图像的快速傅里叶DirectX* 11中的处理

介绍

快速傅里叶变换（FFT）是使用分治方法的离散傅里叶变换（DFT）的实现。DFT可以将任何离散信号（例如图像）变换到频域和从频域变换。一旦在频域中，在图像域中通常昂贵的许多效果就变得微不足道且便宜。图像域和频域之间的变换可能导致显著的开销。此示例演示了一个优化的FFT，它使用计算着色器和共享本地内存（SLM）通过减少内存带宽来提高性能。

本示例将讨论执行FFT的两种技术。第一种技术被称为UAV技术，并且通过在无序访问视图（UAV）之间重复地乒乓数据来操作。第二种技术，SLM（共享本地存储器）技术，是一个更内存带宽有效的方法，显示显着的性能增益时，瓶颈的内存带宽。

上一个示例“在DirectX 10中实现图像处理的快速傅立叶变换”使用像素着色器实现了UAV技术。关于UAV技术和FFT算法的更多信息，请参考该示例。

无人机技术

如原始示例中所述，UAV技术执行通常称为蝶形通过的通过。每一遍需要对真实和虚构输入纹理中的每一个进行两次完整的采样，并且向输出真实和复杂纹理写入一次。根据缓存体系结构的不同，这可能会导致由缓存抖动引起的性能问题。***图1：***示出了UAV技术所需的调度呼叫。

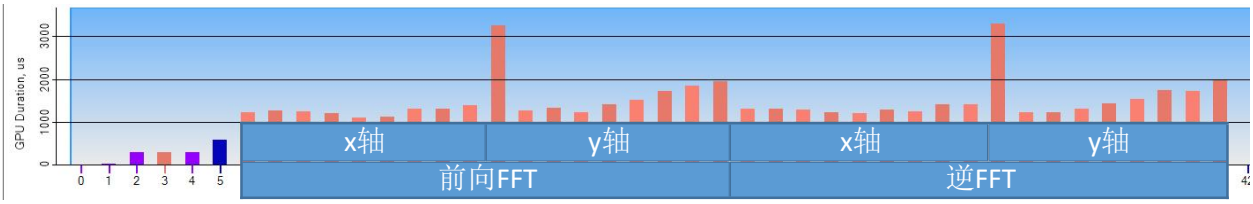

图1：*使用图形性能分析器（GPA）中捕获的UAV技术计算着色器分派调用*

SLM技术

SLM技术对每个FFT轴通道使用单个分派。它为纹理中的每一行分派一个线程组，每个线程组为纹理中的每一列创建一个线程。每行由一个线程组表示，该行内的每个像素由单个线程处理。每个

线程执行其相应像素的所有蝶形通过。由于传递之间的依赖性，线程组中的所有线程必须完成每个传递，然后任何线程才能继续进行下一个传递。***图2***显示了线程组的工作流。

图2：*线程组工作流。为输入纹理*

GroupMemoryBarrierWithGroupSync函数可以在每个线程执行蝶形传递之后使用，以强制执行此同步。

每遍的结果存储在SLM阵列中的交替索引中。源纹理的输入值最初被加载到数组的前半部分，并且每个后续通道将值写入另一半。本质上，UAV技术中使用的纹理之间的乒乓仍然发生，但在SLM内完成。

在每个线程上针对每个通道完成的操作包括：

1. 计算2个索引，用于索引到前一遍
2. 计算2个权重，用于缩放每个索引处的值。
3. 使用索引和权重执行基本运算，并将结果写入SLM阵列的另一侧。
4. 调用GroupMemoryBarrierWithGroupSync与其他线程同步。
5. 重复

所描述的操作在图3中的计算着色器代码中表示。***图4***显示了SLM技术所需的调度调用。

```
groupshared float3pingPongArray[4][LENGTH];
voidButterflyPass（intpassIndex，uintx，uintt0，uintt1，out float3resultR，out
float3resultI）
{
      uint 2指
      数;float 2权重;
      GetButterflyValues（passIndex，x，Indices，Weights）;

      float3 inputR1 =
      pingPongArray[t0][Indices.x];float3 inputI1 =
      pingPongArray[t1][Indices.x];

      float3 inputR2 =
      pingPongArray[t0][Indices.y];float3 inputI2 =
      pingPongArray[t1][Indices.y];

      resultR = inputR1 + Weights.x * inputR2 - Weights.y * inputI2;
      resultI = inputI1 + Weights.y * inputR2 + Weights.x * inputI2;
}

[numthreads（WIDTH，1，1）]
voidButterflySLM（uint3position：SV_DispatchThreadID）
{
      //将整行或整列加载到临时数组pingPongArray[0][position.x].xyz =
      TextureSourceR[position]; pingPongArray[1][position.x].xyz =
      TextureSourceI[position];

      uint4 textureIndices = uint4（0，1，2，

      3）;int i = 0; i =< 0; i++）
      {
            System. out. println（）;
            ButterflyPass（i，position.x，textureIndices.x，textureIndices.y，
                  pingPongArray[textureIndices.z][position.x].xyz，
                  pingPongArray[textureIndices.w][position.x].xyz）;
            textureIndices.xyzw = textureIndices.zwxy;
      }

      //最终蝴蝶将直接写入目标纹理GroupMemoryBarrierWithGroupSync（）;
      ButterflyPass（BUTTERFLY_COUNT - 1，position.x，textureIndices.x，
textureIndices.y，TextureTargetR[position]，TextureTargetI[position]）;
```

图3：*SLM技术实现*

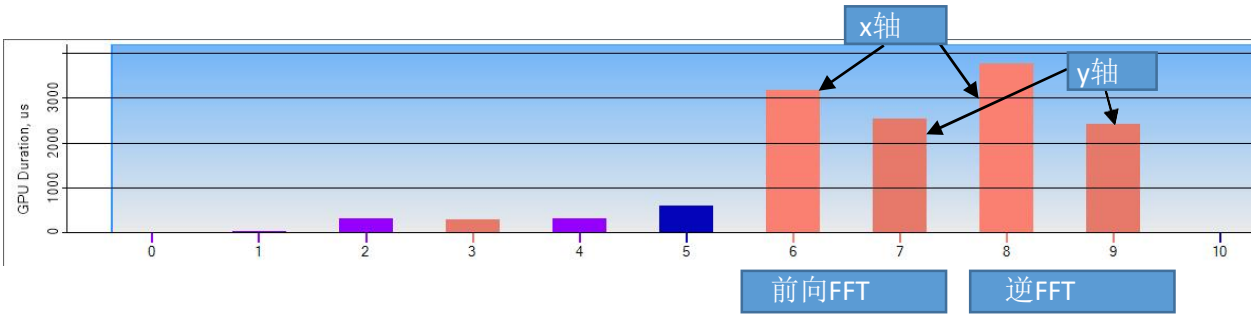

图4：*使用图形性能分析器（GPA）中捕获的SLM技术计算着色器分派调用*

通过微小的修改，该算法也可以被转置以在y轴上执行。

内存带宽分析

使用实平面和复（虚）平面来表示频域中的图像的每个颜色分量。这些飞机需要高精度，因此使用128位R32G32B32A32。

图5中的公式可以用于使用UAV技术计算X轴上的FFT的存储器带宽。BytesRead乘以2两次，因为每个蝶形通道需要4个纹理样本，2个实数和2个虚数。

```
TextureSize = Width * Height * 16
PassCount = log2（Width）
BytesRead = TextureSize * PassCount * 2 * 2
BytesWritten = TextureSize * PassCount * 2
```

图5：*UAV技术*

使用这些公式，***图6***显示了512x256纹理的内存带宽的计算结果：

```
TextureSize = 512 * 256 * 16 = 2097152
PassCount = log2（512）= 9
字节读取= 2097152 * 9 * 2 * 2 = 75497472
BytesWritten = 2097152 * 9 * 2 = 37748736
```

图6：*使用UAV技术*

也就是说，仅对于x轴通道，需要72MB的纹理读取和32MB的纹理写入。y轴变换和逆变换将进一步增加存储器带宽。

SLM技术通过利用每行可以被独立处理的事实来减少存储器带宽。该技术使用计算着色器将整行读取到共享本地内存中，执行所有蝶形通道，然后将结果一次性写入输出纹理。

图7示出了用于计算SLM技术的存储器带宽的简单公式;它们读取和写入每个纹理一次，并且不受通过计数的影响。

```
TextureSize = Width * Height * 4
BytesRead = TextureSize * 2
BytesWritten = TextureSize * 2
```

图7：*使用SLM技术的*

```
TextureSize = 512 * 256 * 16 = 2097152
字节读取= 2097152 * 2 = 4194304
写入字节= 2097152 * 2 = 4194304
```

图8：*使用SLM技术*

在***图8***中计算的4MB读取和写入带宽可以通过避免虚拟纹理的初始读取而进一步改进，因为它将始终为零。这使得读取带宽为2MB，写入带宽为4MB。对于X轴FFT变换，SLM技术将读取和写入带宽分别降低了36和8倍。

使用具有预先计算的权重和索引的蝶形查找纹理进一步增加了带宽。查找纹理存储2个索引和R32G32B32A32格式的2个权重。每个线程组将在所有蝶形通道的过程中消耗整个纹理，并且源纹理中的每行都有一个线程组。***图9***显示了在变换512x256纹理时使用蝶形查找纹理的公式。结果是大约18MB的额外纹理读取（***图10***）。

```
PassCount = log2（宽度）
ButterflyTextureSize = Width * PassCount * 16
BytesRead = ButterflyTextureSize * Height
```

图9：*使用蝶形查找纹理*

```
PassCount = log2（512）= 9
ButterflyTextureSize = 512 * 9 * 16 = 73728
字节读取= 73728 * 256 = 18874368
```

图10：*在512x256源纹理*

本节中的计算估计了理想的内存带宽，但没有考虑执行部分写入时发生的额外读取。这些读取可能是重要的，并且将根据硬件实现而变化。

蝴蝶查找纹理

在每个像素的每个通道上使用的索引和权重的计算可以被预先计算并存储在纹理中。使用蝴蝶纹理减少了计算，但显著增加了内存带宽。在应用每个蝶形之后，每一行将基本上已经采样了整个128位蝶形纹理。根据GPU架构，使用蝴蝶纹理实际上可能会降低性能，如图11所示。

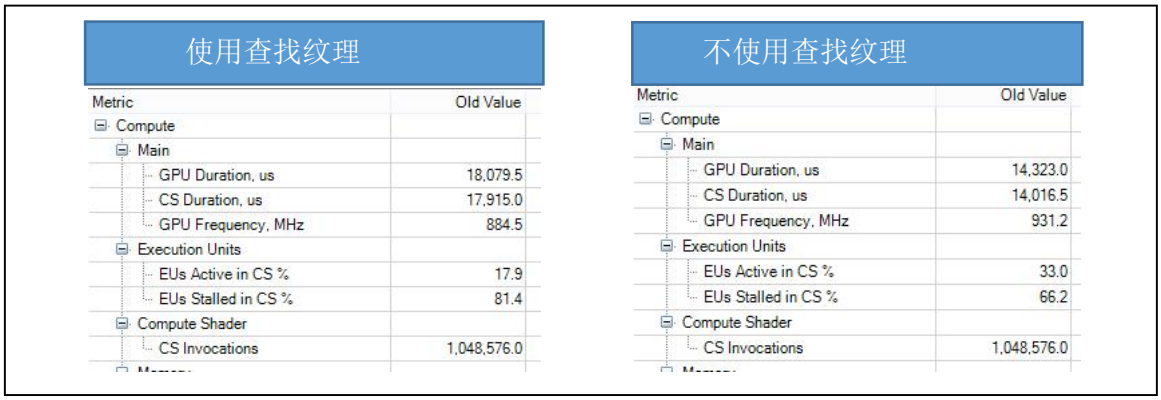

图11： *在英特尔® 酷睿™ i5- 4300 U处理器上运行图形性能分析器（GPA）的结果显示，计算着色器停止、内存读取和持续时间均有所减少。*

图12示出了在给定通过索引和偏移的情况下计算HLSL中的蝶形索引和权重的函数。

```
voidGetButterflyValues（uintpassIndex，uintx，out uint2indices，out float2weights
）
{
      public intcount = 2;
      intmaximum = maximum/2;

      int x& ~（int x）;int halfSectionOffset = x&
       （halfSectionWidth -1）; int sectionOffset = x&
       （sectionWidth -1）;

      sincos（2.0*PI*sectionOffset /（float）sectionWidth，weights.y，weights.x）;
      weights.y =-weights.y;

      index.x = sectionStartOffset + halfSectionOffset;
      index.y = sectionStartOffset + halfSectionOffset + halfSectionWidth;

      if（0）
      {
            indices = reversebits（indices）>>（32 - BUTTERFLY_COUNT）（LENGTH -1）;
      }
}
```

图12： *在计算着色器*

性能

SLM技术在英特尔®和AMD* 的多个GPU上显示出比UAV技术更大的性能优势。***图13***示出了具有各种配置的512x512纹理到频域和从频域的变换时间。

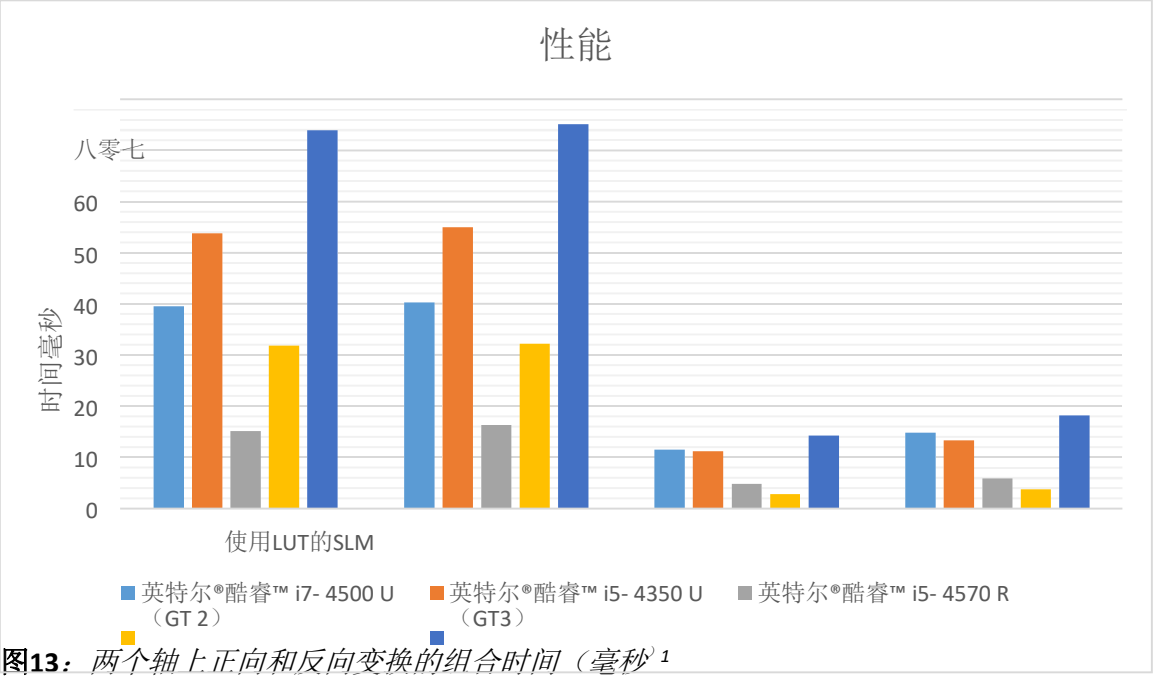

图13：*两个轴上正向和反向变换的组合时间（毫秒）*[1]

进一步优化

在编写此示例的过程中，还出现了一些其他优化，但为了简单起见，将其省略。

包装组件

一个额外的优化将是打包UAV数据以移除未使用的阿尔法通道。遗憾的是，UAV的纹理格式受到限制，并且不支持3分量32位精度纹理（R32G32B32）。然而，实平面和虚平面可以被打包成一个R32G32B32A32纹理和一个R32B32纹理。您可以将实数的所有分量存储在第一个纹理的RGB中，并将虚数分量拆分，以便一个分量位于ARGB纹理的Alpha分量中，而其他两个分量位于R32B32中。

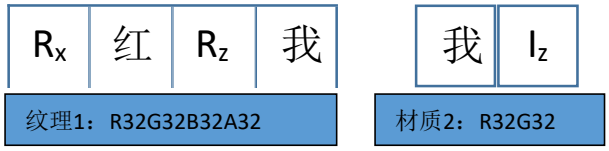

图14：*将实部和虚部值打包到R32G32B32A32纹理和R32G32纹理中*

避免不必要的高精度纹理

为了简化示例，在变换之前将输入图像复制到真实的R32G32B32A32纹理中。逆变换也输出到相同的纹理格式。直接从源纹理读取而不进行复制会减少内存带宽。反转的结果也可以被写入较低精度的纹理格式。

结论

在FFT算法中使用SLM作为暂存器，在分析的集成GPU上产生了实质性的性能增益。对于在纹理中存储临时数据的任何内存受限的多遍算法，您应该考虑使用SLM。可以使用示例代码分析其他GPU上的性能，以确定哪种技术最合适。由于这两种技术的相似性，支持这两种技术应该只需要最少的代码更改。

[1]性能测试中使用的软件和工作负载可能仅针对英特尔微处理器进行了性能优化。性能测试，如SYSmark* 和MobileMark*，使用特定的计算机系统、组件、软件、操作和功能进行测量。这些因素的任何变化都可能导致结果发生变化。您应该查阅其他信息和性能测试，以帮助您全面评估您的购买计划，包括该产品与其他产品结合使用时的性能

配置：**有关更多信息，请访问**http://www.intel.com/performance

通知

本文档中的信息与英特尔产品有关。本文档不以禁止令或其他方式授予任何知识产权的任何明示或暗示许可。除英特尔对此类产品的销售条款和条件中所规定的以外，英特尔不承担任何责任，并且不承担与销售和/或使用英特尔产品有关的任何明示或暗示的保证，包括与特定用途的适用性、适销性或侵犯任何专利、版权或其他知识产权有关的责任或保证。

除非英特尔另行书面同意，否则英特尔产品的设计和预期用途均不适用于英特尔产品故障可能导致人身伤害或死亡的情况。

英特尔可能随时更改规格和产品说明，恕不另行通知。设计者不得依赖任何标记为“保留”或“未定义”的特征或说明的缺失或特性。“英特尔保留这些条款以供将来定义，对于将来更改这些条款所产生的冲突或不兼容性，英特尔不承担任何责任。此处的信息如有更改，恕不另行通知。请勿使用此信息完成设计。

本文档中描述的产品可能包含称为勘误表的设计缺陷或错误，这些缺陷或错误可能导致产品偏离已发布的规范。可根据要求提供当前特征勘误表。

在下产品订单之前，请联系您当地的英特尔销售办事处或分销商，以获取最新的规格

如需获得本文档或其他英特尔文献中引用的带有订单号的文档副本，请致电1-800-548-4725或访问：http://www.intel.com/design/literature.htm

本文档中转载的任何软件源代码均根据软件许可证提供，并且只能根据该许可证的条款使用或复制。

英特尔、英特尔徽标和英特尔酷睿是英特尔公司在美国和/或其他国家/地区。版权所有© 2015英特尔公

司。版权所有

* 其他名称和品牌可能被声称为他人的财产。