

MCS 572: Homework 3
University of Illinois at Chicago (Professor Nicholls)
Fall 2009

Due Friday, October 30 by 2pm.

1. Modify the Monte Carlo code (`monte.c`) to approximate the value $e \approx 2.71828182845905$. Answer the following questions:
 - (a) What is the smallest value of n required to obtain accuracy of 10^{-2} ? 10^{-4} ? 10^{-10} ?
 - (b) What “speedup” do you get for $N = 4$ processes? $N = 8$ processes?
2. Write a one-paragraph summary of your proposed Final Project.
3. Exercises (Heath): 7.1, 7.5, 7.10, 7.13
4. Computer Problems (Heath): 7.3, 7.4, 7.6, 7.7

```

/* compute pi using Monte Carlo method */
#include <math.h>
#include "mpi.h"
#include "mpe.h"
#define CHUNKSIZE      1000
/* We'd like a value that gives the maximum value returned by the function
   random, but no such value is *portable*.  RAND_MAX is available on many
   systems but is not always the correct value for random (it isn't for
   Solaris).  The value ((unsigned(1)<<31)-1) is common but not guaranteed */
#define INT_MAX 1000000000

/* message tags */
#define REQUEST  1
#define REPLY    2
int main( int argc, char *argv[] )
{
    int iter;
    int in, out, i, iters, max, ix, iy, ranks[1], done, temp;
    double x, y, Pi, error, epsilon;
    int numprocs, myid, server, totalin, totalout, workerid;
    int rands[CHUNKSIZE], request;
    MPI_Comm world, workers;
    MPI_Group world_group, worker_group;
    MPI_Status status;

    MPI_Init(&argc,&argv);
    world = MPI_COMM_WORLD;
    MPI_Comm_size(world,&numprocs);
    MPI_Comm_rank(world,&myid);
    server = numprocs-1;          /* last proc is server */
    if (myid == 0)
        sscanf( argv[1], "%lf", &epsilon );
    MPI_Bcast( &epsilon, 1, MPI_DOUBLE, 0, MPI_COMM_WORLD );
    MPI_Comm_group( world, &world_group );
    ranks[0] = server;
    MPI_Group_excl( world_group, 1, ranks, &worker_group );
    MPI_Comm_create( world, worker_group, &workers );
    MPI_Group_free(&worker_group);
    if (myid == server) {        /* I am the rand server */
        do {
            MPI_Recv(&request, 1, MPI_INT, MPI_ANY_SOURCE, REQUEST,
                    world, &status);
            if (request) {
                for (i = 0; i < CHUNKSIZE; ) {
                    rands[i] = random();

```

```

        if (rands[i] <= INT_MAX) i++;
    }
    MPI_Send(rands, CHUNKSIZE, MPI_INT,
             status.MPI_SOURCE, REPLY, world);
}
}
while( request>0 );
}
else {
    /* I am a worker process */
    request = 1;
    done = in = out = 0;
    max = INT_MAX;
    /* max int, for normalization */
    MPI_Send( &request, 1, MPI_INT, server, REQUEST, world );
    MPI_Comm_rank( workers, &workerid );
    iter = 0;
    while (!done) {
        iter++;
        request = 1;
        MPI_Recv( rands, CHUNKSIZE, MPI_INT, server, REPLY,
                 world, &status );
        for (i=0; i<CHUNKSIZE-1; ) {
            x = (((double) rands[i++])/max) * 2 - 1;
            y = (((double) rands[i++])/max) * 2 - 1;
            if (x*x + y*y < 1.0)
                in++;
            else
                out++;
        }
        MPI_Allreduce(&in, &totalin, 1, MPI_INT, MPI_SUM,
                     workers);
        MPI_Allreduce(&out, &totalout, 1, MPI_INT, MPI_SUM,
                     workers);
        Pi = (4.0*totalin)/(totalin + totalout);
        error = fabs( Pi-3.141592653589793238462643);
        done = (error < epsilon || (totalin+totalout) > 1000000);
        request = (done) ? 0 : 1;
        if (myid == 0) {
            printf( "\rpi = %23.20f", Pi );
            MPI_Send( &request, 1, MPI_INT, server, REQUEST,
                     world );
        }
    }
    else {
        if (request)
            MPI_Send(&request, 1, MPI_INT, server, REQUEST,
                    world);
    }
}

```

```
        }
    }
    MPI_Comm_free(&workers);
}

if (myid == 0) {
    printf( "\npoints: %d\nin: %d, out: %d, <ret> to exit\n",
            totalin+totalout, totalin, totalout );
    getchar();
}
MPI_Finalize();
}
```