ACM40640 Assignment 1

ICHEC

Deadline: 1st March 2020 at 23:00

1 Introduction

Please carry out all sections and document the code with approapriate comments. Some tables are required, these can be included as comments in the source code or submitted as a separate file.

The assignments should be your own work. Marks will be deducted if there is copying between students or from online sources.

Please upload the source code and files to BrightSpace by the deadline.

2 Computing π by using Monte Carlo Methods

The Monte Carlo method uses a set of random points (x,y). If d is the distance from the origin then n_d is the number of points where $d <= 1$. So π can be estimated by n_d/n_t, where n_t is the total number of points. See the figure 1 below. Use the code ran2.f90(ran2.c) Fortran(C) code in Section 4 to generate the random numbers. Carry out the following parallelisation steps:

1. Construct a code that generates two random numbers x,y between $0 \rightarrow 1$.
2. Calculate the distance of the point (x,y) from the origin.
3. Use the result to estimate π, using the formula above.
4. Parallelize it by using OpenMP constructs but when updating n_d use a syncronous directive.
5. Make the function ran2 thread safe by adding OMP directives only. You should not need to alter the code. So ignore the Stackoverflow response.
6. You may find that by changing the number of threads changes the result, why?

3 Steady State

You have a 2D array w of 100×100. The first row is 0.0 the last row, first and last columns are 100 and these values remain fixed. Initially all other elements are set to 75.

$$\begin{aligned} w(1,2:99) &= 0.0 \\ w(100,1:100) &= 100.0 \\ w(1:100,1) &= 100.0 \\ w(1:100,100) &= 100.0 \\ w(2:99,2:99) &= 75.0 \end{aligned} \tag{1}$$

Repeatedly update the interior values of $w(2:99,2:99)$ with the equation below;

$$w_{new}(i,j) = (w(i+1,j)+w(i-1,j)+w(i,j+1)+w(i,j-1))/4 \tag{2}$$

Stop when the maximum difference between w and w_{new} is less than 10^{-4}.

1. Create the serial code first.

2. Parallelize the initialisation and update of w, using OpenMP constructs.
3. Note that each iteration w_{new} is independent of the previous w.
4. Use $1, 2, 4$ threads and compute absolute and relative speedup and efficiency.

4 Source Codes

```
! ran2 random generator program
!
! This function returns a random number between 0 ans 1.
! The function is not threadsafe. Below is an example of how to
! use it.
!
! REAL X
! INTEGER SEED
! X = RAN2(SEED)
!

      FUNCTION ran2(idum)

      INTEGER idum,IM1,IM2,IMM1,IA1,IA2,IQ1,IQ2,IR1,IR2,NTAB,NDIV
      REAL ran2,AM,EPS,RNMX

      PARAMETER (IM1=2147483563,IM2=2147483399,AM=1./IM1,IMM1=IM1-1,
     *IA1=40014,IA2=40692,IQ1=53668,IQ2=52774,IR1=12211,IR2=3791,
     *NTAB=32,NDIV=1+IMM1/NTAB,EPS=1.2e-7,RNMX=1.-EPS)

      INTEGER idum2,j,k,iv(NTAB),iy
      SAVE iv,iy,idum2
      DATA idum2/123456789/, iv/NTAB*0/, iy/0/

      if (idum.le.0) then
        idum=max(-idum,1)
        idum2=idum
        do 11 j=NTAB+8,1,-1
          k=idum/IQ1
          idum=IA1*(idum-k*IQ1)-k*IR1
          if (idum.lt.0) idum=idum+IM1
          if (j.le.NTAB) iv(j)=idum
11      continue
        iy=iv(1)
      endif

      k=idum/IQ1
      idum=IA1*(idum-k*IQ1)-k*IR1
      if (idum.lt.0) idum=idum+IM1
      k=idum2/IQ2
      idum2=IA2*(idum2-k*IQ2)-k*IR2
      if (idum2.lt.0) idum2=idum2+IM2
      j=1+iy/NDIV
      iy=iv(j)-idum2
      iv(j)=idum
      if(iy.lt.1)iy=iy+IMM1
      ran2=min(AM*iy,RNMX)
      return
      END
```

```
/*
 * This code generates random numbers between 0 and 1.
 * This code is not threadsafe. Below is an example of
 * how to call ran2.
 *
 *  float x;
 *  long seed;
 *  x = ran2(&seed);
*/
#define IM1 2147483563
#define IM2 2147483399
#define AM (1.0/IM1)
#define IMM1 (IM1-1)
#define IA1 40014
#define IA2 40692
#define IQ1 53668
#define IQ2 52774
#define IR1 12211
#define IR2 3791
#define NTAB 32
#define NDIV (1+IMM1/NTAB)
#define EPS 1.2e-7
#define RNMX (1.0-EPS)
float ran2(long *idum) {
        int j;
        long k;
        static long idum2=123456789;
        static long iy=0;
        static long iv[NTAB];
        float temp;
        if (*idum <= 0) {
                if (-(*idum) < 1) *idum=1;
                else *idum = -(*idum);
                idum2=(*idum);
                for (j=NTAB+7;j>=0;j--) {
                        k=(*idum)/IQ1;
                        *idum=IA1*(*idum-k*IQ1)-k*IR1;
                        if (*idum < 0) *idum += IM1;
                        if (j < NTAB) iv[j] = *idum;
                }
                iy=iv[0];
        }
        k=(*idum)/IQ1;
        *idum=IA1*(*idum-k*IQ1)-k*IR1;
        if (*idum < 0) *idum += IM1;
        k=idum2/IQ2;
        idum2=IA2*(idum2-k*IQ2)-k*IR2;
        if (idum2 < 0) idum2 += IM2;
        j=iy/NDIV;
        iy=iv[j]-idum2;
        iv[j] = *idum;
        if (iy < 1) iy += IMM1;
        if ((temp=AM*iy) > RNMX) return RNMX;
        else return temp;
} /* (C) Copr. 1986-92 Numerical Recipes Software "!15L1. */
```

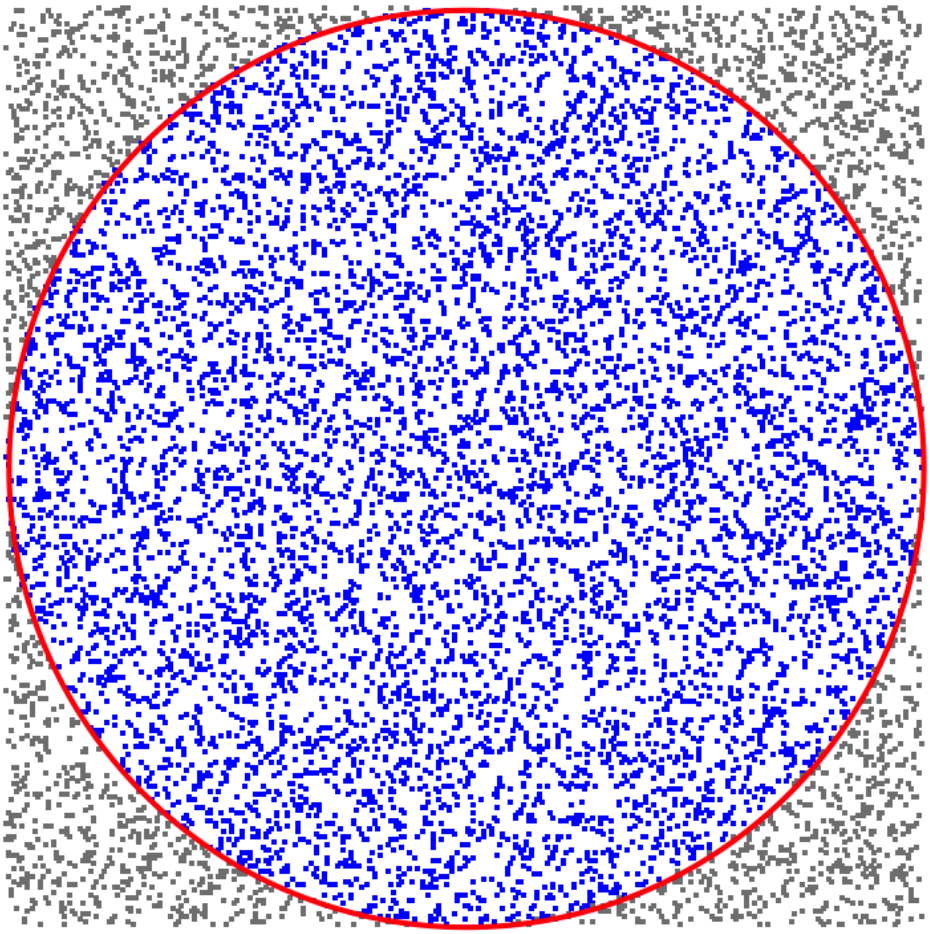

Figure 1. Calculating pi