Xi XIA
2016213482
2016215117

4.12. Using Amdahl's Law, calculate the speedup gain of an application that has a 60 percent parallel component for (a) two processing cores and (b) four processing cores.

S (serial portion) = 1-parallel = 1-60% = 40%

N processing cores

$sppedup \leq \frac{1}{S+\frac{(1-S)}{N}}$

(a) Two processing cores = $\frac{1}{0.4+\frac{(1-0.4)}{2}}$ = 1.43 speedup;

(b) four processing cores = $\frac{1}{0.4+\frac{(1-0.4)}{4}}$ = 1.82 speedup.

4.15. Consider the following code segment:

```
pid_t pid;

pid = fork();
if (pid == 0) { /* child process */
   fork();
   thread_create( . . .);
}
fork();
```

How many unique processes are created?

1. *pid = fork()* before the if statement creates one process. The parent process p creates a new process p1.
2. *fork()* in the if statement creates one process. The parent process p creates a new process p2.
3. After the if statement, parent process p, process p1 and process p2 will execute *fork();* creating three new processes.
   One process p3 is created by parent process p.
   One process p4 is created by process p1.
   One process p5 is created by process p2.

Hence, 6 unique processes (p, p1, p2, p3, p4, p1, p5) are created.

How many unique threads are created?

1. Thread creation is done in if block. Child process p1 is executed in the if block. Therefore, process p1 creates one thread.
2. In the if block one process p2 is created using *fork()*. Therefore, process p2 will also create a thread.

Hence, 2 unique threads are created.

Xi XIA
2016213482
2016215117

4.17. The program shown in Figure 4.16 uses the Pthreads API. What would be the output from the program at LINE C and LINE P?

```
#include <pthread.h>
#include <stdio.h>

#include <types.h>

int value = 0;
void *runner(void *param); /* the thread */

int main(int argc, char *argv[])
{
pid_t pid;
pthread_t tid;
pthread_attr_t attr;

   pid = fork();

   if (pid == 0) { /* child process */
      pthread_attr_init(&attr);
      pthread_create(&tid,&attr,runner,NULL);
      pthread_join(tid,NULL);
      printf("CHILD: value = %d",value); /* LINE C */
   }
   else if (pid > 0) { /* parent process */
      wait(NULL);
      printf("PARENT: value = %d",value); /* LINE P */
   }
}

void *runner(void *param) {
   value = 5;
   pthread_exit(0);
}
```

Figure 4.16 C program for Exercise 4.17.

1. **LINE C:**

The output is CHILD: value = 5

The child process in the thread is forked by parent process and child process each have its own memory space.

After forking, the parent process waits for the completion of child process.

New thread is created for child process and the *runner()* function is called which set the value of the global variable to 5.

Thus, after execution of this line, the value displayed will be 5.

2. **LINE P:**

The output is PARENT: value = 0

After completing the child process, the value of the global variable present in parent process remains 0.

Thus, after execution of this line, the value displayed will be 0.

Xi XIA
2016213482
2016215117

Programming Project 4.22. An interesting way of calculating π is to use a technique known as Monte Carlo, which involves randomization. This technique works as follows: Suppose you have a circle inscribed within a square, as shown in Figure4.18. (Assume that the radius of

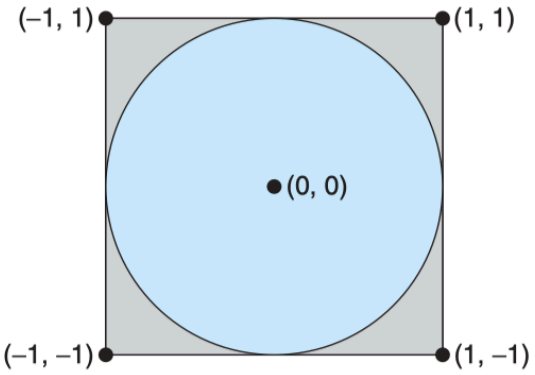

Figure 4.18 Monte Carlo technique for calculating pi.

this circle is 1.) First, generate a series of random points as simple (x, y) coordinates. These points must fall within the Cartesian coordinates that bound the square. Of the total number of random points that are generated, some will occur within the circle. Next, estimate π by performing the following calculation:

$$\pi = 4 \times (number\ of\ points\ in\ circle) \ / \ (total\ number\ of\ points)$$

Write a multithreaded version of this algorithm that creates a separate thread to generate a number of random points. The thread will count the number of points that occur within the circle and store that result in a global variable. When this thread has exited, the parent thread will calculate and output the estimated value of π. It is worth experimenting with the number of random points generated. As a general rule, the greater the number of points, the closer the approximation to π.

In the source-code download for this text, we provide a sample program that provides a technique for generating random numbers, as well as determining if the random (x, y) point occurs within the circle. Readers interested in the details of the Monte Carlo method for estimating π should consult the bibliography at the end of this chapter. In Chapter 5, we modify this exercise using relevant material from that chapter.

```
1.  #include <pthread.h>
2.  #include <stdio.h>
3.  #include <stdlib.h>
4.  #include <math.h>
5.  #include <time.h>
6.
7.  void *worker(void *param);
8.
9.  #define NUMBER_OF_DARTS      50000000
10. #define NUMBER_OF_THREADS    2
11.
12. /* the number of hits in the circle */
13. int circle_count = 0;
14.
15. /*
```

```
 * Generates a double precision random number
 */
double random_double()
{
    return random() / ((double)RAND_MAX + 1);
    //every call to rand gives a sudo random number between 0 and RAND_MAX
}

int main (int argc, const char * argv[]) {
    int darts_per_thread = NUMBER_OF_DARTS/ NUMBER_OF_THREADS;
    int i;

    double estimated_pi;

    /*pthread_t is used to initialise the thread ID*/
    pthread_t workers[NUMBER_OF_THREADS];


    /* seed the random number generator */
    srandom((unsigned)time(NULL));

    /*creat threads*/
    for (i = 0; i < NUMBER_OF_THREADS; i++)
        pthread_create(&workers[i], 0, worker, &darts_per_thread);

    /*wait for the end of a thread, do resource recycling*/
    for (i = 0; i < NUMBER_OF_THREADS; i++)
        pthread_join(workers[i],NULL);

    /* estimate Pi */
    estimated_pi = 4.0 * circle_count / NUMBER_OF_DARTS;

    printf("Pi = %f\n",estimated_pi);

    return 0;
}

void *worker(void *param)
{
    int number_of_darts;
    number_of_darts = *((int *)param);
    int i;
    int hit_count = 0;
    double x,y;

    for (i = 0; i < number_of_darts; i++) {

        /* generate random numbers between -1.0 and +1.0 (exclusive) */
        x = random_double() * 2.0 - 1.0;
        y = random_double() * 2.0 - 1.0;

        if ( sqrt(x*x + y*y) < 1.0 )  //in the circle
            ++hit_count;
    }

    circle_count += hit_count;

    pthread_exit(0);
}
```

Xi XIA
2016213482
2016215117