AKDENIZ UNIVERSITY

*DEPARTMENT OF COMPUTER ENGINEERING*

****

*Parallel Computing – Midterm Project*

*Temmuz Burak Yavuzer*

*20160808026*

***Problem***

1. Write a multithreaded code in either C or Java that computes the pi number using Taylor series.



1. Your solution should ask the total number of operations at the beginning of the execution Do not use a static number.
2. The number of compute nodes (threads) will be dynamic. It will be asked from the user during the execution.
3. You must send a number or a range to the compute nodes.
4. Your application should also display the approximation error and duration of the computation.

**What is Leibniz Formula for Pi**

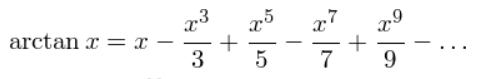
In mathematics, the Leibniz formula for π, named after Gottfried Leibniz, states that



an alternating series. It is also called the Madhava–Leibniz series as it is a special case of a more general series expansion for the inverse tangent function, first discovered by the Indian mathematician Madhava of Sangamagrama in the 14th century, the specific case first published by Leibniz around 1676.

The series for the inverse tangent function, which is also known as Gregory's series, can be given by:

The Leibniz formula for



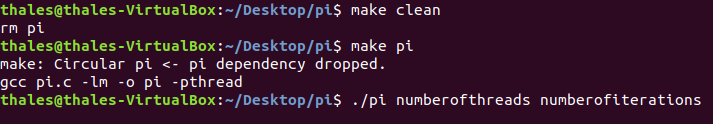
π/4 can be obtained by putting x = 1 into this series.

***System and Environment that Used in the Experiment***

In my experiment I used my computer which have 4 core. Environment is Ubuntu x64 18.04.4 via VirtualBox 6.1 which is using 4 GB of Ram with number of 4 core CPU,Video Memory is 128 MB of GTX1060.

According to these specifications the number of threads can be maximum 4.Parallel computation is increased consecutive form 1 to 4.

***Installation and Usage of the Code***

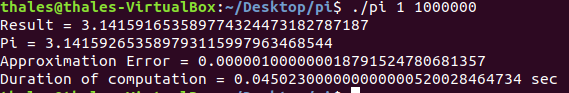
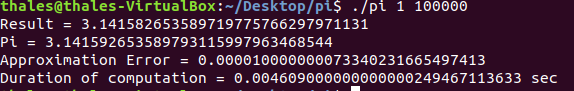
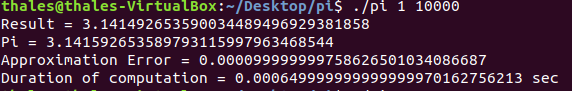
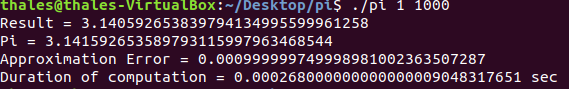


To use the code in Ubuntu we should write “make pi” in terminal.After that to execute the code we should use “./pi numberofthreads numberofiterations” command in terminal.

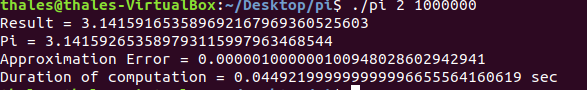
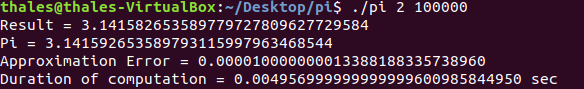
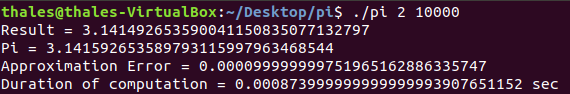
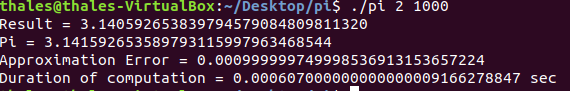
e.g “./pi 4 1000000”

***Each Results for Each Number of Threads***

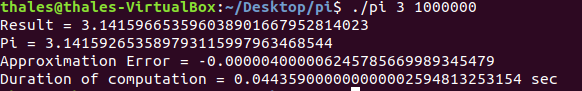
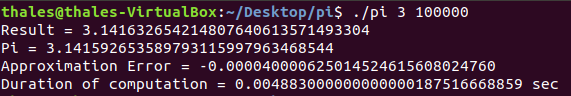
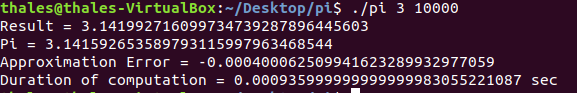
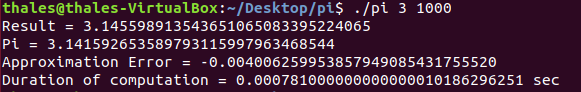
***With 1 Thread***



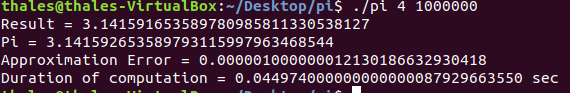
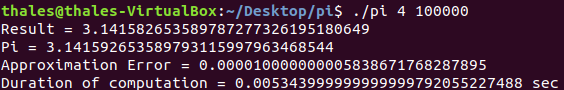
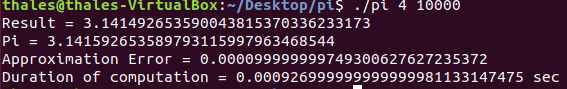
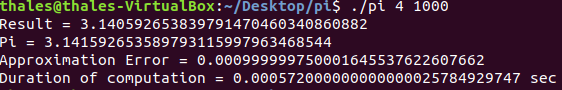
***With 2 Thread***



***With 3 Thread***



***With 4 Thread***



***CONCLUSION***

When we look at the result obtained with 1 thread ,Error of the 1000 iteration is too high but Duration of computation is relatively small when we compare with the 10000 iteration. When we keep the thread number to 1 and increased the number of iteration in multiplies of 10 every result getting better and better but the Duration of computation is getting higher.

When we look at the results obtained with 2 thread, Error of the 1000 iteration is less then Error of the 1000 iteration with the 1 thread but we sacrifice the time. Duration of the computation double of the duration of the computation with 1 thread.

When we look at the all the result of each thread with 1.000.000 iteration. Nearly all the Results very close the each other. Approximation Errors can be ignored because all the errors are too little to consider.

p.s I used my Ubuntu via VirtualBox as a operating system. While running ubuntu, there were other programs and another operating system was running. Because of that you can get different and better results while running the code.