实验报告

实验名称（用 GPU 加速 FFT 程序）

智能 1602 201608010627 任小禹

实验目标

用 GPU 加速 FFT 程序运行，测量加速前后的运行时间，确定加速比。

实验要求

- 采用 CUDA 或 OpenCL（视具体 GPU 而定）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
- 对测量结果进行分析，确定使用 GPU 加速 FFT 程序得到的加速比
- 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 GPU 加速 FFT 程序可能获得的加速比
2. 实际加速比相对于理想加速比差多少？原因是什么？

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>


#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;
```

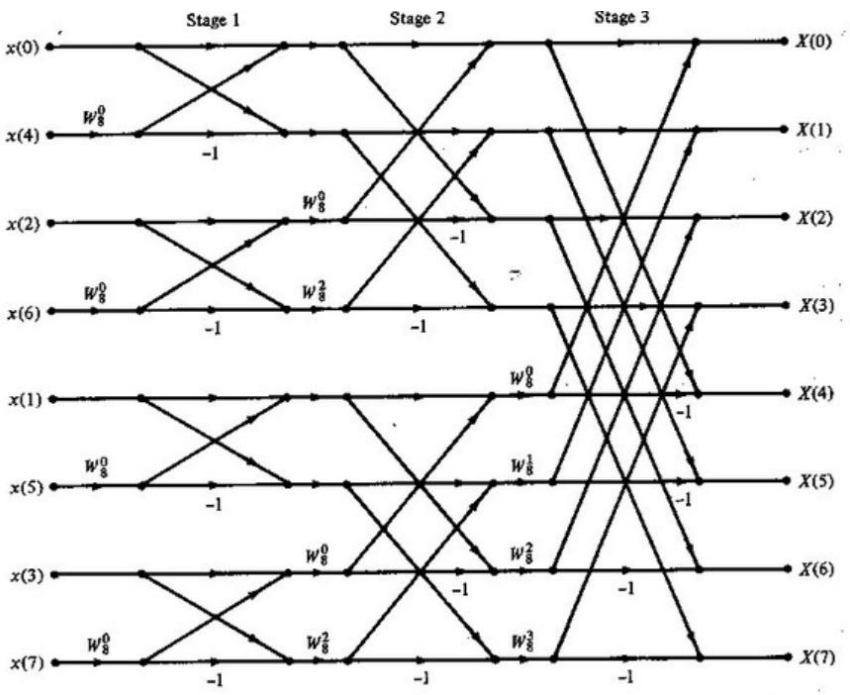
```
                X[k+N/2] = e - w * o;
            }
        }
}

// simple test program
int main () {
        const int nSamples = 64;
        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 );
        }
}

// eof
```

FFT 算法 CUDA 代码

```
#include <assert.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
// Include CUDA runtime and CUFFT
#include <cuda_runtime.h>
#include <cufft.h>
#include<iostream>
// Helper functions for CUDA
//#include <helper_functions.h>
//#include <helper_cuda.h>
//#include "device_launch_parameters.h"

#define pi 3.1415926535
#define LENGTH 500000 //signal sampling points
using namespace std;
int main()
{
clock_t start,finish;
    start = clock();
    int i;
 //   for( i = 0 ;i<30 ;i++){
    // data gen
    float Data[LENGTH] = {1,2,3,4};
    float fs = 1000000.000;//sampling frequency
    float f0 = 200000.00;// signal frequency
    for( i=0;i<LENGTH;i++)
    {
      Data[i] = 1.35*cos(2*pi*f0*i/fs);//signal gen,

    }

    cufftComplex *CompData=(cufftComplex*)malloc(LENGTH*sizeof(cufftComplex));//allocate memory for the data in host
    for(i=0;i<LENGTH;i++)
    {
        CompData[i].x=Data[i];
        CompData[i].y=0;
    }
    cufftComplex *d_fftData;
    cudaMalloc((void**)&d_fftData,LENGTH*sizeof(cufftComplex));// allocate memory for the data in device
    cudaMemcpy(d_fftData,CompData,LENGTH*sizeof(cufftComplex),cudaMemcpyHostToDevice);// copy data from host to device

    cufftHandle plan;// cuda library function handle
    cufftPlan1d(&plan,LENGTH,CUFFT_C2C,1);//declaration
    cufftExecC2C(plan,(cufftComplex*)d_fftData,(cufftComplex*)d_fftData,CUFFT_FORWARD);//execute
    cudaDeviceSynchronize();//wait to be done
    cudaMemcpy(CompData,d_fftData,LENGTH*sizeof(cufftComplex),cudaMemcpyDeviceToHost);// copy the result from device to hos

   for(i=0;i<LENGTH/2;i++)
   {

      printf("i=%d\tf= %6.1fHz\tRealAmp=%3.1f\t",i,fs*i/LENGTH,CompData[i].x*2.0/LENGTH);//print the result:
      printf("ImagAmp=+%3.1fi",CompData[i].y*2.0/LENGTH);
     printf("\n");
   }
     cufftDestroy(plan);
     free(CompData);
     cudaFree(d_fftData);
    finish = clock();
    printf(" running time :%f s\n",double(finish-start)/CLOCKS_PER_SEC );
    }
```

GPU 加速 FFT 程序的可能加速比

通过分析 FFT 算法代码，可以得到该 FFT 算法的并行性体现在节点将同时接受直线边传来的数据和交叉边传来的数据，通过两点 FFT 蝶形计算方法可求出下一个向量。以此类推，经过递归运算，最终将结果数据算出来。在此过程中，并行计算节点之间需要进行数据传递，且在每一个循环阶段，都要进行同步，才可以进行下一步：



如果使用 GPU 进行加速，CUDA 有封装好的 CUFFT 库，这样就不用自己专门实现 FFT 内核函数，直接调用 CUFFT 的 API 函数即可。

CUFFT 可以

1）同时并行处理一批一维离散傅里叶变换；

2）对实数或复数进行的 FFT，结果输出位置可以和输入位置相同（原地变换），也可

以不同；

3）支持流执行：数据传输过程中可以同时执行计算过程。

但是因为调用了 API 函数，具体的加速比需要进行测试。

注意上述分析中未考虑初始化、数据传递等时间，实际加速比可能要比理想情况低。

测试

测试平台

在如下机器上进行了测试：

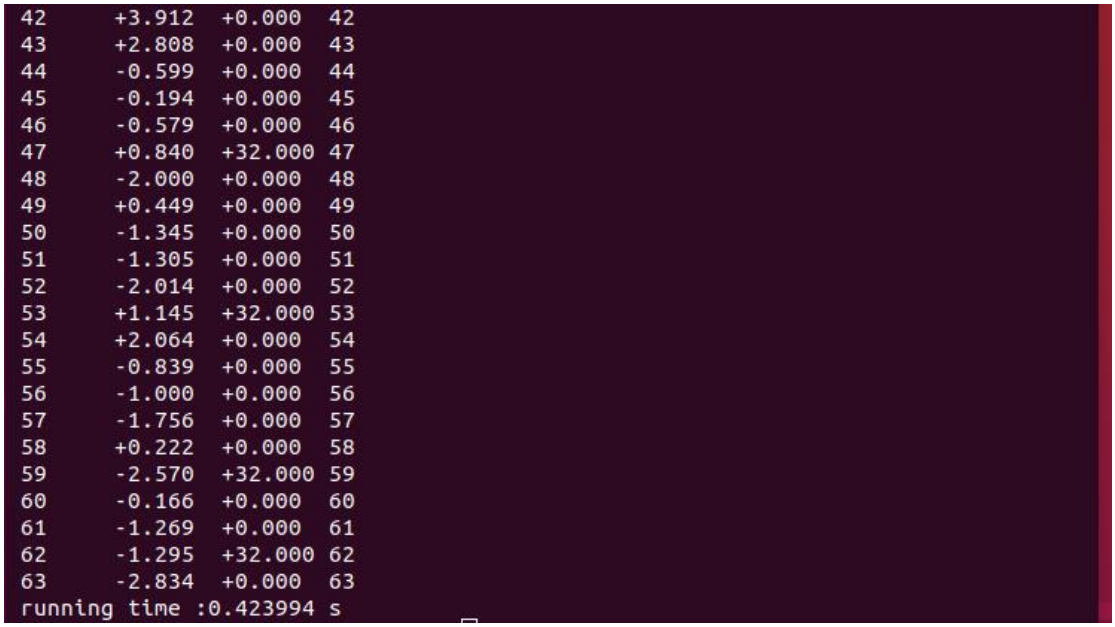
部件	配置	备注
CPU	core i5-6500U	
内存	DDR4 16GB	
GPU	Nvidia Geforce 1050Ti	
显存	DDR5 6GB	
操作系统	Ubuntu 18.04 LTS	中文版

测试记录

FFT 程序运行过程的截图如下：

1）数据规模：10 万

CPU 上 FFT 程序的执行输出



GPU 上 FFT 程序的执行输出

```
i=49978 f= 499780.0Hz   RealAmp=0.0     ImagAmp=+-0.0i
i=49979 f= 499790.0Hz   RealAmp=0.0     ImagAmp=+-0.0i
i=49980 f= 499800.0Hz   RealAmp=0.0     ImagAmp=+-0.0i
i=49981 f= 499810.0Hz   RealAmp=0.0     ImagAmp=+-0.0i
i=49982 f= 499820.0Hz   RealAmp=0.0     ImagAmp=+-0.0i
i=49983 f= 499830.0Hz   RealAmp=0.0     ImagAmp=+0.0i
i=49984 f= 499840.0Hz   RealAmp=0.0     ImagAmp=+-0.0i
i=49985 f= 499850.0Hz   RealAmp=0.0     ImagAmp=+0.0i
i=49986 f= 499860.0Hz   RealAmp=0.0     ImagAmp=+0.0i
i=49987 f= 499870.0Hz   RealAmp=0.0     ImagAmp=+-0.0i
i=49988 f= 499880.0Hz   RealAmp=0.0     ImagAmp=+0.0i
i=49989 f= 499890.0Hz   RealAmp=0.0     ImagAmp=+0.0i
i=49990 f= 499900.0Hz   RealAmp=0.0     ImagAmp=+0.0i
i=49991 f= 499910.0Hz   RealAmp=0.0     ImagAmp=+0.0i
i=49992 f= 499920.0Hz   RealAmp=0.0     ImagAmp=+0.0i
i=49993 f= 499930.0Hz   RealAmp=0.0     ImagAmp=+0.0i
i=49994 f= 499940.0Hz   RealAmp=0.0     ImagAmp=+0.0i
i=49995 f= 499950.0Hz   RealAmp=0.0     ImagAmp=+0.0i
i=49996 f= 499960.0Hz   RealAmp=0.0     ImagAmp=+-0.0i
i=49997 f= 499970.0Hz   RealAmp=0.0     ImagAmp=+-0.0i
i=49998 f= 499980.0Hz   RealAmp=0.0     ImagAmp=+-0.0i
i=49999 f= 499990.0Hz   RealAmp=0.0     ImagAmp=+0.0i
 running time :0.444692 s
```

2）数据规模：50 万

CPU 上 FFT 程序的执行输出

```
42      +3.912  +0.000  42
43      +2.808  +0.000  43
44      -0.599  +0.000  44
45      -0.194  +0.000  45
46      -0.579  +0.000  46
47      +0.840  +32.000 47
48      -2.000  +0.000  48
49      +0.449  +0.000  49
50      -1.345  +0.000  50
51      -1.305  +0.000  51
52      -2.014  +0.000  52
53      +1.145  +32.000 53
54      +2.064  +0.000  54
55      -0.839  +0.000  55
56      -1.000  +0.000  56
57      -1.756  +0.000  57
58      +0.222  +0.000  58
59      -2.570  +32.000 59
60      -0.166  +0.000  60
61      -1.269  +0.000  61
62      -1.295  +32.000 62
63      -2.834  +0.000  63
running time :2.107760 s
```

GPU 上 FFT 程序的执行输出

```
i=249978        f= 499956.0Hz   RealAmp=0.0     ImagAmp=+-0.0i
i=249979        f= 499958.0Hz   RealAmp=0.0     ImagAmp=+0.0i
i=249980        f= 499960.0Hz   RealAmp=0.0     ImagAmp=+-0.0i
i=249981        f= 499962.0Hz   RealAmp=0.0     ImagAmp=+-0.0i
i=249982        f= 499964.0Hz   RealAmp=0.0     ImagAmp=+0.0i
i=249983        f= 499966.0Hz   RealAmp=0.0     ImagAmp=+-0.0i
i=249984        f= 499968.0Hz   RealAmp=0.0     ImagAmp=+0.0i
i=249985        f= 499970.0Hz   RealAmp=0.0     ImagAmp=+-0.0i
i=249986        f= 499972.0Hz   RealAmp=0.0     ImagAmp=+0.0i
i=249987        f= 499974.0Hz   RealAmp=0.0     ImagAmp=+0.0i
i=249988        f= 499976.0Hz   RealAmp=0.0     ImagAmp=+-0.0i
i=249989        f= 499978.0Hz   RealAmp=0.0     ImagAmp=+0.0i
i=249990        f= 499980.0Hz   RealAmp=0.0     ImagAmp=+-0.0i
i=249991        f= 499982.0Hz   RealAmp=0.0     ImagAmp=+-0.0i
i=249992        f= 499984.0Hz   RealAmp=0.0     ImagAmp=+0.0i
i=249993        f= 499986.0Hz   RealAmp=0.0     ImagAmp=+0.0i
i=249994        f= 499988.0Hz   RealAmp=0.0     ImagAmp=+-0.0i
i=249995        f= 499990.0Hz   RealAmp=0.0     ImagAmp=+-0.0i
i=249996        f= 499992.0Hz   RealAmp=0.0     ImagAmp=+0.0i
i=249997        f= 499994.0Hz   RealAmp=0.0     ImagAmp=+-0.0i
i=249998        f= 499996.0Hz   RealAmp=0.0     ImagAmp=+0.0i
i=249999        f= 499998.0Hz   RealAmp=0.0     ImagAmp=+-0.0i
 running time :1.210164 s
```

3）数据规模：100 万

CPU 上 FFT 程序的执行输出

```
43      +2.808  +0.000  43
44      -0.599  +0.000  44
45      -0.194  +0.000  45
46      -0.579  +0.000  46
47      +0.840  +32.000 47
48      -2.000  +0.000  48
49      +0.449  +0.000  49
50      -1.345  +0.000  50
51      -1.305  +0.000  51
52      -2.014  +0.000  52
53      +1.145  +32.000 53
54      +2.064  +0.000  54
55      -0.839  +0.000  55
56      -1.000  +0.000  56
57      -1.756  +0.000  57
58      +0.222  +0.000  58
59      -2.570  +32.000 59
60      -0.166  +0.000  60
61      -1.269  +0.000  61
62      -1.295  +32.000 62
63      -2.834  +0.000  63
running time :4.176054 s
```

GPU 上 FFT 程序的执行输出

```
i=499978        f= 499978.0Hz   RealAmp=0.0     ImagAmp=+0.0i
i=499979        f= 499979.0Hz   RealAmp=0.0     ImagAmp=+0.0i
i=499980        f= 499980.0Hz   RealAmp=0.0     ImagAmp=+0.0i
i=499981        f= 499981.0Hz   RealAmp=0.0     ImagAmp=+-0.0i
i=499982        f= 499982.0Hz   RealAmp=0.0     ImagAmp=+0.0i
i=499983        f= 499983.0Hz   RealAmp=0.0     ImagAmp=+-0.0i
i=499984        f= 499984.0Hz   RealAmp=0.0     ImagAmp=+0.0i
i=499985        f= 499985.0Hz   RealAmp=0.0     ImagAmp=+-0.0i
i=499986        f= 499986.0Hz   RealAmp=0.0     ImagAmp=+0.0i
i=499987        f= 499987.0Hz   RealAmp=0.0     ImagAmp=+0.0i
i=499988        f= 499988.0Hz   RealAmp=0.0     ImagAmp=+-0.0i
i=499989        f= 499989.0Hz   RealAmp=0.0     ImagAmp=+0.0i
i=499990        f= 499990.0Hz   RealAmp=0.0     ImagAmp=+-0.0i
i=499991        f= 499991.0Hz   RealAmp=0.0     ImagAmp=+-0.0i
i=499992        f= 499992.0Hz   RealAmp=0.0     ImagAmp=+0.0i
i=499993        f= 499993.0Hz   RealAmp=0.0     ImagAmp=+0.0i
i=499994        f= 499994.0Hz   RealAmp=0.0     ImagAmp=+-0.0i
i=499995        f= 499995.0Hz   RealAmp=0.0     ImagAmp=+-0.0i
i=499996        f= 499996.0Hz   RealAmp=0.0     ImagAmp=+-0.0i
i=499997        f= 499997.0Hz   RealAmp=0.0     ImagAmp=+-0.0i
i=499998        f= 499998.0Hz   RealAmp=0.0     ImagAmp=+-0.0i
i=499999        f= 499999.0Hz   RealAmp=0.0     ImagAmp=+0.0i
 running time :2.196498 s
```

4）数据规模：200 万

CPU 上 FFT 程序的执行输出

```
42      +3.912  +0.000  42
43      +2.808  +0.000  43
44      -0.599  +0.000  44
45      -0.194  +0.000  45
46      -0.579  +0.000  46
47      +0.840  +32.000 47
48      -2.000  +0.000  48
49      +0.449  +0.000  49
50      -1.345  +0.000  50
51      -1.305  +0.000  51
52      -2.014  +0.000  52
53      +1.145  +32.000 53
54      +2.064  +0.000  54
55      -0.839  +0.000  55
56      -1.000  +0.000  56
57      -1.756  +0.000  57
58      +0.222  +0.000  58
59      -2.570  +32.000 59
60      -0.166  +0.000  60
61      -1.269  +0.000  61
62      -1.295  +32.000 62
63      -2.834  +0.000  63
running time :8.325845 s
```

GPU 上 FFT 程序的执行输出

```
i=999978        f= 499989.0Hz   RealAmp=0.0     ImagAmp=+0.0i
i=999979        f= 499989.5Hz   RealAmp=0.0     ImagAmp=+0.0i
i=999980        f= 499990.0Hz   RealAmp=0.0     ImagAmp=+-0.0i
i=999981        f= 499990.5Hz   RealAmp=0.0     ImagAmp=+0.0i
i=999982        f= 499991.0Hz   RealAmp=0.0     ImagAmp=+-0.0i
i=999983        f= 499991.5Hz   RealAmp=0.0     ImagAmp=+0.0i
i=999984        f= 499992.0Hz   RealAmp=0.0     ImagAmp=+-0.0i
i=999985        f= 499992.5Hz   RealAmp=0.0     ImagAmp=+0.0i
i=999986        f= 499993.0Hz   RealAmp=0.0     ImagAmp=+0.0i
i=999987        f= 499993.5Hz   RealAmp=0.0     ImagAmp=+0.0i
i=999988        f= 499994.0Hz   RealAmp=0.0     ImagAmp=+-0.0i
i=999989        f= 499994.5Hz   RealAmp=0.0     ImagAmp=+0.0i
i=999990        f= 499995.0Hz   RealAmp=0.0     ImagAmp=+0.0i
i=999991        f= 499995.5Hz   RealAmp=0.0     ImagAmp=+0.0i
i=999992        f= 499996.0Hz   RealAmp=0.0     ImagAmp=+0.0i
i=999993        f= 499996.5Hz   RealAmp=0.0     ImagAmp=+0.0i
i=999994        f= 499997.0Hz   RealAmp=0.0     ImagAmp=+-0.0i
i=999995        f= 499997.5Hz   RealAmp=0.0     ImagAmp=+-0.0i
i=999996        f= 499998.0Hz   RealAmp=0.0     ImagAmp=+-0.0i
i=999997        f= 499998.5Hz   RealAmp=0.0     ImagAmp=+0.0i
i=999998        f= 499999.0Hz   RealAmp=0.0     ImagAmp=+0.0i
i=999999        f= 499999.5Hz   RealAmp=0.0     ImagAmp=+0.0i
 running time :4.155659 s
```

数据规模	CPU 运行时间(s)	GPU 运行时间(s)	加速比
10 万	0.423994	0.444692	0.95
50 万	2.107760	1.210164	1.74
100 万	4.176054	2.196498	1.90
1000 万	8.325845	4.155659	2.00

分析和结论

从测试记录来看，使用 GPU 加速 FFT 程序获得的加速比大概为 1.9，我们发现数据规模小的时候，GPU 下运行和 CPU 下运行没有什么区别，而数据规模大时，差别就显现出来了，GPU 下运行速度明显快于 CPU 下运行速度。

造成这种现象的原因为：

1. 初始化会消耗时间；
2. 数据通信会消耗的时间；
3. GPU 上线程调度开销也会造成影响；
4. GPU 上线程之间访存竞争也造成了一定的影响。