ME 614: HPC Homework

Tanmay C. Shidhore

22/02/2017

1 Part a

The plot shown below shows the variation of the absolute Truncation Error with inverse grid spacing for 1, 2, 4, 8 and 16 processors

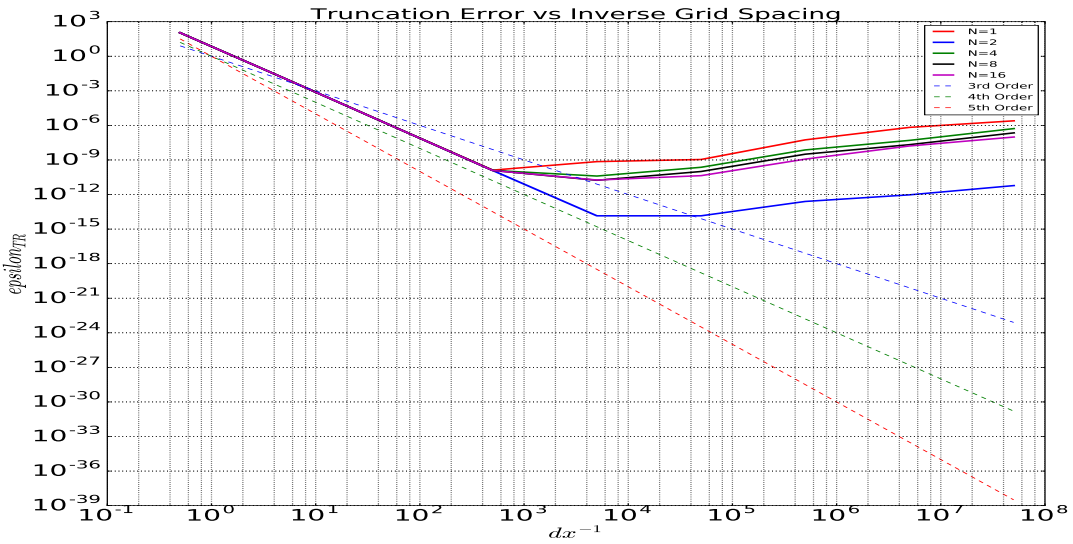

Figure 1: Variation in Absolute Truncation Error with Inverse Grid Spacing for 1,2,4,8 and 16 processors

As seen from the plot, the truncation error decreases initially with an order of accuracy=4, as the grid is refined and starts increasing beyond a certain critical grid spacing. The behaviour or the truncation error doesn't change significantly with increase in the number of processors (Np=2 behaves a bit differently, however, the cause for that is the method in which Simpson's rule is implemented in the code. On an overall basis, all plots have a similar trend, showing an initial decrease and an increase beyond a critical grid spacing value). The reason for this increase in truncation error can be attributed to the pile up of numerical error during each addition operation performed as a part of the integral evaluation. Each quantity in the integral evaluation is being forced to conform to a float128 type. While this does not affect the calculations for coarser grids (less number of operations), This forced truncation of values at every addition starts to pop up as the grid size (and subsequently the number of individual addition operations) is reduced. This phenomenon also precludes the truncation error from reaching the optimal precision. Float128 can accommodate 16 decimal places, but the truncation error (with the exception of Np=2, where the arithmetic seems to allow an error as low as 10^{-14}) only touches 10^{-10}. This happens due to the forced conversion of values to float128 at the end of every operation, which starts poisoning the more significant digits as the number of operations increases.

2 Part b

The following plots show the weak and strong scaling efficiencies for the processors. For the strong scaling part, $10^8 + 1$ points were used and for the weak scaling part, 10^7 points were provided to each processor. The strong and weak scaling efficiencies were calculated as-

$S.efficiency(\%) = t_1 * 100/(N * t_n)$

$W.efficiency(\%) = t_1 * 100/T_n$

where,
t_1 = Time required to complete the task with one processor
n = Number of Processor used, t_n = Time required to complete the same task by n processors
T_n = Time required to complete n such tasks by n processors

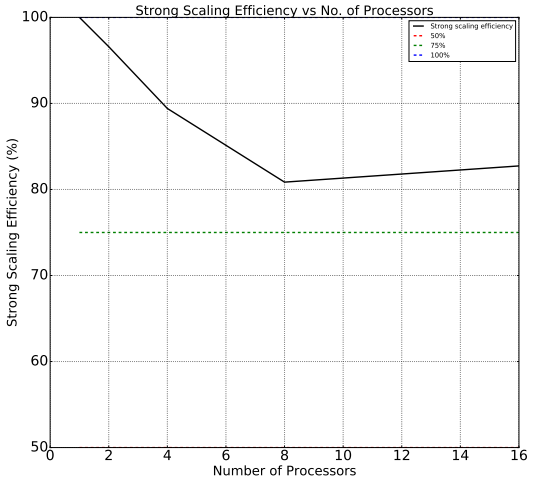

Figure 2: Strong Scaling Efficiency for $10^8 + 1$ points

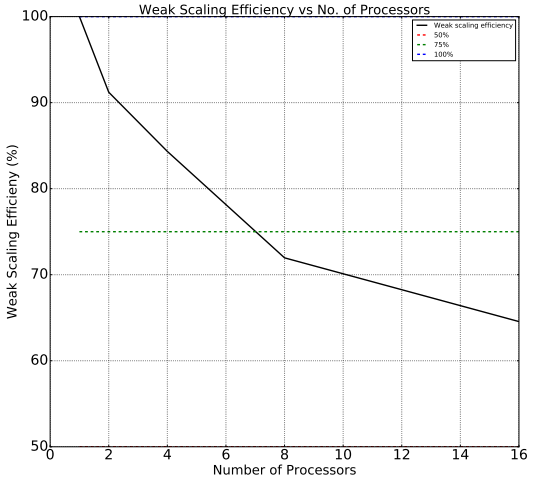

Figure 3: Weak Scaling Efficiency for 10^7 points per processor

Strong scaling is an indication of the scalability of program to multiple processors, whereas weak scaling

indicates the loss in efficiency due to communication overhead among equally loaded processors. From the plots, it can be seen that the strong scaling efficiency plateaus (approximately) around 80%, meaning that adding more than 8 processors does not significantly affect the performance in any way. This would indicate that from the given data, 8 processors would be the optimal number of processors as we are getting a nearly constant scaling efficiency (with respect to linear scaling) even after doubling the number of processors. From the second graph, it can be seen that the weak scaling efficiency decreases significantly as the number of processors are increased. This means that communication time between processors is an important factor, which has a sizeable effect on the computation time, as more processors are added.

3 Part c

The following plot shows the variation of the truncation error with the number of processors for $10^5 + 1$ points

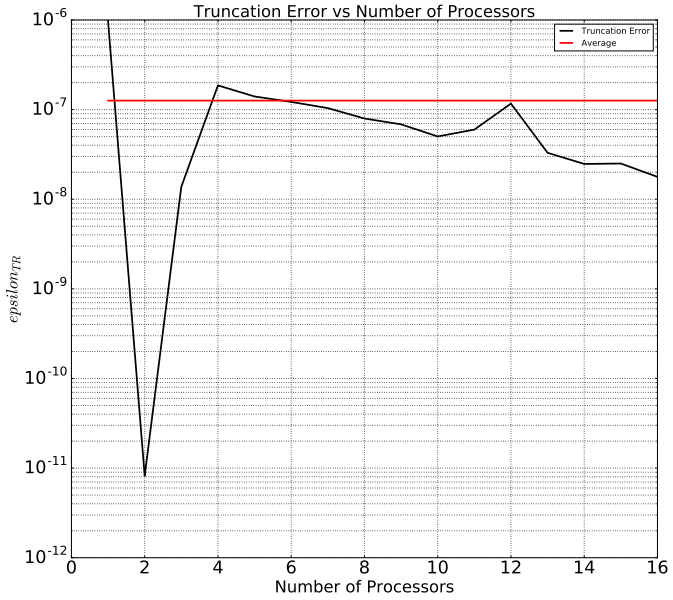

Figure 4: Absolute Truncation Error for for $10^5 + 1$ points

It can be seen that the truncation error does vary, although not significantly (except for Np=2) as one involves more processors. The variation in the truncation error can be due to the fact that the division of the integral evaluation over a larger number of processors automatically increases the number of addition operations needed at the end (during MPI.SUM). The order in which the individual results get added can be quite random, and this, in addition to the forced truncation to float128 can introduce minor variations (in the 6^{th} or 7^{th} decimal place as seen from the plot) in the final numerical result, resulting in a fluctuation in the truncation error. It should also be noted that for higher (100) processors, the round-off error might also start surfacing as a significant contribution to the truncation error variation.