HIGH PERFORMANCE COMPUTING

LAB ASSIGNMENT – 01

BATCH – 04

2303A51221

T.SAI SATHWIK

JUST FOR THE REFERENCE PURPOSE IM KEEPING THE CODE BELOW

```
#Save as io_to_scratch.py:
# io_to_scratch.py
import os, time, platform

def compute(n=300_000):
  s = 0.0
  for i in range(n):
    s += (i % 13) * 0.314159
    return s

if __name__ == "__main__":
  scratch = os.getenv("SLURM_TMPDIR", os.getenv("TMPDIR", "/tmp"))
  result = compute()
  ts = time.strftime("%Y%m%d_%H%M%S")
  out_path = os.path.join(scratch, f"result_{ts}.txt")
  with open(out_path, "w") as f:
    f.write(f"Host={platform.node()}\nResult={result:.6f}\n")
    print("Wrote:", out_path)
```

```
import os, time, platform


def compute(n=300_000):
  s = 0.0
  for i in range(n):
    s += (i % 13) * 0.314159
  return s


if __name__ == "__main__":
  # Portable scratch directory (Windows / Linux / Colab safe)
  scratch = os.getcwd()
```

```
result = compute()

ts = time.strftime("%Y%m%d_%H%M%S")
out_path = os.path.join(scratch, f"result_{ts}.txt")

with open(out_path, "w") as f:
    f.write(f"Host={platform.node()}\n")
    f.write(f"Result={result:.6f}\n")

print("Wrote:", out_path)
```

CPU

```
# CPU
Wrote: /tmp/result_20260128_042249.txt
```

GPU

```
# GPU
Wrote: /tmp/result_20260128_042630.txt
```

TPU

```
# TPU

Wrote: /tmp/result_20260128_042709.txt
```

LOCAL SERVER (VS CODE)

```
PS D:\3-2 SEM\HPC-1221> python -u "d:\3-2 SEM\HPC-1221\HPC.PY"
Wrote: D:\3-2 SEM\HPC-1221\result_20260128_100350.txt
PS D:\3-2 SEM\HPC-1221>
```

HIGH PERFORMANCE COMPUTING

LAB ASSIGNMENT – 02

BATCH – 04

2303A51221

T.SAI SATHWIK

Performance Analysis Across CPU, GPU, TPU and Local Execution

Aim:

To analyze and compare the execution time and performance of a compute-intensive Python program across different execution environments such as Local CPU, Google Colab CPU, GPU, and TPU.

PROBLEM DESCRIPTION

The program computes a pairwise potential among randomly generated 2D points. The algorithm has $O(N^2)$ time complexity, making it compute-intensive and suitable for performance analysis in HPC environments.

JUST FOR THE REFERENCE PURPOSE IM KEEPING THE CODE BELOW

```
import time

import random

import math

import cProfile

import pstats

import io

import tracemalloc


# Generate random 2D points

def gen_points(n, seed=7):

    random.seed(seed)

    points = []

    for _ in range(n):

        x = random.random()

        y = random.random()

        points.append((x, y))

    return points
```

```
# Compute pairwise potential (O(N^2))
def pairwise_potential(points, eps=1e-6):
    n = len(points)
    pot = [0.0] * n

    for i in range(n):
        xi, yi = points[i]
        acc = 0.0
        for j in range(n):
            if i == j:
                continue
            xj, yj = points[j]
            dx = xi- xj
            dy = yi- yj
            r = math.sqrt(dx*dx + dy*dy) + eps
            acc += 1.0 / r
        pot[i] = acc
    return pot

def main():
    N = 800

    pts = gen_points(N)

    tracemalloc.start()
    profiler = cProfile.Profile()
    profiler.enable()

    start = time.perf_counter()
    pot = pairwise_potential(pts)
    end = time.perf_counter()

    profiler.disable()
```

```
    current, peak = tracemalloc.get_traced_memory()
    tracemalloc.stop()

    print(f"N = {N}")
    print(f"Execution Time: {end- start:.3f} seconds")
    print(f"Sample pot[0] = {pot[0]:.6f}")
    print(f"Peak Memory Usage: {peak/1e6:.2f} MB")

    buffer = io.StringIO()
    stats = pstats.Stats(profiler, stream=buffer).sort_stats("cumtime")
    stats.print_stats(10)

    print("\n--- CPU Profiling (Top 10)---")
    print(buffer.getvalue())

if __name__ == "__main__":
    main()
```

CPU – GOOGLE COLAB (OUTPUT)

```
N = 800
Execution Time: 2.677 seconds
Sample pot[0] = 2390.396335
Peak Memory Usage: 0.03 MB

--- CPU Profiling (Top 10) ---
         639209 function calls (639208 primitive calls) in 2.677 seconds

   Ordered by: cumulative time

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    0.850    0.850    1.005    1.005 {built-in method time.sleep}
      2/1    1.413    0.706    0.800    0.800 /tmp/ipython-input-3515698522.py:18(pairwise_potential
   639200    0.414    0.000    0.414    0.000 {built-in method math.sqrt}
        1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Profiler' objects}
        2    0.000    0.000    0.000    0.000 {built-in method posix.getppid}
        2    0.000    0.000    0.000    0.000 {built-in method time.perf_counter}
        1    0.000    0.000    0.000    0.000 {built-in method builtins.len}
```

GPU - GOOGLE COLAB (OUTPUT)

```
••• N = 800
    Execution Time: 2.744 seconds
    Sample pot[0] = 2390.396335
    Peak Memory Usage: 0.03 MB

    --- CPU Profiling (Top 10) ---
             639211 function calls (639210 primitive calls) in 2.745 seconds

       Ordered by: cumulative time

       ncalls  tottime  percall  cumtime  percall filename:lineno(function)
            2    1.711    0.855    2.010    1.005 {built-in method time.sleep}
          2/1    0.620    0.310    0.729    0.729 /tmp/ipython-input-3515698522.py:18(pairwise_potential)
       639200    0.414    0.000    0.414    0.000 {built-in method math.sqrt}
            1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Profiler' objects}
            3    0.000    0.000    0.000    0.000 {built-in method posix.getppid}
            2    0.000    0.000    0.000    0.000 {built-in method time.perf_counter}
            1    0.000    0.000    0.000    0.000 {built-in method builtins.len}
```

OBSERVATIONS

Time will be almost same as CPU

Reason: Python loops do not utilize GPU

TPU – GOOGLE COLAB (OUTPUT)

```
••• N = 800
    Execution Time: 1.410 seconds
    Sample pot[0] = 2390.396335
    Peak Memory Usage: 0.03 MB

    --- CPU Profiling (Top 10) ---
             639207 function calls (639206 primitive calls) in 1.410 seconds

       Ordered by: cumulative time

       ncalls  tottime  percall  cumtime  percall filename:lineno(function)
          2/1    1.197    0.598    0.641    0.641 /tmp/ipython-input-3515698522.py:18(pairwise_potential
       639200    0.213    0.000    0.213    0.000 {built-in method math.sqrt}
            1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Profiler' objects}
            2    0.000    0.000    0.000    0.000 {built-in method time.perf_counter}
            1    0.000    0.000    0.000    0.000 {built-in method posix.getppid}
            1    0.000    0.000    0.000    0.000 {built-in method builtins.len}
```

OBSERVATION

- **No speedup**
- **TPU not utilized by normal Python code**

LOCAL SERVER (VS CODE) (OUTPUT)

```
PS D:\3-2 SEM\HPC-1221> python -u "d:\3-2 SEM\HPC-1221\HPC.PY"
N = 800
Execution Time: 1.146 seconds
Sample pot[0] = 2390.396335
Peak Memory Usage: 0.03 MB

--- CPU Profiling (Top 10) ---
         639205 function calls in 1.146 seconds

   Ordered by: cumulative time

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    0.973    0.973    1.146    1.146 d:\3-2 SEM\HPC-1221\HPC.PY:18(pairwise_potential)
   639200    0.173    0.000    0.173    0.000 {built-in method math.sqrt}
        1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Profiler' objects}
        2    0.000    0.000    0.000    0.000 {built-in method time.perf_counter}
        1    0.000    0.000    0.000    0.000 {built-in method builtins.len}
```

KEY OBSERVATIONS

The program is compute-bound with $O(N^2)$ complexity.

CPU profiling shows most time spent inside nested loops.

GPU and TPU do not provide speedup for pure Python code.

Specialized libraries (NumPy, CUDA, TensorFlow) are required to utilize accelerators.

This experiment demonstrates that merely changing hardware does not guarantee performance improvement. Efficient utilization of GPUs and TPUs requires parallel-aware libraries and optimized code. The serial Python implementation serves as a baseline for future parallel and optimized implementations in High Performance Computing.

-------------------THANKYOU------------------