

pengdada

随笔 - 23 文章 - 1 评论 - 0

[博客园](#) [首页](#) [新随笔](#) [联系](#) [订阅](#) [管理](#)

公告

昵称：pengdada

园龄：1年4个月

粉丝：0

关注：0

[+加关注](#)

<	2017年6月						>
日	一	二	三	四	五	六	
28	29	30	31	1	2	3	
4	5	6	7	8	9	10	
11	12	13	14	15	16	17	
18	19	20	21	22	23	24	
25	26	27	28	29	30	1	
2	3	4	5	6	7	8	

搜索

找找看

OpenCL™ 2.0 – Pipes

copy from <http://developer.amd.com/community/blog/2014/10/31/opencl-2-0-pipes/>

OpenCL™ 2.0 – Pipes

In the [previous post](#), we saw one of the important features of OpenCL™ 2.0, Shared Virtual Memory (SVM). In this blog, we will see another feature of OpenCL 2.0 called “pipes”.

To get the most from our discussions, we recommend the same approach as in the previous post:

Review the code snippets in each blog post along with the explanatory text.

Download the [AMD OpenCL 2.0 Driver located here](#). This page also has a complete list of supported platforms.

Download the sample code from our [OpenCL 2.0 Samples page here](#).

Write your own OpenCL 2.0 samples and share your results with the [OpenCL community](#).

谷歌搜索

常用链接

我的随笔
我的评论
我的参与
最新评论
我的标签

随笔分类

CT(computer tomography)(2)
Embedded System(2)
FPGA (Verilog)(14)
GPGPU (OpenCL)(1)
GPGPU(CUDA)(3)
ImageProcess(1)
Machine Learning
OpenCL(SDAccel) (1)
OpenMP & SSE

随笔档案

2016年6月 (1)
2016年5月 (1)
2016年3月 (14)
2016年2月 (7)

Hardware

FPGA开发圈
VGA
xilinx_eetrend
zedboard

阅读排行榜

The code will run on a variety of AMD platforms, such as the Radeon HD8000 series. The driver page has a complete list of supported product families.

Overview

OpenCL 2.0 introduces a new mechanism for passing data between kernels, called “pipes.” A pipe is essentially a structured buffer containing some space for a set of “packets”—kernel-specified type objects. As the name suggests, these packets of data are ordered in the pipe. There is a write end of the pipe into which the data is written and a read end of the pipe from which the data is read. A pipe is essentially an addition to buffer objects, such as buffers and images. Pipes can only be accessed using the built-in functions provided by the kernel and cannot be accessed from the host.

Special built-in functions, `read_pipe` and `write_pipe`, provide access to pipes from the kernel. A given kernel can either read from or write to a pipe, but not both. Pipes are only coherent at the standard synchronization points; the result of concurrent accesses to the same pipe by multiple kernels (even hardware permitting) is undefined. The host side cannot access a pipe.

It is easy to create pipes. On the host, invoke `clCreatePipe`, and you are done.

You can use the pipes for a variety of functions. You can pass the pipes between kernels. Even better, combine pipes with the device-size enqueue feature in OpenCL 2.0 to dynamically construct computational data flow graphs.

Pipes come in two types: a read pipe, from which a number of packets can be read, and a write pipe, to which a number of packets can be written.

Note: You cannot write to a pipe specified as read-only, nor can you read from a pipe specified as write-only. You cannot both read from and write to a pipe at the same time.

Functions for Accessing Pipes

OpenCL 2.0 adds a new host API function to create a pipe.

```
cl_mem clCreatePipe ( cl_context context, cl_mem_flags flags,  
                    cl_uint pipe_packet_size, cl_uint pipe_max_packets,
```

1. win7 配置微软的深度学习caffe (2100)
2. FPGA中竞争冒险问题的研究(436)
3. FPGA 竞争与冒险(298)
4. Verilog数组表示及初始化(259)
5. 锁存器 Latch v.s. 触发器 Flip-Flop(108)

```
const cl_pipe_properties * properties,  
cl_int *errcode_ret)
```

The memory allocated in this function can pass to kernels as read-only or write-only pipes. The pipe objects can only be passed as kernel arguments or kernel functions and cannot be declared inside a kernel or as program-scoped objects.

Also, the OpenCL 2.0 spec adds a set of built-in functions for operating on the pipes. The important ones are:

- `read_pipe` (pipe p, gentype *ptr): for reading packet from pipe p into ptr.
- `write_pipe` (pipe p, gentype *ptr): for writing packet pointed to by ptr to pipe p.

To ensure you have enough space in the pipe structure for reading and writing (before you actually do it), you can use built-in functions to “reserve” enough space. For example, you could reserve room by calling `reserve_read_pipe` or `reserve_write_pipe`. These functions return a reservation ID, which can be used when the actual operations are performed. Similarly, the standard has built-in functions for workgroup level reservations, such as `work_group_reserve_read_pipe` and `work_group_reserve_write_pipe` and for the workgroup order (in the program). These workgroup built-in functions operate at the workgroup level. Ordering across workgroups is undefined. Calls to `commit_read_pipe` and `commit_write_pipe`, as the names suggest, commit the actual operations (read/write).

Using Pipes—A Simple Example

Let’s look at a typical usage of pipes in the example code (SimplePipe). The code contains two kernels: `producer_kernel`, which writes to the pipe, and `consumer_kernel`, which reads from the same pipe. In the example, the producer writes a sequence of random numbers; the consumer reads them and creates a histogram.

The host creates the pipe, which both kernels will use, as follows:

```
rngPipe = clCreatePipe(context,  
                        CL_MEM_READ_WRITE,  
                        szPipePkt,  
                        szPipe,
```

```
NULL,  
&status);
```

This code makes a pipe that the program kernels can access (read/write). The host creates two kernels, `producer_kernel` and `consumer_kernel`. The producer kernel first reserves enough space for the write pipe:

```
//reserve space in pipe for writing random numbers.  
reserve_id_t rid = work_group_reserve_write_pipe(rng_pipe, szgr);
```

Next, the kernel writes and commits to the pipe by invoking the following functions:

```
write_pipe(rng_pipe,rid,lid, &gfrn);  
work_group_commit_write_pipe(rng_pipe, rid);
```

Similarly, the consumer kernel reads from the pipe:

```
//reserve pipe for reading  
reserve_id_t rid = work_group_reserve_read_pipe(rng_pipe, szgr);  
if(is_valid_reserve_id(rid)) {  
    //read random number from the pipe.  
    read_pipe(rng_pipe,rid,lid, &rn);  
    work_group_commit_read_pipe(rng_pipe, rid);  
}
```

The `consumer_kernel` then uses this set of random number and constructs the histogram. The CPU creates the same histogram and verifies whether the histogram created by the kernel is correct. Here, `lid` is the local id of the work item, obtained by `get_local_id(0)`.

The example code demonstrates how you can use a pipe as a convenient data structure that allows two kernels to communicate. It's really pretty simple.

In OpenCL 1.2, this kind of communication typically involves the host – although kernels can communicate without returning control to the host. Pipes, however, ease programming by reducing the amount of code that some applications require. There are additional examples of pipes used in conjunction with device enqueue, which we will explore in later blogs in this series.

To conclude, using pipes in OpenCL 2.0 can make your code simpler and more readable. Don't believe us? Write your own OpenCL 2.0 programs and tell us about the difference!

Sample code and readme

The sample code demonstrates the use of the pipes feature of OpenCL.2.0 using the SimplePipe (producer/consumer kernels) sample:




- The sample code and readme are provided [here](#).
- Install AMD OpenCL 2.0 Driver located [here](#). This page also has a complete list of supported platforms.
- Build the sample using the instructions in the readme.
- Let us know your feedback and comments on the [Developer Central OpenCL forum](#).

– Prakash Raghavendra

Dr. Prakash Raghavendra is a technical lead at AMD. He has several years of experience in developing compilers and run-time. His postings are his own opinions and may not represent AMD's positions, strategies or opinions. Links to third party sites, and references to third party trademarks, are provided for convenience and illustrative purposes only. Unless explicitly stated, AMD is not responsible for the contents of such links, and no third party endorsement of AMD or any of its products is implied.

分类: [GPGPU \(OpenCL\)](#), [OpenCL\(SDAccel\)](#)



 [pengdada](#)
 [关注 - 0](#)
 [粉丝 - 0](#)

[+加关注](#)

« 上一篇 : [DLatch by Verilog](#)

» 下一篇 : [win7 配置微软的深度学习caffe](#)

posted @ 2016-03-14 15:17 pengdada 阅读(55) 评论(0) 编辑 收藏

注册用户登录后才能发表评论，请 [登录](#) 或 [注册](#)，[访问网站首页](#)。

【推荐】50万行VC++源码: 大型组态工控、电力仿真CAD与GIS源码库

【阿里云】云计算科技红利邀您免费体验！云服务器、云数据库等35+产品，6个月免费使用！

【免费】从零开始学编程，开发者专属实验平台免费实践！

【推荐】又拍云年中大促！现在注册，充值最高送4800元



最新IT新闻:

- Google放出Android Things Console新预览
- 马斯克的Boring隧道最新进展：已完成竖井挖掘
- 王者荣耀近6成玩家是小学生？月入30亿的腾讯并不荣耀
- 腾讯王卡福利：1元500MB全国流量 还可跨省配送
- 华为推出三款新MateBook笔记本电脑 今日开始在美预订

» 更多新闻...

美团云周年庆88元特价机

2核-4G-2M带宽-100G硬盘

最新知识库文章:

- [小printf的故事：什么是真正的程序员？](#)
- [程序员的工作、学习与绩效](#)
- [软件开发为什么很难](#)
- [唱吧DevOps的落地，微服务CI/CD的范本技术解读](#)
- [程序员，如何从平庸走向理想？](#)

» [更多知识库文章...](#)

Copyright ©2017 pengdada