

Bin 的专栏

让更多人了解“机器学习”

个人资料



大饼博士X

访问：254322次

积分：3083

等级：

BLOG > 5

排名：第10040名

原创：59篇 转载：19篇

译文：0篇 评论：67条

about me

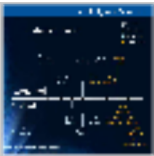
关注机器学习、深度学习算法，高性能硬件；
email：188997452(at)qq.com

文章搜索

weibo

大饼博士

钱柜娱乐开户



机器学习与深度学习笔记
文章：24篇
阅读：140416

文章分类

PRML学习 (6)

机器学习 Machine Learning (40)

信息检索 Information Retrieval (3)

推荐系统 RecSys (0)

深度学习 Deep Learning (24)

论文阅读笔记 (8)

【评论送书】我的世界、架构师、OpenStack Python 创意编程活动 CSDN日报20170510 ——《如何撰写一篇受人欢迎的博客》

OpenCL学习笔记（三）：OpenCL安装，编程简介与helloworld

标签：[技术](#) [openc1](#)

2015-06-14 23:37 3406人阅读 评论(1) 收藏 举报

分类：
[异构加速/高性能计算 \(10\)](#)

版权声明：本文为博主原创文章，欢迎转载分享，请注明本文出自Bin的专栏：/xbinworld

目录(?) [+]

欢迎转载，转载请注明：本文出自Bin的专栏blog.csdn.net/xbinworld。 技术交流QQ群：433250724，欢迎对算法、技术、应用感兴趣的同学加入。

OpenCL安装

安装我不打算花篇幅写，原因是OpenCL实在是可以太多的平台+环境下实现了，包括GPU和FPGA，以及不同的器件支持，在这里我主要把网上可以找到比较不错的经验贴列一下，方便大家，我主要关注了FPGA的，其他GPU的大家网上搜搜吧：

altera openc1 sdk下载：
<https://www.altera.com.cn/products/design-software/embedded-software-developers/openc1/overview.html>

alter的安装指南，《Altera SDK for OpenCL Getting Started Guide》

理论上看上面两个就够了，你需要做的事情包括：
下载openc1 SDK，或者quatuars II软件(含SDK)，下载相应开发板的支持（altera上面有一些，但是其他的可能需要你从相应的供应商那边找了）；还需要openc1的license，不然是不能编译的。

中文的一些经验贴可以看：
《Altera OpenCL入门(beta版)》http://wenku.baidu.com/link?url=bkIyo01jXeWfdGsrA_M0J1zomx6f0lYk0NPf-9-MNaC00kWRmukDwY5yFz0I3Wrctqi5qD3jC8BhQQzjoqw1HXpUgIM68_blz5Cr3vxpaZC

【Altera SoC体验之旅】+ 正式开启OpenCL模式
<http://home.eeworld.com.cn/my/space-uid-169743-blogid-247647.html>

OpenCL编程简介

下面的图简单说明了OpenCL的编程框架，图是用的GPU，其他类似；

随笔 (2)
Matlab (2)
前沿动态 (2)
知识收录、转载区 (15)
异构加速/高性能计算 (11)
三十分种理解系列 (5)
算法 (1)

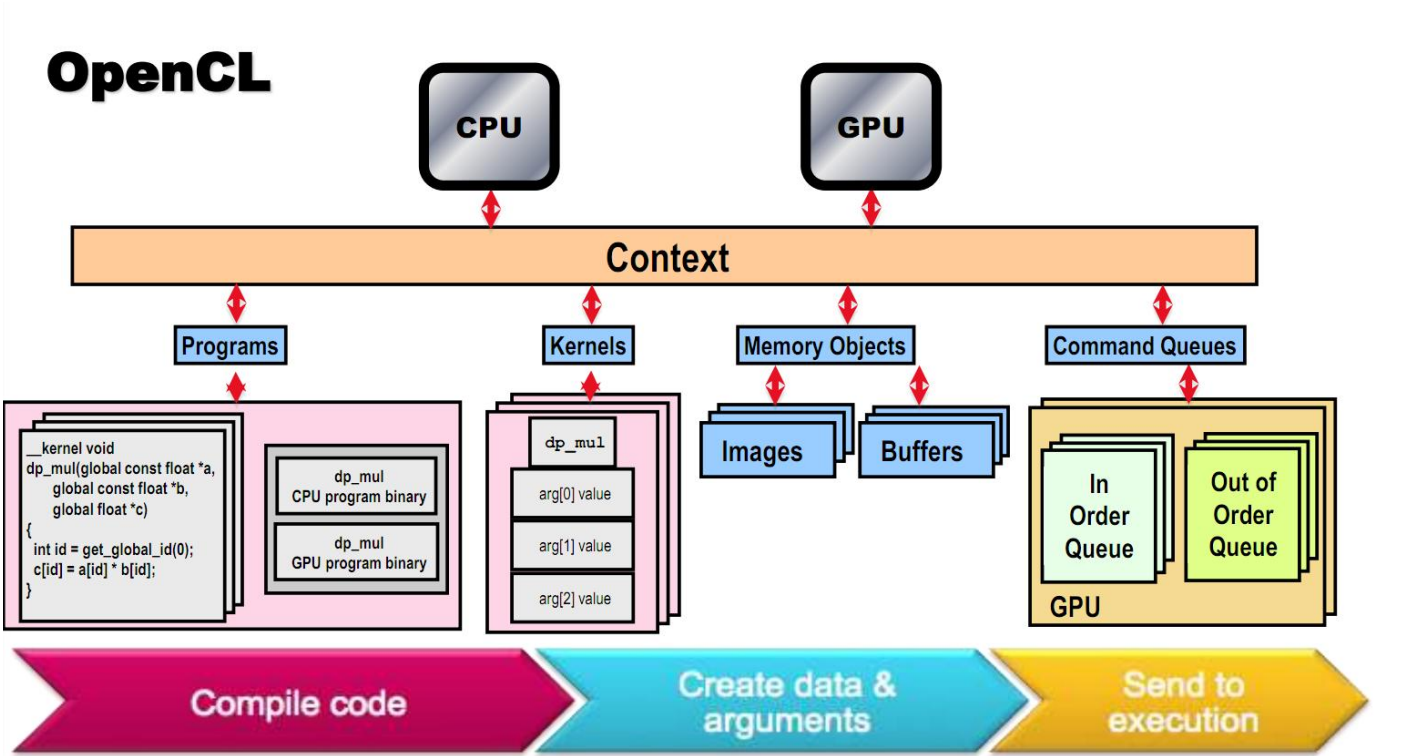
文章存档
2017年04月 (2)
2017年03月 (6)
2017年02月 (7)
2017年01月 (1)
2016年09月 (2)
展开

阅读排行
深度学习方法（五）：卷 (25316)
机器学习方法：回归（二 (13505)
机器学习方法：回归（一 (9589)
matlab绘图的坐标轴数字 (9502)
三十分种理解博弈论“纳什 (9271)
深度学习方法（七）：最 (9106)
机器学习方法：回归（三 (8935)
720P、1080P、4K是什么 (8511)
深度学习方法：受限玻尔 (8189)
自组织神经网络介绍：自 (5551)

评论排行
深度学习方法（五）：卷 (11)
深度学习方法（七）：最 (5)
文章索引：“机器学习方法 (4)
《微微一笑很倾城》中肖 (3)
三十分种理解博弈论“纳什 (3)
机器学习方法(四)：决策树 (3)
深度学习方法：受限玻尔 (3)
深度学习方法：受限玻尔 (3)
计算机方向的一些顶级会议 (3)
机器学习方法：回归（二 (3)

推荐文章
* 程序员4月书讯：Angular来了！
* 程序员要拥抱变化，聊聊钱柜娱乐开户即将支持的Java 8
* 彻底弄懂prepack与webpack的关系
* 用 TensorFlow 做个聊天机器人
* 分布式机器学习的集群方案介绍之HPC实现
* 钱柜娱乐开户 音频系统：从 AudioTrack 到 AudioFlinger

最新评论
自组织神经网络介绍：自组织特 (weixin_38417475: 怎么看评论
深度学习方法（十）：卷积神经 (c337134154: 看论文没看懂，这里讲的很清楚，顶一个
深度学习方法（五）：卷积神经 (少年征途: 您好，我想问一下在



从图中可以看出（参考《OpenCL 编程入门》）：

- 异构计算设备，可以是CPU或GPU。现在也有支持OpenCL的FPGA设备和至强融核协处理设备（MIC）。
- OpenCL的API通过Context（环境上下文）联系在一起。
- 运行设备端的程序，经过了编译->设置参数->运行等步骤。

名词的概念：

Platform（平台）：主机加上OpenCL框架管理下的若干设备构成了这个平台，通过这个平台，应用程序可以与设备共享资源并在设备上执行kernel。实际使用中基本上一个厂商对应一个Platform，比如Intel, AMD都是这样。

Device（设备）：官方的解释是计算单元（Compute Units）的集合。举例来说，GPU是典型的device。Intel和AMD的多核CPU也提供OpenCL接口，所以也可以作为Device。

Context（上下文）：OpenCL的Platform上共享和使用资源的环境，包括kernel、device、memory objects、command queue等。使用中一般一个Platform对应一个Context。

Program：OpenCL程序，由kernel函数、其他函数和声明等组成。

Kernel（核函数）：可以从主机端调用，运行在设备端的函数。

Memory Object（内存对象）：在主机和设备之间传递数据的对象，一般映射到OpenCL程序中的global memory。有两种具体的类型：Buffer Object（缓存对象）和Image Object（图像对象）。

Command Queue（指令队列）：在指定设备上管理多个指令（Command）。队列里指令执行可以顺序也可以乱序。一个设备可以对应多个指令队列。

NDRange：主机端运行设备端kernel函数的主要接口。实际上还有其他的，NDRange是非常常见的，用于分组运算，以后具体用到的时候就知道区别了。

Host端来看，OpenCL的组要执行流程是这样的：

关闭

vgg-f中，输入数据是224，也没有填充边，这样计算下来的话就不能整除，模型自己...

深度学习方法（七）：最新Sque
liuqingjia: 为什么 S11 < (E11+E33) 就能减少input feature map数量???

机器学习方法：回归（二）：稀
kmust路人: 感谢博主的分享

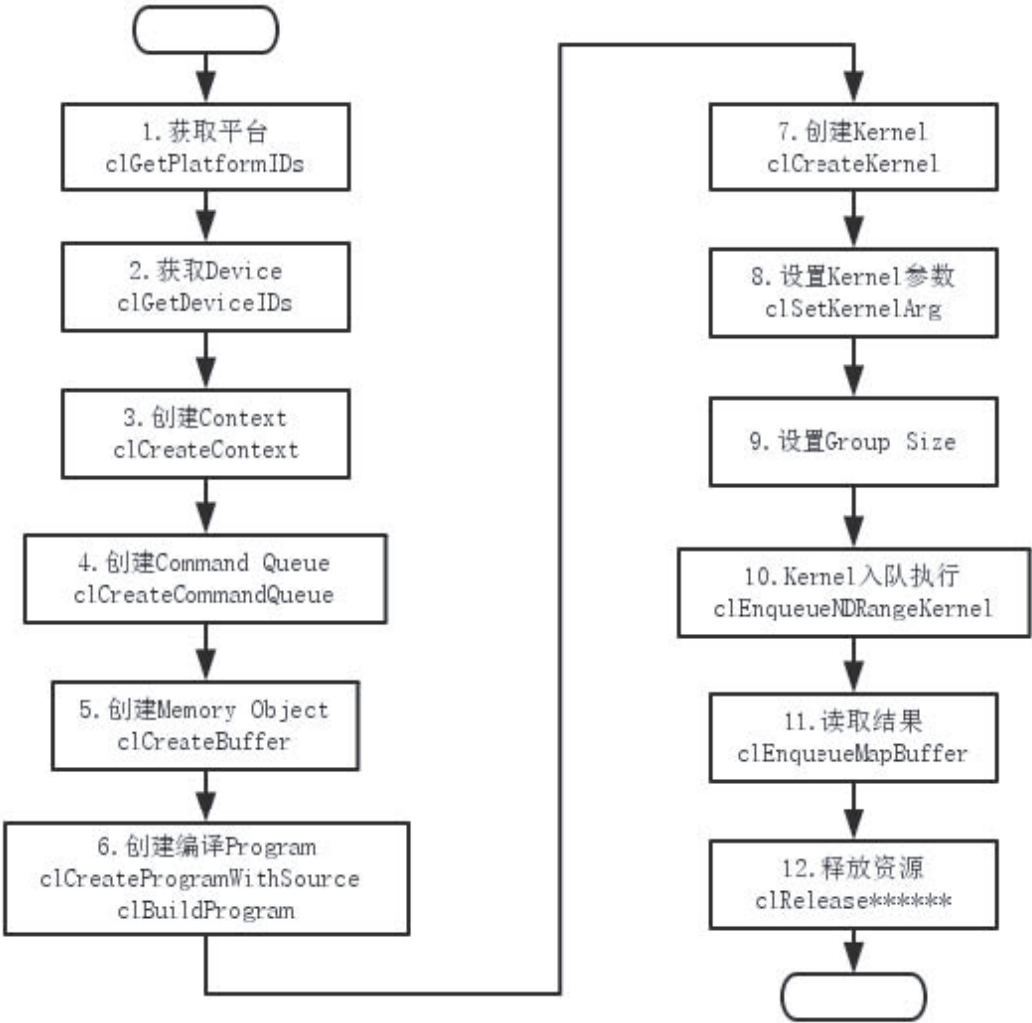
机器学习方法：回归（三）：最
xixihahaxiaozhe: 文中用到相关系数，我总感觉怪怪的

深度学习方法（七）：最新Sque
LZXWAZ: 博主你好，请问一下，我怎么把其他网络模型用Squeezenet进行压缩呢？比如想用Squeezene...

[重磅]Deep Forest，非神经网络
Shiny__duoduo: 你好楼主，可不可以发一下这篇文献给我看看？邮箱843657477@qq.com,十分感谢~

深度学习方法（五）：卷积神经
lg12170226: 学习 谢谢

深度学习方法（七）：最新Sque
Quiet_cb: @qq_35800608:使用classification，配置SqueezeNet的deploy、...



其实基本上大部分简单的程序HOST部分都是差不多的，不用改很多，具体下面看一个例子就知道了。

第一个程序

这里贴一个altera官方的vector add的实例code，基本就是helloworld级别了，不过它的host写的很通用（考虑到对多个device统一编程），可以过一遍看看是不是和上面的图对的上。其实看过这个基本其他的也就差不多了。

Host部分：（Kernel在最后）

```
1 // Copyright (C) 2013-2014 Altera Corporation, San Jose, California, USA. All rights reserved.
2 // Permission is hereby granted, free of charge, to any person obtaining a copy of this
3 // software and associated documentation files (the "Software"), to deal in the Software
4 // without restriction, including without limitation the rights to use, copy, modify, merge,
5 // publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to
6 // whom the Software is furnished to do so, subject to the following conditions:
7 // The above copyright notice and this permission notice shall be included in all copies or
8 // substantial portions of the Software.
9 //
10 // THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
11 // EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES
12 // OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
13 // NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
14 // HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY,
15 // WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
16 // FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
17 // OTHER DEALINGS IN THE SOFTWARE.
18 //
19 // This agreement shall be governed in all respects by the la
20 // by the laws of the United States of America.
21
22 //////////////////////////////////////
23 // This host program executes a vector addition kernel to perform:
24 // C = A + B
25 // where A, B and C are vectors with N elements.
26 //
27 // This host program supports partitioning the problem across multiple OpenCL
28 // devices if available. If there are M available devices, the problem is
29 // divided so that each device operates on N/M points. The host program
30 // assumes that all devices are of the same type (that is, the same binary can
31 // be used), but the code can be generalized to support different device types
```

关闭

```
32 // easily.
33 //
34 // Verification is performed against the same computation on the host CPU.
35 ///////////////////////////////////////////////////////////////////
36
37 #include <stdio.h>
38 #include <stdlib.h>
39 #include <math.h>
40 #include "CL/opencl.h"
41 #include "AOCL_Utils.h"
42
43 using namespace aocl_utils;
44
45 // OpenCL runtime configuration
46 cl_platform_id platform = NULL;
47 unsigned num_devices = 0;
48 scoped_array<cl_device_id> device; // num_devices elements
49 cl_context context = NULL;
50 scoped_array<cl_command_queue> queue; // num_devices elements
51 cl_program program = NULL;
52 scoped_array<cl_kernel> kernel; // num_devices elements
53 scoped_array<cl_mem> input_a_buf; // num_devices elements
54 scoped_array<cl_mem> input_b_buf; // num_devices elements
55 scoped_array<cl_mem> output_buf; // num_devices elements
56
57 // Problem data.
58 const unsigned N = 1000000; // problem size
59 scoped_array<scoped_aligned_ptr<float> > input_a, input_b; // num_devices elements
60 scoped_array<scoped_aligned_ptr<float> > output; // num_devices elements
61 scoped_array<scoped_array<float> > ref_output; // num_devices elements
62 scoped_array<unsigned> n_per_device; // num_devices elements
63
64 // Function prototypes
65 float rand_float();
66 bool init_openccl();
67 void init_problem();
68 void run();
69 void cleanup();
70
71 // Entry point.
72 int main() {
73     // Initialize OpenCL.
74     if(!init_openccl()) {
75         return -1;
76     }
77
78     // Initialize the problem data.
79     // Requires the number of devices to be known.
80     init_problem();
81
82     // Run the kernel.
83     run();
84
85     // Free the resources allocated
86     cleanup();
87
88     return 0;
89 }
90
91 /////// HELPER FUNCTIONS /////
92
93 // Randomly generate a floating-point number between -10 and 10.
94 float rand_float() {
95     return float(rand()) / float(RAND_MAX) * 20.0f - 10.0f;
96 }
97
98 // Initializes the OpenCL objects.
99 bool init_openccl() {
100     cl_int status;
101
102     printf("Initializing OpenCL\n");
```

关闭


```
103
104 if(!setCwdToExeDir()) {
105     return false;
106 }
107
108 // Get the OpenCL platform.
109 platform = findPlatform("Altera");
110 if(platform == NULL) {
111     printf("ERROR: Unable to find Altera OpenCL platform.\n");
112     return false;
113 }
114
115 // Query the available OpenCL device.
116 device.reset(getDevices(platform, CL_DEVICE_TYPE_ALL, &num_devices));
117 printf("Platform: %s\n", getPlatformName(platform).c_str());
118 printf("Using %d device(s)\n", num_devices);
119 for(unsigned i = 0; i < num_devices; ++i) {
120     printf(" %s\n", getDeviceName(device[i]).c_str());
121 }
122
123 // Create the context.
124 context = clCreateContext(NULL, num_devices, device, NULL, NULL, &status);
125 checkError(status, "Failed to create context");
126
127 // Create the program for all device. Use the first device as the
128 // representative device (assuming all device are of the same type).
129 std::string binary_file = getBoardBinaryFile("vectorAdd", device[0]);
130 printf("Using AOCX: %s\n", binary_file.c_str());
131 program = createProgramFromBinary(context, binary_file.c_str(), device, num_devices);
132
133 // Build the program that was just created.
134 status = clBuildProgram(program, 0, NULL, "", NULL, NULL);
135 checkError(status, "Failed to build program");
136
137 // Create per-device objects.
138 queue.reset(num_devices);
139 kernel.reset(num_devices);
140 n_per_device.reset(num_devices);
141 input_a_buf.reset(num_devices);
142 input_b_buf.reset(num_devices);
143 output_buf.reset(num_devices);
144
145 for(unsigned i = 0; i < num_devices; ++i) {
146     // Command queue.
147     queue[i] = clCreateCommandQueue(context, device[i], CL_QUEUE_PROFILING_ENABLE, &status);
148     checkError(status, "Failed to create command queue");
149
150     // Kernel.
151     const char *kernel_name = "vectorAdd";
152     kernel[i] = clCreateKernel(program, kernel_name, &status);
153     checkError(status, "Failed to create kernel");
154
155     // Determine the number of elements processed by this device.
156     n_per_device[i] = N / num_devices; // number of elements handled by this device
157
158     // Spread out the remainder of the elements over the first
159     // N % num_devices.
160     if(i < (N % num_devices)) {
161         n_per_device[i]++;
162     }
163
164     // Input buffers.
165     input_a_buf[i] = clCreateBuffer(context, CL_MEM_READ_ONLY,
166         n_per_device[i] * sizeof(float), NULL, &status);
167     checkError(status, "Failed to create buffer for input A");
168
169     input_b_buf[i] = clCreateBuffer(context, CL_MEM_READ_ONLY,
170         n_per_device[i] * sizeof(float), NULL, &status);
171     checkError(status, "Failed to create buffer for input B");
172
173     // Output buffer.
```

关闭

```
174     output_buf[i] = clCreateBuffer(context, CL_MEM_WRITE_ONLY,
175         n_per_device[i] * sizeof(float), NULL, &status);
176     checkError(status, "Failed to create buffer for output");
177 }
178
179 return true;
180 }
181
182 // Initialize the data for the problem. Requires num_devices to be known.
183 void init_problem() {
184     if(num_devices == 0) {
185         checkError(-1, "No devices");
186     }
187
188     input_a.reset(num_devices);
189     input_b.reset(num_devices);
190     output.reset(num_devices);
191     ref_output.reset(num_devices);
192
193     // Generate input vectors A and B and the reference output consisting
194     // of a total of N elements.
195     // We create separate arrays for each device so that each device has an
196     // aligned buffer.
197     for(unsigned i = 0; i < num_devices; ++i) {
198         input_a[i].reset(n_per_device[i]);
199         input_b[i].reset(n_per_device[i]);
200         output[i].reset(n_per_device[i]);
201         ref_output[i].reset(n_per_device[i]);
202
203         for(unsigned j = 0; j < n_per_device[i]; ++j) {
204             input_a[i][j] = rand_float();
205             input_b[i][j] = rand_float();
206             ref_output[i][j] = input_a[i][j] + input_b[i][j];
207         }
208     }
209 }
210
211 void run() {
212     cl_int status;
213
214     const double start_time = getCurrentTimestamp();
215
216     // Launch the problem for each device.
217     scoped_array<cl_event> kernel_event(num_devices);
218     scoped_array<cl_event> finish_event(num_devices);
219
220     for(unsigned i = 0; i < num_devices; ++i) {
221
222         // Transfer inputs to each device. Each of the host buffers supplied to
223         // clEnqueueWriteBuffer here is already aligned to ensure that DMA is used
224         // for the host-to-device transfer.
225         cl_event write_event[2];
226         status = clEnqueueWriteBuffer(queue[i], input_a_buf[i], CL_FALSE,
227             0, n_per_device[i] * sizeof(float), input_a[i], 0, NULL, &write_event[0]);
228         checkError(status, "Failed to transfer input A");
229
230         status = clEnqueueWriteBuffer(queue[i], input_b_buf[i], CL_FALSE,
231             0, n_per_device[i] * sizeof(float), input_b[i], 0, NULL, &write_event[1]);
232         checkError(status, "Failed to transfer input B");
233
234         // Set kernel arguments.
235         unsigned argi = 0;
236
237         status = clSetKernelArg(kernel[i], argi++, sizeof(cl_mem), &input_a_buf[i]);
238         checkError(status, "Failed to set argument %d", argi - 1);
239
240         status = clSetKernelArg(kernel[i], argi++, sizeof(cl_mem), &input_b_buf[i]);
241         checkError(status, "Failed to set argument %d", argi - 1);
242
243         status = clSetKernelArg(kernel[i], argi++, sizeof(cl_mem), &output_buf[i]);
244         checkError(status, "Failed to set argument %d", argi - 1);
```

关闭

```
245
246 // Enqueue kernel.
247 // Use a global work size corresponding to the number of elements to add
248 // for this device.
249 //
250 // We don't specify a local work size and let the runtime choose
251 // (it'll choose to use one work-group with the same size as the global
252 // work-size).
253 //
254 // Events are used to ensure that the kernel is not launched until
255 // the writes to the input buffers have completed.
256 const size_t global_work_size = n_per_device[i];
257 printf("Launching for device %d (%d elements)\n", i, global_work_size);
258
259 status = clEnqueueNDRangeKernel(queue[i], kernel[i], 1, NULL,
260     &global_work_size, NULL, 2, write_event, &kernel_event[i]);
261 checkError(status, "Failed to launch kernel");
262
263 // Read the result. This the final operation.
264 status = clEnqueueReadBuffer(queue[i], output_buf[i], CL_FALSE,
265     0, n_per_device[i] * sizeof(float), output[i], 1, &kernel_event[i], &finish_event[i]);
266
267 // Release local events.
268 clReleaseEvent(write_event[0]);
269 clReleaseEvent(write_event[1]);
270 }
271
272 // Wait for all devices to finish.
273 clWaitForEvents(num_devices, finish_event);
274
275 const double end_time = getCurrentTimestamp();
276
277 // Wall-clock time taken.
278 printf("\nTime: %0.3f ms\n", (end_time - start_time) * 1e3);
279
280 // Get kernel times using the OpenCL event profiling API.
281 for(unsigned i = 0; i < num_devices; ++i) {
282     cl_ulong time_ns = getStartEndTime(kernel_event[i]);
283     printf("Kernel time (device %d): %0.3f ms\n", i, double(time_ns) * 1e-6);
284 }
285
286 // Release all events.
287 for(unsigned i = 0; i < num_devices; ++i) {
288     clReleaseEvent(kernel_event[i]);
289     clReleaseEvent(finish_event[i]);
290 }
291
292 // Verify results.
293 bool pass = true;
294 for(unsigned i = 0; i < num_devices && pass; ++i) {
295     for(unsigned j = 0; j < n_per_device[i] && pass; ++j) {
296         if(fabsf(output[i][j] - ref_output[i][j]) > 1.0e-5f) {
297             printf("Failed verification @ device %d, index %d\nOutput: %f\nReference: %f\n",
298                 i, j, output[i][j], ref_output[i][j]);
299             pass = false;
300         }
301     }
302 }
303
304 printf("\nVerification: %s\n", pass ? "PASS" : "FAIL");
305 }
306
307 // Free the resources allocated during initialization
308 void cleanup() {
309     for(unsigned i = 0; i < num_devices; ++i) {
310         if(kernel && kernel[i]) {
311             clReleaseKernel(kernel[i]);
312         }
313         if(queue && queue[i]) {
314             clReleaseCommandQueue(queue[i]);
315         }
316     }
```

关闭

```
316     if(input_a_buf && input_a_buf[i]) {
317         clReleaseMemObject(input_a_buf[i]);
318     }
319     if(input_b_buf && input_b_buf[i]) {
320         clReleaseMemObject(input_b_buf[i]);
321     }
322     if(output_buf && output_buf[i]) {
323         clReleaseMemObject(output_buf[i]);
324     }
325 }
326
327 if(program) {
328     clReleaseProgram(program);
329 }
330 if(context) {
331     clReleaseContext(context);
332 }
333 }
334
```

Kernel部分：

```
1 // ACL kernel for adding two input vectors
2 __kernel void vectorAdd(__global const float *x,
3                         __global const float *y,
4                         __global float *restrict z)
5 {
6     // get index of the work item
7     int index = get_global_id(0);
8
9     // add the vector elements
10    z[index] = x[index] + y[index];
11 }
```

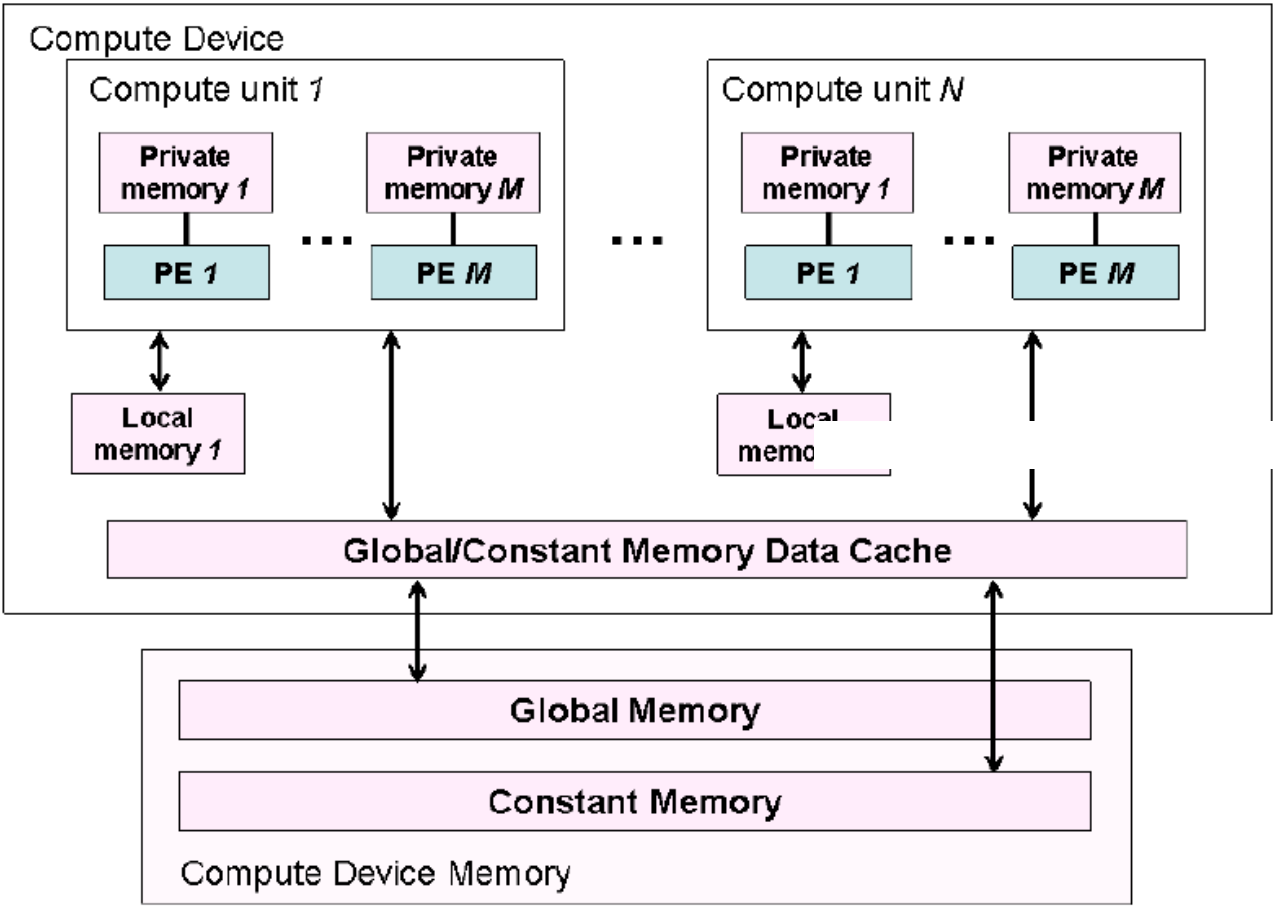
kernel部分代码就这几行，__global是一个限定符，表示用外部存储（比如DDR）来存储，其他语法和标准C语言是一样的，就不多说了。

代码中最重要的就是get_global_id，这个是在多work-item工作模式下的常用手段，通过id确定work-item然后进行操作，所有的item都是一样的，因此就add的函数里面就没有习惯的for()的写法了。可以对kernel的设置进行定制，包括compute unit，SIMD模式等，这样来控制程序的并行性，更大的并行往往性能高，但是更耗资源。

具体的Kernel函数的内容可以参考OpenCL的《The OpenCL Specification 1.0》以及altera的openc1编程指南，后面的笔记我会具体写一下。

内存模型

最后写一下Openc1的内存模型，看下面的示意图：



关闭

用核函数中的内存变量来简单地解释：用clCreateBuffer 创建、用clSetKernelArg 传递的数据在global memory 和constant memory中；核函数中的寄存器变量在private memory 中；核函数的内部变量、缓存等，在local memory 中。图例中可以看到Device 并不直接访问global memory，而是通过Cache 来访问。可以想象当同时运行的work-item，使用的内存都在同一块cache 中，则内存吞吐的效率最高。对应到work group 中，就是在程序设计上尽量使同一个work group 中的work item 操作连续的内存，以提高访存效率。

本篇就到这里。

顶

0

踩

0

上一篇

OpenCL学习笔记（二）：并行编程概念理解

下一篇

深度学习方法：受限玻尔兹曼机RBM（三）模型求解，Gibbs sampling

参考知识库



.NET知识库
3863 关注 | 839 收录



C语言知识库
8944 关注 | 3464 收录



算法与数据结构知识库
16127 关注 | 2320 收录

猜你在找

- C#.NET_面向对象编程技术
- 3D游戏引擎之GPU渲染（DX篇）
- 能听懂Java面向对象编程
- 【CSDN技术主题月】探寻跨平台开发最佳实践
- 视频直播平台架构与核心技术实践

查看评论

您还没有登录,请[\[登录\]](#)或[\[注册\]](#)

* 以上用户言论只代表其个人钱柜娱乐开户，不代表CSDN网站的钱柜娱乐开户或立场

核心技术类目

- 全部主题
- Hadoop
- AWS
- 移动游戏
- Java
- 钱柜娱乐开户
- iOS
- Swift
- 智能硬件
- Docker
- OpenStack
- VPN
- Spark
- ERP
- IE10
- Eclipse
- CRM
- JavaSc
-
- WAP
- jQuery
- BI
- HTML5
- Spring
- Apache
- .NET
- API
- HTML
- SDK
- IIS
- Fedora
- XML
- LBS
- Unity
- Splashtop
- UML
- components
- Windows Mobile
- Rails
- QEMU
- KDE
- Cassandra
- CloudStack
- FTC
- coremail
- OPhone
- CouchBase
- 云计算
- iOS6
- Rackspace
- Web App
- SpringSide
- Maemo
- Compuware
- 大数据
- apttech
- Perl
- Tornado
- Ruby
- Hibernate
- ThinkPHP
- HBase
- Pure
- Solr
- Angular
- Cloud Foundry
- Redis
- Scala
- Django
- Bootstrap

公司简介 | 招贤纳士 | 广告服务 | 联系方式 | 版权声明 | 法律顾问 | 问题报告 | 合作伙伴 | 论坛反馈

网站客服

杂志客服

微博客服

webmaster@csdn.net

400-600-2320

| 北京创新乐知信息技术有限公司 版权所有

| 江苏知之为计算机有限公司 |

京 ICP 证 09002463 号

| Copyright © 1999-2017, CSDN.NET, All Rights Reserved



关闭

关闭