Parallel computation of π decimals

In this small report, we will describe how to approximate π up to a certain number of decimals (chosen by the user when running the program).

The Leibniz formula gives $\pi = 4\ arctan(1) = 4\sum_{k=0}^{\infty} \frac{(-1)^k}{2k+1}$.

Let us define f such as $f(k) = 4\ \frac{(-1)^k}{2k+1} \quad k \in \mathbb{N}$.

As $|f(k)| > |f(k+1)| \quad \forall k \in \mathbb{N}$, and that this sum is obviously an alternating one, we can claim that it converges towards a real number S (which happens to be π) : this is a well-known result known as the Leibniz test (or in French: Theoreme de convergence des series alternees).

Let us denote $\sum_{k=0}^{n} \frac{4 \times (-1)^k}{2k+1}$ as S_n , and $\sum_{k=n+1}^{\infty} \frac{4 \times (-1)^k}{2k+1}$ as R_n.

We have that, $\forall n \in \mathbb{N}, \quad S = S_n + R_n$.

The "Theoreme de convergence des series alternees" also states that we have, $\forall n \in \mathbb{N}, \ \ |R_n| < |f(n+1)|$ i.e R_n is bounded by $|f(n+1)|$.

If we take n such that $|f(n+1)| < 10^{-i}$, we then have $|R_n| = |S - S_n| < 10^{-i}$, i.e $S_n \in [S - 10^{-i}, S + 10^{-i}]$.

Under this condition S_n gives an approximation of π with $i-1$ exact decimals, and an error between of order 1 on the i^{th} decimal (that is, for example, if the i^{th} decimal is supposed to be a 4, our computation will return a number between 3 and 5).

This is the criterium we will use to know where to stop the computation of the sum, depending how many exact decimals we want:

$$|f(n+1)| < 10^{-i} \ \Leftrightarrow\ \tfrac{4}{2n+3} < 10^{-i} \ \Leftrightarrow\ \tfrac{4 \times 10^i - 3}{2} < n \ \Leftrightarrow\ 2 \times 10^i - \tfrac{3}{2} < n$$

If we compute S_n with n such that the previous inequality is verified, then we will have an exact approximation of π up to $i-1$ decimals.

Now, we will describe how to parallelize the computation of this sum so that it gives an approximation of π up to $i-1$ decimals, where i is given as an argument, by the user when running the program.

We can see that we roughly need 2×10^i terms of the sum to reach such precision. It it obvious that we cannot parallelize the computation of the sum by asking different processes to compute the value of different decimals, as the length of the computations are very inequal (indeed, this sum converges very very slowly...).
Therefore we will just equally split the computation of the sum amongst all the processes.

To get $i-1$ exact decimals, I decided we will have to compute a sum of 2×10^i terms (because it has many obvious divisors that we can take as the number of processes to run the computation with, such that the sum is split across them simply and smoothly).

If the number of processes is 'P', and if the rank of a process is 'rank', each process will compute a sum of $\frac{2 \times 10^i}{P}$ terms and store the result in a variable "partial sum". In the program, we can see I used the rank as a way to tell which part of the sum each process has to compute.

Finally, I do a reduction with add in process of rank 0, that will compute the result in variable "total sum" and return it on the screen.