Estimating π in Parallel

A Simple Example Using the Leibniz Series

1. Introduction

We want to approximate π using the classical Leibniz series:

$$\pi = 4 \times \left(1 \; - \; \frac{1}{3} \; + \; \frac{1}{5} \; - \; \frac{1}{7} \; + \; \frac{1}{9} \; - \; \cdots\right).$$

Truncating after n terms, we can write:

$$\pi \approx 4 \times \sum_{i=0}^{n-1} \frac{(-1)^i}{2i+1}.$$

Although this series converges slowly, it serves as a simple example for parallelization.

2. Serial Code

A straightforward serial implementation in C-like pseudocode might be:

```
double sum = 0.0;
double factor = 1.0; // (+1 or -1)

for (int i = 0; i < n; i++) {
    sum += factor / (2*i + 1);
    factor = -factor;   // Flip sign
}

double pi_estimate = 4.0 * sum;
```

Each iteration alternates between adding and subtracting the fraction $\frac{1}{2i+1}$.

3. Parallel Issue

If we try to parallelize by creating multiple threads that add to a `sum` shared variable:

```
sum += factor / (2*i + 1);
```

we risk a **race condition** if two threads do their read/add/write operations on `sum` at the same time. The final result can become inaccurate and inconsistent from one run to another.

3.1 Correct Approaches

- **Local Sums**: Let each thread compute its own partial sum in a local variable, then add all partial sums to a global total *once* after threads finish.
- **Synchronization**: Use a mutex or lock around the critical section:

```
pthread_mutex_lock(&mutex);
sum += local_value;
pthread_mutex_unlock(&mutex);
```

Ensuring only one thread updates `sum` at a time.

4. Conclusion

Estimating π with the Leibniz series is mathematically straightforward but slow to converge. Even so, it illustrates how **unprotected shared variables** in a parallel program can lead to incorrect results. By introducing *local partial sums* or proper *mutex locks*, we can safely parallelize the summation and still obtain a correct approximation of π.