实验报告

实验名称（测量 FFT 程序执行时间）

物联 1601 201608010628 曾彤芳

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
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？
3. 重复 30 次测量然后取平均有什么统计学的依据？

实验内容

FFT 算法代码

```
/* 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<ctime>
#include<iostream>

#define M_PI 3.14159265358979323846 // Pi constant with double precision

using namespace std;

// separate even/odd elements to lower/upper halves of array respectively.
// Due to Butterfly combinations, this turns out to be the simplest way
// to get the job done without clobbering the wrong elements.
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
}

// N must be a power-of-2, or bad things will happen.
// Currently no check for this condition.
```

```
//
// N input samples in X[] are FFT'd and results left in X[].
// Because of Nyquist theorem, N samples means
// only first N/2 FFT results in X[] are the answer.
// (upper half of X[] is a reflection with no new information).
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 () {
	clock_t start,finish;
	double totaltime;
	start=clock();
	//int count=0;
//while(count<30){
   //	count++;
	const int nSamples = 16;
	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

	// 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
      fft2(X,nSamples);

      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 );
      }
//}
      finish=clock();
      totaltime=double (finish-start);
      cout<<"yunxingshijian: "<<totaltime/CLOCKS_PER_SEC<<endl;
}

// 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：

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

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：

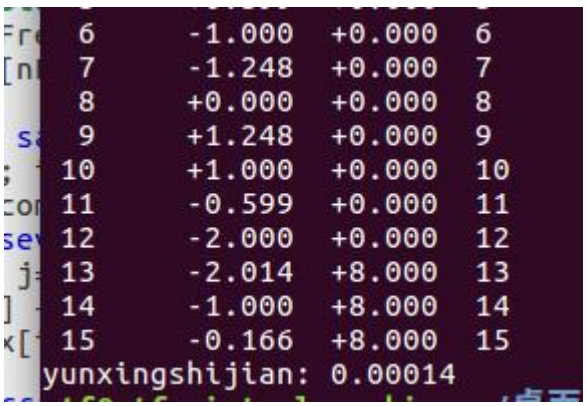
部件	配置	备注
CPU	core i5-6200U	
内存	DDR3 4GB	
操作系统	Ubuntu 16.04 LTS	中文版

测试记录

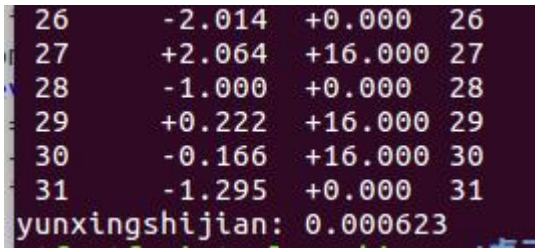
FFT 程序运行过程的截图如下：

FFT 程序的输出

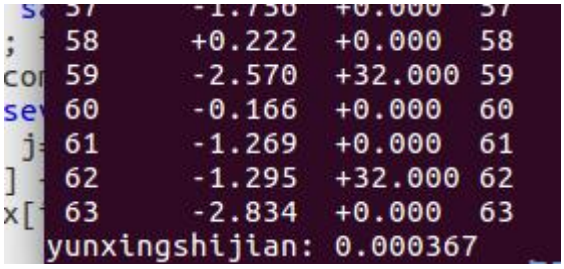
16 条数据



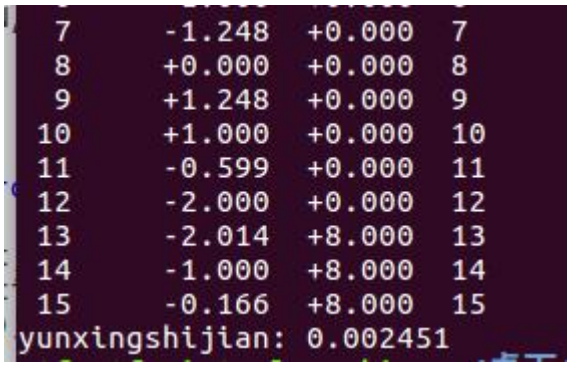
32 条数据



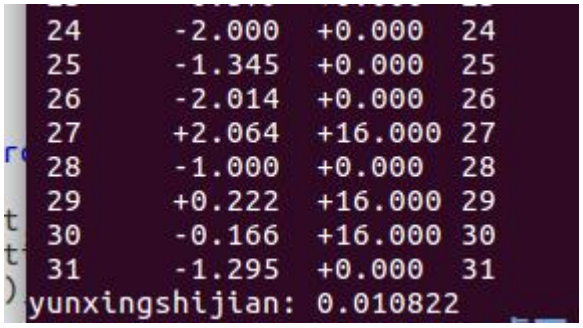
64 条数据



循环 30 次（16 条数据）

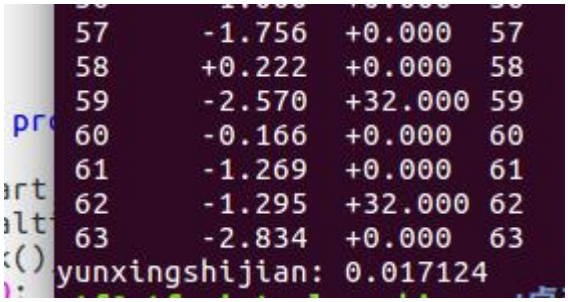


循环 30 次（32 条数据）‘



，

循环 30 次（64 条数据）



思考题解答

1. 测试次数

有四个未知数，因此需要四个方程求解这四个未知数，因此至少需要四个 n。

2. 统计学原理

一次运行时间很短，也会有误差，多次重复运行求平均值，可以减少偶然性，测的结果会更准确一点。

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 O(n log n）。