高技术通讯 2022 年 第32 卷 第5 期:484-492

doi:10.3772/j.issn.1002-0470.2022.05.005

GPU-Hi:GPU RTL 平台实现及效率分析①

张立志②[*,**,***] 赵士彭[*,**,***] 章隆兵[*,**,***]

([*]计算机体系结构国家重点实验室(中国科学院计算技术研究所) 北京 100190)

([**]中国科学院计算技术研究所 北京 100190)

([***]中国科学院大学 北京 100049)

摘 要 实现了寄存器传输级(RTL)图形处理器(GPU)研究平台——GPU-Hi。GPU-Hi 支持 OpenGL 2.0 API,支持统一着色器渲染架构,使用专用集成电路(ASIC)完成图形流水线的固定功能算法,使用单指令多线程(SIMT)架构流处理器完成图形流水线的可编程着色器模块。在使用 28 nm 工艺的情况下,该平台的物理设计面积为 7.9 μm^2。使用 glmark2 的测试集作为性能测试程序,完成了该平台的功能正确性验证,同时使用该测试集研究了 3D 图形应用的计算特性,并进行了 GPU 微结构级的性能分析。测试结果表明,图形应用的光栅化任务与像素着色任务不随图形应用分辨率等比例增大;同时 GPU 硬件的光栅化模块性能受着色程序处理能力与显存访问能力的影响。本平台的实现对 GPU RTL 平台的研究发展有重要的借鉴价值,本文中得到的结论对 GPU 性能优化具有重要参考意义,有力支持了 GPU 硬件研究的发展。

关键词 图形处理器(GPU);性能分析;glmark2;流处理器集群

0 引 言

随着个人电脑与智能手机的普及[1],人们对细腻逼真的3D 图形绘制需求愈发强烈。图形处理器(graphics processing unit, GPU)作为处理3D 图形任务的专用芯片,其研究与发展对提升图形应用绘制效果有重要意义。

现有的典型 GPU 架构有 Fermi 架构[2]与 graphics cone next (GCN)架构[3]。Fermi 架构是 Nvidia 公司于2008 年提出,该架构将顶点着色处理模块与像素着色处理模块合并为基于流处理器的统一渲染模块,统一渲染架构的流处理器与专用集成电路(application-specific integrated circuit, ASIC)实现的固定图形算法[4-7]处理电路共同实现整个图形处理流水线。GCN 架构由超威半导体公司于 2012 年提出,该架构将原本使用的 VLIW + SIMD 改进为 SIMT 架构。但因为商业原因,Nvidia 与 AMD 公司并没有披露这两款架构的设计细节,导致研究者无法直接使用这些架构进行 GPU 研究。

学术界对 GPU 的研究主要集中在通用计算领域,例如 gpgpu-sim[8]、gem5-gpu[9-10]、multi2sim[11]、MIAOW[12]、Accel-Sim[13],通用计算只涉及到 GPU 的流处理器模块,并不涉及 ASIC 模块,这些研究对 GPU 硬件与应用行为的分析更多地集中在流处理器与 cache 模块,并没有对图形应用基础的图形算法模块进行研究分析。

学术界完整实现 GPU 流处理器模块与 ASIC 固定图形算法模块的典型 GPU 架构有 ATTILA[14]与 Emerald[15]。ATTILA 是2006 年提出的一款 GPU 架

① 国家自然科学基金(61521092)和中国科学院重点部署项目(ZDRW-XH-2017-1)资助。

② 男,1992 年生,博士生;研究方向:计算机系统结构,GPU 体系结构;联系人,E-mail: zhanglizhi@loongson.cn。

(收稿日期:2021-03-12)

构,该架构使用高级语言模拟器实现了完整的周期精确模拟,但其很多设计已经不符合现代 GPU 架构。Emerald 由英属哥伦比亚大学于 2019 年提出,在 gpgpu-sim 模拟器中增加了图形算法模块的实现。但 Emerald 并没有对图形算法模块实现周期精确的模拟。ATTILA 与 Emerald 的另一个不足是缺少 RTL 实现,导致其分析结果不够准确。

统观学术界和工业界现有的 GPU 平台,有的封锁实现方法,有的使用古老的 GPU 架构,有的只使用高级语言模拟。GPU 研究领域缺少一款开放实现方法、使用现代 GPU 架构并且进行寄存器传输级(register-transfer level, RTL)实现的 GPU 研究平台。

本文主要贡献为实现了一个包含完整图形功能的 RTL 级的 GPU 研究平台 GPU-Hi,解决了上述问题。并使用 28 nm 工艺完成物理设计,该平台支持 OpenGL 2.0 框架,使用 ASIC 电路实现固定图形算法模块,使用 SIMT 流处理器实现统一着色渲染模块。在实验模块本文基于该平台对 glmark2[16] 测试集进行了应用行为分析并对 GPU 进行硬件效率分析。结果证明,图形应用程序在进行分辨率提升时,图形任务负载不会等比例增加;硬件光栅化模块的真实性能受到着色程序执行效率与访存效率的影响,单一提升 GPU 部分模块硬件配置,GPU 整体性能并不会明显提升。这些结论对图形应用开发人员与 GPU 结构设计人员有借鉴意义。

本文包含以下几个部分。第 1 节介绍了 3D 图形应用背景与 GPU 架构背景;第 2 节描述了 GPU-Hi 的 RTL 实现方法;第 3 节分析了图形应用计算特性;第 4 节展示了 GPU-Hi 的 RTL 实现的参数,并使用 glmark2 测试集进行图形应用行为分析与 GPU 硬件效率分析;第 5 节提出未来研究工作并对本文进行总结。

1 背景介绍

1.1 3D 图形应用背景

目前主流 3D 图形应用程序主要使用 OpenGL[17] 与 DirectX[18-19]2 个图形库进行实现。OpenGL 图形库是一个面向 2D 与 3D 计算机图形学的跨平台语言的应用程序库,1991 年由 SGI 公司提出,后来由 Khronos 组织进行版本更新与维护。DirectX 图形库由微软公司提出,实现内容与 OpenGL 图形库类似。

这 2 个图形库都为 3D 图形程序定义了一条图形处理流水线,这 2 条流水线处理流程基本相同。图形处理流水线将处理流程分为顶点着色阶段、光栅化阶段、像素着色阶段、后片段处理阶段。顶点着色阶段以顶点数据为处理对象,对每一个顶点数据使用顶点着色程序进行顶点的位置计算与属性计算。光栅化阶段将经过计算的顶点数据组织成以三角形为代表的基本图元,并以这些基本图元为处理对象,将图元转换生成屏幕像素点。像素着色阶段以像素点为基本处理单元,对每一个像素点使用像素着色程序进行像素点的颜色计算。OpenGL 图形库使用高级着色器语言(high-level shading language, HLSL)[20] 实现顶点着色器程序与像素点着色器程序,DirectX 图形库则使用 GLSL 语言[21] 进行实现。后片段处理阶段将经过着色的像素点数据进行深度剔除或透明混合操作,处理过后的像素点最终组成在屏幕中显示的图像。

1.2 GPU 架构背景

GPU 是针对 3D 图形应用程序而设计的应用加速芯片。依据 OpenGL 与 DirectX 图形库所定义的图形处理流水线,同时为增强自身处理能力,GPU 将自身架构设计为统一渲染架构。因为顶点着色阶段与片段着色阶段都使用着色程序进行计算,所以统一渲染架构使用同一块流处理器实现这两阶段的处理。基于这两个阶段所具有的并行特性,流处理器使用单指令多线程(single instruction multiple threads,SIMT)架构。根据光栅化单元与后片段处理阶段的流式数据处理特性,使用 ASIC 电路对其进行实现。

2 GPU-Hi 的 RTL 实现方法

2.1 GPU-Hi 简介

GPU-Hi 使用本文作者所设计的一款 GPU 架构进行 RTL 实现,该 GPU 架构已经通过高级语言模

拟器[22-23]完成了正确性验证。GPU-Hi 支持 OpenGL 2.0 渲染框架,流处理器实现为统一渲染架构,各个处理单元支持灵活扩展。本文介绍了 GPU-Hi 的 RTL 实现方法,并使用 GPU-Hi 进行图形应用行为分析与 GPU 硬件性能分析。

GPU-Hi 包括 5 个部分,即命令处理器(command processor, CP)、全局任务调度器(global task scheduler, GTS)、图形处理集群(graphics processing cluster,GPC)、二级缓存(level 2 cache, L2 Cache)和内存控制器(memory controller,MC)。其中图形处理集群包括 6 个模块,即计算处理引擎(compute engine,CE)、几何处理引擎(geometry engine,GE)、图元处理引擎(primitive engine,PE)、局部任务调度器(local task scheduler,LTS)、流处理器集群(stream processor cluster, SPC)和输出合并单元(output merge unit,OMU)。

GPU-Hi 结构如图 1 所示。

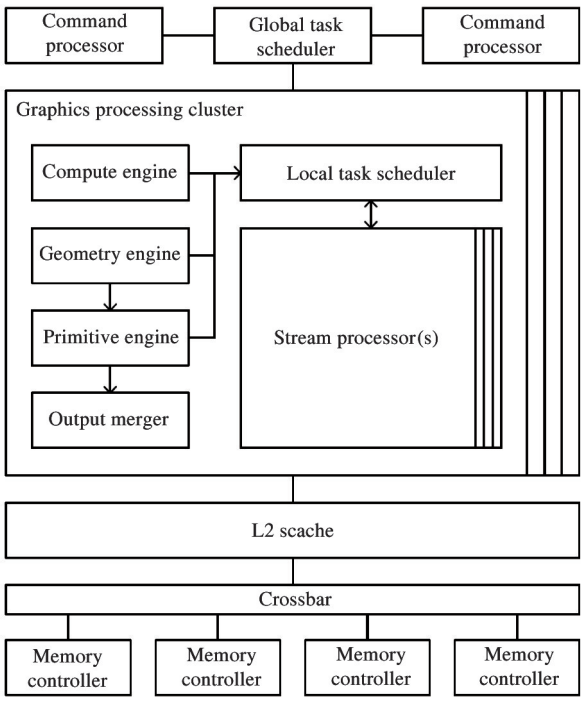

图 1　GPU-Hi 架构图

本节后续部分介绍各模块的具体实现方法。

2.2　命令处理器

命令处理器是 GPU 的整体控制模块,本平台通过在命令处理器模块嵌入一个 GS132 处理器核进行实现。该处理器核解析来自主机中央处理器(central processing unit, CPU)的图形绘制命令与绘制任务参数,针对不同的绘制命令与任务参数配置全局任务调度器和图形处理集群。在 GPU 流水线完成任务的绘制后,命令处理器把绘制结束信息通知给显示输出模块,最终由显示输出模块将结果数据输出在显示器屏幕上。

2.3　全局任务调度器

全局任务调度器从命令处理器接受绘制命令,并根据调度算法将绘制命令分配到对应的图形处理集群中。

2.4　图形处理集群

图形处理集群是 GPU-Hi 的核心处理模块。包括几何处理引擎、图元处理引擎、流处理器集群、局部任务调度器、输出合并单元。

几何处理引擎主要包括顶点索引数据拆分模块与顶点着色任务重定序队列。顶点索引拆分模块使用硬件电路实现,解析命令处理发送来的命令参数,获得顶点数据索引,将顶点数据索引组织为顶点着色任务,并将着色任务发送至局部任务调度器。着色任务包括着色程序的指令地址、数据地址、寄存器信息。顶点着色任务重定序队列由随机存取存储器(random access memory, RAM)实现,保存着色器任务的结果数据,并按照着色器任务生成顺序将结果数据发送到图元处理引擎。

图元处理引擎主要包括固定的光栅化算法模块、像素着色任务生成模块和像素着色任务重定序模块。固定光栅化算法模块实现了 3D 图形光栅化算法。区别于通用 CPU 对数值处理的定点、单精度、双精度数值格式,该模块使用大量定制的运算单元提高运算效率与数值精度。这些定制的运算单元包括单精度数据处理、可变精度数据处理、定点数据处理、多数据融合运算处理等。像素着色器任务生成模块将固定光栅化算法模块生成的像素数据组织为像素着色任务,并为像素着色任务提供指令地址与预赋值的寄存器数据。像素着色任务重定序队列由 RAM 实现,保存顶点着色任务的结果数据,并且按照像素着色任务生成顺序将像素着色任务结果数据发送给输出合并单元。

流处理器集群由多个流处理器模块构成。流处理器模块使用顺序流水线的单指令多线程(SIMT)架构,每周期可以发射一条指令,这条指令可以同时驱动多个线程进行数据的读写与运算。流处理器模块包括融合乘加运算单元、超越函数单元、纹理单元、便笺式存储单元、L1 Cache 单元和寄存器单元。融合乘加单元可以执行融合乘加运算;超越函数单元可以执行正弦、倒数、指数、开平方等运算;纹理单元通过纹理指令来获取纹理数据的地址,并通过 L1 Cache 读取纹理数据,并且对读取得到的纹理数据进行滤波操作;便笺式存储单元可以通过独立寻址的方式为各个运算单元提供数据,减小片外访存压力;L1 Cache 为模块内各运算单元提供数据。寄存器单元是各运算模块最基础的数据存储单元,流处理器设计为所有运算单元无法直接从内存进行数据读写,读取数据时必须先将数据从内存 load 进寄存器,写入数据时必须先将数据写入寄存器。流处理器分别从几何处理引擎与图元处理引擎接收顶点着色任务与像素着色任务,并将处理结果分别返回给几何处理引擎与图元处理引擎。

局部任务调度器用于保存流处理器模块的资源分配与占用情况,并根据资源占用情况将接收到的顶点着色任务与像素着色任务分配调度至流处理器模块。

输出合并模块主要包括深度测试与透明混合模块。这两个模块使用定制的乘加运算单元实现,将从图元处理引擎接收的像素点数据直接进行计算处理,并输出至片外帧缓冲区,供显示输出模块使用。

2.5 二级缓存模块

片上二级缓存使用一个静态 RAM(static RAM, SRAM)实现,用于处理流处理器集群中的 L1 Cache 的访存请求,使用最近最少使用替换策略,在发生访存地址缺失后,通过一个 crossbar 将缺失的访存请求发送至内存控制器进行处理。

2.6 内存控制器

内存控制器负责与片外显存交互,将 L2 Cache 的 Miss 的访存请求通过总线协议(advanced extensible interface, AXI)发送到片外显存进行数据读写访问。

3 图形应用计算特性分析

如 1.1 节所述,图形处理流水线的各个阶段分别使用不同的数据对象作为最小处理单元,这便导致流水线各部分极易产生工作负载不均衡的问题。在早期 GPU 中,顶点着色阶段与像素着色阶段分别由不同的 GPU 硬件模块进行处理,但因为顶点着色任务与像素着色任务的工作任务负载并不相同,很容易出现顶点着色任务过多或者像素着色任务过多的问题,这种负载不均衡问题严重影响了 GPU 的工作效率。现代 GPU 通过统一渲染架构来解决这种问题,即使用统一的流处理器架构同时处理顶点着色任务与像素着色任务。但统一渲染架构只能解决着色器任务的负载不均衡,无法解决光栅化阶段与着色器任务的负载不均衡问题。本节将对这一问题进行分析。

图形处理工作分为需要顺序执行的 3 个阶段,即顶点着色阶段、光栅化阶段和像素点着色阶段。

顶点着色阶段对当前任务中所有顶点数据执行顶点着色程序,顶点着色任务的工作负载与当前任务中顶点数目与顶点着色程序复杂度呈正相关。

光栅化阶段分为以下几个步骤。步骤 1 将顶点数据组织成为图元数据;步骤 2 将图元数据生成以 4×4 pixel 组成的 frag(如图 2);步骤 3 逐个处理 frag 数据,确定该 frag 是否存在覆盖图元的 quad 数据,并计算这些 quad 与 pixel 的属性数据。步骤 1 以顶点数据为对象,每周期读取一个顶点数据,其工作负载由顶点着色阶段传递来的顶点数目决定;步骤2与步骤3以frag数据为处理对象,每周期处理

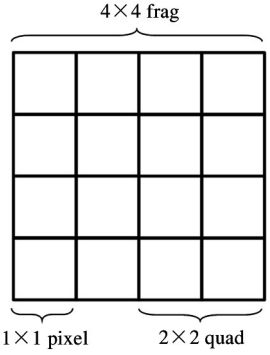

图 2 frag 结构示意

一个 frag 数据,其工作负载由多个因素影响,包括每个顶点的位置信息、绘制任务所配置的屏幕分辨率信息。

像素点着色阶段需要将所有输入来的 quad(如图 2)数据进行像素点着色程序处理。则像素点着色阶段的工作负载依赖于像素着色程序的指令复杂度,同时依赖于光珊化阶段第 3 步生成的 quad 数据数目。

通过对图形处理工作各阶段任务的分析,可以得出结论,图形处理工作在各个阶段,甚至是同一阶段的不同步骤,都有着不同的任务复杂度。同时由于顶点着色阶段、光珊化阶段和像素着色阶段需要顺序执行,任务复杂度不同会导致图形处理负载不均衡。例如,当顶点着色任务负载过重,则会出现光栅化阶段输入数据供给过慢的问题;当像素着色阶段处理过重,则会出现光栅化阶段结果数据输出被堵塞的问题。各个阶段的任务负载又依赖于顶点位置与屏幕分辨率等应用行为,而硬件芯片在设计完成后各模块的处理能力又无法变化,所以各模块的处理效率将极大地受到图形处理各阶段的工作负载影响,即硬件模块的执行效率受到图形应用行为的影响。

4　平台实现及实验分析

本节首先介绍 GPU-Hi 平台的硬件实现情况,而后基于本平台进行了 2 组实验分析。

两组实验分别为:实验 1 为图形应用程序在配置为不同分辨率时,图形应用工作负载的变化;实验 2为流处理器与内存控制器配置为不同规模时,GPU 光栅化模块的性能变化。GPU 作为 3D 图形应用加速器,图形应用的工作负载直接影响 GPU 各硬件模块的计算效率。

4.1　GPU-Hi 物理设计结果

本文使用 28 nm 工艺对该平台进行物理设计,在目标主频为 500 MHz 的情况下,芯片总面积为 7.90 μm^2,主要模块实际面积开销如表 1 所示。

表 1　GPU-Hi 在 28 nm 下面积开销

模块名称	面积/nm^2
命令处理器	339 994
几何处理引擎	948 277
图元处理引擎	914 834
局部任务调度器	45 982
输出合并单元	1 633 777
流处理器集群	3 901 436
内存控制器	109 267

4.2　分辨率对图形任务负载影响

本文选取 glmark2 测试集中 build-horse 进行实验分析。glmark2 是一款 OpenGL 应用程序接口(application programming interface, API)程序测试集。build 测试是一个在顶点着色阶段进行光照渲染的图形程序。

首先通过将分辨率配置为 400 × 300、600 × 450、800 × 600、1000 × 750、1200 × 900,来展示分辨率对图形应用行为的影响。

由于顶点数目属于图形任务固有属性,不随分辨率变化,所以顶点着色阶段的任务负载不随分辨率变化。但图元数据所生成的 frag 数目属于图形任务动态属性,与分辨率相关,即光珊化任务负载会随分辨率变化。

图 3 展示了 build-horse 程序在不同分辨率下光珊化阶段中 frag 数量与 frag _ visi 数量的统计信息。其中 frag 数量是指需要进行像素点遍历处理的 frag 数据数目,代表了光栅化阶段的工作负载情况;frag _ visi 数量是指经过像素点遍历处理后,被标记为可见的 frag 数目,代表了有效的 frag 数据;光珊化有效率是 frag _ visi 数量与 frag 数量的比值。从图中可

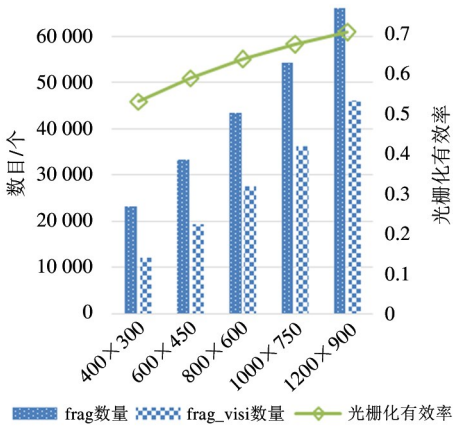

图 3　frag 与 frag _ visi 数据变化

知,虽然 frag 数量与 frag _ visi 数量均随分辨率提高而提高,但两数量的比值,即光栅化有效率最终也随分辨率提高而提高。

图 4 展示了 build-horse 程序在不同分辨率下光栅化阶段中 quad 数量与 pixel 数量的统计信息。其中 quad 数量是指存在可见像素点的 quad 数据数目,代表了像素着色阶段的工作负载;pixel 数量表示真正可见像素点的数量,代表了图形应用真正需要计算的像素数目。由图 2 可知,1 个 quad 数据对应 4 个 pixel 数据,所以像素着色有效率是 pixel 数量与 4 倍 quad 数量的比值。从图中可知,quad 数量与 pixel 数量均随分辨率提高而提高,同时像素着色有效率也随分辨率提高而提高。

由图 3 和图 4 可知,随着分辨率的提高,光栅化有效率与像素着色有效率都随之提高,这是因为在提升分辨率以后,相同屏幕面积下,每一个像素点所占屏幕面积更小,图形应用的 3D 模型建模更加细腻,每一个三角形覆盖的误差区域也就更小。

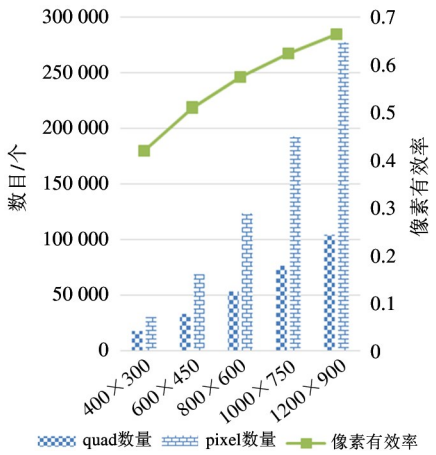

图 4　quad 与 pixel 数据变化

图 5 展示了在分辨率增大后,以 400 × 300 分辨率为基准,各项数据的比例情况,其中代表 pixel 的线与代表像素的线完全重合。图中像素表示当前分辨率下屏幕中的像素数目,例如 400 × 300 分辨率中像素数目为 120 000。frag、quad、pixel 数据意义与图 3、图 4 相同。可以看出,在分辨率增大后,所有数据量都随之增大,但各个数据增大的比例并不相同。代表光栅化任务负载的 frag 数据小于代表像素着色任务负载的 quad 数据变化趋势,同时两者变化趋势均明显小于像素个数的变化趋势。

对图 3 ~ 图 5 的分析可以得出以下结论:(1)在图形任务分辨率提升时,分辨率模块与像素着色模块的工作有效率均有明显提升;(2)在对图像分辨率进行提高时,即提升画质时,各项工作负载并不会随分辨率的变化而等比例变化;(3)在图形任务分辨率提升时,GPU 各处理模块的任务负载变化比例并不相同,即 GPU 各处理模块会产生负载不均衡问题。

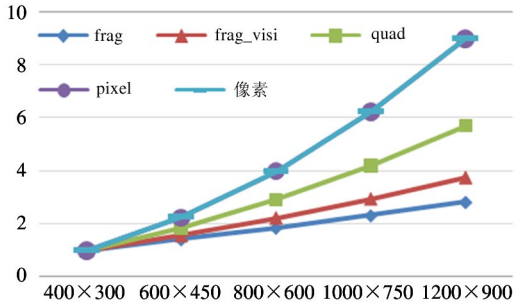

图 5　光栅化各项数据变化比例

4.3　硬件配置对光珊化模块利用率影响

如第 2 节对 GPU 硬件结构描述,光栅化模块负责将图元数据生成为像素点数据,它并不执行着色器指令,也不进行显存的读写操作,指令执行能力与显存读写能力不会直接影响光栅化的效率。但是 GPU 硬件使用顺序的硬件流水线实现图形处理算法,所以在执行图形任务时,GPU 各个硬件模块的执行时间是相同的,即光栅化模块的处理效率会受到流处理器能力与访存能力的间接影响。这里对这一问题进行实验分析。

表 2 展示了应用程序在配置为不同分辨率时,光栅化模块的处理效率变化情况。*total _ time* 列表示 GPU-Hi 的光栅化模块执行当前任务所执行的总周期数,*frag _ num* 表示对应分辨率下光栅化执行的 frag 数据数目,*ra _ rate* 表示光栅化模块处理 frag 的速率,例如第一行表示分辨率为 400 × 300 时,GPU-Hi 的光栅化模块执行了 56 239 周期,处理了 23 395 个 frag 数据,平均每周期处理 0. 415 个 frag 数据。根据表 2 可以看出,在分辨率与光栅化工作负载发生变化时,光栅化速率并没有发生显著改变,并且光栅化效率均不高,距离理想状态下的每周期

处理 1 个 frag 数据的差距很大。这表示光栅化模块并不是 GPU-Hi 的性能瓶颈,处理速率过低的原因可能是因为流处理器集群的处理能力不足,例如顶点着色阶段处理顶点数据的速度过慢,导致光栅化模块接收输入数据的速率太慢;像素着色阶段处理像素点数据过慢,导致光栅化模块生成的 frag 数据无法向后级流水线传递。

表 2　分辨率与光栅化速率统计

分辨率	*total _ time*	*frag _ num*	*ra _ rate*
400 × 300	56 239	23 395	0.415
600 × 450	75 724	33 339	0.440
800 × 600	102 784	43 590	0.424
1000 × 750	136 124	54 423	0.400
1200 × 900	175 028	66 223	0.378

为分析光栅化速率过慢的原因,本文对 GPU 硬件中流处理器集群与 ram 的配置进行更改。流处理器集群配置包括 1 sp 和 2 sp,其中 1 sp 表示 GPU-Hi 中原本的流处理器集群架构,即整个 GPU-Hi 只有一个流处理器模块,所有的顶点着色任务与像素着色任务均使用这一组流处理器进行处理;2 sp 作为对照组,表示使用 2 组流处理器模块来完成顶点着色任务与像素着色任务,可大幅提高着色任务处理能力。ram 配置包括 small ram 和 super ram,其中 small ram 表示 GPU-Hi 中原本的 ram 架构,即整个 GPU-Hi 只有一个 ram 进行显存读写处理,所有需要显存操作的模块均共享该模块;super ram 架构作为对照组,表示为 GPU-Hi 中所有需要进行显存读写的模块均配备独立的访存处理单元,可以大幅提升整个 GPU-Hi 的访存能力。

在实验中,对每个分辨率进行 4 组对照实验:实验组 1 为 1 sp、small ram;对照组 2 为 1 sp、super ram;对照组 3 为 2 sp、small ram;对照组 4 为 2 sp、super ram。4 组对照实验的结果如图 6 和图 7 所示。

由图可知,在 5 种分辨率下,1 sp 与 small ram 均表现出最慢的光栅化速率。

在不更改 ram 而单纯将 1 sp 更改为 2 sp 后,所有分辨率下 frag 处理能力都发生了明显提升,每周期处理数目分别提升了 0.22、0.21、0.17、0.14、0.11,

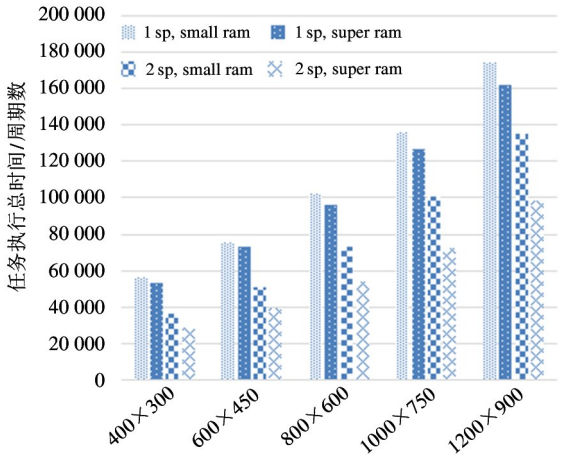

图 6　任务执行总时间随硬件配置变化图

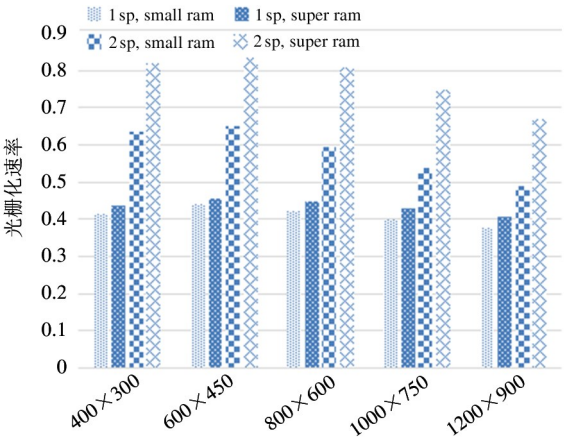

图 7　光栅化速率随硬件配置变化图

提升比例分别为 53.4%、48.3%、40.4%、35.1%、29.1%。这说明 1sp 与 small ram 的配置下,流处理器集群的工作负载过重,其计算能力极大限制了流水线整体的处理效率,并且在分辨率越低的情况下,流处理器集群的计算能力对光栅化的限制越明显,流处理器集群与光栅化模块的负载不均衡问题也越明显。但此时流处理器集群的理论计算能力增加了 100%,光栅化能力虽然有提升,却没有进行等比例增加,最高提升比例仅有 53%,最低提升比例仅仅为 29%,这说明在 2 sp 与 small ram 的配置下,GPU 的瓶颈可能是 ram 的访存能力,而不是流处理器集群的计算能力,下文将增加 ram 配置的对比实验。

在不更改流处理器集群配置而单纯将 small ram 更改为 super ram 后,可以看出 frag 的计算能力并没有明显的提升,5 种分辨率下光栅化每周期性能提升比例仅为 5.1%、3.7%、6.3%、7.1%、7.8%,

这说明在 1 sp 的配置下,small ram 的处理能力已经完全达到了工作负载的需求。

在将 1 sp 更改为 2 sp 同时将 small ram 更改为 super ram 后,相较于 2 sp 与 small ram 的配置,光栅化每周期性能提升比例分别为 28%、28%、35%、38%、37%,可以看出此前 2 sp 在配置 small ram 时,流处理器集群的计算能力的确已经满足工作负载,但 ram 的访存能力无法满足工作负载需求。对比 1 sp与 small ram 的配置,性能提升比例分别达到了 97%、90%、90%、87%、77%。光栅化模块的 frag 处理速度在 600 × 450 分辨率时达到了最高的 0.83 个/s frag 数据,已经接近 1 个/s frag 的理论峰值。

根据以上分析可知,GPU 作为图形应用加速器,其内部各个模块分别使用 ASIC 与 SIMT 等不同架构实现,具有极强的异构架构特性。光栅化模块作为流水线一部分,虽然不直接执行着色器程序,也不进行访存操作,但工作时的实时处理能力受到流处理器处理能力与访存能力的影响。

5 结 论

工业界因为商业问题无法提出一套完整的寄存器传输级 GPU 研究平台;学术界所使用的 GPU 研究平台中,MIAOW RTL 平台没有实现完整的图形算法,而其他研究平台如 ATTILA、Emerald、Teapot,没有通过 RTL 实现。

在这一情况下,本文提出 RTL 级的 GPU-Hi 研究平台,并描述了该平台的硬件实现的方法。使用 GS132 处理器核实现命令控制器,完成 CPU 发送来的任务解析与整条流水线的调度配置;使用 ASIC 电路实现图元处理引擎,完成复杂的 3D 图形光栅化算法;使用 SIMT 架构的流处理器实现流处理器集群,完成顶点着色程序与像素着色程序的执行。

基于前述寄存器传输级 GPU 研究平台通过 glmark2 测试集对 3D 图形应用行为分析及 GPU 硬件负载进行分析,得到以下结论。(1)图形应用的分析表明 3D 图形应用的分辨率变化与分辨率所带来的工作负载变化并不是等比例相关,并且光栅化任务负载与着色器任务负载存在明显的负载不均衡问题;(2)对 GPU 硬件负载的分析表明,虽然光栅化模块并不执行着色器程序指令,也不直接进行显存的读写操作,但其工作效率极大地受到流处理器与访存能力的影响。

随着人工智能的飞速发展,学术界越来越多的关注点投入到了 GPU 在通用计算领域的研究,本文现有平台的实现集中于 GPU 在图形渲染领域的研究,尚未实现通用计算功能。下一步的研究工作需要增加通用计算如 OpenCL 的支持,同时需要进一步完善实验平台的工具链与调试环境,尽快将 GPU-Hi 平台开放给研究人员使用。

参考文献

[1] 胡伟武. 自主 CPU 发展道路及在航天领域应用[J]. 上海航天, 2019(1):1-9

[2] WITTENBRINK C M, KILGARIFF E, PRABHU A. Fermi GF100GPU architecture[J]. *IEEE Micro*, 2011, 31(2): 50-59

[3] MANTOR M. AMD Radeon™ HD 7970 with graphics core next (GCN) architecture[C]// 2012 IEEE Hot Chips 24 Symposium (HCS), Cupertino, USA, 2012: 1-35

[4] OLANO M, GREER T. Triangle scan conversion using 2D homogeneous coordinates[C]// Proceedings of the ACM SIGGRAPH/EURO Graphics Workshop on Graphics Hardware, Interlaken, Switzerland, 1997: 89-95

[5] RIESENFELD R F. Homogeneous coordinates and projective planes in computer graphics[J]. *IEEE Computer Graphics and Applications*, 1981(1): 50-55

[6] BLOOMENTHAL J, ROKNE J. Homogeneous coordinates[J]. *The Visual Computer*, 1994, 11(1): 15-26

[7] BLINN J F, NEWELL M E. Clipping using homogeneous coordinates[C]// Proceedings of the 5th Annual Conference on Computer Graphics and Interactive Techniques, New York, USA, 1978: 245-251

[8] BAKHODA A, YUAN G L, FUNG W W L, et al. Analyzing CUDA workloads using a detailed GPU simulator[C]// 2009 IEEE International Symposium on Performance Analysis of Systems and Software, Boston, USA, 2009: 163-174

[9] POWER J, HESTNESS J, ORR M S, et al. gem5-gpu: a heterogeneous CPU-GPU simulator[J]. *IEEE Computer Architecture Letters*, 2014, 14(1): 34-36

[10] LOWE-POWER J, AHMAD A M, AKRAM A, et al. The gem5 simulator: Version 20.0 +[EB/OL]. http://arxiv.org/abs/2007.03152: arXiv, (2020-07-07), [2021-01-12]

[11] UBAL R, JANG B, MISTRY P, et al. Multi2Sim: a simulation framework for CPU-GPU computing[C]//2012 21st International Conference on Parallel Architectures and Compilation Techniques, Minneapolis, USA, 2012: 335-344

[12] BALASUBRAMANIAN R, GANGADHAR V, GUO Z, et al. Enabling GPGPU low-level hardware explorations with MIAOW: an open-source RTL implementation of a GPGPU[J]. *ACM Transactions on Architecture and Code Optimization*, 2015, 12(2): 21: 1-21: 25

[13] KHAIRY M, SHEN Z, AAMODT T M, et al. Accel-Sim: an extensible simulation framework for validated GPU modeling[C]//2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA), Valencia, Spain, 2020: 473-486

[14] Del Barrio V M, González C, Roca J, et al. ATTILA: a cycle-level execution-driven simulator for modern GPU architectures[C]//2006 IEEE International Symposium on Performance Analysis of Systems and Software, Austin, USA, 2006: 231-241

[15] GUBRAN A A, AAMODT T M. Emerald: graphics modeling for SoC systems[C]//Proceedings of the 46th International Symposium on Computer Architecture, Phoenix, USA, 2019:169-182

[16] GitHub. glmark2[EB/OL]. https://github.com/glmark2/glmark2, (2017),[2021-01-12]

[17] ROST R J, LICEA-KANE B, GINSBURG D, et al. OpenGL shading language[M]. London: Pearson Education, 2009

[18] BARGEN B, DONNELLY P. Inside DirectX, Microsoft Programming Series[M]. Redmond, Washington: Microsoft Press, 1998

[19] CANTLAY I. Directx 11 terrain tessellation[J]. *Nvidia Whitepaper*, 2011, 8(11): 3

[20] ROST R J, LICEA-KANE B, GINSBURG D, et al. OpenGL Shading Language[M]. London: Pearson Education, 2009

[21] Microsoft. High-level shader language (HLSL)[EB/OL]. https://docs.microsoft.com/en-us/windows/win32/direct3dhlsl/dx-graphics-hlsl:Microsoft, (2021), [2021-01-12]

[22] 张立志, 赵士彭, 赵皓宇, 等. 高性能 GPU 模拟器的实现[J]. 高技术通讯, 2020, 30(6): 553-560

[23] 赵士彭, 张立志, 赵皓宇, 等. 高性能 GPU 模拟器驱动设计研究[J]. 高技术通讯, 2020, 30(5): 435-442

GPU-Hi: GPU RTL platform implementation and efficiency analysis

ZHANG Lizhi[*,**,***], ZHAO Shipeng[*,**,***], ZHANG Longbing[*,**,***]

([*]State Key Laboratory of Computer Architecture, Institute of Computer Technology, Chinese Academy of Sciences, Beijing 100190)

([**]Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190)

([***]University of Chinese Academy of Sciences, Beijing 100049)

Abstract

GPU-Hi register-transfer level (RTL) graphics processing unit (GPU) research platform is established. GPU-Hi supports OpenGL 2.0 API (application programming interface), supports unified shader rendering architecture, uses application-specific integrated circuit (ASIC) to complete the fixed function algorithm of graphics pipeline, and uses single instruction multiple thread (SIMT) architecture to complete the programmable shader module of the graphics pipeline. In the case of using a 28 nm process, the physical design area of the platform is 7.9 μm^2. Using glmark2 test set as the performance test program, the functional correctness of the platform is verified. At the same time, the test set is used to study the computing characteristics of 3D graphics applications, and the performance analysis of GPU microstructure level is performed. The test results show that the rasterization tasks and pixel shading tasks of graphics applications do not increase in proportion to the resolution of graphics applications; at the same time, the performance of the rasterization module of GPU hardware is affected by the processing power of the shading program and the ability of video memory access. The implementation of this platform is important for the research and development of the GPURTL platform. The conclusions obtained in this article have important reference significance for GPU performance optimization and strongly support the development of GPU hardware research.

Key words: graphics processing unit (GPU), performance analysis, glmark2, stream processors cluster