**Step 0: Update system and install Python 3**

sudo apt update

sudo apt install python3 python3-pip -y

python3 --version

✅ This ensures Python 3 is installed and ready.

**Step 1: Make sure Hadoop is installed**

You still need Hadoop installed, extracted, and environment variables set:

#e Go to /usr/local

cd /usr/local

# Download Hadoop (skip if already downloaded)

sudowget https://downloads.apache.org/hadoop/common/hadoop-3.3.6/hadoop-3.3.6.tar.gz

# Extract

sudo tar -xvzf hadoop-3.3.6.tar.gz

sudo mv hadoop-3.3.6 hadoop

# Give ownership to your user

sudochown -R $USER:$USER hadoop

Set environment variables in ~/.bashrc:

nano ~/.bashrc

Add at the bottom:

export HADOOP\_HOME=/usr/local/hadoop

export PATH=$PATH:$HADOOP\_HOME/bin:$HADOOP\_HOME/sbin

export JAVA\_HOME=/usr/lib/jvm/java-8-openjdk-amd64

Reload:

source ~/.bashrc

Check:

hadoop version

**Step-by-step fix**

Open your hdfs-site.xml:

sudo nano $HADOOP\_HOME/etc/hadoop/hdfs-site.xml

Update these properties:

<configuration>

<property>

<name>dfs.namenode.name.dir</name>

<value>file:/usr/local/hadoop/hdfs/namenode</value>

</property>

<property>

<name>dfs.datanode.data.dir</name>

<value>file:/usr/local/hadoop/hdfs/datanode</value>

</property>

<property>

<name>dfs.replication</name>

<value>1</value>

</property>

</configuration>

Save and exit (Ctrl + O, then Enter, then Ctrl + X).

**Another step**

sudo nano /usr/local/hadoop/etc/hadoop/hadoop-env.sh

**Now find the line:**

# export JAVA\_HOME=

**Uncomment it (remove #) and set it to:**

export JAVA\_HOME=/usr/lib/jvm/java-8-openjdk-amd64

Save the file → Ctrl + O, Enter, then exit → Ctrl + X.

**step 2: Start Hadoop HDFS & YARN**

# Format HDFS (only first time)

hdfsnamenode -format

# Start HDFS

start-dfs.sh

# Start YARN

start-yarn.sh

# Check Hadoop processes

jps

✅ You should see: NameNode, DataNode, ResourceManager, NodeManager, SecondaryNameNode, Jps

**Step 3: Prepare input file in HDFS**

# Remove old HDFS folders

hdfsdfs -rm -r /input /output

# Create HDFS input directory

hdfsdfs -mkdir /input

# Create a sample text file locally

echo "Hadoop is fun. Hadoop is powerful. MapReduce works!" > ~/input.txt

# Upload file to HDFS

hdfsdfs -put ~/input.txt /input

# Check that it is uploaded

hdfsdfs -ls /input

hdfsdfs -cat /input/input.txt

**Step 4: Create Python Mapper**

nano ~/mapper.py

Paste:

#!/usr/bin/env python3

import sys

for line in sys.stdin:

words = line.strip().split()

for word in words:

print(f"{word}\t1")

Make executable:

chmod +x ~/mapper.py

**Step 5: Create Python Reducer**

nano ~/reducer.py

Paste:

#!/usr/bin/env python3

import sys

from collections import defaultdict

counts = defaultdict(int)

for line in sys.stdin:

word, num = line.strip().split("\t")

counts[word] += int(num)

for word, count in counts.items():

print(f"{word}\t{count}")

Make executable:

chmod +x ~/reducer.py

**Step 6: Run Hadoop Streaming Job**

# Remove old output if exists

hdfsdfs -rm -r /output

# Run the Python MapReduce job

hadoop jar $HADOOP\_HOME/share/hadoop/tools/lib/hadoop-streaming-3.3.6.jar \

-input /input \

-output /output \

-mapper ~/mapper.py \

-reducer ~/reducer.py

**Step 7: Check Output**

hdfsdfs -ls /output

hdfsdfs -cat /output/part-00000

You should see something like:

Hadoop 2

is 2

fun. 1

powerful. 1

MapReduce 1

works! 1

✅ Done thalaaa! This is the easiest Python streaming Hadoop MapReduce workflow:

1. Install Python
2. Upload file to HDFS
3. Write tiny Python mapper & reducer
4. Run Hadoop streaming
5. Check output

**SPARK**

**Step 0: Install Java & Scala (needed for Spark)**

sudo apt update

sudo apt install openjdk-11-jdk scala -y

java -version

scala -version

**Step 1: Download & Install Spark**

cd /usr/local

sudowget https://downloads.apache.org/spark/spark-3.5.6/spark-3.5.6-bin-hadoop3.tgz

sudo tar -xvzf spark-3.5.6-bin-hadoop3.tgz

sudo mv spark-3.5.6-bin-hadoop3 spark

Give permissions:

sudochown -R $USER:$USER /usr/local/spark

**Step 2: Set Environment Variables**

Edit your ~/.bashrc:

nano ~/.bashrc

Add at the bottom:

export SPARK\_HOME=/usr/local/spark

export PATH=$PATH:$SPARK\_HOME/bin:$SPARK\_HOME/sbin

Reload:

source ~/.bashrc

Check Spark version:

spark-shell --version

**Step 3: Start Spark (Standalone mode)**

Start Spark master:

start-master.sh

Start Spark worker:

start-worker.sh spark://localhost:7077

💡 You can also check Spark UI at:  
👉<http://localhost:8080>

**Step 4: Simple Spark WordCount in Python**

Create a file:

nano wordcount.py

Paste:

from pyspark import SparkContext

sc = SparkContext("local", "WordCountApp")

# Load input file

text\_file = sc.textFile("input.txt")

# Split into words and count

counts = text\_file.flatMap(lambda line: line.split(" ")) \

.map(lambda word: (word, 1)) \

.reduceByKey(lambda a, b: a + b)

# Save result

counts.saveAsTextFile("output\_spark")

sc.stop()

**Step 5: Run Spark Job**

First create a sample input file:

echo "Hadoop is fun. Spark is faster. Spark and Hadoop are powerful!" > input.txt

Now run with Spark:

spark-submit wordcount.py

**Step 6: Check Output**

The output will be in a folder called output\_spark. Check with:

ls output\_spark/

cat output\_spark/part-00000

Example output:

('Hadoop', 2)

('is', 2)

('fun.', 1)

('Spark', 2)

('faster.', 1)

('and', 1)

('are', 1)

('powerful!', 1)

**OPENMP AND MPI**

**PART 1: OPENMP (Shared Memory Parallelism)**

OpenMP is used to run multiple threads **inside the same CPU**.

**✅ Example 1: Fibonacci Series using OpenMP**

#include <stdio.h>

#include <omp.h>

int main() {

int n, i;

printf("Enter number of terms: ");

scanf("%d", &n);

int fib[n];

fib[0] = 0;

fib[1] = 1;

#pragmaomp parallel for

for(i = 2; i< n; i++) {

fib[i] = fib[i-1] + fib[i-2];

}

printf("Fibonacci Series: ");

for(i = 0; i< n; i++)

printf("%d ", fib[i]);

printf("\n");

return 0;

}

**💻 How to compile & run:**

gcc -fopenmpfib\_openmp.c -o fib

./fib

**✅ Example 2: Check Prime Numbers using OpenMP**

#include <stdio.h>

#include <omp.h>

int main() {

int n, i, count = 0;

printf("Enter a number: ");

scanf("%d", &n);

#pragma omp parallel for reduction(+:count)

for(i = 1; i<= n; i++) {

if(n % i == 0)

count++;

}

if(count == 2)

printf("%d is a Prime Number\n", n);

else

printf("%d is NOT a Prime Number\n", n);

return 0;

}

**💻 Run it:**

gcc -fopenmpprime\_openmp.c -o prime

./prime

**🖥️ PART 2: MPI (Distributed Parallelism)**

MPI runs multiple **processes**, possibly across different computers (or cores).

**✅ Example 3: Fibonacci using MPI**

#include <stdio.h>

#include <mpi.h>

int main(int argc, char \*argv[]) {

int rank, size, n, i, fib[100];

MPI\_Init(&argc, &argv);

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);

MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);

if(rank == 0) {

printf("Enter number of terms: ");

scanf("%d", &n);

}

MPI\_Bcast(&n, 1, MPI\_INT, 0, MPI\_COMM\_WORLD);

fib[0] = 0;

fib[1] = 1;

for(i = 2; i< n; i++)

fib[i] = fib[i-1] + fib[i-2];

if(rank == 0) {

printf("Fibonacci Series: ");

for(i = 0; i< n; i++)

printf("%d ", fib[i]);

printf("\n");

}

MPI\_Finalize();

return 0;

}

**💻 Compile and run:**

mpiccfib\_mpi.c -o fib\_mpi

mpirun -np 4 ./fib\_mpi

**✅ Example 4: Prime Check using MPI**

#include <stdio.h>

#include <mpi.h>

int main(int argc, char \*argv[]) {

int rank, size, n, i, count = 0, local\_count = 0;

MPI\_Init(&argc, &argv);

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);

MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);

if(rank == 0) {

printf("Enter a number: ");

scanf("%d", &n);

}

MPI\_Bcast(&n, 1, MPI\_INT, 0, MPI\_COMM\_WORLD);

for(i = rank + 1; i<= n; i += size) {

if(n % i == 0)

local\_count++;

}

MPI\_Reduce(&local\_count, &count, 1, MPI\_INT, MPI\_SUM, 0, MPI\_COMM\_WORLD);

if(rank == 0) {

if(count == 2)

printf("%d is Prime\n", n);

else

printf("%d is NOT Prime\n", n);

}

MPI\_Finalize();

return 0;

}

**💻 Run it:**

mpiccprime\_mpi.c -o prime\_mpi

mpirun -np 4 ./prime\_mpi

**⚡ Quick Revision:**

| **Type** | **Compiler** | **Run Command** | **Used For** |
| --- | --- | --- | --- |
| **OpenMP** | gcc -fopenmp | ./output | Shared memory (threads) |
| **MPI** | mpicc | mpirun -np N ./output | Distributed memory ( |

🧩 1️⃣ OpenMP (C Programs)

*(Compile with gccfilename.c -fopenmp -o output and run ./output)*

🔹 Program 1 — Hello World with Threads

#include <stdio.h>

#include <omp.h>

int main() {

#pragmaomp parallel

{

int tid = omp\_get\_thread\_num();

printf("Hello from thread %d\n", tid);

}

return 0;

}

👉 Concept: Each thread prints its own message using omp\_get\_thread\_num().

🔹 Program 2 — Parallel For Loop

#include <stdio.h>

#include <omp.h>

int main() {

int i;

#pragmaomp parallel for

for (i = 0; i< 5; i++) {

printf("Iteration %d executed by thread %d\n", i, omp\_get\_thread\_num());

}

return 0;

}

👉 Concept: Automatically divides loop iterations among threads.

🔹 Program 3 — Reduction (Sum of Array)

#include <stdio.h>

#include <omp.h>

int main() {

int i;

int sum = 0;

int a[5] = {1, 2, 3, 4, 5};

#pragma omp parallel for reduction(+:sum)

for (i = 0; i< 5; i++) {

sum += a[i];

}

printf("Total Sum = %d\n", sum);

return 0;

}

👉 Concept: Each thread adds partial sums, and OpenMP handles combining results.

🔹 Program 4 — Critical Section Example

#include <stdio.h>

#include <omp.h>

int main() {

int count = 0;

#pragmaomp parallel

{

#pragmaomp critical

{

count++;

printf("Thread %d incremented count = %d\n", omp\_get\_thread\_num(), count);

}

}

return 0;

}

👉 Concept: #pragma omp critical ensures safe access to shared variables.

⚙️ 2️⃣ MPI (Message Passing Interface)

*(Compile with mpiccfilename.c -o output and run with mpirun -np 4 ./output)*

🔹 Program 1 — Hello World

#include <mpi.h>

#include <stdio.h>

int main(int argc, char\*\* argv) {

MPI\_Init(&argc, &argv);

int rank, size;

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);

MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);

printf("Hello from process %d of %d\n", rank, size);

MPI\_Finalize();

return 0;

}

👉 Concept: Each process prints its rank (unique ID).

🔹 Program 2 — Point-to-Point Communication

#include <mpi.h>

#include <stdio.h>

int main(int argc, char\*\* argv) {

MPI\_Init(&argc, &argv);

int rank;

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);

int number;

if (rank == 0) {

number = 100;

MPI\_Send(&number, 1, MPI\_INT, 1, 0, MPI\_COMM\_WORLD);

printf("Process 0 sent number %d\n", number);

} else if (rank == 1) {

MPI\_Recv(&number, 1, MPI\_INT, 0, 0, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

printf("Process 1 received number %d\n", number);

}

MPI\_Finalize();

return 0;

}

👉 Concept: Process 0 sends data, Process 1 receives it.

🔹 Program 3 — Broadcast

#include <mpi.h>

#include <stdio.h>

int main(int argc, char\*\* argv) {

MPI\_Init(&argc, &argv);

int rank, number;

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);

if (rank == 0) {

number = 42;

}

MPI\_Bcast(&number, 1, MPI\_INT, 0, MPI\_COMM\_WORLD);

printf("Process %d received number %d\n", rank, number);

MPI\_Finalize();

return 0;

}

👉 Concept: Rank 0 broadcasts a value to all other processes.

🔹 Program 4 — Scatter and Gather

#include <mpi.h>

#include <stdio.h>

int main(int argc, char\*\* argv) {

MPI\_Init(&argc, &argv);

int rank, size;

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);

MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);

int send\_data[4] = {10, 20, 30, 40};

int recv\_data;

MPI\_Scatter(send\_data, 1, MPI\_INT, &recv\_data, 1, MPI\_INT, 0, MPI\_COMM\_WORLD);

printf("Process %d received %d\n", rank, recv\_data);

recv\_data \*= 2; // Modify data

MPI\_Gather(&recv\_data, 1, MPI\_INT, send\_data, 1, MPI\_INT, 0, MPI\_COMM\_WORLD);

if (rank == 0) {

printf("Final gathered data: ");

for (int i = 0; i< size; i++) printf("%d ", send\_data[i]);

printf("\n");

}

MPI\_Finalize();

return 0;

}

👉 Concept: Distribute (scatter) and collect (gather) data among processes.