Parallel Numerical Integration with Open MPI
Lab #1

Jason Boutte

March 13, 2015

0.1 Finding n

I found n by letting x be a number divisible by 2, e.g. 4096. The approximate integral is calculated by adding x to a variable and using it as the number of trapezoids. The result is used to calculate the absolute relative true error, and compared to 5.0^{-13}. If the absolute relative true error is not less than or equal to the value then x is added to the variable and the process is repeated. If the error is less than or equal to the value then x is subtracted from the variable and x is then divided by 2. At this point we've found a minimum and the whole process is repeated until x is equal 1, in which case we've found a minimum number of trapezoids to result in an approximate integral value with 14 correct significant digits. The value I used for n was 2387968.

0.1.1 Absolute relative error

I used absolute relative true error. My n value of 2,387,968 resulted in an absolute relative true error of 1.1358118166260693^{-14}.

0.2 Timings

Run	Time (Seconds)
1	4.494059^{-01}
2	4.506230^{-01}
3	4.497890^{-01}
4	4.497440^{-01}
5	4.500141^{-01}

Table 1: 1 Process

Run	Time (Seconds)
1	2.253621^{-01}
2	2.248940^{-01}
3	2.248309^{-01}
4	2.248290^{-01}
5	2.251148^{-01}

Table 2: 2 Processes

Run	Time (Seconds)
1	5.777979^{-02}
2	5.934191^{-02}
3	5.668807^{-02}
4	5.980706^{-02}
5	5.667305^{-02}

Table 3: 8 Processes

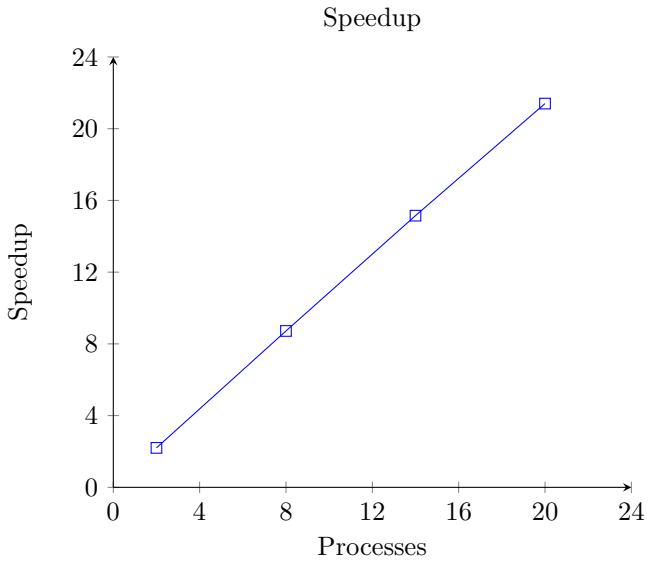
Run	Time (Seconds)
1	3.507686^{-02}
2	3.340387^{-02}
3	3.262115^{-02}
4	3.407717^{-02}
5	3.264308^{-02}

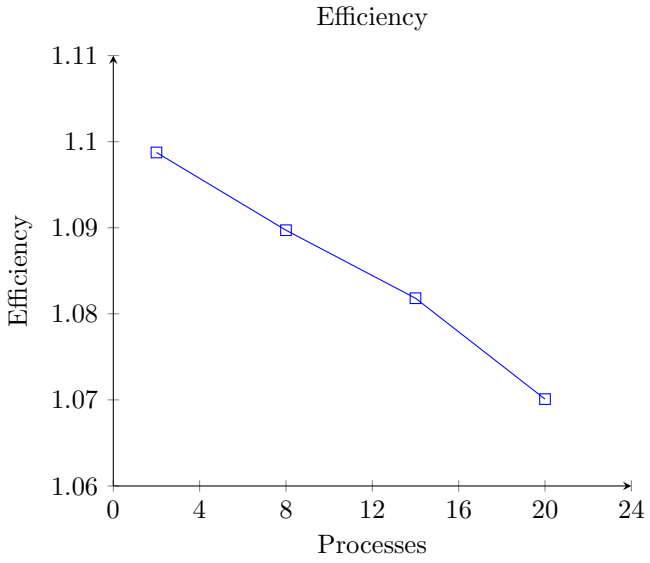
Table 4: 14 Processes

Run	Time (Seconds)
1	2.458596^{-02}
2	2.370000^{-02}
3	2.407598^{-02}
4	2.467895^{-02}
5	2.308488^{-02}

Table 5: 20 Processes

0.3 Speedup and Efficiency





0.4 Conclusion

The speed at which you can approximate an integral using the Trapezoidal Method greatly benefits from parallelization. By simply doubling the processes performing the integration the speed was doubled and by utilizing 20 processes the speed was approximately 20 times faster than a single process. But all this speed comes at a cost, a loss of accuracy. The accuracy loss at 2 and 8 processes wasn't very great, we lost 1 significant digit of the 14 we were looking at. But once the processes reached 14 and 20 the loss was far greater, losing almost 9 significant digits of the 14. This loss of accuracy is most likely caused by the software rounding the floating point values during the reduce call. Though our hardware is capable of calculating floating point at high precision, the software communicates these values at a lower precision. This explanation maps well to our observation. If we lose 2 significant values each time we add a process we can see how a few processes wouldn't have great losses but 20 processes each losing 2 values would add up quickly. From this lab I've seen the power of parallelizing large problems but also witnessed the drawbacks. When solving problems using these tools we must keep in mind that loss in precision may occur. We must account for it in the design and decide an acceptable amount of loss and weigh the benefits of speed versus accuracy.

0.5 Discussion

Utilizing 20 processes saw a huge loss in precision due to the software rounding floating point numbers. I decided the best way to counter this would be to have each process calculate its value at a higher precision to negate the loss in com-

munication. After multiple attempts I found that around 4,400,000 trapezoids could bring my absolute relative true error back down to an acceptable value. Using 20 processes I was now only losing 1 significant value of the 14. While this worked for 20 processes it can be seen that increasing the number of trapezoids and processes would still be bottlenecked by rounding.