文章编号：1672-2892(2010)03-0368-04

利用 CUDA 实现上位机的实时目标跟踪

邓 浩[1]，杨 菲[2]，潘旭东[1]，游安清[1]

(1.中国工程物理研究院 应用电子学研究所，四川 绵阳 621900；
2.中国工程物理研究院 电子工程研究所，四川 绵阳 621900)

摘 要：在上位机进行实时目标跟踪，使用传统的 CPU 进行计算往往由于数据处理量大而消耗很多计算时间，影响实时性和跟踪效果。近年来，nVidia 公司提出的 CUDA 架构利用 GPU 进行并行计算，极大提高了运算速度。本文在介绍 CUDA 架构的特性及软硬件实现原理的基础上，利用 CUDA 来实现上位机的实时目标跟踪，并与传统方法的计算速度进行了比较。结果表明，CUDA 的应用使上位机目标跟踪的实时性得到了很大提升，可以将其应用于其它众多领域。

关键词：并行计算；图像处理单元；统一计算设备架构；目标跟踪
中图分类号：TN918 **文献标识码：**A

Real-time target tracking on upper computer using CUDA

DENG Hao[1], YANG Fei[2], PAN Xu-dong[1], YOU An-qing[1]

(1.Institute of Applied Electronics, China Academy of Engineering Physics, Mianyang Sichuan 621900, China;
2.Institute of Electronic Engineering, China Academy of Engineering Physics, Mianyang Sichuan 621900, China)

Abstract: Real-time target tracking on upper computer using CPU will spend much time because of a great amount of calculation, which will affect the tracking. In recent years, nVidia Corporation has put forward a compute architecture called Compute Unified Device Architecture(CUDA) which calculates in parallel using Graphics Processing Units(GPU). This study introduced the architecture and principle of CUDA, then tracked target in real time using this method, and compared the calculating speed of CUDA with that of traditional method. The result showed that CUDA could speed up calculation and be well used in real-time target tracking on upper computer.

Key words: parallel calculation; Graphics Processing Units; Compute Unified Device Architecture; target tracking

在上位机实现的实时目标跟踪，其基本原理都是首先通过采集卡把摄像头得到的模拟视频转换为数字图像，再利用某种跟踪算法通过 CPU 的计算从数字图像中得到目标的信息。但是由于图像的数据量大，跟踪算法越来越复杂等原因，计算耗时也越来越长，不断发展的计算机硬件往往也难以满足实时的要求(PAL 制式的摄像头两帧图像间隔时间仅 40 ms)，极大影响跟踪效果。

近几年来，图像处理单元(GPU)的发展极为迅猛，可编程的 GPU 已发展成为一种高度并行化、多线程、多核心的处理器。由于 GPU 通常具有很高的内存带宽，以及海量的执行单元，因此逐渐开始利用 GPU 来实现一些计算工作，即基于 GPU 的通用计算(General-Purpose computation on GPU，GPGPU)。在此背景下，全球最大的图形图像处理芯片开发公司 nVidia 提出了一套 GPGPU 模型——统一计算设备架构(CUDA)。CUDA 专门为并行计算设计，数据并行处理会将数据元素映射到并行处理线程进行加速计算，且 CUDA 的编程语法与 C 语言类似，这样一来，利用 CUDA 实现上位机的实时目标跟踪就有了可能。

1 GPU 和 CUDA 的基本理论

1.1 GPU 和 CPU 的比较

浮点运算能力是评价科学计算最主要的指标之一，目前，nVidia 和 Intel 两大公司主流 GPU 和 CPU 的浮点

收稿日期：2009-11-04；修回日期：2009-12-15

运算能力及内存带宽比较如图 1 所示[1]。由图 1 可以看出，GPU 的浮点计算能力和内存带宽都远超过 CPU，原因在于 GPU 专为计算密集型、高度并行化的计算而设计，能使更多晶体管用于数据处理，而非数据缓存和流控制[2]。简单地说，GPU 专用于解决可表示为数据并行计算的问题。

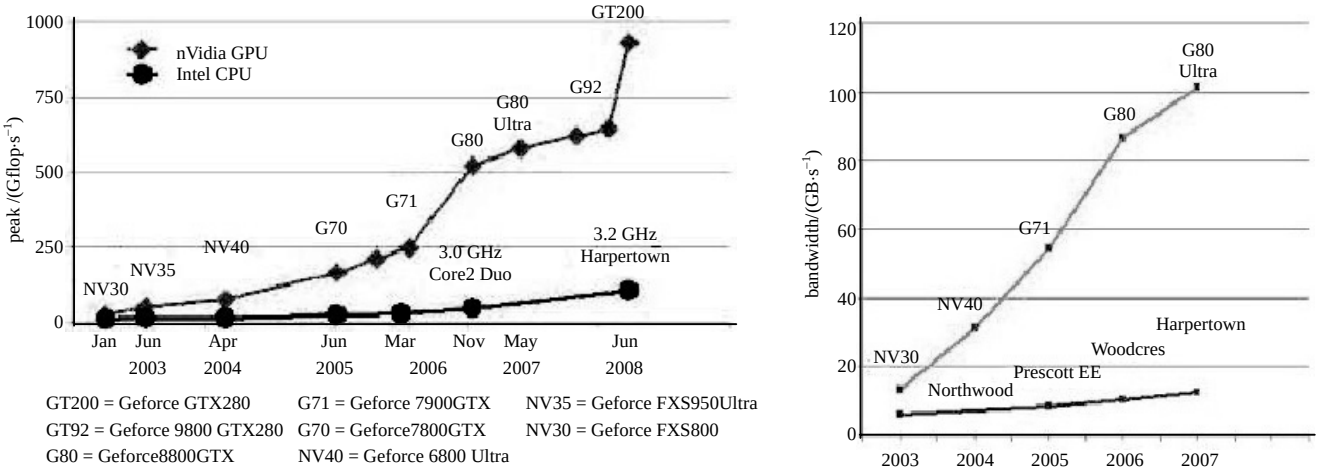

Fig.1 Comparison of calculation ability and bandwidth between GPU and CPU
图 1 GPU 和 CPU 的浮点计算能力和带宽比较

使用 GPU 来进行计算工作，与使用 CPU 相比，主要优点有[3]：

GPU 通常具有更大的内存带宽。例如新一代的 nVidia 旗舰产品 GeforceGTX295 具有超过 200 GB 的内存带宽，而 Intel 的高端 CPU 内存带宽则在 20 GB 左右。

GPU 具有更多的执行单元。例如 GeforceGTX285 有 240 个流处理器(stream processor)，而 CPU 的执行单元虽然频率较高，但数目少得多。和 CPU 相比，GPU 的价格相对低廉。

总之，GPU 适合同时进行大量类似的工作；而 CPU 比较灵活，能同时进行变化较多的工作。

1.2 CUDA 架构

CUDA 是 nVidia 公司提出的并行编程模型和软件环境，它使用 C 语言为基础，可以编译生成直接在 GPU 上执行的程序，CUDA 的核心有 3 个重要抽象概念：线程组层次结构、共享存储器、屏蔽同步[4]。这些抽象提供了数据并行化和线程并行化，并嵌套于粗粒度的数据并行化和任务并行化之中。这样的分解保留了语言表达，允许线程在解决各子问题时协作，同时支持透明的可伸缩性，可以安排在任何可用 GPU 上处理各个子问题。CUDA 软件栈的构架如图 2 所示[1]。

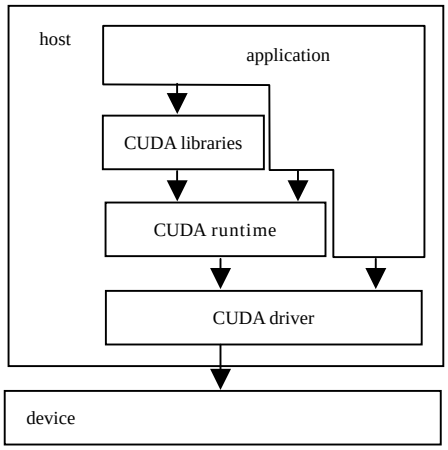

Fig.2 Software constitution of CUDA
图 2 CUDA 架构软件栈

1.3 CUDA 编程与执行

CUDA 架构下，一个完整程序分为 2 个部份：host 端和 device 端。host 端是指在 CPU 上执行的部份，而 device 端则是在 GPU 执行的部份，又称为内核(kernel)，内核函数用 C 语言编写。通常 host 端程序会将参与计算的数据准备好，复制到显卡的内存中，再由 GPU 执行 device 端程序，完成后再由 host 端程序将结果从显卡的内存中取回。值得注意的是 kernel 函数将 GPU 的线程(thread)并行执行多次，这与 host 端程序按照语句顺序执行的方式不同。

线程是 GPU 执行 kernel 函数时的最小单位。一个或多个线程可以组成一个块(block)，每一个块所能包含的线程数目是有限的，

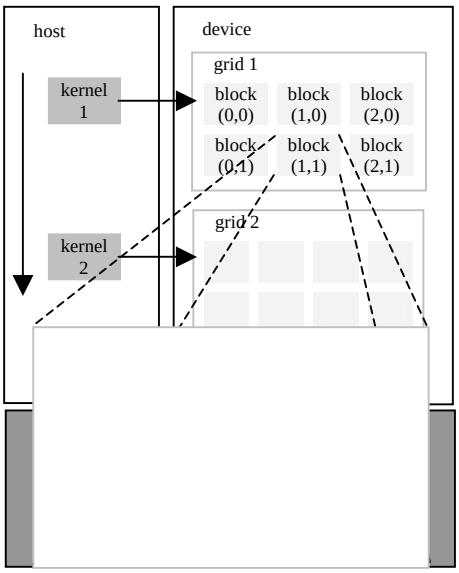

Fig.3 Relation of thread, block and grid in CUDA
图 3 CUDA 中线程、块和网格的相互关系

若干个执行相同程序的块，又可以组成网格(grid)。网格对应着内核的执行，在目前的 CUDA 架构中，一个内核只允许一个网格存在。线程、块和网格的相互关系如图 3 所示[1]。内核的执行就是 GPU 中的流处理器运行线程组的过程。

2 利用 CUDA 实现目标的跟踪

2.1 问题的数学模型及其传统解决方法

在某些背景较暗，而目标亮度较高的情况下，跟踪图像经过了背景去噪、对比度提升等预处理后，目标跟踪简化为在图像中寻找最亮区域的问题，主要数学模型如图 4 所示。

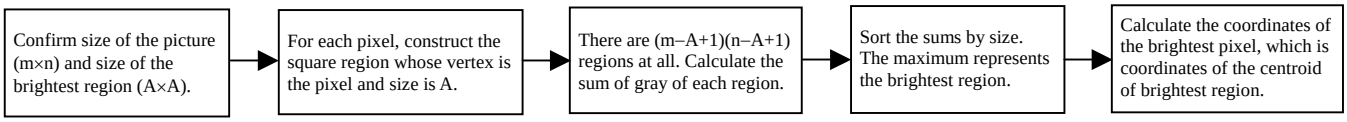

Fig.4 Mathematic model of target tracking
图 4 目标跟踪的数学模型

由图 4 可知，计算过程中最重要且计算量最大的是第 3 步。传统的解决方法一般是利用双重循环，依次取出图像的每一个像素，计算出该像素代表的区域的灰度值之和，主要代码如下：

```
for (int i = 0; i < n; i++)
{
    for (int j = 0; j < m; j++)
    {
        W(i, j) = a(i, j)+ a(i+1, j)+…+a(i+A–1, j)+ a(i+A-1, j+1)+…+ a(i+A–1, j+A–1);// a(i, j)代表像素灰度
    }
}
```

由上述代码可以看出，求图像中每个区域的亮度由 CPU 顺序执行，程序的执行时间主要依赖于 CPU 的计算速度。

2.2 CUDA 的并行化处理

分析上述问题，图像中每个区域的灰度和其实相对独立，可以并行处理，依据 CUDA 的编程模型，求每一个区域亮度和的工作用一个线程实现，线程的分布和图像的元素排布一一对应。由于每个块含有的线程数目是有限的，因此根据图像的尺寸构建出合适的块，整个问题对应一个网格。

核心函数主要代码如下：

```
__global__ static void BrightnessCal(int nWidth, int A, unsigned int * sumgrey)
{
    int tidx = blockIdx.x * blockDim.x + threadIdx.x;   // 取得每一个线程在 Grid 中的 index
    int tidy = blockIdx.y * blockDim.y + threadIdx.y;
    for (int i = tidy; i < tidy + bm; i++)
    {
        for (int j = tidx; j < tidx + bm; j++)              // bm 为最亮区域的尺寸
        sum += tex2D(tex, j, i);                            // tex2D(tex, j, i)为位置(j, i)的像素灰度值
    }
    syncthreads();
    sumgrey[tidy * (nWidth – A + 1) + tidx] = sum;   // 有效区域内除右下波门区域之外，剩下的点连续排列
}
```

2.3 计算速度的比较

在同一计算平台上，分别使用传统方法和 CUDA 对同一跟踪视频(PAL 制式，分辨率 640×480)进行处理。比较 2 种方法找出的最亮点坐标，结果完全一致，典型的跟踪结果如图 5 所示，计算时间的比较如图 6 所示。

由图 6 可以看出，使用 CUDA 进行目标跟踪，单幅图像的平均计算耗时缩减为传统计算的 1/8 左右，计算时间更加稳定，视频的帧率由原来 10 帧左右上升至 25 帧，大大提升了跟踪效果。

接下来使用 CUDA 和传统方法对不同分辨率的视频进行目标跟踪计算，计算时间随着视频尺寸的变化情况如下(其中每项时间都是运行程序 10 次取的平均值)。

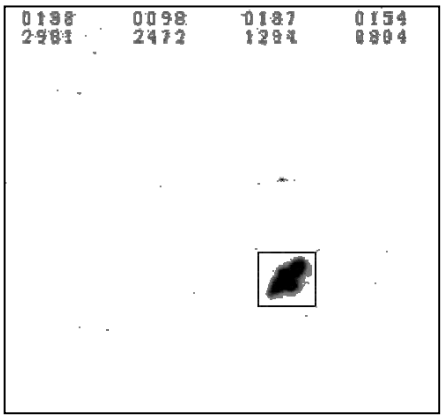

Fig.5 Typical result of target tracking
图 5 典型的目标跟踪结果

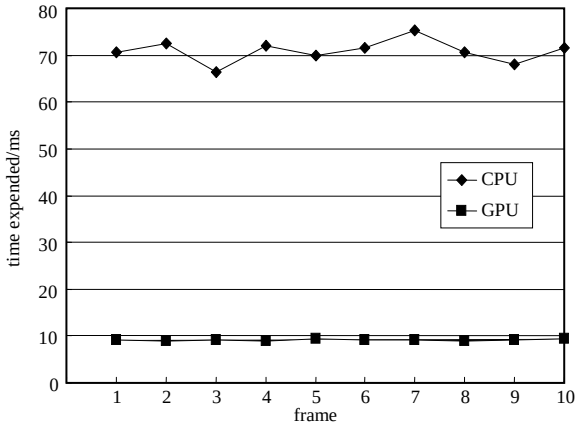

Fig.6 Time-consuming of two calculation methods
图 6 两种方法计算耗时比较

由表 1 可以看出，使用 CUDA 计算架构在 GPU 上计算，速度均远高于传统模式的 CPU，且随着计算量的不断增大，这个差距越来越大。这是因为 CPU 的计算是顺序执行的，计算时间随着计算量的增长线性增加；而在一定的计算量范围之内，GPU 每个线程的计算时间是一定的，计算时间的增长仅仅由于更多的线程和块进行硬件交互引起的必要的开销[5]。另外，使用传统方法进行计算时，CPU 易受到其它程序进程的影响，导致计算时间有不小的抖动；而 GPU 计算一般不存在这种情况，稳定度更高。

表 1 CPU 和 GPU 的计算用时

Table1 Time-consuming of CPU and GPU

image size	time-consuming of CPU(Pentium D 3.4 GHz)/ms	time-consuming of GPU(9800GT)/ms	speedup
320×240	19.6	3.1	6.3×
704×576	80.5	11.2	7.2×
800×600	108.0	14.3	7.6×
1 024×768	170.4	17.5	9.7×

3　结论

CUDA 是一种新型的硬件和软件架构，用于将数据在 GPU 上进行计算的发放和管理，它的出现开辟了科学计算的新纪元，并行计算使得许多传统的计算在速度上有了数量级的提高。利用 CUDA 使基于上位机的目标跟踪的实时性得到了很好的提升，对于模拟视频的处理时间远远小于两帧图像的时间间隔，所需的主要工作仅仅是算法的移植而已。CUDA 在 3D 渲染、数据分析、图像和媒体处理等领域也有相当出色的表现。今后可开展平台进行相关研究，使更多的传统工作受益于 CUDA。

参考文献：

[1] NVIDIA. NVIDIA CUDA Programming Guide V2.1[R]. nVidia Corporation, 2008.
[2] 邓仰东. NVIDIA CUDA 超大规模并行程序设计训练课程[R]. 北京:清华大学, 2008.
[3] 张舒,褚艳利. GPU 高性能运算之 CUDA[M]. 北京:中国水利水电出版社, 2009.
[4] NVIDIA. NVIDIA CUDA Device Architecture V2.0[R]. nVidia Corporation, 2008.
[5] NVIDIA. CUDA Tutorial[R]. nVidia Corporation, 2008.

作者简介：

邓　浩(1982–)，男，四川省泸州市人，硕士，研究实习员，主要从事软件设计. email:denghao0404@sina.com.

杨　菲(1981–)，女，四川省绵阳市人，硕士，研究实习员，主要从事 FPGA、DSP 等硬件设计开发.

潘旭东(1972–)，男，四川省绵阳市人，学士，高级工程师，主要从事软硬件开发.

游安清(1975–)，男，武汉市人，博士，高级工程师，主要从事软件开发及算法设计.