

硕 士 研 究 生 读 书 报 告



题目 用Brook在显卡上进行流运算

作者姓名 郭奕庭

作者学号 NB15042

指导教师 李启雷

学科专业 移动互联网与游戏开发

所在学院 软件学院

提交日期 二○一五年12月22

Brook for GPUs: Stream Computing on Graphics Hardware

A Dissertation Submitted to

Zhejiang University

in partial fulfillment of the requirements for

the degree of

Master of Engineering

Major Subject: Software Engineering

Advisor: Li Qi Lei

By

Guo Yi Ting

Zhejiang University, P.R. China

2015

摘要

在本文中，我们呈现了基于GPUs编程的Brook编程环境，它是一个利用GPU硬件来执行运算的系统。Brook扩展自C语言，增加了基本的数据平行结构，使得可以把GPU看成一个流处理器。我们提供了一个编译器和一个运行系统，这样便抽象和虚拟了很多图形硬件的处理细节。此外，我们还提供了用GPU做运算的效率的分析，并拿它和CPU进行比较，来确定在哪些算法上面，GPU上运算的效果会比CPU上好。我们用了5个应用算法来做这样的评估，它们分别是：SAXPY、SGEMV BLAS运算、图像段处理、快速傅立叶变换和光线跟踪算法。用这些程序，我们展现了我们的Brook在GPU上运行的速度是CPU的7倍

**关键词**： 可编程图形硬件, 数据并行计算, 流计算, GPU 计算, Brook

Abstract

In this paper, we present Brook for GPUs, a system for general-purpose computation on programmable graphics hardware. Brook extends C to include simple data-parallel constructs, enabling the use of the GPU as a streaming co- processor. We present a compiler and runtime system that abstracts and virtualizes many aspects of graphics hardware. In addition, we present an analysis of the effectiveness of the GPU as a compute engine compared to the CPU, to deter- mine when the GPU can outperform the CPU for a particu- lar algorithm. We evaluate our system with five applications, the SAXPY and SGEMV BLAS operators, image segmen- tation, FFT, and ray tracing. For these applications, we demonstrate that our Brook implementations perform com- parably to hand-written GPU code and up to seven times faster than their CPU counterparts.

**Keywords：**Programmable Graphics Hardware, Data Parallel Computing, Stream Computing, GPU Computing, Brook

1引言

最近这些年，商用图形硬件已经快速的发展，从一个固定功能的管线到了拥有可编程的向量和段处理器。当这些为了实时的渲染而生的新功能都已经被引进的时候，我们可以发现，这些处理器特性指令集已经足够来执行超越了原来只做图像的领域。诸如线性代数、人体模拟、光线跟踪器等应用已经被证明能够运行在GPU上了。最早，GPU上的编程只能使用汇编语言，非常麻烦。然后，微软的HLSL，NVIDIA的Cg和OpenGL的GLsLang允许在一个更高的抽象层次来编写渲染器。然而，这些编程语言都不能帮助程序员来控制图形管线的其它方面，比如说分配纹理内存、加载一个渲染程序或者构建一个图形初始化。所以，上面这些程序的运行都需要额外知识，这些知识是关于最新的图形API和对于现代硬件的特点和限制的理解。此外，用户只能被迫的依赖一些图形相关的语句来表达他们的算法，比如纹理和三角面片等。所以，GPU编程通用性受到了限制，只有图形开发的高手才会使用。这篇论文提到的Brook是一个编程环境，它让开发者像面对一个平行处理流一样来操作GPU。

**2.**背景

**2.1**流硬件的演化

关于可编程的图形硬件要追溯到早期的可编程帧缓冲期架构。其中最有影响力的可编程图形系统是UNC的PixelPlanes系列，这种技术的顶峰代表则为PixelFlow机器。这些系统内嵌了像素处理器，就像一个SIMD的处理器，且有着芯片充当缓冲内存。Peercy et al 展示了OPENGL的架构师如何抽象成一个SIMD初期里。每一个渲染的执行都是通过执行一个SIMD的指令来完成的，这些指令会被换成一个基本的运算操作，来自动的更新缓存。用了这种抽象结构，我们就能够使用OPENGL 1.2和它的扩展。Thompson et al 发现GPU被作为一个通用的向量运算器来执行一个基于高层的图形库的软件扩展，这些扩展可以执行一系列的浮点数的运算。

SIMD和向量运算处理器包括对单指令的读取、执行和一个存储器。这些重要的成果扩宽了它们的使用。当今的图形处理硬件在运行程序时选择加载和存储指令数据到临时的寄存器而不是专门的内存。这是向量和流运算的抽象的主要区别。

流体系结构的计算机非常感兴趣的话题建筑[波夫和1995年沃特林顿;戈卡莱和戈默索尔1997]。例如，设想一下流处理器[Kapasi等。 2002]证明流的有效性为广泛的媒体应用，包括图形和成像[欧文等人。 2000。该StreamC/ KernelC编程环境提供了一个抽象，它允许程序员地图应用的想象处理器

[2002马特森。拉布隆特等。 [2004]研究了成效作为GPU的流处理器通过评估的一个流式虚拟机的性能映射到图形硬件。在提出的编程模型本文可以很容易地被编译为他们的虚拟机。

**2.2**图形硬件编程

现代可编程图形加速器等ATI X800XT和NVIDIA GeForce 6800(ATI 2004 b;英伟达2004]功能可编程顶点和片段处理器。每个处理器执行用户指定程序组成的4路SIMD着色器程序指令(Lindholm et al . 2001年)。这些指令包括标准的数学操作,如3 -或4组件点产品,texture-fetch指令,和一些有所抱怨指令。GPU的基本执行模型如图1所示。对于每个顶点处理或片段,图形硬件图形原语在只读输入的地方寄存器。材质是执行和结果输出寄存器。在执行期间,材质可以访问大量的临时寄存器以及主机应用程序设定的常量。珀塞尔等。[2002]描述GPU可以考虑一个流处理器中执行内核,写片段或顶点着色器,在溪流中存储的数据几何和纹理。可以编写内核使用一个品种高层,c语言,如Cg HLSL,GLslang。然而,即使有这些语言,应用程序还必须执行显式图形API调用来组织数据流和调用内核。例如,流管理是由程序员,要求数据要手动挤在纹理和转移从硬件。内核调用需要加载和绑定的渲染着色程序和几何学。因此,计算并不表示为一组内核的作用于流,而是作为一个序列阴影的基本图形的操作。即使对那些精通图形编程,表达算法以这种方式可以是一个艰巨的任务。这些语言也无法虚拟化的约束底层硬件。例如,流元素限于原生地支持浮动,float2 float3,float4类型,而不是允许userdefined更加复杂结构。此外,程序员必须始终请注意材质指令等硬件的限制计数、着色器输出的数量和结构尺寸。在那里一些工作在缓解一些材质语言这些约束。成龙等。[2002]提供了一个算法细分大着色器自动分解成更小的着色器绕过着色器长度和输入约束,但做的不探索多个着色器输出。迈克尔et al .(2002;2004]开发了Sh,一个系统,允许将着色器定义和使用元编程语言执行基于c++。上海的主要目的是作为一个阴影系统,虽然它已被证明执行其他类型的计算。然而,它并不提供一些基本的操作中常见的通用计算,如收集和减少。一般来说,今天编写的代码来执行计算在gpu开发在一个高度graphics-centric环境,对于那些试图映射其他带来困难应用图形硬件上。

**3.Brook**流编程模式

Brook是一个发达的语言流处理器比如斯坦福大学的梅里马克河流超级计算机[玩弄et al . 2003],想象处理器(·卡帕西的文章et al . 2002年),但奥斯汀旅行处理器(Sankaralingam et al . 2003年),和麻省理工生处理器(泰勒et al . 2002年)。我们有Brook适应图形硬件的功能,只会讨论Brook在GPU中架构在这篇文章中。

**3.1**流

流是一组数据,可以在操作并行执行。流与后面的语法声明类似于数组,即浮动s < 10 5 > 2 -表示维的花车。每个流组成元素。在这个例子中,由50年代是一个流元素类型的浮动。流是指它的形状维度。在本例中,10 s形状的流通过5。然而,流类似于C数组访问权流数据仅限于内核(在下面描述)streamRead和streamWrite运营商,转移和内存之间的数据流。流可以包含元素类型的浮动,Cg向量类型,如float2 float3 float4,和结构这些原生类型组成。例如,一个流射线可以被定义为:

typedef struct ray\_t {

float3 o;

float3 d;

float tmax;

} Ray;

Ray r<100>;

支持用户定义的内存类型,尽管中常见通用的语言,是没有找到在今天的一个特色图形api。Brook为用户提供了方便复杂的数据结构和编译时类型检查。

**3.2**内核

Brook内核是特殊函数,指定的内核关键字,操作流。调用一个内核流执行隐式循环遍历的元素流,调用内核为每个元素的身体。内核是一个例子所示

kernel void saxpy (float a, float4 x<>, float4 y<>,

out float4 result<>) {

result = a\*x + y;

}

void main (void) {

float a;

float4 X[100], Y[100], Result[100];

float4 x<100>, y<100>, result<100>;

... initialize a, X, Y ...

streamRead(x, X); // copy data from mem to stream

streamRead(y, Y);

saxpy(a, x, y, result); // execute kernel on all elements

streamWrite(result, Result); // copy data from stream to mem

}

内核接受几种类型的参数:•输入流,包含了内核的只读数据处理。•输出流,指定的关键字,存储内核计算的结果。Brook强加没有限制输出流内核的数量可能有。•收集流,C数组指定的语法([]数组):收集流允许任意的索引检索元素。在内核中,元素获取或“聚集”,通过数组索引操作符。数组[我]。像常规的输入流,收集流是只读的。•所有non-stream参数是只读常量。如果一个内核称为不同的输入和输出流形状,溪隐式地调整每个输入流输出的形状相匹配。这是通过重复(123 - 111222333)或大步(13579 - 13579)在每个维度的元素。一定的限制是允许dataparallel放在内核执行。内存访问仅限于读收集Brook,类似于纹理获取。的操作可能引入流元素之间的副作用,如写静态或全局变量,不允许内核。流允许输入和输出参数相同的内核(就地计算)提供也用作收集流在内核中。一个示例内核计算ray-triangle的十字路口如下所示。

kernel void krnIntersectTriangle(Ray ray<>, Triangle tris[],

RayState oldraystate<>,

GridTrilist trilist[],

out Hit candidatehit<>) {

float idx, det, inv\_det;

float3 edge1, edge2, pvec, tvec, qvec;

if(oldraystate.state.y > 0) {

idx = trilist[oldraystate.state.w].trinum;

edge1 = tris[idx].v1 - tris[idx].v0;

edge2 = tris[idx].v2 - tris[idx].v0;

pvec = cross(ray.d, edge2);

det = dot(edge1, pvec);

inv\_det = 1.0f/det;

tvec = ray.o - tris[idx].v0;

candidatehit.data.y = dot( tvec, pvec ) \* inv\_det;

qvec = cross( tvec, edge1 );

candidatehit.data.z = dot( ray.d, qvec ) \* inv\_det;

candidatehit.data.x = dot( edge2, qvec ) \* inv\_det;

candidatehit.data.w = idx;

} else {

candidatehit.data = float4(0,0,0,-1);

}

}

Brook迫使程序员来区分数据流到内核作为输入流,这是使用数组访问收集的内核。这种区别允许不同的系统来管理这些(数据)流。输入流元素规律但访问身体不重用,因为每个内核调用操作吗在一个不同的元素。收集流可能会访问随机,和元素可能被重用。正如珀塞尔et al。(2002)观察到,今天的图形硬件没有这两种内存存取类型之间的区别。作为一个结果,输入流数据可以污染传统的缓存和惩罚在收集操作位置。使用内核的区别流编程向量编程。内核执行任意函数评价而向量运算符包括简单的数学操作。向量操作总是需要临时变量读取和写入到一个大向量寄存器文件。在相反,内核捕捉额外的位置通过存储临时变量在当地注册存储。通过减少带宽主内存,因为只有算术强度增加内核计算的最终结果写回内存。

**3.3**简化

而内核提供一种机制应用一个函数一组数据,减少提供一种方式来表述数据并行处理方法计算一个值从一组记录。例子还原操作包括算术求和,计算最多,矩阵乘积。为了执行减少平行,我们需要减少操作联想:◦◦(b)c =◦◦(b c)。这使得系统评估最适合哪个订单的减少底层架构。接受一个输入流并减少生产输出一个较小的相同类型的流,或者一个圈价值。为减少指定的输出降低关键字。阅读和写作的减少允许参数时,计算的减少两个值。如果减少输出参数是一个元素,它将得到所有的输入流的减少值元素。如果参数是一个流的形状输入和输出流用于确定有多少相邻的元素输入减少生产输出的每个元素。下面的例子演示了如何停工检修减少可用于执行矩阵向量乘法y = Ax

kernel void mul (float a<>, float b<>, out float c<>) {

c = a \* b;

}

reduce void sum (float a<>, reduce float r<>) {

r += a;

}

float A<50,50>;

float x<1,50>;

float T<50,50>;

float y<50,1>;

...

mul(A,x,T);

sum(T,y);

In this example, we first multiply A by x with the mul kernel.

Since x is smaller than T in the first dimension, the elements

of x are repeated in that dimension to create a matrix of

equal size of T. The sum reduction then reduces rows of T

because of the difference in size of the second dimension of

T and y.

**3.4**额外的语言特性

在本节中,我们提出额外的Brook语言特性应该提到,但不会讨论吗进一步本文。读者更感兴趣细节是鼓励阅读。

**4.**在图形硬件上执行

Brook编译和运行时系统地图Brook语言上现有的可编程GPU api。该系统包含两个组件:source-to-source brcc编译器,和Brook运行时(BRT),图书馆提供运行时支持内核执行。编译器基于cTool[Flisakowski 2004],C一个开源解析器,这是修改以支持Brook语言原语。编译器将Brook内核映射到Cg的着色器由供应商提供的材质时编译器翻译成GPU组装。此外,brcc发出c++代码使用BRT来调用内核。附录A提供了一个前后一个编译内核的例子。BRT是与体系结构无关的软件层提供了一个公共接口的每个后端支持由编译器。Brook目前支持三后端;一个OpenGL和举后端和一个参考CPU实现。创建一个跨平台实现提供了三个主要的好处。首先,我们展示语言的可移植性,以允许用户选择最好的后端硬件。其次,我们可以比较不同的图形api GPU的性能计算。最后,我们可以优化特定于api的功能,也有如OpenGL的0到n纹理处理的支持和举的直接纹理渲染功能。以下部分描述如何Brook地图流、内核和减少语言原语上GPU。

**4.1**流

Brook将流表示为浮点纹理图形硬件。通过这种表示,streamRead和streamWrite运营商上传和下载纹理数据,收集操作表示为依赖纹理读取,隐式重复和跨运营商实现与纹理采样。当前图形api,但是,只提供浮动,float2 float3和float4纹理格式。支持用户定义的流结构、BRT商店每个成员的结构不同硬件结构。许多应用程序作者可能希望可视化结果一条Brook的计算。BRT提供了c++接口它允许用户绑定的溪水流作为本地图形API纹理可以交互式地呈现在传统的图形应用程序。这个选项需要Brook提供流在一个固定的,记录结构布局。默认情况下,流存储作为一个纹理以相同的维数为流形状。更大的挑战是硬件带来的局限性在纹理大小和形状。浮点纹理是有限的两个维度,4096 \* 4096的最大大小英伟达和2048到2048年在ATI的硬件。如果我们直接纹理映射流形状形状,然后Brook项目不能创建流超过两个维度或1 d流超过2048或4096的元素。为了解决这个限制,brcc提供了编译器选项包装结构的流数据跨多个行。这允许arbitrary-sized流假设适合在一个结构的元素数量。为了访问一个元素,它的位置在流,brcc插入代码转换和流之间的位置相应的纹理坐标。Cg代码如下所示用于stream-to-texture地址转换,允许吗流的四维度包含尽可能多的元素texel最大尺寸的2 d纹理。

float2 \_\_calculatetexpos( float4 streamIndex,

float4 linearizeConst, float2 reshapeConst ) {

float linearIndex = dot( streamIndex, linearizeConst );

float texX = frac( linearIndex );

float texY = linearIndex - texX;

return float2( texX, texY ) \* reshapeConst;

}

我们的地址转换实现的限制在图形硬件精度可用。在计算流的纹理坐标位置,我们转换按比例缩小的整数索引的位置。如果任何范围指数超过最大可表示的连续的整数显卡的浮点格式(16777216NVIDIA的s23e8格式,131072 ATI的24位s16e7格式)然后没有足够的精度独一无二的地址正确的元素。例如,我们的实现有效地增加了最大1 d流大小便携式Brook计划从2048年到131072年的元素ATI的硬件上。最终,这些限制在纹理寻址点需要一个更一般的内存寻址模型在未来的gpu。

**4.2**内核

流数据存储在纹理,使用GPU的Brook片段处理器执行一个核函数流元素。brcc编译一个内核的身体Cg材质。流从纹理参数初始化,收集操作取代纹理获取,和non-stream参数通过常量寄存器。然后应用NVIDIA或微软材质编译器生成的Cg代码产生GPU组装。执行内核,单个四含BRT的问题相同数量的碎片在输出元素流。内核输出到当前呈现渲染目标。举后端直接呈现包含输出流的纹理数据。然而,OpenGL没有提供一个轻量级的绑定机制纹理渲染目标。OpenGL Pbuffers提供这个功能,不过,Bolz et al。[2003]发现,与Pbuffers可以切换渲染目标的性能受到很大影响。因此,我们的OpenGL后端呈现一个浮点pbuf和拷贝结果输出流的纹理。proposedProgram指令MFLOPS放缓texld算术3611年Mat4Mult4 8日16Mat4Mult1 20 1683 53%Cloth4 54 5086Cloth1 12 102 2666 47%表1:这个表展示了性能成本分离内核包含比支持输出由硬件。包括指令数和观察矩阵乘法和布内核的性能4-output硬件和1输出的硬件上执行使用NVIDIA举后端。经济放缓是单式输出的相对性能下降实现。Superbuffer规范[珀西2003],它允许直接在OpenGL纹理渲染功能,应该缓解这一限制。内核映射到片段着色器的任务是复杂的由有限数量的着色器输出中可用今天的硬件。当内核使用更多的输出流比支持的硬件(或使用一个输出流的结构类型),brcc分裂内核为多个为了计算的所有输出。对于每一个传球,编译器生成一个完整的内核代码的副本,但只有分配内核输出着色器输出的一个子集。我们利用积极的死代码消除由今天的着色器删除任何编译器计算这并不有助于写输出在通过。为了测试我们的pass-splitting技术的有效性,我们应用两个内核:Mat4Mult,繁殖两个流4 x4矩阵,生产一个4 x4矩阵(4 float4s)输出流;模拟和布料particle-based布与弹簧约束、生产更新粒子位置和速度。我们测试了两个版本每个内核。Mat4Mult4 Cloth4编译与硬件支持4 float4输出,只需要一个通过完成。Mat4Mult1和Cloth1编译的硬件只有一个输出,迫使运行时为每个输出生成单独的着色器。如表1所示,该技术的有效性取决于内核之间的共享计算量输出。Mat4Mult内核的计算干净地分离为每个输出,着色器编译器正确地识别,输出矩阵的每一行都可以单独计算。因此,总数计算所需的算术运算的结果不4-output和1输出版本之间的不同。然而,结构加载的总数增加因为每个传递必须加载所有16个元素之一的输入矩阵。布的内核,位置和速度输出的内核代码(一个力计算)必须重复如果输出计算在不同的着色器。因此,有近两倍许多指令4-output 1输出版本的版本。两个应用程序与multipleoutput表现得更好支持,有效地证明了我们的系统利用输出硬件,而透明的可伸缩性对于系统仅支持。

**4.3**简化

当前图形硬件没有原生支持为减少。通过多通道BRT实现减少方法类似于克鲁格和科·[2003]。减少在执行O(log n),其中n是比大小的输入和输出流。对于每一个传球,减少操作读取8相邻流元素,和输出的减少值。因为每个通过生产2 - 8值较少,Brooklineartime减排计算。每个减少通过特定的大小是一个函数的大小减少流和内核。我们已经为基准计算2的总和20 float4利用2.4和元素。分别79毫秒我们的英伟达和ATI举后端,4.1和1.3毫秒OpenGL后端。一个优化CPU实现执行这个减少14.6毫秒。举和之间的性能差异OpenGL实现主要是由于复制的成本从输出结果pbuf纹理,所述以上。减少我们的多通道实现,GPU必须访问内存明显多于一个优化CPU实现减少流。如果图形硬件提供了持久的寄存器可以积累结果跨多个片段,我们可以减少流在一个传递一个值。我们模拟了图形硬件的性能与这个理论能力通过测量的时间来执行一个内核读取单个流元素,将其添加到一个常数和问题杀死一个片段,以防止任何写操作。基准测试这个内核举在相同流如上理论产量减少的时候。41,只要毫秒分别在NVIDIA和ATI的硬件。

**4.4**评估和程序

我们现在检查几个科学应用程序的性能Brook在gpu上使用。对于每个测试,我们评估Brook使用OpenGL和举一个后端ATI Radeon X800 XT铂运行4.4版本驱动程序和pre-release1 NVIDIA GeForce 6800版本运行60.80驱动程序,运行Windows 7。为我们的CPU比较,我们使用3 GHz的英特尔奔腾4处理器英特尔芯片组875便士运行Windows 7,除非另有指出。

**5.1**程序

我们在Brook实现各种算法。的应用程序是有三个原因:代表不同类型的算法中执行数字应用程序;他们是重要的算法广泛的在计算机图形学和一般科学计算;优化CPU或基于gpu实现与我们的实现可以使性能比较在Brook。布拉斯特区SAXPY SGEMV例程:BLAS(基本线性代数)子程序库是一个低级的集合线性代数子例程(劳森et al . 1979年)。SAXPY执行向量规模和笔操作,y = ax + y,在x和y是向量是一个标量。SGEMV是一个单精度稠密矩阵向量产品了缩放向量添加、y =αAx +βy x,y是一个向量,是一个矩阵和α,β是标量。矩阵向量运算许多数值应用程序的关键,doubleprecision吗的变体SAXPY核心计算内核使用的LINPACK基准则[2004]世界上排名最高的超级计算机。我们比较性能与优化的商业英特尔数学内核库[英特尔2004]SAXPY和阿特拉斯布拉斯特区图书馆SGEMV[惠利et al . 2001],我们可以最快的公共CPU执行来定位。参考GPU的比较,我们实现了一个hand-optimized SAXPY举的版本和一个优化OpenGL SGEMV实现。对于这些测试,我们使用向量或矩阵的大小10242。部分执行的2 d版本Perona一起和马利克[1990]非线性、diffusion-based播种,区域增长算法,提出了Sherbondy et al。[2003],在2048年到2048年的形象。分割被广泛用于医疗图像处理和数字合成。我们比较我们的Brook与手工编码的OpenGL实现实现CPU上执行我们的测试系统。每一个进化迭代分割的内核需要32浮点运算、读取10浮点数作为输入,并写道2浮动作为输出。优化CPU实现特别适应于执行是最大限度地缓存友好吗计算的奔腾4。傅里叶变换FFT:我们的应用程序执行一个2 dCooley-Tukey快速傅里叶变换(FFT)[1965]4频道1024年到1024年复杂的信号。快速傅里叶变换算法在许多图形应用程序是很重要的,framebuffer图像后处理等,搜寻地外文明计划和科学应用等项目(沙利文et al . 1997年)。我们的实现使用三个内核:水平和垂直一维FFT,每个10次,有点逆转内核调用一次。水平和垂直每个执行5浮点FFT内核操作/输出值。总浮点运算执行,基于benchFFT[冻结器和约翰逊2003]项目,等于5·w·h··log2渠道(w·h)。基准Brook反对竞争GPU的算法,我们比较我们的结果和自定义OpenGL实现可以从ATI(ATI 2004)。比较对大量的CPU,我们基准优化FFTW-3软件库编译与英特尔(Intel c++编译器2003]。雷是一个简化版的GPU射线示踪剂在珀塞尔等。[2002]。这个应用程序包含三个内核,雷设置,ray-triangle十字路口(部分3所示),和阴影。对于CPU比较,我们比较瓦尔德的发表结果的[2004]hand-optimized组装可以达到每秒100射线在吗奔腾4 3.0 ghz处理器。我们观察到GPU实现表现良好针对CPU同行。Brook举ATI版本的SAXPY和段约7和执行4.7倍的速度比等效的CPU实现。SAXPY说明,即使是一个内核执行只有一个疯狂的指令能够超CPU额外的内部GPU上的可用带宽。FFT是我们穷相对于执行应用程序CPU。Brook只能实现。99的速度CPU版本。FFTW块的内存访问非常有效地利用处理器缓存。(没有这个优化,从1224年有效CPU MFLOPS滴204年)。尽管阻塞,Brook能够大致匹配CPU的性能。我们还可以举的相对性能进行比较和OpenGL后端。举是80%以内的手工编码的GPU性能的实现。OpenGL后端然而高效相比要少得多参考实现。这在很大程度上是由于需要复制OpenGL的输出数据pbuf纹理(参见4.2)。这是特别明显与SGEMV测试必须执行一个多通道还原操作。手编码版本使用应用具体知识来避免该副本。我们观察到,因为这些应用程序,ATI一般执行比NVIDIA。我们认为这可能是由于更高的浮点纹理ATI的带宽。我们观察1.2 Gfloats /秒的浮点结构带宽英伟达ATI的4.5 Gfloats /秒相比,峰值时,可观察到的计算性能支持NVIDIA与400亿年每秒繁殖与ATI 330亿。在某些情况下,我们的Brook实现超越参考GPU实现。NVIDIA的FFT结果,Brook执行比OpenGL的引用FFT ATI提供的代码。我们也比·莫兰和天使的NVIDIA[2003]的特定实现同样的利润。与SGEMV类似的趋势显示,举溪实现优于手工编码OpenGL。由于我们假定这些差异是相对的举和OpenGL驱动程序的性能。这些应用程序提供角度上的性能使用GPU通用计算的Brook。然而,性能数字不包括streamRead和streamWrite操作转移的成本最初的和最后的数据可以的GPU严重影响整个应用程序的性能。下一节将探讨这种开销是如何影响性能和调查的条件使用GPU总体性能超过了CPU。

**5.2** 模块化性能

许多Brook应用程序包含的总体结构复制数据到GPU streamRead,表演一个内核调用序列,并复制结果返回与streamWrite CPU。执行相同的计算在CPU上不需要这些额外的数据传输操作。考虑到成本的转移会影响GPU是否会超越一个特定的CPU算法。为了研究这种效果,我们认为一个程序下载n记录的GPU,执行内核所有n记录,并读取返回结果。的时间来执行这个操作在GPU和CPU是:Tgpu = n(Tr + Kgpu)Tcpu = nKcpuTgpu和Tcpu GPU上的运行时间和CPU分别Tr是相关的传输时间下载和阅读单个记录,和Kgpu和Kcpu所需的时间来执行一个给定的内核在一个记录。这个简单的执行时间模型假定在高峰,内核执行时间和数据传输处理速度是线性元素的总数/转移。GPU比CPU时Tgpu < Tcpu。使用这种关系,我们可以表明:Tr < Kcpu−Kgpu如图所示,这个关系,执行的性能优势内核在GPU(Kcpu−Kgpu)必须是足够的隐藏数据传输成本(Tr)。从这个分析,我们可以做一些基本的结论关于算法的类型将受益于执行在GPU上。首先,的相对性能两个平台显然是重要的。加速定义随着时间的执行内核的CPU与GPU,年代≡Kcpu / Kgpu。越大加速对于一个给定的内核,越有可能将在GPU执行得更好。其次,一个算法执行大量的计算相对于可能传输数据所花费的时间由计算时间。这种关系是计算强度,γ≡Kgpu / Tr的算法。一个算法的计算强度越高,适合GPU的计算。用到上面的关系,我们可以推导出的关系加速和计算强度之间的关系。γ>1s−1计算强度的概念类似于算术强度,由玩弄等。[2003]的数字浮点操作每个字读的内核。它认为整个计算强度不同在设备上执行一个算法与成本数据集的转移成本的设备。非常相关的GPU计算强度一般不会在相同的地址空间中运行主机处理器。对我们的成本模型中,我们假设Kgpu的参数和Tr是独立于流元素的数量n。在现实中,我们发现这通常不是这样的短流。gpu更有效地传输数据在中期大大小的量。更重要的是,有开销发行一个内核。每个内核调用产生一定的固定数量的CPU时间设置并发出GPU上的内核。与多个back-toback内核调用,这个设置成本在CPU上可以重叠与内核执行的GPU。对内核的操作大的溪流,GPU将成为限制因素。然而,内核的操作短流,CPU可能无法保持GPU内核问题足够快吗忙了。图4显示了1000次迭代的平均执行时间合成的内核与各自的运行时。正如所料,这两个运行时显示了膝盖,发行并运行内核转换从有限的CPU设置内核执行有限的GPU。为我们的合成应用程序将执行43疯狂的指令,ATI运行时穿过膝盖以上时执行750 k和2 m浮点运算和英伟达十字架在650 k OpenGL和浮点操作举。我们的分析表明,有两个关键的应用程序GPU的性能所必需的有效利用。首先,为了超越CPU,大量的工作执行必须克服是一个函数的转移成本的计算算法和强度硬件的加速。第二,完成的工作量每个内核调用应该足够大,以隐藏设置内核所需的成本问题。我们预计,尽管与新硬件的具体数字可能会有所不同,计算强度,加速,将内核开销继续决定有效的GPU的利用率

**6.**讨论

我们的计算表明强度分析读/写带宽为建立类型是很重要的在GPU中表现出色的应用程序。理想情况下,未来gpu将执行异步读写操作与计算。这个解决方案改变了GPU执行时间是马克斯Tr和Kgpu更良好的表达。也有可能未来的流硬件将共享相同的内存作为CPU,消除数据传输的需要。也可以把虚拟化的硬件约束GPU接近一个流处理器。Brook虚拟化两个流计算的关键方面,这个号码内核输出和流的尺寸和大小。多个输出编译可以通过搜索改进空间划分内核计算的可行方法产生所需的输出,类似于一个泛化RDS算法提出的成龙等。[2002]。同样的算法将虚拟化输入参数的数量以及指令总数。我们已经开始将这种算法为brcc可喜的成果。此外,Brook应该考虑的几个特征为未来流GPU硬件。变量输出有条件地允许内核输出零个或多个数据每个输入。变量输出是有用的应用程序展示数据放大,如镶嵌,以及应用程序选择操作部分的输入数据。我们目前正在研究这些应用程序和添加通过多通道算法能力到Brook。它是可以想见,未来可以扩展到包括硬件此功能使全新的类流媒体的应用程序。其次,流计算gpu将极大受益从最近的顶点纹理和浮点混合操作。与这些功能,我们可以实现Brook平行间接的读-修改-写操作符,ScatterOp GatherOp,这是用于处理和构建数据结构存储在流。一个特性,gpu的支持我们想让Brook是谓词内核计算。例如,珀塞尔等。[2002]能够加速计算通过使用GPU的深度吗测试,以防止一些内核操作的执行。总之,Brook编程环境提供一个简单但有效的工具对gpu计算。Brookgpu已经发布的一个开源项目(Brook2004),我们希望这一努力将使它应用程序开发人员更容易捕获性能流计算的好处的GPU图形社区和超越。通过提供简单的访问在消费者图形硬件的计算能力,流计算有可能重新定义的GPU不仅仅是一个渲染引擎,但其原理计算引擎的电脑。

**7.**确认

我们想感谢ATI和英伟达的提供他们的硬件,特别是马克·西格尔和尼克Triantos。蒂姆·珀塞尔提供我们的Brook射线跟踪的实现。比尔马克提供宝贵的反馈我们提交。我们还要感谢以下人他们帮助Brook的设计语言:比尔磨磨蹭蹭的,快弗朗索瓦·玛坦Erez蒂姆•奈特Jayanth Gummaraju巴拉,Eric Darve,诺阿Fatica来自斯坦福大学;来自美国宇航局的蒂姆·巴斯和艾伦·雷,彼得•马特森肯•麦肯齐埃里克•Schweitz查理•加勒特瓦什从水库,和理查德·利特维诺夫市Lethin实验室。Brook的GPU实现支持美国国防部高级研究计划局。ATI提供了额外的支持,IBM,NVIDIA和索尼。Brook的编程语言与支持部门了吗的能源,国家核安全管理局,在子囊联盟计划(合同LLL-B341491),美国国防部高级研究计划局智能记忆项目(合同mda904 - 98 r - s855),和美国国防部高级研究计划局多形态计算架构项目(合同F29601 -00-2-0085)。由NVIDIA提供额外的支持奖学金,Rambus斯坦福大学研究生奖学金,和斯坦福大学工程学院的奖学金项目。

参考文献

[1] D. Comaniciu, V. Ramesh, and P. Meer, Kernel-based object tracking. IEEE Transactions on Pattern Analysis and Machine Intelligence, 25(5): 564-577, 2003.

[2] Bolz, J., Farmer, I., Grinspun, E., and Schro ̈der, P.2013. Sparse matrix solvers on the GPU: conjugate gradi- ents and multigrid. ACM Trans. Graph. 22, 3, 917–924.

[3] Bove, V., and Watlington, J. 1995. Cheops: A reconfig- urable data-flow system for video processing. IEEE Trans. on Circuits and Systems for Video Technology (April), 140– 149.

[4] Brook, 2014. Brook project web page. http://brook.sourceforge.net

[5] Buck, I. 2014. Brook specification v.0.2. Tech. Rep. CSTR 2003-04 10/31/03 12/5/03, Stanford University.

[6] Carr, N. A., Hall, J. D., and Hart, J. C. 2012. The Ray Engine. In Proceedings of Graphics hardware, Eurographics Association, 37–46.

[7] Chan, E., Ng, R., Sen, P., Proudfoot, K., and Han- rahan, P. 2012. Efficient partitioning of fragment shaders for multipass rendering on programmable graphics hard- ware. In Proceedings of Graphics hardware, Eurographics Association, 69–78.

[8] Cooley, J. W., and Tukey, J. W. 1965. An algorithm for the machine calculation of complex Fourier series. Mathe- matics of Computation 19 (April), 297–301.

[9] Dally, W. J., Hanrahan, P., Erez, M., Knight, T. J., Labont, F., Ahn, J.-H., Jayasena, N., Kapasi, U. J., Das, A., Gummaraju, J., and Buck, I. 2013. Merrimac: Supercomputing with Streams. In Proceedings of SC2003, ACM Press.

[10] Dongarra, J. 2014. Performance of various computers using standard linear equations software. Tech. Rep. CS- 89-85, University of Tennessee, Knoxville TN.

[11] England, N. 1986. A graphics system architecture for interactive application-specific display functions. In IEEE CGA, 60–70.

[12] Flisakowski, S., 2014. cTool library. http://ctool.sourceforge.net.

Frigo, M., and Johnson, S. G., 2013. benchFFT home page. http://www.fftw.org/benchfft.