实验报告

实验名称（测量 FFT 程序执行时间）

智能 1602 201607030227 马琛迎

实验目标

测量 FFT 程序运行时间，确定其时间复杂度。

实验要求

- 采用 C/C++编写程序
- 根据自己的机器配置选择合适的输入数据大小 n，至少要测试多个不同的 n (参见思考题)
- 对于相同的 n，建议重复测量 30 次取平均值作为测量结果 (参见思考题)
- 对测量结果进行分析，确定 FFT 程序的时间复杂度
- 回答思考题，答案加入到实验报告叙述中合适位置

思考题

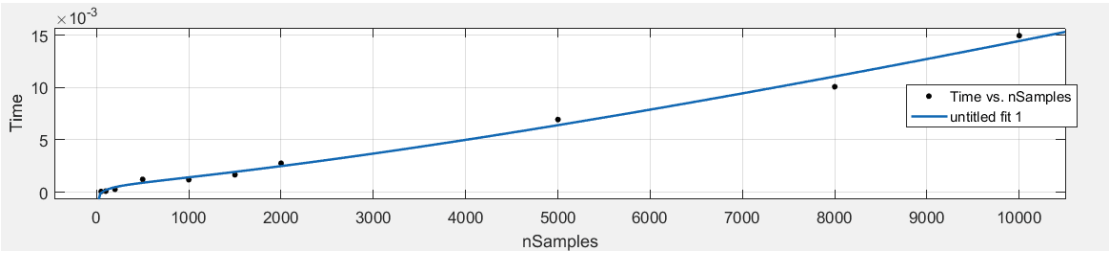
1. 分析 FFT 程序的时间复杂度，得到执行时间相对于数据规模 n 的具体公式

   对于不同的 nSamples，每个 nSamples 测量 30 次运行时间，得到平均值作为该规模的运行时间，测得多组规模（nSamples）与运行时间（Time）一一对应的值，通过 matlab 的 CFTOOL 输入已知公式，拟合出未知数。

   已知公式：

   $$a * n * logn + \frac{b}{3} * n + \sqrt{2} * c * logn + d$$

   拟合效果：

   得到未知数的值：

a = 4.874e-07,

b = -9.845e-06,

c = 0.0003159,

d = -0.001747。

2. 根据上一点中的分析，至少要测试多少不同的 n 来确定执行时间公式中的未知数？

   从拟合效果看来，测量 10 组已经足够，但是要采用合适的数据进行测试。

3. 重复 30 次测量然后取平均有什么统计学的依据？

   可以减小误差，因为对于运行的程序，CPU 每次运行时间都会有所差别，所以多次测量取平均值能够提高准确度。

实验内容

FFT 算法运行时间测试代码

```
/* fft.cpp
 *
 * This is a KISS implementation of
 * the Cooley-Tukey recursive FFT algorithm.
 * This works, and is visibly clear about what is happening where.
 *
 * To compile this with the GNU/GCC compiler:
 * g++ -o fft fft.cpp -lm
 *
 * To run the compiled version from a *nix command line:
 * ./fft
 *
 */
#include <complex>
#include <cstdio>
#include<cstring>
#include<ctime>


#define M_PI 3.14159265358979323846 // Pi constant with double precision

using namespace std;

void separate (complex<double>* a, int n) {
```

```
    complex<double>* b = new complex<double>[n/2];  // get temp heap storage
    for(int i=0; i<n/2; i++)    // copy all odd elements to heap storage
        b[i] = a[i*2+1];
    for(int i=0; i<n/2; i++)    // copy all even elements to lower-half of a[]
        a[i] = a[i*2];
    for(int i=0; i<n/2; i++)    // copy all odd (from heap) to upper-half of a[]
        a[i+n/2] = b[i];
    delete[] b;                // delete heap storage
}

void fft2 (complex<double>* X, int N) {
    if(N < 2) {
        // bottom of recursion.
        // Do nothing here, because already X[0] = x[0]
    } else {
        separate(X,N);      // all evens to lower half, all odds to upper half
        fft2(X,     N/2);   // recurse even items
        fft2(X+N/2, N/2);   // recurse odd  items
        // combine results of two half recursions
        for(int k=0; k<N/2; k++) {
            complex<double> e = X[k    ];   // even
            complex<double> o = X[k+N/2];   // odd
                            // w is the "twiddle-factor"
            complex<double> w = exp( complex<double>(0,-2.*M_PI*k/N) );
            X[k    ] = e + w * o;
            X[k+N/2] = e - w * o;
        }
    }
}

// simple test program
int main () {
    //const int nSamples = 64;
          //scanf("%d",nSamples);
          int nSamples;
    double nSeconds = 1.0;                    // total time for sampling
    double sampleRate = nSamples / nSeconds;    // n Hz = n / second
    double freqResolution = sampleRate / nSamples; // freq step in FFT result
    complex<double> x[nSamples];               // storage for sample data
    complex<double> X[nSamples];               // storage for FFT answer
    const int nFreqs = 5;
    double freq[nFreqs] = { 2, 5, 11, 17, 29 }; // known freqs for testing

          clock_t start,end;
          double time;
          while(scanf("%d",&nSamples),nSamples!=0)
          {
```

```
                time = 0;
                for(int j=0; j<30; j++)
                {
                // generate samples for testing
                for(int i=0; i<nSamples; i++) {
                x[i] = complex<double>(0.,0.);
                // sum several known sinusoids into x[]
                for(int j=0; j<nFreqs; j++)
                    x[i] += sin( 2*M_PI*freq[j]*i/nSamples );
                X[i] = x[i];       // copy into X[] for FFT work & result
                }
                // compute fft for this data
                start = clock();
                fft2(X,nSamples);
                         end = clock();
                         time = time + ((double)(end - start) / CLOCKS_PER_SEC);
                }
                printf("nSamples = %d\tAverage Time = %lf\n",nSamples,time/30);
        }


//    printf("  n\tx[]\tX[]\tf\n");      // header line
//    // loop to print values
//    for(int i=0; i<nSamples; i++) {
//        printf("% 3d\t%+.3f\t%+.3f\t%g\n",
//            i, x[i].real(), abs(X[i]), i*freqResolution );
        return 0;
}

// eof
```

FFT 程序时间复杂度分析

通过分析 FFT 算法代码，可以得到该 FFT 算法的时间复杂度具体公式为：

$$4.874e\text{-}07*n*\log n+(-9845e\text{-}06/3)*n+\sqrt{2}*0.0003159*\log n-0.0017474$$

其中 n 为数据大小，未知数有：

1. a
2. b
3. c
4. d

测试

测试平台

在如下机器上进行了测试：

部件	配置	备注
CPU	core i5-6500U	
内存	DDR3 4GB	
操作系统	Ubuntu 16.04 LTS	中文版

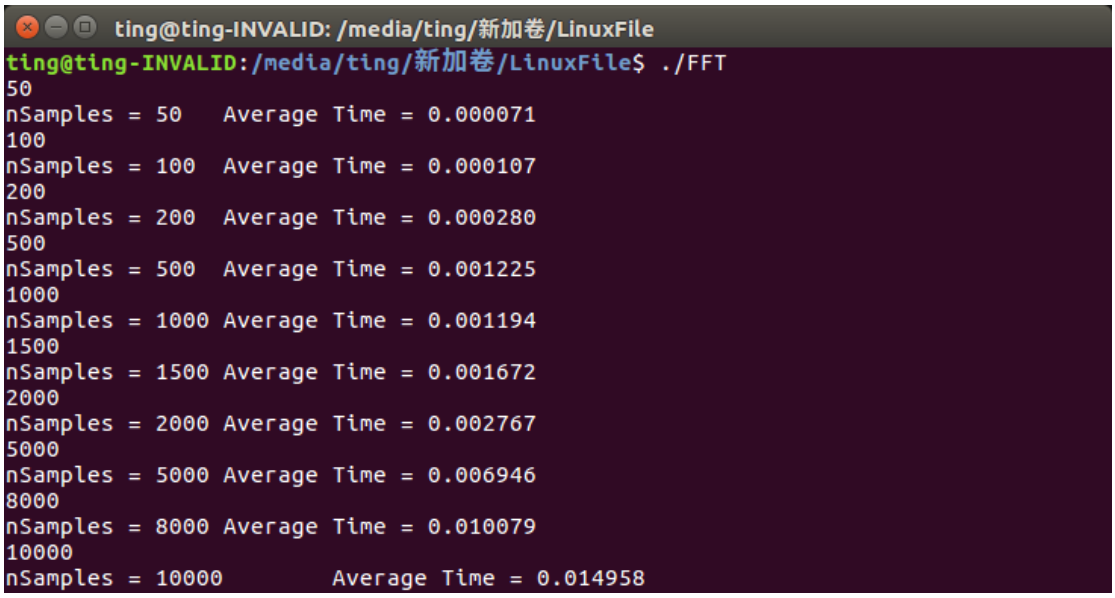
测试记录

FFT 程序运行过程的截图如下：

FFT 程序输出：

```
ting@ting-INVALID:/media/ting/新加卷/LinuxFile$ g++ FFT.cpp -o FFT
ting@ting-INVALID:/media/ting/新加卷/LinuxFile$ ./FFT
50
  n      x[]      X[]      f
  0     +0.000   +5.044   0
  1     +2.181   +16.169  1
  2     +1.740   +9.444   2
  3     -0.081   +3.482   3
  4     +2.423   +10.159  4
  5     +0.000   +9.262   5
  6     +1.689   +6.027   6
  7     +0.835   +5.874   7
  8     -2.797   +2.008   8
  9     +1.408   +2.115   9
 10     +1.176   +14.282  10
 11     +1.124   +7.419   11
 12     +0.539   +3.019   12
 13     +0.288   +5.220   13
 14     +0.388   +2.580   14
 15     +0.000   +3.464   15
 16     -0.133   +15.173  16
 17     -4.607   +7.562   17
 18     -1.129   +4.366   18
 19     -0.307   +10.550  19
 20     -1.902   +9.075   20
 21     +0.734   +8.644   21
 22     -1.451   +5.253   22
 23     +0.777   +2.235   23
 24     +1.684   +0.504   24
 25     +0.000   +2.747   25
 26     -1.684   +2.937   26
 27     -0.777   +4.047   27
 28     +1.451   +8.872   28
 29     -0.734   +12.104  29
 30     +1.902   +5.903   30
 31     +0.307   +4.207   31
 32     +1.129   +14.816  32
 33     +4.607   +9.669   33
 34     +0.133   +3.725   34
 35     +0.000   +3.690   35
 36     -0.388   +2.395   36
 37     -0.288   +2.867   37
 38     -0.539   +13.392  38
 39     -1.124   +9.053   39
 40     -1.176   +2.190   40
 41     -1.408   +4.197   41
 42     +2.797   +4.894   42
 43     -0.835   +5.394   43
 44     -1.689   +13.996  44
 45     -0.000   +5.463   45
 46     -2.423   +3.817   46
 47     +0.081   +12.848  47
 48     -1.740   +6.874   48
 49     -2.181   +3.914   49
ting@ting-INVALID:/media/ting/新加卷/LinuxFile$
```

FFT 程序运行时间输出：



分析和结论

从测试记录来看，FFT 程序的执行时间随数据规模增大而增大，其时间复杂度为：

4.874e-07*n*logn+(-9845e-06/3)*n+√2*0.0003159*logn-0.0017474