1 习题 4.2

解：① 当固定负载时，N 为常数，当节点数为 n 时，加速比为：

$$S = \frac{T_1}{T_n} = \frac{CN^3}{\frac{CN^3}{n} + \frac{bN^2}{\sqrt{n}}} = \frac{n}{1 + \frac{b}{CN}\sqrt{n}}$$

当 $n \to \infty$ 时，$S \sim \frac{CN}{b}\sqrt{n}$，而加速经验公式为 $p/log\ p \leqslant S \leqslant p$，说明随着节点数的增加，此算法在这台并行机上的加速效果逐渐变差，主要受到通信开销的制约。

② 当固定时间时，不同节点数的并行机所能完成的工作负载不同，这里用矩阵的阶数代表工作负载。当节点数为 n 时，设固定的时间下，串行算法能完成的工作负载为 N，而并行算法能完成的工作负载为 N_p，则有：

$$CN^3 = CN_p^3/n + bN_p^2/\sqrt{n}$$

加速比定义为使用串行算法完成并行算法的工作负载的时间 T_e 与并行算法的时间 T_p 之比：

$$S' = \frac{T_e}{T_p} = \frac{CN_p^3}{CN^3} = \frac{N_p^3}{N^3}$$

则 S' 满足方程：

$$S' + \frac{b\sqrt{n}}{CN}S'^{\frac{2}{3}} = n$$

经过变换，方程变为：

$$\frac{b}{CN}\left(\frac{S'}{n^{\frac{3}{4}}}\right)^{\frac{2}{3}} + \frac{1}{n^{\frac{1}{4}}}\left(\frac{S'}{n^{\frac{3}{4}}}\right) = 1$$

当 $n \to \infty$ 时，$S \sim \left(\frac{CN}{b}\right)^{\frac{3}{2}} n^{\frac{3}{4}}$，说明在固定时间的情形下，随着节点数的增加，此算法在这台并行机上的加速效果也会逐渐变差，受到通信开销的制约，但比固定负载时的情况好，因为当固定时间的情形下并行分量会提高。

2 习题 4.11

解：按照题意，设串行分量为 f，有

$$\frac{1}{f+\frac{1-f}{p}}=p-1$$

解得

$$f=\frac{1}{(p-1)^2}$$

3 习题 4.14

解：并行计算加速的基本原理是将一个算法中的可并行执行的部分放到多个处理器上同时执行，而并行算法的加速比中的 p 是指它能够将问题分解成能够并行执行的任务数量。因此对于一个具有良好可扩放性的并行算法，任务的规模（个数）应该要随着问题的规模的增加而增加，这样才能充分利用更多的处理器，提高 p。

1. 以下是上三角方程组回代解法的串行算法的形式化描述。（算法 10.1）

输入：$A_{n*n}, b = (b_1, ..., b_n)^T$，输出：$x = (x_1, ..., x_n)^T$

```
Begin
(1) for  i=n  downto  1  do
    (1.1) x_i=b_i/a_{ii}
    (1.2) for  j=1  to  i-1  do
              b_j=b_j-a_{ji}x_i
              a_{ji}=0
          endfor
    endfor
End
```

(1) 请指出串行算法哪些部分可以并行化。

(2) 写出并行算法的形式化描述（需要注明计算模型类型），分析你的算法的时间复杂度。

解：

(1) 若使用 PRAM-CREW 模型，（1.1）和（1.2）部分可以并行化，使用 n-1 个处理器，先先同时读取 b_i 和 a_{ii} 计算出 x_i，然后对 (A, b) 的第 1 行到第 i-1 并行执行运算。

(2) 使用 PRAM-CREW 模型，并行算法的形式化描述为

```
Begin
    for  i=n  downto  1  do
        for  j=1  to  i-1  par-do
            x_i=b_i/a_{ii}
            b_j=b_j-a_{ji}x_i
            a_{ji}=0
        endfor
    endfor
End
```

算法的时间复杂度为 $O(n)$。

2. 习题 7-10

解：

(1) 步骤（1）到步骤（6）中步骤（1）（4）（6）是常数时间，其他步骤的时间复杂度为 $O(\log n)$，且步骤（2）到步骤（6）迭代了 $\log n$ 次，因此总的时间复杂度 $t(n) = O(\log^2 n)$。算法的所有步骤中最多有 n^2 个处理器并行，因此 $p(n) = O(n^2)$。

(2) 求解过程如下表：

顶点	1	2	3	4	5	6	7	8
(1)D	1	2	3	4	5	6	7	8
(2)C	8	6	3	6	7	2	2	1
(3)C	8	6	3	6	7	2	2	1
(4)D	8	6	3	6	7	2	2	1
(5)C	1	2	3	2	2	6	6	8
(6)D	1	2	3	2	2	2	2	1
(2)C	1	2	3	1	1	2	2	2
(3)C	2	1	3	2	2	2	2	1
(4)D	2	1	3	2	2	2	2	1
(5)C	1	2	3	1	1	1	1	2
(6)D	1	1	3	1	1	1	1	1
(2)C	1	1	3	1	1	1	1	1
(3)C	1	1	3	1	1	1	1	1
(4)D	1	1	3	1	1	1	1	1
(5)C	1	1	3	1	1	1	1	1
(6)D	1	1	3	1	1	1	1	1
输出	1	1	3	1	1	1	1	1

T15.1

1. 相当于同步进行一个相同的FFT
2. 如果没有通信体的话，进程1将会中断，而进程2可以执行完

T15.3

```
float data[1024], buff[10];
for(int i = 0; i < 10; i++) {
    buff[i] = data[32 * i];
}
MPI_Send(buff, 10, MPI_FLOAT, dest, tag, MPI_COMM_WORLD);
```

T15.13(1)

```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
#include<math.h>
typedef struct pos {
  float x;
  float y;
} pos;
int main() {
  int a = 100, b = 50, l = 30;
  pos start = (pos){0, 0};
  pos end = (pos){0, 0};
  time_t t;
  srand((unsigned) time(&t));
  unsigned long total = 0, counter = 0;
  clock_t begin_time = clock();
  for (long i = 0; i < 1000000; i++) {
    start.x = rand() % a;
    start.y = rand() % b;
    int x_bias = rand() % (2 * l + 1) - l;
    int y_bias = ((rand() % 100) >= 50) ?
                 (sqrt(l * l - x_bias * x_bias)) :
                 -(sqrt(l * l - x_bias * x_bias));
    end.x = start.x + x_bias;
    end.y = start.y + y_bias;
    if ((end.x < 0) || (end.x >= a) || (end.y < 0) || (end.y >= b)) {
      counter++;
    }
    total++;
  }
  clock_t end_time = clock();
  float seconds = (float)(end_time - begin_time) / CLOCKS_PER_SEC;
  printf("time consumed: %.2f s\n", seconds);
  double pi = (2 * (double) l * ((double) a + (double) b) - (double) l *
(double) l);
```

```
    pi = pi / ((double) counter / (double) total * (double) a * (double) b);
    printf("Pi: %g", pi);
    return 0;
}
```

最后计算得到的结果如下，花费时间0.11s，计算得到的pi为6位的结果，精度十分低