A PARALLEL APPROACH TO THE STABLE MATCHING PROBLEM

Naveen Udhayasankar, CSE 633 Parallel Algorithms, Final Presentation

University at Buffalo The State University of New York

What is the stable matching problem?

Given N men and N women, where each person has ranked all members of the opposite gender in order of preference, marry the men and women together such that there are no two people of opposite gender who would both rather have each other than their current partners. If there are no such people, all the marriages are “stable”.

https://en.wikipedia.org/wiki/Stable_marriage_problem

What is the stable matching problem?

Consider the following example.

- Let there be two men m1 and m2 and two women w1 and w2.
- Let m1's list of preferences be {w1, w2}
- Let m2's list of preferences be {w1, w2}
- Let w1's list of preferences be {m1, m2}
- Let w2's list of preferences be {m1, m2}

The matching { {m1, w2}, {m2, w1} } is not stable because m1 and w1 would prefer each other over their assigned partners. The matching {m1, w1} and {m2, w2} is stable because there are no two people of opposite sex that would prefer each other over their assigned partners.

https://www.geeksforgeeks.org/stable-marriage-problem/

Forming a stable matching

- **Gale Shapley Algorithm (1962)**
  - David Gale and Lloyd Shapley proved that, for any equal number of men and women, it is always possible to solve the stable matching problem.
  - Always favors one gender over the other.
  - Initially all persons are free.
  - The men start off by proposing the woman at the beginning of their preference list.
  - The women receive the proposal and accept it if they are free. Else, they compare it with their existing proposal and select the best out of the two proposals.
  - The result of the algorithm doesn't depend on the order in which the men/the women propose.

Gale Shapley Algorithm

A sequential version of the Gale Shapley algorithm, in which the men propose first.

Algorithm 1 Gale - Shapley Algorithm for stable matching

$m \epsilon M \leftarrow free$
$w \epsilon W \leftarrow free$
while ∃ free man m who has a woman w to propose to **do**
  $w \leftarrow$ first woman on m's list to whom m hasn't yet proposed
  if ∃ some pair (m', w) **then**
    if w prefers m to m' **then**
      $m' \leftarrow free$
      $(m, w) \leftarrow engaged$
    else
      $(m', w) \leftarrow engaged$
    end if
  end if
end while

https://en.wikipedia.org/wiki/Stable_marriage_problem

Proof of termination

Proposition:

The algorithm described for stable matching terminates.

Proof:

Assume there are n men and n women involved. A man must propose to at most n women before being accepted or being rejected by the final one. So at most n^2 proposals may occur, after which the algorithm terminates.

Proof of engagement

Proposition:

At the end of the algorithm, all men/women are married.

Proof:

- Assume towards contradiction that m is an unmarried man at the termination of the algorithm.
- Then there is some unmarried woman w, since there are the same number of men and women and no one can be married to more than one person.
- So if a woman gets even one proposal, she is married when the algorithm terminates. This means that the woman received no proposals.
- But, in order for the algorithm to terminate man m must be married, which he is not, or have been rejected by every woman, including w.
- So m must have proposed to w, which is a contradiction. So m must be married at the termination of the algorithm.

Proof of stable matching

Proposition:

The described algorithm produces a stable matching.

Proof:

- Assume towards contradiction that the algorithm produces an unstable matching for an instance of the stable marriage problem.
- Then there exists a pair, m, w', who are not matched by the algorithm, such that m prefers w' to his assigned partner w, and w' prefers m to her assigned partner m'.
- Then m proposed to w' before he proposed to w, since w' is before w on his list.
- But a woman can only reject a man if she receives a proposal from a man she prefers.
- So if a woman rejects a man, she prefers her final marriage partner to the rejected man. So w' prefers m' to m, which is a contradiction. So the G-S algorithm produces a stable matching.

Gale Shapley Algorithm - Parallelized

A parallel version of the Gale Shapley algorithm.

Algorithm 2 A parallelized Gale - Shapley Algorithm for stable matching

```
if master process then
    Send preference lists to male and female processes
    Listen for termination messages from female processes
    if ∄ women that are not engaged then
        Terminate the algorithm
    end if
else if male process then
    Listen for incoming messages
    if message is from master then
        Receive preference list
    else if message is from female process then
        Propose to the next preferred woman
    end if
    Propose to the first preferred woman
else if female process then
    Listen for incoming messages
    if message is from master then
        Receive preference list
    else if message is from male process then
        if Not engaged then
            Accept proposal
        else if check if received proposal is better than current engagement
then
            Accept received proposal
        else
            Retain current engagement
        end if
        Send signal to master that engagement is successful
    end if
end if
```

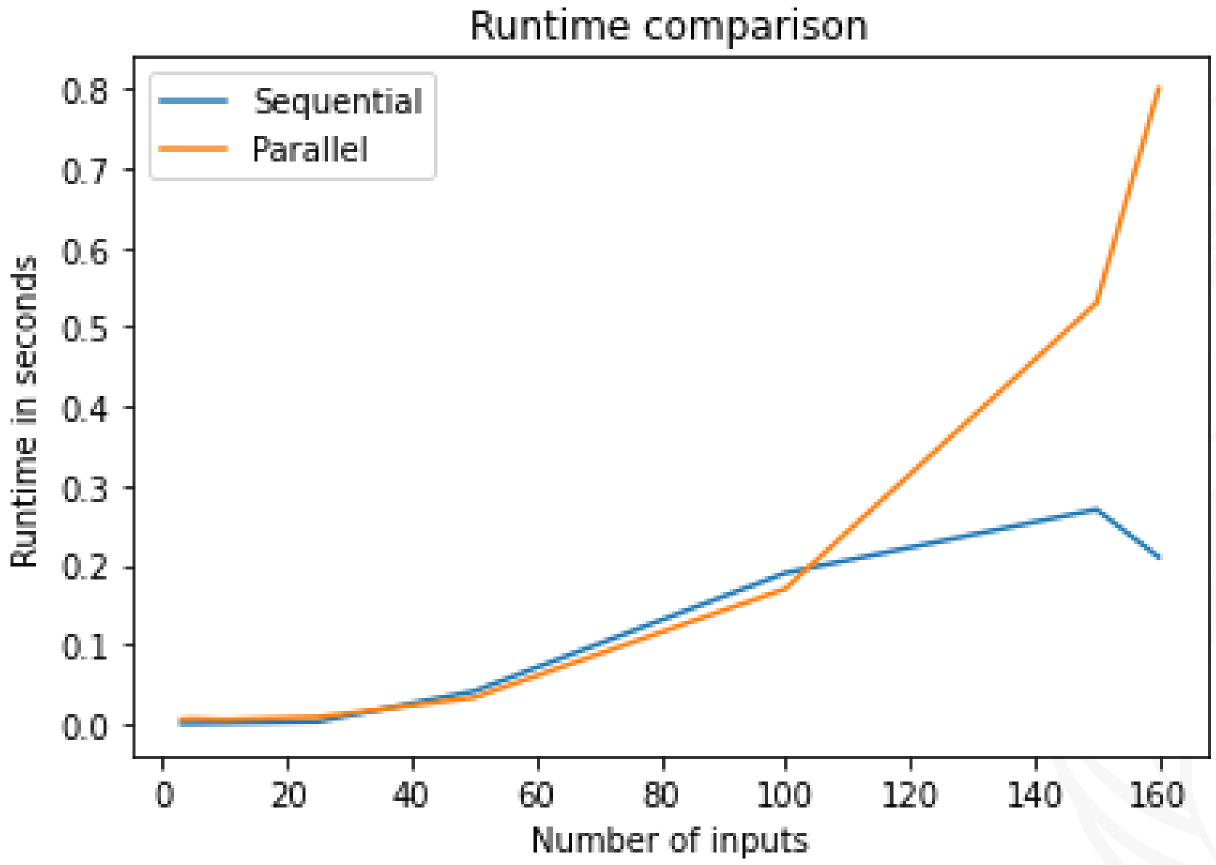
Gale Shapley Algorithm - Parallelized

- There are N men and N women.
- A master-worker approach is followed, there are 2N workers and 1 master, hence, total processes in 2N+1.
- The master sends out the preference lists and termination messages to the male and female processes and listens for termination messages from the female processes.
- The male processes send out proposal messages to the female processes and listen for termination messages from the master or rejection messages from the female processes.
- The female processes listen for preference lists from the master or proposal messages from the male processes and send out termination messages to the master indicating engagement or rejection messages to the male processes.

Runtime comparison

N	3	5	10	25	50	100	150	160	200
sequential	0.00035	0.00037	0.0009	0.0029	**0.041**	**0.19**	0.27	0.21	0.14
parallel	0.0056	0.0064	0.0052	0.0086	**0.033**	**0.17**	0.53	0.80	OOM

Runtime comparison



Learnings

- This is a really naïve implementation that does not provide considerable speedup.
- The amount of work done in communicating the data between the master and the workers is heavier than the actual computation.
- The computations in the sequential code are just unit time computation, except for a few like finding the indices.
- The parallel code, however, spends more time in communicating the data along with the unit time computations.
- The runtime also depends on the preference lists of the men and the women.
- The worst-case runtime is $O(N^2)$.

Story so far…

- The speedup provided by the parallelization is not significant.
- Infact, the runtimes for the parallel algorithm gets worse as the number of processes increase.
- Since, the communication is too expensive when compared to the amount of work to be done, the stable matching problem is best solved sequentially.

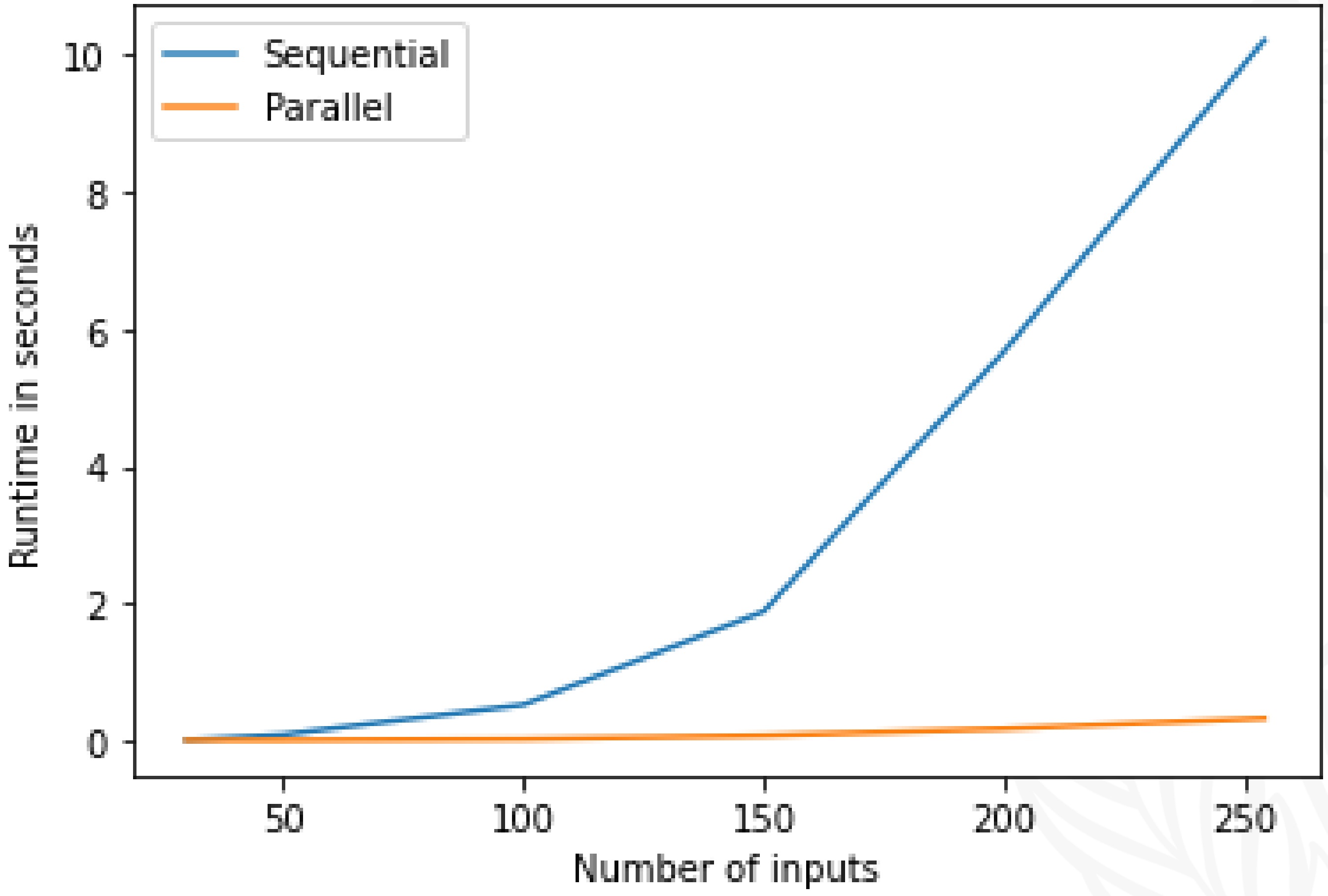
Ordering inputs

- The runtime also depends on the order of the preference lists.
- When the lists are ordered randomly, the sequential code outperforms the parallel implementation.
- A pattern seems to emerge when there is some inherent ordering to the preference lists of both the men and women.
- This is especially true in social networks, where the graph nodes are connected based on the similarities and/or differences between them.

Ordering inputs – Worst case scenario

N	30	50	100	150	200	254
sequential	0.011	0.088	0.524	1.886	5.683	10.216
parallel	0.010	0.010	0.026	0.076	0.175	0.327

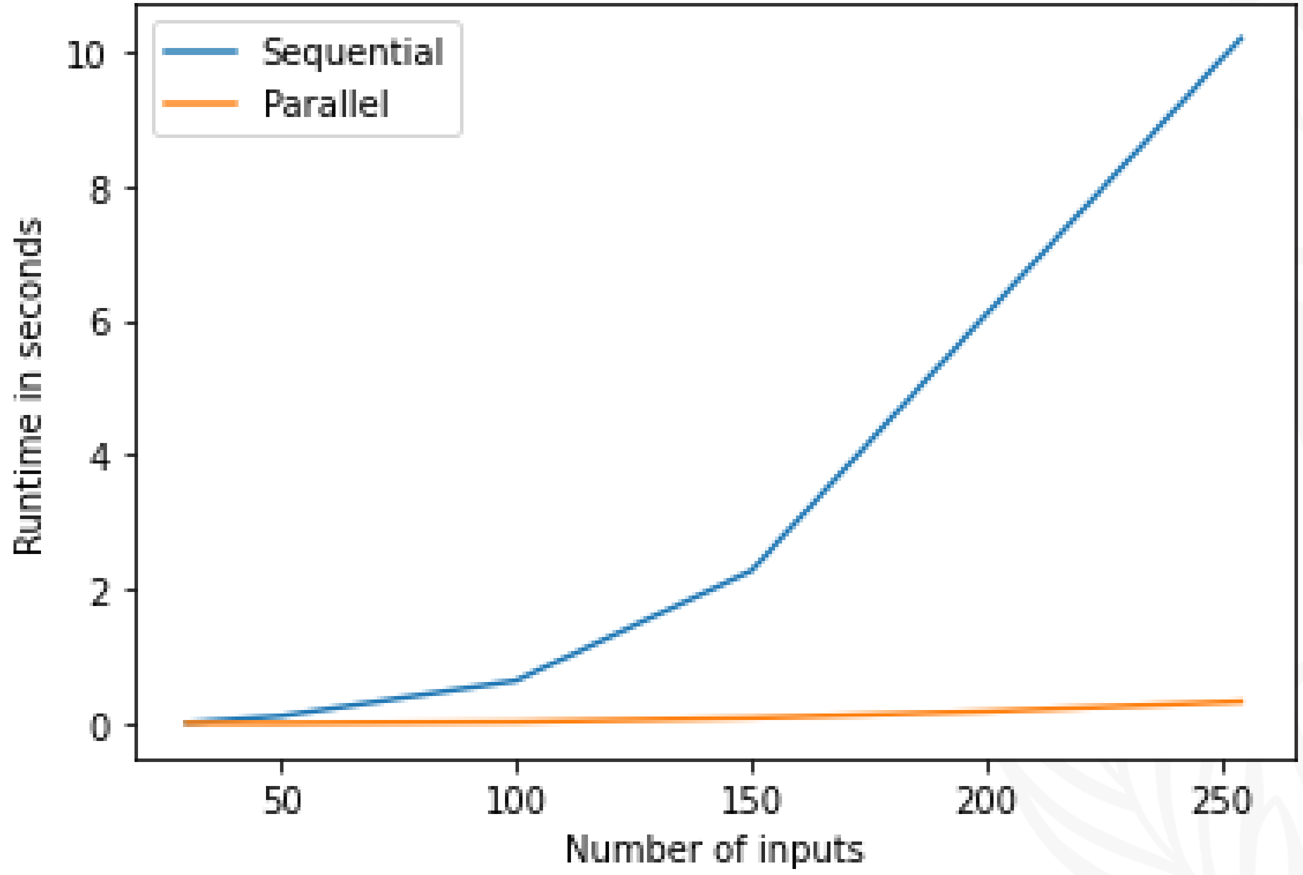
Ordering inputs – Worst case scenario



Ordering inputs – Half Rotated

N	30	50	100	150	200	254
sequential	0.012	0.116	0.64	2.271	6.088	10.185
parallel	0.007	0.006	0.026	0.086	0.185	0.332

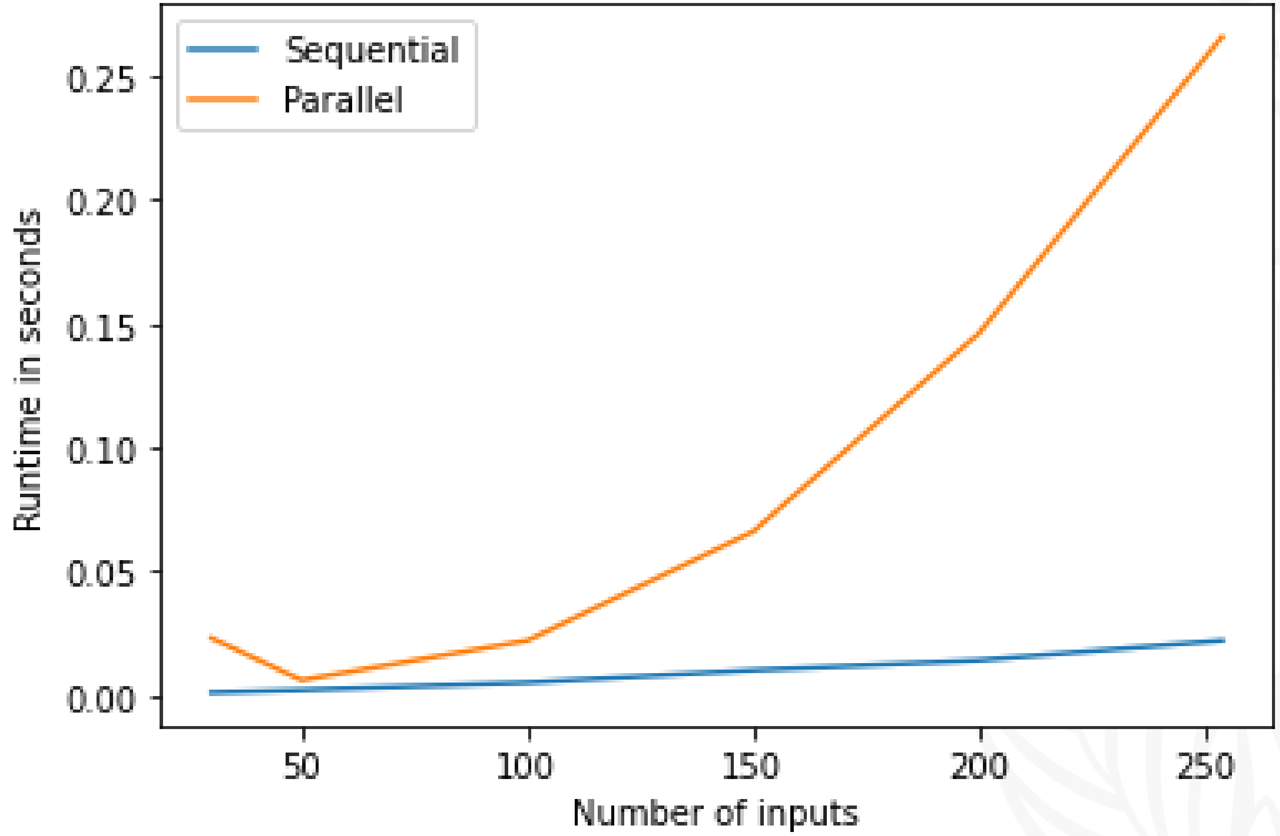
Ordering inputs – Half Rotated



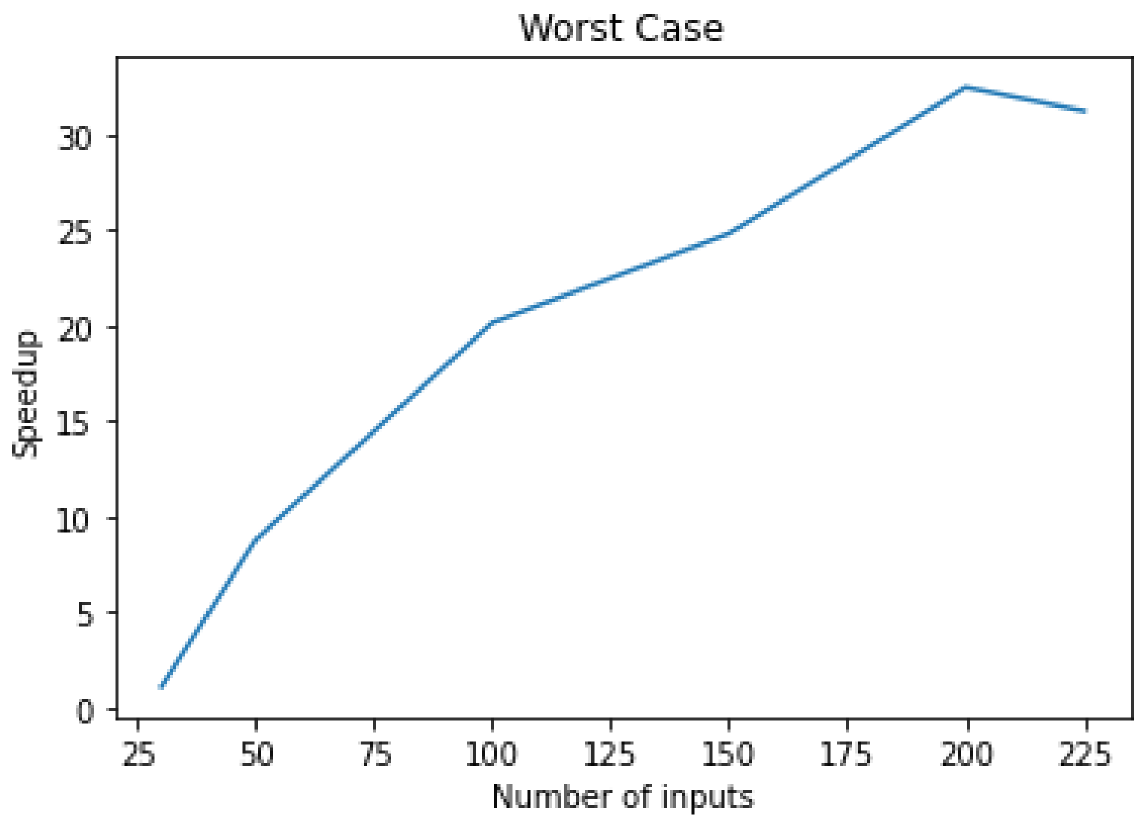
Ordering inputs – Shifted each step

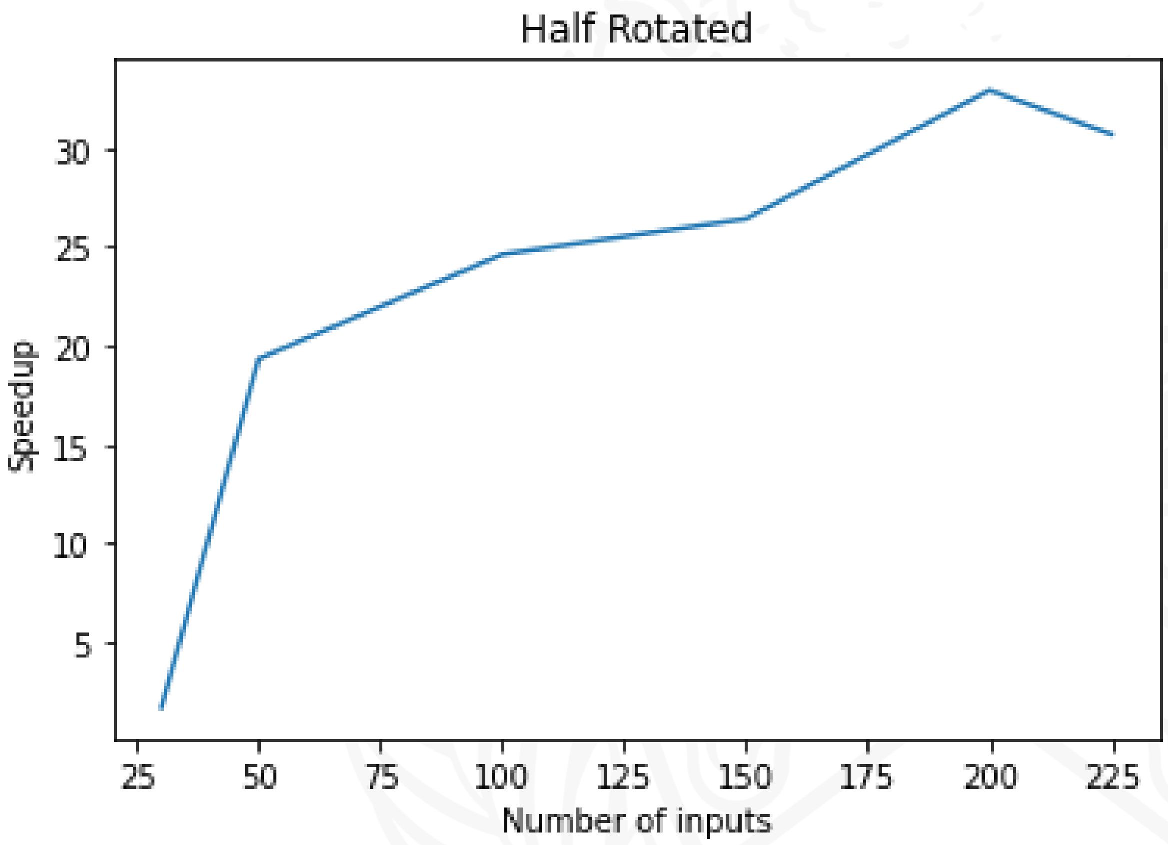
N	30	50	100	150	200	254
sequential	0.001	0.002	0.005	0.01	0.014	0.022
parallel	0.023	0.006	0.022	0.066	0.146	0.266

Ordering inputs – Shifted each step



Speedup


Worst Case
Speedup
0
5
10
15
20
25
30
25
50
75
100
125
150
175
200
225
Number of inputs


Half Rotated
Speedup
5
10
15
20
25
30
25
50
75
100
125
150
175
200
225
Number of inputs

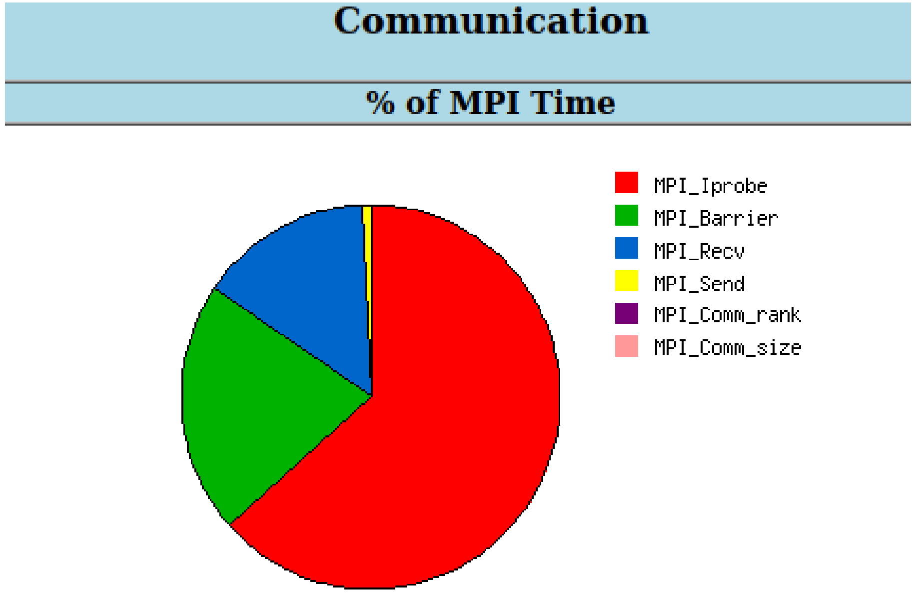
Profiling results - Sequential

```
# command   : ./gs_seq sample_inputs.txt
# start     : Thu May 12 09:35:11 2022   host      : naveen-VirtualB
# stop      : Thu May 12 09:35:11 2022   wallclock : 0.14
# mpi_tasks : 1 on 1 nodes               %comm     : 0.00
# mem [GB]  : 0.01                       gflop/sec : 0.00
#
#           :       [total]        <avg>          min          max
# wallclock :          0.14         0.14         0.14         0.14
# MPI       :          0.00         0.00         0.00         0.00
# %wall     :
#   MPI     :                       0.00         0.00         0.00
# #calls    :
#   MPI     :             3            3            3            3
# mem [GB]  :          0.01         0.01         0.01         0.01
#
###################################################################
```

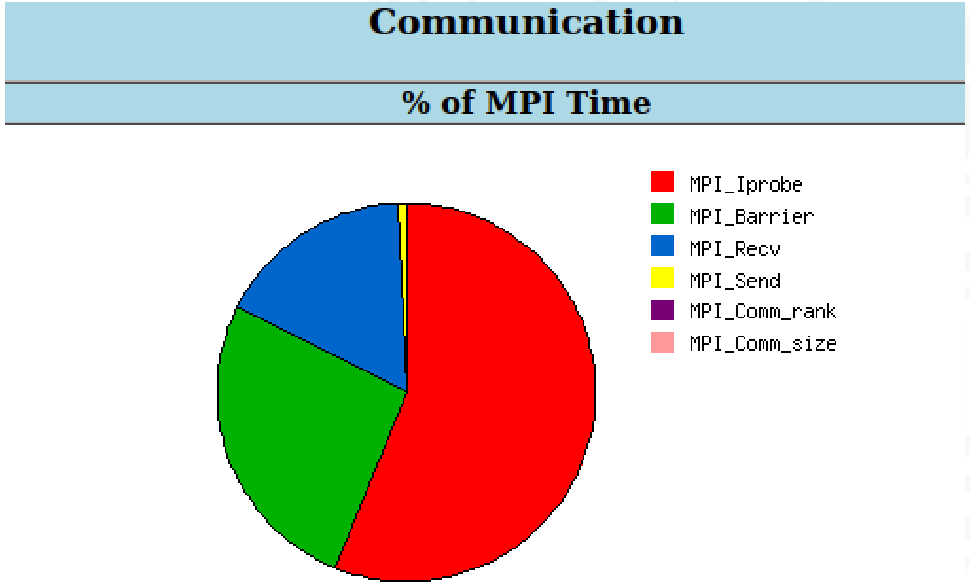
Profiling results - Parallel

```
##IPMv2.0.6########################################################
#
# command   : ./gs_parallel sample_inputs.txt
# start     : Thu May 12 09:41:44 2022   host      : naveen-VirtualB
# stop      : Thu May 12 09:41:55 2022   wallclock : 10.62
# mpi_tasks : 51 on 1 nodes              %comm     : 56.07
# mem [GB]  : 0.80                       gflop/sec : 0.00
#
#           :       [total]        <avg>          min          max
# wallclock :        523.99        10.27         9.99        10.62
# MPI       :        293.82         5.76         5.18         6.22
# %wall     :
#   MPI     :                      56.08        49.84        60.96
# #calls    :
#   MPI     :       2407246        47200        28510        59469
# mem [GB]  :          0.80         0.02         0.01         0.02
#
###################################################################
```

What makes the parallel implementation slower?



Random Inputs



Ordered Inputs

Conclusion

- The runtime of the algorithm greatly depends on the order of the inputs.
- There is a significant speedup provided by the parallelization when there is some inherent ordering in the inputs.
- This is true in case of real world matching, but at the same time random inputs are more likely.
- During randomized inputs, the algorithm spends more time in communicating the data between the processors which is expensive when compared to sequential iteration which renders the parallel implementation ineffective in such scenarios.