MATH 440: Assignment 2
Josh Manis
Due: March 22, 2018

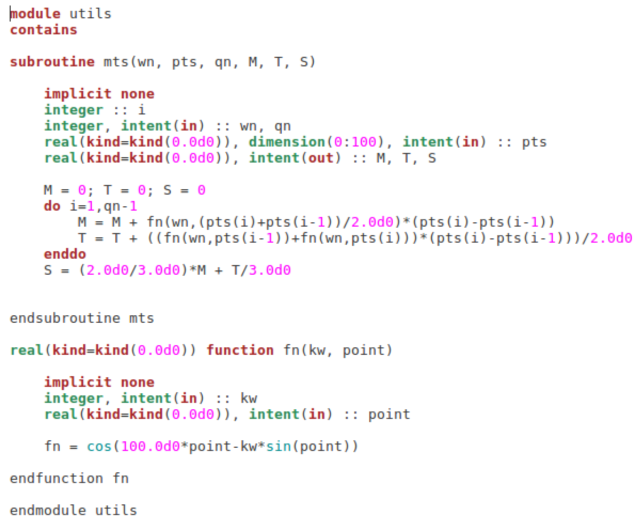
The goal of this assignment was to approximate the definite integral of a highly oscillatory function on an interval of the real number line through difference methods using parallel computing. Let $K \in \{100, 101, \ldots, 10000\}$ be a fixed wavenumber and $f(K, \cdot) : [0, \pi] \subset \mathbb{R} \mapsto \mathbb{R}$, then define

$$H(K) := \int_0^\pi f(K, x)\,\mathrm{d}x \approx \sum_{i=1}^{K} \int_{a_i}^{b_i} f(K, x)\,\mathrm{d}x \approx \sum_{i=1}^{K} A(K, i),$$

where $A(K, i)$ is an approximation to the partial integral of $f(K, \cdot)$ on $[a_i, b_i] \subset \mathbb{R}$ of width π/K. Now take 101 quadrature points, $x_{i,j}$, on $[a_i, b_i]$, in order to compute $A(K, i)$ using the midpoint method, $A_{M,101}(K, i)$, the trapezoid method, $A_{T,101}(K, i)$, and Simpson's method, $A_{S,101}(K, i)$, where

$$\begin{aligned}
a_i &= x_{i,0} < x_{i,1} < \cdots < x_{i,100} < x_{i,101} = b_i, \\
\xi_{i,j} &= \frac{x_{i,j-1} + x_{i,j}}{2}, \\
h_{i,j} &= x_{i,j} - x_{i,j-1}, \\
A_{M,101}(K, i) &:= \sum_{j=1}^{101} f\left(K, \xi_{i,j}\right) h_{i,j}, \\
A_{T,101}(K, i) &:= \sum_{j=1}^{101} \frac{1}{2} \Big[f\left(K, x_{i,j-1}\right) + f\left(K, x_{i,j-1}\right) \Big] h_{i,j}, \\
A_{S,101}(K, i) &:= \frac{2}{3} A_{M,101}(K, i) + \frac{1}{3} A_{T,101}(K, i), \\
& i \in \{1, 2, \ldots, K\},\ j \in \{1, 2, \ldots, 101\}.
\end{aligned}$$

The above was achieved with a main file and a module written in Fortran 90 compiled with mpif90. The code is as follows:



```
module utils
contains

subroutine mts(wn, pts, qn, M, T, S)

    implicit none
    integer :: i
    integer, intent(in) :: wn, qn
    real(kind=kind(0.0d0)), dimension(0:100), intent(in) :: pts
    real(kind=kind(0.0d0)), intent(out) :: M, T, S

    M = 0; T = 0; S = 0
    do i=1,qn-1
        M = M + fn(wn,(pts(i)+pts(i-1))/2.0d0)*(pts(i)-pts(i-1))
        T = T + ((fn(wn,pts(i-1))+fn(wn,pts(i)))*(pts(i)-pts(i-1)))/2.0d0
    enddo
    S = (2.0d0/3.0d0)*M + T/3.0d0


endsubroutine mts

real(kind=kind(0.0d0)) function fn(kw, point)

    implicit none
    integer, intent(in) :: kw
    real(kind=kind(0.0d0)), intent(in) :: point

    fn = cos(100.0d0*point-kw*sin(point))

endfunction fn

endmodule utils
```

```fortran
program main

    use mpi
    use utils

    implicit none
    integer :: ierr, rank, num_cores, k, i, j, num2receive, tag
    real(kind=kind(0.0d0)), parameter :: pi = 4*atan(1.0d0)
    real(kind=kind(0.0d0)) :: M, T, S, PM, PT, PS, HM, HT, HS, stime, etime
    real(kind=kind(0.0d0)), dimension(:), allocatable :: mypts, allpts
    integer, dimension(mpi_status_size) :: mystatus
    integer, dimension(:), allocatable :: num2send, displs

    call mpi_init(ierr)
    call mpi_comm_rank(mpi_comm_world,rank,ierr)
    call mpi_comm_size(mpi_comm_world,num_cores,ierr)
    stime = mpi_wtime()

    open(unit=13,file="m.dat",action="write",status="replace")
    open(unit=14,file="t.dat",action="write",status="replace")
    open(unit=15,file="s.dat",action="write",status="replace")

    if (rank == 0) then
        write(*,*) "Using", num_cores, "cores."
        write(*,"(a5, a30, a30, a30)") "K", "Midpoint_H(K)", "Trapezoidal_H(K)", "Simpson's_H(K)"
    endif
    do k = 100, 10000
        allocate(allpts(0:k))
        allocate(mypts(0:k/num_cores))
        allocate(num2send(0:num_cores-1))
        allocate(displs(0:num_cores-1))

        allpts = (/(j*pi/k,j=0,k)/)
        mypts = (/(-1.0,j=0,k/num_cores)/)
        displs = (/(0,j=0,num_cores-1)/)
        do i=0,num_cores-1
            num2send(i) = k/num_cores
            if (i<=modulo(k,num_cores)) then
                num2send(i) = num2send(i)+1
            endif
            if (i /= 0) then
                displs(i) = displs(i-1) + num2send(i)
            endif
        enddo
        num2receive = num2send(rank)

        call mpi_scatterv(allpts,num2send,displs,mpi_double_precision,&
                          mypts,num2receive,mpi_double_precision,0,&
                          mpi_comm_world,ierr)



        PM = 0; PT = 0; PS = 0
        do i=0,size(mypts)-1
            M = 0; T = 0; S =0
            if (mypts(i) > -1) then
                call mts(k,(/(mypts(i)+j*pi/100/k,j=0,100)/),101,M,T,S)
                PM = PM + M
                PT = PT + T
                PS = PS + S
            endif
        enddo
        deallocate(allpts,mypts,num2send,displs)

        call mpi_barrier(mpi_comm_world, ierr)

        tag = 0
        if (rank == 0) then
            HM = PM
            HT = PT
            HS = PS
            do i = 1, num_cores-1
                call mpi_recv(PM,1,mpi_double_precision,i,tag,mpi_comm_world,mystatus,ierr)
                call mpi_recv(PT,1,mpi_double_precision,i,tag,mpi_comm_world,mystatus,ierr)
                call mpi_recv(PS,1,mpi_double_precision,i,tag,mpi_comm_world,mystatus,ierr)
                HM = HM + PM
                HT = HT + PT
                HS = HS + PS
            enddo
        else
            call mpi_send(PM,1,mpi_double_precision,0,tag,mpi_comm_world,ierr)
            call mpi_send(PT,1,mpi_double_precision,0,tag,mpi_comm_world,ierr)
            call mpi_send(PS,1,mpi_double_precision,0,tag,mpi_comm_world,ierr)
        endif

        if (rank == 0) then
            if (modulo(k, 1000) == 0) then
                write(*,"(i5, es30.15e3, es30.15e3, es30.15e3)") k, HM, HT, HS
            endif
            write(13,*) k, HM
            write(14,*) k, HT
            write(15,*) k, HS
        endif
    enddo

    close(13); close(14); close(15)

    etime = mpi_wtime()

    if (rank == 0) then
        write(*,*) "Time taken: ", etime-stime
    endif

    call mpi_finalize(ierr)

endprogram main
```

Running the above program will compute $H(K)$ for all $K \in \{100, 101, \ldots, 10000\}$, writing output to the console when $1000 \mid K$ and writing all values of $H(K)$ to a file. Some computed values using 8 cores are as follows:

K	$H_{Mid}(K)$	$H_{Trap}(K)$	$H_{Simp}(K)$
1000	3.819159292450652E-002	3.819146669047201E-002	3.819155084649498E-002
2000	-4.844391641984440E-002	-4.844393643020983E-002	-4.844392308996622E-002
3000	-3.642203217829408E-002	-3.642200581893822E-002	-3.642202339184213E-002
4000	-1.417394345379661E-002	-1.417389887290094E-002	-1.417392859349807E-002
5000	1.237969805213174E-002	1.237974211425157E-002	1.237971273950503E-002
6000	2.957259450025162E-002	2.957262597415612E-002	2.957260499155312E-002
7000	2.671023841147271E-002	2.671025176115547E-002	2.671024286136696E-002
8000	6.324420119174937E-003	6.324415852441528E-003	6.324418696930464E-003
9000	-1.641023755211308E-002	-1.641025431321439E-002	-1.641024313914685E-002
10000	-2.484826969529740E-002	-2.484829140135555E-002	-2.484827693065011E-002

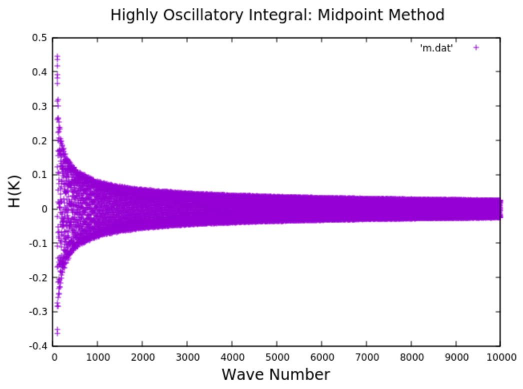
Some computed values using 16 cores are as follows:

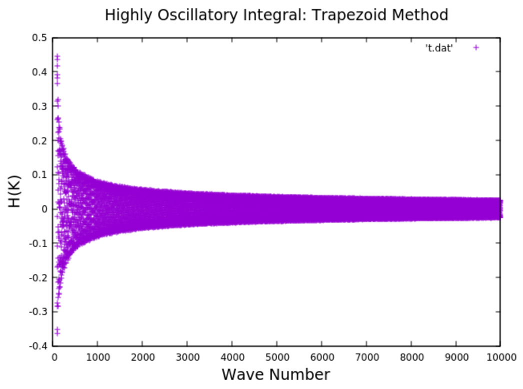
K	$H_{Mid}(K)$	$H_{Trap}(K)$	$H_{Simp}(K)$
1000	3.462382957648013E-002	3.462385379110549E-002	3.462383764802190E-002
2000	-4.837051599804793E-002	-4.837054657167911E-002	-4.837052618925834E-002
3000	-3.644134234250336E-002	-3.644134045213760E-002	-3.644134171238143E-002
4000	-1.392567124702175E-002	-1.392564881449324E-002	-1.392566376951225E-002
5000	1.336853847205303E-002	1.336853524529409E-002	1.336853739646673E-002
6000	2.958710956778173E-002	2.958714345406040E-002	2.958712086320795E-002
7000	2.713810248334630E-002	2.713810085494336E-002	2.713810194054531E-002
8000	6.025862388101258E-003	6.025892103065393E-003	6.025872293089307E-003
9000	-1.674350874433848E-002	-1.674350783603062E-002	-1.674350844156919E-002
10000	-2.521455576408334E-002	-2.521453767524605E-002	-2.521454973447090E-002

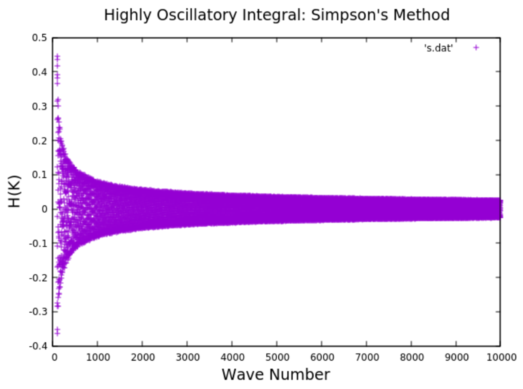
Something to note is that the computed values depend on the number of cores used. This is probably not good and is because of the way the subintervals are created and scattered to the cores. In order to compute the integral, the program creates K starting points, one for each of the K subintervals, $\left\{0, \frac{\pi}{K}, \frac{2\pi}{K}, \cdots, \frac{(K-1)\pi}{K}\right\}$. These starting points are then scattered, using `mpi_scatterv`, to each of the cores in a balanced way. The cores then loop through their starting points and evaluate an integral by creating 101 quadrature points for each starting point, $\left\{\frac{i\pi}{K}, \frac{i\pi}{K} + \frac{\pi}{100K}, \cdots, \frac{(i+1)\pi}{K}\right\}$. Each core calculates a partial sum in this way and then all of the partial sums are collected, using `mpi_recv` and `mpi_send`, by the master core which calculates the full sum. Calculation time is then calculated using `mpi_walltime`. By running the program with different numbers of cores, the complexity of the program can be determined. Time taken by number of cores used is as follows:

Number of Cores	Time to Compute (s)
1	572.56358560398803
2	303.70187161699869
4	170.86698438599706
8	92.290132593014278
16	49.949442379001994
32	30.678848117997404

On average, doubling the amount of cores used, decreases the amount of time taken by a factor of 1.8, this suggests that the complexity for this program is linear. The values of $H(K)$ and K for $K \in \{100, 101, \ldots, 10000\}$ were then graphed using `gnuplots`, these graphs are shown below:







We see that as K grows large the value of $H(K)$ appears to be approaching 0. This makes sense as we are calculating the integral of cos on $[0, \pi]$.