COL380
Assignment 1

Utkarsh Singh 2021CS10558
Hemang Sidana 2021CS10078
Shubham 2021CS10112

Contents

1 Pseudo Code

Inputs: $a(n, n)$
Outputs: $\pi(n)$, $l(n, n)$, and $u(n, n)$

Algorithm 1: LU Decomposition with Partial Pivoting

```
Data: a(n, n)
Result: π(n), l(n, n), u(n, n)
Initialize π as a vector of length n
Initialize u as an n × n matrix with 0s below the diagonal
Initialize l as an n × n matrix with 1s on the diagonal and 0s above the diagonal
for i = 1 to n do
    π[i] = i
end
for k = 1 to n do
    max = 0
    for i = k to n do
        if max < |a(i, k)| then
            max = |a(i, k)|
            k' = i
        end
    end
    if max == 0 then
        error (singular matrix)
    end
    swap π[k] and π[k']
    swap a(k, :) and a(k', :)
    swap l(k, 1 : k − 1) and l(k', 1 : k − 1)
    u(k, k) = a(k, k)
    for i = k + 1 to n do
        l(i, k) = a(i, k) / u(k, k)
        u(k, i) = a(k, i)
    end
    do in parallel
        for i = k + 1 to n do
            for j = k + 1 to n do
                a(i, j) = a(i, j) − l(i, k) × u(k, j)
            end
        end
    end
end
```

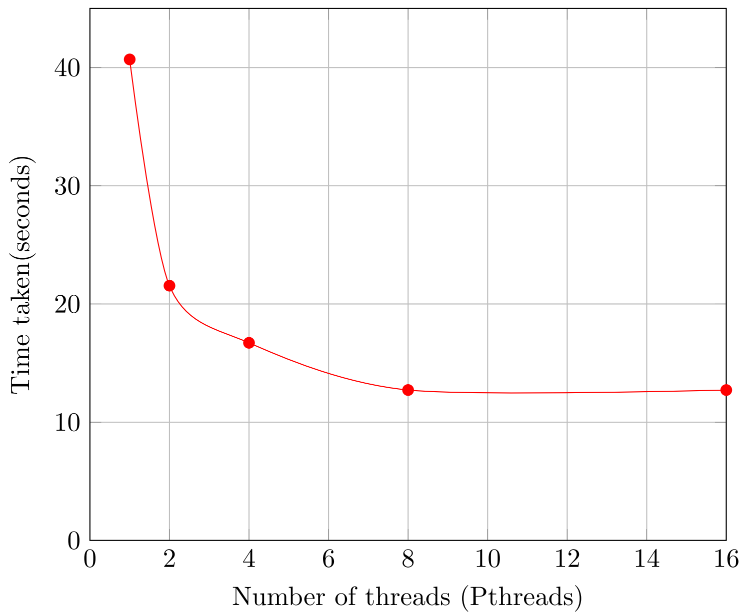
2 Explanation

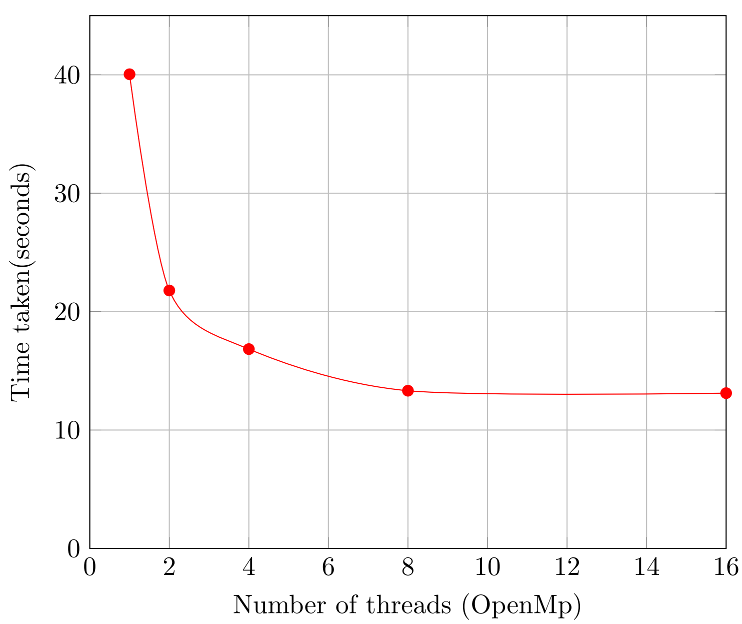
- Time Complexity of program is O(n^3).
- Time Complexity of each iteration of outermost for loop is O(n^2).
- Time Complexity of nested for loops in each iteration is O(n^2) while the remaining part is linear in time.
- We parallelized those nested for loops (line number 27-31) as they don't have any **loop-carried dependencies**.
- During parallelization, iterations of the outer loop are distributed equally among the threads.
- Within the loop at line 27, each thread operates on a separate segment of matrix 'a'.
- As a result, no thread accesses the same memory region concurrently.
- This partitioning of work ensures that memory is not shared across threads during this loop's execution.
- The absence of shared memory access eliminates the need for synchronization mechanisms between threads within this loop.
- We found that the part we parallelized was taking approximately 99% of the total time.
- We also checked runtimes after parallelizing other sections of the code but it increased the time due to **parallel overheads**.

3 Observations

1. **Initial Steep Decline:** There is a significant decrease in time as the number of threads increases from 1 to 4, suggesting that the task benefits greatly from parallel processing and the workload is efficiently distributed across multiple threads.
2. **Diminishing Returns:** The curve flattens as the number of threads exceeds 4. This can be attributed to:
   - Overhead in thread management.
   - Saturation of CPU resources.
   - Memory bandwidth becoming a limiting factor.
3. **Plateauing Effect:** The graph becomes horizontal, indicating no further significant decrease in processing time with additional threads due to:
   - Utilization of all available CPU cores.
   - Inherent serial components in the task.
   - Overhead from synchronization and inter-thread communication.
4. **Minor Variations:** Slight fluctuations in the flat portion of the curve may result from CPU scheduling inconsistencies, background process interruptions, or other system activities.

4 Program Time





5 Parallel Efficiency

$$\epsilon = \frac{T_1}{p * T_p}$$

where T_1 is time taken by single thread
T_p is time taken by p threads
ϵ is parallel efficiency of the program