Q5 A)

Serial implementation

program poisson_serial.f90

```
   use pgm, only : pgm_write, pgm_read
   implicit none
   integer,parameter                       :: dp= selected_real_kind(15,300)
   !halo size for sim space, used to provide halo for 1D sim-data decomposition
   integer,parameter                       :: halo=2
   !key system properties
   real(kind=dp)                           ::
h,a,b,q1,x1,y1,q2,x2,y2,w,accuracy_threshold,q_max,pgm_scale_factor
   !integer step counters and grid dimensions
   integer                                 :: N,max_steps,a_int_length,b_int_length
   !integer locations on grid
   integer                                 ::
x1_int_location,y1_int_location,x2_int_location,y2_int_location
   !simulation space
   real(kind=dp),allocatable,dimension(:,:)          :: box
   !iteger sim space representation less than 1000x1000
   integer,allocatable,dimension(:,:)                :: int_box
   !boolean flag variables
   logical                                 :: debug, converged, all_phi_updates_skipped, specific_case_b
   !pgm filename variables
   character(20)                               :: out_file='phi_print.pgm',out_file_b='phi_print_b.pgm'


   !read in parameters
   call read_in()
   call check_input_values()
   call initial_properties()

   !allocate simulation space
   allocate(box(a_int_length,b_int_length))

   !set inital simulation space conditions
   box=0.0_dp
   !place charges
   box(x1_int_location+halo/2,y1_int_location+halo/2) = q1
   box(x2_int_location+halo/2,y2_int_location+halo/2) = q2

   !carry out SOR sycles untill convergence reached
   do N=1,max_steps

      if(debug.eqv..true.)then
         if(mod(N,1000_dp)==0)then
            print*,"N: ",N
         endif
      endif

      all_phi_updates_skipped=.true.

      !do a SOR pass over grid. will be MPI thread in MPI version
```

```
      call SOR_pass(all_phi_updates_skipped)

      if(all_phi_updates_skipped.eqv..true.)then

         !no changes to phi values, all at convergence, therefore SOR completed
         print*, "System converged after: ",N," SOR cycles"
         converged=.true.
         exit
      endif
   enddo

   !check if covergence or just max_steps reached
   if(converged.eqv..false.)then
      print*, "System hasn't converged after: ",max_steps," SOR cycles"
   endif

   call pgm_out()

   !deallocate sim space at end of program
   deallocate(box)

   contains

      subroutine SOR_pass(skipped_phi_updates)
         logical,intent(inout)      ::skipped_phi_updates
         integer                    :: i,j
         real(kind=dp)              :: updated_phi

         !boundary values left of of loops to maintian a grounded edge for the system
         do i=2,a_int_length-1
            do j=2,b_int_length-1

               !calcuate new phi value
               updated_phi=new_phi(box(i,j),i,j)

               !if not converged, update value in sim space and boolean flag
               if(abs(box(i,j)-updated_phi)>accuracy_threshold)then

                  box(i,j)=updated_phi
                  skipped_phi_updates=.false.
               endif

!                if(debug.eqv..true.)then
!                   print*,"New Phi value: ",updated_phi
!                   print*,"skipped_phi_update: ",skipped_phi_updates
!                endif
            enddo
         enddo

      endsubroutine

      subroutine initial_properties()
```

```
        !set default flags
        converged=.false.
        all_phi_updates_skipped=.false.
        N=0

        !find max charge value q_max
        q_max=q1
        if (q2>q_max)q_max=q2

        !convert real lengths and positions to integers
        a_int_length = int(a*h)+halo
        b_int_length = int(b*h)+halo
        x1_int_location = int(x1*h)+halo/2
        y1_int_location = int(y1*h)+halo/2
        x2_int_location = int(x2*h)+halo/2
        y2_int_location = int(y2*h)+halo/2
    endsubroutine

    subroutine pgm_out()

      if(specific_case_b.eqv..true.)then
        !print out required coords
        print*, "r_A= ",box(int(3.0_dp*h)+halo/2,int(3.0_dp*h)+halo/2)
        print*, "r_B= ",box(int(1.5_dp*h)+halo/2,int(4.5_dp*h)+halo/2)
        print*, "r_C= ",box(int(1.2_dp*h)+halo/2,int(0.2_dp*h)+halo/2)
      endif

      !allocate int array for pgm output

      !set pgm size for file, max is 999x999 so need to scale down larger grids
      allocate(int_box(int(a*h/pgm_scale_factor),int(b*h/pgm_scale_factor)))

      !routine to do correctly scaled version of the below ,such that it does not go OOB for the
pgm file
      !int_box = int(box)
      box = box*128_dp/maxval(abs(box))+128_dp

      call scale_to_int_array(pgm_scale_factor)

      if(specific_case_b.eqv..true.)then
        !write out to pgm file using pgm module from 2nd yr labs
        call pgm_write(int_box,out_file_b)
      else
        !write out to pgm file using pgm module from 2nd yr labs
        call pgm_write(int_box,out_file)
      endif
      !deallocate int sim space at end of program
      deallocate(int_box)

    endsubroutine

    subroutine scale_to_int_array(scale_factor)gfortran -Wall -Wextra -fcheck=all -c pgm.f90
```

gfortran -Wall -Wextra -fcheck=all pgm.o -gp -o poisson_serial.exe poisson_serial.f90

gfortran -Wall -Wextra -fcheck=all pgm.o -pg -fopenmp -o poisson_serial.exe poisson_serial.f90

./hpc_b.sh

serial test pc - home desktop

```
Architecture:        x86_64
CPU op-mode(s):      32-bit, 64-bit
Byte Order:          Little Endian
CPU(s):              4
On-line CPU(s) list: 0-3
Thread(s) per core:  1
Core(s) per socket:  4
Socket(s):           1
NUMA node(s):        1
Vendor ID:           GenuineIntel
CPU family:          6
Model:               94
Model name:          Intel(R) Core(TM) i5-6500 CPU @ 3.20GHz
Stepping:            3
CPU MHz:             3475.772
CPU max MHz:         3600.0000
CPU min MHz:         800.0000
BogoMIPS:            6399.96
Virtualisation:      VT-x
L1d cache:           32K
L1i cache:           32K
L2 cache:            256K
L3 cache:            6144K
NUMA node0 CPU(s):   0-3
Flags:               fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush
dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp lm constant_tsc art arch_perfmon
pebs bts rep_good nopl xtopology nonstop_tsc cpuid aperfmperf pni pclmulqdq dtes64 monitor
ds_cpl vmx smx est tm2 ssse3 sdbg fma cx16 xtpr pdcm pcid sse4_1 sse4_2 x2apic movbe popcnt
tsc_deadline_timer aes xsave avx f16c rdrand lahf_lm abm 3dnowprefetch cpuid_fault
invpcid_single pti ssbd ibrs ibpb stibp tpr_shadow vnmi flexpriority ept vpid ept_ad fsgsbase
tsc_adjust bmi1 hle avx2 smep bmi2 erms invpcid rtm mpx rdseed adx smap clflushopt intel_pt
xsaveopt xsavec xgetbv1 xsaves dtherm ida arat pln pts hwp hwp_notify hwp_act_window
hwp_epp md_clear flush_l1d
```

```
        real(kind=dp),intent(in)   :: scale_factor
        integer                    :: i,j,increment

        increment= int(scale_factor)
        !iterate over sim space, sampling at the correct interval for pgm file

        do i=halo,a_int_length-increment-halo/2,increment
           do j=halo,b_int_length-increment-halo/2,increment
```

```
            !max grey value in pgm is 255, multiply by max charge value q_max/255 to increase
constrast
            int_box(i/increment,j/increment) = int(box(i,j))

            enddo
          enddo
        endsubroutine

        real(kind=dp) function new_phi(old_phi,x_coord,y_coord)
          real(kind=dp), intent (in) ::  old_phi
          integer, intent (in)       :: x_coord,y_coord

          !calculate new_phi value at current i,j exit
          new_phi = old_phi + w*(U(x_coord,y_coord)-old_phi)

        endfunction new_phi

        real(kind=dp) function U(x_coord,y_coord)
          integer, intent (in)     :: x_coord,y_coord
          real(kind=dp)            :: charge_term

          !check if grid point is at a charge location
          !if so, set charge_term to q value
          !else charge_term=0.0
          charge_term=0.0_dp

          !check against charge 1
          if((x_coord==x1_int_location).and.(y_coord==y1_int_location))then
            charge_term=q1
          endif

          !check againts charge 2
          if((x_coord==x2_int_location).and.(y_coord==y2_int_location))then
            charge_term=q2
          endif

          !calucate U value for current coords
          U=0.25_dp*(box(x_coord+1,y_coord)+box(x_coord-1,y_coord)
+box(x_coord,y_coord+1)+box(x_coord,y_coord-1)+charge_term)

        endfunction U

        subroutine read_in()

        !routine to read input file with error handling
          read(*,*,end=200,err=800) debug
          read(*,*,end=200,err=800) specific_case_b
          read(*,*,end=200,err=800) a
          read(*,*,end=200,err=800) b
          read(*,*,end=200,err=800) q1
          read(*,*,end=200,err=800) x1
          read(*,*,end=200,err=800) y1
```

```
      read(*,*,end=200,err=800) q2
      read(*,*,end=200,err=800) x2
      read(*,*,end=200,err=800) y2
      read(*,*,end=200,err=800) h
      read(*,*,end=200,err=800) w
      read(*,*,end=200,err=800) max_steps
      read(*,*,end=200,err=800) accuracy_threshold
      read(*,*,end=200,err=800) pgm_scale_factor

      !sucessful read
      return
    200 continue
      print*,'read_in: error end of file in std. in'
      stop
    800 continue
      print*,'read_in: error in std. in'
      stop
    endsubroutine read_in

    subroutine check_input_values()

      !check input file is read correctly
      if(debug.eqv..true.)then
        print*,"Debug values"
        print*, "Debug: ",debug
        print*, "specific_case_b: ",specific_case_b
        print*, "a:",a
        print*, "b: ",b
        print*, "q1: ",q1
        print*, "x1: ",x1
        print*, "y1: ",y1
        print*, "q2: ",q2
        print*, "x2: ",x2
        print*, "y2: ",y2
        print*, "h: ",h
        print*, "w: ",w
        print*, "max_steps: ",max_steps
        print*, "accuracy_threshold: ",accuracy_threshold
        print*, "pgm_scale_factor: ",pgm_scale_factor
      endif
      return
    endsubroutine check_input_values

endprogram poisson_serial
```

Shell script
hpc_b.sh

```
#!/bin/sh
rm -f poisson.in

#define properties

#debug options
debug=true
specific_case_b=true

#box dimensions
a=4.0
b=6.0

#particle 1 properties
q1=1.0
# x1=3.0
# y1=3.0

# # part b coordinates
x1=3.1415926535897932
y1=2.7182818284590452

#particle 2 properties
q2=-2.0
# x2=2.0
# y2=5.0

# # part b coordinates
x2=1.6180339887487848
y2=4.6692016091029906

#grid size/ resolution (number of array indexes per unit box size)
# e.g. a*h = 4*100 box size = 400

h=200

#scale factor must be chosen such that a*h/pgm_scale_factor & b*h/pgm_scale_factor < 100,
#and both integers for correct .pgm production
pgm_scale_factor=2

#convergence factor - MUST be 1 <=w< 2
w=1.6

#Max iterations over box
# max_steps=100000000
max_steps=100000

#accuracy degree for convergence
accuraccy=0.0001
```

```
#create input file
echo $debug >> poisson.in
echo $specific_case_b >> poisson.in
echo $a >> poisson.in
echo $b >> poisson.in
echo $q1 >> poisson.in
echo $x1 >> poisson.in
echo $y1 >> poisson.in
echo $q2 >> poisson.in
echo $x2 >> poisson.in
echo $y2 >> poisson.in
echo $h >> poisson.in
echo $w >> poisson.in
echo $max_steps >> poisson.in
echo $accuraccy >> poisson.in
echo $pgm_scale_factor >> poisson.in

#poission SOR run
./poisson_serial.exe < poisson.in

rm -f poisson.in
```

Benchmark
convergence= 1x10-4
Flat profile:

Each sample counts as 0.01 seconds.

% time	cumulative seconds	self seconds	calls	self s/call	total s/call	name
51.28	20.13	20.13	1305600000	0.00	0.00	u.3501
25.53	30.16	10.02	1360	0.01	0.03	sor_pass.3517
22.39	38.95	8.79	1305600000	0.00	0.00	new_phi.3505
0.82	39.27	0.32	1	0.32	0.32	read_in.3499
0.00	39.27	0.00	1	0.00	39.27	MAIN__
0.00	39.27	0.00	1	0.00	0.00	check_input_values.3497
0.00	39.27	0.00	1	0.00	0.00	initial_properties.3515
0.00	39.27	0.00	1	0.00	0.00	pgm_out.3513
0.00	39.27	0.00	1	0.00	0.00	scale_to_int_array.3510

% time — the percentage of the total running time of the program used by this function.

cumulative seconds — a running sum of the number of seconds accounted for by this function and those listed above it.

self seconds — the number of seconds accounted for by this function alone. This is the major sort for this listing.

calls — the number of times this function was invoked, if this function is profiled, else blank.

self ms/call — the average number of milliseconds spent in this function per call, if this function is profiled, else blank.

total ms/call — the average number of milliseconds spent in this function and its descendents per call, if this function is profiled, else blank.

name — the name of the function. This is the minor sort for this listing. The index shows the location of the function in the gprof listing. If the index is in parenthesis it shows where it would appear in the gprof listing if it were to be printed.

Call graph (explanation follows)

granularity: each sample hit covers 2 byte(s) for 0.03% of 39.27 seconds

```
index % time    self  children    called     name
                0.00   39.27       1/1          main [2]
[1]    100.0    0.00   39.27       1         MAIN__ [1]
               10.02   28.93    1360/1360        sor_pass.3517 [3]
                0.32    0.00       1/1          read_in.3499 [6]
                0.00    0.00       1/1          check_input_values.3497 [7]
                0.00    0.00       1/1          initial_properties.3515 [8]
                0.00    0.00       1/1          pgm_out.3513 [9]
-----------------------------------------------
                                    <spontaneous>
[2]    100.0    0.00   39.27                 main [2]
                0.00   39.27       1/1          MAIN__ [1]
-----------------------------------------------
               10.02   28.93    1360/1360        MAIN__ [1]
[3]     99.2   10.02   28.93    1360         sor_pass.3517 [3]
                8.79   20.13 1305600000/1305600000     new_phi.3505 [4]
-----------------------------------------------
                8.79   20.13 1305600000/1305600000     sor_pass.3517 [3]
[4]     73.7    8.79   20.13 1305600000         new_phi.3505 [4]
               20.13    0.00 1305600000/1305600000     u.3501 [5]
-----------------------------------------------
               20.13    0.00 1305600000/1305600000     new_phi.3505 [4]
[5]     51.3   20.13    0.00 1305600000         u.3501 [5]
-----------------------------------------------
                0.32    0.00       1/1          MAIN__ [1]
[6]      0.8    0.32    0.00       1         read_in.3499 [6]
-----------------------------------------------
                0.00    0.00       1/1          MAIN__ [1]
[7]      0.0    0.00    0.00       1         check_input_values.3497 [7]
-----------------------------------------------
                0.00    0.00       1/1          MAIN__ [1]
[8]      0.0    0.00    0.00       1         initial_properties.3515 [8]
-----------------------------------------------
                0.00    0.00       1/1          MAIN__ [1]
[9]      0.0    0.00    0.00       1         pgm_out.3513 [9]
                0.00    0.00       1/1          scale_to_int_array.3510 [10]
-----------------------------------------------
                0.00    0.00       1/1          pgm_out.3513 [9]
[10]     0.0    0.00    0.00       1         scale_to_int_array.3510 [10]
-----------------------------------------------
```

This table describes the call tree of the program, and was sorted by
the total amount of time spent in each function and its children.

Each entry in this table consists of several lines. The line with the
index number at the left hand margin lists the current function.
The lines above it list the functions that called this function,

and the lines below it list the functions this one called.
This line lists:

index A unique number given to each element of the table.
Index numbers are sorted numerically.
The index number is printed next to every function name so
it is easier to look up where the function is in the table.

% time This is the percentage of the `total' time that was spent
in this function and its children. Note that due to
different viewpoints, functions excluded by options, etc,
these numbers will NOT add up to 100%.

self This is the total amount of time spent in this function.

children This is the total amount of time propagated into this
function by its children.

called This is the number of times the function was called.
If the function called itself recursively, the number
only includes non-recursive calls, and is followed by
a `+' and the number of recursive calls.

name The name of the current function. The index number is
printed after it. If the function is a member of a
cycle, the cycle number is printed between the
function's name and the index number.

For the function's parents, the fields have the following meanings:

self This is the amount of time that was propagated directly
from the function into this parent.

children This is the amount of time that was propagated from
the function's children into this parent.

called This is the number of times this parent called the
function `/' the total number of times the function
was called. Recursive calls to the function are not
included in the number after the `/'.

name This is the name of the parent. The parent's index
number is printed after it. If the parent is a
member of a cycle, the cycle number is printed between
the name and the index number.

If the parents of the function cannot be determined, the wconvergence= 1x10-4ord
`<spontaneous>' is printed in the `name' field, and all the other
fields are blank.

For the function's children, the fields have the following meanings:

self This is the amount of time that was propagated directly from the child into the function.

children This is the amount of time that was propagated from the child's children to the function.

called This is the number of times the function called this child `/' the total number of times the child was called. Recursive calls by the child are not listed in the number after the `/'.

name This is the name of the child. The child's index number is printed after it. If the child is a member of a cycle, the cycle number is printed between the name and the index number.

If there are any cycles (circles) in the call graph, there is an entry for the cycle-as-a-whole. This entry shows who called the cycle (as parents) and the members of the cycle (as children.) The `+' recursive calls entry shows the number of function calls that were internal to the cycle, and the calls entry for each member shows, for that member, how many times it was called from other members of the cycle.

Index by function name

convergence= 1x10-5
Flat profile:

Each sample counts as 0.01 seconds.

% time	cumulative seconds	self seconds	calls	self s/call	total s/call	name
50.18	229.52	229.52	1944218112	0.00	0.00	u.3501
27.38	354.73	125.21	15447	0.01	0.03	sor_pass.3517
21.48	452.96	98.23	1944218112	0.00	0.00	new_phi.3505
0.98	457.43	4.47	1	4.47	4.47	read_in.3499
0.00	457.44	0.01	1	0.01	457.45	MAIN__
0.00	457.45	0.01	1	0.01	0.01	pgm_out.3513
0.00	457.45	0.00	1	0.00	0.00	check_input_values.3497
0.00	457.45	0.00	1	0.00	0.00	initial_properties.3515
0.00	457.45	0.00	1	0.00	0.00	scale_to_int_array.3510

% time — the percentage of the total running time of the program used by this function.

cumulative seconds — a running sum of the number of seconds accounted for by this function and those listed above it.

self seconds — the number of seconds accounted for by this function alone. This is the major sort for this listing.

calls — the number of times this function was invoked, if this function is profiled, else blank.

self ms/call — the average number of milliseconds spent in this function per call, if this function is profiled, else blank.

total ms/call — the average number of milliseconds spent in this function and its descendents per call, if this function is profiled, else blank.

name — the name of the function. This is the minor sort for this listing. The index shows the location of the function in the gprof listing. If the index is in parenthesis it shows where it would appear in the gprof listing if it were to be printed.

Call graph (explanation follows)

granularity: each sample hit covers 2 byte(s) for 0.00% of 457.45 seconds

```
index % time    self  children    called     name
                0.01  457.44       1/1          main [2]
[1]    100.0    0.01  457.44       1         MAIN__ [1]
              125.21  327.75   15447/15447       sor_pass.3517 [3]
                4.47    0.00       1/1          read_in.3499 [6]
                0.01    0.00       1/1          pgm_out.3513 [7]
                0.00    0.00       1/1          check_input_values.3497 [8]
                0.00    0.00       1/1          initial_properties.3515 [9]
-----------------------------------------------
                                    <spontaneous>
[2]    100.0    0.00  457.45                 main [2]
                0.01  457.44       1/1          MAIN__ [1]
-----------------------------------------------
              125.21  327.75   15447/15447       MAIN__ [1]
[3]     99.0  125.21  327.75   15447         sor_pass.3517 [3]
              98.23  229.52 1944218112/1944218112     new_phi.3505 [4]
-----------------------------------------------
              98.23  229.52 1944218112/1944218112     sor_pass.3517 [3]
[4]     71.6   98.23  229.52 1944218112         new_phi.3505 [4]
             229.52    0.00 1944218112/1944218112     u.3501 [5]
-----------------------------------------------
             229.52    0.00 1944218112/1944218112     new_phi.3505 [4]
[5]     50.2  229.52    0.00 1944218112         u.3501 [5]
-----------------------------------------------
                4.47    0.00       1/1          MAIN__ [1]
[6]      1.0    4.47    0.00       1         read_in.3499 [6]
-----------------------------------------------
                0.01    0.00       1/1          MAIN__ [1]
[7]      0.0    0.01    0.00       1         pgm_out.3513 [7]
                0.00    0.00       1/1          scale_to_int_array.3510 [10]
-----------------------------------------------
                0.00    0.00       1/1          MAIN__ [1]
[8]      0.0    0.00    0.00       1         check_input_values.3497 [8]
-----------------------------------------------
                0.00    0.00       1/1          MAIN__ [1]
[9]      0.0    0.00    0.00       1         initial_properties.3515 [9]
-----------------------------------------------
                0.00    0.00       1/1          pgm_out.3513 [7]
[10]     0.0    0.00    0.00       1         scale_to_int_array.3510 [10]
-----------------------------------------------
```

This table describes the call tree of the program, and was sorted by
the total amount of time spent in each function and its children.

Each entry in this table consists of several lines. The line with the
index number at the left hand margin lists the current function.
The lines above it list the functions that called this function,

and the lines below it list the functions this one called.
This line lists:

index A unique number given to each element of the table. Index numbers are sorted numerically. The index number is printed next to every function name so it is easier to look up where the function is in the table.

% time This is the percentage of the `total' time that was spent in this function and its children. Note that due to different viewpoints, functions excluded by options, etc, these numbers will NOT add up to 100%.

self This is the total amount of time spent in this function.

children This is the total amount of time propagated into this function by its children.

called This is the number of times the function was called. If the function called itself recursively, the number only includes non-recursive calls, and is followed by a `+' and the number of recursive calls.

name The name of the current function. The index number is printed after it. If the function is a member of a cycle, the cycle number is printed between the function's name and the index number.

For the function's parents, the fields have the following meanings:

self This is the amount of time that was propagated directly from the function into this parent.

children This is the amount of time that was propagated from the function's children into this parent.

called This is the number of times this parent called the function `/' the total number of times the function was called. Recursive calls to the function are not included in the number after the `/'.

name This is the name of the parent. The parent's index number is printed after it. If the parent is a member of a cycle, the cycle number is printed between the name and the index number.

If the parents of the function cannot be determined, the word `<spontaneous>' is printed in the `name' field, and all the other fields are blank.

For the function's children, the fields have the following meanings:

self This is the amount of time that was propagated directly from the child into the function.

children This is the amount of time that was propagated from the child's children to the function.

called This is the number of times the function called this child `/' the total number of times the child was called. Recursive calls by the child are not listed in the number after the `/'.

name This is the name of the child. The child's index number is printed after it. If the child is a member of a cycle, the cycle number is printed between the name and the index number.

If there are any cycles (circles) in the call graph, there is an entry for the cycle-as-a-whole. This entry shows who called the cycle (as parents) and the members of the cycle (as children.) The `+' recursive calls entry shows the number of function calls that were internal to the cycle, and the calls entry for each member shows, for that member, how many times it was called from other members of the cycle.

Index by function name

OPM Implementation
poisson_omp.f90

```
program poisson_serial
   use pgm, only : pgm_write, pgm_read
   use omp_lib
   use omp_lib_kinds
   implicit none
   integer,parameter                    :: dp= selected_real_kind(15,300)
   !halo size for sim space
   integer,parameter                    :: halo=2
   !key system properties
   real(kind=dp)                        ::
h,a,b,q1,x1,y1,q2,x2,y2,w,accuracy_threshold,q_max,pgm_scale_factor
   !integer step counters and grid dimensions
   integer                              :: N,max_steps,a_int_length,b_int_length
   !integer locations on grid
   integer                              ::
x1_int_location,y1_int_location,x2_int_location,y2_int_location
   !simulation space
   real(kind=dp),allocatable,dimension(:,:)      :: box
   !iteger sim space representation less than 1000x1000
   integer,allocatable,dimension(:,:)            :: int_box
   !boolean flag variables
   logical                              :: debug, converged, all_phi_updates_skipped, specific_case_b
   !pgm filename variables
   character(20)                           :: out_file='phi_print.pgm',out_file_b='phi_print_b.pgm'

  !omp variables
   integer                              ::threads,my_thread=-1

   !establish no of OMP threads

   !$OMP PARALLEL DEFAULT(PRIVATE)
   threads = OMP_GET_NUM_THREADS()
   my_thread=OMP_GET_THREAD_NUM()
   if(debug.eqv..true.)then
      print *,"Thread Num: ",threads+1," of",threads
   endif
   !$OMP END PARALLEL


   !read in parameters
   call read_in()
   call check_input_values()
   call initial_properties()

   !allocate simulation space
   allocate(box(a_int_length,b_int_length))

   !set inital simulation space conditions
   box=0.0_dp
   !place charges
```

```
   box(x1_int_location+halo/2,y1_int_location+halo/2) = q1
   box(x2_int_location+halo/2,y2_int_location+halo/2) = q2

   !carry out SOR sycles untill convergence reached
   do N=1,max_steps

      if(debug.eqv..true.)then
         if(mod(N,1000_dp)==0)then
            print*,"N: ",N
         endif
      endif

      all_phi_updates_skipped=.true.

      !do a SOR pass over grid. will be MPI thread in MPI version
      call SOR_pass(all_phi_updates_skipped)

      if(all_phi_updates_skipped.eqv..true.)then

         !no changes to phi values, all at convergence, therefore SOR completed
         print*, "System converged after: ",N," SOR cycles"
         converged=.true.
         exit
      endif
   enddo

   !check if covergence or just max_steps reached
   if(converged.eqv..false.)then
      print*, "System hasn't converged after: ",max_steps," SOR cycles"
   endif

   call pgm_out()

   !deallocate sim space at end of program
   deallocate(box)

   contains

      subroutine SOR_pass(skipped_phi_updates)
         logical,intent(inout)      ::skipped_phi_updates
         integer                    :: i,j
         real(kind=dp)              :: updated_phi

         !boundary values left of of loops to maintian a grounded edge for the system

         !$OMP parallel do private(i,j) DEFAULT(PRIVATE) shared(a_int_length,b_int_length)
schedule(dynamic)
         do i=2,a_int_length-1
            do j=2,b_int_length-1

               !calcuate new phi value
               updated_phi=new_phi(box(i,j),i,j)
```

```
            !if not converged, update value in sim space and boolean flag
            if(abs(box(i,j)-updated_phi)>accuracy_threshold)then

               box(i,j)=updated_phi
               skipped_phi_updates=.false.
            endif

!             if(debug.eqv..true.)then
!                print*,"New Phi value: ",updated_phi
!                print*,"skipped_phi_update: ",skipped_phi_updates
!             endif
          enddo
        enddo
        !$OMP end parallel do

      endsubroutine

      subroutine initial_properties()
          !set default flags
          converged=.false.
          all_phi_updates_skipped=.false.
          N=0

          !find max charge value q_max
          q_max=q1
          if (q2>q_max)q_max=q2

          !convert real lengths and positions to integers
          a_int_length = int(a*h)+halo
          b_int_length = int(b*h)+halo
          x1_int_location = int(x1*h)+halo/2
          y1_int_location = int(y1*h)+halo/2
          x2_int_location = int(x2*h)+halo/2
          y2_int_location = int(y2*h)+halo/2
      endsubroutine

      subroutine pgm_out()

        if(specific_case_b.eqv..true.)then
          !print out required coords
          print*, "r_A= ",box(int(3.0_dp*h)+halo/2,int(3.0_dp*h)+halo/2)
          print*, "r_B= ",box(int(1.5_dp*h)+halo/2,int(4.5_dp*h)+halo/2)
          print*, "r_C= ",box(int(1.2_dp*h)+halo/2,int(0.2_dp*h)+halo/2)
        endif

        !allocate int array for pgm output

        !set pgm size for file, max is 999x999 so need to scale down larger grids
        allocate(int_box(int(a*h/pgm_scale_factor),int(b*h/pgm_scale_factor)))
```

```
        !routine to do correctly scaled version of the below ,such that it does not go OOB for the
pgm file
        !int_box = int(box)
        box = box*128_dp/maxval(abs(box))+128_dp

        call scale_to_int_array(pgm_scale_factor)

        if(specific_case_b.eqv..true.)then
           !write out to pgm file using pgm module from 2nd yr labs
           call pgm_write(int_box,out_file_b)
        else
           !write out to pgm file using pgm module from 2nd yr labs
           call pgm_write(int_box,out_file)
        endif
        !deallocate int sim space at end of program
        deallocate(int_box)

    endsubroutine

    subroutine scale_to_int_array(scale_factor)
        real(kind=dp),intent(in)   :: scale_factor
        integer                    :: i,j,increment

        increment= int(scale_factor)
        !iterate over sim space, sampling at the correct interval for pgm file

        !$OMP parallel do private(i,j) shared(increment,a_int_length,b_int_length)
DEFAULT(PRIVATE) schedule(dynamic)
        do i=halo,a_int_length-increment-halo/2,increment
           do j=halo,b_int_length-increment-halo/2,increment

           !max grey value in pgm is 255, multiply by max charge value q_max/255 to increase
constrast
           int_box(i/increment,j/increment) = int(box(i,j))

           enddo
        enddo
       !$OMP end parallel do
    endsubroutine

    real(kind=dp) function new_phi(old_phi,x_coord,y_coord)
        real(kind=dp), intent (in) ::  old_phi
        integer, intent (in)       :: x_coord,y_coord

        !calculate new_phi value at current i,j exit
        new_phi = old_phi + w*(U(x_coord,y_coord)-old_phi)

    endfunction new_phi

    real(kind=dp) function U(x_coord,y_coord)
        integer, intent (in)     :: x_coord,y_coord
        real(kind=dp)            :: charge_term
```

```
        !check if grid point is at a charge location
        !if so, set charge_term to q value
        !else charge_term=0.0
        charge_term=0.0_dp

        !check against charge 1
        if((x_coord==x1_int_location).and.(y_coord==y1_int_location))then
           charge_term=q1
        endif

        !check againts charge 2
        if((x_coord==x2_int_location).and.(y_coord==y2_int_location))then
           charge_term=q2
        endif

        !calucate U value for current coords
        U=0.25_dp*(box(x_coord+1,y_coord)+box(x_coord-1,y_coord)
+box(x_coord,y_coord+1)+box(x_coord,y_coord-1)+charge_term)

    endfunction U

    subroutine read_in()

    !routine to read input file with error handling
        read(*,*,end=200,err=800) debug
        read(*,*,end=200,err=800) specific_case_b
        read(*,*,end=200,err=800) a
        read(*,*,end=200,err=800) b
        read(*,*,end=200,err=800) q1
        read(*,*,end=200,err=800) x1
        read(*,*,end=200,err=800) y1
        read(*,*,end=200,err=800) q2
        read(*,*,end=200,err=800) x2
        read(*,*,end=200,err=800) y2
        read(*,*,end=200,err=800) h
        read(*,*,end=200,err=800) w
        read(*,*,end=200,err=800) max_steps
        read(*,*,end=200,err=800) accuracy_threshold
        read(*,*,end=200,err=800) pgm_scale_factor

        !sucessful read
        return
    200 continue
        print*,'read_in: error end of file in std. in'
        stop
    800 continue
        print*,'read_in: error in std. in'
        stop
    endsubroutine read_in

    subroutine check_input_values()
```

```
        !check input file is read correctly
        if(debug.eqv..true.)then
           print*,"Debug values"
           print*, "Debug: ",debug
           print*, "specific_case_b: ",specific_case_b
           print*, "a:",a
           print*, "b: ",b
           print*, "q1: ",q1
           print*, "x1: ",x1
           print*, "y1: ",y1
           print*, "q2: ",q2
           print*, "x2: ",x2
           print*, "y2: ",y2
           print*, "h: ",h
           print*, "w: ",w
           print*, "max_steps: ",max_steps
           print*, "accuracy_threshold: ",accuracy_threshold
           print*, "pgm_scale_factor: ",pgm_scale_factor
        endif
        return
    endsubroutine check_input_values

endprogram poisson_serial
```

Shell script
hpc_b_omp.sh

```
#!/bin/sh
rm -f poisson_mpi.in

#define properties

#debug options
debug=true
specific_case_b=true

#box dimensions
a=4.0
b=6.0

#particle 1 properties
q1=1.0
# x1=3.0
# y1=3.0

# # part b coordinates
x1=3.1415926535897932
y1=2.7182818284590452

#particle 2 properties
q2=-2.0
# x2=2.0
# y2=5.0

# # part b coordinates
x2=1.6180339887487848
y2=4.6692016091029906

#grid size/ resolution (number of array indexes per unit box size)
# e.g. a*h = 4*100 box size = 400

h=200

#scale factor must be chosen such that a*h/pgm_scale_factor & b*h/pgm_scale_factor < 100,
#and both integers for correct .pgm production
pgm_scale_factor=2

#convergence factor - MUST be 1 <=w< 2
w=1.6

#Max iterations over box
# max_steps=100000000
max_steps=100000

#accuracy degree for convergence
accuraccy=0.0001
```

```
#(MPI version)
thread_count=4

export OMP_NUM_THREADS=$thread_count

#create input file
echo $debug >> poisson_mpi.in
echo $specific_case_b >> poisson_mpi.in
echo $a >> poisson_mpi.in
echo $b >> poisson_mpi.in
echo $q1 >> poisson_mpi.in
echo $x1 >> poisson_mpi.in
echo $y1 >> poisson_mpi.in
echo $q2 >> poisson_mpi.in
echo $x2 >> poisson_mpi.in
echo $y2 >> poisson_mpi.in
echo $h >> poisson_mpi.in
echo $w >> poisson_mpi.in
echo $max_steps >> poisson_mpi.in
echo $accuraccy >> poisson_mpi.in
echo $pgm_scale_factor >> poisson_mpi.in
echo $threads >> poisson_mpi.in

#poission SOR run
./poisson_omp.exe < poisson_mpi.in

mv phi_print_b.pgm phi_print_b_threads_${thread_count}_h_${h}_w_${w}.pgm

rm -f poisson_mpi.in
```

pgm module code

```
-from yr2 compuational labs
module pgm

   implicit none
   private
   public  ::pgm_write, pgm_read

   integer           :: istat
   integer           :: out_unit = 20 !I/O variables

   contains
      subroutine pgm_write(lattice,filename)
      integer,dimension(:,:),intent(in) ::lattice
      character(len=*),intent(in)       ::filename
      integer,dimension(2)              ::dimensions
      integer                       ::j,i

      dimensions = shape(lattice)
      open(file=filename,unit=out_unit,status='replace',action='write',iostat = istat)
      if(istat /= 0) stop "Error opening file"
      !pgm magic number
      write (out_unit,11) 'P2'
      !width, height
```

```
    write (out_unit,12) dimensions(1),dimensions(2)
    !max gray value
    write (out_unit,13) 255

    !loop through grid
    do j= 1,dimensions(2)
       do i= 1,dimensions(1)
          write (out_unit,*) lattice(i,j)
          enddo
       enddo
       close (unit=out_unit,iostat = istat)
       if(istat /= 0) stop "Error closing file"

       11 format(a2)
       12 format(i3,1x,i3)
       13 format(i5)

    close(unit= out_unit,iostat = istat)
    if(istat /= 0) stop "Error opening file"
    print*, "plot complete"

endsubroutine

subroutine pgm_read(array,filename)
    integer,allocatable,dimension(:,:)::array
    character (len=*),intent(in)      ::filename
    integer,dimension(2)              ::dimensions

    open(file=filename,unit=out_unit,status='old',action='read',iostat = istat)
    if(istat /= 0) stop "Error opening file"
    !pgm magic number
    read (out_unit,11)
    !width, height
    read (out_unit,12) dimensions(1),dimensions(2)
    !max gray value
    read (out_unit,13)

    print*,dimensions

    allocate (array(dimensions(1),dimensions(2)), stat = istat)
    if(istat/=0) stop "Error allocating array"


    read (out_unit,*) array(:,:)

    close (unit=out_unit,iostat = istat)
    if(istat /= 0) stop "Error closing file"

    11 format(a2)
    12 format(i3,1x,i3)
    13 format(i5)
```

```
        print*, "read complete"


    endsubroutine
endmodule pgm
```

Compiler comands and technical details
gfortran -Wall -Wextra -fcheck=all -c pgm.f90

gfortran -Wall -Wextra -fcheck=all pgm.o -gp -o poisson_serial.exe poisson_serial.f90

To run use:
./hpc_b.sh

chmod +x hpc_b.sh to make executable

gfortran -Wall -Wextra -fcheck=all pgm.o -pg -fopenmp -o poisson_omp.exe poisson_omp.f90

To run use:
./hpc_b_omp.sh

chmod +x hpc_b_omp .sh to make executable

serial test pc - home desktop

Architecture: x86_64
CPU op-mode(s): 32-bit, 64-bit
Byte Order: Little Endian
CPU(s): 4
On-line CPU(s) list: 0-3
Thread(s) per core: 1
Core(s) per socket: 4
Socket(s): 1
NUMA node(s): 1
Vendor ID: GenuineIntel
CPU family: 6
Model: 94
Model name: Intel(R) Core(TM) i5-6500 CPU @ 3.20GHz
Stepping: 3
CPU MHz: 3475.772
CPU max MHz: 3600.0000
CPU min MHz: 800.0000
BogoMIPS: 6399.96
Virtualisation: VT-x
L1d cache: 32K
L1i cache: 32K
L2 cache: 256K
L3 cache: 6144K
NUMA node0 CPU(s): 0-3
Flags: fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp lm constant_tsc art arch_perfmon pebs bts rep_good nopl xtopology nonstop_tsc cpuid aperfmperf pni pclmulqdq dtes64 monitor ds_cpl vmx smx est tm2 ssse3 sdbg fma cx16 xtpr pdcm pcid sse4_1 sse4_2 x2apic movbe popcnt tsc_deadline_timer aes xsave avx f16c rdrand lahf_lm abm 3dnowprefetch cpuid_fault invpcid_single pti ssbd ibrs ibpb stibp tpr_shadow vnmi flexpriority ept vpid ept_ad fsgsbase tsc_adjust bmi1 hle avx2 smep bmi2 erms invpcid rtm mpx rdseed adx smap clflushopt intel_pt xsaveopt xsavec xgetbv1 xsaves dtherm ida arat pln pts hwp hwp_notify hwp_act_window hwp_epp md_clear flush_l1d

Paramiter considerations and improvements

from a series of tests shown in folder w=1.6 and h=200 were found to be the best accuracy/ computational exnese trade off for a convergense threshold of 10-4

a: 4.0000000000000000
b: 6.0000000000000000
q1: 1.0000000000000000
x1: 3.1415926535897931
y1: 2.7182818284590451
q2: -2.0000000000000000
x2: 1.6180339887487849
y2: 4.6692016091029904
h: 200.00000000000000
w: 1.6000000000000001
accuracy_threshold: 1.0000000000000000E-004

Serial code runs on a single core thus sees no paralelism improvement

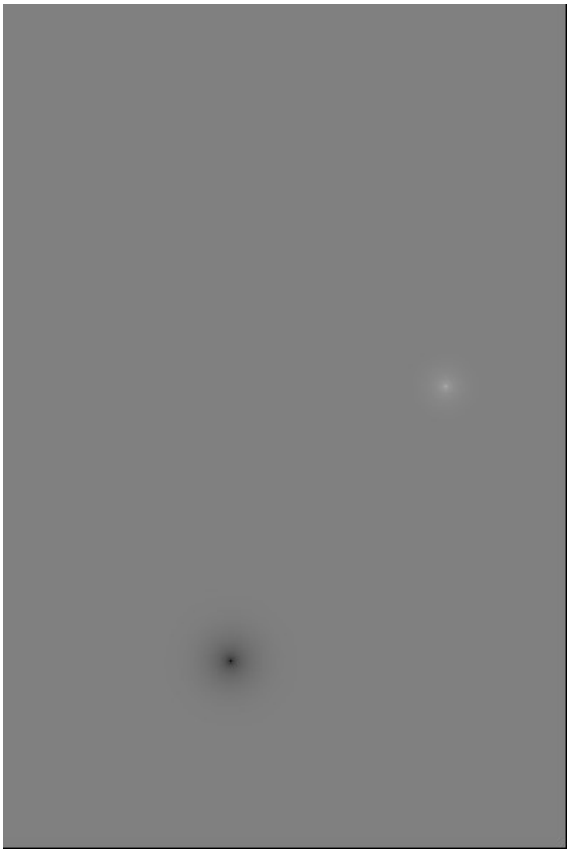
OpenMP parralelises the nested loops iterating over the system thus is faster

MPI with tata decomosition into 1D would be faster but have larger overheads for bookkeeping

B)

Convergence threshold 1x10-4

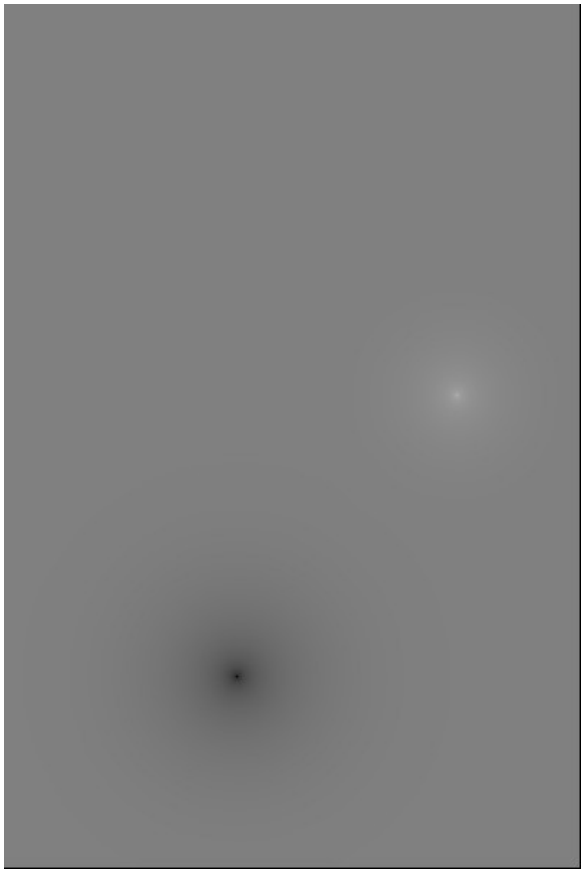
Debug values
Debug: T
specific_case_b: T
a: 4.0000000000000000
b: 6.0000000000000000
q1:   1.0000000000000000
x1:   3.1415926535897931
y1:   2.7182818284590451
q2:   -2.0000000000000000
x2:   1.6180339887487849
y2:   4.6692016091029904
h: 200.00000000000000
w: 1.6000000000000001
max_steps: 100000
accuracy_threshold: 1.0000000000000000E-004
pgm_scale_factor: 2.0000000000000000
N: 1000
System converged after: 1360 SOR cycles
r_A=   4.0952656402260738E-004
r_B= -0.14014413019377747
r_C=   0.0000000000000000
plot complete



pgm printout of sim space, for convergence threshold 10-5, black area is -2 charge, white is +1

Convergence threshold 1x10-5

Debug values
Debug: T
specific_case_b: T
a: 4.0000000000000000
b: 6.0000000000000000
q1: 1.0000000000000000
x1: 3.1415926535897931
y1: 2.7182818284590451
q2: -2.0000000000000000
x2: 1.6180339887487849
y2: 4.6692016091029904
h: 200.00000000000000
w: 1.6000000000000001
max_steps: 100000
accuracy_threshold: 1.0000000000000001E-005
pgm_scale_factor: 2.0000000000000000
N: 1000
N: 2000
N: 3000
N: 4000
N: 5000
N: 6000
N: 7000
N: 8000
N: 9000
N: 10000
N: 11000
N: 12000
N: 13000
N: 14000
N: 15000
System converged after: 15447 SOR cycles
r_A= 0.11793419137221928
r_B= -0.48080027757877947
r_C= 0.0000000000000000
plot complete



pgm printout of sim space, for convergence threshold 10-5, black area is -2 charge, white is +1