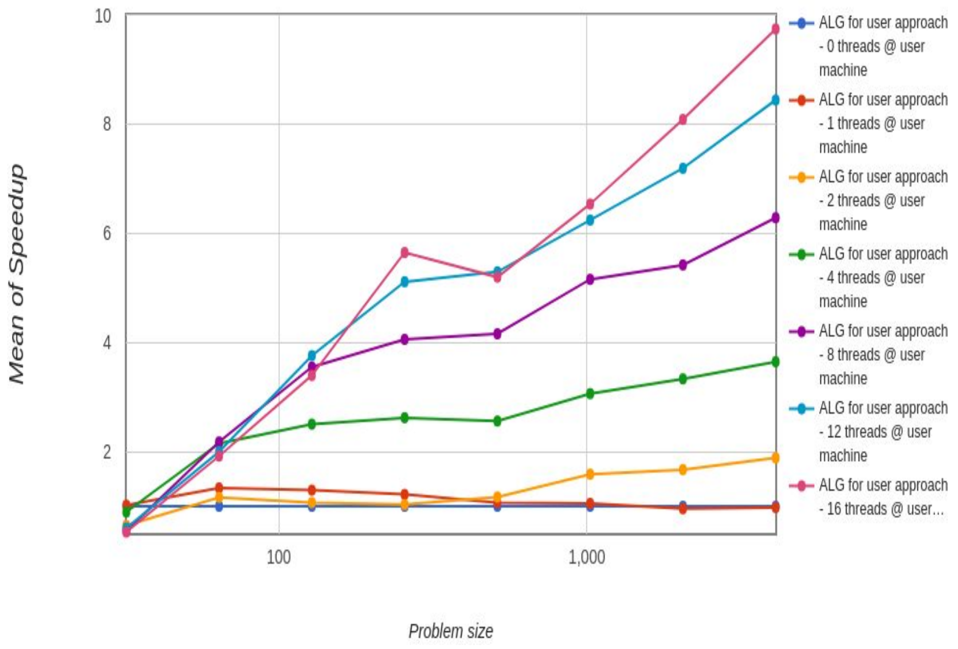
Complexity: $O(N^2\log_2(N)/p)$

Curve Based Analysis : Speedup Curves

Approach 1

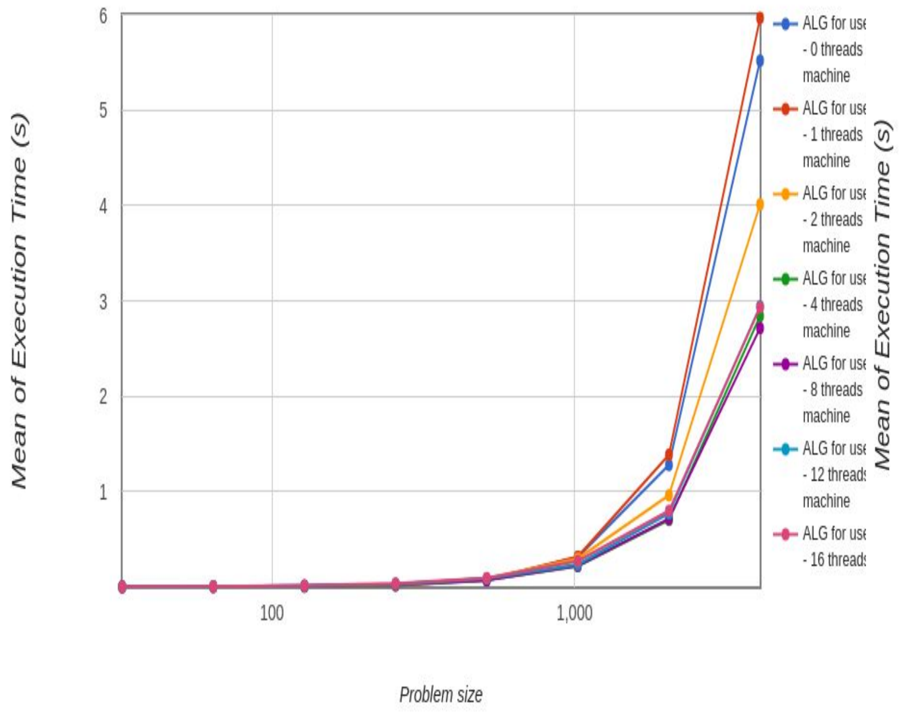


Approach 2

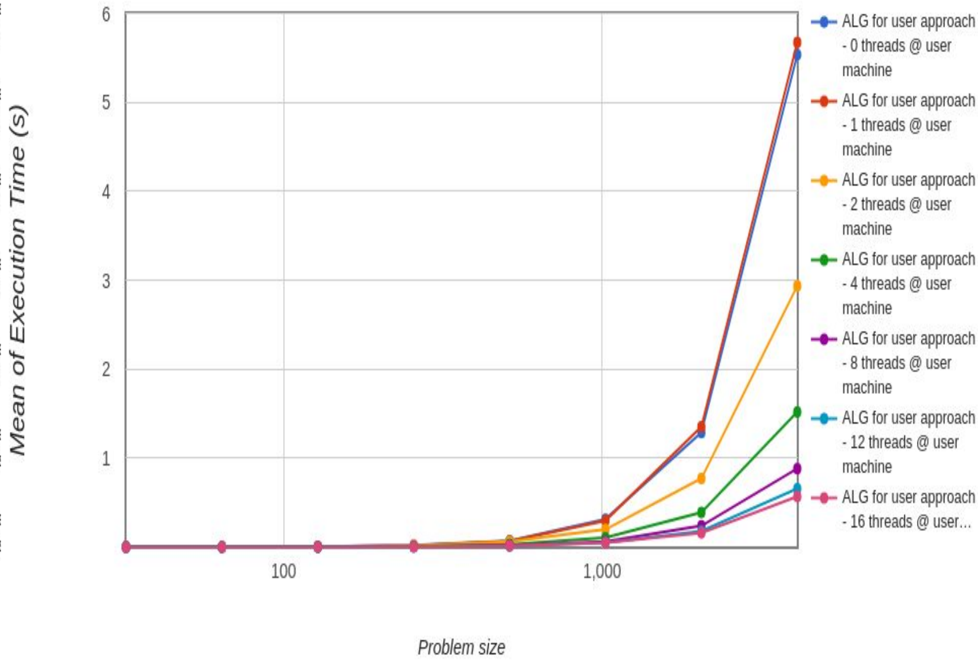


Curve Based Analysis : Problem Size vs Time

Approach 1



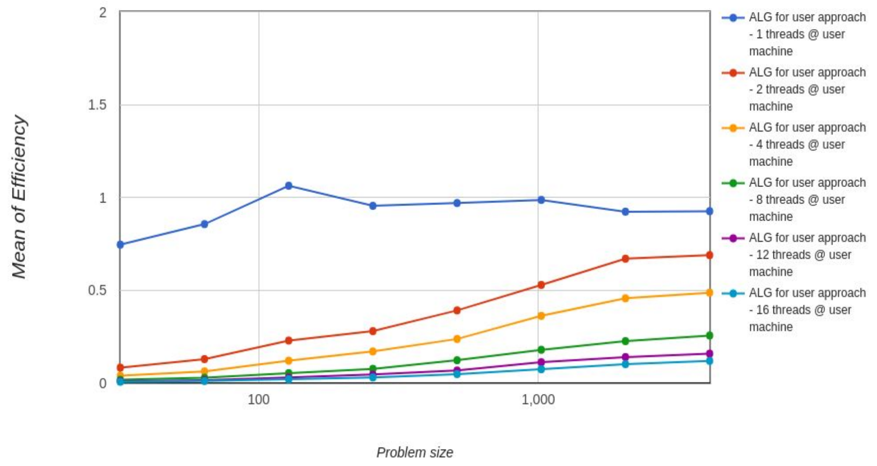
Approach 2

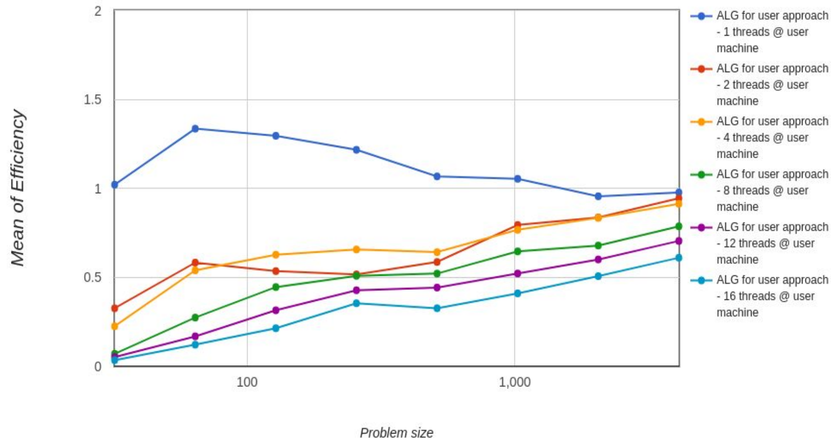


Efficiency Curves

Approach 1

Approach 2





The above approach is weakly scalable not efficient.

The above approach is scalable for larger values of N and the efficiency is increased by increasing both N and p.

Observations:

Approach 1:

- Amount of communication depends on number of cores and problem size N both
- Idling of processors make speedup and time curves saturate
- Initially less computations per processor, but it increases gradually in logarithmic fashion
- For getting more speedup, N needs to be increased for $p>4$ as computations in comparison of communication needs to be increased
- Cache Coherence adds to overhead
- Synchronization is necessary (implicit barrier)

Approach 2:

- No good speedup for smaller problem sizes
- Code scalable for higher problem sizes when cores are less. To increase scalability, increase problem size N
- Coarse grained granularity as each processor does many computations sequentially and has little communication
- There is no need of synchronization as each row's 1D FFT is independent.

Karp Flatt Metric Analysis

Approach 1:
The idle time and the overhead increases as the serial fraction is increasing.

p	2	4	6	8	10	12	14	16
speedup	1.376	1.942	1.983	2.036	1.972	1.88	1.885	1.885
e	0.453	0.353	0.405	0.418	0.452	0.489	0.494	0.499

Approach 2:
The overhead is more for lesser number of cores and higher number of cores. Thus, the optimal number of cores is about 6-12 here.

p	2	4	6	8	10	12	14	16
speedup	1.886	3.646	5.118	6.284	7.442	8.44	9.039	9.741
e	0.060	0.032	0.035	0.039	0.038	0.038	0.042	0.043

Conclusion

- Second approach of dividing the data among the processors in such a way that there is almost no synchronization is better way to parallelize 2D FFT even when large computations are done by a processor.
- Approach 2 faces the problem of overhead when we increase the number of processors.
- It is not necessary that we parallelize the 1D FFT as it has major overhead even when a processor does relatively smaller number of computations.
- Block transpose method cannot be further optimized as the problem of cache misses will persist.