

中山大学移动信息工程学院本科生实验报告

(2015 学年春季学期)

课程名称: OpenCL 与高性能计算

任课教师: 黄凯老师

课程名称	OpenCL 与高性能计算	实验内容	基于 OpenCL 加速车辆检测
实现的功能	通过 OpenCL 的优化, 车辆检测的程序运行速度有明显的提高, 时间大大减少。		
小组成员姓名和学号	12353013 陈坚达, 12353020 陈秋良, 12353022 陈胜杰 12353021 陈上宇, 12353088 柯柳, 12353050 谷枫		

一. 实验题目

基于 OpenCL 的车辆检测。

二. 实验原理

1. OpenCL 编程

OpenCL 是一个为异构平台编写程序的框架, 此异构平台可由 CPU, GPU 或其他类型的处理器组成。OpenCL 编程的主要步骤为:

- 初始化平台, 调用两次 `clGetPlatformIDs` 函数, 第一次获取可用的平台数量, 第二次获取一个可用的平台。
- 初始化设备, 调用两次 `clGetDeviceIDs` 函数, 第一次获取可用的设备数量, 第二次获取一个可用的设备。
- 创建上下文, 调用 `clCreateContext` 函数, 上下文 context 可能会管理多个设备 device。
- 创建命令队列, 调用 `clCreateCommandQueue` 函数, 上下文 context 将命令发送到设备对应的 command queue, 设备就可以执行命令队列里的命令。
- 创建设备缓存 (内存), Buffer 由上下文 context 创建, 这样上下文管理的多个设备就会共享 Buffer 中的数据, buffer 中保存内核执行需要的数据。
- 把数据从 host 内存拷贝到 device 内存, 调用 `clEnqueueWriteBuffer` 函数。
- 创建程序对象, 程序对象就代表你的程序源文件或者二进制代码数据。
- 创建 kernel, 根据程序对象, 生成 kernel 对象, 表示设备程序的入口。
- 设置 kernel 的参数, 调用 `clSetKernelArg` 函数
- 配置 work-item 结构, 维数, group 组成等
- 放入队列执行, 将 kernel 对象, 以及 work-item 参数放入命令队列中进行执行。
- 把 device 执行结果拷贝到 host 内存
- 释放 OpenCL 资源, 程序结束。

2. 车辆检测

车辆检测主要是利用 OpenCV 来实现的。

车辆检测算法主要使用了 HOG(特征)+SVM(分类器)的方法, 这种方法最早由 Navneet Dalal 等人提出用于行人检测。此后, 很多研究人员也提出了很多改进的检测算法, 但基本都以该算法为基础框架, 就连 state of art 的 DPM 算法, 也是用改进的 HOG 和改进的 SVM 加上弹簧模型实现的。

SVM 是一种基于最优化分隔的监督学习模型, 因此它有训练和预测两个过程。SVM 的

好处在于训练过程比较快，而且给大家的代码中已经带有一个训练好的 SVM 模型，因此 SVM 的训练过程不在此处详细描述。

HOG (Histogram of Oriented Gradient, 方向梯度直方图) 特征是一种在计算机视觉和图像处理中用来进行物体检测的特征描述子。它通过计算和统计图像局部区域的梯度方向直方图来构成特征。在一副图像中，局部目标的表象和形状能够被梯度或边缘的方向密度分布很好地描述。本质上是梯度的统计信息，而梯度主要存在于边缘的地方。首先将图像分成小的连通区域，把它叫块 (block)；对光照变化和阴影获得更好的效果，在每个块中做对比度归一化，再将块分成更小的连通区域，叫做细胞单元 (cell)。然后采集细胞单元中各像素点的梯度的或边缘的方向直方图。最后把这些直方图组合起来就可以构成特征描述子。

SVM(Support Vector Machine, 支持向量机)，它是一种二类分类模型，其基本模型定义为特征空间上的间隔最大的线性分类器，其学习策略便是间隔最大化，最终可转化为一个凸二次规划问题的求解。对于一个已经训练完成的 SVM 模型，用于分隔高维空间的超平面已经被确定，只需要将被预测的向量和支撑向量相乘求得截距，就可以预测向量所属的类别。

对于个固定大小的图片，提取 HOG 特征的向量维度是相同的，如果提取一张和训练集图片相同大小的图片的 HOG 特征，就能用 SVM 判断这张图片有车一类还是没有车一类。车辆检测问题是找出车在一张大尺度图片中的位置，我们通过枚举所有位置，截取这个位置的图片提取 HOG 特征，用 SVM 做预测，判断这个位置是否是车，从而解决“找到车的位置”这个问题。这种方法一般称之为滑动窗口(Slide Window)。

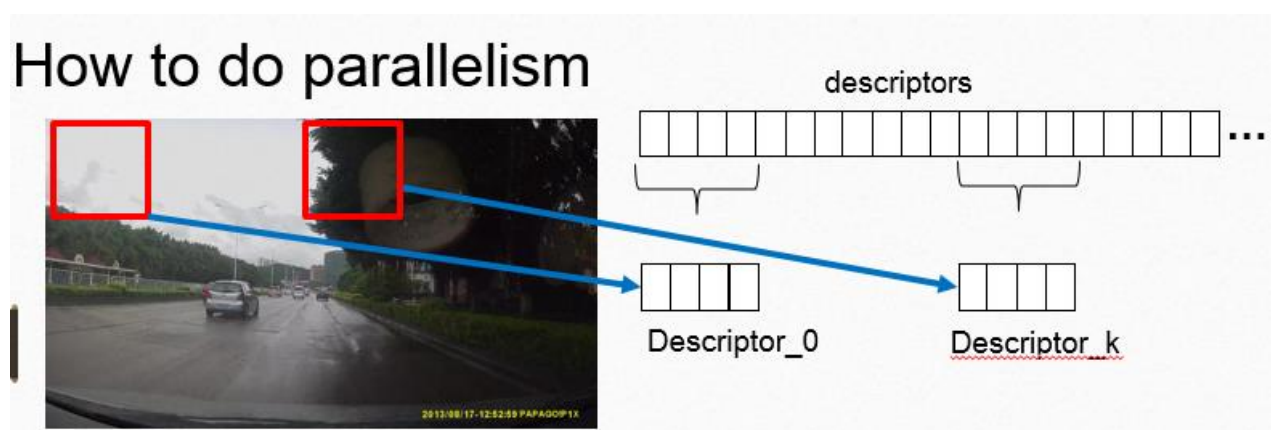


图 2.1 滑动窗口示意图以及并行思路

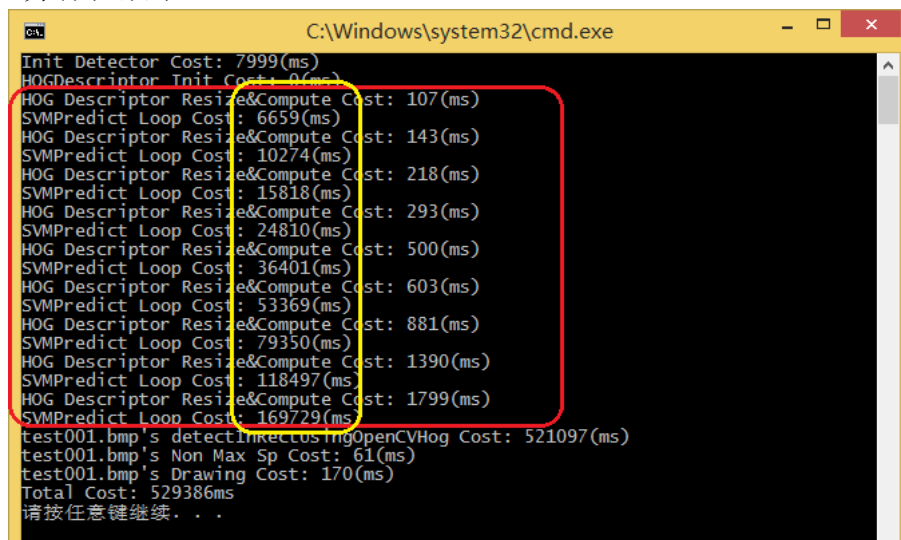
由于滑动窗口枚举位置这个过程是可以并行化的，并且要枚举的位置越多，并行化程度越高，一般来说要枚举的位置是远多于 CPU 核心数量，这时使用 GPU 做运算更加适合。对于枚举的每个位置，我们需要做两件事：提取 HOG 特征，用 SVM 预测类别。OpenCV 在提取 HOG 特征这个步骤用了并行化以外的优化，例如，内存优化，避免重复计算在已经上一个细胞单元计算过的梯度，最终达到很高的效率。而 SVM 预测这一步没有办法做到内存优化，所以需要用到 OpenCL。

三. 设计和实现过程

1. 寻找可并行化的部分。

分析原工程，通过向其添加必要的时间函数获取各关键部分运行时间并打印出来，如下图 3.1.1。从图可以明显看到，对于已经提取出 HOG 特征的图片进行处理，是最耗时的部分；而该部分每次循环中则需花费大部分时间用于做 SVMPredict 处理。因此，将最耗时的 SVMPredict 这一部分移植到内核中，通过 GPU 进行并行化处理，将有效降低整个代码的运

行时间，最大化并行化效果。



```
C:\Windows\system32\cmd.exe
Init Detector Cost: 7999(ms)
HOGDescriptor Init Cost: 94(ms)
HOG Descriptor Resize&Compute Cost: 107(ms)
SVMPredict Loop Cost: 6659(ms)
HOG Descriptor Resize&Compute Cost: 143(ms)
SVMPredict Loop Cost: 10274(ms)
HOG Descriptor Resize&Compute Cost: 218(ms)
SVMPredict Loop Cost: 15818(ms)
HOG Descriptor Resize&Compute Cost: 293(ms)
SVMPredict Loop Cost: 24810(ms)
HOG Descriptor Resize&Compute Cost: 500(ms)
SVMPredict Loop Cost: 36401(ms)
HOG Descriptor Resize&Compute Cost: 603(ms)
SVMPredict Loop Cost: 53369(ms)
HOG Descriptor Resize&Compute Cost: 881(ms)
SVMPredict Loop Cost: 79350(ms)
HOG Descriptor Resize&Compute Cost: 1390(ms)
SVMPredict Loop Cost: 118497(ms)
HOG Descriptor Resize&Compute Cost: 1799(ms)
SVMPredict Loop Cost: 169729(ms)
test001.bmp's detectInRectUsingOpenCVHog Cost: 521097(ms)
test001.bmp's Non Max Sp Cost: 61(ms)
test001.bmp's Drawing Cost: 170(ms)
Total Cost: 529386ms
请按任意键继续. . .
```

图 3.1.1

2. 将关键部分移植进内核。

我们开始考虑将该部分移植进内核，而该部分是通过调用 `SVMDelector` 的一个方法 `SVMPredict` 实现。`SVMDelector` 则是已经封装了的一个 `SVM` 和滑动窗口的一个类。`SVMPredict` 正是通过调用 `SVM` 开源库中的函数实现，所以，我们要将这一部分移植到内核中，避免不了在内核中调用 `SVM` 开源库中的函数。至于原先该部分函数的调用，我们可以将参数和返回值均转化为内存对象，内核通过参数对应的内存对象得到 `SVM` 处理参数，经过处理后将结果写入返回值对应内存对象。最后在 `Host` 程序中从返回值内存对象中得到处理结果，保存到结果记录集中，然后继续下一轮处理。

3. 遇到问题与思考。

按照上述思路移植内核，代码编写完成后编译执行，从内核编译信息中得到错误，提示 `include` 指定头文件不存在。原因是内核中直接 `include` “`svm.h`”，`svm` 开源库，在 `CL` 在线编译的时候并未指定，所以发生了错误。解决这个问题的办法尝试了很久，始终未解决。我们的方案是，通过给 `VS` 配置 `CL` 插件 `Code Builder`，利用这个工具离线编译内核，获得内核二进制代码。`CL` 是允许通过二进制的方式加载内核的，这样就可以在运行时避免在线编译，避免上述的问题。这个方案还在尝试中。

4. 另辟蹊径。

当我们的思路遇到瓶颈时，却意外发现 `google code` 上一个 `libSVM` 的 `OpenCL` 移植版，而我们在之前所做的一切正是在实现一个与此类似的项目，所以我们决定，先停止手头的方案，尝试将该 `libSVM` 移植到我们项目中。具体的移植过程请参考附录工程使用说明。此外，在本项目中，我们将 `OpenCL` 编程中平台设备初始化、上下文/命令队列创建等一系列流程封装到类中，充分提高这部分代码的复用性，更保证了 `Host` 代码的整洁性。具体的关键部分代码如下图，图 3.4.1 显示代码并未进行并行化；图 3.4.2 则是并行化后的结果，其中 `SvmPredict` 封装了并行化的 `SVM` 处理函数 `predict` 函数，`predict` 函数调用 `OpenCLToolsPredict` 类中的 `predict` 方法，该方法如图 3.4.3 所示，涉及内存对象创建、内核创建和参数设置、`NDRange` 设置和内核执行、`Device` 内存拷贝以及最后的对象释放。另外，`OpenCLToolsPredict` 继承自 `OpenCLBase`，该基类封装了 `OpenCL` 的通用方法，包括平台、设备初始化，上下文创建，命令队列创建，内核程序加载、预编译，内核创建，编译等。

```

121 // 这一块进行OpenCL并行化处理
122 for (int i = 0; i <= (int)img_zoom.cols - 64; i += winSize.width)
123 {
124
125 #pragma omp parallel for
126     for (int j = 0; j <= (int)img_zoom.rows - 64; j += winSize.height)
127     // 参数 _global is_predict_probability; descriptor
128     // _local i; j; descriptor
129     {
130         // descriptor临时获取descriptors数据 长度1764
131         vector<float> descriptor; // vector转为数组操作.....
132         descriptor.assign(descriptors.begin() + ((i / winSize.width) + (j / winSize.height) * nWin.width) * 1764,
133             descriptors.begin() + ((i / winSize.width) + (j / winSize.height) * nWin.width + 1) * 1764);
134         //cout << "descriptors.size: " << descriptors.size() << endl;
135         //cout << "descriptor.size: " << descriptor.size() << endl;
136
137         if (this->is_predict_probability)
138         {
139             //cout << "is_predict_probability" << endl; // 总是true
140             double score[10];
141
142             int result = SVMFPredict(descriptor, score);
143
144             if (result == 1) // 输出结果如何并行处理??? 每个item一个存放位置, copy到host后如果! NULL就将该数组元素push_back
145             {
146                 #pragma omp critical
147                 {
148                     outputRects.push_back(Rect(int(i / zoom), int(j / zoom), int(64 / zoom), int(64 / zoom)));
149                     scores.push_back(score[0]);
150                 }
151             }
152         }
153     }
154 }
155 }
156 }

```

图 3.4.1

```

SvmPredict predictor;
// 载入Model
predictor.loadModel();

startExc = clock();
while (times < zoom_times * 3)
{
    resize(img, img_zoom, Size(int(img.cols * zoom), int(img.rows * zoom)));

    vector<float> descriptors;
    Size winSize(16, 16);
    Size nWin((img_zoom.cols - 64) / winSize.width + 1, (img_zoom.rows - 64) / winSize.height + 1);

    hog.compute(img_zoom, descriptors, Size(16, 16));

    float* descriptorsArr = new float[descriptors.size()];
    for (int i = 0; i < descriptors.size(); i++) {
        descriptorsArr[i] = descriptors[i];
    }

    Matrix<float> mat(descriptorsArr, 1764, descriptors.size() / 1764);

    unsigned char *result = predictor.predict(&mat, 1764, descriptors.size() / 1764);

    int rowWidth = ((int)img_zoom.cols - 64) / winSize.width;

    for (int i = 0; i <= (int)img_zoom.rows - 64; i += winSize.height)
    {
        for (int j = 0; j <= (int)img_zoom.cols - 64; j += winSize.width)
        {
            double score[10] = { 0 }; // 不评分
            int index = (i / winSize.height) * rowWidth + (j / winSize.width);
            if ((int)result[index] == 1)
            {
                outputRects.push_back(Rect(int(j / zoom), int(i / zoom), int(64 / zoom), int(64 / zoom)));
                scores.push_back(score[0]);
            }
        }
    }
}

```

图 3.4.2

```

uchar* OpenCLToolsPredict::predict(svm_model* model, const Matrix<float>* parameters) {
    //cout << "Create Buffer for Model and Image Parameters..." << endl;
    createBuffers(parameters, model); // 创建内存对象
    //cout << "Finish Create Buffer for Model and Image Parameters..." << endl;

    setKernelArgs(parameters->getHeight(), parameters->getWidth(), model); // 创建和内核和设置参数

    size_t local_ws = workGroupSize[0];
    int numValues = parameters->getHeight() * parameters->getWidth();
    size_t global_ws = shrRoundUp(local_ws, numValues);
    err = clEnqueueNDRangeKernel(command_queue, kernel[0], 1, NULL, &global_ws, &local_ws, 0, NULL, NULL); // 设置NDRange并执行内核
    err_check(err, "OpenCLTools::predict clEnqueueNDRangeKernel");
    size_t size = parameters->getHeight() * sizeof(cl_uchar);
    uchar* retVec = new uchar(parameters->getHeight() * parameters->getWidth()); // 获取内核结果
    err = clEnqueueReadBuffer(command_queue, clPredictResults, CL_TRUE, 0, size, retVec, 0, NULL, NULL);
    err_check(err, "OpenCLTools::predict clEnqueueReadBuffer");
    clFlush(command_queue); // 释放对象
    clFinish(command_queue);
    return retVec;
}

```

图 3.4.3

5. 运行和比较。

移植成功后，我们通过同样的获取运行时间的方式来获取并行化后的该 SVMPredict 部分代码的运行时间，通过比较其与原始程序运行时间上的差异来分析并行化的效果。具体请看下面的实验结果和数据分析。

四. 实验结果和数据分析

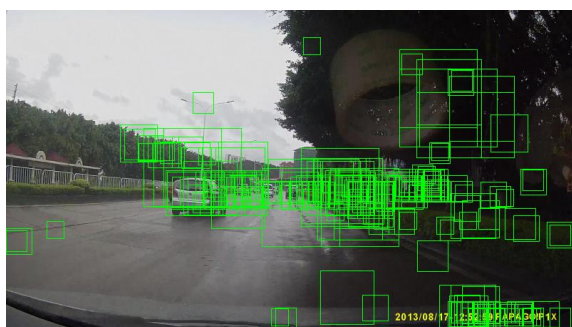


图 4.1.1

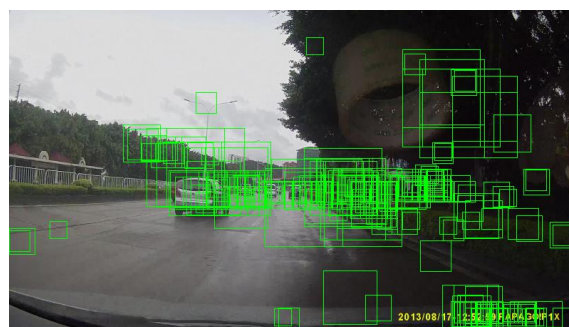


图 4.1.2

1. 图 4.1.1 是未使用 OpenCL 并行化的检测结果，图 4.1.2 则是 OpenCL 并行化后的检测结果。可以清楚看到，二者在可视范围内是基本一样的，证明了我们并行化的正确性。
2. 下图显示了经过并行化和未经并行化的代码在各部分模块执行时间上的差异。

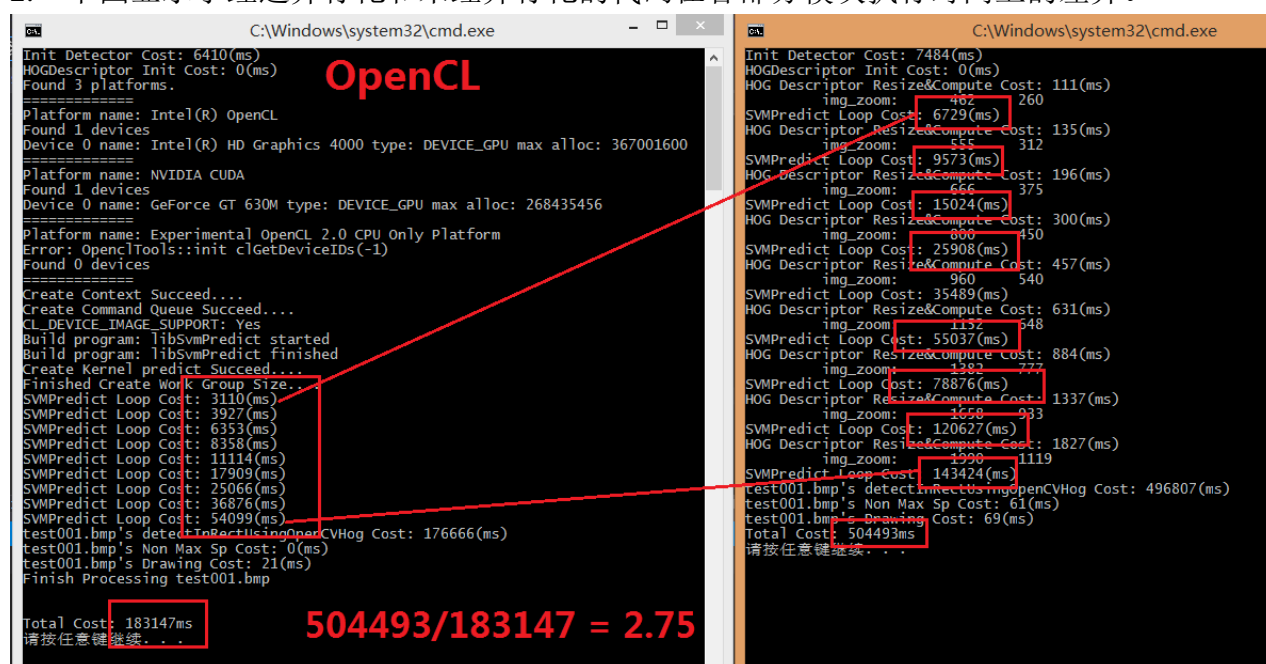


图 4.2

从图 4.2 可以看到，并行化的 Predict 函数在执行速率上较未并行化有显著的提升，最终该部分的耗时，并行化后的执行时间只有原先的 1/2.75，OpenCL 并行化对本项目的优化效果明显。

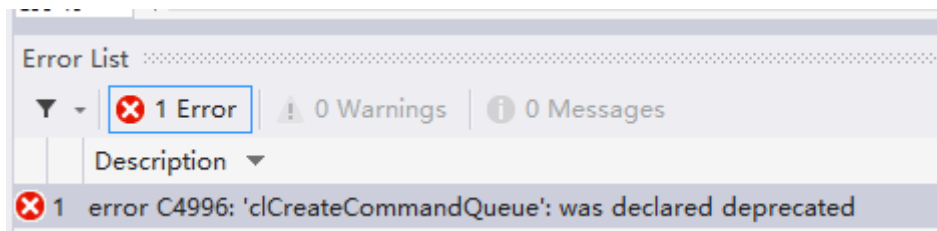
经过 libsvm 分类处理之后，可以对得到的图片做聚类处理得到车辆检测的结果如下：



图 4.3 车辆识别，检测一辆车的示意图

五. 项目总结

通过本次的项目，我们学会了使用编写 OpenCL 程序，包括初始化平台、设备，创建内存 buffer，数据拷贝，创建 kernel，执行 kernel 等。通过 OpenCL 来实现并行计算，从而加速程序。在实验的过程中也遇到了一些问题，通过查找资料 and 搜索最终得以解决。如：



这个是因为该函数已经是旧版本的，所以不支持，需要添加 `#pragma warning(disable : 4996)` 来使用，让编译器不报错。

经过这个学期的学习和项目，总体感觉 OpenCL 是一个非常强大的工具，在具体编写代码的时候，主要是按照固定的步骤来写，对于每个程序来说，初始化等等都是相似的，重点是（1）如何编写 kernel 函数，这个需要根据程序的具体功能以及数据的格式来确定，（2）内存对象的创建和拷贝，（3）kernel 参数的确定和设置。当然，还有一点最重要的就是分析程序如何并行才能得到更好的加速效果，这个需要多练习和体会。最后项目的实验结果表明，经过 openCL 的并行优化之后，车辆识别的程序运行时间大大减少了。

另外，对于 OpenCL 的内存模型还需要更深入的了解，如宿主机内存、设备内存、全局内存、私有内存等，总体感觉 openCL 在这一块的接口有点复杂，不太好使用。如果选择不恰当，对程序的性能会有影响。

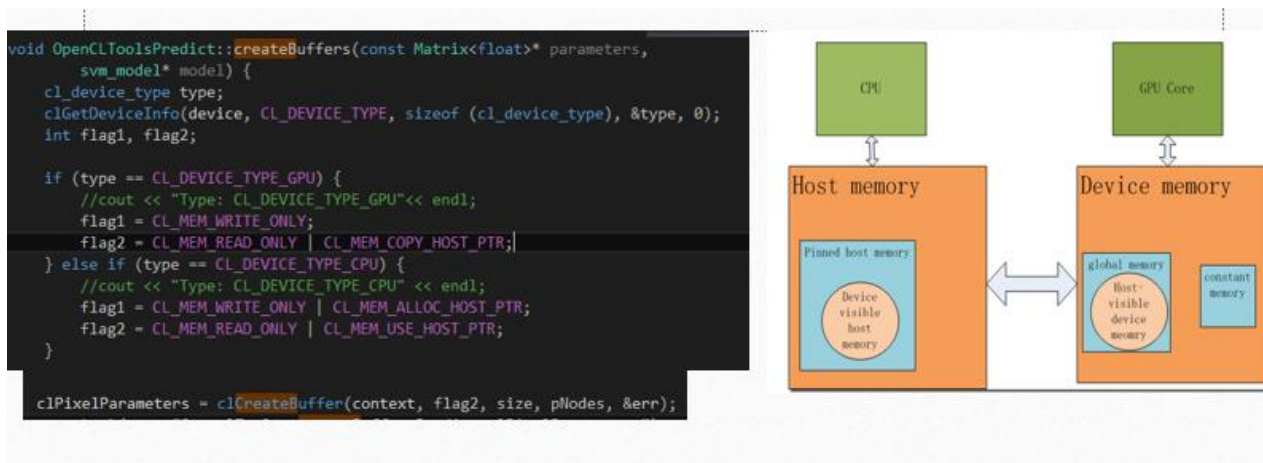


图 5.1 opencil 内存模型

虽然项目最后成功做出，但是过程中仍然有不少的插曲。我们一开始是准备完全从 0 开始自己写，但是在实现的过程中遇到了很多问题。最后移植了网上的部分代码，每一个实现与选择都是团队合作探讨的结果，大家在开发的过程中不断学习、讨论交流、共同优化我们的试验成果。

由于时间的关系，本次项目上还有一些需要完善的地方没有完善好。我们会在不足与后续工作中详细说明，如果有时间有机会一定要继续完成。

整个的过程虽然很多困难，但大家齐心协力做出这个项目，坚持总是值得的。我们经过这次实验，切实学到了很多 OpenCL 的知识，我们学会很多计算机视觉、图像处理，感受到了 OpenCV 和 OpenCL 这种 1+1>2 的强大力量，进一步巩固实际动手能力，培养严谨的实验作风。最后要感谢老师，感谢 TA，给我们的项目提供了宝贵的意见，帮助我们克服困难取得成果。

团队的协作和项目管理：

在刚开始做项目的时候，我们就认识到团队的合作是非常重要的。所以在项目初，我们就进行了团队人员的分工，然后安排了定期的开会讨论和在线讨论等。

我们的代码采用 github 做版本控制，全部上传到 github，方便每个人的查看和修改。

branch: **master**

co-Design-opencil / opencil / +

Update LaneDetectTest3.cpp

zhiqiu

authored 6 minutes ago

latest commit 52760016b0

..

VehicleDetection

Update MemTracker.cpp

7 minutes ago

vehicle_detection

Update rectangles.cpp

8 minutes ago

LaneDetectTest3.cpp

Update LaneDetectTest3.cpp

6 minutes ago

README.md

Update README.md

7 minutes ago

hello.cpp

add

2 months ago

main.cl

add

2 months ago






图 5.2 项目 github 截图

不足与后续工作：

另外，本项目还存在继续完善的地方。原本的项目使用的是带 **probability** 的 **SVM**，而并行化的 **SVM** 并没有对 **probability** 进行过多处理。因为并行化的 **SVM** 是通过参考开源项目获得，如需改进需要在 **OpenCL** 代码中增加 **probability** 部分的计算。

附录：

1. 提交文件详情

 SlideWindow	2015/7/10 1:04
 SlideWindowCL	2015/7/10 1:04
 OpenCL Final presentation G2.ppt	2015/6/24 14:48
 README.pdf	2015/7/10 9:05
 OpenCL Report G2.pdf	2015/7/10 11:33

2. 项目工程使用说明

详见 **README** 文件。