ICS 311: Parallel and Distributed Computing

Name: P. Manohar Reddy
2022BCD0027

1) Trapezoidal Rule:

$$\int_a^b f(x)\,dx \approx (b-a)\frac{1}{2}\left[f(a)+f(b)\right]$$

Pseudocode using MPI:

```
MPI_Init();
MPI_Comm_size(MPI_COMM_WORLD, &P);
MPI_Comm_rank(MPI_COMM_WORLD &Processor);
Let the interval be [a,b]
Let N be the total no. of trapezoids
local_a = a+rank(processor)(b-a)/p
local_b = a+rank(processor+1)(b-a)/p
calculations = (b-a)/N
local_sum=0
for i=local_a to local_b -h in step increment by h
    do: local_sum += [function(i)+function(i+h))(h/2)
end for
MPI_Reduce(local_sum, global_sum, 1, MPI_Double, MPI_Sum,
                                                    0);
if processor==0 then:
        print(global_sum);
    end if
MPI_Finalize();
```

Pseudocode using Openmp:

let [a,b] be interval of f" to find area
let N be the total no. of trapezoids
work_division = (b-a)/N
global_sum = 0;
# pragma omp parallel [
    local_sum = 0;
    # pragma omp for
        for i=0 to N-1 do:
            local_sum += [f(a+i*h) + f(a+(i+1)*h)]·(h/2)
        end for
    # pragma omp critical
        global_sum += local_sum;
}
print (global_sum);

2) Simpson's Rule:

$$\int_a^b f(x)dx \approx \frac{(b-a)}{b}\left[f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right]$$

if $h = \left(\frac{b-a}{2}\right)$

$$\int_a^b f(x)dx \approx \frac{h}{2}\left[f(a) + 4f(a+h) + f(b)\right]$$

Pseudocode using MPI:

MPI_Init ();
mpi_comm_size (mpi_comm_world; & no. of pro);
mpi_comm_rank (mpi_comm_world; & rank of pro);
[et [a,b] be the interval

```
N is no. of segments.
if N is odd then:
    N+=1;
end if
Local_a = a+rank of pro*(b-a)/no.of pro;
local_b = a+(rank of pro+1)*(b-a)/no.of pro;
h= (b-a)/N;
local_sum=0;
for i=local_a to local_b i+=2 do:
    local_sum += f(a+i*h)+4*f(a+(i+1)h)+f(a+(i+2)h);
end for
MPI_Reduce (local_sum, global_sum, 1, MPI_Double, MPI_sum,0);
if rankofpro ==0 then:
    global_sum *= (h/3);
    print (global_sum);
end if
MPI_Finalize ();
```

Pseudocode using Openmp:

```
N is no-of segments [a,b] is interval
if N is odd then:
    N+=1;
end if
h= (b-a)/N;
total = 0;
```

```
#pragma omp parallel {
    sum = 0;
    #pragma omp for
    for i=0 to N-1 do:
        if i%2 == 0 then;
            sum += f(a+i(h));
        else:
            sum += 4 f(a+i*h);
    # pragma omp critical
        total += sum;
}
total *= h/3;
print (total);
```

3) Jacobian Method of solving Linear Equations:

Pseudocode using MPI:

```
MPI_Init();
MPI_comm_size (MPI_comm_WORLD, &p);
MPI_comm_rank (MPI_comm_WORLD, &id);
let us Assume matrix A has n rows,
if rank==0 then:
    divide n rows among p processes.
    let div = n/p
    local_A [div][n]
    local_v [div] // local guess
```

```
MPI_Scatter(A, div*n, MPI_Double, local_A, div*N, MPI_double,
            0, MPI_comm_WORLD);
mpi_ scatter(b, div, MPI_Double, local_v, div, MPI_Double, 0,
                                    MPI_comm_WORLD);

repeat until converge
    for each row i do:
        sum = 0
        for each element j do:
            if j!=i then:
                sum += A[i][j] * local_v[j]
        new_x[i] = (b[i] - sum)/A[i][i]
    gather new_x from all and update local_v
    if norm(new_x-local_v) < tolerance
    update local_v = new_x

    if rank == 0:
        print(local_v)

    MPI_Finalize();
```

Pseudcode using Openmp:

```
initalize A matrix, vector b, x vector is initial guess
repeat until convergence:
    #pragma omp parallel for
        for each_row i in A;
            sum=0
            for each_column j in i:
```

```
        if j != i :
            sum += A[i][j] * x[j]
    new_x[i] = (b[i] - sum) / A[i][i]

#pragma omp parallel for reduction (+: diff)
  for each i in range:
      diff += abs(new_x[i] - x[i])
  if diff < tolerance : break,
  x = new_x
print(x);
```

4) Odd-Even Sort :

Pseudocode using mpi :

```
MPI_Init();
MPI_comm_rank (MPI_COMM_WORLD, & rank);
MPI_Comm_size (MPI_COMM_WORLD, & P);
segment_size = total_size / P
MPI_Scatter (array, segment[segment_size -1], MPI_Init,
                rank+1, 0, MPI_COMM_WORLD);
for phase = 1 to num_phases do,
    if phase is odd then :
        if rank % 2 != 0 and rank < p-1 then:
            MPI_send (& local_segment[segment_size-1],,
            MPI_Init, rank+1, 0, MPI_COMM_WORLD);
```

Receive
MPI_~~Reduce~~(&incoming, 1, MPI_Init, rank+1, 0,
MPI_comm_WORLD, MPI_STATUS_IGNORE);

if incoming < local_segment[segment_size-1] then:
local_segment[segment_size-1] = incoming

else if rank%2 == 0 and rank > 0 then:
MPI_Recv(&incoming, 1, MPI_Init, rank-1, 0, mpi_comm_
WORLD, MPI_STATUS_IGNORE);
MPI_send(&local_segment[0], 1, MPI_INT, rank-1, 0,
MPI_comm_WORLD)
if incoming > local_segment[0] then:
local_segment[0] = incoming
else:
Repeat the above if else reverse
MPI_Barrier(MPI_comm_WORLD)
MPI_Gather(local_segment, segment_size, MPI_INT, array,
segment_size, MPI_INT, 0, MPI_comm_WORLD);
if rank == 0 then:
merge the sOrted segments
end if
MPI_Finalize();

Pseudocode using OpenMp:

```
#pragma omp parallel shared (array, num_elements)
  for phase = 1 to num_phases do:
    if phase is odd then:
      #pragma omp for
      for i=1 to num_elements-1 i+=2 do:
        if array[i] > array[i+1] then:
          swap array[i], array[i+1]
    else:
      #pragma omp for
        for i=0 to num_elements -2  i+=2 do:
          if array[i] > array[i+1] then
            swap array[i], array[i+1]
    #pragma omp_Barrier()
  print(array).
```