Parallel Prim's Algorithm for Minimum Spanning Tree (MST)

```
import dask
import dask.distributed as dd
import networkx as nx
import numpy as np
import time

def find_min_vertex(graph, selected, mst, num_vertices):
    min_weight = float('inf')
    min_index = None

    # Find the vertex with the minimum weight that is not yet selected
    for i in range(num_vertices):
        if not selected[i] and mst[i] is not None and graph[i][mst[i]] < min_weight:
            min_weight = graph[i][mst[i]]
            min_index = i

    return min_index, min_weight

def update_mst(graph, selected, mst, min_index, num_vertices):
    # Mark the selected vertex as visited
    selected[min_index] = True

    # Update the minimum spanning tree
    for i in range(num_vertices):
        if not selected[i] and graph[i][min_index] < float('inf'):
            if mst[i] is None or graph[i][min_index] < graph[i][mst[i]]:
                mst[i] = min_index

def generate_random_graph(num_vertices, max_weight):
    graph = np.random.randint(1, max_weight, size=(num_vertices, num_vertices))
```

```
    np.fill_diagonal(graph, 0)
    return graph

def parallel_prim(graph):
    num_vertices = len(graph)

    # Initialize the minimum spanning tree with the first vertex
    mst = [None] * num_vertices
    mst[0] = 0

    # Initialize the list to keep track of selected vertices
    selected = [False] * num_vertices

    start_time = time.time()  # Start measuring the runtime

    client = dd.Client()  # Create a Dask client

    graph_future = client.scatter(graph)  # Scatter the graph data to workers

    while not all(selected):
        min_index_future = client.submit(find_min_vertex, graph_future, selected, mst, num_vertices)
        min_index, min_weight = min_index_future.result()

        selected[min_index] = True

        update_mst(graph, selected, mst, min_index, num_vertices)

    client.close()  # Close the Dask client

    end_time = time.time()  # Stop measuring the runtime
    runtime = end_time - start_time
```

```
    return mst, runtime


if __name__ == '__main__':
    num_vertices = 1000  # Number of vertices in the random graph
    max_weight = 100  # Maximum weight for edge weights
    # Generate a random graph
    graph = generate_random_graph(num_vertices, max_weight)
    mst, runtime = parallel_prim(graph)
    print("Minimum Spanning Tree:", mst)
    print("Runtime:", runtime, "seconds")
```

Output:

Minimum Spanning Tree: [0, 114, 69, 20, 93, 190, 61, 181, 155, 54, 121, 41, 0, 42, 86, 123, 114, 32, 87, 117, 33, 20, 64, 64, 4, 105, 76, 64, 192, 12, 69, 99, 16, 69, 126, 13, 81, 149, 99, 155, 37, 164, 122, 133, 153, 149, 38, 57, 153, 63, 105, 75, 27, 13, 139, 43, 27, 148, 112, 33, 30, 37, 124, 7, 36, 185, 36, 151, 103, 11, 120, 22, 148, 147, 3, 85, 12, 89, 57, 100, 58, 74, 41, 178, 8, 160, 74, 170, 43, 142, 38, 111, 99, 130, 20, 115, 66, 184, 88, 100, 66, 135, 145, 66, 67, 135, 194, 18, 79, 107, 141, 11, 119, 41, 149, 142, 143, 113, 153, 139, 100, 94, 34, 12, 194, 83, 131, 128, 84, 38, 46, 81, 68, 177, 8, 122, 155, 143, 30, 120, 69, 36, 8, 82, 69, 186, 6, 58, 38, 54, 39, 126, 187, 31, 19, 149, 21, 20, 145, 111, 32, 75, 187, 142, 12, 121, 76, 111, 10, 27, 14, 4, 13, 46, 139, 82, 148, 141, 78, 126, 19, 13, 64, 133, 170, 45, 147, 17, 25, 117, 169, 152, 21, 12, 81, 79, 175, 112, 19, 124]

Runtime: 9.458676815032959 seconds