CS 301
High-Performance Computing

Lab 9 - Calculation of pi using random numbers.

Problem 1: Calculation of pi.

Jay Dobariya (202101521)
Akshar Panchani (202101522)

April 15, 2024

Contents

1 Introduction

The estimate of basic constants such as π is very important in computational mathematics. The ratio of a circle's circumference to its diameter, or π, is commonly used in a variety of scientific and technical fields. It is roughly equivalent to 3.14159. While mathematical formulas like the Leibniz series or the Gauss-Legendre algorithm may be used to determine π with accuracy, these approaches are computationally demanding since they frequently call for intricate mathematical processes.

As an alternative, the Monte Carlo technique provides a simple yet effective way to estimate π. A useful way to approximate π with some degree of precision is to use the Monte Carlo approach, which uses random sampling techniques inside a predefined geometric domain. By creating random points and then analyzing their geographical distribution, this approach makes use of statistical inference and probability theory to enable the calculation of π.

2 Hardware Details

2.1 Hardware Details for LAB207 PCs

- Architecture: x86_64
- CPU op-mode(s): 32-bit, 64-bit
- Byte Order: Little Endian
- CPU(s): 6
- On-line CPU(s) list: 0-5
- Thread(s) per core: 1
- Core(s) per socket: 6
- Socket(s): 1
- NUMA node(s): 1
- Vendor ID: GenuineIntel
- CPU family: 6
- Model: 155
- Model name: Intel(R) Core(TM) i5-8500 CPU @ 3.00GHz
- Stepping: 10
- CPU MHz: 799.992
- CPU max MHz: 4100.0000
- CPU min MHz: 800.0000

- BogoMIPS: 6000.00
- Virtualization: VT-x
- L1d cache: 192KB
- L1i cache: 192KB
- L2 cache: 1.5MB
- L3 cache: 9MB
- NUMA node0 CPU(s): 0-5

2.2 Hardware Details for HPC Cluster

- Architecture: x86_64
- CPU op-mode(s): 32-bit, 64-bit
- Byte Order: Little Endian
- CPU(s): 24
- On-line CPU(s) list: 0-23
- Thread(s) per core: 2
- Core(s) per socket: 6
- Socket(s): 2
- NUMA node(s): 2
- Vendor ID: GenuineIntel
- CPU family: 6
- Model: 63
- Model name: Intel(R) Xeon(R) CPU E5-2620 v3 @ 2.40GHz
- Stepping: 2
- CPU MHz: 2642.4378
- BogoMIPS: 4804.69
- Virtualization: VT-x
- L1d cache: 32K
- L1i cache: 32K
- L2 cache: 256K

- L3 cache: 15360K
- NUMA node0 CPU(s): 0-5,12-17
- NUMA node1 CPU(s): 6-11,18-23

3 Problem 2

3.1 Brief description of the problem

The task is to use random locations inside a unit square to estimate the value of π.

3.1.1 Algorithm description

The algorithm can be described as follows:

1. Initialize variables and filenames.
2. Set up a parallel region using OpenMP with the specified number of threads.
3. Generate random points within the unit square.
4. Calculate the distance of each point from the origin.
5. Increment the count of points inside the unit circle.
6. Calculate the value of π using the ratio of points inside the circle to the total number of points.

3.2 The complexity of the algorithm- serial

$O(N)$ is the complexity of the serial implementation, where N is the input array's size.

3.3 Information about parallel implementation

The parallel implementation utilizes OpenMP for thread-level parallelism. OpenMP, a popular API for shared-memory parallel programming, is used in the algorithm's parallel implementation. By offering a collection of compiler directives, runtime library functions, and environment variables that make it simple for programmers to represent parallelism in their code, OpenMP streamlines parallel programming.

3.4 The complexity of the algorithm - Parallel

Although the parallel execution improves performance, the parallel implementation's complexity stays at $O(N)$.

3.5 Profiling using HPC Cluster (with gprof)

The screenshots of profiling using the HPC Cluster are given below

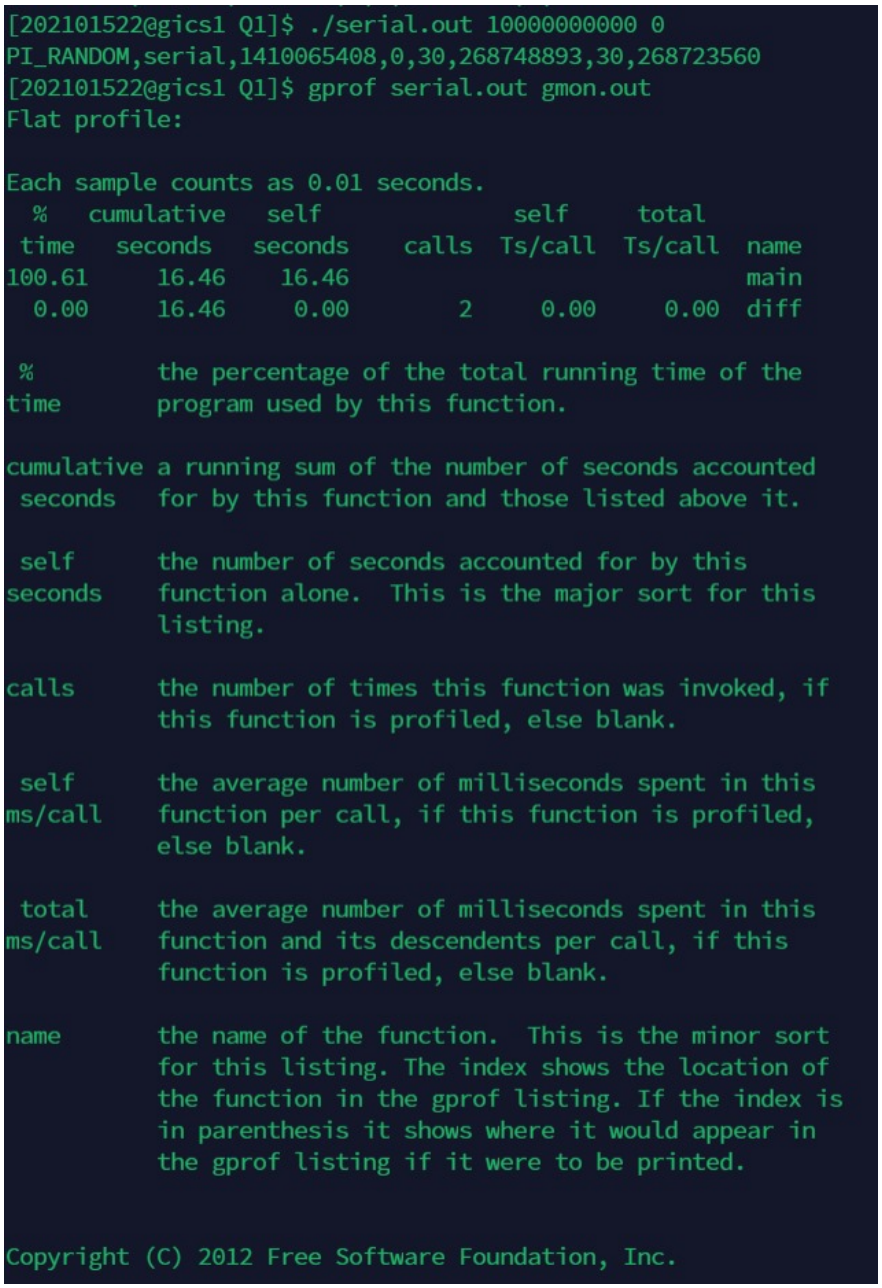

Figure 1: Screenshot of the terminal from HPC Cluster

4 Lab207 PC Graph and HPC Cluster Graphs

4.0.1 Graph of Total time vs Problem size for LAB207 PCs

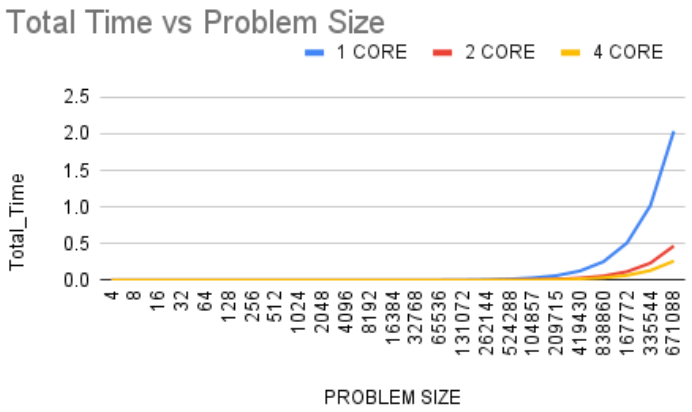

Figure 2: Graph of Total time vs Problem size for **(Hardware: LAB207 PC, Problem: PI_RANDOM)**.

4.0.2 Graph of Algorithm time vs Problem size for LAB207 PCs

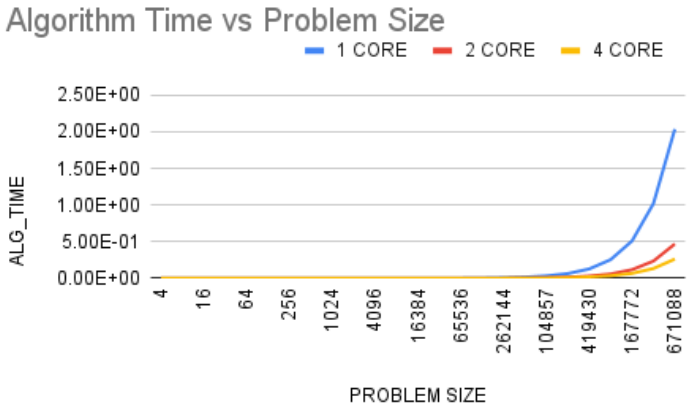

Figure 3: Graph of Algorithm time vs Problem size for **(Hardware: LAB207 PC, Problem: PI_RANDOM)**.

4.0.3 Graph of Speedup vs Core for LAB207 PCs

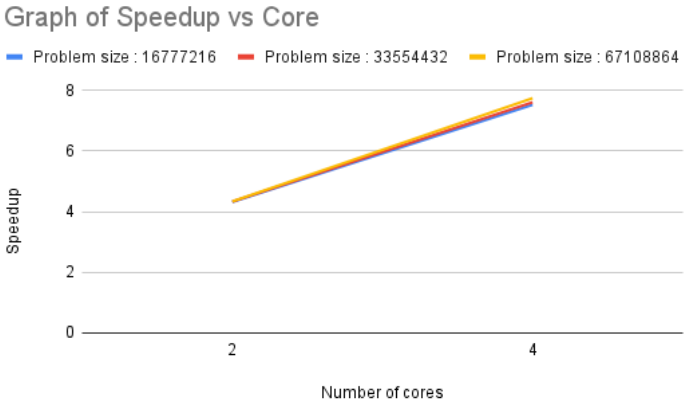

Figure 4: Graph of Speedup vs Core for **(Hardware: LAB207 PC, Problem: PI_RANDOM)**.

4.0.4 Graph of Speedup vs Problem size for LAB207 PCs

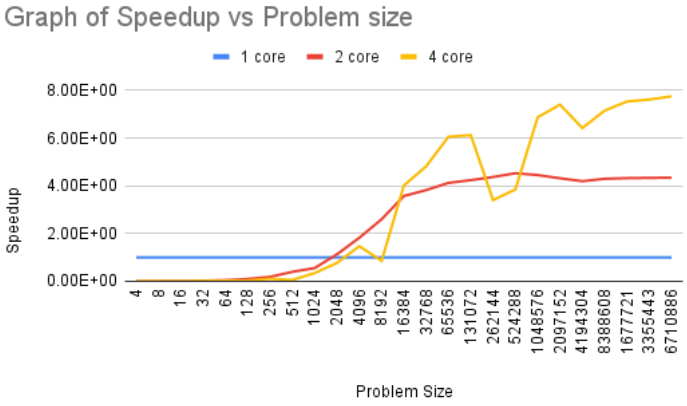

Figure 5: Graph of Speedup vs Problem size for **(Hardware: LAB207 PC, Problem: PI_RANDOM)**.

4.0.5 Graph of Total Time vs Problem size for Cluster

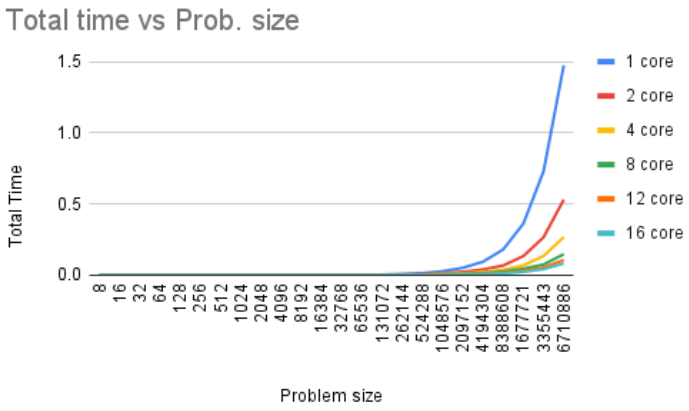

Figure 6: Graph of Total Time vs Problem size for **(Hardware: HPC CLUSTER, Problem: PI_RANDOM)**.

4.0.6 Graph of Algorithm time vs Problem size for Cluster

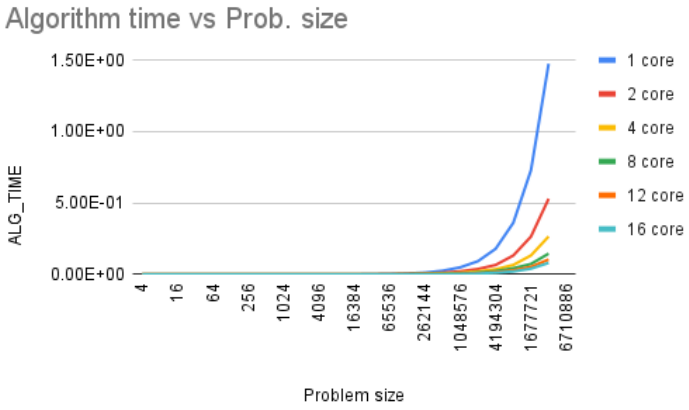

Figure 7: Graph of Algorithm time vs Problem size for **(Hardware: HPC CLUSTER, Problem: PI_RANDOM)**.

4.0.7 Graph of Speedup vs Core for Cluster

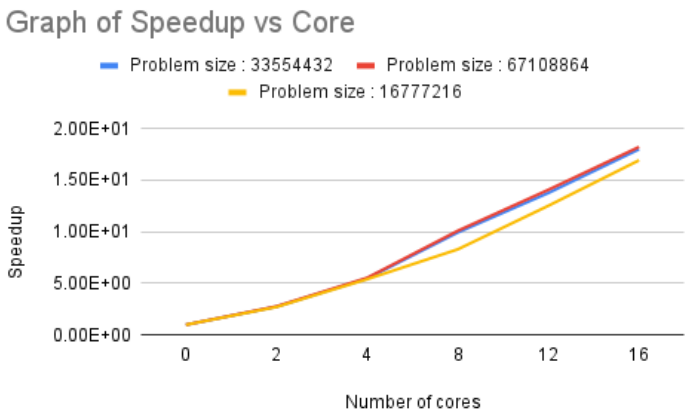

Figure 8: Graph of Speedup vs Core for **(Hardware: HPC CLUSTER, Problem: PI_RANDOM)**.

4.0.8 Graph of Speedup vs Problem size for Cluster

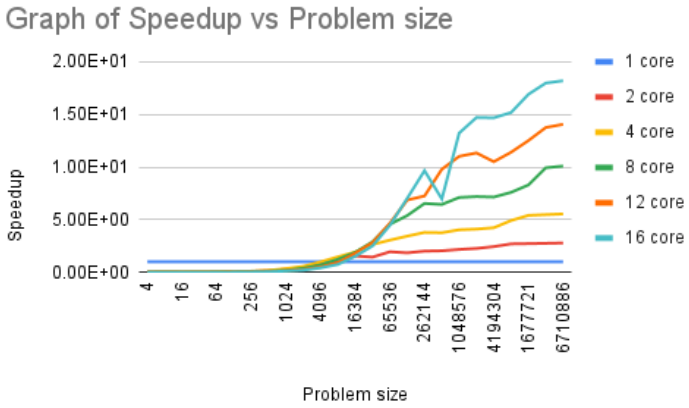

Figure 9: Graph of Speedup vs Problem size for **(Hardware: HPC CLUSTER, Problem: PI_RANDOM)**.

5 Conclusions

- To sum up, we have seen both serial and parallel versions of the Monte Carlo approach to π estimation. Large-scale calculations can benefit from the parallel implementation's notable speedup over the serial one.