实验报告

实验名称（用 **GPU** 加速 **FFT** 程序）

智能 1501 201508010528 耿昊

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 的算法可以参考这里。

```
/* 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
```

Cufft 的代码如下：

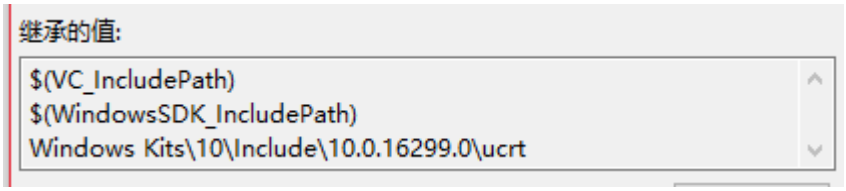
```
#include <assert.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include<Windows.h>
#include<iostream>
// Include CUDA runtime and CUFFT
#include <cuda_runtime.h>
#include <cufft.h>
// Helper functions for CUDA
//#include "device_launch_parameters.h"
#define pi 3.1415926535
#define LENGTH 100000 //signal sampling points
int main()
{
    printf("AAAAAAAAAAAAAAAAAAAA");
    // data gen
    float Data[LENGTH] = { 2,5,11,17,29 };
    float fs = 1000000.000;//sampling frequency
    float f0 = 200000.00;// signal frequency
    DWORD start2, end2;
    start2 = GetTickCount();
    cudaEvent_t start, stop;
    cudaEventCreate(&start);
    cudaEventCreate(&stop);
    cudaEventRecord(start, 0);
    for (int i = 0; i < LENGTH; i++)
    {
        Data[i] = 1.35*cos(2 * pi*f0*i / fs);//signal gen,
    }
    cufftComplex *CompData = (cufftComplex*)malloc(LENGTH *
sizeof(cufftComplex));//allocatememory for the data in host
```

```
        int i;
    for (i = 0; i < LENGTH; i++)
    {
        CompData[i].x = Data[i];
        CompData[i].y = 0;
    }
    //end2 = GetTickCount();
    //cudaEventRecord(stop, 0);
    //cudaEventSynchronize(stop);
    //float time;
    //cudaEventElapsedTime(&time, start, stop);
    cufftComplex *d_fftData;
    cudaMalloc((void**)&d_fftData, LENGTH * sizeof(cufftComplex));// allocate memory
for thedata 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 host
        end2 = GetTickCount();
    cudaEventRecord(stop, 0);
    cudaEventSynchronize(stop);
    float time;
    cudaEventElapsedTime(&time, start, stop);
    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);
        printf("ImagAmp=+%3.1fi", CompData[i].y*2.0 / LENGTH);
        printf("\n");
    }
    printf("cpu time: %d ms\n", end2 - start2);
    printf("gpu time = %3.1f ms\n", time);
    cufftDestroy(plan);
    free(CompData);
    cudaFree(d_fftData);
    getchar();
}
```

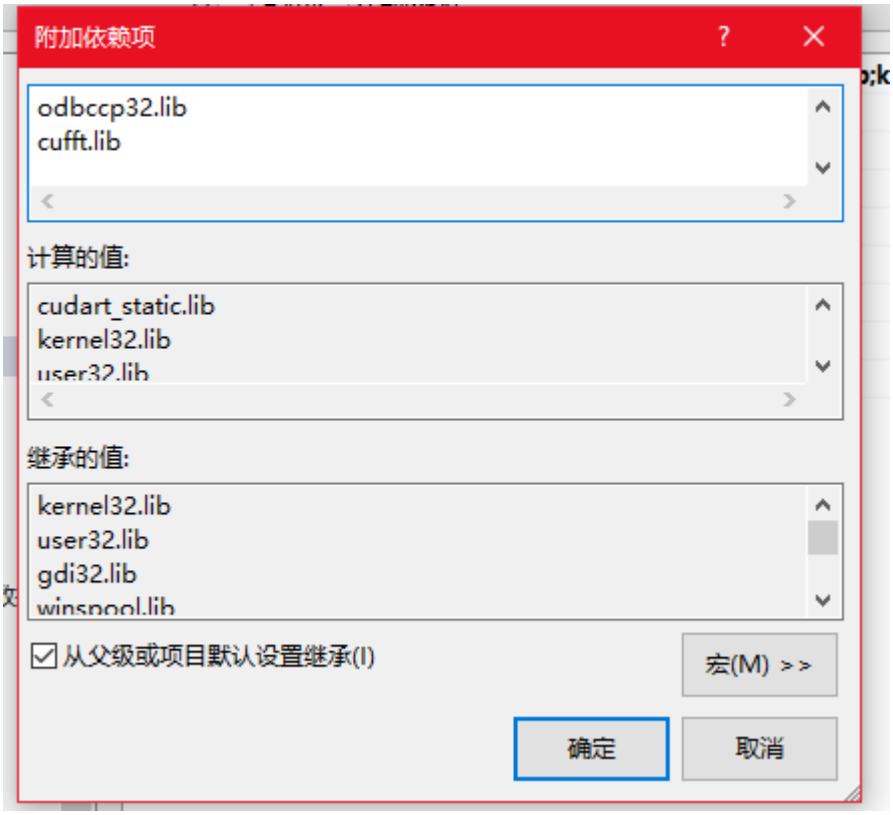
Windows10+cuda10+vs2017 情况下出现错误的解决:

1.许多基本源文件报错（无法打开源文件）
解决方案：
在解决方案上右键->属性->VC++目录->包含目录 ， 增加“Windows Kits\10\Include\10.0.16299.0\ucrt”路径。 这种办法只能一次解决一个解决方案的问题。

要想以后每次打开项目的时候都直接可以用，那就随便建一个 CUDA 项目，在“视图->其他窗口->属性管理器 Release->Microsoft.Cpp.x64.user->VC++目录->包含目录”中增加“Windows Kits\10\Include\10.0.16299.0\ucrt”路径。



2.运行 cufft 示例程序出现的无法解析外部符号错误
解决方案：
配置属性–>链接器–>输入–>附加依赖项，在其中添加以下依赖项：
cuda.lib
cudadevrt.lib
cudart.lib
cudart_static.lib
OpenCL.lib
在使用 cufft 的情况下则要添加 cufft.lib



以上方法来自于 https://blog.csdn.net/leelitian3/article/details/83272272 和

https://bbs.csdn.net/topics/350242812

GPU 加速 FFT 程序的可能加速比

通过分析FFT算法代码，我们预期的优化部分为变换过程中x轴与y轴的计算过程。

但是因为直接调用cufft库，无法看到内部加速过程，最终加速比由实验测试结果进行推断。

注意理论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。

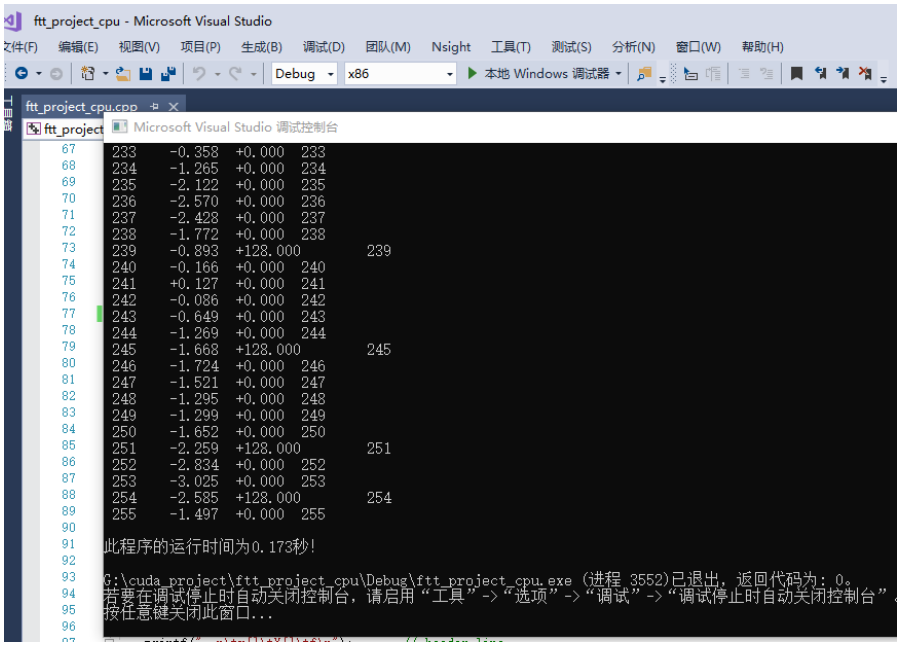
测试

测试平台

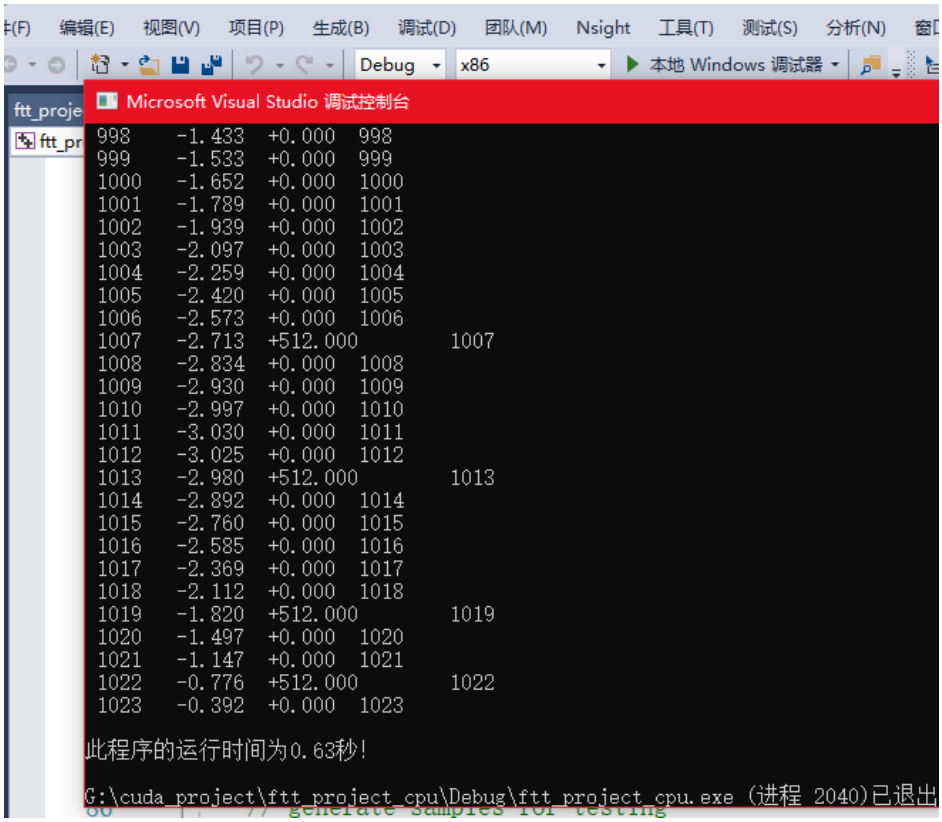
在如下机器上进行了测试：

部件	配置	备注
CPU	core i5-8300H	
内存	DDR4 16GB	
GPU	Nvidia Geforce GTX 1050Ti	
显存	4GB	
操作系统	Windows 10 version1809	

未经过加速的 ftt 程序运行结果为：
N=256：


ftt_project_cpu - Microsoft Visual Studio
Debug
x86
本地 Windows 调试器
Microsoft Visual Studio 调试控制台
233 -0.358 +0.000 233
234 -1.265 +0.000 234
235 -2.122 +0.000 235
236 -2.570 +0.000 236
237 -2.428 +0.000 237
238 -1.772 +0.000 238
239 -0.893 +128.000 239
240 -0.166 +0.000 240
241 +0.127 +0.000 241
242 -0.086 +0.000 242
243 -0.649 +0.000 243
244 -1.269 +0.000 244
245 -1.668 +128.000 245
246 -1.724 +0.000 246
247 -1.521 +0.000 247
248 -1.295 +0.000 248
249 -1.299 +0.000 249
250 -1.652 +0.000 250
251 -2.259 +128.000 251
252 -2.834 +0.000 252
253 -3.025 +0.000 253
254 -2.585 +128.000 254
255 -1.497 +0.000 255
此程序的运行时间为0.173秒!
G:\cuda_project\ftt_project_cpu\Debug\ftt_project_cpu.exe (进程 3552)已退出，返回代码为: 0。
若要在调试停止时自动关闭控制台，请启用“工具”->“选项”->“调试”->“调试停止时自动关闭控制台”。
按任意键关闭此窗口...

N=1024:


Debug
x86
本地 Windows 调试器
Microsoft Visual Studio 调试控制台
998 -1.433 +0.000 998
999 -1.533 +0.000 999
1000 -1.652 +0.000 1000
1001 -1.789 +0.000 1001
1002 -1.939 +0.000 1002
1003 -2.097 +0.000 1003
1004 -2.259 +0.000 1004
1005 -2.420 +0.000 1005
1006 -2.573 +0.000 1006
1007 -2.713 +512.000 1007
1008 -2.834 +0.000 1008
1009 -2.930 +0.000 1009
1010 -2.997 +0.000 1010
1011 -3.030 +0.000 1011
1012 -3.025 +0.000 1012
1013 -2.980 +512.000 1013
1014 -2.892 +0.000 1014
1015 -2.760 +0.000 1015
1016 -2.585 +0.000 1016
1017 -2.369 +0.000 1017
1018 -2.112 +0.000 1018
1019 -1.820 +512.000 1019
1020 -1.497 +0.000 1020
1021 -1.147 +0.000 1021
1022 -0.776 +512.000 1022
1023 -0.392 +0.000 1023
此程序的运行时间为0.63秒!
G:\cuda_project\ftt_project_cpu\Debug\ftt_project_cpu.exe (进程 2040)已退出

测试记录

FFT 程序的测试输入文件请见这里。
FFT 程序运行过程的截图如下:
CPU 上 FFT 程序的执行输出
GPU 上 FFT 程序的执行输出
数据规模 10000

数据规模 100000

数据规模 1000000

数据规模	CPU 时间（ms）	GPU 时间（ms）	加速比
10000	453	189.5	2.39
100000	484	184.5	2.62
1000000	496	189.5	2.61

分析和结论

从测试记录来看，使用 GPU 加速 FFT 程序获得的加速比在理想的情况下为 2.61，因为 GPU 加速的优势只有在数据量大时才能够体现，当数据量小的时候，更多的时间用于数据拷贝和传输，所以反而性能比 CPU 要差很多.。
造成实际加速比与理想加速比不同的原因为：
1. 程序的初始化需要消耗时间；
2. 数据之间的通信需要消耗时间；
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造成的影响，也会影响最终结果。