

---

---

# CS 301

## High-Performance Computing

---

---

### Lab 4 - Performance evaluation of codes

Problem A-2: Calculation of pi using series

Jay Dobariya (202101521)  
Akshar Panchani (202101522)

February 28, 2024

# Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>Hardware Details</b>	<b>3</b>
2.1	Hardware Details for LAB207 PCs . . . . .	3
2.2	Hardware Details for HPC Cluster . . . . .	4
<b>3</b>	<b>Problem A2</b>	<b>5</b>
3.1	Brief description of the problem . . . . .	5
3.1.1	Algorithm description . . . . .	5
3.2	The complexity of the algorithm (serial) . . . . .	5
3.3	Information about parallel implementation . . . . .	5
3.4	The complexity of the algorithm (Parallel) . . . . .	5
3.5	Profiling using HPC Cluster (with gprof) . . . . .	6
3.6	Graph of Problem Size vs Runtime . . . . .	7
3.6.1	Graph of Problem Size vs Totaltime for LAB207 PCs . . . . .	7
3.6.2	Graph of Problem Size vs Algorithm time for LAB207 PCs . . . . .	7
3.6.3	Graph of Problem Size vs Totaltime for HPC Cluster . . . . .	8
3.6.4	Graph of Problem Size vs Algorithm time for HPC Cluster . . . . .	8
<b>4</b>	<b>Conclusions</b>	<b>8</b>

# 1 Introduction

The Leibniz formula for Pi provides a method to approximate the value of  $\pi$  using an infinite series.

$$\pi = 4/1 - 4/3 + 4/5 - 4/7 + 4/9 - 4/11 + \dots$$

It involves summing a sequence of terms where each term is derived by taking 4 divided by an odd number, with the signs alternating between positive and negative. By utilizing a finite number of terms (up to a specified value of N), we can obtain an approximation of  $\pi$ . The accuracy of this approximation improves with a larger value of N, allows us for a more precise estimation of the mathematical constant  $\pi$ .

## 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 A2

### 3.1 Brief description of the problem

In the problem A2, The Leibniz formula for Pi is used in order to get the value of constant Pi. This formula provides a practical and iterative approach for computing and is a valuable tool in numerical and computational methods which ultimately helps in improving in the High performance computing.

#### 3.1.1 Algorithm description

1. Initialize variables:  $sum = 0$ ,  $sign = 1$ .
2. For  $i$  from 1 to  $N$  (number of terms to sum):
  - (a) Add  $sign \times \frac{4}{2i-1}$  to  $sum$ .
  - (b) Toggle the sign:  $sign = -sign$ .
3. Calculate the final approximation of  $\pi$ :  $\pi \approx sum$ .

### 3.2 The complexity of the algorithm (serial)

The algorithm has an  $O(N)$  time complexity, where  $N$  is the number of terms in the series. This is due to the algorithm doing constant time operations for every term, iterating over each one precisely once.

### 3.3 Information about parallel implementation

The provided code implements the Leibniz formula for calculating the value of  $\pi$  in parallel using OpenMP directives. Here's a brief explanation of the parallel implementation:

- The program takes two command-line arguments:  $n$  (number of iterations) and  $p$  (number of threads).
- It then calculates  $\pi$  using the Leibniz formula, which approximates  $\pi$  as the sum of an alternating series. Each term of the series is computed in parallel using OpenMP parallelization.
- The number of iterations ( $n$ ) determines the accuracy of the approximation, and the number of threads ( $p$ ) determines the degree of parallelism.
- The result is written to an output file with a filename based on the problem name, approach name, number of iterations, and number of threads.

### 3.4 The complexity of the algorithm (Parallel)

The complexity of the algorithm can be analyzed as follows:

- **Parallelism:** The parallel implementation uses OpenMP to distribute the iterations across multiple threads, allowing for concurrent execution on multiple CPU cores. The number of threads ( $p$ ) determines the level of parallelism, with each thread computing a subset of the total iterations. However, the overhead of parallelization and synchronization may impact performance for very small values of  $n$  or large values of  $p$ .

### 3.5 Profiling using HPC Cluster (with gprof)

The screenshots of profiling using the HPC Cluster are given below

```
[202101522@gics0 Q1]$ gprof serial.out gmon.out
Flat profile:

Each sample counts as 0.01 seconds.
%   cumulative   self           self       total
time  seconds  seconds   calls   Ts/call  Ts/call  name
100.17    5.41    5.41           2     0.00    0.00   main
  0.00    5.41    0.00           2     0.00    0.00   diff

%           the percentage of the total running time of the
time         program used by this function.

cumulative  a running sum of the number of seconds accounted
seconds     for by this function and those listed above it.

self        the number of seconds accounted for by this
seconds     function alone.  This is the major sort for this
            listing.

calls       the number of times this function was invoked, if
            this function is profiled, else blank.

self        the average number of milliseconds spent in this
ms/call     function per call, if this function is profiled,
            else blank.

total       the average number of milliseconds spent in this
ms/call     function and its descendents per call, if this
            function is profiled, else blank.

name        the name of the function.  This is the minor sort
            for this listing.  The index shows the location of
            the function in the gprof listing.  If the index is
            in parenthesis it shows where it would appear in
            the gprof listing if it were to be printed.

Copyright (C) 2012 Free Software Foundation, Inc.

Copying and distribution of this file, with or without modification,
are permitted in any medium without royalty provided the copyright
notice and this notice are preserved.
```

Figure 1: Screenshot of the terminal from HPC Cluster

### 3.6 Graph of Problem Size vs Runtime

#### 3.6.1 Graph of Problem Size vs Totaltime for LAB207 PCs

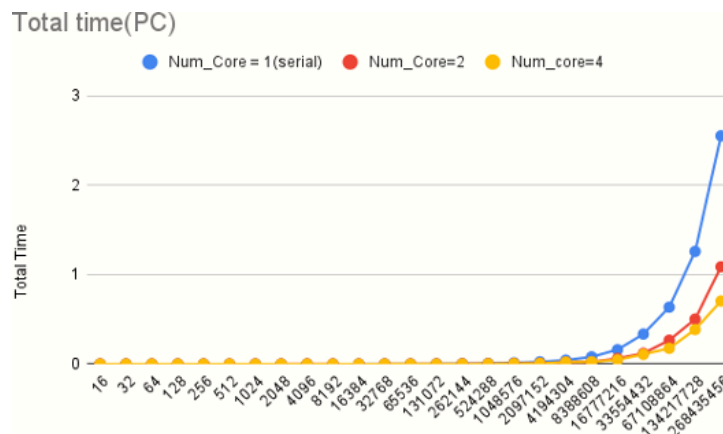


Figure 2: Total mean execution time (Total time) vs Problem size plot for **problem size  $10^8$**  (Hardware: LAB207 PC, Problem: PI\_SERIES).

#### 3.6.2 Graph of Problem Size vs Algorithm time for LAB207 PCs

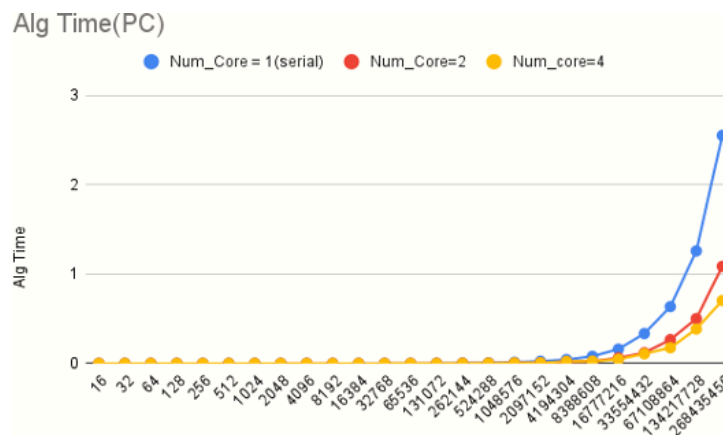


Figure 3: Total mean execution time (Algorithm time) vs Problem size plot for **problem size  $10^8$**  (Hardware: LAB207 PC, Problem: PI\_SERIES).

### 3.6.3 Graph of Problem Size vs Totaltime for HPC Cluster

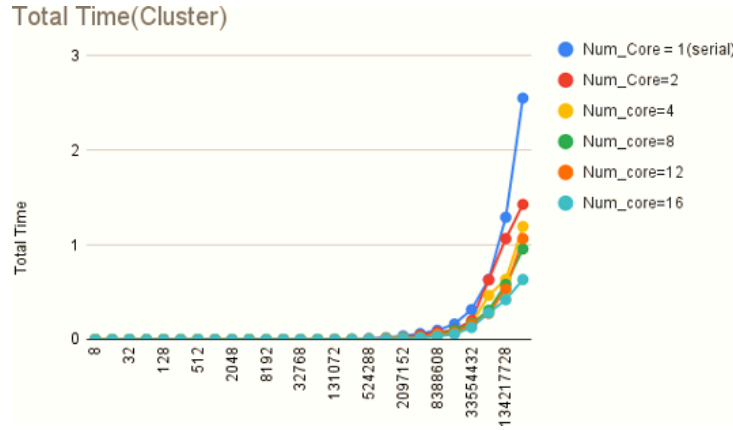


Figure 4: Total mean execution time (Total time) vs Problem size plot for **problem size  $10^8$**  (Hardware: HPC Cluster, Problem: PI\_SERIES).

### 3.6.4 Graph of Problem Size vs Algorithm time for HPC Cluster

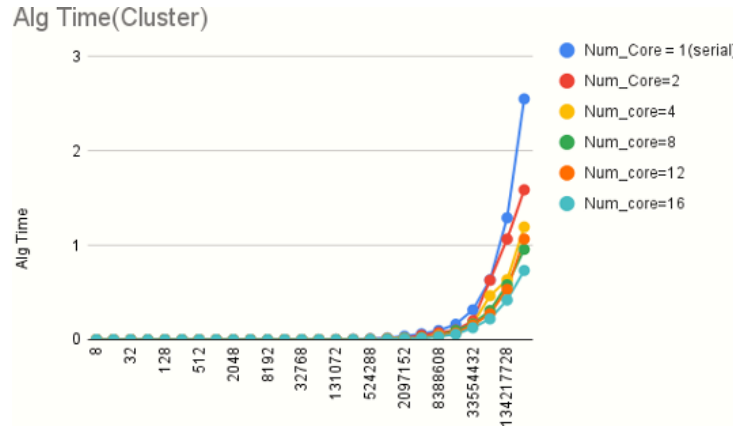


Figure 5: Total mean execution time (Algorithm time) vs Problem size plot for **problem size  $10^8$**  (Hardware: HPC Cluster, Problem: PI\_SERIES).

## 4 Conclusions

- An extensive examination of the lab PC and HPC cluster performance should shed light on how well parallel computing works for Leibniz formula-based Pi approximation, this is shown through the comparison shown in the graph. The curve depicts the information needed.
- Overall, the parallel implementation improves the performance of the algorithm by utilizing multiple threads to compute the terms of the series concurrently.