DDA LAB

Simran Kaur

311443

Exercise 9

Implementing Parallel Stochastic Gradient Descent

In [3]:
```
import matplotlib.pyplot as plt
```

The main task here is to divide data over each worker and they individually train the model over that data and finally returns the weights and biases which are then averaged and used as the weights to test the model on the test set. As at one point of time, each worker trains the model on the data shown to it, the accuracy would not be good and hence as the number of workers are increased accuracy decreases. Though the training would be fast which would result in a speedup as more and more workers are used.

In [ ]:
```
import torch
from torch.nn import Module
from torch import nn
import torchvision
import torchvision.transforms as transforms
from torch.utils.data import DataLoader, Dataset
from torch.utils.data import DistributedSampler
import torch.nn.functional as F
import matplotlib.pyplot as plt
import numpy as np
from mpi4py import MPI

# Setting MPI
comm = MPI.COMM_WORLD
rank = comm.Get_rank()
size = comm.Get_size()

# CNN
class Network(Module):
    def __init__(self):
        super(Network, self).__init__()
        self.conv1 = nn.Conv2d(1, 10, kernel_size = 5)
        self.pool1 = nn.MaxPool2d(kernel_size = 2, stride = 2)
        self.conv2 = nn.Conv2d(10, 30, kernel_size = 5)
        self.pool2 = nn.MaxPool2d(kernel_size = 2, stride = 2)
        self.drop1 = nn.Dropout(0.15)
```

```
        self.pool3 = nn.MaxPool2d(kernel_size = 2, stride = 2)
        self.fc1 = nn.Linear(120, 60)
        self.fc2 = nn.Linear(60, 10)
        self.soft = nn.LogSoftmax(dim = 1)

    def forward(self, x):
        y = F.relu(self.pool1(self.conv1(x)))
        y = F.relu(self.pool2(self.conv2(y)))
        y = self.pool3(self.drop1(y))
        y = y.view(-1, 120)
        y = F.relu(self.fc1(y))
        y = self.fc2(y)
        y = self.soft(y)
        return y

# Training the model
def train(train_data, epochs, criterion):
    model = Network()
    optimizer = torch.optim.SGD(model.parameters(), lr = 0.001)
    model.train()
    for epoch in range(epochs):
        for i, batch in enumerate(train_data):
            optimizer.zero_grad()
            yhat = model(batch[0])
            loss = criterion(yhat.squeeze(), batch[1].squeeze())
            loss.backward()
            optimizer.step()
    return model

# Getting Accuracy for the Test data
def test(model, test_data, criterion):
    model.eval()
    correct_pred = 0
    with torch.no_grad():
        for input, label in test_data:
            yhat = model(input)
            pred = torch.max(yhat.data, 1)[1]
            correct_pred += (pred == label).sum().item()

    return 100. * correct_pred / len(test_data.dataset)



epochs = 20
criterion = nn.NLLLoss()

trainingSet = torchvision.datasets.MNIST(root = './data', train = True, transform =
testSet = torchvision.datasets.MNIST(root = './data', train = False, transform = tra

# Here sampler itself make partitions for the data to be sent on different workers
train_data = DataLoader(trainingSet, sampler = DistributedSampler(
            dataset = trainingSet,
            num_replicas = size,
            rank = rank
        ), batch_size=128)

test_data = DataLoader(testSet, batch_size = 128, shuffle = False)

start_time = MPI.Wtime()

model = train(train_data, epochs, criterion)


if rank != 0:
```

```
    model_params = []
    # parameters from all workers except master node are flattened and stored in a l
    for name, param in model.named_parameters():
        model_params.append(param.data.detach().numpy().flatten())

    comm.send(model_params,0)


if rank == 0:
    # parameters from the master node are extracted using get_parameters
    conv1W = model.get_parameter('conv1.weight').detach().numpy()
    conv1B = model.get_parameter('conv1.bias').detach().numpy()
    conv2W = model.get_parameter('conv2.weight').detach().numpy()
    conv2B = model.get_parameter('conv2.bias').detach().numpy()
    fc1W = model.get_parameter('fc1.weight').detach().numpy()
    fc1B = model.get_parameter('fc1.bias').detach().numpy()
    fc2W = model.get_parameter('fc2.weight').detach().numpy()
    fc2B = model.get_parameter('fc2.bias').detach().numpy()

    for i in range(1,size):
        model_params = comm.recv(source=i)

        # From all the workers the respective parameters are added to return average
        conv1W = np.add(conv1W , np.array(model_params[0]).reshape([10, 1, 5, 5]))
        conv1B = np.add(conv1B , np.array(model_params[1]))
        conv2W = np.add(conv2W , np.array(model_params[2]).reshape([30, 10, 5, 5]))
        conv2B = np.add(conv2B , np.array(model_params[3]))
        fc1W = np.add(fc1W , np.array(model_params[4]).reshape([60, 120]))
        fc1B = np.add(fc1B , np.array(model_params[5]))
        fc2W = np.add(fc2W , np.array(model_params[6]).reshape([10, 60]))
        fc2B = np.add(fc2B , np.array(model_params[7]))

    # to average over the total number of workers
    conv1W /= size
    conv1B /= size
    conv2W /= size
    conv2B /= size
    fc1W /= size
    fc1B /= size
    fc2W /= size
    fc2B /= size

    #Updating model parameters
    # the model is tested using average parameters received from all workers
    with torch.no_grad():
        for name, param in model.named_parameters():
            if name == 'conv1.weight':
                param.copy_(torch.tensor(conv1W))
            if name == 'conv1.bias':
                param.copy_(torch.tensor(conv1B))
            if name == 'conv2.weight':
                param.copy_(torch.tensor(conv2W))
            if name == 'conv2.bias':
                param.copy_(torch.tensor(conv2B))
            if name == 'fc1.weight':
                param.copy_(torch.tensor(fc1W))
            if name == 'fc1.bias':
                param.copy_(torch.tensor(fc1B))
            if name == 'fc2.weight':
                param.copy_(torch.tensor(fc2W))
            if name == 'fc2.bias':
                param.copy_(torch.tensor(fc2B))

        print(f'----------Number of Processes {size}----------------')
```

```
        total_time_taken = MPI.Wtime() - start_time
        print(f'Total time taken for training: {total_time_taken}')

        accuracy = test(model, test_data, criterion)
        print(f'Accuracy on the Test Set : {accuracy}')
```

As we can see the overall training time is decreasing when the number of workers are increased but at the same time accuracy is going down.

```
(base) D:\mastersdata\DDALab\Exercise9>mpiexec -n 1 python PSGD.py
----------Number of Processes 1----------------
Total time taken for training: 375.00888530001976
Accuracy on the Test Set : 91.34

(base) D:\mastersdata\DDALab\Exercise9>mpiexec -n 2 python PSGD.py
----------Number of Processes 2----------------
Total time taken for training: 193.06337129999883
Accuracy on the Test Set : 35.57

(base) D:\mastersdata\DDALab\Exercise9>mpiexec -n 3 python PSGD.py
----------Number of Processes 3----------------
Total time taken for training: 142.55066619999707
Accuracy on the Test Set : 20.05

(base) D:\mastersdata\DDALab\Exercise9>mpiexec -n 4 python PSGD.py
----------Number of Processes 4----------------
Total time taken for training: 123.48240189999342
Accuracy on the Test Set : 10.28

(base) D:\mastersdata\DDALab\Exercise9>mpiexec -n 5 python PSGD.py
----------Number of Processes 5----------------
Total time taken for training: 103.03977919998579
Accuracy on the Test Set : 11.35
```

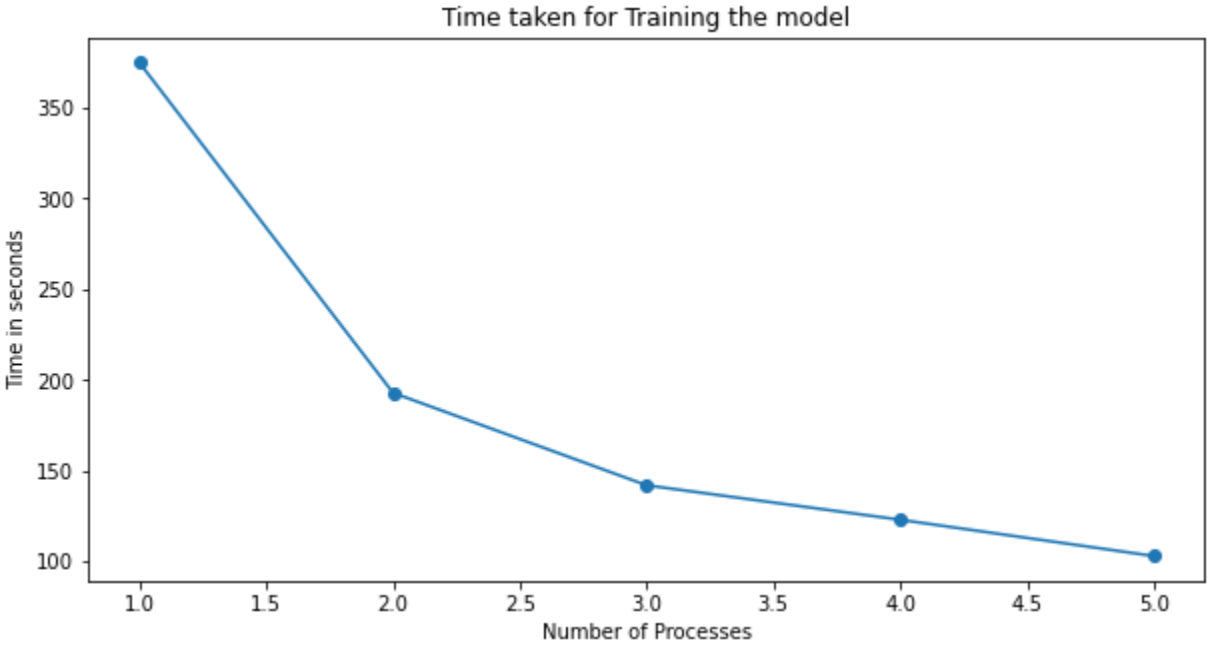
In [22]:

```
xaxis = [1, 2, 3, 4, 5]
acc = [91.34, 35.57, 20.05, 10.28, 11.35]
time = [375, 193, 142, 123, 103]
speedup = [time[0]/time[0], time[0]/time[1], time[0]/time[2], time[0]/time[3], time[
```

Training time

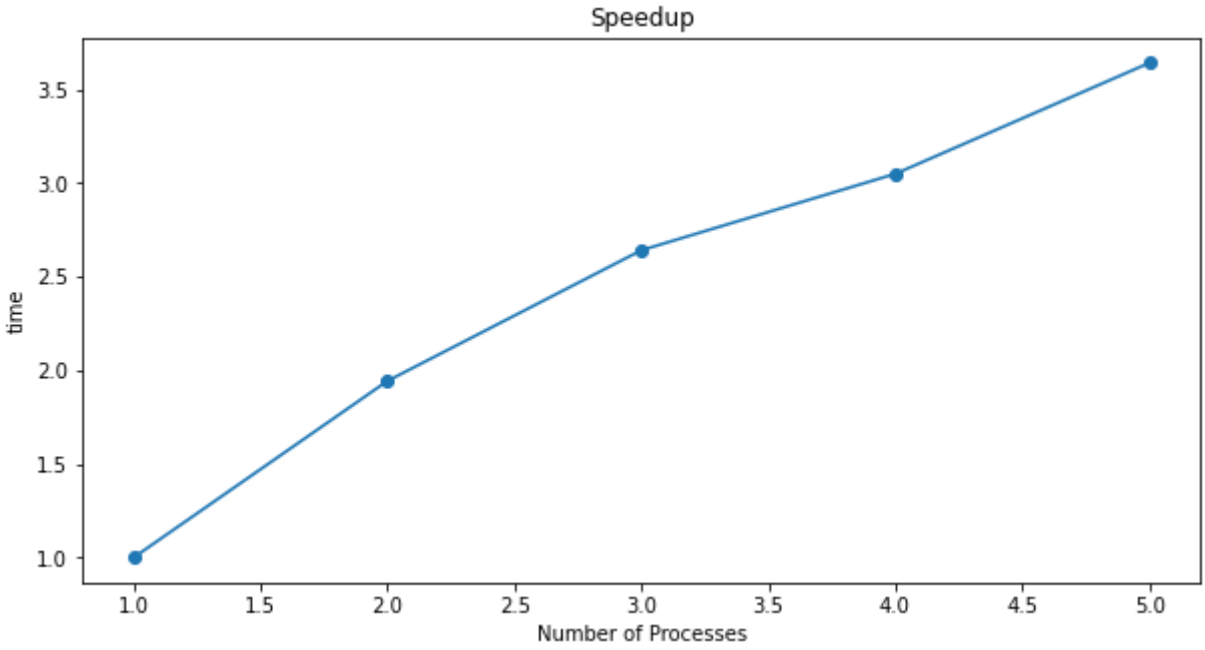
In [20]:

```
plt.figure(figsize=(10,5))
plt.plot(xaxis, time, '-o')
plt.xlabel('Number of Processes')
plt.ylabel('Time in seconds')
plt.title('Time taken for Training the model')
plt.show()
```



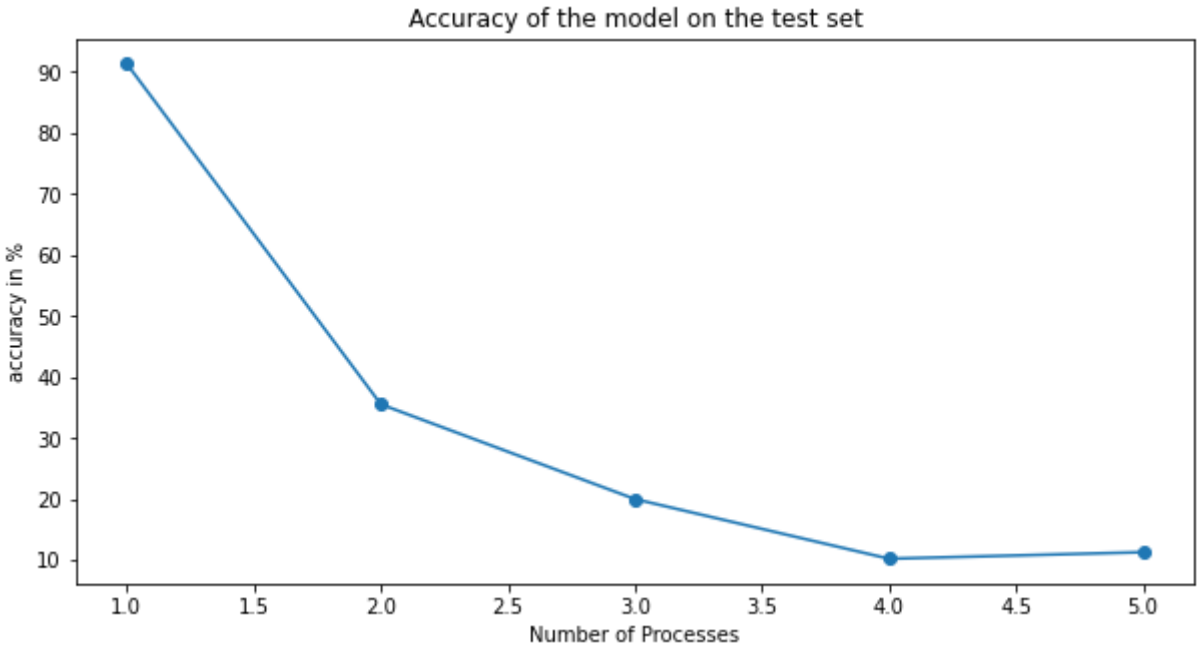
Speedup

In [23]:
```
plt.figure(figsize=(10,5))
plt.plot(xaxis, speedup, '-o')
plt.xlabel('Number of Processes')
plt.ylabel('time')
plt.title('Speedup')
plt.show()
```



Accuracy on the Test set

In [14]:
```
plt.figure(figsize=(10,5))
plt.plot(xaxis, acc, '-o')
plt.xlabel('Number of Processes')
plt.ylabel('accuracy in %')
plt.title('Accuracy of the model on the test set')
plt.show()
```



PyTorch distributed execution

The advantage of pytorch multiprocessing over MPI is that the model weights are shared over each worker instead of them seperately calculating weights and then averaging at the end. In this way each worker even if it has not seen the entire data can somehow make better predictions by seeing the weights calculated by the other workers. Thus the accuracy doesnot become very low as we increase number of processes but the fluctuation in the accuracy is there as we can see below.

In [ ]:

```
import torch
import torch.nn.functional as F
from torch.nn import Module
from torchvision import datasets,transforms
from torch import nn
import torchvision
from torch.utils.data import DataLoader, Dataset
import matplotlib.pyplot as plt
import numpy as np
import torch.multiprocessing as mp
from torch.utils.data import DistributedSampler
import time

# Network
class Network(Module):
    def __init__(self):
        super(Network, self).__init__()
        self.conv1 = nn.Conv2d(1, 10, kernel_size = 5)
        self.pool1 = nn.MaxPool2d(kernel_size = 2, stride = 2)
        self.conv2 = nn.Conv2d(10, 30, kernel_size = 5)
        self.pool2 = nn.MaxPool2d(kernel_size = 2, stride = 2)
        self.drop1 = nn.Dropout(0.15)
        self.pool3 = nn.MaxPool2d(kernel_size = 2, stride = 2)
        self.fc1 = nn.Linear(120, 60)
        self.fc2 = nn.Linear(60, 10)
        self.soft = nn.LogSoftmax(dim = 1)
```

```
    def forward(self, x):
        y = F.relu(self.pool1(self.conv1(x)))
        y = F.relu(self.pool2(self.conv2(y)))
        y = self.pool3(self.drop1(y))
        y = y.view(-1, 120)
        y = F.relu(self.fc1(y))
        y = self.fc2(y)
        y = self.soft(y)
        return y

# Training the model
def train(model, criterion, training_loader, worker_rank, epochs):

    optimizer = torch.optim.SGD(model.parameters(), lr = 0.001)
    model.train()

    for _ in range(epochs):
        for batch_idx, (input, label) in enumerate(training_loader):
            optimizer.zero_grad()
            yhat = model(input)
            loss = criterion(yhat.squeeze(), label.squeeze())
            loss.backward(retain_graph = True)
            optimizer.step()

    return model

# Testing the model accuracy on the test set
def test(model, criterion, test_data):
    model.eval()
    correct_pred = 0
    with torch.no_grad():
        for input, label in test_data:
            output = model(input)
            pred = torch.max(output.data, 1)[1]
            correct_pred += (pred == label).sum().item()

    print(f'Accuracy : {100. * correct_pred / len(test_data.dataset)}')

if __name__ == '__main__':

    Total_workers = 5
    epochs = 20
    mp.set_start_method('spawn')
    model = Network()
    model.share_memory()

    criterion = nn.NLLLoss()

    trainingSet = torchvision.datasets.MNIST(root = './data', train = True, transfor
    testSet = torchvision.datasets.MNIST(root = './data', train = False, transform =


    test_data = DataLoader(testSet, batch_size = 128, shuffle=False)


    processes = []
    start_time = time.perf_counter()
    for i in range(Total_workers):

        train_data = torch.utils.data.DataLoader(trainingSet, sampler=DistributedSam
                dataset = trainingSet,
                num_replicas = Total_workers,
                rank = i
            ), batch_size = 128)
```

```
        p = mp.Process(target = train, args=(model, criterion, train_data, i, epochs
        p.start()                          # start the process
        processes.append(p)

    for p in processes:
        p.join()                        # waiting for all the processes to get over

    total_time_taken = time.perf_counter() - start_time

    print(f'----------Number of Processes {Total_workers}----------------')

    print(f'Total time taken for training: {total_time_taken}')

    test(model, criterion, test_data)
```

```
(base) D:\mastersdata\DDALab\Exercise9>C:/Users/simra/anaconda3/python.exe d:/mastersdata/DDALab/Exercise9/HogWild.py
----------Number of Processes 1----------------
Total time taken for training: 282.0228252
Accuracy : 87.82

(base) D:\mastersdata\DDALab\Exercise9>C:/Users/simra/anaconda3/python.exe d:/mastersdata/DDALab/Exercise9/HogWild.py
----------Number of Processes 2----------------
Total time taken for training: 187.0977022
Accuracy : 91.5

(base) D:\mastersdata\DDALab\Exercise9>C:/Users/simra/anaconda3/python.exe d:/mastersdata/DDALab/Exercise9/HogWild.py
----------Number of Processes 3----------------
Total time taken for training: 159.7601177
Accuracy : 84.8

(base) D:\mastersdata\DDALab\Exercise9>C:/Users/simra/anaconda3/python.exe d:/mastersdata/DDALab/Exercise9/HogWild.py
----------Number of Processes 4----------------
Total time taken for training: 136.92202989999998
Accuracy : 87.15

(base) D:\mastersdata\DDALab\Exercise9>C:/Users/simra/anaconda3/python.exe d:/mastersdata/DDALab/Exercise9/HogWild.py
----------Number of Processes 5----------------
Total time taken for training: 147.1601827
Accuracy : 87.18
```

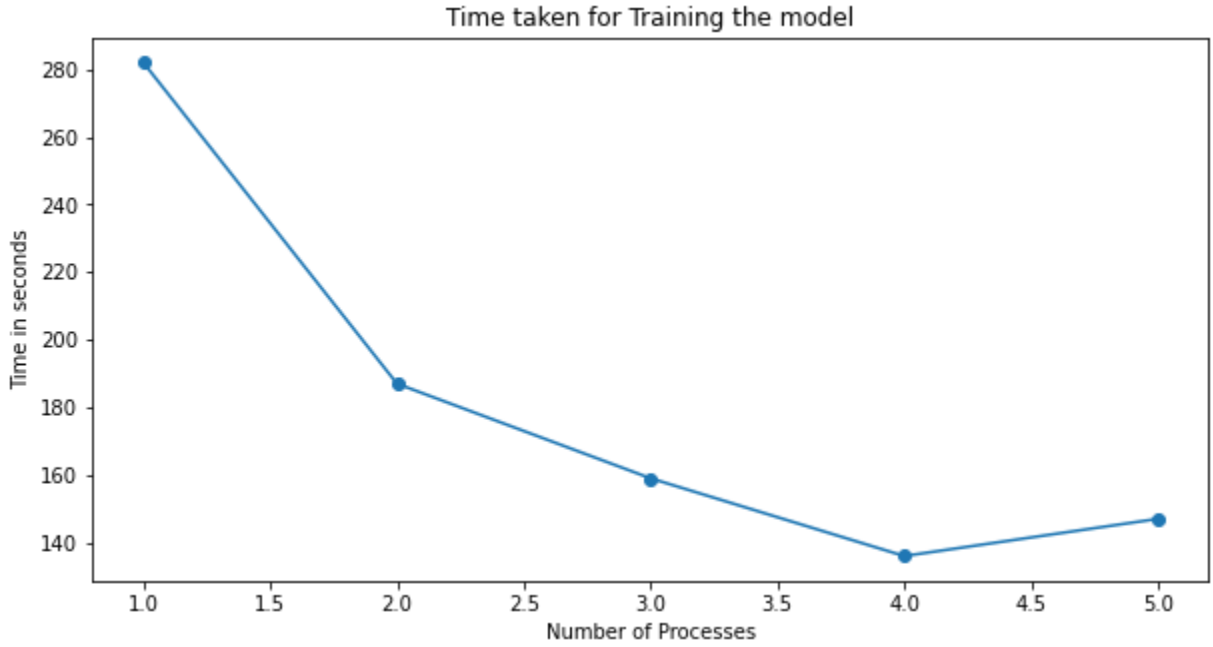
In [27]:

```
xaxis = [1, 2, 3, 4, 5]
accuracy = [87.82, 91.5, 84.8, 87.15, 87.18]
timetaken = [282, 187, 159, 136, 147]
speedup_mp = [timetaken[0]/timetaken[0], timetaken[0]/timetaken[1], timetaken[0]/tim
           timetaken[0]/timetaken[3], timetaken[0]/timetaken[4]]
```

Time taken for training the model

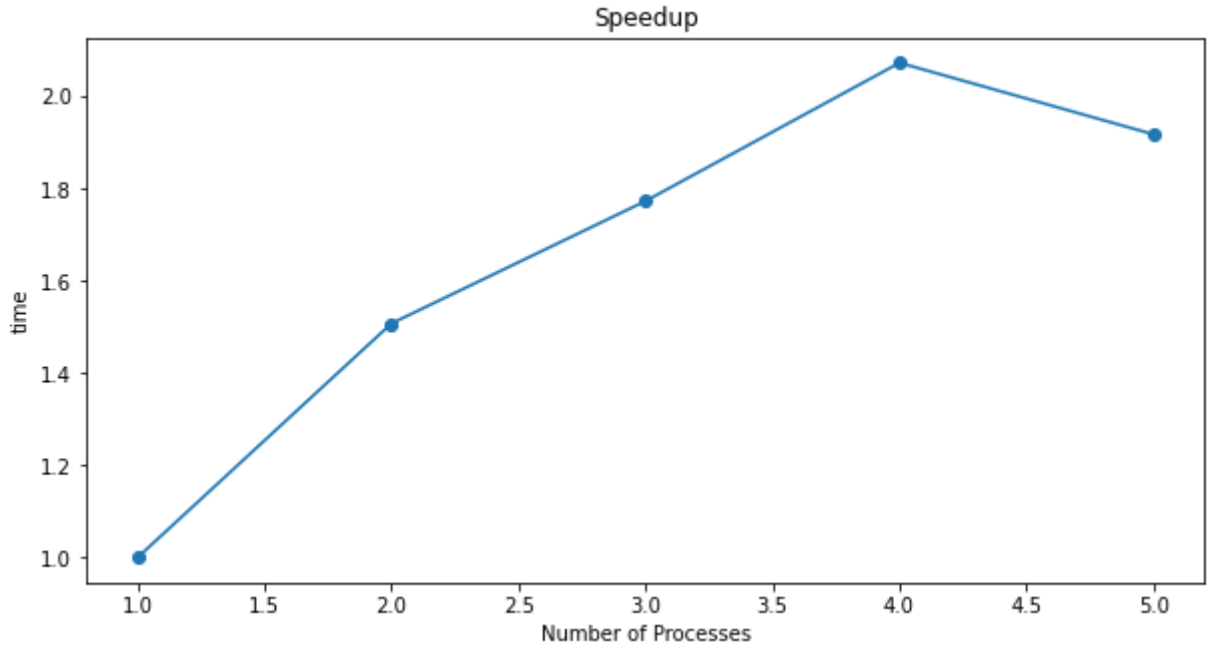
In [25]:

```
plt.figure(figsize=(10,5))
plt.plot(xaxis, timetaken, '-o')
plt.xlabel('Number of Processes')
plt.ylabel('Time in seconds')
plt.title('Time taken for Training the model')
plt.show()
```



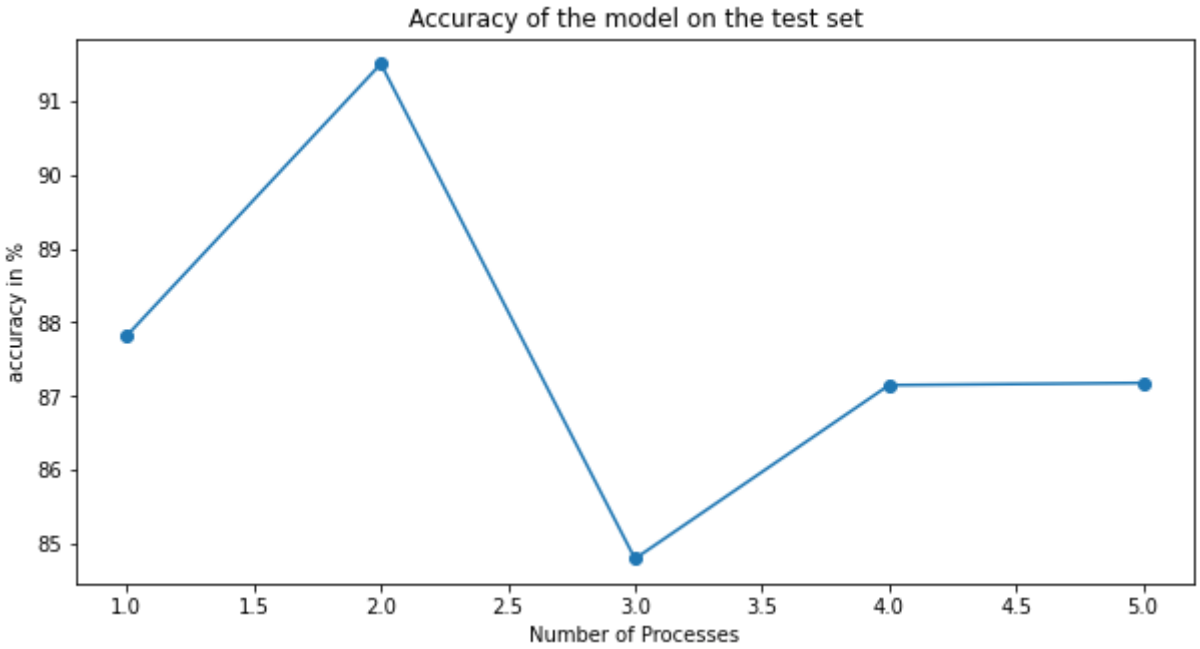
Speedup

In [28]:
```
plt.figure(figsize=(10,5))
plt.plot(xaxis, speedup_mp, '-o')
plt.xlabel('Number of Processes')
plt.ylabel('time')
plt.title('Speedup')
plt.show()
```



Accuracy on the Test set

In [29]:
```
plt.figure(figsize=(10,5))
plt.plot(xaxis, accuracy, '-o')
plt.xlabel('Number of Processes')
plt.ylabel('accuracy in %')
plt.title('Accuracy of the model on the test set')
plt.show()
```


Accuracy of the model on the test set
accuracy in %
91
90
89
88
87
86
85
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
Number of Processes