MAE 267 - Homework 3
Logan Halstrom
22 October 2015

Integration and timing results for runs with various numbers of processors (Table 1):

Table 1: Timing and Solver Results

Number of CPUs	Wall Time (s)	Wall Time Per Processor (s)	π Result	π Error
1	7.2956e-05	7.2956e-05	3.14159265358979356009	4.4409e-16
2	1.7095e-04	8.5473e-05	3.14159265358979400418	8.8818e-16
4	0.0013	3.3724e-04	3.14159265358979311600	0.0000e+00
8	0.0011	1.3199e-04	3.14159265358979267191	-4.4409e-16

Somehow, the solution for π with 4 processors matched Python's stored value exactly (indicated by zero error for the NCPUS=4 case.)

Solution wall time results as a function of number of processors used (Figure 1:

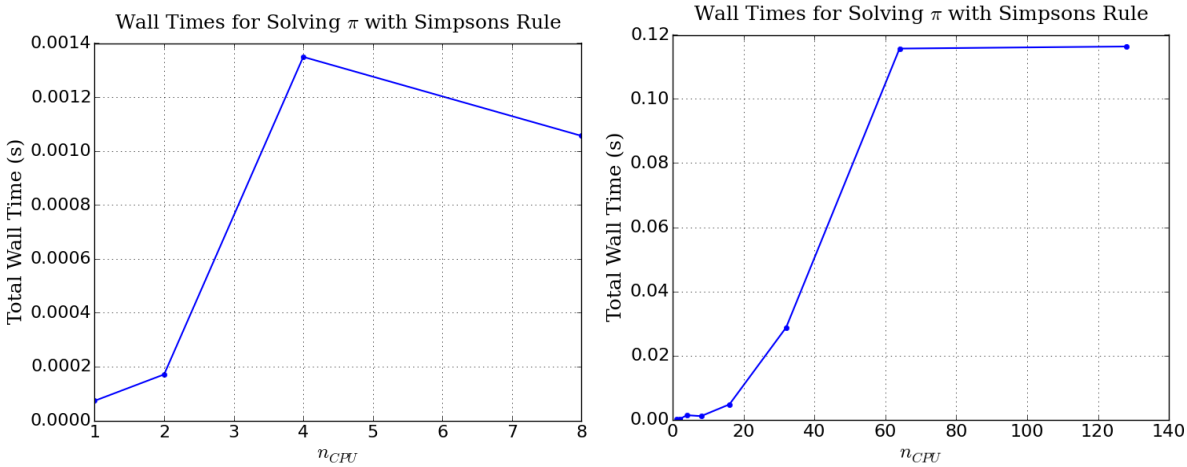

Fig. 1: Total wall times for solving π using Simpson's rule, with 1-8 CPUs on the left and 1-128 on the right

In general, increasing numbers of processors tend to increase the total wall time as more communication time is required for a given solution.

Parallel Wrapper Code

```
      PROGRAM CALCPIP
      ! TO COMPILE: mpif90 -o calcpip -O3 simp.f calcpip.f
      ! TO RUN COMPILED EXECUTABLE: mpirun -n number_of_processors executable_name
      IMPLICIT NONE

      include "mpif.h"
      REAL(KIND=8)  :: A,AK,B,BK,H,PI,SUBPI
      REAL(KIND=8) :: start, end, walltime
      INTEGER :: K,MYID,N,NK,NPROCS
      INTEGER :: IERROR,TAG,STATUS(MPI_STATUS_SIZE)

      ! INITIALIZE MPI
      CALL MPI_Init(IERROR)

      ! DETERMINE MY PROCESSOR ID
      ! ARGUMENTS: COMM, MYID, IERROR
      CALL MPI_Comm_rank(MPI_COMM_WORLD,MYID,IERROR)
```

```
18
19        ! FIND OUT HOW MANY PROCESSORS ARE USED
20        ! ARGUMENTS: COMM, NPROCS, IERROR
21        CALL MPI_Comm_size(MPI_COMM_WORLD,NPROCS,IERROR)
22
23        ! have the first processor only read user i/o
24        IF(MYID == 0) THEN
25          ! INPUT FILE
26          OPEN (UNIT = 2, FILE = 'a.in')
27          ! READ NUMBER OF SUB-INTERVALS FROM FIRST LINE
28          READ(2,*) N
29  !         !READ THE NUMBER OF SUB-INTERVALS
30  !         PRINT *,'INPUT THE NUMBER OF SUB-INTERVALS'
31  !         READ(*,*) N
32          PRINT *, 'INTEGRATING', N, 'SUB-INTERVALS'
33          PRINT *, 'RUNNING ON', NPROCS, 'PROCESSORS'
34          ! CLOCK TIME OF RUN
35          start = MPI_Wtime()
36          IF(N < NPROCS) GO TO 1000
37        END IF
38
39        ! BROADCAST THE NUMBER OF SUB-INTERVALS
40        ! ARGUEMENTS: BUFFER, COUNT, DATATYPE, ROOT, COMM, IERROR
41        CALL MPI_Bcast(N,1,MPI_INTEGER,0,MPI_COMM_WORLD,IERROR)
42
43  !     N = 10000   !DEFINE NUMBER OF INTEGRATION INTERVALS
44        A = 0.0d0   !DEFINE INTERVAL START
45        B = 1.0d0   !DEFINE INTERVAL STOP
46        H  = (B-A)/REAL(N)
47
48        ! N INTERVALS MUST BE EVENLY DIVISIBLE BY NPROCS
49        NK = N/NPROCS
50        AK = A + REAL(MYID)*REAL(NK)*H
51        BK = AK + REAL(NK)*H
52
53        ! COMPUTE LOCAL INTEGRAL
54  !      CALL TRAP(AK,BK,NK,SUBPI)
55        CALL SIMP(AK,BK,NK,SUBPI)
56
57        ! SET UP A MASTER-SLAVE RELATIONSHIP WHERE THE MASTER
58        ! IS RESPONSIBLE FOR ACCUMULATING THE SUB-INTEGRALS
59        ! AND WRITING OUT THE ANSWER
60
61        ! MYID 0 is the first processor, have it do the addition
62        IF(MYID == 0) THEN
63          ! SUM UP THE INTEGRALS FROM THE OTHER PROCESSORS
64          PI = SUBPI
65          ! ADD THE SUBPI'S FROM THE OTHER PROCESSORS
66          ! ARGUMENTS: BUFFER, COUNT, DATATYPE, SOURCE, TAG,
67          !            COMM, STATUS, IERROR
68          DO K = 1,NPROCS-1
69             CALL MPI_Recv(SUBPI,1,MPI_DOUBLE_PRECISION,K,TAG,
70       &                   MPI_COMM_WORLD,STATUS,IERROR)
71             PI = PI + SUBPI
72          END DO
73          PRINT *,'PI = ',PI
74        ELSE
75          ! SEND THE INTEGRAL TO THE MASTER
76          ! ARGUMENTS: BUFFER, COUNT, DATATYPE, DEST, TAG,
77          !            COMM, IERROR
78          CALL MPI_Send(SUBPI,1,MPI_DOUBLE_PRECISION,0,TAG,
79       &                MPI_COMM_WORLD,IERROR)
80        END IF
81
82        IF(MYID == 0) THEN
83          ! END CLOCK TIME OF RUN
84          end = MPI_Wtime()
85          walltime = end - start
86          ! OUTPUT
```

```
        OPEN (UNIT = 1, FILE = 'results.dat')
        WRITE (1,*), "Simpson's Rule"
        WRITE (1,*), "# of sub-intervals:"
        WRITE (1,*), N
        WRITE (1,*), "# of Processors:"
        WRITE (1,*), NPROCS
        WRITE (1,*), "Result for Pi:"
        WRITE (1,*), PI
        WRITE (1,*), "Total wall time (seconds):"
        WRITE (1,*), walltime
        WRITE (1,*), "Wall time per CPU (seconds):"
        WRITE (1,*), ( walltime/REAL(NPROCS) )
        CLOSE (1)
      END IF

      ! TERMINATE MPI
 1000 CALL MPI_Finalize(IERROR)

      STOP
      END
```

Simpson's Rule Subroutine

```
      SUBROUTINE SIMP(A,B,N,INTEGRAL)
      !CALCULATE INTEGRAL OF FUNCTION WITH EVEN INTERVAL (SIMPSONS RULE)
      IMPLICIT NONE
      REAL(kind=8)  :: A,B,F,H,INTEGRAL,X
      INTEGER :: N,I
      ! FUNCTION DEFINITION
      F(X) = 4.0D0/(1.D0+X**2)
      ! INTERVAL DEFINITION (dx)
         ! constant interval
      H = (B-A)/REAL(N)
      ! INITIALIZE INTEGRAL
         ! Loop goes through pairs of odd and even subscript terms
         ! Here, sum the ends of the interval, with the first odd term (x1)
         ! which wouldn't fit into the structure of the loop (a + h) is 2nd term
      INTEGRAL = F(A) + 4.0D0 * F(A + H) + F(B)
      ! LOOP THROUGH SIMPOSON RULE PAIRS
         ! loop though even numbers (2,4,6) and calc odd within loop
      DO I = 2, N-2, 2
         ! start at 3rd term (x2=A+2*dx)
         !CURRENT EVEN TERM
         X = A + REAL(I)*H
         ! add even/odd pairs for each loop iteration (i.e. 2*x2 + 4*x3, etc)
         !x is current even term, x+h is adjacent odd term
         INTEGRAL = INTEGRAL + 2.0D0 * F(X) + 4.0D0 * F(X + H)
      END DO
      INTEGRAL = H / 3.0D0 * INTEGRAL
      RETURN
      END SUBROUTINE SIMP
```

Sample Output

```
Simpson's Rule
# of sub-intervals:
        256
# of Processors:
          8
Result for Pi:
  3.1415926535897927
Total wall time (seconds):
  1.0559558868408203E-003
Wall time per CPU (seconds):
  1.3199448585510254E-004
```

Plotting Code

```
"""PLOT CPU TIMES
Logan Halstrom
21 OCTOBER 2015

DESCRIPTION:
"""

import sys
sys.path.append('/Users/Logan/lib/python')
from lplot import *
from lutil import TexTable
import os
import matplotlib.pyplot as plt
import numpy as np

def ReadData(path):
    """Read cpu times and results for given results file"""
    ifile = open(path, 'r')
    for i, line in enumerate(ifile):
        if i == 4:
            #READ NUMBER OF PROCESSORS
            nproc = int(line)
        if i == 6:
            pi = float(line)
            #READ PI
        if i == 8:
            #READ CPU WALLTIME
            walltime = float(line)
        if i == 10:
            #READ WALLTIME PER PROCESSOR
            walltimeper = float(line)
    return nproc, pi, walltime, walltimeper

def DictData(inttype, N, ncpus):
    """Read data and store in dictionary"""
    #INITIALIZE DICT OF EMPTY LISTS WITH GIVEN KEYS
    data = {k: [] for k in keys}
    for np in ncpus:
        filename = '{}{}_i{}_np{}.dat'.format(savedir, inttype, N, np)
        out = ReadData(filename)
        #MAKE LISTS FOR ALL RESULTS VARIABLES
        for o, key in zip(out, keys):
            data[key].append(o)
    return data

savedir = 'Results/'
savetype = '.pdf'
#DICTIONARY KEYS (# of procs, pi result, walltime, walltime per cpu)
keys = ['np', 'pi', 't', 'tper']
mark=10

def main():

    #TYPE OF INTEGRATION
        #'simp' or 'trap'
    inttype = 'simp'
    #NUMBER OF SUB-INTERVALS
        #integer
    N = 256
    #LIST OF NUMBER OF CPUS TO PLOT
    ncpus = [1, 2, 4, 8]

    #READ DATA
        #dictionary in a dictionary
    data = {}
    #simpson's rule results
    data['simp'] = DictData('simp', N, ncpus)
```

```
    #simpson's with lots of procs
    data['simplong'] = DictData('simp', N, [1, 2, 4, 8, 16, 32, 64, 128])
    #trapezoid rule results
    data['trap'] = DictData('trap', N, ncpus)
    for d in data.values():
        #error in calculating pi
        d['err'] = np.subtract(d['pi'], np.pi)

    #TABULATE DATA
    A = np.zeros((len(data['simp']['np']), 4))
    for i, n in enumerate(data['simp']['np']):
        for j, k in enumerate(['t', 'tper', 'pi', 'err']):
            A[i,j] = data['simp'][k][i]
    print(A)
    rows = [str(s) for s in data['simp']['np']]
    columns = ['# of\\\\CPUs', 'Wall Time (s)', 'Wall Time\\\\Per Processor',
                    '$\pi$ Result', '$\pi$ Error']
    TexTable('{}results_simp.tex'.format(savedir), A, rows, columns, 4,
                'results', 'Timing and Solver Results')


    #PLOT WALL TIME VS # OF CPUS
    for key in ['simp', 'simplong']:
        title = 'Wall Times for Solving $\pi$ with Simpson''s Rule'
        xlbl = '$n_{CPU}$'
        ylbl = 'Total Wall Time (s)'
        _, ax = PlotStart(title, xlbl, ylbl)
        ax.plot(data[key]['np'], data[key]['t'], color='blue',
                    linestyle='-', linewidth=line, marker='.', markersize=mark)
        ax.grid()
        SavePlot('{}walltimes_{}{}'.format(savedir, key, savetype))

    #PLOT WALL TIME PER CPU VS # OF CPUS
    title = 'Wall Time Per Processor for Solving $\pi$ with Simpson''s Rule'
    xlbl = '$n_{CPU}$'
    ylbl = 'Wall Time Per CPU(s)'
    _, ax = PlotStart(title, xlbl, ylbl)
    ax.plot(data['simplong']['np'], data['simplong']['tper'], color='blue',
                linestyle='-', linewidth=line, marker='.', markersize=mark)
    ax.grid()
    SavePlot('{}walltimesPer_{}{}'.format(savedir, 'simp', savetype))

    #PLOT WALL TIME VS # OF CPUS BOTH METHODS
    title = 'Wall Times for Solving $\pi$'
    xlbl = '$n_{CPU}$'
    ylbl = 'Total Wall Time (s)'
    _, ax = PlotStart(title, xlbl, ylbl)
    ax.plot(data['simp']['np'], data['simp']['t'], color='blue',
                linestyle='-', linewidth=line, marker='.', markersize=mark,
                label='Simpson''s')
    ax.plot(data['trap']['np'], data['trap']['t'], color='green',
                linestyle='-', linewidth=line, marker='x', markersize=mark,
                label='Trapezoid')
    ax.grid()
    leg = ax.legend(loc='best', fancybox=True, framealpha=0.5)
    SavePlot('{}walltimes_both{}'.format(savedir, savetype))

    #PLOT PI RESULTS
    title = 'Error In $\pi$ (Simpson''s Rule)'
    xlbl = '$n_{CPU}$'
    ylbl = 'Error (\pi_{Solver} - $\pi$)'
    _, ax = PlotStart(title, xlbl, ylbl)
    # ax.plot(data['np'], data['t'], color=color,
    #             linestyle='-', linewidth=line, marker=marker, markersize=mark,
    #             label='{} $\delta^*$'.format(self.info['name']),)
    # ax.plot([ncpus[0], ncpus[-1]], [np.pi, np.pi], color='black',
    #             linewidth=line, label='$\pi$')
    ax.plot(data['simplong']['np'], data['simplong']['err'], color='blue',
                linestyle='-', linewidth=line, marker='.', markersize=mark,
```

```
                label='Simpson''s')
    # ax.plot(data['trap']['np'], data['trap']['pi'], color='green',
    #             linestyle='-', linewidth=line, marker='x', markersize=mark,
    #             label='Trapezoid')
    ax.grid()
    # plt.ylim([min(data['simp']['pi']), max(data['simp']['pi'])])
    leg = ax.legend(loc='best', fancybox=True, framealpha=0.5)
    SavePlot('{}piErr_simp{}'.format(savedir, savetype))


if __name__ == "__main__":



    main()
```