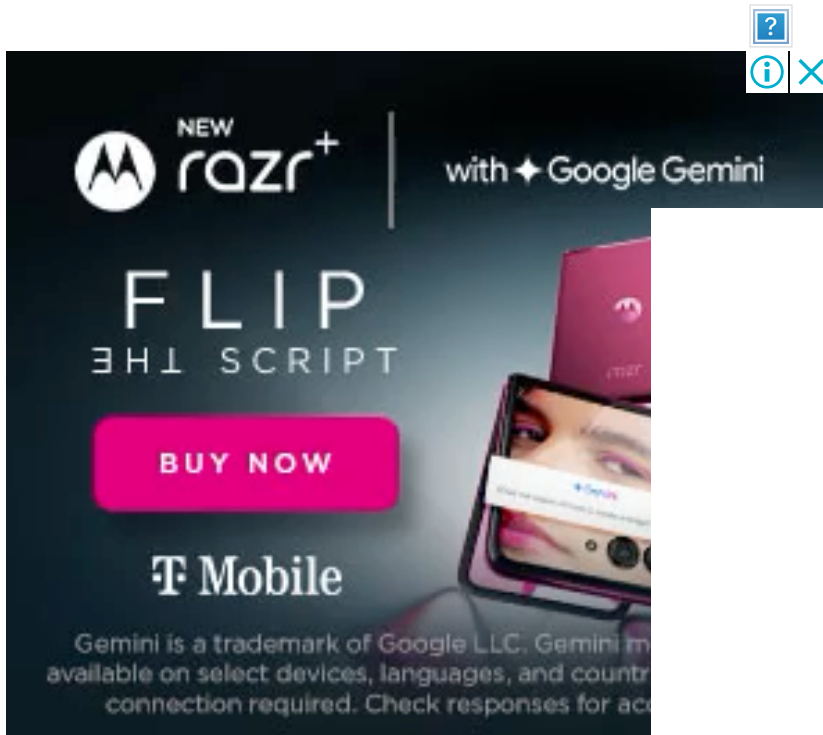




# 谭升的博客

人工智能基础



## 【CUDA 基础】2.2 给核函数计时

📅 2018-03-08 | 📁 [CUDA](#) | [Freshman](#) | 💬 0 | 👁

**Abstract:** 本文介绍CUDA核函数计时方法

**Keywords:** gettimeofday, nvprof

### 给核函数计时

继续更新CUDA，同时概率和数学分析也在更新，欢迎大家访问[www.face2ai.com](http://www.face2ai.com)

昨天晚上开始折腾ubuntu，上一篇用腾讯云搭建服务器来调试CUDA，现在有机子了，所以装个ubuntu准备调试cuda，但是出现了下面的纠结问题，搞了将近五个多小时，才解决，首先我的笔记本是联想R720 1050Ti的显卡，安装ubuntu 16.04 发现源中的驱动安装好后，安装CUDA 9.1 local版本出现问题，没办法安

装成功，以为是驱动问题，安装新的驱动也不行，于是想起来之前用的是17.04，打开镜像网站发现17.04已经不再支持了，找了old版本中，找到下载安装，发现没有源可以用，放弃，安装17.10，开机就出错，于是又退回16.04，安装自带的驱动，安装了cuda 9.0 run版，成功了，安装cmake，ssh-server，于是我们成功了：

A terminal window titled 'CUDA\_Freshman' showing the execution of a CUDA program. The user runs './build/4\_sum\_arrays\_timer/sum\_arrays\_timer' in the directory '~/Project/CUDA\_Freshman'. The output shows it is using device 0 (GeForce GTX 1050 Ti) with a vector size of 32. The execution configuration is <<<1,32>>> and the time elapsed is 0.000010 seconds. The result check is successful.

```
tony@tony-Lenovo:~/Project/CUDA_Freshman$ ./build/4_sum_arrays_timer/sum_arrays_timer
Using device 0: GeForce GTX 1050 Ti
Vector size:32
Execution configuration<<<1,32>>> Time elapsed 0.000010 sec
Check result success!
tony@tony-Lenovo:~/Project/CUDA_Freshman$
```

编程模型中我们介绍了内存，线程相关的知识，接着我们启动了我们的核函数，这些只是大概的勾勒出CUDA编程的外貌，通过前几篇可以写出一般的可运行程序，但是想获得最高的效率，需要反复的优化，以及对硬件和编程细节的详细了解，怎么评估效率，时间是个很直观的测量方式。

## 用CPU计时

使用cpu计时的方法是测试时间的一个常用办法，我记得很有趣的一件事时，我们在写C程序的时候最多使用的计时方法是：

```
1  clock_t start, finish;
2  start = clock();
3  // 要测试的部分
4  finish = clock();
```

```
5 duration = (double)(finish - start) / CLOCKS_PER_SEC;
```

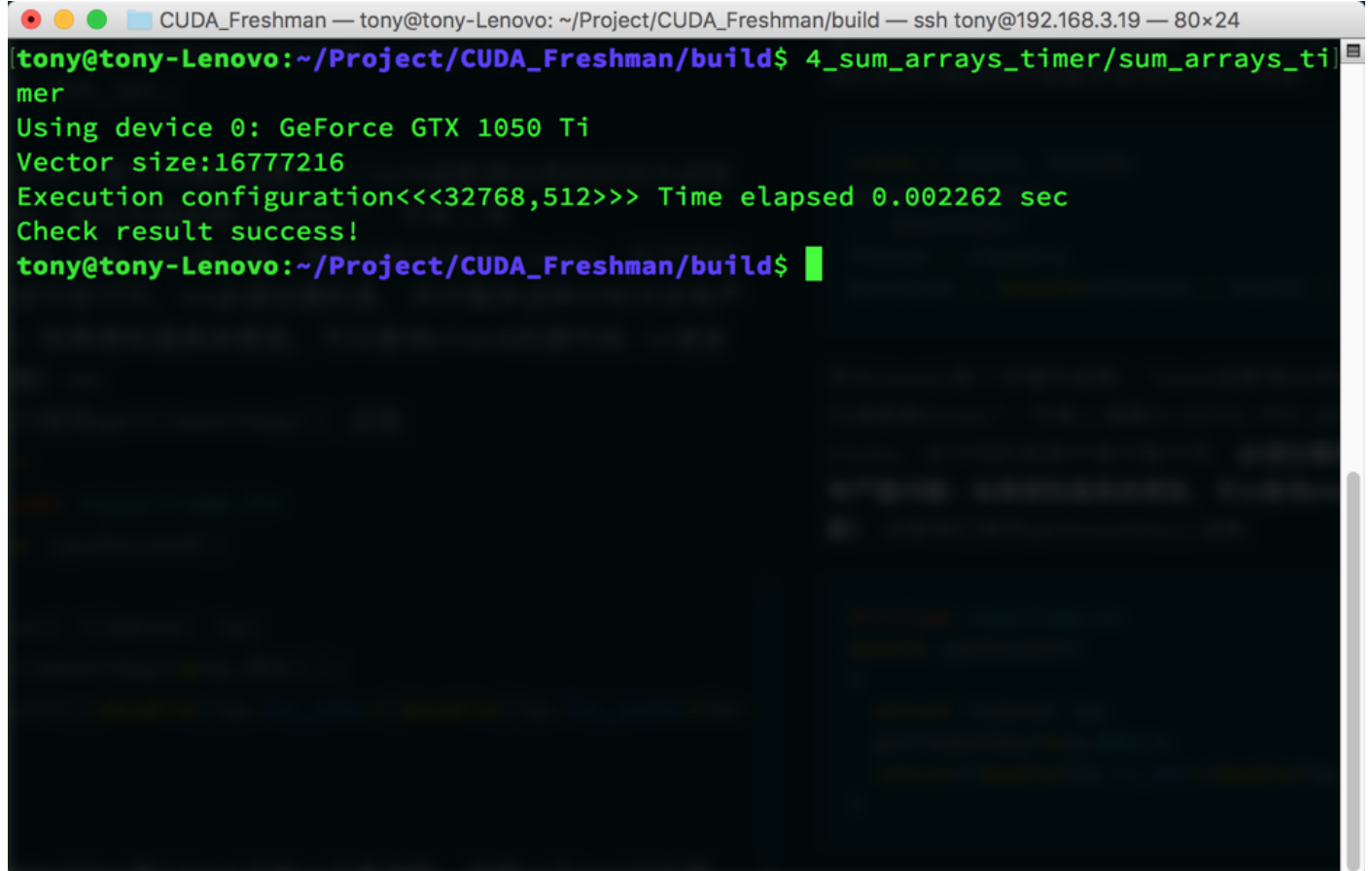
其中clock()是个关键的函数，“clock函数测出来的时间为进程运行时间，单位为滴答数(ticks)”；字面上理解CLOCKS\_PER\_SEC这个宏，就是每秒中多少clocks，在不同的系统中值可能不同。必须注意的是，并行程序这种计时方式有严重问题！如果想知道具体原因，可以查询clock的源代码（c语言标准函数）

这里我们使用gettimeofday() 函数

```
1  #include <sys/time.h>
2  double cpuSecond()
3  {
4      struct timeval tp;
5      gettimeofday(&tp, NULL);
6      return ((double) tp.tv_sec + (double) tp.tv_usec * 1e-6);
7  }
```

gettimeofday是linux下的一个库函数，创建一个cpu计时器，从1970年1月1日0点以来到现在的秒数，需要头文件sys/time.h

那么我们使用这个函数测试核函数运行时间：

A terminal window titled 'CUDA\_Freshman — tony@tony-Lenovo: ~/Project/CUDA\_Freshman/build — ssh tony@192.168.3.19 — 80x24'. The prompt is 'tony@tony-Lenovo:~/Project/CUDA\_Freshman/build\$'. The user has entered '4\_sum\_arrays\_timer/sum\_arrays\_timer'. The output shows: 'Using device 0: GeForce GTX 1050 Ti', 'Vector size:16777216', 'Execution configuration<<<32768,512>>> Time elapsed 0.002262 sec', and 'Check result success!'. The prompt is now 'tony@tony-Lenovo:~/Project/CUDA\_Freshman/build\$' with a green cursor.

```
tony@tony-Lenovo:~/Project/CUDA_Freshman/build$ 4_sum_arrays_timer/sum_arrays_timer
Using device 0: GeForce GTX 1050 Ti
Vector size:16777216
Execution configuration<<<32768,512>>> Time elapsed 0.002262 sec
Check result success!
tony@tony-Lenovo:~/Project/CUDA_Freshman/build$
```

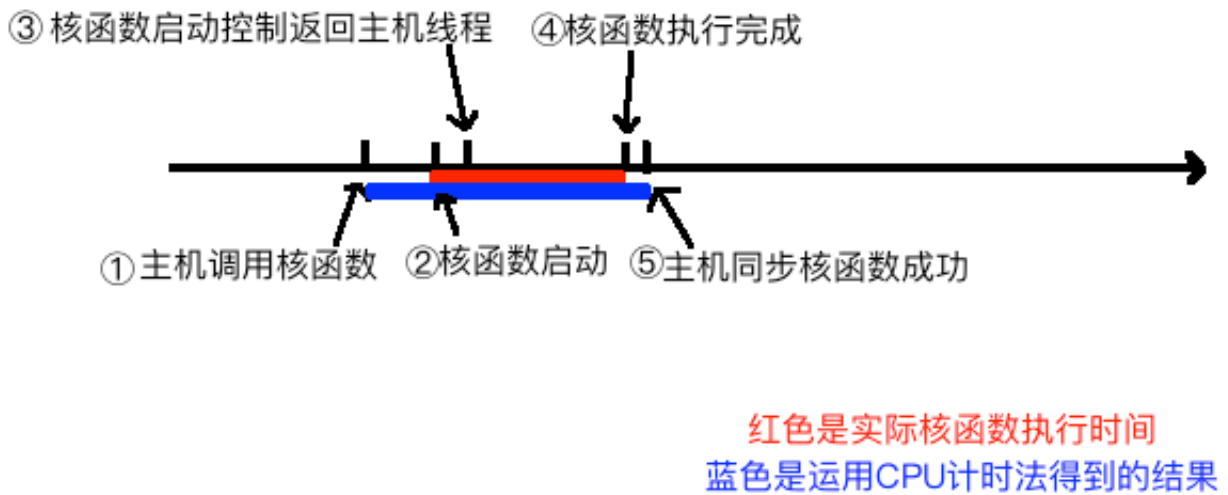
我把代码部分贴出来，完整的访问代码库：[https://github.com/Tony-Tan/CUDA\\_Freshman](https://github.com/Tony-Tan/CUDA_Freshman)

```
1  #include <cuda_runtime.h>
2  #include <stdio.h>
3  #include "freshman.h"
4
5  __global__ void sumArraysGPU(float*a,float*b,float*res,int N)
6  {
7      int i=blockIdx.x*blockDim.x+threadIdx.x;
8      if(i < N)
9          res[i]=a[i]+b[i];
10 }
11 int main(int argc,char **argv)
12 {
13     // set up device.....
14
15     // init data .....
16
17     //timer
18     double iStart,iElaps;
19     iStart=cpuSecond();
20     sumArraysGPU<<<grid,block>>>(a_d,b_d,res_d,nElem);
21     cudaDeviceSynchronize();
22     iElaps=cpuSecond()-iStart;
23
24     // .....
25 }
```

主要分析计时这段，首先iStart是cpuSecond返回一个秒数，接着执行核函数，核函数开始执行后马上返回主机线程，所以我们要加一个同步函数等待核函数执行完毕，如果不加这个同步函数，那么测试的时间是从调用核函数，到核函数返回给主机线程的时间段，而不是核函数的执行时间，加上了

```
1  cudaDeviceSynchronize();
```

函数后，计时是从调用核函数开始，到核函数执行完并返回给主机的时间段，下面图大致描述了执行过程的不同时间节点：



我们可以大概分析下核函数启动到结束的过程：

1. 主机线程启动核函数
2. 核函数启动成功
3. 控制返回主机线程
4. 核函数执行完成
5. 主机同步函数检测到核函数执行完

我们要测试的是2~4的时间，但是用CPU计时方法，只能测试1~5的时间，所以测试得到的时间偏长。

接着我们调整下我们的参数，来看看不同线程维度对速度的影响，看看计时能不能反映出来点问题，这里我们考虑一维线程模型

- 2的幂次数据量  $1 < 24$ ，16兆数据：
  - 每个块256个线程

```
CUDA_Freshman — tony@tony-Lenovo: ~/Project/CUDA_Freshman/build — ssh tony@192.168.3.19 — 80×24
tony@tony-Lenovo:~/Project/CUDA_Freshman/build$ 4_sum_arrays_timer/sum_arrays_timer
Using device 0: GeForce GTX 1050 Ti
Vector size:16777216
Execution configuration<<<65536,256>>> Time elapsed 0.002313 sec
Check result success!
tony@tony-Lenovo:~/Project/CUDA_Freshman/build$
```

- 每个块512个线程

```
CUDA_Freshman — tony@tony-Lenovo: ~/Project/CUDA_Freshman/build — ssh tony@192.168.3.19 — 80×24
tony@tony-Lenovo:~/Project/CUDA_Freshman/build$ 4_sum_arrays_timer/sum_arrays_timer
Using device 0: GeForce GTX 1050 Ti
Vector size:16777216
Execution configuration<<<32768,512>>> Time elapsed 0.002263 sec
Check result success!
tony@tony-Lenovo:~/Project/CUDA_Freshman/build$
```

- 每个块1024个线程

```
CUDA_Freshman — tony@tony-Lenovo: ~/Project/CUDA_Freshman/build — ssh tony@192.168.3.19 — 80×24
tony@tony-Lenovo:~/Project/CUDA_Freshman/build$ 4_sum_arrays_timer/sum_arrays_timer
Using device 0: GeForce GTX 1050 Ti
Vector size:16777216
Execution configuration<<<16384,1024>>> Time elapsed 0.002282 sec
Check result success!
tony@tony-Lenovo:~/Project/CUDA_Freshman/build$
```

- 2的非幂次数据量 ( $1 \ll 24$ )+1, 16兆加一个数据:
  - 每个块256个线程

```
CUDA_Freshman — tony@tony-Lenovo: ~/Project/CUDA_Freshman/build — ssh tony@192.168.3.19 — 80×24
tony@tony-Lenovo:~/Project/CUDA_Freshman/build$ 4_sum_arrays_timer/sum_arrays_timer
Using device 0: GeForce GTX 1050 Ti
Vector size:33554432
Execution configuration<<<131072,256>>> Time elapsed 0.004364 sec
Check result success!
tony@tony-Lenovo:~/Project/CUDA_Freshman/build$
```

- 每个块512个线程

```
CUDA_Freshman — tony@tony-Lenovo: ~/Project/CUDA_Freshman/build — ssh tony@192.168.3.19 — 80×24
tony@tony-Lenovo:~/Project/CUDA_Freshman/build$ 4_sum_arrays_timer/sum_arrays_timer
Using device 0: GeForce GTX 1050 Ti
Vector size:33554432
Execution configuration<<<65536,512>>> Time elapsed 0.004400 sec
Check result success!
tony@tony-Lenovo:~/Project/CUDA_Freshman/build$
```



- 每个块1024个线程

A terminal window titled 'CUDA\_Freshman — tony@tony-Lenovo: ~/Project/CUDA\_Freshman/build — ssh tony@192.168.3.19 — 80x24'. The prompt is 'tony@tony-Lenovo:~/Project/CUDA\_Freshman/build\$'. The user has entered '4\_sum\_arrays\_timer/sum\_arrays\_timer'. The output shows: 'Using device 0: GeForce GTX 1050 Ti', 'Vector size:33554432', 'Execution configuration<<<32768,1024>>> Time elapsed 0.004366 sec', and 'Check result success!'. The prompt returns to 'tony@tony-Lenovo:~/Project/CUDA\_Freshman/build\$'.

对于我这个cpu这三个参数的性能差距比较小，但是需要注意的是当数据不能被完整切块的时候性能滑铁卢了，这个我们可以使用一点小技巧，比如只传输可完整切割数据块，然后剩下的1，2个使用cpu计算，这种技巧后面有介绍，以及包括如何选择系数。我们本篇之关系计时函数的工作状态，目前看起来还不错。

## 用nvprof计时

CUDA 5.0后有一个工具叫做nvprof的命令行分析工具，后面还要介绍一个图形化的工具，现在我们来学习一下nvprof，学习工具主要技巧是学习工具的功能，当你掌握了一个工具的全部功能，那就是学习成功了。

nvprof的用法如下：

```
1 $ nvprof [nvprof_args] <application>[application_args]
```

于是我们执行命令得到

```
CUDA_Freshman — tony@tony-Lenovo: ~/Project/CUDA_Freshman/build/4_sum_arrays_timer — ssh tony@192.168.3.19 —...
[tony@tony-Lenovo:~/Project/CUDA_Freshman/build/4_sum_arrays_timer$ nvprof ./sum_arrays_timer
==25386== NVPROF is profiling process 25386, command: ./sum_arrays_timer
Using device 0: GeForce GTX 1050 Ti
Vector size:16777216
===== Error: unified memory profiling failed.
tony@tony-Lenovo:~/Project/CUDA_Freshman/build/4_sum_arrays_timer$
```

出现错误：

```
1  ===== Error: unified memory profiling failed.
```

原因是权限问题，因为安全原因，Mac os和Linux当你调试程序时，一个程序（比如IDE）要接入别的进程（被调试进程），这时候需要权限保证安全，否则一些坏程序会肆意干扰别的程序，出现问题，所以操作系统不允许线程间任意通信。

解决办法是加上sudo，但是还是有问题，超级用户的环境变量里没有nvprof：

```
CUDA_Freshman — tony@tony-Lenovo: ~/Project/CUDA_Freshman/build/4_sum_arrays_timer — ssh tony@192.168.3.19 —...
[tony@tony-Lenovo:~/Project/CUDA_Freshman/build/4_sum_arrays_timer$ sudo nvprof ./sum_arrays_timer
sudo: nvprof: 找不到命令
tony@tony-Lenovo:~/Project/CUDA_Freshman/build/4_sum_arrays_timer$
```

如果我们使用完整的nvprof路径加上sudo执行

```
CUDA_Freshman — tony@tony-Lenovo: ~/Project/CUDA_Freshman/build/4_sum_arrays_timer — ssh tony@192.168.3.19 — 123x28
/usr/local/cuda/bin/nvprof
tony@tony-Lenovo:~/Project/CUDA_Freshman/build/4_sum_arrays_timer$ sudo /usr/local/cuda/bin/nvprof ./sum_arrays_timer
==25442== NVPROF is profiling process 25442, command: ./sum_arrays_timer
Using device 0: GeForce GTX 1050 Ti
Vector size:16777216
Execution configuration<<<16384,1024>>> Time elapsed 0.002269 sec
Check result success!
==25442== Profiling application: ./sum_arrays_timer
==25442== Profiling result:
   Type  Time(%)   Time     Calls   Avg       Min       Max  Name
GPU activities:  61.43%  26.535ms      2  13.268ms  13.181ms  13.354ms  [CUDA memcpy HtoD]
               33.70%  14.557ms      1  14.557ms  14.557ms  14.557ms  [CUDA memcpy DtoH]
               4.86%   2.1011ms      1   2.1011ms  2.1011ms  2.1011ms  sumArraysGPU(float*, float*, float*, int)
API calls:      72.81%  131.50ms      3  43.832ms  293.94us  130.89ms  cudaMalloc
               22.99%  41.510ms      3  13.837ms  13.326ms  14.742ms  cudaMemcpy
               2.49%   4.4985ms      3   1.4995ms  237.61us  2.1331ms  cudaFree
               1.24%   2.2402ms      1   2.2402ms  2.2402ms  2.2402ms  cudaDeviceSynchronize
               0.20%   365.47us     94   3.8870us    125ns   160.97us  cuDeviceGetAttribute
               0.19%   352.14us      1   352.14us   352.14us   352.14us  cuDeviceGetProperties
               0.03%   55.473us      1   55.473us   55.473us   55.473us  cuDeviceTotalMem
               0.02%   39.777us      1   39.777us   39.777us   39.777us  cuDeviceGetName
               0.01%   23.903us      1   23.903us   23.903us   23.903us  cudaLaunch
               0.00%   4.5850us      1   4.5850us   4.5850us   4.5850us  cudaSetDevice
               0.00%   1.5450us      3     515ns    135ns    1.0970us  cuDeviceGetCount
               0.00%   1.2670us      4     316ns    131ns     657ns  cudaSetupArgument
               0.00%   1.0350us      1   1.0350us   1.0350us   1.0350us  cudaConfigureCall
               0.00%     718ns      2     359ns    182ns    536ns  cuDeviceGet
tony@tony-Lenovo:~/Project/CUDA_Freshman/build/4_sum_arrays_timer$
```

工具不仅给出了kernel执行的时间，比例，还有其他cuda函数的执行时间，可以看出核函数执行时间只有4%左右，其他内存分配，内存拷贝占了大部分事件，nvprof给出的核函数执行时间2.1011ms，上面cpuSecond计时结果是2.282ms

可见，nvprof可能更接近真实值。

nvprof这个强大的工具给了我们优化的目标，分析数据可以得出我们重点工作要集中在哪部分。

## 理论界限最大化

得到了实际操作值，我们需要知道的是我们能优化的极限值是多少，也就是机器的理论计算极限，这个极限我们永远也达不到，但是我们必须明确的知道，比如理论极限是2秒，我们已经从10秒优化到2.01秒了，基本就没有必要再继续花大量时间优化速度了，而应该考虑买更多的机器或者更新的设备。

各个设备的理论极限可以通过其芯片说明计算得到，比如说：

- Tesla K10 单精度峰值浮点数计算次数： $745\text{MHz核心频率} \times 2\text{GPU/芯片} \times (8\text{个多处理器} \times 192\text{个浮点计算单元} \times 32\text{核心/多处理器}) \times 2\text{ OPS/周期} = 4.58\text{ TFLOPS}$
- Tesla K10 内存带宽峰值： $2\text{GPU/芯片} \times 256\text{位} \times 2500\text{MHz内存时钟} \times 2\text{ DDR/8位/字节} = 320\text{ GB/s}$
- 指令比： $\text{字节 } 4.58\text{ TFLOPS} / 320\text{ GB/s} = 13.6\text{ 个指令：1个字节}$

## 总结

本文我们简单介绍了CUDA核函数的计时方法，以及如何评估理论时间下届，也就是效率的极限值，了解性能瓶颈和性能极限，是优化性能的第一步。

转载请标明来源[www.face2ai.com](http://www.face2ai.com)

本文作者：谭升

本文链接：<https://face2ai.com/CUDA-F-2-2-核函数计时/>

版权声明：本博客所有文章除特别声明外，均采用 [CC BY-NC-SA 4.0](https://creativecommons.org/licenses/by-nc-sa/4.0/) 许可协议。转载请注明出处！

### 相关文章

- [【CUDA 基础】3.3 并行性表现](#)
- [【Julia】整型和浮点型数字](#)
- [【Julia】变量](#)
- [【Julia】开始使用Julia](#)