1. Consider a simple loop that calls a function dummy containing a programmable delay. All invocations of the function are independent of the others. Partition this loop across four threads using static, dynamic, and guided scheduling. Use different parameters for static and guided scheduling. Document the result of this experiment as the delay within the dummy function becomes large.

2. Implement and test the OpenMP program for computing a matrix-matrix (50 x 50) product. Use the OMP_NUM_THREADS environment variable to control the number of threads and plot the performance with varying numbers of threads. Consider three cases in which (i) only the outermost loop is parallelized; (ii) the outer two loops are parallelized; and (iii) all three loops are parallelized. What is the observed result from these three cases?

1、

Code:

```
#include <omp.h>
#include <stdio.h>
#include <time.h>

int dummy( int ele )
{
    int i,j;
    int result;
    for(i = 0; i < ele * 10; i++)
    {
        for(j = 0; j < i; j++)
        {
            result = result + j * i;
        }
    }
    return result;
}

int main()
{
    int i;
    int result[10];
    clock_t start, end;
    start = clock();
    #pragma omp parallel for schedule(static, 2)
    for(i = 0; i < 10; i++)
    {
        result[i] = dummy(i);
    }
```

```
    end = clock();
    printf("Total time for static = %lf\n", (double)(end - start));

    start = clock();
    #pragma omp parallel for schedule(dynamic, 2)
    for(i = 0; i < 10; i++)
    {
        result[i] = dummy(i);
    }
    end = clock();
    printf("Total time for dynamic = %lf\n", (double)(end - start));

    start = clock();
    #pragma omp parallel for schedule(guided, 4)
    for(i = 0; i < 10; i++)
    {
        result[i] = dummy(i);
    }
    end = clock();
    printf("Total time for guided = %lf\n", (double)(end - start));

    return 0;
}
```

Result：
Original dummy() function:

```
Total time for static = 1.000000
Total time for dynamic = 0.000000
Total time for guided = 0.000000
```

Dummy() function enlarge 100 times:

```
Total time for static = 4.000000
Total time for dynamic = 1.000000
Total time for guided = 1.000000
```

Dummy() function enlarge 10000 times:

```
Total time for static = 158.000000
Total time for dynamic = 149.000000
Total time for guided = 151.000000
```

2、
Code:

```
#include <omp.h>
#include <stdio.h>
#include <time.h>

int main()
{
    int matrixA[50][50];
    int matrixB[50][50];
    int i, j, k;
    for (i = 0; i < 50; i++) //构造矩阵
    {
        for (j = 0; j < 50; j++)
        {
            matrixA[i][j] = i + j;
            matrixB[i][j] = i * j;
        }
    }
    int result[50][50];

    clock_t start, end;
    start = clock();
#pragma omp parallel num_threads(4)
    for (i = 0; i < 50; i++) //矩阵计算
    {
        for (j = 0; j < 50; j++)
        {
            for (k = 0; k < 50; k++)
            {
                result[i][j] = matrixA[i][k] * matrixB[k][j] +
result[i][j];
            }
        }
    }
    end = clock();
    printf("Total time for (i) = %lf\n", (double)(end - start));

    start = clock();
#pragma omp parallel num_threads(4)
    for (i = 0; i < 50; i++) //矩阵计算
    {
#pragma omp parallel num_threads(4)
        for (j = 0; j < 50; j++)
```

```
        {
            for (k = 0; k < 50; k++)
            {
                result[i][j] = matrixA[i][k] * matrixB[k][j] +
result[i][j];
            }
        }
    }
    end = clock();
    printf("Total time for (ii) = %lf\n", (double)(end - start));

    start = clock();
#pragma omp parallel num_threads(4)
    for (i = 0; i < 50; i++) //矩阵计算
    {
#pragma omp parallel num_threads(4)
        for (j = 0; j < 50; j++)
        {
#pragma omp parallel num_threads(4)
            for (k = 0; k < 50; k++)
            {
                result[i][j] = matrixA[i][k] * matrixB[k][j] +
result[i][j];
            }
        }
    }
    end = clock();
    printf("Total time for (iii) = %lf\n", (double)(end - start));

    return 0;
}
```

Result:
Number of threads = 1:

```
Total time for (i) = 13.000000
Total time for (ii) = 5.000000
Total time for (iii) = 60.000000
```

Number of threads = 2:

```
Total time for (i) = 11.000000
Total time for (ii) = 4.000000
Total time for (iii) = 30.000000
```

Number of threads = 3:

```
Total time for (i) = 11.000000
Total time for (ii) = 4.000000
Total time for (iii) = 23.000000
```

Number of threads = 4:

```
Total time for (i) = 13.000000
Total time for (ii) = 4.000000
Total time for (iii) = 18.000000
```

Graph: