OpenCL在图像、视频开源项目中的应用

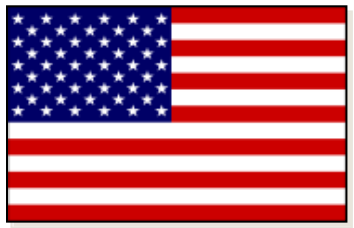
美国多核技术有限公司
高级软件经理
高　鹏
peng@multicorewareinc.com

目录

- 美国多核简介
- OpenCL在视频处理应用介绍
  - FFmpeg
  - x264
  - Handbrake
- OpenCL在图像处理应用介绍
  - GIMP
  - ImageMagick
- OpenCL案例分析
  - 视频处理中的去噪算法并行化
- OpenCL应用中的问题
  - 问题陈述
  - 问题解决方案
- OpenCL 异构计算前景展望

美国多核介绍

- 成立于2008年
- 在全球多地有研发中心
- 拥有世界最大的异构开发团队
- 博士/硕士所占比例超过 50%以上

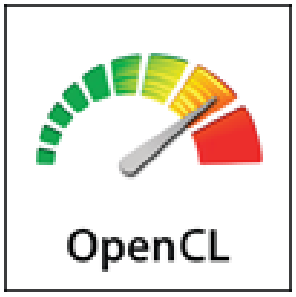


St. Louis
Champaign IL

Changchun
Beijing

Chennai

美国多核介绍



- 2011 AFDS – 展示了基于异构开发的工具链。包括 PPA, Task Manager, Slot Maximizer, Data Layout etc.
- 2012 AFDS – MCW 是主要的发起者之一
- MCW is member of HSA基金会, MCW CTO Wen-Mei Hwu领导其工具技术组
- Dedicate to support open source projects:
  - x264, ffmpeg, OpenCV, VLC, Handbrake, GIMP, ImageMagick, Crypto++ …

美国多核介绍

- ARM的新一代显示芯片也支持OpenCL与RenderScript。基于 RenderScript，我们开发了高级图像编辑应用和视频转换应用，并在CES 2013 以及MWC2013进行展示。
- 为Telestream 公司提供高性能高清视频处理工具，伦敦奥运会的高清转播中就用到了我们的相关技术。
- 我们的合作伙伴还包括ViVu ,华盛顿大学, 伊利诺伊大学 等

美国多核介绍

多核目前主要工作基于如下：

- 高性能计算方面的编译器工具链的开发
  - 编译器工具
  - 适用于异构环境
  - 任务和内存管理
  - 数据布局优化
  - 程序概要分析及调试
  - 源到源转换
- 基于应用的产品和工程
  - 视频处理和图像处理
  - 地震数据分析
  - 医疗成像
  - 生物信息技术（Blast算法/NIH参考模型加速3500倍）
  - CFD（计算流体力学）
  - 半导体晶片缺陷识别
  - RIP引擎并行化
  - 加密技术

OpenCL
在视频处理应用介绍 ---FFmpeg

FFmpeg 是一个使用非常广泛的开源视频处理软件，支持目前所有主流视频格式的解码。同时，FFmpeg还提供了非常多的视频特效处理功能，每个功能以一个filter的形式提供。国内很多播放器厂家都使用过或借鉴过其中的代码。

我们的工作主要是如下几方面：

- 视频处理中的解码部分的优化
  使用AMD 的OpenDecode 实现H264，Mpeg2 and VC1；
  使用OpenCL优化V8的解码；

- 视频特效处理部分的优化
  使用OpenCL 优化deshake, sharp, scaling

- 视频编码部分的优化
  使用AMD的OpenEncode 实现H264和Mpeg2的编码

- 整个处理流程的优化
  修改现有机制从而实现在不同的处理部分共享GPU buffer.减少DMA的传输开销；

OpenCL
在视频处理应用介绍 ---FFmpeg

输入：1080p(ivf)
输出：不写入文件

Filter	FPS		OpenCL/No OpenCL
	A10M GPU (Using OpenCL)	A10M CPU (No OpenCL)	
Deshake	25	1.2	20.83333333
Sharpen	85	1.4	60.71428571
Unsharp	43	23	1.869565217
Thumbnail	174	74	2.351351351

OpenCL
在视频处理应用介绍 --- x264

X264始于2003年，从开源社区的MPEG4-ASP编码器Xvid小有所成时开始，经过几年的开发，现在成为一个最受欢迎的H264视频编码器。

它的特点在于灵活，高效，在同样的片源、码率情况下，用x264编码出的视频一定会比rmvb画质更好。

优化工作主要是优化x264里面lookahead模块。

- 运动估计
- 帧内预测
- 帧间预测
- 双向预测
- 帧间/帧内的并行求和

OpenCL
在视频处理应用介绍 --- x264

从测试结果看，优化后的 x264在处理速度上和视频质量上都高于未优化的版本

	OpenCL x264		Public x264	
	FPS	PSNR	FPS	PSNR
Very fast	24.65	44.664	23.04	44.685
Faster	14.76	45.09	13.94	45.088
Faster	9.35	45.353	9.02	45.35
Medium	7.63	45.309	7.48	45.313
Medium (--b-adapt=2)	7.24	45.358	6.88	45.339
slow	4.81	45.395	4.72	45.373
slower	1.73	45.552	1.72	45.526

Input Clip	POTC 1080p
Target Bitrate	13 Mbps
frames	1000
Platform	A10M

OpenCL
在视频处理应用介绍 --- handbrake

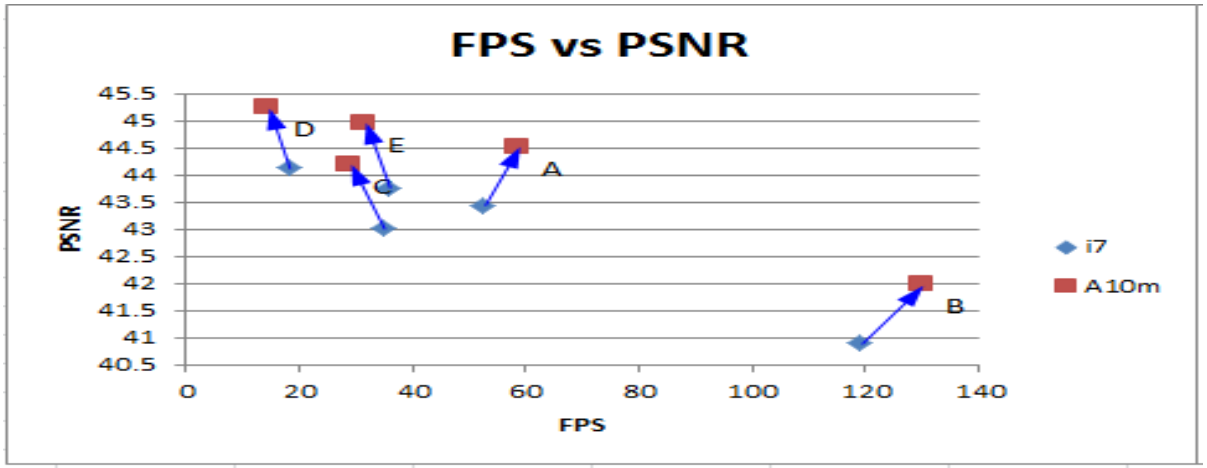
Handbrake是一个开源，跨平台（Win/Linux/ios），多线程视频转码器，可以处理常见的多媒体文件和任何非保护的DVD或蓝光光碟。支持输出的文件容器包括：MP4（M4V）、MKV，视频编码支持：H.264(x264)、MPEG-4、MPEG-2 (libav)、Theora(libtheora)，音频编码支持：AAC、CoreAudio AAC/HE-AAC (OS X Only)、MP3、Flac、AC3、Vorbis。视频处理支持Title/chapter/seconds/frames选择，多任务队列，可选择Deinterlacing、Decomb、Detelecine、Deblock、Grayscale, Cropping、scaling处理，支持视频预览，内置常见输出设备（如：ipad、iphone）

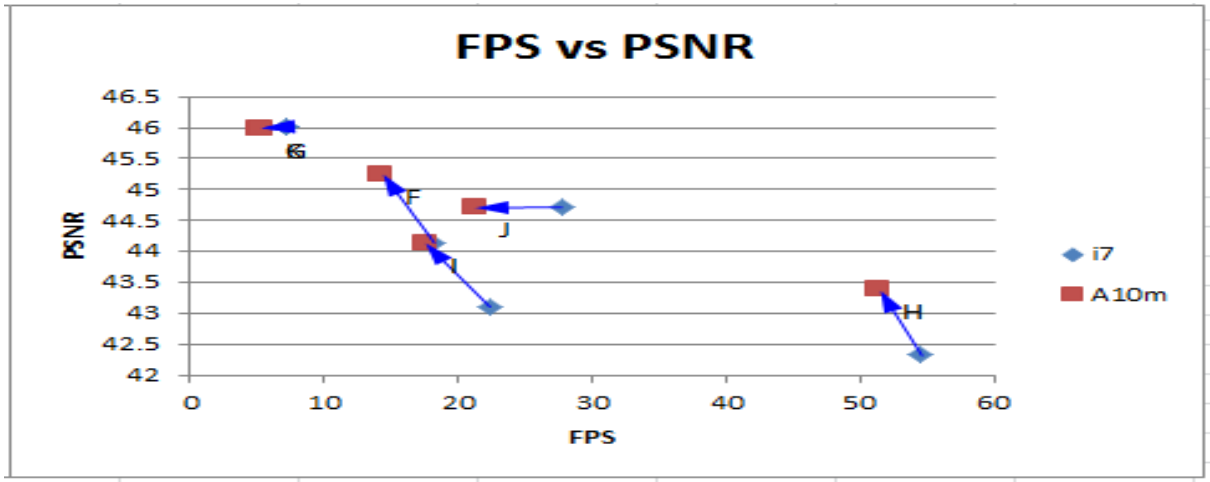
我们做了如下优化：

- Dxva2硬解，包括：H264（m2ts/vob），Mpeg2（m2ts/vob），VC1（m2ts/wmv）
- OpenCL优化，硬解后的NV12转YUV，scale filter，decomb filter
- GUI，添加启动硬解和OpenCL优化的控件
- 添加控制编译Dxva2硬解和OpenCL的编译选项（支持32/64位编译）

OpenCL
在视频处理应用介绍 --- handbrake

Output Target	Chart Label
universal	A
iPod	B
iPhone iPod Touch	C
iPad	D
AppleTV	E
AppleTV2	F
AppleTV3	G
Android	H
Android Tablet	I
Normal	J
High Profile	K



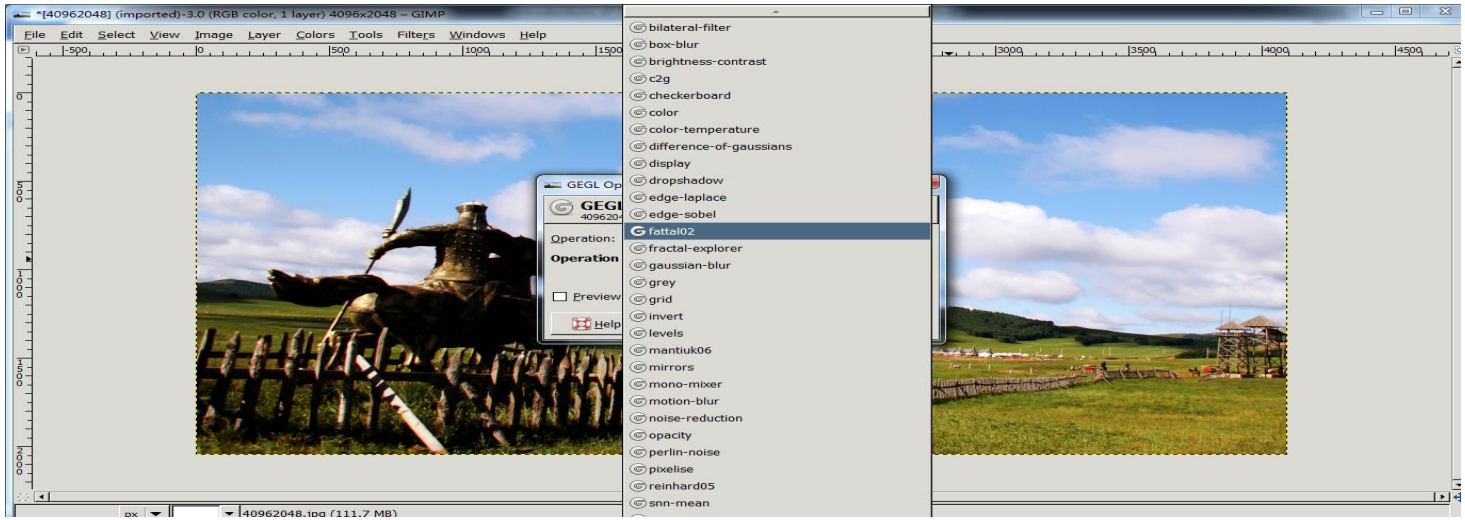


输入视频:
1080p MPEG2 m2ps)

OpenCL
在图像处理应用介绍 --- GIMP

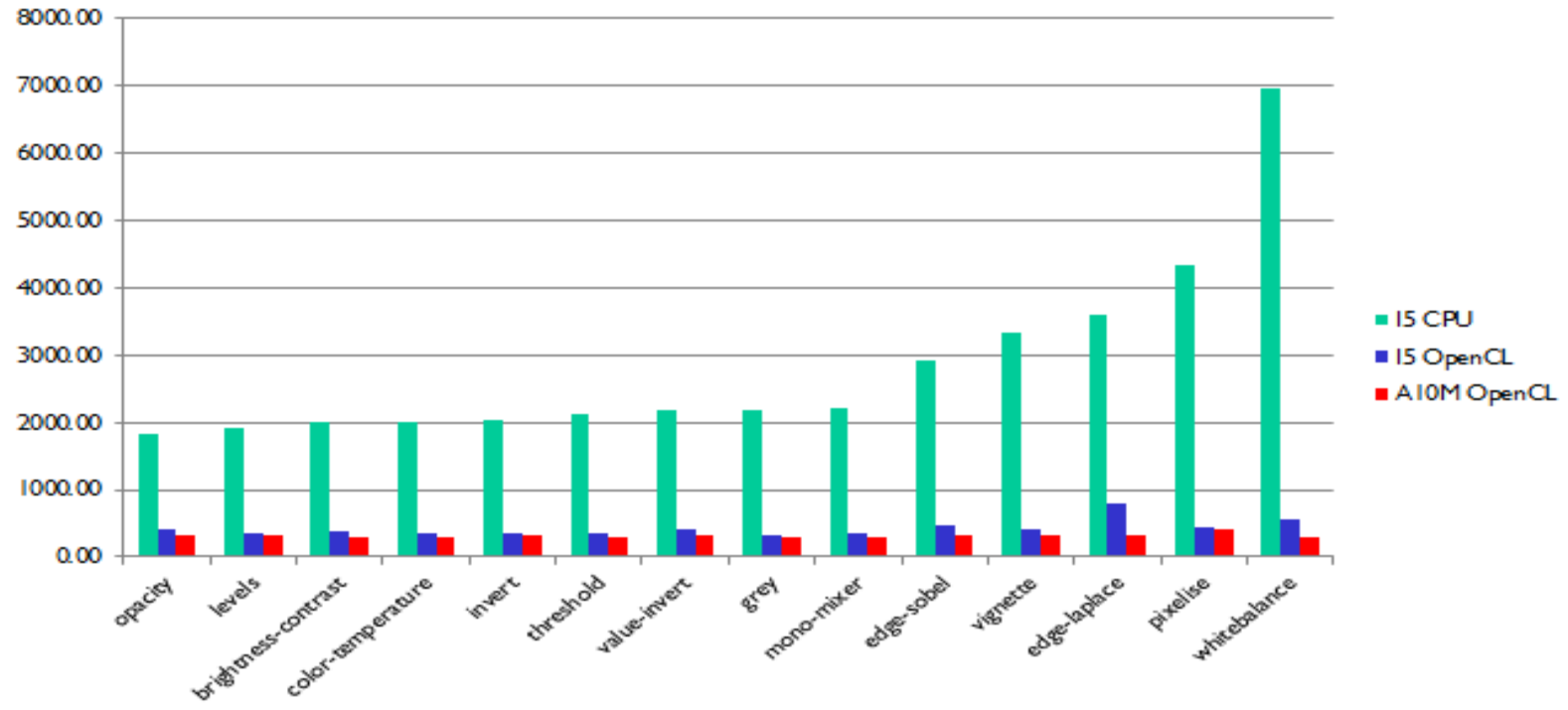
GIMP 是 GNU 图像处理程序(GNU Image Manipulation Program)的缩写。包括几乎所有图象处理所需的功能，号称Linux下的PhotoShop。GIMP在Linux系统推出时就风靡了许多绘图爱好 者的喜爱，它的接口相当轻巧，但其功能却不输于专业的绘图软件；它提供了各种的影像处理工具、滤镜，还有许多的组件模块，对于要制作一个又酷又炫的网页按 钮或网站Logo来说是一个非常方便好用的绘图软件，因为它也提供了许多的组件模块，你只要稍加修改一下，便可制作出一个属于你的网页按钮或网站 Logo。

在2012的年AMD 的Fusion开发者峰会中，由一个巴西的开发者展示了我们优化后的GIMP。



OpenCL
在图形处理应用介绍 --- GIMP

- Performance data



OpenCL
在图像处理应用介绍 --- ImageMagick

ImagMagick 是一个跨平台的图形处理软件，它的功能涵盖了所有图形处理中的常用操作，被誉为Linux上的photoshop. 在这个项目中，我们的工作主要从如下方面来做的。

- 使用OpenCL 优化了一些常用图像处理操作

  大约有20 个左右的功能用OpenCL实现，其中获得的最大加速比达到～10X。

- JPEG 解码器的优化

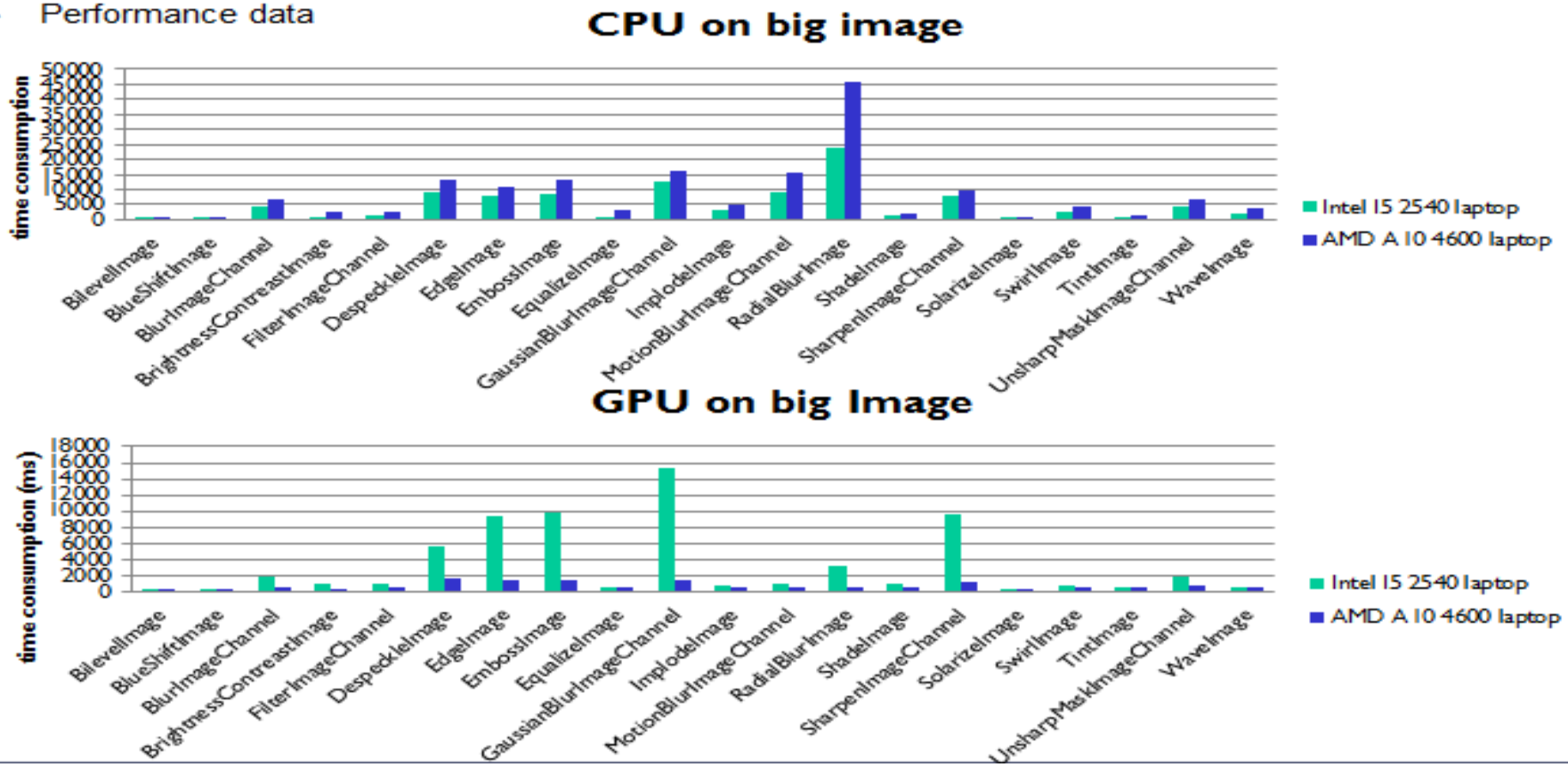
  优化了 imageMagick中的jpeg 的解码，其中包括用SSE实现Huffman解码和用OpenCL来实现反量化，反变换以及图像重采样的功能。

- 处理流程的优化

  尽可能的保证数据保存在GPU中，从而避免在不同的处理阶段中数据需要在GPU和CPU 之间做频繁的拷贝，

OpenCL
在图形处理应用介绍 --- ImageMagick

- Performance data

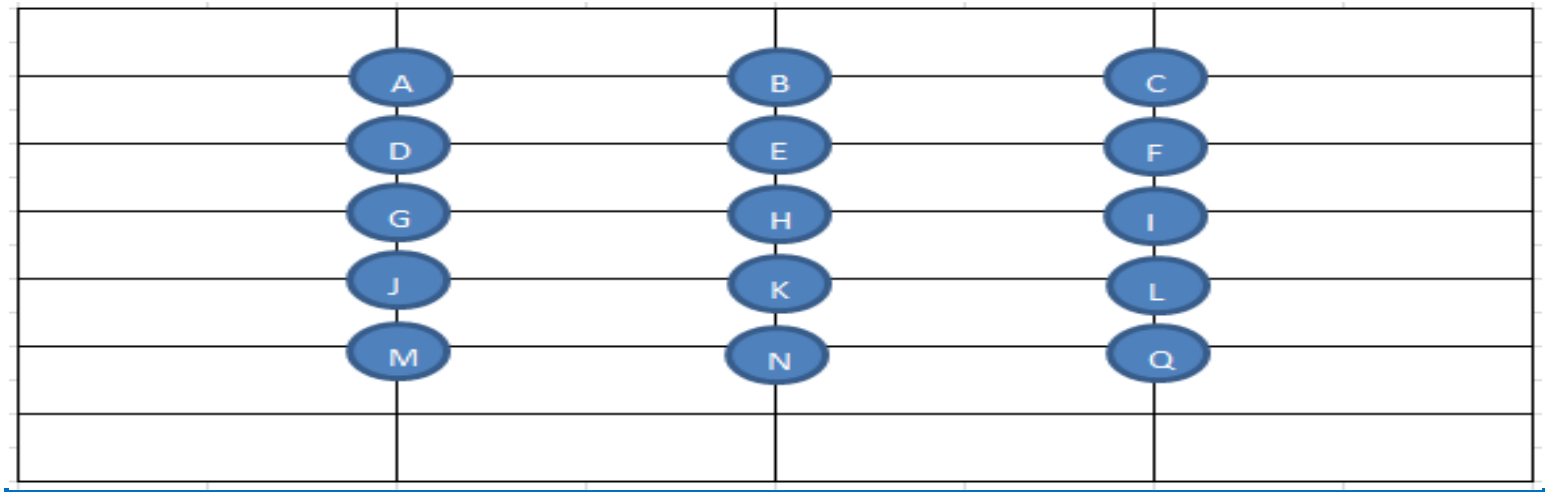


OpenCL 案例分析

视频处理中去噪算法(de-noise)并行化

- De-noise 算法在视频处理中是一个使用频繁的功能。
- De-noise 算法是对输入的一帧，对该帧中的所有像素点进行计算。帧数据的存储格式就相当于一个二维数组。通常情况下，一帧数据是以Yuv420p格式存储的。Y,U,V分别存储亮度和色度信息。
- 由于De-noise 的算法中存在如下数据依赖关系：

在下面的图形中， A, B,C,D,E,F,G,H,I,J,K,L,M,N,Q 分别代表不同的像素点，在该去噪算法中，如果需要计算H像素点，就必须先计算E和G点，因为这2个点在计算H点需要用到。



OpenCL 案例分析

视频处理中去噪算法并行化

```
/*****************************************************************************
 * LowPassMul: Low-pass filter calculation
 *****************************************************************************/
static inline unsigned int LowPassMul(unsigned int PrevMul,
                                      unsigned int CurrMul,
                                      int* Coef)
{
    int dMul= PrevMul-CurrMul;
    unsigned int d=((dMul+0x10007FF)>>12);
    return CurrMul + Coef[d];
}

/*****************************************************************************
 * Denoise: denoise video function
 *****************************************************************************/
static void Denoise(unsigned char *Frame,
                    unsigned char *FrameDest,
                    unsigned int *LineAnt,
                    unsigned short **FrameAntPtr,
                    int W, int H, int sStride, int dStride,
                    int *Horizontal, int *Vertical, int *Temporal)
{
    /**
     *  do other things in here
     */

    /**
     * First line has no top neighbor. Only left one for each pixel and
     * last frame
     */
    for (X = 1; X < W; X++)
    {
        LineAnt[X] = PixelAnt = LowPassMul(PixelAnt, Frame[X]<<16, Horizontal);
        PixelDst = LowPassMul(FrameAnt[X]<<8, PixelAnt, Temporal);
        FrameAnt[X] = (PixelDst+0x1000007F)>>8);
        FrameDest[X]= (PixelDst+0x10007FFF)>>16);
    }

    for (Y = 1; Y < H; Y++)
    {
        unsigned int PixelAnt;
        unsigned short* LinePrev=&FrameAnt[Y*W];
        sLineOffs += sStride, dLineOffs += dStride;
        /* First pixel on each line doesn't have previous pixel */
        PixelAnt = Frame[sLineOffs]<<16;
        LineAnt[0] = LowPassMul(LineAnt[0], PixelAnt, Vertical);
        PixelDst = LowPassMul(LinePrev[0]<<8, LineAnt[0], Temporal);
        LinePrev[0] = (PixelDst+0x1000007F)>>8);
        FrameDest[dLineOffs]= (PixelDst+0x10007FFF)>>16);

        for (X = 1; X < W; X++)
        {
            unsigned int PixelDst;
            /* The rest are normal */
            PixelAnt = LowPassMul(PixelAnt, Frame[sLineOffs+X]<<16, Horizontal);
            LineAnt[X] = LowPassMul(LineAnt[X], PixelAnt, Vertical);
            PixelDst = LowPassMul(LinePrev[X]<<8, LineAnt[X], Temporal);
            LinePrev[X] = (PixelDst+0x1000007F)>>8);
            FrameDest[dLineOffs+X]= (PixelDst+0x10007FFF)>>16);
        }
    }
}
```

de-noise的C 代码。所有的像素，除了第一行的数据，都在一个具有2层循环的语句中被处理。

OpenCL 案例分析

视频处理中去噪算法并行化

```
65  {
66          /*Y plane */
67          Denoise(p_Y_PLANE_src,  p_Y_PLANE_out,
68                  priv->Line,  &priv->Frame[0],
69                  i_Y_PLANE_src_pitch,  i_Y_PLANE_src_lines,
70                  i_Y_PLANE_src_pitch,  i_Y_PLANE_out_pitch,
71                  priv->Coefs[0],
72                  priv->Coefs[0],
73                  priv->Coefs[1]);
74
75          /* U plane */
76          Denoise(p_U_PLANE_src,  p_U_PLANE_out,
77                  priv->Line,  &priv->Frame[1],
78                  i_U_PLANE_src_pitch,  i_U_PLANE_src_lines,
79                  i_U_PLANE_src_pitch,  i_U_PLANE_out_pitch,
80                  priv->Coefs[2],
81                  priv->Coefs[2],
82                  priv->Coefs[3]);
83
84          /* V plane */
85          Denoise(p_V_PLANE_src,  p_V_PLANE_out,
86                  priv->Line,  &priv->Frame[2],
87                  i_V_PLANE_src_pitch,  i_V_PLANE_src_lines,
88                  i_V_PLANE_src_pitch,  i_V_PLANE_out_pitch,
89                  priv->Coefs[2],
90                  priv->Coefs[2],
91                  priv->Coefs[3]);
92      }
```

在C代码中,Y, U 和V 被串行处理.

OpenCL 案例分析

视频处理中去噪算法并行化

该算法的并行化思想主要是分解行和列之间的依赖关系。

- 首先，需要把在行和列上的这种像素依赖关系分拆为可以单独处理的部分，在我们的优化方案中，我们，就是把计算过程分拆行与列这2部分来计算， 其中一个部分是实现以列为处理单元来并处理,另外一个计算部分是实现以行为处理单元来并行处理.

- 功能分解
  - 考虑用第一个kernel来并行处理每帧数据中的行数据。算法在这一步时，只考虑列与列之间的依赖关系，也就是只考虑左右像素点的依赖关系，上下像素之间的关系可以不用考虑。
  - 考虑用第二个kernel来并行处理每帧数据中的列数据。算法在这一步时，只有行与行间的依赖关系，也就是只考虑上下像素点的依赖关系，左右像素之间的关系可以不用考虑。

OpenCL 案例分析

视频处理中去噪算法并行化

```
/*
*worktimes:
*     global workitems : Y_W_cl + U_W_cl + V_W_cl
*     work group size : 256
*/

__kernel void deNoise(__global unsigned int *Y_PixelAnt,
                      __global unsigned char *YUV_FrameDest,
                      int U_offset,
                      int V_offset,
                      __global unsigned short* Y_FrameAnt,
                      int Y_W,
                      int Y_H,
                      int Y_W_cl,
                      int Y_dStride,
                      __constant int *Y_Vertical,
                      __constant int *Y_Temporal,
                      __global unsigned int *U_PixelAnt,
                      __global unsigned short* U_FrameAnt,
                      int U_W,
                      int U_H,
                      int U_W_cl,
                      int U_dStride,
                      __constant int *U_Vertical,
                      __constant int *U_Temporal,
                      __global unsigned int *V_PixelAnt,
                      __global unsigned short* V_FrameAnt,
                      int V_W,
                      int V_H,
                      int V_dStride,
                      __constant int *V_Vertical,
                      __constant int *V_Temporal)
{

    int gidx=get_global_id(0);

    if (gidx < Y_W_cl)  /* Y plane */
    {
        if (gidx < Y_W)
        {
            cacl2(Y_PixelAnt, YUV_FrameDest, Y_FrameAnt, Y_W, Y_H, Y_dStride, Y_Vertical, Y_Temporal, gidx);
        }
    }
    else if (gidx < (Y_W_cl + U_W_cl))  /* U plane */
    {
        if (gidx < (Y_W_cl + U_W))
        {
            cacl2(U_PixelAnt, (YUV_FrameDest + U_offset), U_FrameAnt, U_W, U_H, U_dStride, U_Vertical, U_Temporal, (gidx - Y_W_cl));
        }
    }
    else
    {
        if (gidx < (Y_W_cl + U_W_cl + V_W)) /* V plane */
        {
            cacl2(V_PixelAnt, (YUV_FrameDest + V_offset), V_FrameAnt, V_W, V_H, V_dStride, V_Vertical, V_Temporal, (gidx - Y_W_cl - U_W_cl));
        }
    }

}
```

二维数据中的一列的数据被一个GPU中的work-item来处理，并且Y,U,V将在不同的GPU 的work group中被处并行理。

OpenCL 案例分析

视频处理中去噪算法并行化

```
/*
* workitmes :
*           global workitems : Y_H_cl + U_H_cl + V_H
*           workgroup size : 256
*/
__kernel void PixelAnt(__global unsigned char *YUV_Frame,
                       int U_offset,
                       int V_offset,
                       __global unsigned int *Y_PixelAnt,
                       int Y_W,
                       int Y_H,
                       int Y_H_cl,
                       __constant int *Y_Horizontal,
                       __global unsigned int *U_PixelAnt,
                       int U_W,
                       int U_H,
                       int U_H_cl,
                       __constant int *U_Horizontal,
                       __global unsigned int *V_PixelAnt,
                       int V_W,
                       int V_H,
                       __constant int *V_Horizontal)

{
    int gidx = get_global_id(0);

    if (gidx < Y_H_cl)  /* Y plane */
    {
        if (gidx < Y_H)
        {
            cacl(YUV_Frame, gidx, Y_PixelAnt, Y_W, Y_H, Y_Horizontal);
        }
    }
    else if (gidx < (Y_H_cl + U_H_cl))  /* U plane */
    {
        if (gidx < (Y_H_cl + U_H))
        {
            cacl((YUV_Frame + U_offset), (gidx - Y_H_cl), U_PixelAnt, U_W, U_H, U_Horizontal);
        }
    }
    else
    {
        if (gidx < (Y_H_cl + U_H_cl + V_H)) /* V plane */
        {
            cacl((YUV_Frame + V_offset), (gidx - Y_H_cl - U_H_cl), V_PixelAnt, V_W, V_H, V_Horizontal);
        }
    }

}
```

二维数据中的一行的数据被一个GPU中的work-item来处理，并且Y,U,V将在不同的GPU 的work group中被处并行理。

OpenCL 案例分析

视频处理中去噪算法并行化

```
130 inline void cacl(__global unsigned char * src , int gidx, __global unsigned int * dst , int w , int h, __constant int * ref)
131 {
132     uchar16 uchar16_Frame[2];
133     uchar16 u16_dst;
134     //uint16 uint16_PixelAnt;
135     uint8 uint8_PixelAnt[2];
136     uint * d = uint8_PixelAnt;
137     __global uchar * srcrow = src + gidx*w;
138     __global uint * dstrow = dst + gidx*w;
139     uchar*  char_16;
140
141     uint refHead = srcrow[0] << 16 ;
142
143     bool cur = 0;
144     uchar16_Frame[cur] = vload16(0, (srcrow + 0));
145
146     for (int j = 0; j < (w >> 4); j++)
147     {
148         char_16 = &uchar16_Frame[cur];
149         cur = 1 - cur;
150         uchar16_Frame[cur] = vload16(0, (srcrow + ((j + 1) << 4)));
151
152         if (j > 0)
153             d[0] =  LowPassMul(refHead, char_16[0]<<16, ref);
154         else
155             d[0] = refHead;
156
157         for(int i = 1; i< 16; i++)
158         {
159             d[i] = LowPassMul(d[i-1], char_16[i]<<16, ref);
160         }
161
162         refHead = d[15];
163
164         //vstore16(uint16_PixelAnt , 0, (dstrow + (j << 4)));
165         vstore8(uint8_PixelAnt[0] , 0, (dstrow + (j << 4)));
166         vstore8(uint8_PixelAnt[1] , 0, (dstrow + (j << 4) + 8));
167     }
168
169 }
```

OpenCL 案例分析

视频处理中去噪算法并行化

在该算法的优化中需要注意如下的问题：

- 因为我们的设计是让一帧的Y,U和V同时在GPU里面被处理。因为在这个算法中，亮度和色度是没有依赖关系的。
- 因为在实际的处理中，视频帧的大小不会总是GPU 的work-group的大小的倍数，因此，我们需要对数据进行填充以保证在一个GPU中的一个work-group中的所有线程总是处理相同面的数据（或者是Y，或者是U，或者V中的数据），不能出现work-group中部分线程处理的是Y中的数据，而另外一些在该work-group中的线程处理的是U或V中的数据。

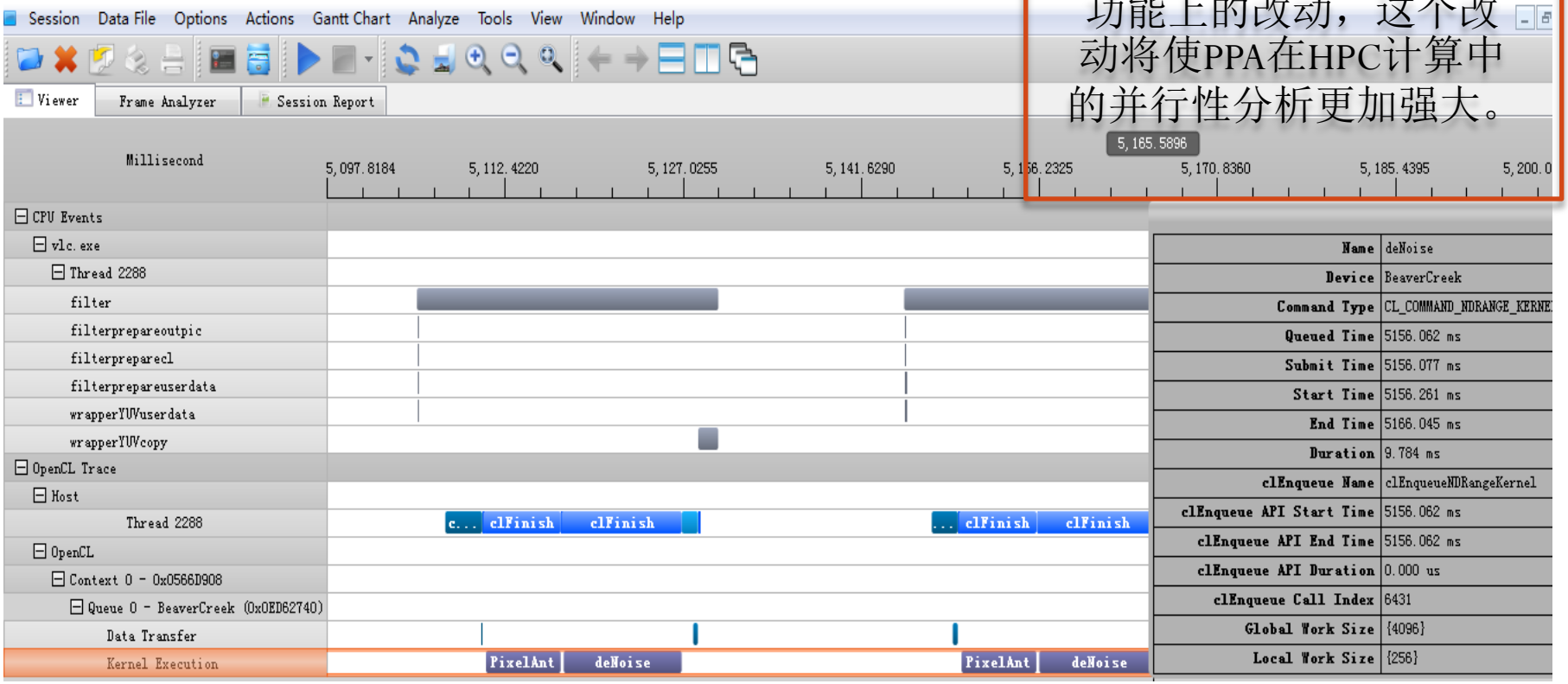
例如： 要处理的视频大小是1920X1080. 每帧的数据格式yuv420p, GPU的 work-group 大小是256. 该算法中在并行处理行数据的过程中，处理Y需要的GPU 线程总数是1024个（本来是1080，填充为1024），处理U 需要的GPU 线程总数是是768（本来是540）.则总的GPU线程数是1024 + 768 *2 = 2560 。 这就意味着GPU用2560个线程同时处理一帧里的所有数据。

OpenCL 案例分析

视频处理中去噪算法并行化

性能分析—PPA在性能分析中的应用

目前，我们正在对PPA做功能上的改动，这个改动将使PPA在HPC计算中的并行性分析更加强大。



OpenCL 案例分析

视频处理中去噪算法并行化

De-noise 性能数据 – CPU VS GPU

video_size:1280x720							
i3-2330M(laptop)		i5-2540M(laptop)		i7-2760QM(laptop)		i7-2600K(desktop)	
cpu	intel-opencl	cpu	intel-opencl	cpu	intel-opencl	cpu	intel-opencl
Total Time (enable PPA)	Total Time (enable PPA)	Total Time (enable PPA)	Total Time (enable PPA)	Total Time (enable PPA)	Total Time (enable PPA)	Total Time (enable PPA)	Total Time (enable PPA)
35	64	21	47	19	21	17.5	21
video_size:1920x1080							
Trinity(A10) + GTX480		Trinity(A10) + HD7970		Trinity(A10) + HD7660D		Trinity(A10) + GTX460	
cpu	enable-opencl	cpu	enable-opencl	cpu	enable-opencl	cpu	enable-opencl
Total Time (enable PPA)	Total Time (enable PPA)	Total Time (enable PPA)	Total Time (enable PPA)	Total Time (enable PPA)	Total Time (enable PPA)	Total Time (enable PPA)	Total Time (enable PPA)
22	13	22	14	22	17	22	16

OpenCL 在 应用中的问题 --问题陈述

通过我们在大量的工程中使用OpenCL, 发现有如下问题：

- 现有算法或是规范都是基于串行程序的思想来制定的。在解决某些问题的时候，必须考虑在不影响输出结果的同时，如何从串行的算法转换为并行的算法，这对OpenCL编程人员在算法方面要有比较好的基础。
- 对OpenCL编程人员来说，即需要对OpenCL规范非常熟悉，同时也需要对设备的特性要有足够深入的了解。因为即使是相同的设备不同的型号，提供的并行计算能力也是不同的。
- 对OpenCL 编程人员来说，还需要掌握相关专业领域知识，否则的话，即使非常了解硬件设备特性和非常了解OpenCL，也很难实现出高度并行化的OpenCL 代码。因为这要求OpenCL开发人员具有更高的要求。
- 现有基于OpenCL实现的通用代码或是第三方提供的库主要在科学计算，研究性应用 或是专业领域的应用居多，但涉及普通民众生活的应用不多。
- 设备本身的特性还有许多限制，比如 在GPU和CPU 之间的数据传输，这个是目前性能影响最大的因素，AMD提供的zero copy 特性以及HAS就提供了对数据拷贝性能的优化。

OpenCL 在 应用中的问题 --问题解决方案

对于上面提出的问题。在我们的工作中，我们也在试图从软件层面去解决或是降低这些问题的难度。已经或是正在做的有如下的工作:

- 形成一些通用的库。比如在图像处理方面，把我们在工作中所遇到的问题进行归纳，总结，形成一些比较通用的库，这样在后续的开发中，就能快速的解决一些问题。
  - 在视频处理方面，产品VPL是一款视频处理方面的库。同时，我们正做一些前沿的工作，比如H265 相关的工作，比如如何从低分比率视频创建高分辨率视频。
  - 在图形处理方面，也在形成一些新的产品。
  - 在通用数学库方面，也在形成一些新的产品。
- 我们也在做一些自动化的工具，这些工具能让用户更好的发挥并行计算的能力，降低开发者难度，缩减用户部署应用所需的时间。
  - 产品MxPA. 是一款把OpenCL程序转换成CPU上运行的动态库，并提供和OpenCL官方标准一致的运行环境， 让openCL程序不需要修改就可以在各种CPU上运行.目前支持x86 和x86_64平台，其他平台的支持也正在我们的产品计划中。.
  - 产品PPA，是一款并行路劲分析器，该工具给开发者提供了一个了解运行时用户应用程序的执行情况，协助用户找到用户应用程序的瓶颈。

OpenCL 异构计算前景展望

随着硬件的发展，云计算以及客户终端更加小型化，智能化的发展。基于客户端的数据密集型计算将会成为一个发展方向。特别是基于移动方向的发展。

同时，随着并行计算的普及化，并行计算将会走进普通百姓的生活中。

Thank You!