In [ ]:
```
from subprocess import call
import numpy as np
import matplotlib.pyplot as plt

from collections import namedtuple
```

In [ ]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.m
ount("/content/drive", force_remount=True).
```

In [ ]:
```
%cd /content/drive/MyDrive/parallel_chess_engine
```

```
/content/drive/MyDrive/parallel_chess_engine
```

In [ ]:
```
%pwd
```

Out[ ]:
```
'/content/drive/MyDrive/parallel_chess_engine'
```

In [ ]:
```
def simulation(progName, nGames=10, depth=4):


    fileName = "results.txt"
    #To store the timing values (means) for a specific depth
    oldMeanPerDepth = []
    newMeanPerDepth = []

    #To store the number of nodes visited from root until a specific depth
    oldNodeNumPerDepth = []
    newNodeNumPerDepth = []

    #To store the timing values (means) for all depths [THose that are going to be use
    oldGlobalMean = []
    newGlobalMean = []

    #To store the number of noes visited for all depths [THose that are going to be us
    oldNodeNum = []
    newNodeNum = []

     #the result
    stats = namedtuple("Stats", ["oldGlobalMean", "newGlobalMean", "oldNodeNum", "newN

    #generate all the possible depths
    depths = range(1, depth + 1)

    for h in depths:
        for g in range(nGames):
            retCode = call([progName, str(h)])
            with open(fileName, "r") as f: #filename is the file path to the file gene
                #first two lines in the file represent the number of nodes visited for
                oldNodeNumPerDepth.append(float(f.readline().strip(" ")))
                newNodeNumPerDepth.append(float(f.readline().strip(" ")))

                #we calculate for each depth, for each game the mean of the time taken
                old = list(map(float, f.readline().strip().split()))
                new = list(map(float, f.readline().strip().split()))
                oldMeanPerDepth.append(sum(old) / len(old))
```

```
                newMeanPerDepth.append(sum(new) / len(new))

        #we assigne the infomation relative to each depth to the global lists
        oldGlobalMean.append(sum(oldMeanPerDepth) / len(oldMeanPerDepth))
        newGlobalMean.append(sum(newMeanPerDepth) / len(newMeanPerDepth))
        oldNodeNum.append(int(sum(oldNodeNumPerDepth) / len(oldNodeNumPerDepth)))
        newNodeNum.append(int(sum(newNodeNumPerDepth) / len(newNodeNumPerDepth)))
        oldMeanPerDepth  = []
        newMeanPerDepth = []
        oldNodeNumPerDepth = []
        newNodeNumPerDepth = []

    #assign everything to the named tuple
    stats.oldGlobalMean = oldGlobalMean
    stats.newGlobalMean = newGlobalMean
    stats.oldNodeNum = oldNodeNum
    stats.newNodeNum = newNodeNum
    return stats
```

In [ ]:
```
def plotting(stats, old_name, new_name, depth=4):

    # Generate all the possible depths
    depths = range(1, depth + 1)

    # Only for plotting purposes
    offset = 0.2
    figure = plt.subplot(121)
    figure.bar(np.array(depths) - offset, stats.oldGlobalMean[:depth], width=2 * offse
    figure.bar(np.array(depths) + offset, stats.newGlobalMean[:depth], width=2 * offse
    plt.xlabel("Depth")
    plt.ylabel("Execution time (s)")
    plt.legend([old_name, new_name])

    figure = plt.subplot(122)
    figure.bar(np.array(depths) - offset, stats.oldNodeNum[:depth], width=2 * offset,
    figure.bar(np.array(depths) + offset, stats.newNodeNum[:depth], width=2 * offset,
    plt.xlabel("Depth")
    plt.ylabel("Visited nodes")
    plt.legend([old_name, new_name])
    plt.show()
    plt.close()
```

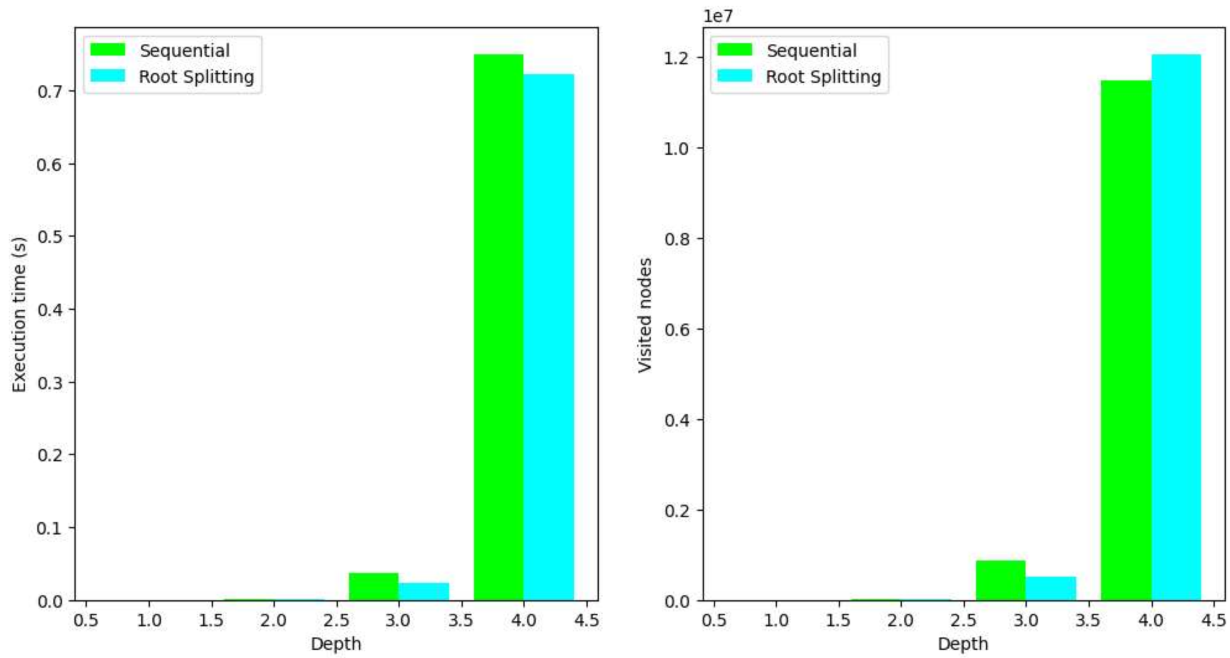
In [ ]:
```
!chmod +rx serial_vs_parallel.c
```

In [ ]:
```
!gcc -o serial_vs_parallel serial_vs_parallel.c
```

In [ ]:
```
stats = simulation("./serial_vs_parallel", nGames=3)

# Extracting the last execution times for Sequential and Parallel
sequential_execution_time = stats.oldGlobalMean[-1]
parallel_execution_time = stats.newGlobalMean[-1]

print(f"Sequential Execution Time: {sequential_execution_time} seconds")
print(f"Parallel Execution Time: {parallel_execution_time} seconds")

# Extracting the last visited nodes for Sequential and Parallel
sequential_visited_nodes = stats.oldNodeNum[-1]
parallel_visited_nodes = stats.newNodeNum[-1]
```

```
print(f"Sequential Visited Nodes: {sequential_visited_nodes}")
print(f"Parallel Visited Nodes: {parallel_visited_nodes}")

%matplotlib inline
import matplotlib
matplotlib.rcParams['figure.figsize'] = (12, 6)

plotting(stats, "Sequential", "Root Splitting")
```

```
Sequential Execution Time: 0.7502195066666668 seconds
Parallel Execution Time: 0.72327698 seconds
Sequential Visited Nodes: 11472930
Parallel Visited Nodes: 12064469
```



In [ ]:

```
stats.oldGlobalMean[-1]/stats.newGlobalMean[-1]
```

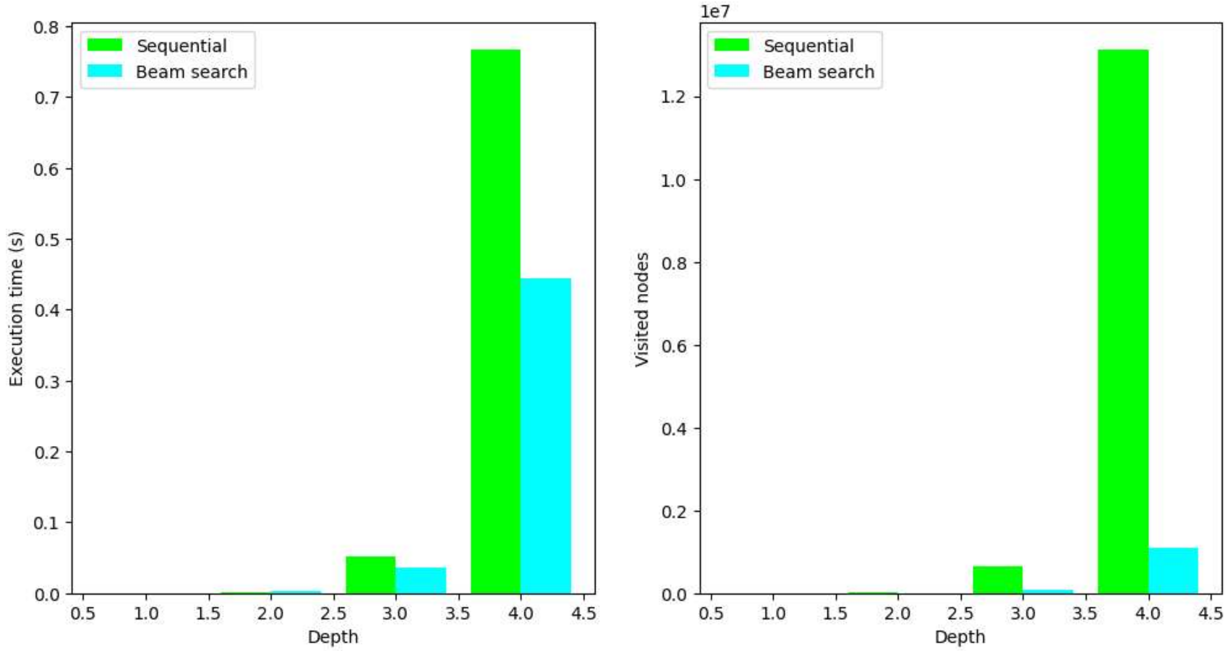
Out[ ]:

```
1.037250634835173
```

In [ ]:

```
!chmod +rx serial_vs_beam.c
!gcc -o serial_vs_beam serial_vs_beam.c

stats = simulation("./serial_vs_beam", nGames=3)

# Extracting the last execution times for Sequential and Root Splitting
sequential_execution_time = stats.oldGlobalMean[-1]
root_splitting_execution_time = stats.newGlobalMean[-1]

print(f"Sequential Execution Time: {sequential_execution_time} seconds")
print(f"Root Splitting Execution Time: {root_splitting_execution_time} seconds")

# Extracting the last visited nodes for Sequential and Root Splitting
sequential_visited_nodes = stats.oldNodeNum[-1]
root_splitting_visited_nodes = stats.newNodeNum[-1]

print(f"Sequential Visited Nodes: {sequential_visited_nodes}")
print(f"Root Splitting Visited Nodes: {root_splitting_visited_nodes}")

plotting(stats, "Sequential", "Beam search")
```

```
Sequential Execution Time: 0.7670814800000002 seconds
Root Splitting Execution Time: 0.44373152666666665 seconds
Sequential Visited Nodes: 13128497
Root Splitting Visited Nodes: 1082572
```



In [ ]:
```
stats.oldGlobalMean[-1]/stats.newGlobalMean[-1]
```

Out[ ]: 1.7287062872506593