###### L20 – MPI Example, Numerical Integration

**1. Numerical Integration**

Let’s now consider a real problem to write a parallel code for. Let’s take the case of numerical integration.

In 1-dimension the definite integral is defined as:

And we can use the mean value theorem to approximate this integral by:

Where the points *xi* should cover the entire interval of integration. In the limit of large *N*, *EN* should approach the exact value of *E*. But, this is crude.

Instead let’s consider how to do this as a Monte Carlo simulation. Monte Carlo methods typically use random sampling to investigate the problem. Hence, the term Monte Carlo comes from the Monte Carlo casino located in Monaco.

For illustration let’s calculate the integral of:

|  |
| --- |
|  |

Let’s just consider the interval from 0 to π. We can determine the value of this integral analytically as *E* = π/2 ~ 1.5708.

If we let fmax be the maximum value of the function, and xmax be the maximum range to calculate the integral over, we can define a rectangle with area xmax × fmax.

|  |
| --- |
|  |

The integral of our function would just be the area of this rectangle that occurs under the curve. That is the red area in the figure above. To use a Monte-Carlo estimate we basically just hang a picture of this function up on the wall and randomly throw darts at it. Then we tabulate the number of darts that fall above the curve and the number of darts that fall below the curve. For example, let’s assume we threw the following 5 darts, marked by circles.

|  |
| --- |
|  |

Here the two red darts fell below the curve, and the three black darts fell above the curve. We would estimate the integral as:

Not a great first guess. But, let’s keep throwing darts. Let’s do this a bit more automatically, and write a Fortran90 code to do it for us.

|  |
| --- |
| **PROGRAM monte**  **IMPLICIT NONE**  **REAL(KIND=8), DIMENSION(:), ALLOCATABLE :: xrand, yrand**  **REAL(KIND=8) :: xmax, fmax, fr, En**  **INTEGER(KIND=4) :: N, Kseed, J, Nbelow**  **N = 1000**  **xmax = 3.141592654**  **fmax = 1.0**  **ALLOCATE(xrand(N))**  **ALLOCATE(yrand(N))**  **Kseed = 1**  **CALL RANDOM\_SEED(SIZE=Kseed)**  **CALL RANDOM\_NUMBER(xrand(:))**  **CALL RANDOM\_NUMBER(yrand(:))**  **xrand(:) = xrand(:)\*xmax**  **yrand(:) = yrand(:)\*fmax**  **Nbelow = 0**  **DO J=1,N**  **fr = (cos(xrand(J)))\*\*2**  **IF (yrand(J) <= fr) THEN**  **Nbelow = Nbelow + 1**  **ENDIF**  **ENDDO**  **!estimate integral**  **En = (float(Nbelow)/float(N))\*(fmax\*xmax)**  **write(\*,\*) En**  **write(\*,\*) Nbelow**  **OPEN(UNIT=1,FILE='darts.xy')**  **DO J=1,N**  **write(1,\*) xrand(J), yrand(J)**  **ENDDO**  **CLOSE(1)**  **END PROGRAM monte** |

Here are the actual positions using 1,000 darts from the above code:

|  |
| --- |
|  |

In this case, I got 483 darts below the curve (red darts), and an estimate of the integral as:

Which is a much better estimate, but not quite good enough yet.

Now, try running the code more times and fill in the following table. Be sure to add to the code an estimate of the error.

|  |  |  |
| --- | --- | --- |
| ***N*** | ***EN*** | **Error (%)** |
| 1,000 |  |  |
| 10,000 |  |  |
| 100,000 |  |  |
| 1,000,000 |  |  |

Where the error is defined as:

Obviously, to improve on our estimate we need to throw more darts. So how can we do this efficiently? Well let’s let multiple processors work on the problem simultaneously.

**2. Parallel Monte Carlo Approach**

So, to write this as a parallel application one needs to think about how to go about the problem. In this case, it seems quite simple. We just let each processor throw the same number of darts at the dartboard, making sure that each processor is throwing a different set of darts, and tabulate all of the darts thrown that lie under the curve.

First, let’s consider for a brief moment a thing or two about random numbers. How are they generated? Are they really random? Well, the answer is in general no, because you need to have a program generate the numbers, i.e., an algorithm, and thus the numbers are ***pseudorandom***. For, example check out the cellular automaton RULE 30 from Wolfram (just Wikipedia Rule 30), which is a pretty simple way of generating random numbers.

If we call the random number generator in Fortran as we did above,

**CALL RANDOM\_NUMBER(xrand(:))**

This will return the same numbers every time. But, calling it again and again, will produce new sets of numbers. Hence, for our parallel application we will try multiple random number calls.

OK, so let’s set up the problem. If we were doing this on four processors we might think about this as follows:

|  |
| --- |
|  |

I did this as above for 4 cases, each with 1,000 dart throws with a different random seed in each case and got the # of darts below the curve as indicated.

Then, the integral estimate would be:

Which is a minor improvement. So, let’s pursue this in a more generalized sense, and write a code that allows us to use a variable number of processors. We will just modify the code above to achieve this.

|  |
| --- |
| **PROGRAM monte**  **USE mpi**  **IMPLICIT NONE**  **REAL(KIND=8), DIMENSION(:), ALLOCATABLE :: xrand, yrand**  **REAL(KIND=8) :: xmax, fmax, fr, En**  **INTEGER(KIND=4) :: N, Kseed, J, Nbelow, Ntotal, Nbtemp**  **INTEGER(KIND=4) :: mpi\_ierr, nprocs, myrank**  **INTEGER(KIND=4) :: mstatus(MPI\_STATUS\_SIZE)**  **! Initialize the MPI environment**  **!---------------------------------------------------------------------!**  **CALL MPI\_Init(mpi\_ierr)**  **CALL MPI\_Comm\_size(MPI\_COMM\_WORLD,nprocs,mpi\_ierr)**  **CALL MPI\_Comm\_rank(MPI\_COMM\_WORLD,myrank,mpi\_ierr)**  **!---------------------------------------------------------------------!**  **! Initialize parameters**  **!---------------------------------------------------------------------!**  **N = 1000**  **xmax = 3.141592654**  **fmax = 1.0**  **!---------------------------------------------------------------------!**  **!Allocate memory**  **!---------------------------------------------------------------------!**  **ALLOCATE(xrand(N))**  **ALLOCATE(yrand(N))**  **!---------------------------------------------------------------------!**  **! Populate random numbers differently for each rank**  **!---------------------------------------------------------------------!**  **Kseed = 1**  **DO J=1,(myrank+1)**  **CALL RANDOM\_SEED(SIZE=Kseed)**  **CALL RANDOM\_NUMBER(xrand(:))**  **CALL RANDOM\_NUMBER(yrand(:))**  **ENDDO**  **xrand(:) = xrand(:)\*xmax**  **yrand(:) = yrand(:)\*fmax**  **!---------------------------------------------------------------------!**  **!Now calculate Nbelow on each processor**  **!---------------------------------------------------------------------!**  **Nbelow = 0**  **DO J=1,N**  **fr = (cos(xrand(J)))\*\*2**  **IF (yrand(J) <= fr) THEN**  **Nbelow = Nbelow + 1**  **ENDIF**  **ENDDO**  **!write out value of Nbelow for each rank**  **write(\*,\*) "myrank= ", myrank, "Nbelow= ", Nbelow**  **!---------------------------------------------------------------------!**  **! Now tabulate all of the Nbelows and total throws onto rank 0**  **!---------------------------------------------------------------------!**  **! First send the messages**  **IF (myrank > 0) THEN**  **CALL MPI\_Send(Nbelow,1,MPI\_INTEGER,0,myrank,MPI\_COMM\_WORLD,mpi\_ierr)**  **ENDIF**  **! Now receive them on rank 0**  **IF (myrank == 0) THEN**  **Ntotal = N**  **DO J=1,(nprocs-1)**  **CALL MPI\_Recv(Nbtemp,1,MPI\_INTEGER,J,J,MPI\_COMM\_WORLD,mstatus,mpi\_ierr)**  **Ntotal = Ntotal + N**  **Nbelow = Nbelow + Nbtemp**  **ENDDO**  **write(\*,\*) "Ntotal= ", Ntotal, "Nbelow= ", Nbelow**  **ENDIF**  **!---------------------------------------------------------------------!**  **!estimate integral**  **!---------------------------------------------------------------------!**  **IF (myrank == 0) THEN**  **En = (float(Nbelow)/float(Ntotal))\*(fmax\*xmax)**  **write(\*,\*) "En= ", En**  **ENDIF**  **!---------------------------------------------------------------------!**  **CALL MPI\_Finalize(mpi\_ierr)**  **END PROGRAM monte** |

**3. Homework**

Calculate the following integrals: