MPI Example Programs: Calculating *pi*

Table of Contents

- Quick Summary
- Description of Problems
- Parallel Implementation
- Instructions for Compiling and Running dboard_pi
- Instructions for Compiling and Running int_pi
- Cleanup

Quick Summary

There are two example programs for calculating *pi*:

- dboard_pi:

  This is a very simple parallel program in which each task independently approximates the solution. The amount of work done by each task remains the same as the number of tasks increases, but the averaged solution becomes more correct. There are two versions: one demonstrating sends and receives (pi_send) and the other using collective communications (pi_reduce).

- int_pi:

  This program calculates *pi* using an integral approximation. This code is of particular interest, because there are several programs included that illustrate the conversion of the serial code int_pi to parallel. The final version is int_pi2 (.f or .c).

Description of Problem

- dboard_pi:

  Picture a circle with radius 1 centered at the origin. The circle sits inside a square whose corners are at (-1,-1), (-1,1), (1,1) and (1,-1). The area of the circle divided by the area of the square is pi/4.

  Think of this as a dartboard. The darts hit at x and y coordinates which are random numbers between -1 and 1. Darts must fall within the square, and may also fall within the circle. The program approximates the value of pi by dividing the number of darts that fall within the circle by the total darts thrown, and multiplying by four.

  A full description of this problem is available in Fox et al., 1988, Solving Problems on Concurrent Processors, volume 1, page 207.

- int_pi:

  int_pi uses a simpe integral approximation to calculate *pi*

Parallel Implementation

- dboard_pi:

  There are two ways to benefit from parallelism: you may run the same program in less time, or run a larger program in the same amount of time. This example takes the latter approach.

  The serial calculation of pi involves throwing 5000 darts for each of ten iterations, with the cumulative average reported at each iteration. For the parallel implementation, each task performs this process independently, reporting its calculated pi value to the master task (task ID 0), which prints the cumulative average. The more tasks that participate, the more accurate the calculated value of pi.

  The code is SPMD, i.e. each task runs the same executable. There are two versions. Pi_send uses low-level sends and receives to collect the pi values. Pi_reduce uses the collective communication reduction routine, with the pre-defined reduction function for double precision floating point vector addition.

Instructions for Compiling and Running dboard_pi

1. Copy the needed files to your working directory. You may copy the files from within your browser, or use the cp command. Some browsers add or replace characters when files are copied.

   Tutorial directory: /afs/pdc.kth.se/public/www/training/Tutor/MPI/Templates/pi/dboard/

   Files:

   - pi_send.f or pi_send.c
   - pi_reduce.f or pi_reduce.c
   - dboard.f or dboard.c
   - make.pi.f or make.pi.c

2. Compile using the make file:

   For Fortran:

```
make -f make.pi.f pi_send
                or
make -f make.pi.f pi_reduce
```

For C:

```
make -f make.pi.c pi_send
                or
make -f make.pi.c pi_reduce
```

3. Specify how many nodes to run on (choose *n* <= 4):

   setenv MP_PROCS *n* (or use -procs n)

4. Execute the master program:

```
pi_send
        or
pi_reduce
```

Instructions for Compiling and Running int_pi

1. First, copy the source files to your current working directory. You may copy lab and solution files from within your browser, or use the cp command. Some browsers add or replace characters when files are copied.

   Tutorial directory: /afs/pdc.kth.se/public/www/training/Tutor/MPI/Templates/pi/int/

   Files:

   - int_pi.f or int_pi.c is the original serial version of the int_pi code.
   - int_pi1.f or int_pi1.c is the partially parallelized code converted to SPMD, but with no division of work and no message-passing.
   - int_pi2.f or int_pi2.c is the final parallelized version.
   - make.int_pi.f or make.int_pi.c is a makefile that invokes the necessary compiler options, include directories and library directories.

2. Compile the code:

```
mpxlf -o int_pi2 int_pi2.f
        or
mpcc -o int_pi2 int_pi2.c
```

   or, use the make file:

```
make -f make.int_pi.f
        or
make -f make.int_pi.c
```

3. Specify how many nodes to run on:

   setenv MP_PROCS *n* (or use -procs n)

4. Execute the master program:

   int_pi2

Cleanup

Some or all of the following files will be left in your directory, and should be removed to save space:

- dboard_pi:

```
pi_send.c          pi_send.f
pi_reduce.c        pi_reduce.f
dboard.c           dboard.f
make.pi.c          make.pi.f
pi_send
pi_reduce
```

- int_pi:

```
int_pi.f           int_pi.c
int_pi1.f          int_pi1.c
int_pi2.f          int_pi2.c
make.int_pi.f      make.int_pi.c
```

Copyright Feedback

Changed by:$Author: kinsella $,$Date: 1999/07/12 09:21:07 $