High Performance Data Compilation for Sports Analytics

Tyler Gourley

Outline

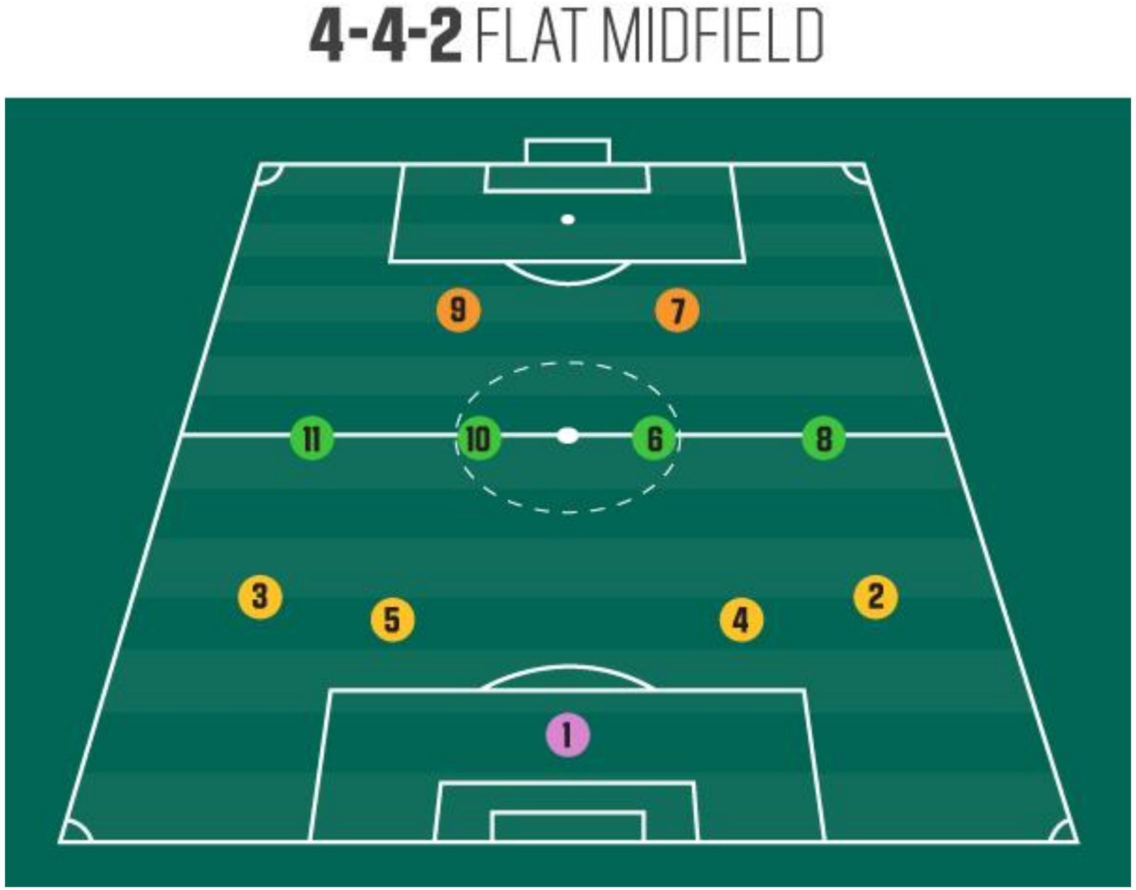
- Background
- Optimization I (Set Intersection)
- Optimization II (Parallelization)
- Results

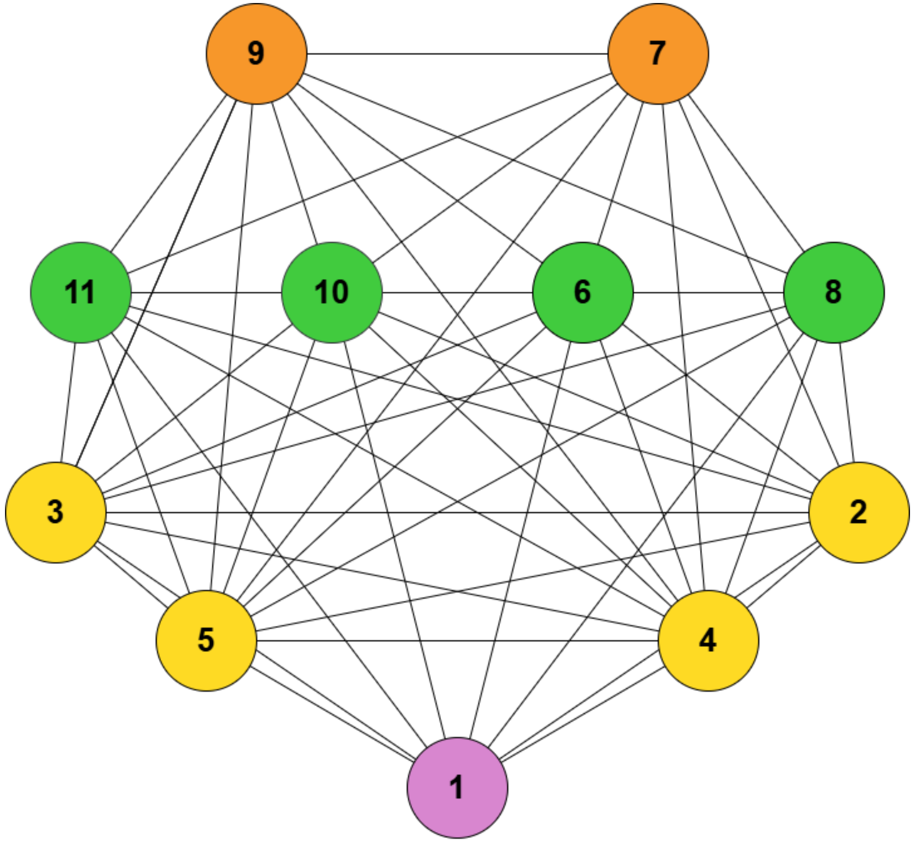
The Question...

Can we predict the outcome of soccer matches based on team chemistry?

Graphical Representation

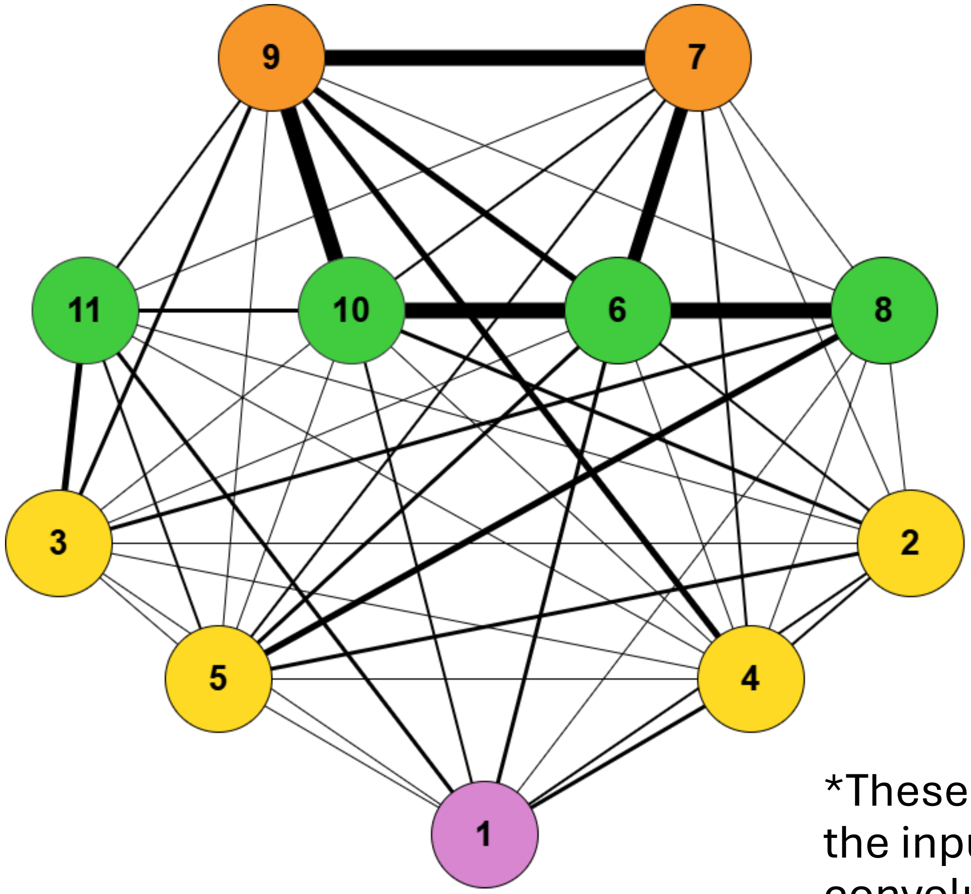
Each match will have 2 Complete, Weighted Graphs (one for each team)





Numerical Representation

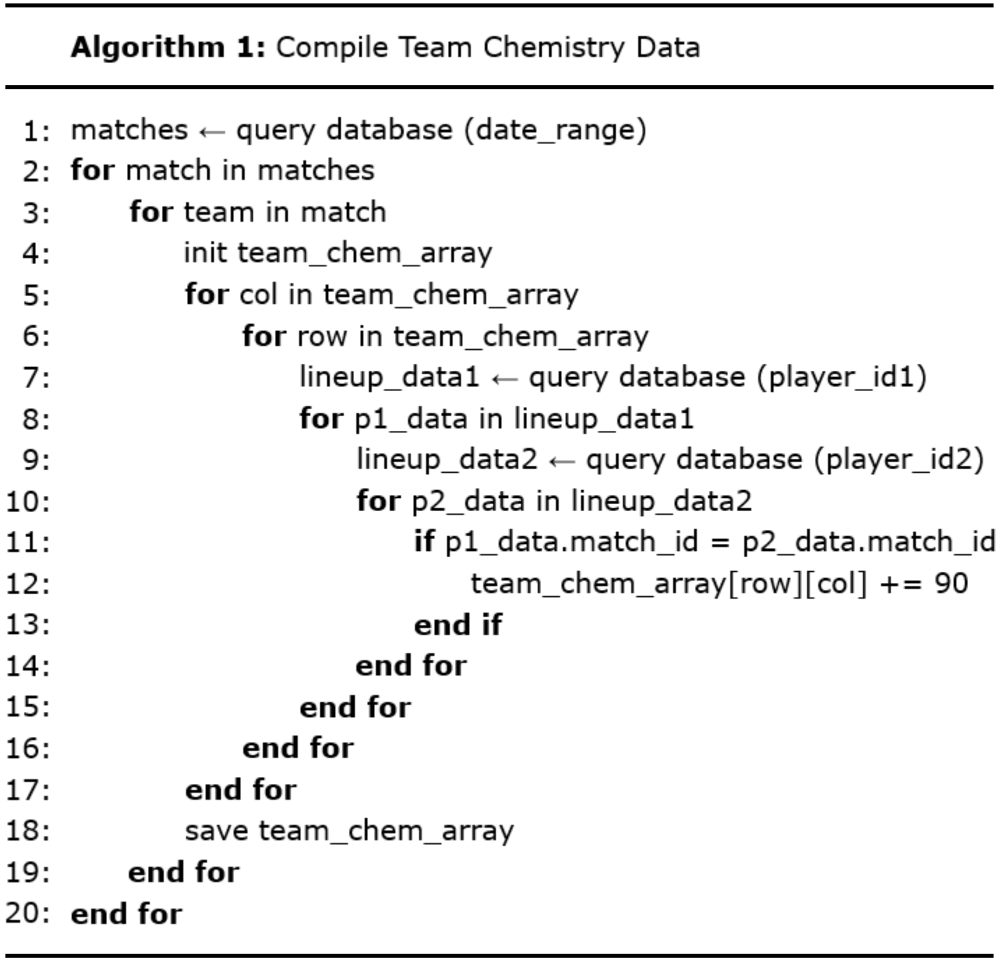
Convert the Weighted Graphs to Numpy Arrays



*These arrays will be the input to our convolutional neural net

0	900	810	900	810	720	810	900	990	900	630
0	0	810	810	810	720	630	810	720	540	360
0	0	0	720	630	990	900	720	810	810	630
0	0	0	0	810	990	900	810	1440	810	630
0	0	0	0	0	990	900	1530	990	810	630
0	0	0	0	0	0	2250	2160	1350	2250	630
0	0	0	0	0	0	0	900	2160	720	630
0	0	0	0	0	0	0	0	990	810	630
0	0	0	0	0	0	0	0	0	2070	630
0	0	0	0	0	0	0	0	0	0	630
0	0	0	0	0	0	0	0	0	0	0

Data Compilation Algorithm



Algorithm 1: Compile Team Chemistry Data

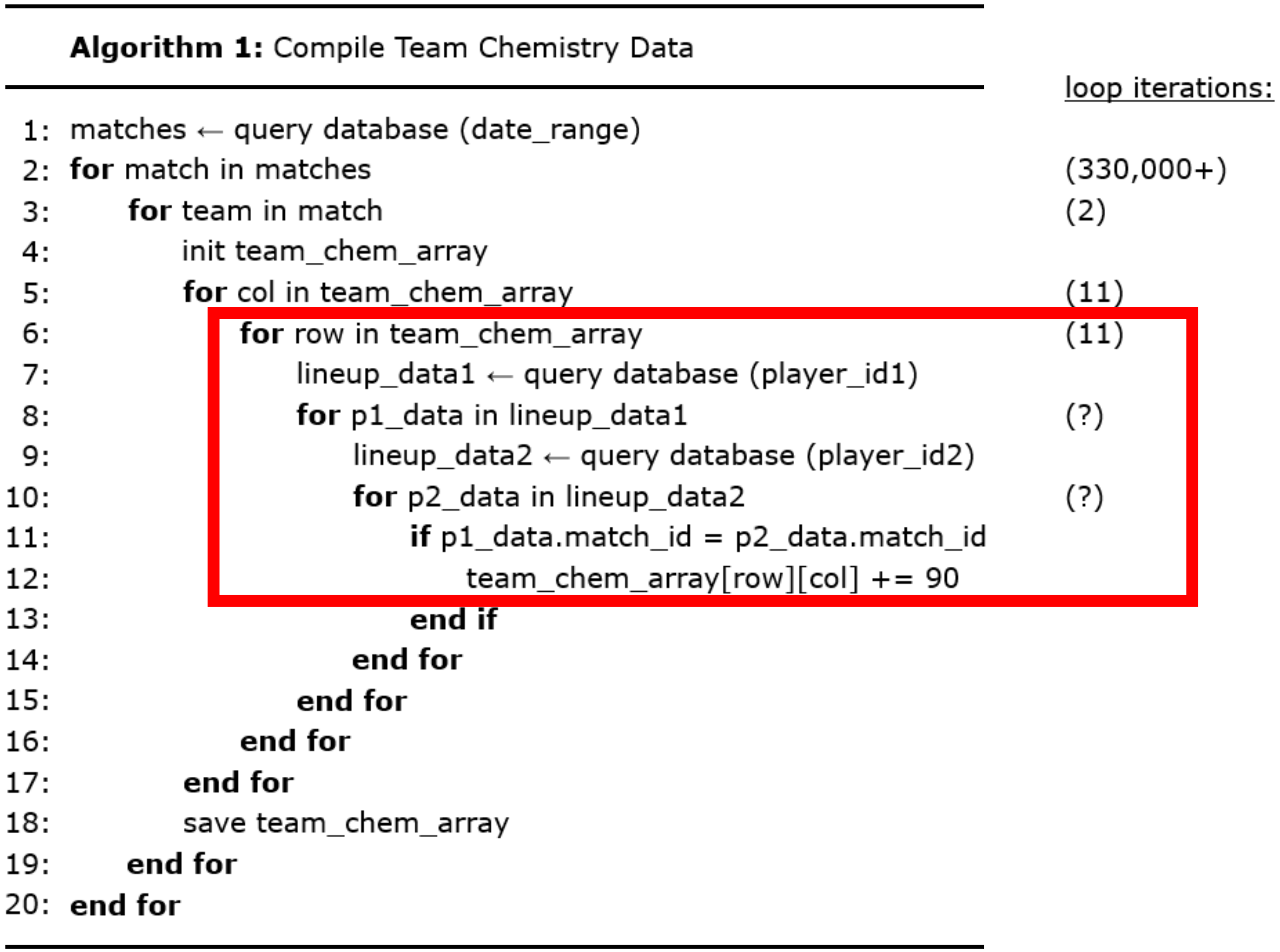
```
                                                          loop iterations:

matches ← query database (date_range)
for match in matches                                      (330,000+)
    for team in match                                     (2)
        init team_chem_array
        for col in team_chem_array                        (11)
            for row in team_chem_array                    (11)
                lineup_data1 ← query database (player_id1)
                for p1_data in lineup_data1               (?)
                    lineup_data2 ← query database (player_id2)
                    for p2_data in lineup_data2           (?)
                        if p1_data.match_id = p2_data.match_id
                            team_chem_array[row][col] += 90
                        end if
                    end for
                end for
            end for
        end for
        save team_chem_array
    end for
end for
```

Our database contains the playing time data for over 330,000 soccer matches

We iterate over every cell in the upper triangular array (representing the 55 player combinations possible)

For every cell, we need to query the database again to determine how many game minutes the players played together

Data Compilation Algorithm



Algorithm 1: Compile Team Chemistry Data

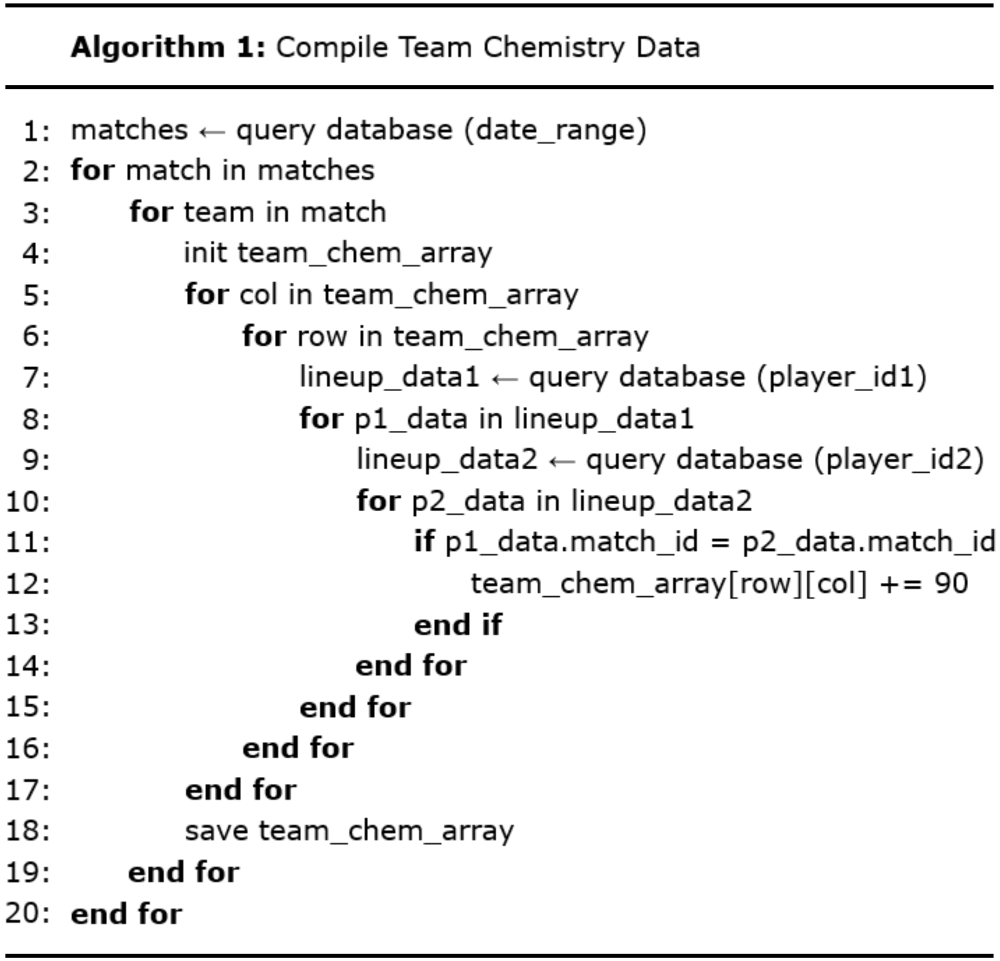
```
                                                          loop iterations:
matches ← query database (date_range)
for match in matches                                      (330,000+)
    for team in match                                     (2)
        init team_chem_array
        for col in team_chem_array                        (11)
            for row in team_chem_array                    (11)
                lineup_data1 ← query database (player_id1)
                for p1_data in lineup_data1               (?)
                    lineup_data2 ← query database (player_id2)
                    for p2_data in lineup_data2           (?)
                        if p1_data.match_id = p2_data.match_id
                            team_chem_array[row][col] += 90
                        end if
                    end for
                end for
            end for
        end for
        save team_chem_array
    end for
end for
```

The bulk of the complexity comes from this inner 2 for loops

The results from these queries can be very large, and provide redundant information

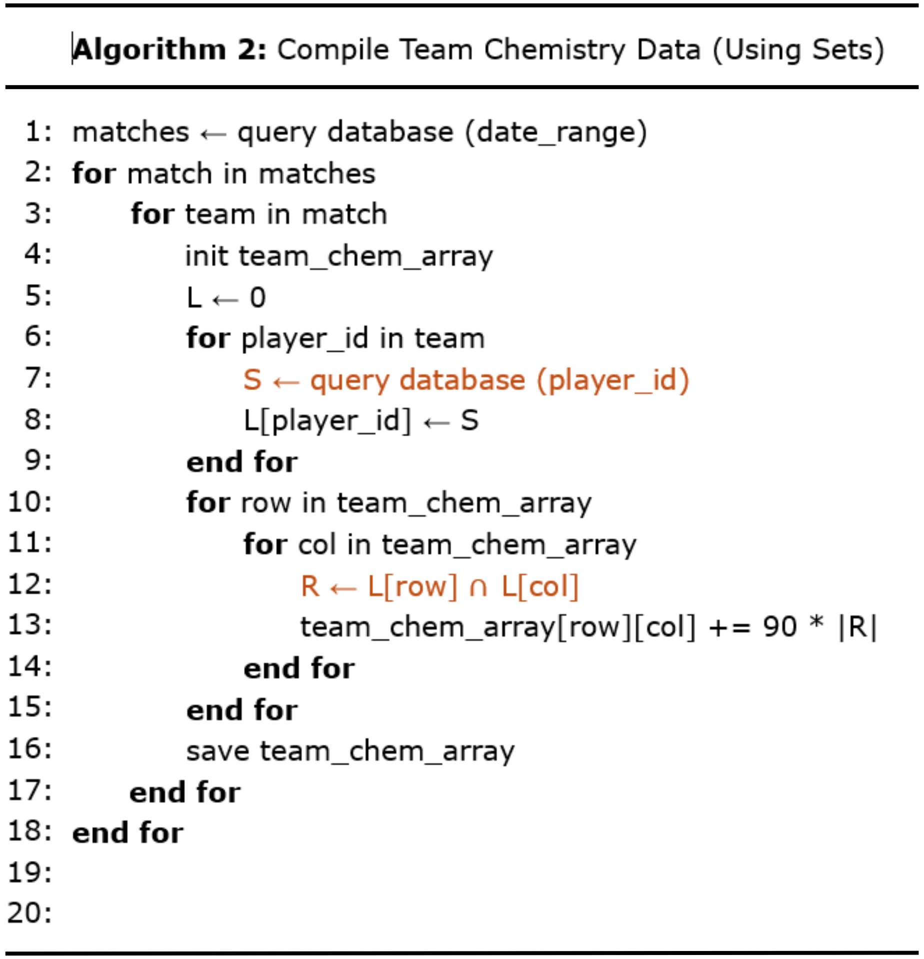
We can do better

Algorithm Optimization Using Sets



Algorithm 1: Compile Team Chemistry Data

```
                                                                 loop iterations:
matches ← query database (date_range)
for match in matches                                             (330,000+)
    for team in match                                            (2)
        init team_chem_array
        for col in team_chem_array                               (11)
            for row in team_chem_array                           (11)
                lineup_data1 ← query database (player_id1)
                for p1_data in lineup_data1                      (?)
                    lineup_data2 ← query database (player_id2)
                    for p2_data in lineup_data2                  (?)
                        if p1_data.match_id = p2_data.match_id
                            team_chem_array[row][col] += 90
                        end if
                    end for
                end for
            end for
        end for
        save team_chem_array
    end for
end for
```



Algorithm 2: Compile Team Chemistry Data (Using Sets)

```
                                                                 loop iterations:
matches ← query database (date_range)
for match in matches                                             (330,000+)
    for team in match                                            (2)
        init team_chem_array
        L ← 0
        for player_id in team                                    (11)
            S ← query database (player_id)
            L[player_id] ← S
        end for
        for row in team_chem_array                               (11)
            for col in team_chem_array                           (11)
                R ← L[row] ∩ L[col]
                team_chem_array[row][col] += 90 * |R|
            end for
        end for
        save team_chem_array
    end for
end for
```

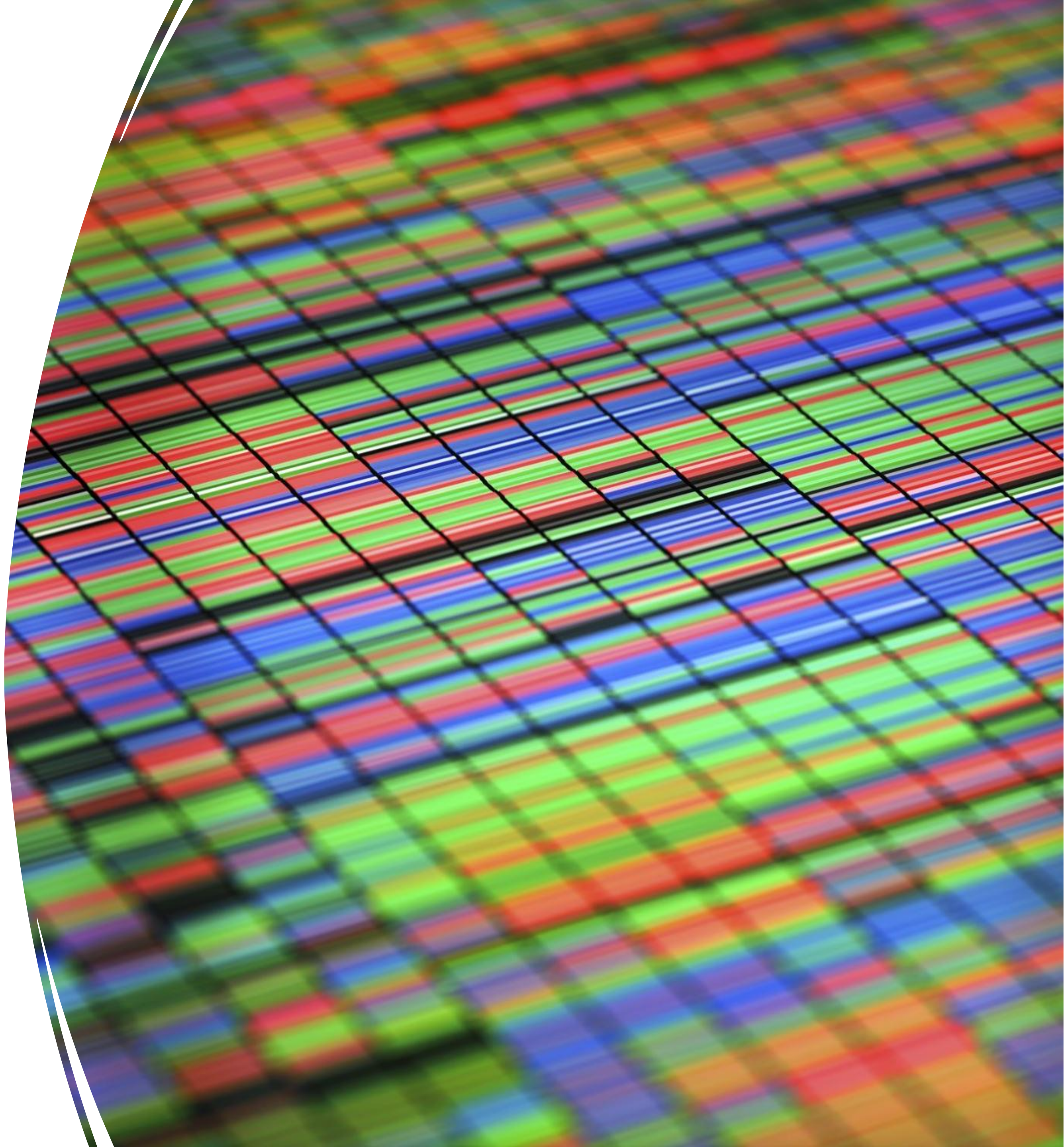
Parallelization

Python Processes (multiprocessing package)

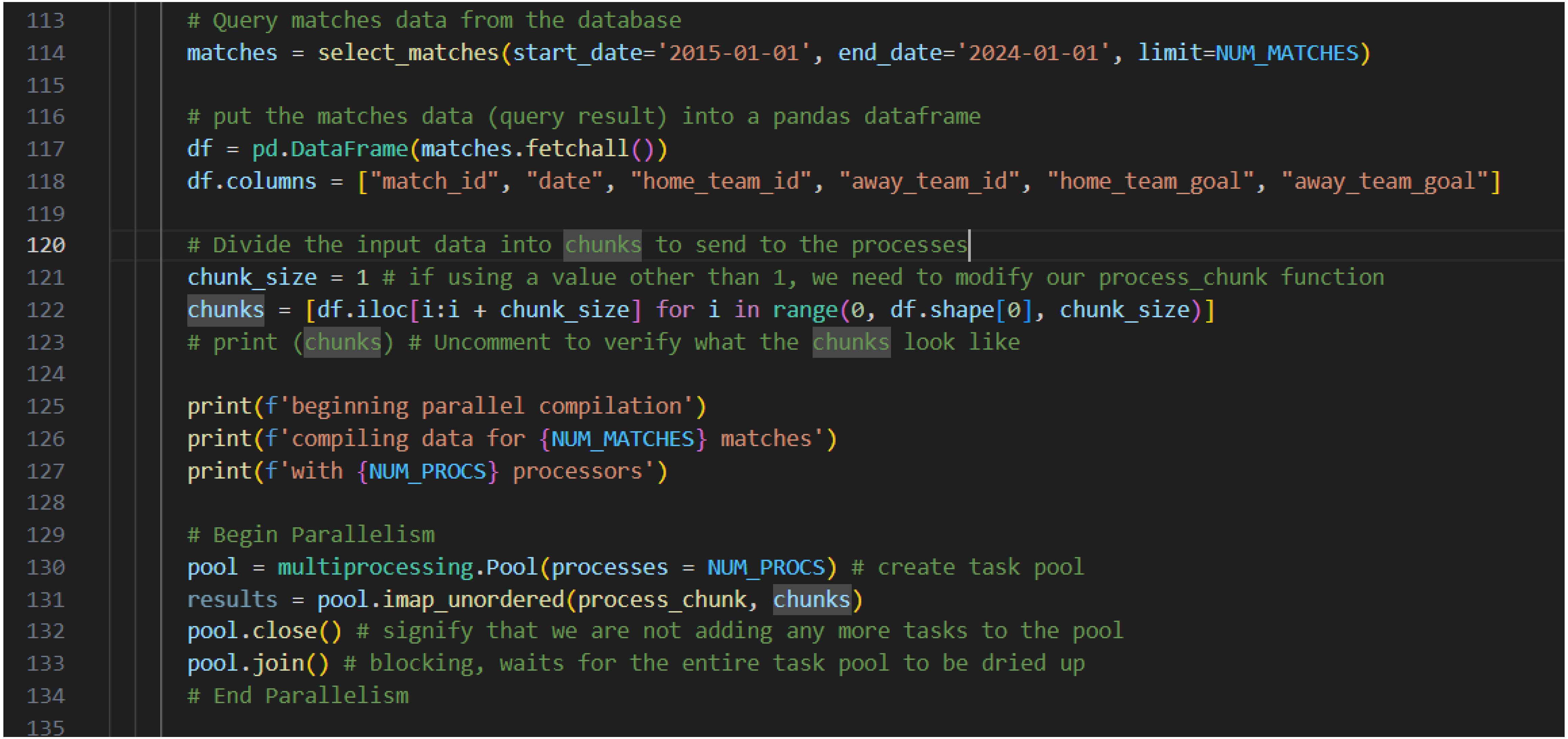
Chunk the data into Tasks

Create a Task Pool

Add tasks to the pool



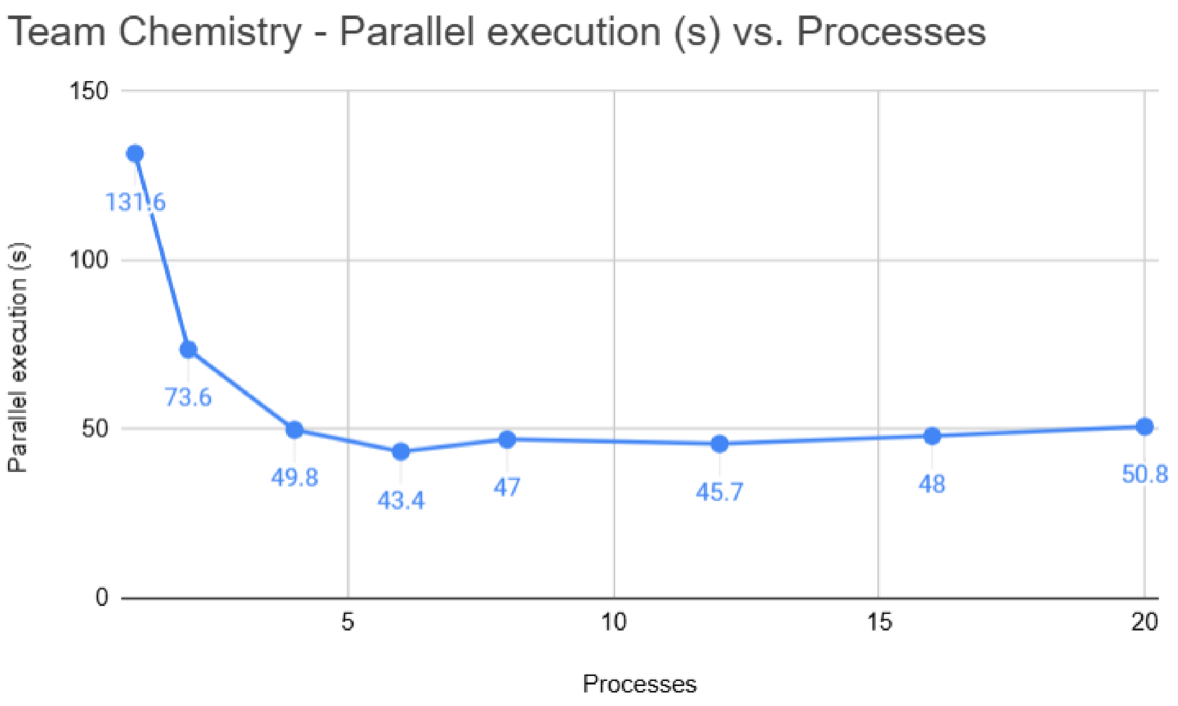
Parallel Implementation Code

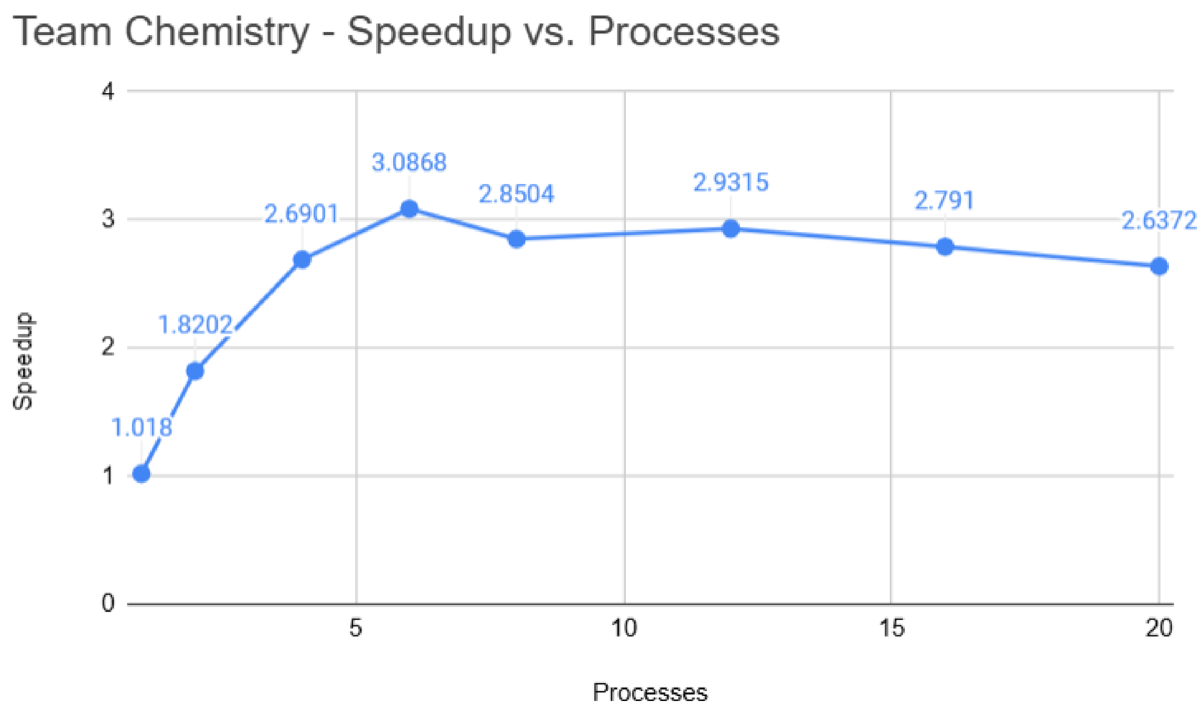


```python
        # Query matches data from the database
        matches = select_matches(start_date='2015-01-01', end_date='2024-01-01', limit=NUM_MATCHES)

        # put the matches data (query result) into a pandas dataframe
        df = pd.DataFrame(matches.fetchall())
        df.columns = ["match_id", "date", "home_team_id", "away_team_id", "home_team_goal", "away_team_goal"]

        # Divide the input data into chunks to send to the processes
        chunk_size = 1 # if using a value other than 1, we need to modify our process_chunk function
        chunks = [df.iloc[i:i + chunk_size] for i in range(0, df.shape[0], chunk_size)]
        # print (chunks) # Uncomment to verify what the chunks look like

        print(f'beginning parallel compilation')
        print(f'compiling data for {NUM_MATCHES} matches')
        print(f'with {NUM_PROCS} processors')

        # Begin Parallelism
        pool = multiprocessing.Pool(processes = NUM_PROCS) # create task pool
        results = pool.imap_unordered(process_chunk, chunks)
        pool.close() # signify that we are not adding any more tasks to the pool
        pool.join() # blocking, waits for the entire task pool to be dried up
        # End Parallelism

```

Preliminary Results

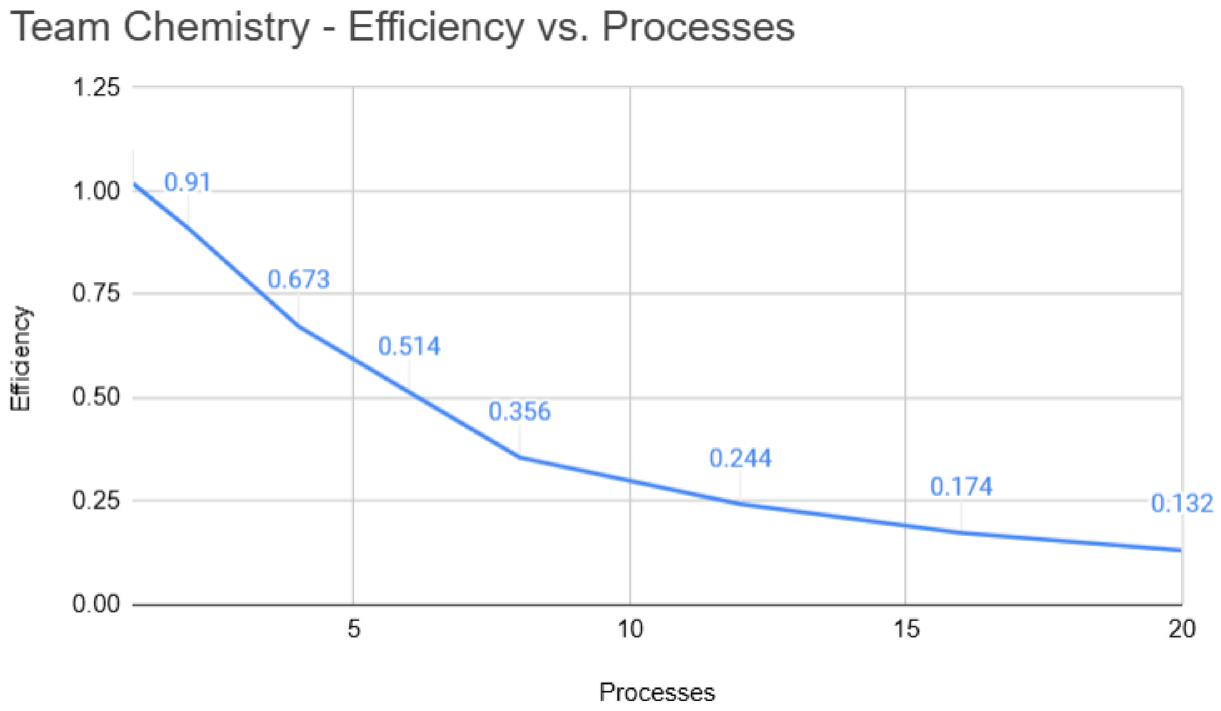
The performance metrics were run on a representative 200 match portion of the overall database (330,000+ matches)

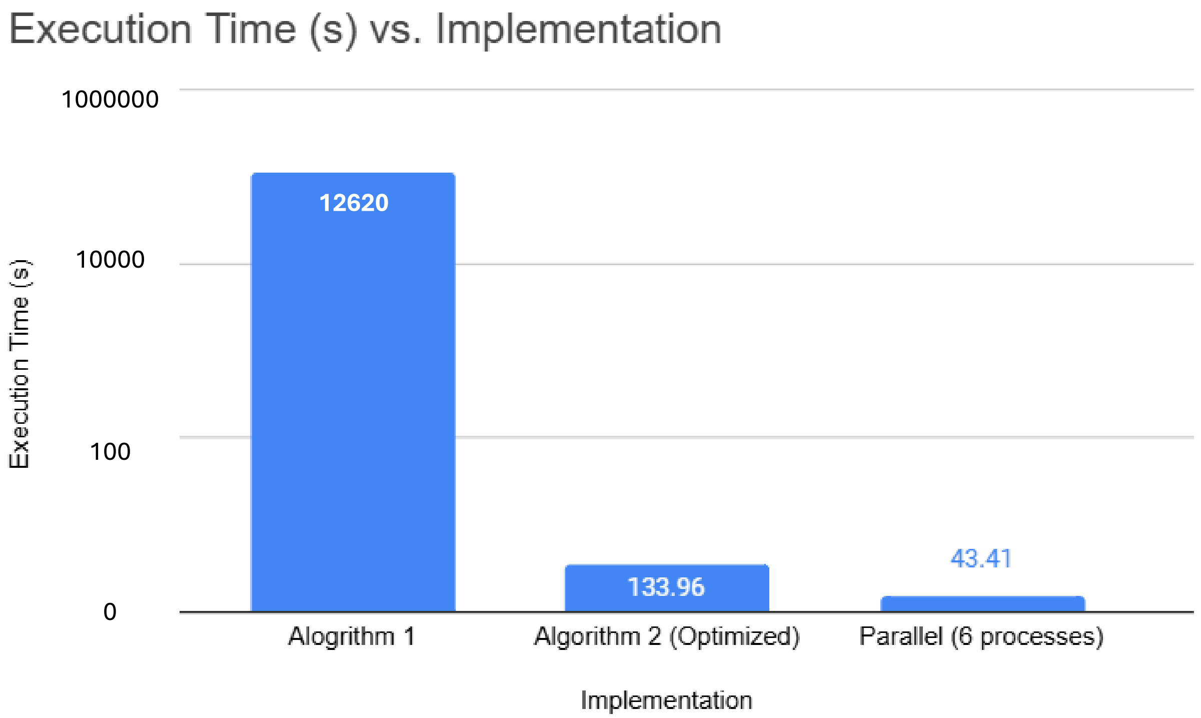




Preliminary Results (Cont.)

We achieved a max Speedup of 3.086 when using 6 processes (about 300x faster when compared to original code)





Questions?