

## 原 HNU Comparch LAB4 用GPU加速FFT程序

2019年01月07日 13:17:17 [LighTInGFight](#) 阅读数: 5



0



### 一、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。

### 二、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

### 三、实验内容

使用CUDA进行FFT加速

使用到的代码如下

- FFT.h

```
1 typedef struct complex //复数类型
2 {
3     float real;          //实部
4     float imag;          //虚部
5 }complex;
6
7 #define PI 3.1415926535
8
9 void complex_plus(complex a, complex b, complex *c); //复数加
10 void complex_mul(complex a, complex b, complex *c); //复数乘
11 void complex_sub(complex a, complex b, complex *c); //复数减法
12 void complex_div(complex a, complex b, complex *c); //复数除法
13 void complex_abs(complex f[], float out[], float n); //复数数组取模
14
15 void fft(int N, complex f[]); //傅立叶变换 输出也存在数组f中
16 void ifft(int N, complex f[]); // 傅里叶逆变换
17
18 void conjugate_complex(int n, complex in[], complex out[]);
```

- CUDA.h

```
1 #include "FFT.h"
2 #include <iostream>
3 #include <time.h>
4 #include "cuda_runtime.h"
5 #include "device_launch_parameters.h"
6 #include "../include/cufft.h"
7
8 #define NX 4096 // 有效数据个数
9 #define N 5335 // 补0之后的数据长度
10 #define MAX 1<<12
11 #define BATCH 1
12 #define BLOCK_SIZE 1024
13 using std::cout;
14 using std::endl;
15
16 complex m[MAX], l[MAX];
17
```

```

18 | bool isEqual(cufftComplex *idataA, cufftComplex *idataB, const long int size)
19 | {
20 |     for (int i = 0; i < size; i++)
21 |     {
22 |         if (abs(idataA[i].x - idataB[i].x) > 0.000001 || abs(idataA[i].y - idataB[i].y) > 0.000001)
23 |             return false;
24 |     }
25 |
26 |     return true;
27 | }
28 |
29 |
30 |
31 | /**
32 |  * 功能: 实现 cufftComplex 数组的尺度缩放, 也就是乘以一个数
33 |  * 输入: idata 输入数组的头指针
34 |  * 输出: odata 输出数组的头指针
35 |  * 输入: size 数组的元素个数
36 |  * 输入: scale 缩放尺度
37 |  */
38 | static __global__ void cufftComplexScale(cufftComplex *idata, cufftComplex *odata, const long int size, float scale)
39 | {
40 |     const int threadID = blockIdx.x * blockDim.x + threadIdx.x;
41 |
42 |     if (threadID < size)
43 |     {
44 |         odata[threadID].x = idata[threadID].x * scale;
45 |         odata[threadID].y = idata[threadID].y * scale;
46 |     }
47 | }
48 |
49 |
50 | void conjugate_complex(int n, complex in[], complex out[])
51 | {
52 |     int i = 0;
53 |     for (i = 0; i < n; i++)
54 |     {
55 |         out[i].imag = -in[i].imag;
56 |         out[i].real = in[i].real;
57 |     }
58 | }
59 |
60 | void complex_abs(complex f[], float out[], int n)
61 | {
62 |     int i = 0;
63 |     float t;
64 |     for (i = 0; i < n; i++)
65 |     {
66 |         t = f[i].real * f[i].real + f[i].imag * f[i].imag;
67 |         out[i] = sqrt(t);
68 |     }
69 | }
70 |
71 |
72 | void complex_plus(complex a, complex b, complex *c)
73 | {
74 |     c->real = a.real + b.real;
75 |     c->imag = a.imag + b.imag;
76 | }
77 |
78 | void complex_sub(complex a, complex b, complex *c)
79 | {
80 |     c->real = a.real - b.real;
81 |     c->imag = a.imag - b.imag;
82 | }
83 |
84 | void complex_mul(complex a, complex b, complex *c)
85 | {
86 |     c->real = a.real * b.real - a.imag * b.imag;
87 |     c->imag = a.real * b.imag + a.imag * b.real;
88 | }

```



```

89 |
90 | void complex_div(complex a, complex b, complex *c)
91 | {
92 |     c->real = (a.real * b.real + a.imag * b.imag) / (b.real * b.real + b.imag * b.imag);
93 |     c->imag = (a.imag * b.real - a.real * b.imag) / (b.real * b.real + b.imag * b.imag);
94 | }
95 |
96 | #define SWAP(a,b) tempr=(a);(a)=(b);(b)=tempr
97 |
98 | void Wn_i(int n, int i, complex *Wn, char flag)
99 | {
100 |     Wn->real = cos(2 * PI*i / n);
101 |     if (flag == 1)
102 |         Wn->imag = -sin(2 * PI*i / n);
103 |     else if (flag == 0)
104 |         Wn->imag = -sin(2 * PI*i / n);
105 | }
106 |
107 | //傅里叶变化
108 | void fft(int NN, complex f[])
109 | {
110 |     complex t, wn; //中间变量
111 |     int i, j, k, m, n, l, r, M;
112 |     int la, lb, lc;
113 |     /*----计算分解的级数M=log2(N)----*/
114 |     for (i = NN, M = 1; (i = i / 2) != 1; M++);
115 |     /*----按照倒位序重新排列原信号----*/
116 |     for (i = 1, j = NN / 2; i <= NN - 2; i++)
117 |     {
118 |         if (i < j)
119 |         {
120 |             t = f[j];
121 |             f[j] = f[i];
122 |             f[i] = t;
123 |         }
124 |         k = NN / 2;
125 |         while (k <= j)
126 |         {
127 |             j = j - k;
128 |             k = k / 2;
129 |         }
130 |         j = j + k;
131 |     }
132 |
133 |     /*----FFT算法----*/
134 |     for (m = 1; m <= M; m++)
135 |     {
136 |         la = pow(2, m); //la=2^m代表第m级每个分组所含节点数
137 |         lb = la / 2;    //lb代表第m级每个分组所含蝶形单元数
138 |                             //同时它也表示每个蝶形单元上下节点之间的距离
139 |         /*----蝶形运算----*/
140 |         for (l = 1; l <= lb; l++)
141 |         {
142 |             r = (l - 1) * pow(2, M - m);
143 |             for (n = l - 1; n < NN - 1; n = n + la) //遍历每个分组, 分组总数为N/la
144 |             {
145 |                 lc = n + lb; //n,lc分别代表一个蝶形单元的上、下节点编号
146 |                 Wn_i(NN, r, &wn, 1); //wn=Wnr
147 |                 complex_mul(f[lc], wn, &t); //t = f[lc] * wn 复数运算
148 |                 complex_sub(f[n], t, &(f[lc])); //f[lc] = f[n] - f[lc] * Wnr
149 |                 complex_plus(f[n], t, &(f[n])); //f[n] = f[n] + f[lc] * Wnr
150 |             }
151 |         }
152 |     }
153 | }
154 |
155 | //傅里叶逆变换
156 | void ifft(int NN, complex f[])
157 | {
158 |     int i = 0;
159 |     conjugate_complex(NN, f, f);

```



```

160     fft(NN, f); 161     conjugate_complex(NN, f, f);
162     for (i = 0; i < NN; i++)
163     {
164         f[i].imag = (f[i].imag) / NN;
165         f[i].real = (f[i].real) / NN;
166     }
167 }
168
169 bool IsEqual2(complex idataA[], complex idataB[], const int size)
170 {
171     for (int i = 0; i < size; i++)
172     {
173         if (abs(idataA[i].real - idataB[i].real) > 0.000001 || abs(idataA[i].imag - idataB[i].imag) > 0.000001)
174             return false;
175     }
176
177     return true;
178 }

```

- Main

```

1  int main()
2  {
3      cufftComplex *data_dev; // 设备端数据头指针
4      cufftComplex *data_Host = (cufftComplex*)malloc(NX*BATCH * sizeof(cufftComplex)); // 主机端数据头指针
5      cufftComplex *resultFFT = (cufftComplex*)malloc(N*BATCH * sizeof(cufftComplex)); // 正变换的结果
6      cufftComplex *resultIFFT = (cufftComplex*)malloc(NX*BATCH * sizeof(cufftComplex)); // 先正变换后逆变换的结果
7
8      // 初始数据
9      for (int i = 0; i < NX; i++)
10     {
11         data_Host[i].x = float((rand() * rand()) % NX) / NX;
12         data_Host[i].y = float((rand() * rand()) % NX) / NX;
13     }
14
15
16     dim3 dimBlock(BLOCK_SIZE); // 线程块
17     dim3 dimGrid((NX + BLOCK_SIZE - 1) / dimBlock.x); // 线程格
18
19     cufftHandle plan; // 创建cuFFT句柄
20     cufftPlan1d(&plan, N, CUFFT_C2C, BATCH);
21
22     // 计时
23     clock_t start, stop;
24     double duration;
25     start = clock();
26
27     cudaMalloc((void*)&data_dev, sizeof(cufftComplex)*N*BATCH); // 开辟设备内存
28     cudaMemset(data_dev, 0, sizeof(cufftComplex)*N*BATCH); // 初始为0
29     cudaMemcpy(data_dev, data_Host, NX * sizeof(cufftComplex), cudaMemcpyHostToDevice); // 从主机内存拷贝到设备内存
30
31     cufftExecC2C(plan, data_dev, data_dev, CUFFT_FORWARD); // 执行 cuFFT, 正变换
32     cudaMemcpy(resultFFT, data_dev, N * sizeof(cufftComplex), cudaMemcpyDeviceToHost); // 从设备内存拷贝到主机内存
33
34     cufftExecC2C(plan, data_dev, data_dev, CUFFT_INVERSE); // 执行 cuFFT, 逆变换
35     cufftComplexScale << <dimGrid, dimBlock >> > (data_dev, data_dev, N, 1.0f / N); // 乘以系数
36     cudaMemcpy(resultIFFT, data_dev, NX * sizeof(cufftComplex), cudaMemcpyDeviceToHost); // 从设备内存拷贝到主机内存
37
38     stop = clock();
39     duration = (double)(stop - start) * 1000 / CLOCKS_PER_SEC;
40     cout << "FFT经过GPU加速之后的时间为" << duration << "ms" << endl;
41
42     cufftDestroy(plan); // 销毁句柄
43     cudaFree(data_dev); // 释放空间
44
45
46     if (IsEqual(data_Host, resultIFFT, NX))
47         cout << "逆变化检测通过." << endl;

```

```
48     else
49         cout << "逆变换检测不通过。" << endl;
50
51
52
53     //cpu fft
54     for (long int i = 0; i < MAX; i++) {
55         m[i].real = float((rand() * rand()) % 4096) / 4096;
56         l[i].real = m[i].real;
57         l[i].imag = m[i].imag = 0;
58     }
59     clock_t start2, end2;
60     double duration1;
61     start2 = clock();
62     fft(MAX, m);
63     ifft(MAX, m);
64     end2 = clock();
65     duration1 = (double)(end2 - start2) * 1000 / CLOCKS_PER_SEC;
66     cout << "FFT经过CPU加速之后的时间为 " << duration1 << " ms" << endl;
67
68     if (IsEqual2(m, l, MAX))
69         cout << "逆变化检测通过。" << endl;
70     else
71         cout << "逆变化检测不通过。" << endl;
72
73     return 0;
74 }
```



## 四、测试平台

CPU: i7-6500U

GPU: NVIDIA GTX960M

内存: DDR3 8GB

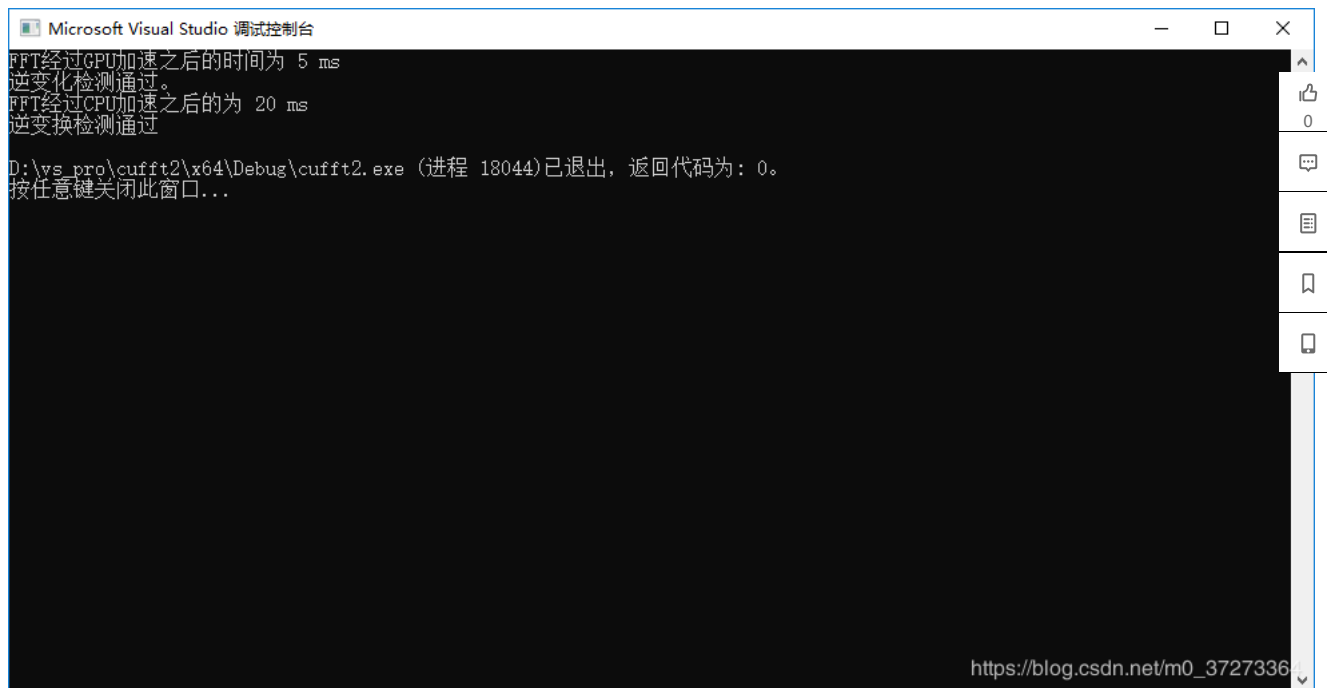
操作系统: Windows 10 专业版

编译器: Microsoft Visual Studio Enterprise 2017

## 五、测试记录

分别测试数据集 $2^{14}$ 、 $2^{15}$ 、 $2^{16}$ 、 $2^{17}$ 、 $2^{18}$ 时的运行情况

$2^{14}$



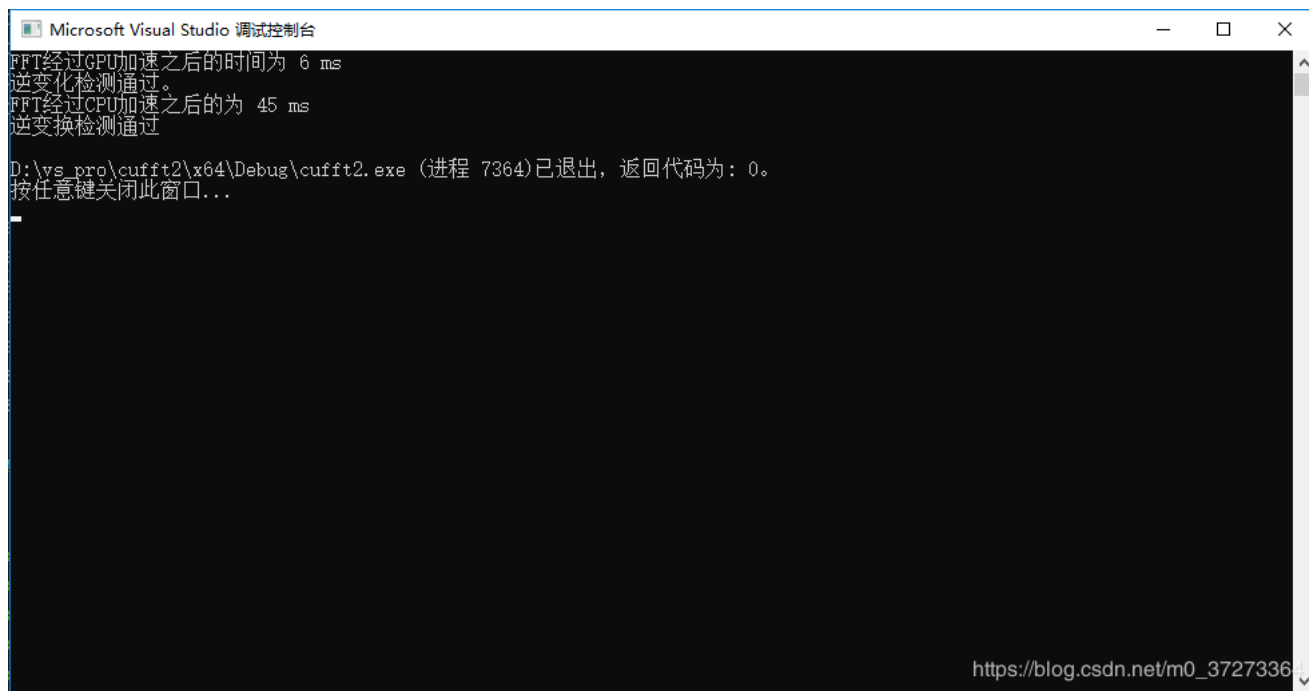
Microsoft Visual Studio 调试控制台

```
FFT经过GPU加速之后的时间为 5 ms  
逆变化检测通过。  
FFT经过CPU加速之后的为 20 ms  
逆变换检测通过
```

D:\vs\_pro\cufft2\x64\Debug\cufft2.exe (进程 18044)已退出, 返回代码为: 0。  
按任意键关闭此窗口...

[https://blog.csdn.net/m0\\_37273364](https://blog.csdn.net/m0_37273364)

2^15



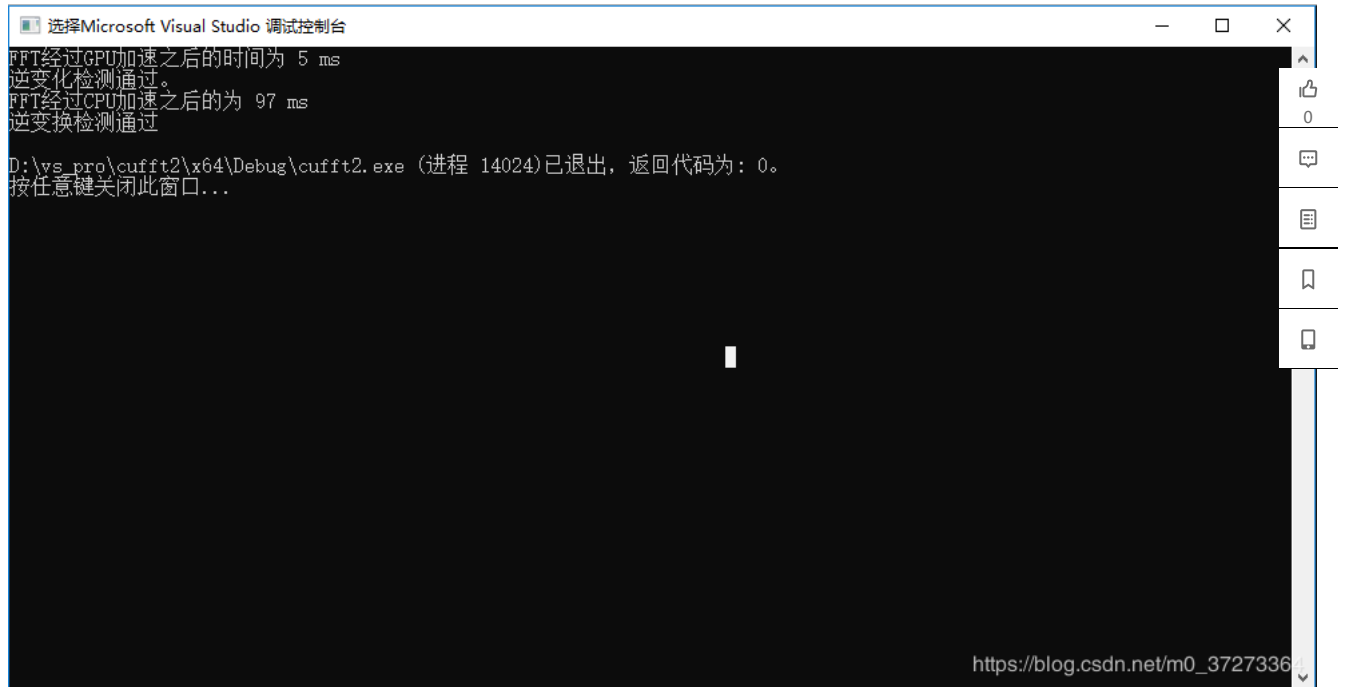
Microsoft Visual Studio 调试控制台

```
FFT经过GPU加速之后的时间为 6 ms  
逆变化检测通过。  
FFT经过CPU加速之后的为 45 ms  
逆变换检测通过
```

D:\vs\_pro\cufft2\x64\Debug\cufft2.exe (进程 7364)已退出, 返回代码为: 0。  
按任意键关闭此窗口...

[https://blog.csdn.net/m0\\_37273364](https://blog.csdn.net/m0_37273364)

2^16

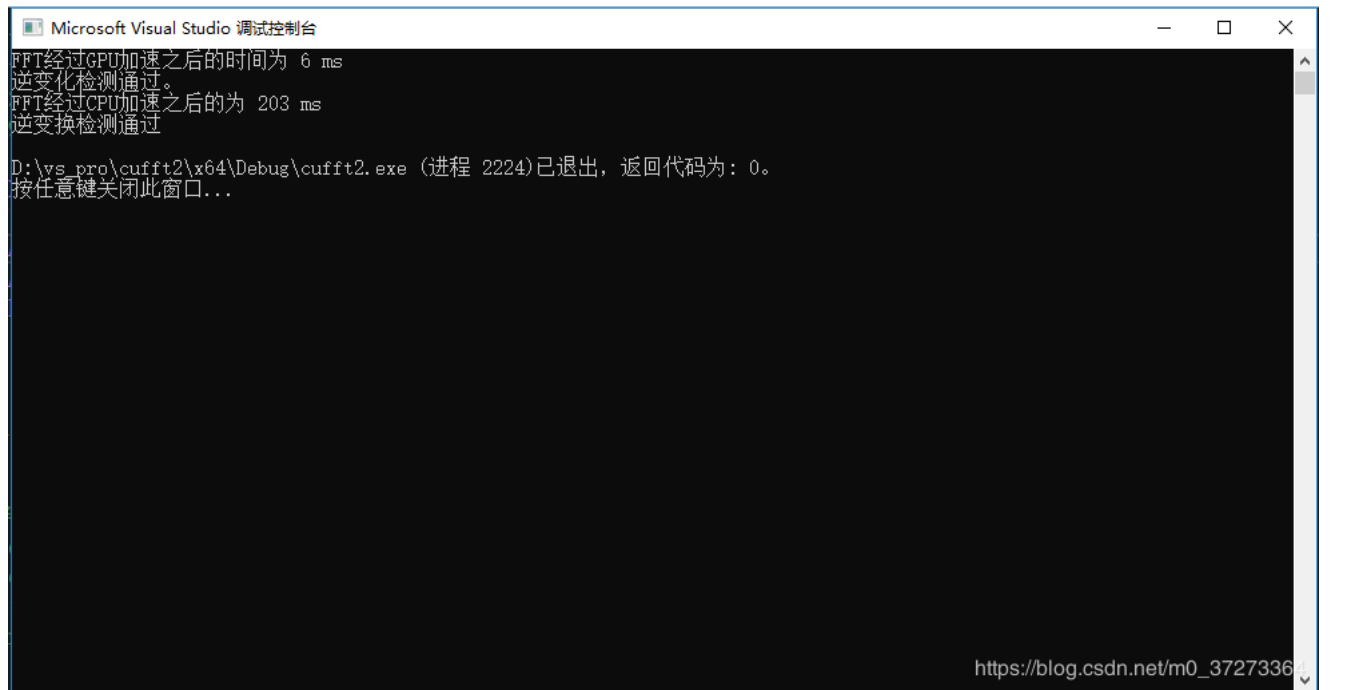


```
选择Microsoft Visual Studio 调试控制台
FFT经过GPU加速之后的时间为 5 ms
逆变换检测通过。
FFT经过CPU加速之后的为 97 ms
逆变换检测通过

D:\vs_pro\cufft2\x64\Debug\cufft2.exe (进程 14024)已退出, 返回代码为: 0。
按任意键关闭此窗口...
```

[https://blog.csdn.net/m0\\_37273364](https://blog.csdn.net/m0_37273364)

2<sup>17</sup>

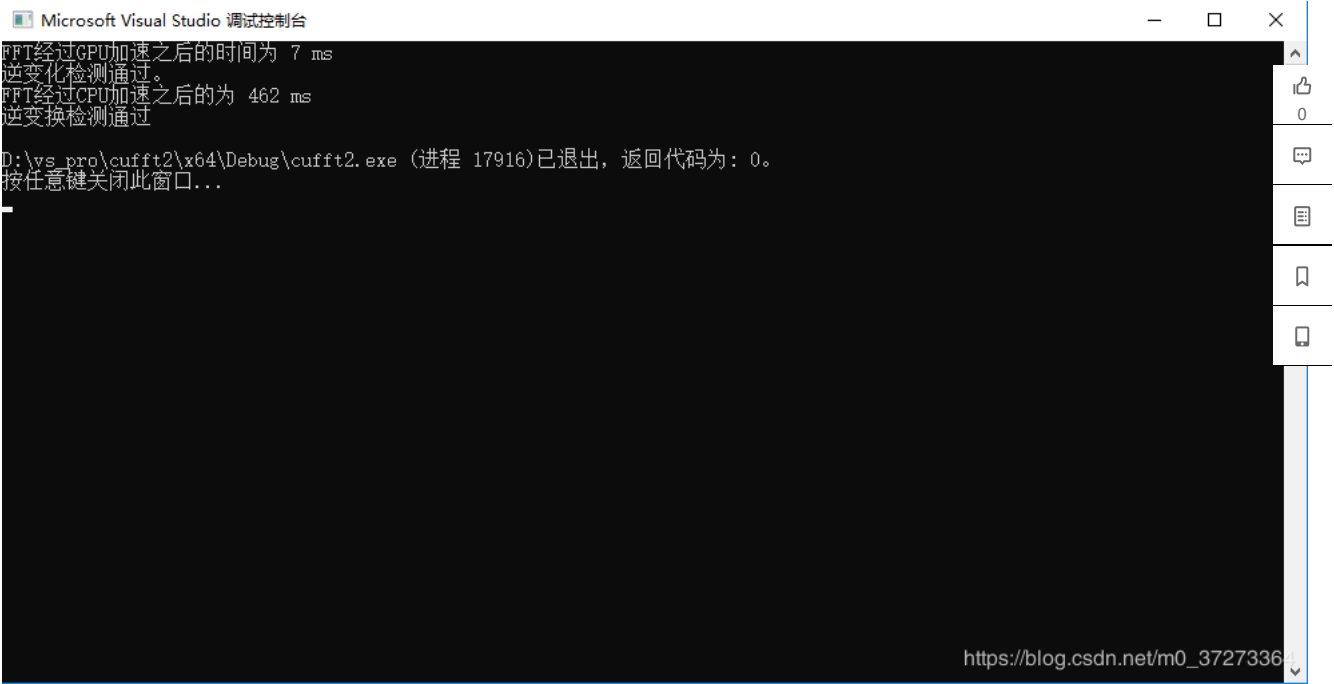


```
Microsoft Visual Studio 调试控制台
FFT经过GPU加速之后的时间为 6 ms
逆变换检测通过。
FFT经过CPU加速之后的为 203 ms
逆变换检测通过

D:\vs_pro\cufft2\x64\Debug\cufft2.exe (进程 2224)已退出, 返回代码为: 0。
按任意键关闭此窗口...
```

[https://blog.csdn.net/m0\\_37273364](https://blog.csdn.net/m0_37273364)

2<sup>18</sup>



六、实验结果分析

根据运行的时间对比，可以看出，随着数据集的增加，CPU的处理时间和数据集的增速基本同步（数据集增大两倍，CPU处理时间增大两倍的）处理时间基本不变。

之前在做小数据集的时候发现经过GPU的加速运行时间和CPU的运行时间不相上下，甚至比CPU还要慢。

通过分析可以知道，当数据集较小时，数据在相关寄存器/内存的传送的时间对总时间的贡献更大主要，而当数据集很大的时候，计算时间对总大。为了比较性能，于是就没有放小数据集的运行情况。

可以看到，对于大数据集，经过GPU加速，计算时间基本维持在了一个常数时间（其中也有程序运行的偶然性以及计时精度的问题，实际上运行性增加，但是速率比CPU的2倍速慢）

七、思考题

分析GPU加速FFT程序可能获得的加速比

理论上，如果GPU可以一次存储完所有的数据，那么相对于CPU，数据传送的时间可以忽略，而CPU能够一次处理的数据块的大小有限，理论是GPU一次可以处理的数据块大小/CPU一次可以处理的数据块大小，因为实际上GPU就是对多个数据块的并行运算。但是这个加速比不可知（需要查手册，但是我没有找到）

实际加速比相对于理想加速比差多少？原因是什么？

原因：

每次运行程序时，数据在CPU/GPU和内存之间的传送时间不同（依赖于当时计算机的运行情况）

GPU上进行的类多线程方式的时间开销以及写回/验证时的可能存在的写冲突

收藏

分享

深度学习卷积算法的GPU加速实现方法

参考链接：<http://blog.csdn.net/u010620604/article/details/52464529> <http://blog.csdn.net/guoyilin...>

博主设置当前文章不允许评论。

MATLAB上的GPU加速计算——学习笔记

4957

MATLAB目前只支持Nvidia的显卡。如果你的显卡是AMD的或者是Intel的，就得考虑另寻它路了。 M..

来自： [xiaotianlan的专栏](#)

傅立叶变换—FFT(cuda实现)

1305

傅立叶变换—FFT(cuda实现)

来自： [jacke121的专栏](#)